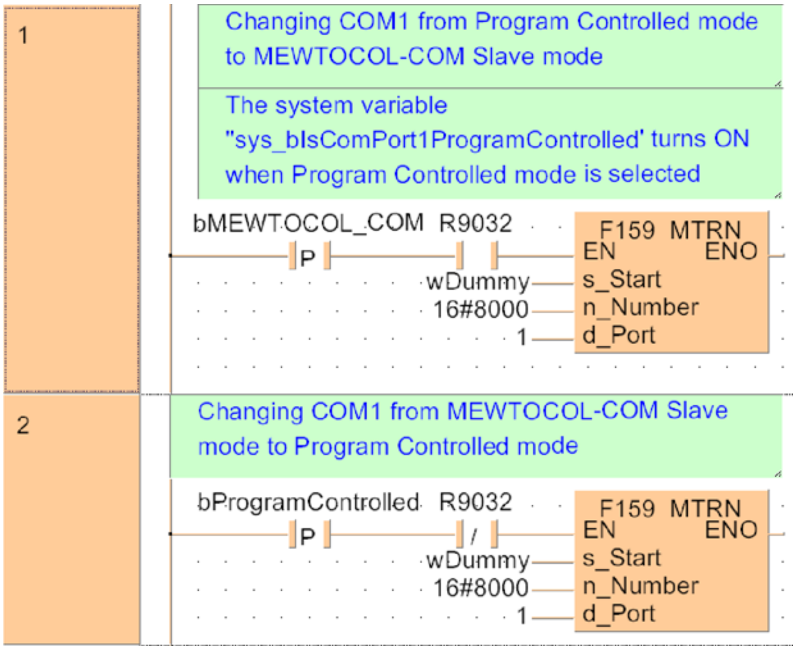
Panasonic

Control FPWIN Pro

FP SERIES FP-Sigma FP-X

Programming



Panasonic FP Control FPWIN Pro FP-Sigma, FP-X Programming Panasonic Electric Works Europe AG ACGM0132V20END 04/2006

BEFORE BEGINNING

Liability and Copyright for the Hardware

This manual and everything described in it are copyrighted. You may not copy this manual, in whole or part, without written consent of Panasonic Electric Works Europe AG (PEWEU).

PEWEU pursues a policy of continuous improvement of the design and performance of its products, therefore, we reserve the right to change the manual/product without notice. In no event will PEWEU be liable for direct, special, incidental, or consequential damage resulting from any defect in the product or its documentation, even if advised of the possibility of such damages.

We invite your comments on this manual. Please email us at:

tech-doc@eu.pewg.panasonic.com.

Please direct support matters and technical questions to your local Panasonic representative.

LIMITED WARRANTY

If physical defects caused by distribution are found, PEWEU will replace/repair the product free of charge. Exceptions include:

- When physical defects are due to different usage/treatment of the product other than described in the manual.
- When physical defects are due to defective equipment other than the distributed product.
- When physical defects are due to modifications/repairs by someone other than PEWEU.
- When physical defects are due to natural disasters.

Important Symbols

One or more of the following symbols may be used in this manual:

Warning.
The warning triangle indicates especially important safety instructions. If they are not adhered to, the results could be:

- **fatal or critical injury and/or**
- **significant damage to instruments or their contents, e.g. data**

◆ **NOTE**

Contains important additional information.

◆ **EXAMPLE**

Contains an illustrative example of the previous text section.

◆ **PROCEDURE**

Indicates that a step-by-step procedure follows.

◆ **REFERENCE**

Indicates where you can find additional information on the subject at hand.

◆ **CAUTION**

Indicates that you should proceed with caution.

◆ KEYPOINTS

Summarizes key points in a concise manner.

◆ SHORTCUTS

Provides helpful keyboard shortcuts.

◆ EXPLANATION

Provides brief explanation of a function, e.g. why or when you should use it.

next page

Indicates that the text will be continued on the next page.

The manual uses the following conventions to indicate elements from the user interface or the keyboard:

"Data Field"	Data field entries and option names are rendered in quotation marks.
[Button]	Buttons are indicated by square brackets.
<Key>	Keys are indicated by pointed brackets

Table of Contents

Part I Basics

Part II IEC Instructions

Part III F/P Instructions

Part III F/P Instructions

Part III F/P Instructions

Part III F/P Instructions

Part III F/P Instructions

Chapter 1

Basics

1.1 Operands

In FPWIN Pro the following operands are available:

- in- and outputs (X/Y) as well as internal memory areas
- internal relays
- special internal relays
- timers and counters
- data registers
- special data registers
- file registers
- link registers and relays

The number of operands which are available depends on the PLC-type and its configuration. To see how many of the respective operands are available, see your hardware description.

1.1.1 Inputs/Outputs

The amount of inputs/outputs available depends on the PLC and unit type. Each input terminal corresponds to one input **X,** each output terminal corresponds to one output **Y**.

In system register 20 you set whether an output can be used once or more during the program.

☞ **Outputs which do not exist physically can be used like flags. These flags are non-holding, which means their contents will be lost, e.g. after a power failure.**

1.1.2 Internal Relays

Internal Relays are memory areas where you can store interim results. Internal relays are treated like internal outputs.

In system register no. 7 you define which internal relays are supposed to be holding/non-holding. Holding means that its values will be retained even after a power failure.

The number of available internal relays depends on the PLC type (see hardware description of your PLC).

1.1.3 Special Internal Relays

Special internal relays are memory areas which are reserved for special PLC functions. They are automatically set/reset by the PLC and are used:

- to indicate certain system states, e.g. errors

- as an impulse generator
- to initialize the system
- as ON/OFF control flag under certain conditions

  such as when some flags get a certain status if data are ready for transmission in a PLC network.

The number of special internal relays available depends on the PLC type (see hardware description of your PLC).

☞ **Special internal relays can only be read.**

1.1.4 Timers and Counters

Timers and Counters use one common memory and address area.

Define in system registers 5 and 6 how the memory area is to be divided between timers and counters and which timers/counters are supposed to be holding or non-holding. Holding means that even after a power failure all data will be saved, which is not the case in non-holding registers.

Entering a number in system register 5 means that the first counter is defined. All smaller numbers define timers.

For example, if you enter zero, you define counters only. If you enter the highest value possible, you define timers only.

In the default setting the holding area is defined by the start address of the counter area. This means all timers are holding and all counters are non-holding. You can of course customize this setting and set a higher value for the holding area, which means some of the timers, or if you prefer, all of them can be defined as holding.

In addition to the timer/counter area, there is a memory area reserved for the set value (SV) and the elapsed value (EV) of each timer/counter contact. The size of both areas is 16 bits (WORD). In the SV and EV area one INTEGER value from 0 to 32,767 can be stored.

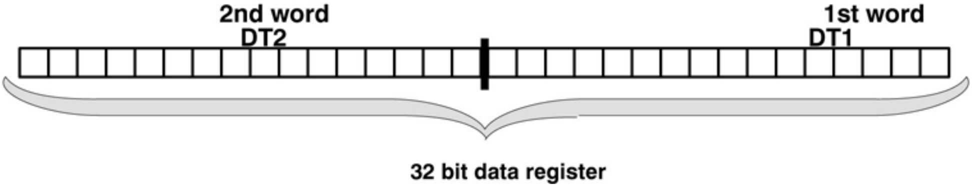
Timer/Counter No.	SV	EV	Relay
TM0	SV0	EV0	T0
. . .	. . .	. . .	. . .
TM99	SV99	EV99	T99
CT100	SV100	EV100	C100
. . .	. . .	. . .	. . .

While a timer or counter is being processed, the respective acual value can be read and under certain conditions be edited.

☞ **After changing the settings in system register 5, do not forget to adjust the addresses of the timers/counters in your PLC program because they correspond to the TM/CT numbers.**

1.1.5 Data Registers (DT)

Data registers have a width of 16 bits. You can use them, for example, to write and read constants/parameters. If an instruction requires 32 bits, two 16-bit data registers are used. If this is the case, enter the address of the first data register with the prefix DDT instead of DT. The next data register (word) will be used automatically (for more information, please refer to addresses (see page 7)).



Data registers can be holding or non-holding. Holding means that even after a power failure all data will be saved. Set the holding/non-holding areas in system register 8 by entering the start address of the holding area.

The amount of data registers available depends on the PLC type (see hardware description).

1.1.6 Special Data Registers (DT)

Special data registers are like the special internal relays reserved for special functions and are in most cases set/reset by the PLC.

The register has a width of 16 bits (data type = WORD). The amount of special data registers available depends on the PLC type (see hardware description).

Most special data registers can only be read. Here some exceptions:

- interrupts and scan time (DT9027, DT9023-DT9024)...
-

1.1.6.1 Data Transfer To and From Special Data Registers

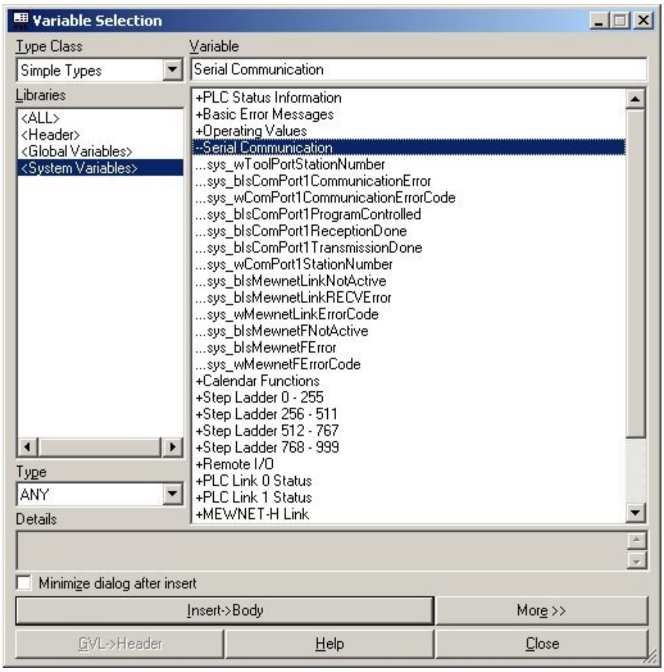
FPWIN Pro offers three possibilities to read from or write to special relays/special data registers.

1. Via system variables (recommended from version 5.1 onwards)

For each special data register and relay a system variable exists according to the following syntax:

```
sys_ * _system variable
     |
     └ b   BOOL
     └ w   WORD
     └ dw  DWORD
     └ i   INT
     └ di  DINT
```

You can insert these system variables into the body via the "Variable Selection" dialog.



In addition these system variables are also displayed under **Monitor** → **Special Relays and Registers** as the last entries in the comments, e.g. "**sys_w_HSC_ControlFlags**".

Example for accessing the special data for HSC

Example for accessing the special data for the RTC

2. via global variables
3. via direct addresses in the body

1.1.7 File Registers (FL)

Some PLC types (see hardware description) provide additional data registers which can be used to increase the number of data registers. File registers are used in the same way as data registers. Set the holding/non-holding area in system register 9. Holding means that even after a power failure all data will be saved.

1.1.8 Link Relays and Registers (L/LD)

Link relays have a width of 1 bit (BOOL). In system registers 10-13 and 40-55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non-holding area

Link registers have a width of 16 bits (WORD). In system registers 10-13 and 40-55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non-holding area

1.2 Addresses

In the List of Global Variables, enter the physical address in the field “Address” for each global variable used in the PLC program.

The operand and the address number are part of the address. In FPWIN Pro you can use either FP and/or IEC addresses. The following abbreviations are used:

Meaning	FP	IEC
Input	X	I
Output	Y	Q
Memory (internal memory area)	R	M0
Timer relay	T	M1
Counter relay	C	M2
Set value	SV	M3
Elapsed value	EV	M4
Data register	DT/DDT	M5

You find the register numbers (e.g. DT9000/DT90000) in your hardware description. The next two sections show how FP and IEC addresses are composed.

1.2.1 FP Addresses

An address represents the hardware address of an in-/output, register, or counter.

For example, the hardware address of the 1st input and the 4th output of a PLC is:

- X0 (X = input, 0 = first relay)
- Y3 (Y = output, 3 = fourth relay)

Use the following address abbreviations for the memory areas. You find the register numbers in your hardware description.

Memory Area	Abbr. FP	Example
Memory (internal memory area)	R	R9000: self diagnostic error
Timer relay	T	T200: timer relay no. 200 (settings in system register 5+6)
Counter relay	C	C100: counter relay no. 100 (settings in system register 5+6)
Set value	SV	SV200 (set value for counter relay 200)
Elapsed value	EV	EV100 (elapsed value for timer relay 100)
Data register	DT	DT9001/DT90001 (signals power failure)

1.2.2 IEC Addresses

The composition of an IEC-1131 address depends on:

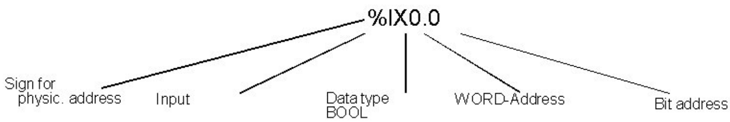
- operand type
- data type
- slot no. of the unit (word address)
- relay no. (bit address)
- PLC type

In- and Outputs are the most important components of a programmable logic controller (PLC). The PLC receives signals from the input relays and processes them in the PLC program. The results can either be stored or sent to the output relays, which means the PLC controls the outputs.

A PLC provides special memory areas, in short “M”, to store interim results, for example.

If you want to read the status of the input 1 of the first module and control the output 4 of the second module, for example, you need the physical address of each in-/output. Physical FPWIN Pro addresses are composed of the per cent sign, an abbreviation for in-/output, an abbreviation for the data type and of the word and bit address:

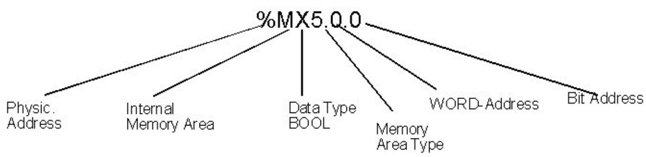
Example **IEC address for an input**



The per cent sign is the indicator of a physical address. “I” means input, “X” means data type BOOL. The first zero represents the word address (slot no.) and the second one the bit address. Note that counting starts with zero and that counting word and bit addresses differs among the PLC types.

Each PLC provides internal memory areas (M) to store interim results, for example. When using internal memory areas such as data registers, do not forget the additional number (here 5) for the memory type:

Example **IEC address for an internal memory area**



Bit addresses do not have to be defined for data registers, counters, timers, or the set and actual values.

According to IEC 1131, abbreviations for **in- and output** are “I” and “O”, respectively. Abbreviations for the **memory areas** are as follows:

Memory Type	No.	Example
Internal Relay (R)	0	%MX0.900.0 = internal relay R9000

Memory Type	No.	Example
Timer (T)	1	%MX1.200 = counter no. 200
Counter (C)	2	%MX2.100 = counter no. 100
Set Value counters/timers (SV)	3	%MW3.200 = set value of the counter no. 200
Elapsed Value counters/timers (EV)	4	%MW4.100 = elapsed value of the timer no. 100
Data Registers (DT, DDT)	5	%MW5.9001 = data register DT9001 %MD5.90001 = 32-bit data register DDT90001

☞ **Tables with hardware addresses can be found in the hardware description of your PLC.**

The following data types are available:

Data Type	Abbreviation	Range of Values	Data Width
BOOL	BOOL	0 (FALSE), 1 (TRUE)	1 bit
INTEGER	INT	-32,768 to 32,768	16 bits
DOUBLE INTEGER	DINT	-2,147,438,648 to 2,147,438,647	32 bits
WORD	WORD	0 to 65,535	16 bits
DOUBLE WORD	DWORD	0 to 4,294,987,295	32 bits
TIME 16-bit	TIME	T#0.00s to T#327.67s	16 bits
TIME 32-bit	TIME	T#0,00s to T#21 474 836.47s	32 bits
STRING	STRING	1 to 255 bytes (ASCII)	8 bits per bytes
REAL	REAL	-1,175494 x 10^{-38} to -3,402823 x 10^{-38} and 1,175494 x 10^{-38} to 3,402823 x 10^{-38}	32 bits

☞ **Please take into account that not all data types can be used with each IEC command.**

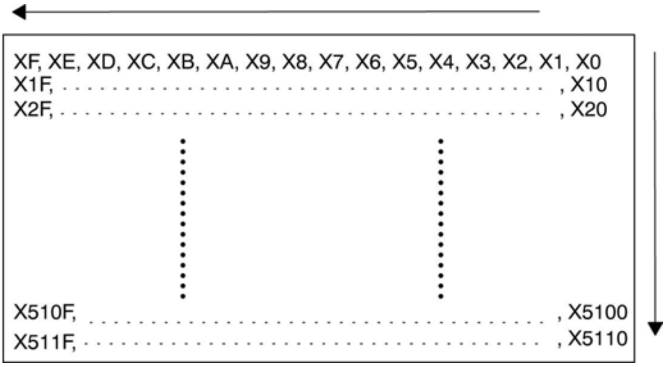
☞

- **Find the tables with all memory areas in your hardware description.**
- **When using timers, counters, set/elapsed values, and data registers, the bit address does not have to be indicated.**
- **You can also enter the register number (R9000, DT9001/90001) or the FP address, e.g. “X0” (input 0), instead of the IEC address.**

1.2.3 Specifying Relay Addresses

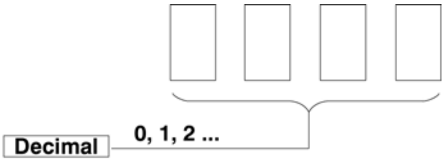
External input relay (X), external output relay (Y), internal relay (R), link relay (L) and pulse relay (P)The lowest digit for these relay's adresses is expressed in hexadecimals and the second and higher digits are expressed in decimals as shown below.

Example **Configuration of external input relay (X)**



1.2.4 Timer Contacts (T) and Counter Contacts (C)

Addresses of timer contacts (T) and counter contacts (C) correspond to the **TM** and **CT** instruction numbers and depend on the PLC type.



e.g. for FP2:
T0, T1 .... T2999
C3000, C3001 .... C3072

☞ **Since addresses for timer contacts (T) and counter contacts (C) correspond to the TM and CT instruction numbers, if the TM and CT instruction sharing is changed by system register 5, timer and counter contact sharing is also changed.**

1.2.5 External Input (X) and Output Relays (Y)

- The external input relays available are those actually allocated for input use.
- The external output relays actually allocated for output can be used for turning ON or OFF external devices. The other external output relays can be used in the same way as internal relays.

I/O allocation is based on the combination of I/O and intelligent modules installed.

1.2.6 Word Representation of Relays (WX, WY, WR, and WL)

The external input relay (X), external output relay (Y), internal relay (R) and link relay (L) can also be expressed in word format. The word format treats 16-bit relay groups as one word. The word expressions for these relays are word external input relay (WX), word external output relay (WY), word internal relay (WR) and word link relay (WL), respectively.

Example:

Configuration of word external input relay (WX)

☞ **Since the contents of the word relay correspond to the state of its relays (components), if some relays are turned ON, the contents of the word change.**

1.3 Constants

A constant represents a fixed value. Depending on the application, a constant can be used as an addend, multiplier, address, in-/output number, set value, etc.

There are 3 types of constants:

- decimal
- hexadecimal
- BCD

1.3.1 Decimal Constants

Decimal constants can have a width of either 16 or 32 bits.

Range 16 bit: -32,768 to 32,768

Range 32 bit: -2,147,483,648 to 2,147,483,648

Constants are internally changed into 16-bit binary numbers including character bit and are processed as such. Simply enter the decimal number in your program.

1.3.2 Hexadecimal Constants

Hexadecimal constants occupy fewer digit positions than binary data. 16 bit constants can be represented by 4-digit, 32-bit constants by 8-digit hecadecimal constants.

Range 16 bit: 8000 to 7FFF

Range 32 bit: 80000000 to 7FFFFFFFF

Enter e.g.: 16#7FFF for the hexadecimal value 7FFF in your program.

1.3.3 BCD Constants

BCD is the abbreviation for Binary Coded Decimal.

Range 16 bit: 0 to 9999

Range 32 bit: 0 to 99999999

Enter BCD constants in the program either as:

binary: 2#0001110011100101 or
hexadecimal: 16#9999

1.4 Data Types

FPWIN Pro provides elementary and user defined data types.

Elementary data types

Data Type	Abbreviation	Value Range	Data Width
BOOL	BOOL	0 (FALSE) or 1 (TRUE)	1 bit
INTEGER	INT	-32,768 to 32,768	16 bits
DOUBLE INTEGER	DINT	-2,147,483,648 to 2,147,483,647	32 bits
WORD	WORD	0 to 65,535	16 bits
DOUBLE WORD	DWORD	0 to 4,294,967,295	32 bits
TIME 16- bit	TIME	T#0,00s to T#327.67s	16 bits
TIME 32 -bit	TIME	T#0,00s to T#21 474 836,47s	32 bits
STRING	STRING	1 to 255 bytes (ASCII)	8 bits per byte
REAL	REAL	-1,175494 x 10^{-38} to -3,402823 x 10^{-38} and 1,175494 x 10^{-38} to 3,402823 x 10^{-38}	32 bits

A data type has to be assigned to each variable.

User defined data types

We differentiate between **array** and **D**ata **U**nit **T**ypes (DUT). An array consists of several elementary data types which are all of the same type. A DUT consists of several elementary data types but of different data types. Each represents a new data type.

1.4.1 BOOL

Variables of the data type BOOL are binary variables. They can only have the value 0 or 1, and always have a width of 1 bit.

The condition 0 corresponds to **FALSE** (e.g. initial value in the POU header) and means that the variable is switched off. In this case we also speak of the variable not being set.

The condition 1 corresponds to **TRUE** (e.g. initial value in the POU header) and means that the variable is switched on. In this case we also speak of the variable being set.

The default initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0 (FALSE). In this case the variable is not set during the PLC program start. If this is not the case, the initial value may also be set to TRUE.

1.4.2 INT

Variable values of the data type INTEGER are natural numbers without decimal places. The range of values for INTEGER values is from -32768 to 32767

The default initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0

INTEGER numbers can be entered in DEC-, HEX- or BIN- format:

Decimal Number	Hexadecimal Number	Binary Number
1234	16#4D2	2#10011010010
-1234	16#FB2E	2#1111101100101110

1.4.3 DINT

Variable values of the data type DOUBLE INTEGER are natural numbers without decimal places. The value range for a DOUBLE INTEGER values is from -2147483648 to 2147483647

The default initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0

DOUBLE INTEGER numbers can be entered in DEC-, HEX- or BIN- format.

Decimal Number	Hexadecimal Number	Binary Number
123456789	16#75BCD15	2#111010110111100110100010101
-123456789	16#F8A432EB	2#11111000101001000001100101110

1.4.4 STRING

The data type STRING consists of a series, i.e. string, of ASCII characters up to 255 characters (default setting under **Extras** → **Options** → **Compile options** → **Code Generation**). All ASCII characters are considered as characters.

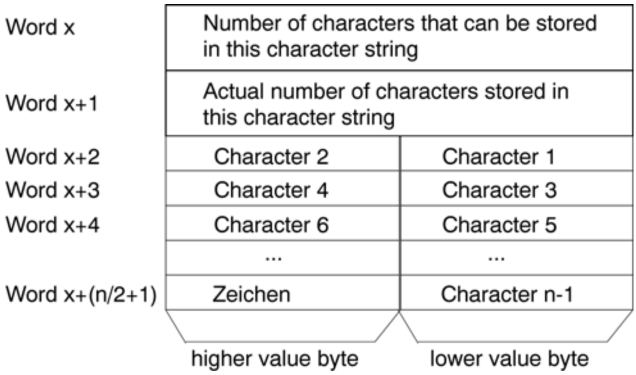
The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list is ' ' that corresponds to an empty string.

Internal memory structure of strings on the PLC

Each character of the string is stored in a byte. The memory to which a string is allotted consists of a head (2 words) that contains the following information:

- The first word contains the number of characters that are reserved in the memory for this string (the standard value is 32 characters).

- The second word contains the actual number of characters that are stored in this string.



You may declare the number of characters (n) in a string and thereby determine the size of the allotted memory.

The following condition applies: Reserved memory = 2 words (for the head) + (n+1)/2 words (for the characters)

◆ **NOTE**

Since the memory is organized according to words, it is always rounded up to the next larger whole number.

String Literals (according to IEC 61131-3)

A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character (').

Three-character combinations of the dollar sign ($) followed by two hexadecimal digits are to be interpreted as the hexadecimal representation of the eight-bit character code.

Two-character combinations beginning with the dollar sign are to be interpreted as shown in the table:

Combination	Interpretation When Printed
$$	Dollar sign ($24)
$'	Single quote ($27)
$L or $l	Line feed ($0A)
$N or $n	New line ($0D$0A)
$P or $p	Form feed (page) ($0C)
$R or $r	Carriage return ($0D)
$T or $t	Tab ($09)

Examples of String Literals

Example	Explanation
''	Empty string (length zero)

Example	Explanation
'A'	String of length one containing the single character A
' '	String of length one containing the space character
'$''	String of length one containing the single quote character
'RL'	String of length two containing CR and LF characters
'$$1.00'	String of length five which would print as "$1.00"
'$02$03'	String of length two containing STX and ETX characters

◆ NOTE

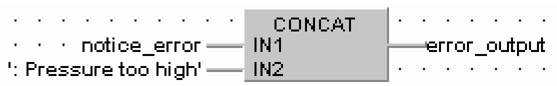
The hexadecimal value 0 ($00) cannot be entered in a string literal. But the string commands can handle these values correctly if they occur in string variables. So you can, for example, also use strings in supplying telegrams with raw data for data communication.

Strings as constants

It is possible to enter values of the data type STRING directly as constants into a function or a function block. A declaration in the POU Header is not necessary in this case.

Transfer of the character string constant ': Pressure too high' to a function.

LD Body



IL Body

```
LD        notice_error
CONCAT    ': Pressure too high'
ST        error_output
```

Transfer of character strings to functions or function blocks

When character strings are transferred, only as many characters that fit into the target string are transferred. Various examples include:

Example 1	Copy of a character string **source** to a character string **target** that has less reserved memory than **source**. For a description of these examples, please refer to the online help under the keyword ‘Example 1…5 for STRING’.
Example 2	Copy of a constant character string to a character string that has less reserved memory than the constant.
Example 3	A longer character string is attached to the input contact of a function than is reserved for the input variable in the POU header of the function.
Example 4	A longer character string is calculated for a function than the value of the function can return.
Example 5	A function returns a longer character string than the target variable can store.

The replace functions (E_)INT_TO_STRING (see page 162), (E_)DINT_TO_STRING (see page 165), (E_)REAL_TO_STRING (see page 168),

(E_)TIME_TO_STRING (see page 170) etc. require relatively many system resources (program steps and processing time). Therefore define User-defined functions when you use these functions repeatedly.

Restrictions:

- **When using the data type STRING in PLCs that do not employ String instructions per se (FP-Sigma):**
- **Can only be used for initializations in the header, as constant in the body, as function or function block argument or in the following commands:**

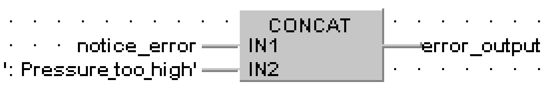
  - BOOL_TO_STRING (see page 156)
  - CONCAT (see page 212)
  - DINT_TO_STRING (see page 165)
  - DWORD_TO_STRING (see page 160)
  - EQ (see page 86)
  - FIND (see page 216)
  - INT_TO_STRING (see page 162)
  - LEN (see page 204)
  - MOVE (see page 28)
  - NE (see page 92)
  - REAL_TO_STRING (see page 168)
  - SEL (see page 200)
  - TIME_TO_STRING (see page 170)
  - WORD_TO_STRING (see page 158)
- **The functions Concat and Find require relatively many system resources (program steps, labels and processing time). Therefore define User-defined functions when you use these functions repeatedly. Only use these functions when absolutely necessary or define User-defined functions when you use these functions repeatedly.**

1.4.4.1 Strings as Constants

It is possible to enter values of the data type STRING directly as constants into a function or a function block. A declaration in the POU Header is not necessary in this case.

Example **Transfer of the character string constant ': Pressure too high' to a function.**

LD Body



1.4.4.2 Transfer of Character Strings to Functions or Function Blocks

When character strings are transferred, only as many characters that fit into the target string are transferred. Various examples include:

1. Copy of a character string source to a character string target that has less reserved memory than source.
2. Copy of a constant character string to a character string that has less reserved memory than the constant.
3. A longer character string is attached to the input contact of a function than is reserved for the input variable in the POU header of the function.
4. A longer character string is calculated for a function than the value of the function can return.
5. A function returns a longer character string than the target variable can store.

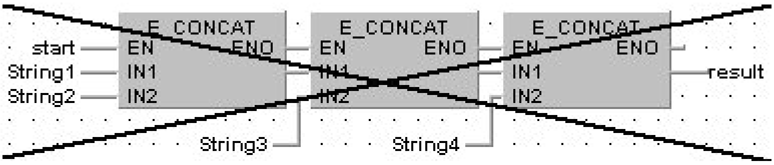
1.4.4.3 String with EN/ENO

Using STRING instructions with enable input (EN) and enable output (ENO) in ladder diagrams (LD) and function block diagrams (FBD)

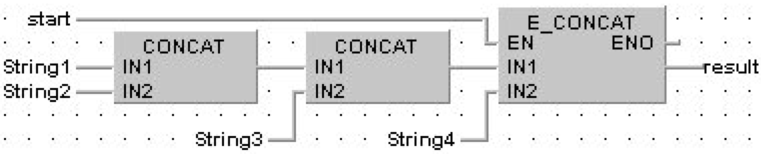
STRING instructions with EN/ENO contacts cannot be connected to each other in LD and FBD.

However, you may use this configuration if the instructions in question are first connected to each other and then an instruction with EN/ENO is used in the final position. The enable input (EN) determines the output of its overall result.

This arrangement is not possible:



This arrangement is possible:



Using STRING instructions in instruction lists (IL)

STRING instructions with EN/ENO may be connected to each other in IL. Nevertheless, in order to avoid intermediate variables, it is suggested that you use a conditional jump instead of connecting a series of functions with EN/ENO.

POU Header of a program with a dummy string

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	
1	VAR	String1	STRING[4]	'This'	
2	VAR	String2	STRING[3]	' is'	
3	VAR	String3	STRING[2]	' a'	
4	VAR	String4	STRING[5]	' test'	
5	VAR	result	STRING[32]	''	
6	VAR	help_string	STRING[32]	''	

IL Body

```
(* When start = TRUE then calculate
   result = String1 + String2 + String3 + String4 *)

LD          start
E_CONCAT    String1, String2, help_string
E_CONCAT    help_string, String3, help_string
E_CONCAT    help_string, String4, result
```

POU Header of a program with a conditional jump

	Class	Identifie	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	
1	VAR	String1	STRING[4]	'This'	
2	VAR	String2	STRING[3]	' is'	
3	VAR	String3	STRING[2]	' a'	
4	VAR	String4	STRING[5]	' test'	
5	VAR	result	STRING[32	''	

IL Body

```
1
        (* When start = TRUE then calculate
           result = String1 + String2 + String3 + String4 *)

        LDN         start
        JMPC        marker
        LD          String1
        CONCAT      String2
        CONCAT      String3
        CONCAT      String4
        ST          result

2
marker: (* Insert code for network 2 here *)
```

The difficulty of programming with a dummy string lies in correctly choosing its length. When connecting unconditional string instructions in series, this is calculated automatically.

1.4.5 WORD

A variable of the data type WORD consists of 16 binary states. The switching states of 16 in/outputs can be combined as a unity in one word (WORD).

The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0

You can enter WORD values in (DEC-), HEX- or BIN- format.

(Decimal Number)	Hexadecimal Number	Binary Number
1234	16#4D2	2#10011010010
64302	16#FB2E	2#1111101100101110

1.4.6 DWORD

A variable of the data type DOUBLE WORD consists of 32 binary states. The switching states of 32 in/outputs can be combined as a unity in one DOUBLE WORD.

The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0

You can enter numbers in (DEC-), HEX- or BIN- format.

(Decimal Number)	Hexadecimal Number	Binary Number
123456789	16#75BCD15	2#111010110111100110100010101
4171510507	16#F8A432EB	2#11111000101001000011001011100

1.4.7 ARRAY and Data Unit Type

ARRAYs

An array is a group of variables which all have the **same** elementary data type and that are grouped together, one after the other, in a continuous data block. This variable group itself is a variable and must hence be declared for this reason. In the program you can either use the whole array or individual array elements.

NOTE

An array cannot be used as a variable by another array.

Data types valid for arrays are:

- BOOL
- INT
- DINT
- REAL

- WORD
- DWORD
- TIME
- STRING

Arrays can be 1, 2 or 3-dimensional. In each dimension, an array can have several fields.

Data Unit Type

A **D**ata **U**nit **T**ype (DUT) is a group of variables composed of several **different** elementary data types (BOOL, WORD etc.). These groups are used when tables are edited, such as for the bit sample edition in the F164_SPD0 command (FP1, FP-M) of the FP Library (see online help). You can use the bit sample edition of this command for regulating the speed of a motor via a speed governor, for example. Define a DUT in the DUT pool first. Then you can use the DUT in the “Type” field of the global variable list or of a POU header similarly to the integer, BOOL etc. data types. In the program you can then use either the whole DUT or individual variables of the DUT.

◆ **NOTE**

A DUT cannot be used as a variable by another DUT.

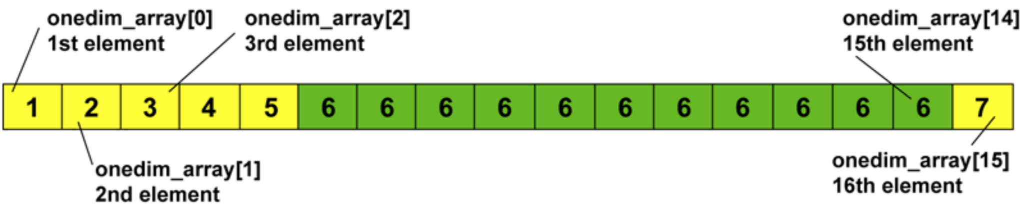
For details on working with ARRAYs or DUTs, please refer to the online help or programming manuals.

1.4.7.1 One dimensional ARRAY

Global Variables

	Class	Identifier	FP A...	IEC Address	Type	Initial
0	VAR_GLOBAL	onedim_array	DT0	%MW5.0	ARRAY [0..15] OF INT	[1,2,3,4,5,10(10),7]

The declared array can be imagined as follows:



Initialize 1-dim arrays with values

If subsequent array elements are initialized with the same value, the abbreviated writing **number(value)** is possible.

* **number** stands for the number of array elements

* **value** stands for the initialization value

In the example **element 1** was initialized with value 1, **element 2** with value 2, etc.

Use 1-dim array elements in the program

You may use a one-dimensional array element by entering identifier[Var1].

* identifier (name of the array, see field Identifier)

* **Var1** is a variable of the type INT or a constant which has to be within the value range of the array declaration. For this example Var1 is assigned to the range 0...15

In the example you call up the third array element (**Element 3**) with **onedim_array[2]**. If you wish to assign a value to this element in an IL program, for example, enter the following:

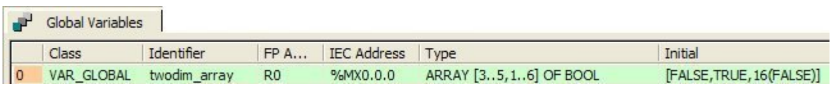
```
LD    current_temp
ST    onedim_array[2]
```

Addresses of 1-dim array elements

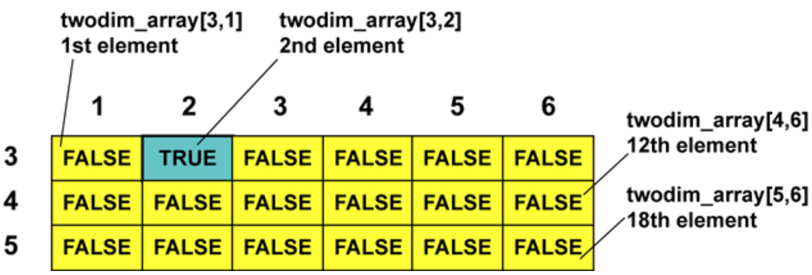
The array elements of the one-dimensional array are subsequently saved in the memory of the PLC starting with element 1. The following memory allocation results for the example described above:

Array element name	Array element	FP Address	IEC Address
onedim_array[0]	Element 1	DT0	%MW5.0
onedim_array[1]	Element 2	DT1	%MW5.1
onedim_array[2]	Element 3	DT2	%MW5.2
onedim_array[3]	Element 4	DT3	%MW5.3
onedim_array[4]	Element 5	DT4	%MW5.4
...	...	...	...
onedim_array[13]	Element 14	DT13	%MW5.13
onedim_array[14]	Element 15	DT14	%MW5.14
onedim_array[15]	Element 16	DT15	%MW5.15

1.4.7.2 Two dimensional ARRAY



The declared array can be imagined as follows:



Initialize 2-dim arrays with values

The initialization of arrays with values starts with the first array element (element 1) and ends with the last array element (element 18). The initialization values are entered one after another into the field initial and are separated from each other by commas.

If subsequent array elements are initialized with the same value, the abbreviated writing **number(value)** is possible.

* **number** stands for the number of array elements

* **value** stands for the initialization value

In the example **element 1** was initialized with the value FALSE, **element 2** with the value TRUE and the remaining array elements are initialized with FALSE.

Use 2-dim array elements in the program

You may use a two-dimensional array element by entering identifier[Var1Var2].

* identifier (name of the array, see field Identifier)

* **Var1** and **Var2** are variables of the type INT or constants which have to be within the value range of the array declaration. For this example Var1 is assigned to the range 3...5 and Var2 to the range 1...6.

In the example you call up the element 12 with **twodim_array[4,6]**. If you wish to assign a value to this element in an IL program, for example, enter the following:

```
LD    motor_on
ST    twodim_array[4,6]
```

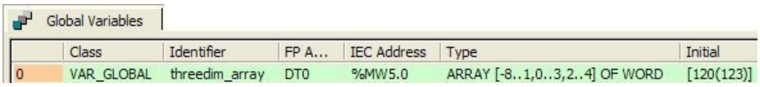
Addresses of 2-dim array elements

The array elements of the two-dimensional array are subsequently saved in the memory of the PLC starting with element 1. The following memory allocation results for the example described above:

Array element name	Array element	FP Address	IEC Address
twodim_array[3,1]	Element 1	R0	%MX0.0.0
twodim_array[3,2]	Element 2	R1	%MX0.0.1
twodim_array[3,3]	Element 3	R2	%MX0.0.2
...	...	...	...
twodim_array[3,6]	Element 6	R5	%MX0.0.5
twodim_array[4,1]	Element 7	R6	%MX0.0.6
twodim_array[4,2]	Element 8	R7	%MX0.0.7
...	...	...	...
twodim_array[5,4]	Element 16	RF	%MX0.0.15
twodim_array[5,5]	Element 17	R10	%MX0.1.0
twodim_array[5,6]	Element 18	R11	%MX0.1.1

1.4.7.3 Three dimensional ARRAY

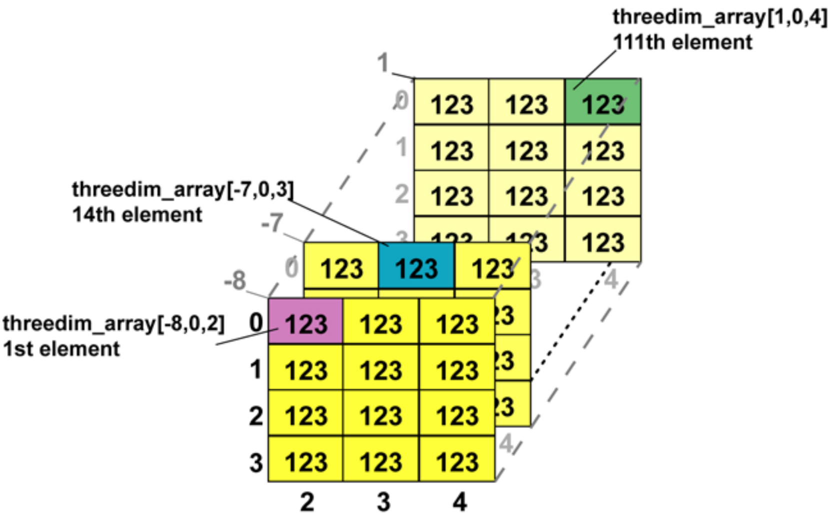
Declaration in the global variable list:



Global Variables

	Class	Identifier	FP A...	IEC Address	Type	Initial
0	VAR_GLOBAL	threedim_array	DT0	%MW5.0	ARRAY [-8..1,0..3,2..4] OF WORD	[120(123)]

The declared array can be imagined as follows:



Initialize 3-dim arrays with values

The initialization of arrays with values starts with the first array element (element 1) and ends with the last array element (element 111). The initialization values are entered one after another into the field initial and are separated from each other by commas.

If subsequent array elements are initialized with the same value, the abbreviated writing **number(value)** is possible.

* **number** stands for the number of array elements

* **value** stands for the initialization value

In the example all array elements were initialized with the value 123.

Use array elements in the program

Accessing a three-dimensional array is possible if you enter identifier[Var1,Var2,Var3,Var4].* identifier is the name of the array, (see field Identifier)

* **Var1, Var2** and **Var3** are variables of the type INT or constants which have to be within the value range of the array declaration (see field Type). For this example Var1 is assigned to the range 8...1, Var2 to the range 0...3 and Var3 to the range 2...4.

In the example you call up element 15 with **threedim_array[-7,0,4]**. If you wish to assign a value to this element in an IL program, for example, enter the following:

```
LD    binaer_value
ST    threedim_array[-7,0,4]
```

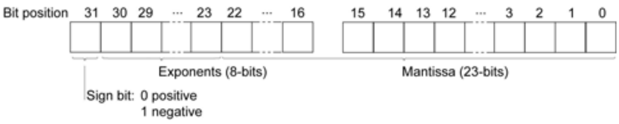
Addresses of 3-dim array elements

The array elements of the three-dimensional array are subsequently saved in the memory of the PLC starting with element 1. The following memory allocation results for the example described above:

Array element name	Array element	FP Address	IEC Address
threedim_array[-8,0,2]	Element 1	DT0	%MW5.0
threedim_array[-8,0,3]	Element 2	DT1	%MW5.1
threedim_array[-8,0,4]	Element 3	DT2	%MW5.2
threedim_array[-8,1,2]	Element 4	DT3	%MW5.3
threedim_array[-8,1,3]	Element 5	DT4	%MW5.4
...	...	...	...
threedim_array[-8,3,3]	Element 11	DT10	%MW5.10
threedim_array[-8,3,4]	Element 12	DT11	%MW5.11
threedim_array[-7,0,2]	Element 13	DT12	%MW5.12
threedim_array[-7,0,3]	Element 14	DT13	%MW5.13
...	...	...	...
threedim_array[1,3,2]	Element 118	DT117	%MW5.117
threedim_array[1,3,3]	Element 119	DT118	%MW5.118
threedim_array[1,3,4]	Element 120	DT119	%MW5.119

1.4.8 REAL

Variables of the data type REAL are real numbers or floating point constants. There are up to seven effective digits. The mantissa is 23 bits and the exponent is 8 bits (Based on IEEE754).



The value range for REAL values is between -3.402823*E38 to -1.175494*E-38, 0.0, +1.175494*E-38 to +3.402823*E38.

The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0.0

You can enter REAL values in the following format:

[+-] Integer.Integer [(Ee) [+-] Integer]

Examples:

5.983e-7

-33.876e12

3.876e3

0.000123

123.0

☞ **The REAL value always has to be entered with a decimal point (e.g. 123.0).**

Chapter 2

Data Transfer Instructions

MOVE

Move value to specified destination

Description MOVE assigns the unchanged value of the input variable to the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MOVE (see page 934)**

Data types

Data type	I/O	Function
all data types	input	source
all data types	output as input	destination

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

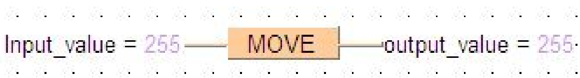
	Class	Identifier	Type	Initial	Comment
0	VAR	Input_value	INT	0	all types allowed
1	VAR	output_value	INT	0	all types allowed

In this example the input variable (**input_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

Input_value is assigned to **output_value** without being modified.

LD



ST

```
output_value:= input_value;
```

Chapter 3

Arithmetic Instructions

ADD

Add

Description This function adds the input variables IN1 + IN2 +... and writes the addition result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of ADD (see page 923)**

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **The data type REAL is available only for FP-Sigma.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	augend
INT, DINT, REAL	2nd input	addend
INT, DINT, REAL	output as input	sum

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

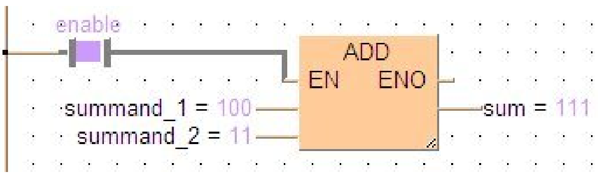
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	summand_1	INT	0	
2	VAR	summand_2	INT	0	
3	VAR	sum	INT	0	

In this example the input variables (**summand_1, summand_2** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), **summand_1** is added to **summand_2**. The result is written into **sum**.

LD



SUB

Subtract

Description The content of the accumulator is subtracted from the operand defined in the operand field. The result is transferred to the accumulator.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of SUB (see page 935)**

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **The data type REAL is available only for FP-Sigma.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	minuend
INT, DINT, REAL	2nd input	subtrahend
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

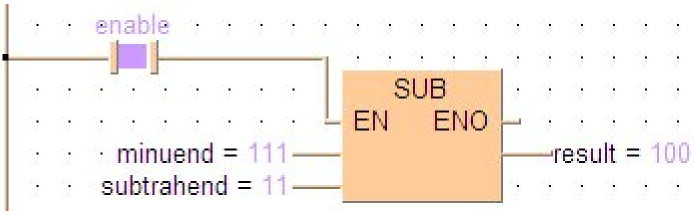
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	minuend	INT	0	
2	VAR	subtrahend	INT	0	
3	VAR	result	INT	0	

In this example the input variables (**minuend, subtrahend** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set, s**ubtrahend** (data type INT) is subracted from **minuend**. The result will be written into **result** (data type INT).

LD



MUL Multiply

Description MUL multiplies the values of the input variables with each other and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MUL (see page 934)**

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **The data type REAL is available only for FP-Sigma.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	multiplicand
INT, DINT, REAL	2nd input	multiplicator
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

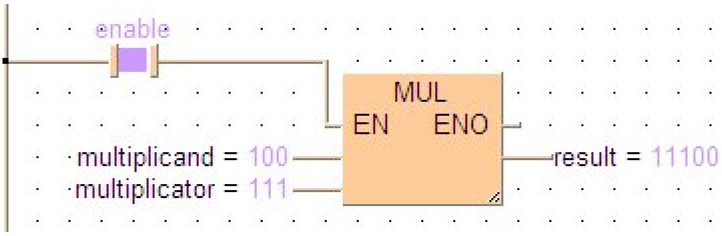
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	multiplicand	INT	0	
2	VAR	multiplicator	INT	0	
3	VAR	result	INT	0	

In this example the input variables (**multiplicand, multiplicator** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), the **multiplicant** is multiplied with the **multiplicator**. The result will be written into **result**.

LD


enable
MUL
EN
ENO
multiplicand = 100
multiplicator = 111
result = 11100

DIV

Divide

Description DIV divides the value of the first input variable by the value of the second.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DIV (see page 924)**

- **Input and output variables must be of one of the above noted data types. All operands must be of the same data type.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	dividend
INT, DINT, REAL	2nd input	divisor
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

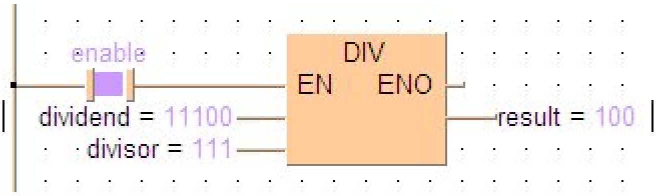
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	dividend	INT	0	
2	VAR	divisor	INT	0	
3	VAR	result	INT	0	

In this example the input variables (**dividend, divisor** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), **dividend** is divided by **divisor**. The result is written into **result**.

LD



ABS **Absolute Value**

Description ABS calculates the value in the accumulator into an absolute value. The result is saved in the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of ABS (see page 923)**

Data types

Data type	I/O	Function
INT, DINT, REAL	input	input data type
INT, DINT, REAL	output as input	absolute value

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

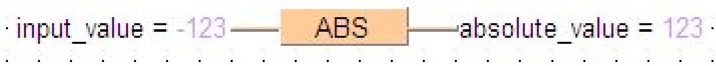
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	INT	0	
1	VAR	absolute_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

Input_value of the data type INTEGER is converted into an absolute value of the data type INTEGER. The converted value is written into **absolute_value**.

LD



ST

```
absolute_value:=ABS(input_value);
```

MOD

Modular arithmetic division, remainder stored in output variable

Description MOD divides the value of the first input variable by the value of the second. The rest of the integral division (5 : 2 : 2 + rest = 1) is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MOD (see page 934)**

With FP1-C14/C16 MOD cannot be used for a 32-bit division (DINT) as this will cause a compiler error.

Data types

Data type	I/O	Function
INT, DINT	1st input	dividend
INT, DINT	2nd input	divisor
INT, DINT	output as input	remainder

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initia	Comment
0	VAR	dividend	INT	11	
1	VAR	divisor	INT	4	
2	VAR	remainder	INT	0	11 divided by 4 = 2 with remainder of 3 3 is written into output variable

Body

This example uses variables. You may also use constants for the input variables. Dividend (11) is divided by divisor (4). The remainder (3) of the division is written in **remainder**.

LD

ST

```
remainder:= dividend MOD divisor;
```

SQRT

Square root

Description SQRT calculates the square root of an input variable of the data type REAL (value ≥ 0.0). The result is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of SQRT (see page 935)**

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	square root of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not ≥ 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	REAL	0.0	number >= 0
1	VAR	output_value	REAL	0.0	number >= 0

This example uses variables. You may also use a constant for the input variable.

Body

The square root of **input_value** is calculated and written into **output_value**.

LD

```
ST   output_value:= SQRT(input_value);
```

SIN

Sine with Radian Input Data

Description SIN calculates the sine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of SIN (see page 934)**

Data types

Data type	I/O	Function
REAL	input	input value, angle data in radians
REAL	output as input	SINE of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

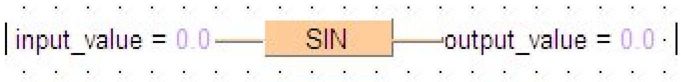
POU Header: All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	input_value	REAL	0.0
1	VAR	output_value	REAL	0.0

This example uses variables. You may also use a constant for the input variable.

Body: The sine of **input_value** is calculated and written into **output_value**.

LD



ASIN

Arcsine

Description ASIN calculates the arcsine of the input variable and writes the angle data in radians into the output variable. The function returns a value from - $\pi/2$ to $\pi/2$.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of ASIN (see page 923)**

Data types

Data type	I/O	Function
REAL	input	input value between -1 and +1
REAL	output as input	arcsine of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not ≥ -1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

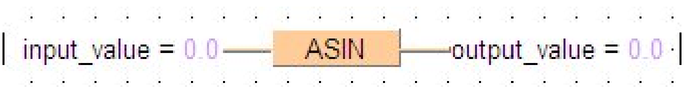
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between -1 and +1
1	VAR	output_value	REAL	0.0	angle data in radians -Pi/2 to Pi/2

This example uses variables. You may also use a constant for the input variable.

Body

The arc sine of **input_value** is calculated and written into **output_value**.

LD



```
ST   output_value:=ASIN(input_value);
```

COS Cosine

Description COS calculates the cosine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of COS (see page 924)**

Data types

Data type	I/O	Function
REAL	input	input value, angle data in radians
REAL	output as input	cosine of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

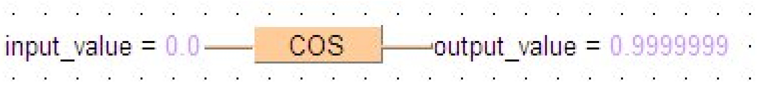
POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	cosine

This example uses variables. You may also use a constant for the input variable.

Body The cosine of **input_value** is calculated and written into **output_value**.

LD



ST
```
output_value:=COS(input_value);
```

ACOS

Arccosine

Description ACOS calculates the arccosine of the input variable and writes the angle data in radians into the output variable. The function returns a value from 0.0 to π.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of ACOS (see page 923)**

Data types

Data type	I/O	Function
REAL	input	input value between -1 and +1
REAL	output as input	arccosine of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not ≥ -1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

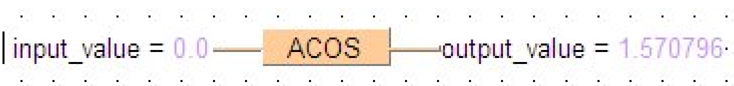
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between -1 and +1
1	VAR	output_value	REAL	0.0	angle data in radians 0.0 to pi

This example uses variables. You may also use a constant for the input variable.

Body

The arc cosine of **input_value** is calculated and written into **output_value**.

LD



```
ST   output_value:=ACOS(input_value);
```

TAN Tangent

Description TAN calculates the tangent of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of TAN (see page 935)**

Data types

Data type	I/O	Function
REAL	input	input value in radians
REAL	output as input	tangent of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	tangent

This example uses variables. You may also use a constant for the input variable.

Body

The tangent of **input_value** is calculated and written into **output_value**.

LD

ATAN Arctangent

Description ATAN calculates the arctangent of the input variable (value ± 52707176) and writes the angle data in radians into the output variable. The function returns a value greater than -π/2 and smaller than π/2.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.

PLC types: **Availability of ATAN (see page 923)**

Data types

Data type	I/O	Function
REAL	input	input value between -52707176 and +52707176
REAL	output as input	arctangent of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between +/-52707176
1	VAR	output_value	REAL	0.0	angle in radians >-Pi/2 and < Pi/2

This example uses variables. You may also use a constant for the input variable.

Body The arc tangent of **input_value** is calculated and written into **output_value**.

LD

ST
```
output_value:=ATAN(input_value);
```

LN Natural logarithm

Description LN calculates the logarithm of the input variable (value > 0.0) to the base e (Euler's number = 2.7182818) and writes the result into the output variable. This function is the reversion of the EXP (see page 57) function.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of LN** **(see page 933)**

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	natural logarithm of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number > 0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You may also use a constant for the input variable.

Body

The logarithm of **input_value** is calculated to the base e and written into **output_value**.

LD

ST output_value:=LN(input_value);

LOG

Logarithm to the Base 10

Description LOG calculates the logarithm of the input variable (value > 0.0) to the base 10 and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.

PLC types: **Availability of LOG (see page 933)**

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	logarithm of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

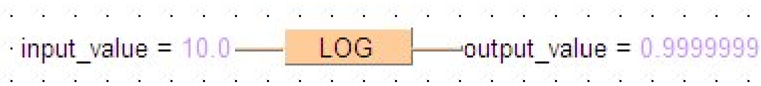
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number>0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You may also use a constant for the input variable.

Body

The logarithm of **input_value** is calculated to the base 10 and written into **output_value**.

LD



ST
```
output_value:=LOG(input_value);
```

EXP

Exponent of input variable to base e

Description

EXP calculates the power of the input variable to the base e (Euler's number = 2.7182818) and writes the result into the output variable. The input variable has to be greater than -87.33 and smaller than 88.72. This function is the reversion of the LN (see page 53) function.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

PLC types: **Availability of EXP (see page 925)**

Data types

Data type	I/O	Function
REAL	input	input value between -87.33 and +88.72
REAL	output as input	exponent of input variable to base e

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL or input variable is not > -87.33 and < 88.72
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

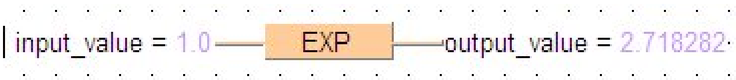
	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	REAL	0.0	>-87.33 and < 88.72
1	VAR	output_value	REAL	0.0	number >0

This example uses variables. You may also use a constant for the input variable.

Body

The power of **input_value** is calculated to the base e and written into **output_value**.

LD



ST

```
output_value:=EXP(input_value);
```

EXPT

Raises 1st input variable by the power of the 2nd input variable

Description EXPT raises the first input variable to the power of the second input variable (OUT = $IN1^{IN2}$) and writes the result into the output variable. Input variables have to be within the range -1.70141 x 10 E^{38} to 1.70141 x 10 E^{38}.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of EXPT (see page 925)**

Data types

Data type	I/O	Function
REAL	1st input	input value
REAL	2nd input	exponent of the input value
REAL	output as 1st input	result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- first and the second input variable do not have the data type REAL
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- output variable is zero
R9009	%MX0.900.9	for an instant	- processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

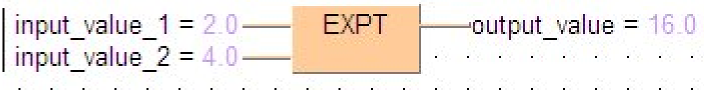
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initia	Comment
0	VAR	input_value_1	REAL	0.0	number from -1.70141x10^38 to 1.70141x10^38
1	VAR	input_value_2	REAL	0.0	number from -1.70141x10^38 to 1.70141x10^38
2	VAR	output_value	REAL	0.0	number from -1.70141x10^38 to 1.70141x10^38

In this example the input variables (**input_value_1** and **input_value_2**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **Input_value_1** is raised to the power of **input_value_2**. The result is written into **output_value**.

LD



ST

```
output_value:=input_value_1**input_value_2;
```

CRC16 Cyclic Redundancy Check

Description This function calculates the CRC16 (**C**yclic **R**edundancy **C**heck) for all PLC types by using 8 bytes (8 bits) specified with the parameter **NumberOfBytes** and the starting address **StartAddress**.

Symbol:

Depending on the PLC type, one of the following two implementations of the function will be used:

PLCs which support the instruction F70_BCC (see page 426) with the parameter s1=10 to calculate CRC16 (FP-Sigma) use F70_BCC directly.

☞ **The number of steps can increase up to approx. 200 when CRC16 is used as a sub-program.**

PLC types: **Availability of CRC16** **(see page 924)**

Data types

Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
StartAddress	ANY	Starting address for the calculation of the check sum. For PLCs which do not support the instruction F70_BCC (see page 426) with CRC16 calculation (), the starting address must be in the DT or FL area.
NumberOfBytes	INT	The number of bytes (8 bits), beginning with AdrStart, on which the CRC16 calculation is performed.
Output variables (VAR_OUTPUT):		
CRC	ANY16	The calculated check sum, which is only valid if the flag **IsValid** is set to TRUE.
IsValid	BOOL	Flag indicating whether the calculated check sum is valid or not. For PLCs which do not support the instruction F70_BCC (see page 426) with CRC16 calculation () the CRC is not valid: - during the first eight execution scans when an internal table is built - if the address area of the variable StartAddress is not in the DT or FL area. For PLCs that support the instruction F70_BCC with CRC16 calculation, the CRC is always valid.

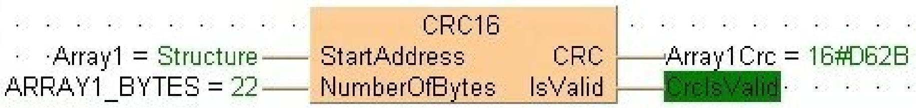
Example In this example, the same POU header is used for all programming languages.

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..10] OF INT	[0,1,2,3,4,5,6,7,8,9,10]
1	VAR	ARRAY1_BYTES	INT	22
2	VAR	Array1Crc	WORD	0
3	VAR	CrcIsvalid	BOOL	FALSE

Body LD



ST

```
CRC16(StartAddress := Array1,
      NumberOfBytes := ARRAY1_BYTES,
      CRC => Array1Crc,
      IsValid => CrcIsvalid);
```

Chapter 4

Bitwise Boolean Instructions

AND Logical AND operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical AND operation. The result is transferred to the accumulator.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of AND (see page 923)**

- **All operands must be of the same data type.**
- **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical AND operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	0	1	0
	1	0	0
	1	1	1

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

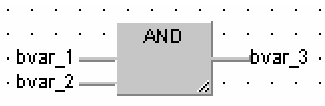
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	input 1
1	VAR	bvar_2	BOOL	FALSE	input 2
2	VAR	bvar_3	BOOL	FALSE	output

Body

bvar_1 will be logically AND-linked with **bvar_2**. The result will be written into the output variable **bvar_3**.

LD



ST

```
bvar_3:= bvar_1&bvar_2;
```

OR

Logical OR operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical OR operation. The result is transferred to the accumulator.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of OR (see page 934)**

- **All operands must be of the same data type.**
- **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical OR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	1	0	1
	0	1	1
	1	1	1

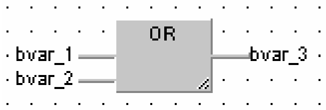
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	input 1
1	VAR	bvar_2	BOOL	FALSE	input 2
2	VAR	bvar_3	BOOL	FALSE	output

Body **bvar_1** and **bvar_2** are linked with a logical OR. The result will be written in **bvar_3**. This example uses variables. You may also use constants for the input variables.

LD



ST

```
bvar_3:= var_1 OR bvar_2;
```

XOR

Exclusive OR operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical XOR operation. The result is transferred to the accumulator.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of XOR (see page 936)**

- **All operands must be of the same data type.**
- **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical XOR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	1	0	1
	0	1	1
	1	1	0

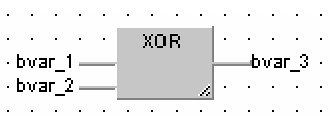
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	input 1
1	VAR	bvar_2	BOOL	FALSE	input 2
2	VAR	bvar_3	BOOL	FALSE	output

Body The Boolean variables **bvar_1** and **bvar_2** are logically EXCLUSIVE-OR linked and the result is written in **bvar_3.**

LD



ST

```
var_3:= var_1 XOR var_2;
```

NOT

Bit inversion

Description NOT performs a bit inversion of input variables. The result will be written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of NOT (see page 934)**

☞ **All operands must be of the same data type.**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	input	input for NOT operation
BOOL, WORD, DWORD	output as input	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

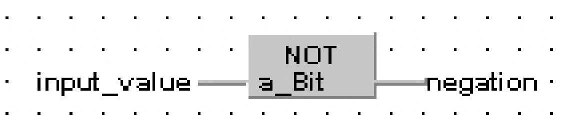
	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	WORD	0	type:BOOL, WORD or DWORD
1	VAR	negation	WORD	0	type:BOOL, WORD or DWORD

This example uses variables. You may also use a constant for the input variable.

Body

The bits of **input_value** are inversed (0 is inversed to 1 and vice versa). The inversed result is written into **negation.**

LD



ST

```
negation:= NOT(input_value);
```

Chapter 5

Bitshift Instructions

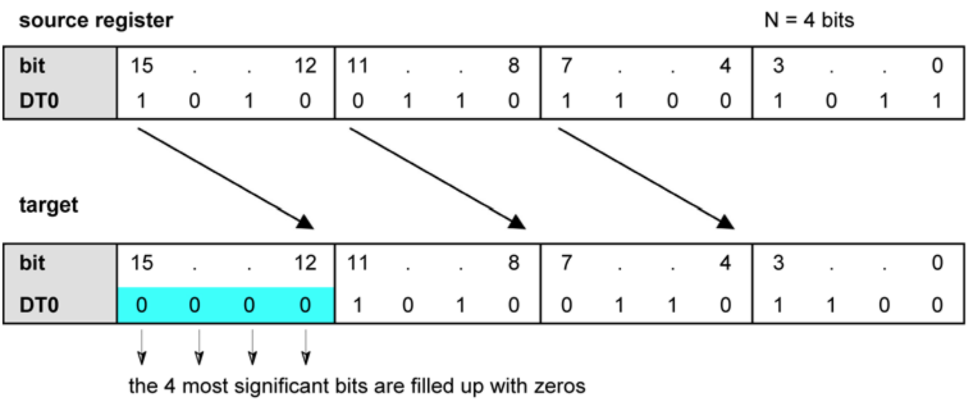
SHR

Shift bits to the right

Description SHR shifts a bit value by a defined number of positions (N) to the right and fills the vacant positions with zeros.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Bit shift to the right, zero-filled on left:



PLC types: **Availability of SHR (see page 934)**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is shifted to the right
BOOL, WORD, DWORD	output as input	result

- **If the second input variable N (the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

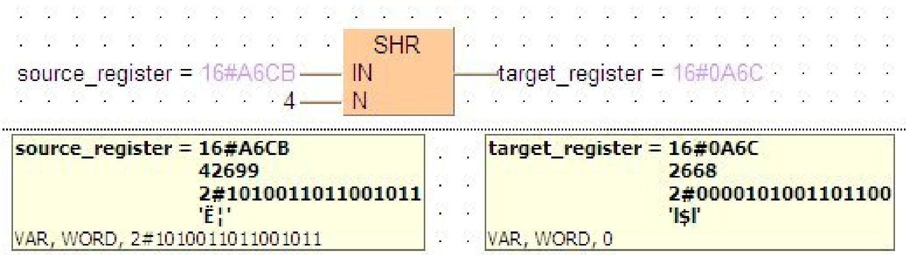
	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

The last N bits (here 4) of **source_register** are right-shifted. The vacant positions on the left are filled with zeros. The result is written into **target_register**.

LD

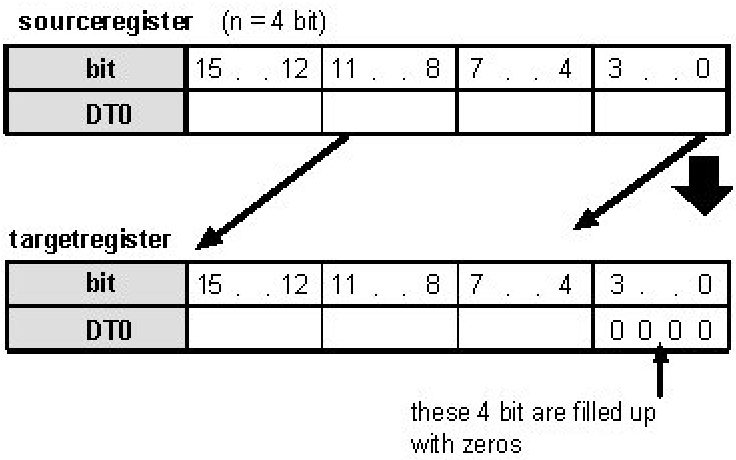


SHL Shift bits to the left

Description SHL shifts a bit value by a defined number of positions (N) to the left and fills the vacant positions with zeros.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Bit shift to the left, zero-filled on right:



PLC types: **Availability of SHL (see page 934)**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is shifted to the left
BOOL, WORD, DWORD	output as input	result

- **If the second input variable N (the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

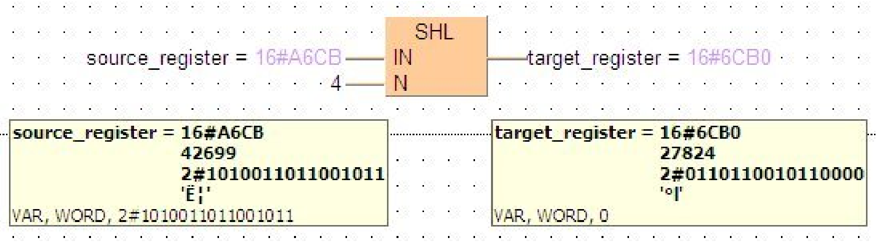
	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

The first N bits (here 3) of **source_register** are left-shifted, the vacant positions on the right are filled with zeros. The result is written into **target_register**.

LD

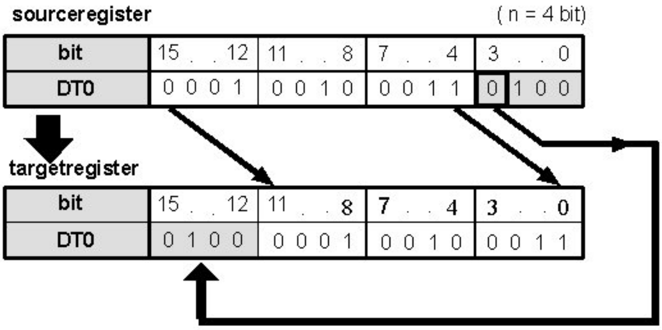


ROR

Rotate N bits the right

Description ROR rotates a defined number (N) of bits to the right.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.



PLC types: **Availability of ROR (see page 934)**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is rotated to the right
BOOL, WORD, DWORD	output as input	result

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

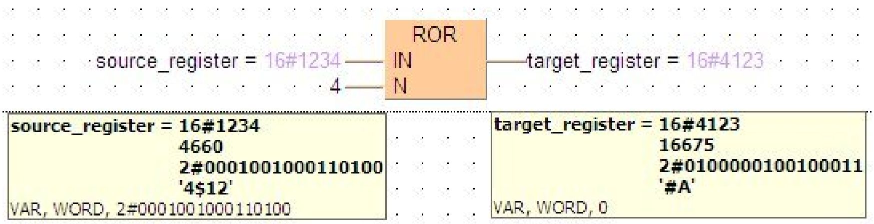
POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body The first N bits (here N = 3) of **source_register** are right-rotated. The result will be written into **target_register**.

LD

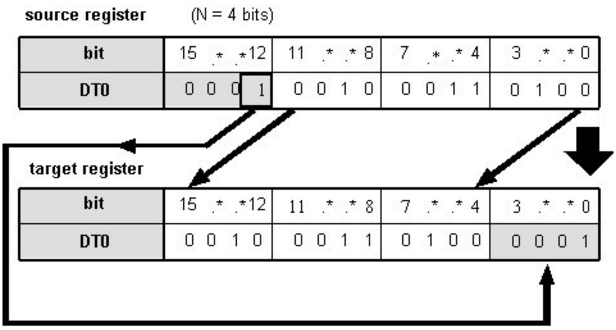


ROL

Rotate N bits to the left

Description ROL rotates a defined number (N) of bits to the left.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.



PLC types: **Availability of ROL (see page 934)**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is rotated to the left
BOOL, WORD, DWORD	output as input	result

☞ **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

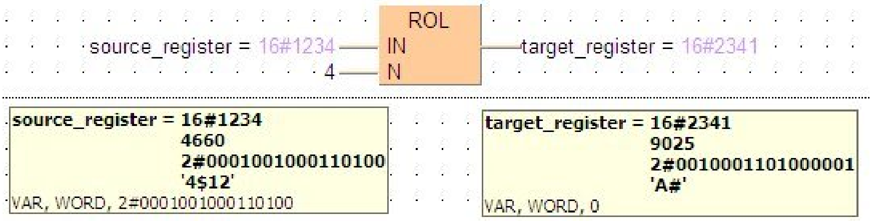
POU Header: All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body The last N bits (here 3) of **source_register** are left-rotated. The result will be written in **target_register**.

LD



Chapter 6

Comparison Instructions

GT Greater than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of GT** **(see page 932)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is greater than the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is greater than the second value AND the second value greater than third etc., TRUE will be written into result, otherwise FALSE.

Example IIn this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

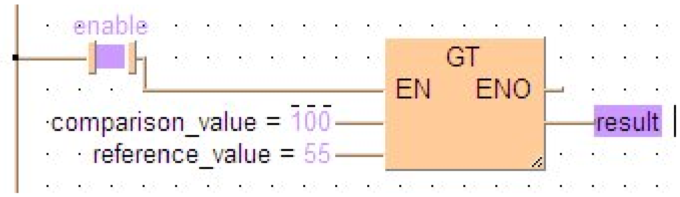
	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

LD



GE Greater than or equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of GE (see page 932)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is greater than or equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is greater than or equal to the second value AND the second value is greater than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

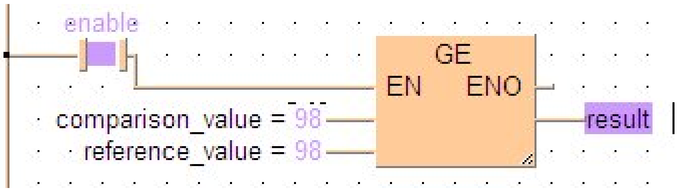
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than or equal to the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

LD



EQ Equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If both values are equal, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of EQ (see page 925)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is equal to the second value AND the second value is equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

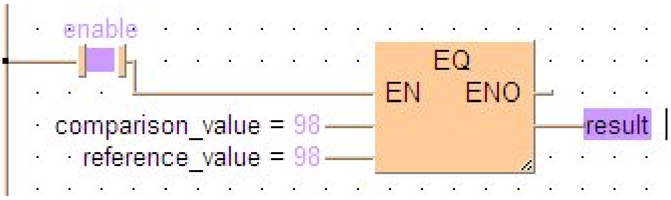
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the variable **comparison_value** is compared with the variable **reference_value**. If the values of the two variables are identical, the value TRUE will be written into **result**, otherwise FALSE.

LD



LE

Less than or equal to

Description

The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of LE (see page 933)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is less than or equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than or equal to the second value AND the second value is less than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

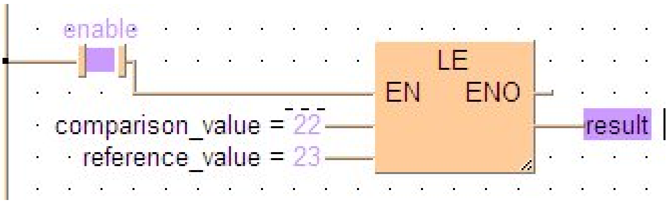
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the variable **reference_value**. If the **comparison_value** is less than or equal to the **reference_value**, TRUE will be written into **result**, otherwise FALSE.

LD



LT Less than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of LT (see page 933)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is less than the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than the second value AND the second value is less than the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

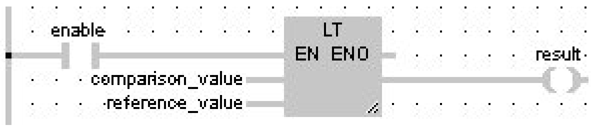
	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is less than or equal to the **reference_value**, TRUE will be written into **result**, otherwise FALSE.

LD



NE

Not equal

Description The content of the accumulator is compared with the operand defined in the operand field. If both values are not equal, "TRUE" is stored in the accumulator, otherwise "FALSE".

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of NE (see page 934)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is unequal to the reference value, otherwise FALSE

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is not equal to the second value AND the second value is not equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

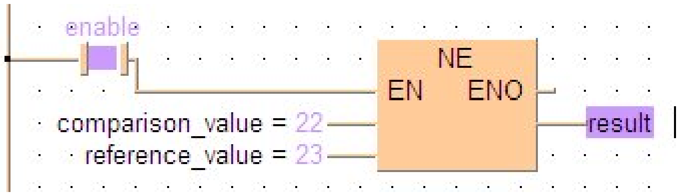
	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the two values are unequal, TRUE will be written into **result**, otherwise FALSE.

LD



Chapter 7

Conversion Instructions

WORD_TO_BOOL

WORD in BOOL

Description WORD_TO_BOOL converts a value of the data type WORD into a value of the data type BOOL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_BOOL (see page 936)**

If the value of WORD_value = 0 (16#0000), the conversion result will be = 0 (FALSE), in any other case = 1 (TRUE).

Data types

Data type	I/O	Function
WORD	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

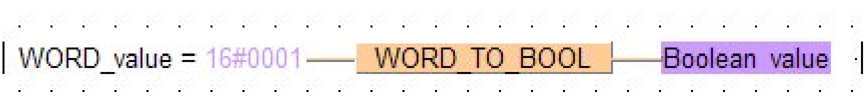
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD (16-bit) is converted into a Boolean value (1-bit). The result will be written into **Boolean_value**.

LD



ST

```
Boolean_value:=WORD_TO_BOOL(WORD_value);
```

DWORD_TO_BOOL DOUBLE WORD in BOOL

Description DWORD_TO_BOOL converts a value of the data type DOUBLE WORD into a value of the data type BOOL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_BOOL (see page 924)**

If the variable DWORD_value has the value 0 (16#00000000) the conversion result will be FALSE, in any other case it will be TRUE.

Data types

Data type	I/O	Function
DWORD	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	
1	VAR	Boolean_value	BOOL	FALSE	

This example uses variables. You may also use a constant for the input variable.

Body **DWORD_value** of the data type DOUBLE WORD is converted into a Boolean value (1-bit). the converted value is written into **Boolean_value**.

LD

ST

```
Boolean_value:=DWORD_TO_BOOL(DWORD_value);
```

INT_TO_BOOL

INTEGER into BOOL

Description INT_TO_BOOL converts a value of the data type INT into a value of the data type BOOL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_BOOL (see page 932)**

Data types

Data type	I/O	Function
INT	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

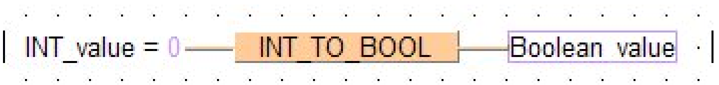
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value (16-bit) of the data type INTEGER is converted into a Boolean value. The result is written into **Boolean_value.**

LD



ST

```
Boolean_value:=INT_TO_BOOL(INT_value);
```

If INT_value has the value 0, the conversion result will be 0 (FALSE), in any other case it will be 1 (TRUE).

DINT_TO_BOOL

DOUBLE INTEGER into BOOL

Description DINT_TO_BOOL converts a value of the data type DINT into a value of the data type BOOL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_BOOL (see page 924)**

Data types

Data type	I/O	Function
DINT	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

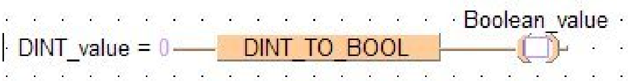
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	Boolean_value	BOOL	FALSE	

In this example the input variable (**DINT_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type BOOL. The converted value in written into **Boolean_value**.

LD



ST

```
Boolean_value:=DINT_TO_BOOL(DINT_value);
```

If the variable **DINT_value** has the value 0, the conversion result is FALSE, in any other case TRUE.

BOOL_TO_WORD

BOOL into WORD

Description BOOL_TO_WORD converts a value of the data type BOOL into a value of the data type WORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL_TO_WORD (see page 923)**

Data types

Data type	I/O	Function
BOOL	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

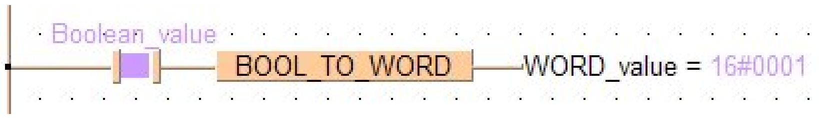
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	WORD_value	WORD	0	

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

The **Boolean_value** of the data type BOOL is converted into a value of the data type WORD. The converted value is written into **WORD_value**.

LD



ST

```
IF Boolean_value THEN
    WORD_value:=BOOL_TO_WORD(Boolean_value);
END_IF;
```

BOOL16_TO_WORD

BOOL16 to WORD

Description This function copies a variable of the special data type BOOL16 (an array with 16 elements of the data type BOOL or a DUT of 16 members of the data type BOOL) at the input to the data type WORD at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL16_TO_WORD** **(see page 923)**

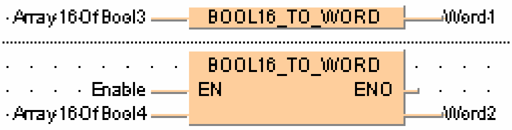
Data types

Data type	Comment
ARRAY (see page 20) of BOOL	ARRAY with 16 elements
WORD	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array16OfBool3	ARRAY [0..15] OF BOOL	[16(FALSE)]
2	VAR	Array16OfBool4	ARRAY [0..15] OF BOOL	[16(FALSE)]
3	VAR	Word1	WORD	0
4	VAR	Word2	WORD	0

Body with and without EN/ENO:



BOOLS_TO_WORD

16 Variables of the data type BOOL to WORD

Description This function converts 16 values of the data type BOOL bit-wise to a value of the data type WORD.

The inputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

PLC types: **Availability of BOOLS_TO_WORD (see page 923)**

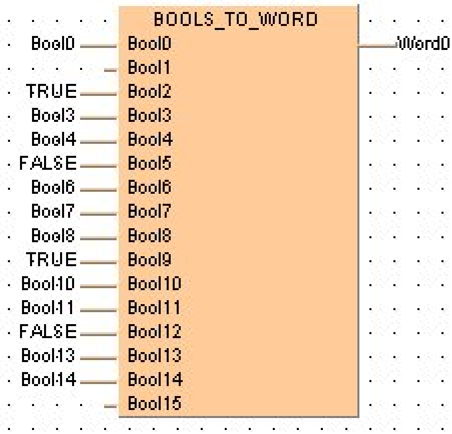
Data types

Variable	Data type	Function
BOOL0 ... BOOL15	BOOL	16 input variables of the data type BOOL
	WORD	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Word0	WORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE
11	VAR	Bool11	BOOL	FALSE
12	VAR	Bool12	BOOL	FALSE
13	VAR	Bool13	BOOL	FALSE
14	VAR	Bool14	BOOL	FALSE
15	VAR	Bool15	BOOL	FALSE

Body with and without EN/ENO:



DWORD_TO_WORD

DOUBLE WORD in WORD

Description DWORD_TO_WORD converts a value of the data type DOUBLE WORD into a value of the data type WORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_WORD (see page 924)**

☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
DWORD	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

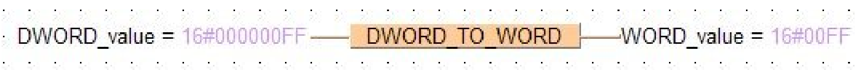
	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DOUBLE WORD (32-bit) is converted into a value of the data type WORD (16-bit). The converted value is written into **WORD_value**.

LD



ST

```
WORD_value:=DWORD_TO_WORD(DWORD_value);
```

INT_TO_WORD

INTEGER into WORD

Description INT_TO_WORD converts a value of the data type INT into a value of the data type WORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_WORD (see page 933)**

The bit combination of the input variable is assigned to the output variable.

Data types

Data type	I/O	Function
INT	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

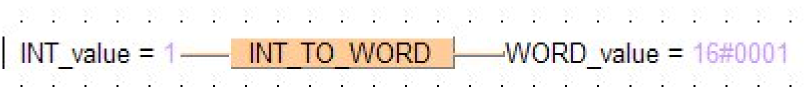
	Class	Identifier	Type	Initial	Comm...
0	VAR	WORD_value	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type WORD. The result is written into **WORD_value**.

LD



ST

```
WORD_value:=INT_TO_WORD(INT_value);
```

DINT_TO_WORD

DOUBLE INTEGER into WORD

Description

DINT_TO_WORD converts a value of the data type DINT into a value of the data type WORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_WORD (see page 924)**

☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
DINT	input	input data type
WORD	output	conversion result

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

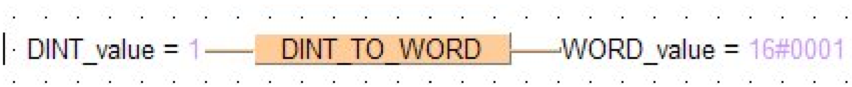
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER (32-bit) is converted into a value of the data type WORD (16-bit). The converted value is written into **WORD_value**.

LD



ST

```
WORD_value:=DINT_TO_WORD(DINT_value);
```

TIME_TO_WORD

TIME into WORD

Description TIME_TO_WORD converts a value of the data type TIME into a value of the data type WORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TIME_TO_WORD (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Examples:

Input variable	Output variable
T#123.4s	1234
T#1.00s	16#0064

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

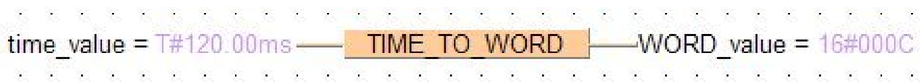
	Class	Identifier	Type	Initial
0	VAR	time _value	TIME	T#0s
1	VAR	WORD_value	WORD	0

This example uses variables. You may also use a constant for the input variable.

Body

Time_value of the data type TIME is converted into a value of the data type WORD. The result will be written into the output variable **WORD_value**.

LD



ST

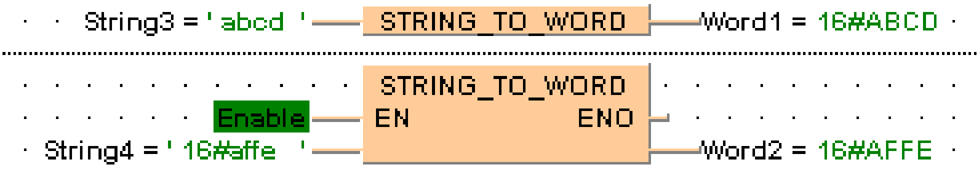
```
WORD_value:=TIME_TO_WORD(time_value);
```

STRING_TO_WORD STRING (hexadecimal format) to WORD

Description This function converts a STRING in hexadecimal format to a value of the data type WORD.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type WORD via a sub-program of approx. 270 steps that is also used in the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

Example with and without EN/ENO:



Permissible format:

'[Space][Hexadecimal numbers][Space]' e.g. ' afFE '

Permissible characters:

Space	All characters except for "+" (plus), "-" (minus) and all hexadecimal numbers
Hexadecimal numbers	Hexadecimal numbers in the ranges "0 - 9", "A - F" or "a - f".

The analysis ends with the first non-hexadecimal number.

PLC types: **Availability of STRING_TO_WORD (see page 935)**

Data types

Data type	Comment
STRING	input variable
WORD	output variable

STRING_TO_WORD _STEPSAVER

STRING (Hexadecimal Format right-justified) to WORD

Description This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format to a value of the data type WORD.

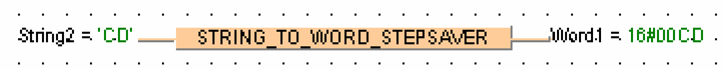
Examples

Input	Defined as	Results in
'D'	STRING[1]	16#D
'CD'	STRING[2]	16#CD
'BCD'	STRING[3]	16#BCD
'ABCD'	STRING[4]	16#ABCD
'0ABCD'	STRING[5]	16#ABCD
'00ABCD'	STRING[6]	16#ABCD

The basic instruction F72_A2HEX (see page 602) is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A-F” .

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example



Data types

Data type	Comment
STRING	Input variable
WORD	Output variable

Acceptable Format for STRING[4]:

'Hex1Hex2Hex3Hex4' e.g. perhaps 'AFFE'

Acceptable characters:

Hex1 to Hex4	Hexadecimal numbers in the range "0 - 9“ or "A - F“ (not "a - f").

PLC types: **Availability of STRING_TO_WORD_STEPSAVER (see page 935)**

BOOL_TO_DWORD

BOOL into DOUBLE WORD

Description BOOL_TO_DWORD converts a value of the data type BOOL into a value of the data type DWORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL_TO_DWORD (see page 923)**

Data types

Data type	I/O	Function
BOOL	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

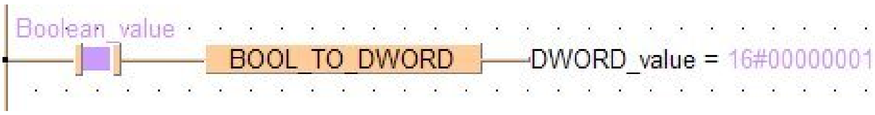
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	DWORD_value	DWORD	0	

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

The **Boolean_value** of the data type BOOL is converted into a value of the data type DOUBLE INTEGER. The converted value is written into **DWORD_value**.

LD



ST

```
IF Boolean_value THEN
      DWORD_value:=BOOL_TO_DWORD(Boolean_value);
END_IF;
```

BOOL32_TO_DWORD **BOOL32 to DOUBLE WORD**

Description This function copies a variable of the special data type BOOL32 (an array with 32 elements of the data type BOOL or a DUT of 32 members of the data type BOOL) at the input to the data type DWORD at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL32_TO_DWORD (see page 923)**

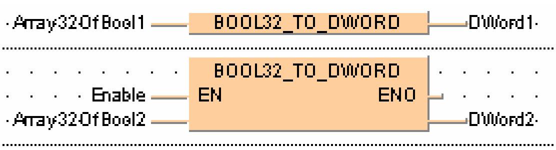
Data types

Data type	Comment
ARRAY (see page 20) of BOOL	ARRAY with 32 elements
DWORD	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31] OF BOOL	[32(FALSE)]
2	VAR	Array32OfBool2	ARRAY [0..31] OF BOOL	[32(FALSE)]
3	VAR	DWord1	DWORD	0
4	VAR	DWord2	DWORD	0

Body with and without EN/ENO:



BOOLS_TO_DWORD

32 Variables of the data type BOOL to DWORD

Description This function converts 32 values of the data type BOOL bit-wise to a value of the data type DWORD.

The inputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

PLC types: **Availability of BOOLS_TO_DWORD (see page 923)**

Data types

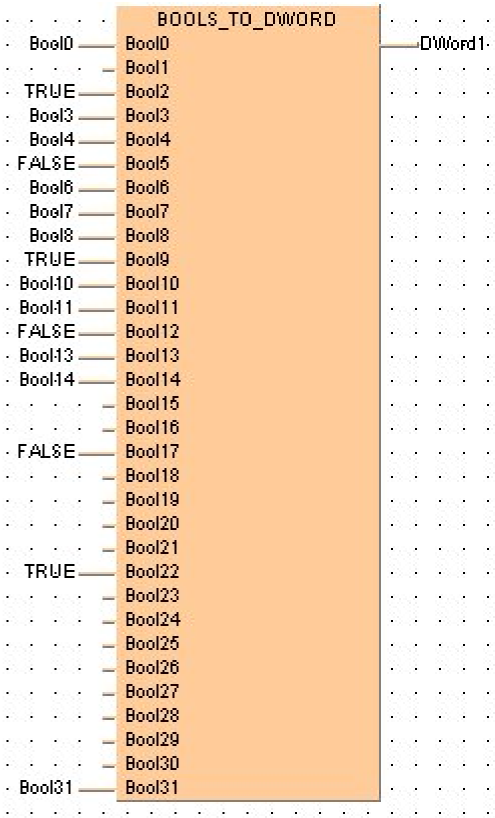
Variable	Data type	Function
BOOL0 ... BOOL31	BOOL	32 input variables of the data type BOOL
	DWORD	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	dWord1	DWORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE

etc. to Bool31

Body with and without EN/ENO:



WORD_TO_DWORD

WORD in DOUBLE WORD

Description WORD_TO_DWORD converts a value of the data type WORD into a value of the data type DWORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_DWORD (see page 936)**

The bit combination of WORD_value is assigned to DWORD_value.

Data types

Data type	I/O	Function
WORD	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

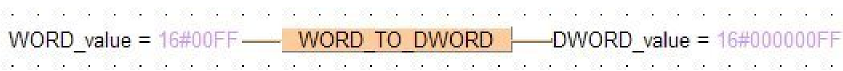
	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	DWORD_value	DWORD	0

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type DOUBLE WORD. The result will be written into **DWORD_value**.

LD



ST

```
DWORD_value:=WORD_TO_DWORD(WORD_value);
```

INT_TO_DWORD

INTEGER into DOUBLE WORD

Description INT_TO_DWORD converts a value of the data type INT into a value of the data type DWORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_DWORD (see page 933)**

Data types

Data type	I/O	Function
INT	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type DOUBLE WORD (32-bit). The result is written into **DWORD_value**.

LD

ST

```
DWORD_value:=INT_TO_DWORD(INT_value);
```

DINT_TO_DWORD

DOUBLE INTEGER into DOUBLE WORD

Description DINT_TO_DWORD converts a value of the data type DINT into a value of the data type DWORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_DWORD (see page 924)**

The bit combination of the input variable is assigned to the output variable.

Data types

Data type	I/O	Function
DINT	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type DOUBLE WORD. The converted value is written into **DWORD_value**.

LD

ST

```
DWORD_value:=DINT_TO_DWORD(DINT_value);
```

TIME_TO_DWORD

TIME into DOUBLE WORD

Description TIME_TO_DWORD converts a value of the data type TIME into a value of the data type DWORD. The time 10ms corresponds to the value 1, e.g. an input value of T#1s is converted to the value 100 (16#64).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TIME_TO_DWORD (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

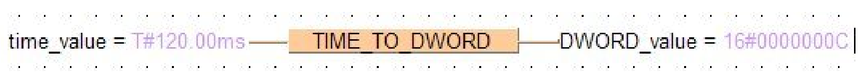
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#120ms	
1	VAR	DWORD_value	DWORD	0	result: 16#C

This example uses variables. You may also use a constant for the input variable.

Body **time_value** of the data type TIME is converted to value of the data type DWORD and written into the output variable **DWORD_value**.

LD



ST

```
DWORD_value:=TIME_TO_DWORD(time_value);
```

STRING_TO_DWORD

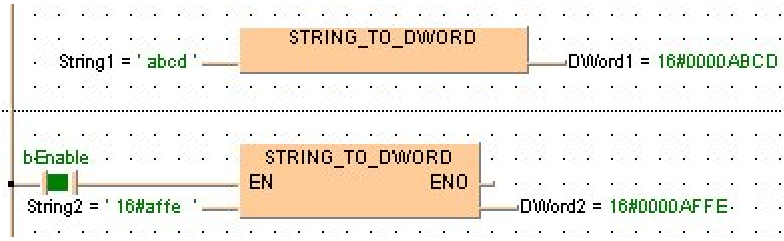
STRING (Hexadecimal Format) to DOUBLE WORD

Description This function converts a string in hexadecimal formal to a value of the data type DWORD.

At first the string is converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD in a subprogram of approximately 270 steps, which is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

See also: STRING_TO_DWORD_STEPSAVER

Example with and without EN/ENO:



Acceptable Format:

´[Space][Hexadecimal number][Space]´ e.g. perhaps ' afFE '

Acceptable characters:

Space	Space " "
Signs	Plus "+" and minus "-"
Hexadecimal numbers	Hexadecimal numbers in the range "0 - 9" or "A - F" or "a - f".

The analysis ends with the first non-decimal number.

PLC types: **Availability of STRING_TO_DWORD (see page 935)**

Data types

Data type	Comment
STRING	Input variable
DWORD	Output variable

STRING_TO_DWORD_STEPSAVER

STRING (Hexadecimal Format right-justified) to DOUBLE WORD

Description This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format to a value of the data type DWORD.

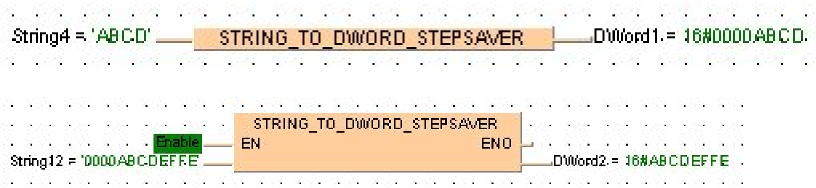
Examples:

Input	Defined as	Results in
'FE'	STRING[2]	16#FE
'EFFE'	STRING[4]	16#EFFE
'CDEFFE'	STRING[6]	16#CDEFFE
'ABCDEFFE'	STRING[8]	16#ABCDEFFE
'00ABCDEFFE'	STRING[10]	16#ABCDEFFE

The basic instruction F72_A2HEX (see page 602) is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A - F” .

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example



Data types

Data type	Comment
STRING	Input variable
DWORD	Output variable

Acceptable Format for STRING[8]:

'Hex1Hex2Hex3Hex4Hex5Hex6Hex7Hex8' e.g. perhaps '001AAFFE'

Acceptable characters:

Hex1 to Hex8	Hexadecimal numbers in the range "0 - 9“ or "A - F“ (not "a - f").

PLC types: **Availability of STRING_TO_DWORD_STEPSAVER (see page 935)**

BOOL_TO_INT

BOOL into INTEGER

Description BOOL_TO_INT converts a value of the data type BOOL into a value of the data type INT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL_TO_INT (see page 923)**

Data types

Data type	I/O	Function
BOOL	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

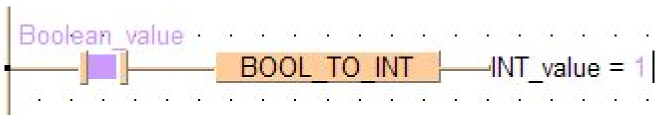
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	INT_value	INT	0	

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

The **Boolean_value** of the data type BOOL is converted into a value of the data type INTEGER. The converted value is written into **INT_value**.

LD



ST

```
IF Boolean_value THEN
    INT_value:=BOOL_TO_INT(Boolean_value);
END_IF;
```

BOOL16_TO_INT

BOOL16 to INTEGER

Description This function copies a variable of the special data type BOOL16 (an array with 16 elements of the data type BOOL or a DUT of 16 members of the data type BOOL) at the input to the data type INT at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL16_TO_INT (see page 923)**

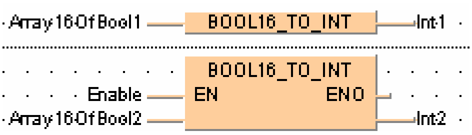
Data types

Data type	Comment
ARRAY (see page 20) of BOOL	ARRAY with 16 elements
INT	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array16OfBool1	ARRAY [0..15] OF BOOL	[16(FALSE)]
2	VAR	Array16OfBool2	ARRAY [0..15] OF BOOL	[16(FALSE)]
3	VAR	Int1	INT	0
4	VAR	Int2	INT	0

LD body with and without EN/ENO:



BOOLS_TO_INT 16 Variables of the data type BOOL to INT

Description This function converts 16 values of the data type BOOL bit-wise to a value of the data type INT.

The inputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

PLC types: **Availability of BOOLS_TO_INT (see page 923)**

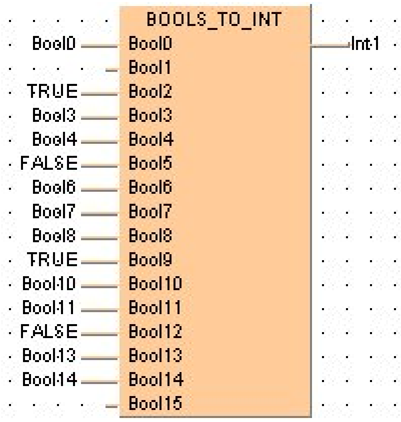
Data types

Variable	Data type	Function
BOOL0 ... BOOL15	BOOL	16 input variables of the data type BOOL
	INT	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Int1	INT	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE
11	VAR	Bool11	BOOL	FALSE
12	VAR	Bool12	BOOL	FALSE
13	VAR	Bool13	BOOL	FALSE
14	VAR	Bool14	BOOL	FALSE
15	VAR	Bool15	BOOL	FALSE

Body with and without EN/ENO:



WORD_TO_INT

WORD in INTEGER

Description WORD_TO_INT converts a value of the data type WORD into a value of the data type INT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_INT (see page 936)**

The bit combination of WORD_value is assigned to INT_value.

Data types

Data type	I/O	Function
WORD	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

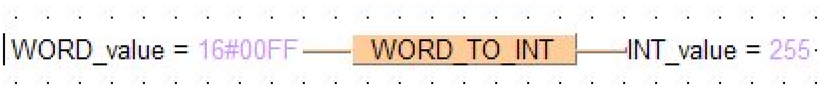
	Class	Identifier	Type	Initial	Comm...
0	VAR	WORD_value	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type INTEGER. The result will be written into **INT_value**.

LD



ST

```
INT_value:=WORD_TO_INT(WORD_value);
```

BCD_TO_INT

BCD into INTEGER

Description BCD_TO_INT converts binary coded decimal numbers (BCD) into binary values of the type INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BCD_TO_INT (see page 923)**

Data types

Data type	I/O	Function
WORD	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	BCD_value_16bit	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

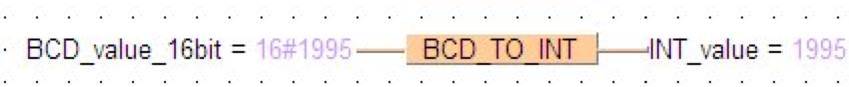
BCD constants can be indicated in Control FPWIN Pro as follows:

2#0001100110010101 or
16#1995

Body

BCD_value_16bit of the data type WORD is converted into an INTEGER value. The converted value is written into output variable **INT_value**.

LD



ST

```
INT_value:=BCD_TO_INT(BCD_value_16bit);
```

DWORD_TO_INT

DOUBLE WORD in INTEGER

Description DWORD_TO_INT converts a value of the data type DWORD into a value of the data type INT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_INT (see page 924)**

The first 16 bits of the input variable are assigned to the output variable.

Data types

Data type	I/O	Function
DWORD	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	
1	VAR	INT_value	INT	0	

In this example the input variable (**DWORD _value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

DWORD_value of the data type DOUBLE WORD (32-bit) is converted into an INTEGER value (16-bit). The converted value is written into **INT_value**.

LD

ST

```
INT_value:=DWORD_TO_INT(DWORD_value);
```

DINT_TO_INT

DOUBLE INTEGER into INTEGER

Description DINT_TO_INT converts a value of the data type DINT into a value of the data type INT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_INT (see page 924)**

The value of the input variable should be between -32768 and 32767.

Data types

Data type	I/O	Function
DINT	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER (32-bit) is converted into a value of the data type INTEGER (16-bit). The converted value is written into **INT_value**.

LD

ST

```
INT_value:=DINT_TO_INT(DINT_value);
```

REAL_TO_INT

REAL into INTEGER

Description REAL_TO_INT converts a value of the data type REAL into a value of the data type INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of REAL_TO_INT (see page 934)**

Data types

Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

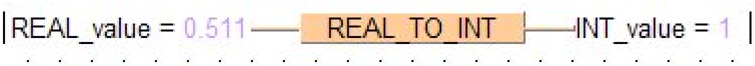
POU Header: All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	REAL_value	REAL	0.0
1	VAR	INT_value	INT	0

This example uses variables. You may also use a constant for the input variable.

Body: **REAL_value** of the data type REAL is converted into a value of the data type INTEGER. The converted value is stored in **INT_value**.

LD



ST

```
INT_value:= REAL_TO_INT(REAL_value);
```

TRUNC_TO_INT

Truncate (cut off) decimal digits of REAL input variable, convert to INTEGER

Description TRUNC_TO_INT cuts off the decimal digits of a REAL number and delivers an output variable of the data type INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TRUNC_TO_INT (see page 936)**

- **Cutting off the decimal digits decreases a positive number towards zero and increases a negative number towards zero.**
- **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	- output variable is greater than a 16-bit INTEGER
R9009	%MX0.900.9	for an instant	- output variable is zero

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

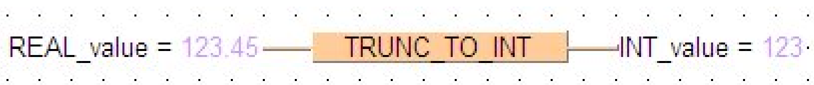
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw. -32768.99 ... +32767
1	VAR	INT_value	INT	0	number betw. -32768 ... +32768

This example uses variables. You may also use a constant for the input variable.

Body

The decimal digits of **REAL_value** are cut off. The result is stored as a 16-bit INTEGER in **INT_value**.

LD



```
ST   INT_value:=TRUNC_TO_INT(REAL_value);
```

TIME_TO_INT

TIME into INTEGER

Description TIME_TO_INT converts a value of the data type TIME into a value of the data type INT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TIME_TO_INT (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

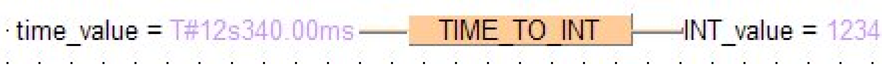
	Class	Identifier	Type	Initial
0	VAR	time _value	TIME	T#0s
1	VAR	INT_value	INT	0

This example uses variables. You may also use a constant for the input variable.

Body

Time_value of the data type TIME is converted into a value of the data type INTEGER. The result will be written into the output variable **INT_value**.

LD



ST

```
INT_value:=TIME_TO_INT(time_value);
```

STRING_TO_INT

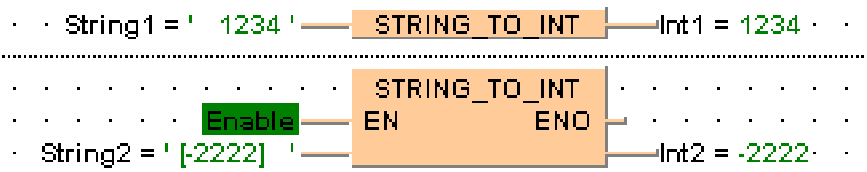
STRING (decimal format) to INTEGER

Description

This function converts a STRING in decimal format to a value of the data type INT.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type INT via a sub-programm of approx. 270 steps that is also used in the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

Example with and without EN/ENO:



Permissible format:

'[Space][Sign][Decimal numbers][Space]' e.g. ' 123456 '

Permissible characters:

Space	All characters except for "+“ (plus), "-" (minus) and all decimal numbers
Sign	"+“ (plus), "-" (minus)
Decimal numbers	Decimal numbers "0 - 9“

The analysis ends with the first non-decimal number.

PLC types: **Availability of STRING_TO_INT (see page 935)**

Data types

Data type	Comment
STRING	input variable
INT	output variable

STRING_TO_INT_STEPSAVER

STRING (Decimal Format right-justified) to INTEGER

Description This function converts a right-justifed decimal number in a string to a value of the data type INT.

The basic instruction F76_A2BIN (see page 616) with approx. 7 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number "0 - 9", not a "+" or "- " or not a space.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example

Acceptable Format:

'[Space][Sign][Decimal number]' e.g. ' 123456'

Acceptable characters:

Space	Space "␣"
Signs	Plus "+" and minus "-"
Decimal Number	Decimal numbers "0" - "9"

PLC types: **Availability of STRING_TO_INT_STEPSAVER (see page 935)**

Data types

Data type	Comment
STRING	Input variable
INT	Output variable

BOOL_TO_DINT

BOOL into DOUBLE INTEGER

Description BOOL_TO_DINT converts a value of the data type BOOL into a value of the data type DINT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL_TO_DINT (see page 923)**

Data types

Data type	I/O	Function
BOOL	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

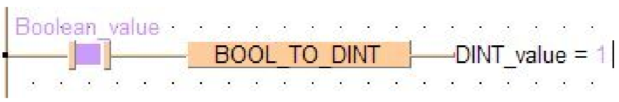
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	DINT_value	DINT	0	

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

The **Boolean_value** of the data type BOOL is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST

```
IF Boolean_value THEN
    DINT_value:=BOOL_TO_DINT(Boolean_value);
END_IF;
```

BOOL32_TO_DINT

BOOL32 to DOUBLE INTEGER

Description This function **copies a variable of the special data type BOOL32 (an array with 32 elements of the data type BOOL or a DUT of 32 members of the data type BOOL)** at the input to the data type DINT at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL32_TO_DINT (see page 923)**

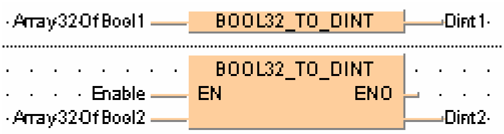
Data types

Data type	Comment
ARRAY (see page 20) of BOOL	ARRAY with 32 elements
DINT	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31] OF BOOL	[32(FALSE)]
2	VAR	Array32OfBool2	ARRAY [0..31] OF BOOL	[32(FALSE)]
3	VAR	Dint1	DINT	0
4	VAR	Dint2	DINT	0

Body with and without EN/ENO:



BOOLS_TO_DINT

32 Variables of the data type BOOL to DINT

Description This function converts 32 values of the data type BOOL bit-wise to a value of the data type DINT.

The inputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

PLC types: **Availability of BOOLS_TO_DINT (see page 923)**

Data types

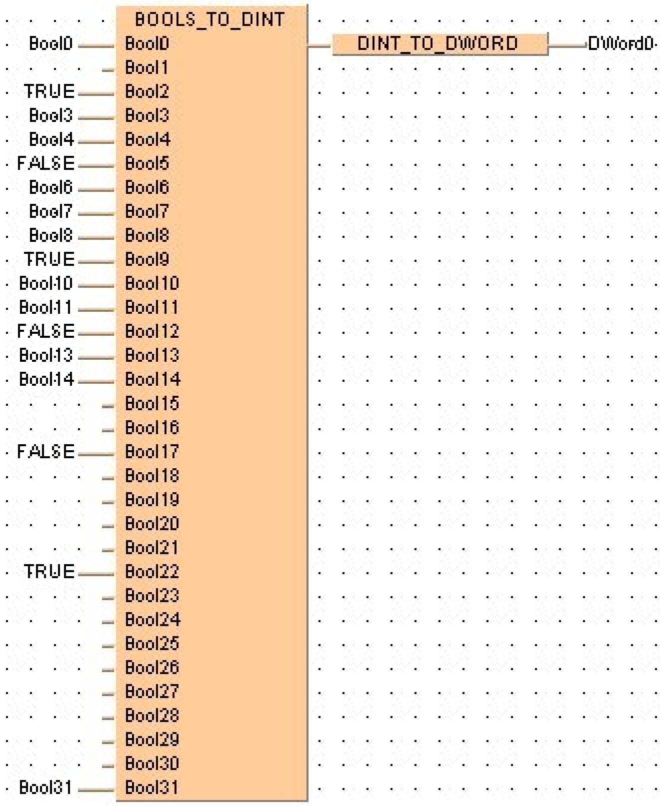
Variable	Data type	Function
BOOL0 ... BOOL31	BOOL	32 input variables of the data type BOOL
	DINT	output variable

POU header:

	Class	Identifier	Type	Initial
0	VAR	dWord1	DWORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE

etc. to Bool31

Body:


BOOLS_TO_DINT
DINT_TO_DWORD
DWord0
Bool0
TRUE
Bool3
Bool4
FALSE
Bool6
Bool7
Bool8
TRUE
Bool10
Bool11
FALSE
Bool13
Bool14
FALSE
TRUE
Bool31
Bool0
Bool1
Bool2
Bool3
Bool4
Bool5
Bool6
Bool7
Bool8
Bool9
Bool10
Bool11
Bool12
Bool13
Bool14
Bool15
Bool16
Bool17
Bool18
Bool19
Bool20
Bool21
Bool22
Bool23
Bool24
Bool25
Bool26
Bool27
Bool28
Bool29
Bool30
Bool31

WORD_TO_DINT

WORD in DOUBLE INTEGER

Description WORD_TO_DINT converts a value of the data type WORD into a value of the data type DINT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_DINT (see page 936)**

Data types

Data type	I/O	Function
WORD	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

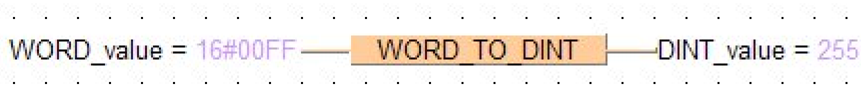
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type INTEGER. The result will be written into **DINT_value**.

LD



ST

```
DINT_value:=WORD_TO_DINT(WORD_value);
```

BCD_TO_DINT

BCD into DOUBLE INTEGER

Description BCD_TO_DINT converts a BCD value (binary coded decimal integer) of the data type DOUBLE WORD into a binary value of the data type DOUBLE INTEGER in order to be able to process a BCD value in the double word format.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BCD_TO_DINT (see page 923)**

Data types

Data type	I/O	Function
DWORD	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	BCD_value_32bit	DWORD	0	
1	VAR	DINT_value	DINT	0	

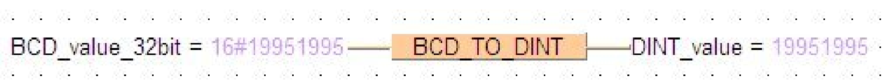
This example uses variables. You may also use a constant for the input variable.

BCD constants can be indicated in Panasonic MEW Control as follows:
2#0001100110010101000110011001010 1 or
16#19951995

Body

BCD_value_32bit of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST

```
DINT_value:=BCD_TO_DINT(BCD_value_32bit);
```

DWORD_TO_DINT DOUBLE WORD in DOUBLE INTEGER

Description DWORD_TO_DINT converts a value of the data type DOUBLE WORD into a value of the data type DOUBLE INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_DINT (see page 924)**

The bit combination of the input variable is assigned to the output variable.

Data types

Data type	I/O	Function
DWORD	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

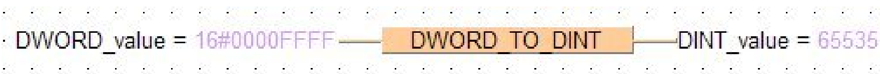
	Class	Identifier	Type	Initial
0	VAR	DWORD_value	DWORD	0
1	VAR	DINT_value	DINT	0

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST

```
DINT_value:=DWORD_TO_DINT(DWORD_value);
```

INT_TO_DINT INTEGER into DOUBLE INTEGER

Description INT_TO_DINT converts a value of the data type INT into a value of the data type DINT.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_DINT (see page 933)**

Data types

Data type	I/O	Function
INT	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

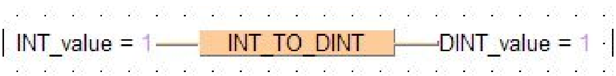
POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a value of the data type DOUBLE INTEGER. The result will be written into **DINT_value**.

LD



ST

```
DINT_value:=INT_TO_DINT(INT_value);
```

REAL_TO_DINT

REAL into DOUBLE INTEGER

Description REAL_TO_DINT converts a value of the data type REAL into a value of the data type DOUBLE INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of REAL_TO_DINT (see page 934)**

Data types

Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

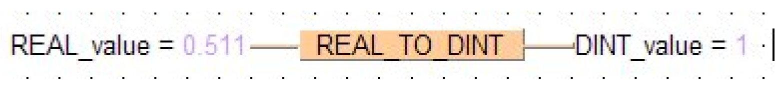
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constant for the input variable.

Body

REAL_value of the data type REAL is converted into a value of the data type DOUBLE INTEGER. The converted value is stored in **DINT_value**.

LD



ST

```
DINT_value:= REAL_TO_DINT(REAL_value);
```

TRUNC_TO_DINT

Truncate (cut off) decimal digits of REAL input variable, convert to DOUBLE INTEGER

Description TRUNC_TO_DINT cuts off the decimal digits of a REAL number and delivers an output variable of the data type DOUBLE INTEGER.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TRUNC_TO_DINT (see page 936)**

☞ **Cutting off the decimal digits decreases a positive number towards zero and increases a negative number towards zero.**

Data types

Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	- output variable is greater than a 32-bit DOUBLE INTEGER
R9009	%MX0.900.9	for an instant	- output variable is zero

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw. -2147483.000 ... +2147483.000
1	VAR	DINT_value	DINT	0	number betw. -2147483 ... +2147483

This example uses variables. You may also use a constant for the input variable.

Body

The decimal digits of **REAL_value** are cut off. The result is stored as a 32-bit DOUBLE INTEGER in **DINT_value**.

LD

ST

```
DINT_value:=TRUNC_TO_DINT(REAL_value);
```

TIME_TO_DINT

TIME into DOUBLE INTEGER

Description TIME_TO_DINT converts a value of the data type TIME into a value of the data type DINT. The time 10ms corresponds to the value 1, e.g. an input value of T#1m0s is converted to the value 6000.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TIME_TO_DINT (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

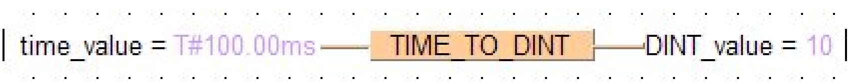
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#100ms	
1	VAR	DINT_value	DINT	0	result: 10

This example uses variables. You may also use a constant for the input variable.

Body

time_value of the data type TIME is converted to value of the data type DOUBLE INTEGER. The result is written into the output variable **DINT_value**.

LD



ST

```
DINT_value:=TIME_TO_DINT(time_value);
```

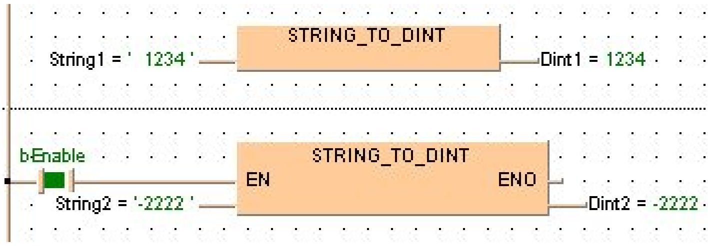
STRING_TO_DINT

STRING (Decimal Format) to DOUBLE INTEGER

Description This function converts a string in decimal formal to a value of the data type DINT.

At first the string is converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DINT in a subprogram of approximately 270 steps, which is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

Example with and without EN/ENO:



Acceptable Format:

'[Space][Sign][Decimal number][Space]' e.g. ' 123456 '

Acceptable characters:

Space	Space “ “
Signs	Plus “+” and minus “-“
Decimal Numbers	Decimal numbers "0“ - "9“

The analysis ends with the first non-decimal number.

PLC types: **Availability of STRING_TO_DINT** **(see page 935)**

Data types

Data type	Comment
STRING	Input variable
DINT	Output variable

STRING_TO_DINT_STEPSAVER

STRING (Decimal Format right-justified) to DOUBLE INTEGER

Description

This function converts a right-justifed decimal number in a string to a value of the data type DINT.

The basic instruction F78_DA2BIN (see page 622) with approx. 11 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number "0 - 9", not a "+" or "- " or not a space.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example

Acceptable Format:

'[Space][Sign][Decimal number]' e.g. ' 123456'

Acceptable characters:

Space	Space "␣"
Signs	Plus "+" and minus "-"
Decimal Numbers	Decimal numbers "0" - "9"

PLC types: **Availability of STRING_TO_DINT_STEPSAVER (see page 935)**

Data types

Data type	Comment
STRING	Input variable
DINT	Output variable

INT_TO_REAL

INTEGER into REAL

Description INT_TO_REAL converts a value of the data type INTEGER into a value of the data type REAL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_REAL (see page 933)**

Data types

Data type	I/O	Function
INT	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	REAL_value	REAL	0.0

In this example the input variable (**INT_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

INT_value of the data type INTEGER is converted into a value of the data type REAL.The converted value is stored in **REAL_value**.

LD

ST

```
REAL_value:=INT_TO_REAL(INT_value);
```

DINT_TO_REAL

DOUBLE INTEGER into REAL

Description DINT_TO_REAL converts a value of the data type DOUBLE INTEGER into a value of the data type REAL.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_REAL (see page 924)**

Data types

Data type	I/O	Function
DINT	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

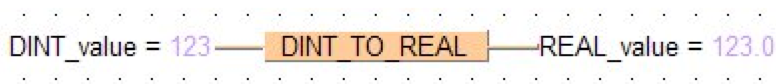
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constant for the input variable

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type REAL. The converted value is stored in **REAL_value**.

LD



ST

```
REAL_value:=DINT_TO_REAL(DINT_value);
```

TIME_TO_REAL TIME into REAL

Description TIME_TO_REAL converts a value of the data type TIME to a value of the data type REAL. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when TIME = 10ms, REAL = 1.0; when TIME = 1s, REAL = 100.0. The resolution amounts to 10ms.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of TIME_TO_REAL (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_time	TIME	T#1h1m1s	
1	VAR	result_time	REAL	0.0	result: here 366100.0

This example uses variables. You may also use a constant for the input variable.

LD

ST

```
result_real:=TIME_TO_REAL(input_time);
```

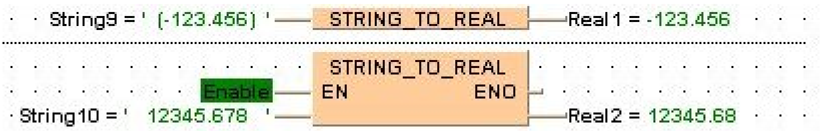
STRING_TO_REAL

STRING to REAL

Description function converts a STRING in floating-point format into a value of the data type REAL.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type REAL via a sub-program that requires approximately 290 steps.

Example with and without EN/ENO:



Permissible format:

'[Space][Sign][Decimal numbers].[Decimal numbers][Space]' e.g. ' -123.456 '

Permissible characters:

Space	All characters except for "+“ (plus), "-" (minus) and all decimal numbers
Decimal numbers	Decimal numbers "0“-"9“

The analysis ends with the first non-decimal number.

PLC types: **Availability of STRING_TO_REAL (see page 935)**

Data types

Data type	Comment
STRING	input variable
REAL	output variable

WORD_TO_TIME

WORD in TIME

Description WORD_TO_TIME converts a value of the data type WORD into a value of the data type TIME.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_TIME (see page 936)**

Data types

Data type	I/O	Function
WORD	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Examples:

Input variable	Output variable
12345	T#123.45s
16#0012	T#180.00ms

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

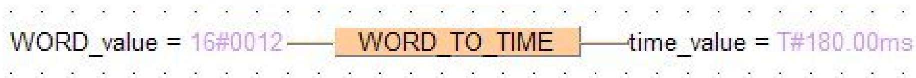
	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	time_value	TIME	T#0s

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD (16-bit) is converted into a value of the data type TIME (16-bit). The result will be written into the output variable **time_value.**

LD



ST

```
time_value:=WORD_TO_TIME(WORD_value);
```

DWORD_TO_TIME

DOUBLE WORD in TIME

Description DWORD_TO_TIME converts a value of the data type DWORD into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. the input value 12345 (16#3039) is converted to a TIME T#2m3s450.00ms.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_TIME (see page 924)**

Data types

Data type	I/O	Function
DWORD	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	example value: 16#3039
1	VAR	time_value	TIME	T#0s	result: T#2m3s450.00ms
2	VAR				

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DWORD (32-bit) is converted to value of the data type TIME (16-bit). The result is written into the output variable **time_value**.

LD

```
DWORD_value = 16#00003039 —— DWORD_TO_TIME —— time_value = T#2m3s450.00ms
```

ST

```
time_value:=DWORD_TO_TIME(DWORD_value);
```

INT_TO_TIME

INTEGER into TIME

Description INT_TO_TIME converts a value of the data type INT into a value of the data type TIME. The resolution is 10ms, e.g. when the INT value = 350, the TIME value = 3s500ms.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_TIME (see page 933)**

Data types

Data type	I/O	Function
INT	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

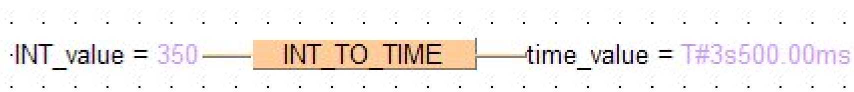
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	time_value	TIME	T#0s	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type TIME. The result will be written into the output variable **time_value**.

LD



ST

```
time_value:=INT_TO_TIME(INT_value);
```

DINT_TO_TIME

DOUBLE INTEGER into TIME

Description DINT_TO_TIME converts a value of the data type DINT into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. an input value of 123 is converted to a TIME T#1s230.00ms.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_TIME (see page 924)**

Data types

Data type	I/O	Function
DINT	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	time_value	TIME	T#0s	result: T#1s230.00ms

This example uses variables. You may also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER is converted to value of the data type TIME. The result is written into the output variable **time_value**.

LD

ST

```
time_value:=DINT_TO_TIME(DINT_value);
```

REAL_TO_TIME

REAL into TIME

Description REAL_TO_TIME converts a value of the data type REAL to a value of the data time TIME. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when REAL = 1.0, TIME = 10ms; when REAL = 100.0, TIME = 1s. The value of the data type real is rounded off to the nearest whole number for the conversion.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of REAL_TO_TIME (see page 934)**

Data types

Data type	I/O	Function
REAL	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list). Since constants are entered directly at the function's input contact pins, only the output variable need be declared in the header.

POU Header

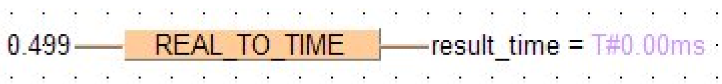
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	result_time	TIME	T#0s
1	VAR			

Body

By clicking on the monitor icon  while in the online mode, you can see the result 0.00ms immediately. Since the value at the REAL input contact is less than 0.5, it is rounded down to 0.0.

LD



ST

```
result_time:= REAL_TO_TIME(0.499);
```

BOOL_TO_STRING

BOOL into STRING

Description The function BOOL_TO_STRING converts a value of the data type BOOL to a value of the data type STRING[2]. The resulting string is represented by ' 0' or ' 1'.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of BOOL_TO_STRING (see page 923)**

Data types

Data type	I/O	Function
BOOL	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, input and output variables are declared that are used in the function.

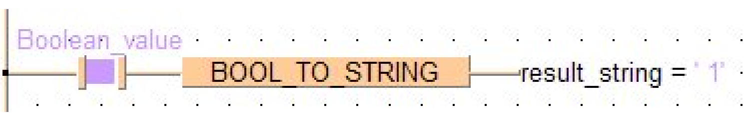
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	TRUE	example value
1	VAR	result_string	STRING[2]	''	result: here ' 1'

The input variable **input_value** of the data type BOOL is intialized by the value TRUE. The output variable **result_string** is of the data type STRING[2]. It can store a maximum of two characters. You can declare a character string that has more than one character, e.g. STRING[5]. From the 5 characters reserved, only 2 are used.
Instead of using the variable **input_value,** you can write the constants TRUE or FALSE directly to the function’s input contact in the body.

Body

The **input_value** of the data type BOOL is converted into STRING[2]. The converted value is written to **result_string**. When the variable **input_value** = TRUE, **result_string** shows ' 1'.

LD



ST
```
IF Boolean_value THEN
    output_value:=BOOL_TO_STRING(input_value);
END_IF;
```

Example 2: If you wish to have the result 'TRUE' or 'FALSE' instead of ' 0' or ' 1', you cannot use the function BOOL_TO_STRING. This example illustrates how you create a STRING[5] that contains the characters 'TRUE' or 'FALSE' from an input value of the data type BOOL.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

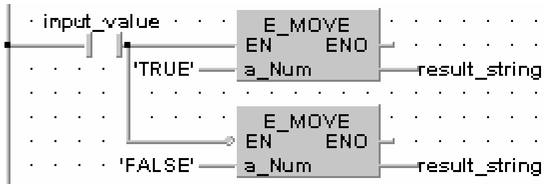
POU header

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	TRUE	example value
1	VAR	result_string	STRING[5]	''	result: here 'TRUE'

In this example, both an input variable **input_value** of the data type BOOL and an output variable **result_string** of the data type STRING[5] are declared.

Body In order to realize the intended operation, the standard function E_MOVE is used. It assigns the value of its input to its output unchanged. At the input, the STRING constant 'TRUE' or 'FALSE' is attached. In essence a "BOOL to STRING" conversion occurs, since the Boolian variable **input_variable** at the enable input (EN) contact decides the output of STRING.

LD



WORD_TO_STRING WORD into STRING

Description

The function WORD_TO_STRING converts a value of the data type WORD to a value of the data type STRING. It generates a result string in hexadecimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Explanation

Input	Output defined as	Results in
16#ABCD	STRING[1]	'D'
	STRING[2]	'CD'
	STRING[3]	'BCD'
	STRING[4]	'ABCD'
	STRING[5]	'0ABCD'
	STRING[6]	'00ABCD'
	and so on...	

PLC types: **Availability of WORD_TO_STRING (see page 936)**

Data types

Data type	I/O	Function
WORD	input	input data type
STRING	output	conversion result

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, input and output variables are declared that are used in the function.

	Class	Identifier	Type	Initial	Comment
0	VAR	WORD_value	WORD	0	example value
1	VAR	result_string	STRING[6]	''	result here: '00ABCD'

The input variable **input_value** of the data type WORD is intialized by the value 16#ABCD. The output variable **result_string** is of the data type STRING[6]. It can store a maximum of 6 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function’s input contact in the body.

Body The **input_value** of the data type WORD is converted into STRING[6]. The converted value is written to **result_string**. When the variable **input_value** = 16#ABCD, **result_string** shows '00ABCD'.

LD

ST

```
restult_string:=WORD_TO_STRING(input_value);
```

Example 2: This example illustrates how you create STRING[4] out of the data type WORD in which the leading part of the string '16#' is cut out.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU Header

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	WORD	16#1234	example value
1	VAR	result_string	STRING[7]	''	result: here '16#1234'

In this example, both an input variable **input_value** of the data type WORD and an output variable **result_string** of the data type STRING[4] are declared.

Body In carrying out the operation in question, the standard function RIGHT is attached to the function WORD_TO_STRING. RIGHT creates a right-justified character string of length L.

In the example, the output string of WORD_TO_STRING function is added at the input of the RIGHT function. At the L input of RIGHT, the INT constant 4 determines the length of the STRING to be replaced. Out of the variable **input_value** = 16#1234, the **result_string** 1234 results from the data type conversion and the RIGHT function.

LD

DWORD_TO_STRING DOUBLE WORD into STRING

Description The function DWORD_TO_STRING converts a value of the data type DWORD to a value of the data type STRING. It generates a result string in hexadecimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Explanation

Input	Output defined as	Results in
16#ABCDEFFE	STRING[2]	'FE'
	STRING[4]	'EFFE'
	STRING[6]	'CDEFFE'
	STRING[8]	'ABCDEFFE'
	STRING[10]	'00ABCDEFFE'
	STRING[12]	'0000ABCDEFFE'
	and so on...	

PLC types: **Availability of DWORD_TO_STRING (see page 924)**

Data types

Data type	I/O	Function
DWORD	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, input and output variables are declared that are used in the function.

	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	example value: 16#ABCDEFFE
1	VAR	result_string	STRING[10]	''	result: '00ABCDEFFE'

The input variable **input_value** of the data type DWORD is intialized by the value 16#ABCDEFFE. The output variable **result_string** is of the data type STRING[10]. It can store a maximum of 10 characters. Instead of using the variable **input_value,** you can

enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type DWORD is converted into STRING[10]. The converted value is written to **result_string**. When the variable **input_value** = 16#ABCDEFFE, **result_string** shows '00ABCDEFFE'.

LD

ST
```
result_string:=DWORD_TO_STRING(input_value);
```

Example 2: This example illustrates how you create STRING[10] out of the data type DWORD in which the leading part of the string '16#' is replaced by the string '0x'. The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU Header

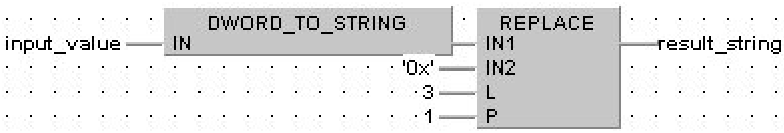
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DWORD	16#12345678	example value
1	VAR	result_string	STRING[10]	''	result: here '0x12345678'

In this example the input variables **input_value** of the data type DWORD and an output variable **result_string** of the data type STRING[10] are declared.

Body In carrying out the operation in question, the standard function REPLACE is attached to the function DWORD_TO_STRING. REPLACE replaces one section of a character string with another.

In the example, the output string of DWORD_TO_STRING function is added at input IN1 of the REPLACE function. At input IN2, the STRING constant '0x' is added as the replacement STRING. At the L input of REPLACE, the INT constant 3 determines the length of the STRING to be replaced. The P input determines the position at which the replacement begins. In this case it is the INT number 1. From the variable **input_value** = 16#12345678, the **result_string** = '0x12345678' results after undergoing the data type conversion and REPLACE function.

LD



INT_TO_STRING

INTEGER into STRING

Description

The function INT_TO_STRING converts a value of the data type INT to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading spaces up to the maximum number of characters defined for the string.

Explanation

Function used	String1 defined as	Result
String1:=INT_TO_STRING(-12345)	STRING[1]	'5'
	STRING[2]	'45'
	STRING[3]	'345'
	STRING[4]	'2345'
	STRING[5]	'12345'
	STRING[6]	'-12345'
	STRING[7]	'␣-12345'
	STRING[8]	'␣␣-12345'
	and so on...	

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_STRING (see page 933)**

Data types

Data type	I/O	Function
INT	input	input data type
STRING	output	conversion result

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

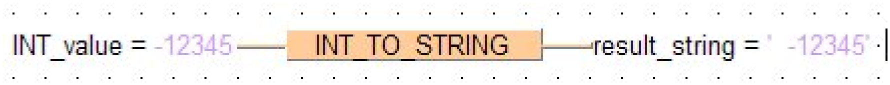
In the POU header, input and output variables are declared that are used in the function.

	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	-12345	example value
1	VAR	result_string	STRING[8]	''	result here: ' -12345'

The input variable **input_value** of the data type INT is intialized by the value -12345. The output variable **result_string** is of the data type STRING[8]. It can store a maximum of 8 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function’s input contact in the body.

Body The **input_value** of the data type INT is converted into STRING[8]. The converted value is written to **result_string**. When the variable **input_value** = -12345, **result_string** shows '␣␣-12345'.

LD



ST

```
result_string:= INT_TO_STRING(input_value);
```

Example 2: This example illustrates how you create a STRING[2] that appears right justified out of the data type INT.
The example is programmed in LD, ST and IL. The same POU header is used for both programming languages.

POU Header

	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	INT	12	example value
1	VAR	result_string	STRING[2]	''	result: here '12'

In this example, both an input variable input_value of the data type INT and an output variable result_string of the data type STRING[2] are declared.

Body In carrying out the operation in question, the standard function RIGHT (see page 208) is attached to the function INT_TO_STRING. RIGHT creates a right-justified character string with the length L.
In the example, the variable input_variable = 12 is converted by INT_TO_STRING to the dummy string '   12'. The function RIGHT then creates the result_string '12'.

LD

ST

```
result_string:=RIGHT(IN:=INT_TO_STRING(input_value), L:=2);
```

INT_TO_STRING_ LEADING_ZEROS

INTEGER into STRING

Description The function INT_TO_STRING_LEADING_ZEROS converts a value of the data type INT (positive values) to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

Example:

Data types

Data type	I/O	Function
INT	input	input data type
STRING	output	conversion result

Explanation

Function used	String1 defined as	Result
String1:=INT_TO_STRING(25)	STRING[1]	'5'
	STRING[2]	'25'
	STRING[3]	'025'
	STRING[4]	'0025'
	STRING[5]	'00025'
	STRING[6]	'000025'
	STRING[7]	'0000025'
	STRING[8]	'00000025'
	and so on...	

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_STRING_LEADING_ZEROS (see page 933)**

DINT_TO_STRING

DOUBLE INTEGER into STRING

Description The function DINT_TO_STRING converts a value of the data type DINT to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading spaces up to the maximum number of characters defined for the string.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Explanation

Function used	String1 defined as	Result
String1:=DINT_TO_STRING(-12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'␣-12345678'
	STRING[12]	'␣␣␣-12345678'
	and so on...	

PLC types: **Availability of DINT_TO_STRING (see page 924)**

Data types

Data type	I/O	Function
DINT	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

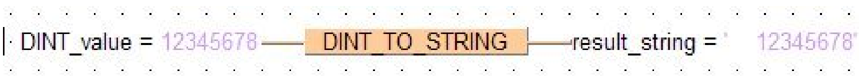
In the POU header, input and output variables are declared that are used in the function.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DINT	12345678	example value
1	VAR	result_string	STRING[11]	''	result: here ' 12345678'

The input variable **input_value** of the data type DINT is intialized by the value 12345678. The output variable **result_string** is of the data type STRING[11]. It can store a maximum of 11 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function’s input contact in the body.

Body The **input_value** of the data type DINT is converted into STRING[11]. The converted value is written to **result_string**. When the variable **input_value** = 12345678, **result_string** shows '␣␣␣12345678'.

LD



ST

```
result_string:=DINT_TO_STRING(input_value);
```

Example 2: This example illustrates how you create, from an input value of the data type DINT, a STRING[14] that contains a DINT number representation with commas after every three significant figures.

The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU Header

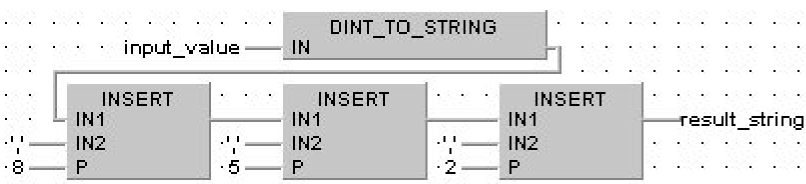
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DINT	1234567890	example value
1	VAR	result_string	STRING[14]	''	result: here '1,234,567,890'

In this example, both an input variable **input_value** of the data type DINT and an output variable **result_string** of the data type STRING[14] are declared.

Body In carrying out the operation in question, three standard functions INSERT are attached successively to the function DINT_TO_STRING. Each INSERT function inserts the attached character string at input IN2 into the character string at input IN1. The position at which the character string is to be introduced is determined by INT value at input P.

In the example all three INSERT functions insert the assigned STRING constant ',' after each three significant figures at input IN2. The correct position of each comma is determined by an INT constant at each respective P input. Out of the variable **input_value** = 1234567890, the **result_string** 1,234,567,890 results from the data type conversion and the three INSERT functions.

LD

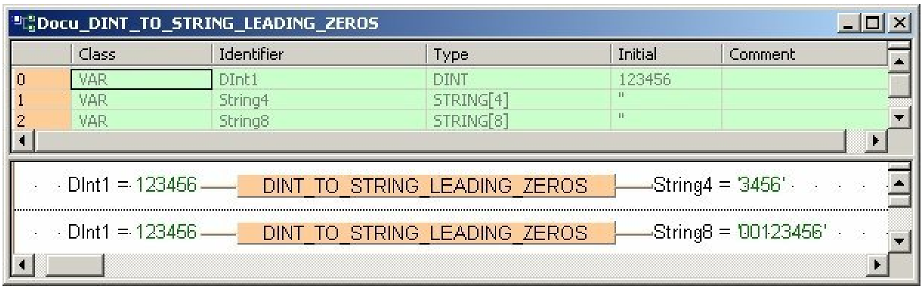


DINT_TO_STRING_ LEADING_ZEROS

DOUBLE INTEGER into STRING

Description This function converts a value of the data type DINT (positive value) to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

Example



Explanation

Function used	String1 defined as	Result
String1:=DINT_TO_STRING(12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'0012345678'
	STRING[12]	'000012345678'
	and so on...	

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_STRING_LEADING_ZEROS (see page 924)**

Data types

Data type	I/O	Function
DINT	input	input data type
STRING	output	conversion result

REAL_TO_STRING

REAL into STRING

Description The function REAL_TO_STRING converts a value from the data type REAL into a value of the data type STRING[15], which has 7 spaces both before and after the decimal point. The resulting string is right justified within the range '-999999.0000000' to '9999999.0000000'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of -12.0 the STRING '␣␣-12.0').

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **The function requires approximately 160 steps of program memory. For repeated use you should integrate it into a user function that is only stored once in the memory.**

PLC types: **Availability of REAL_TO_STRING (see page 934)**

Data types

Data type	I/O	Function
REAL	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, input and output variables are declared that are used in the function.

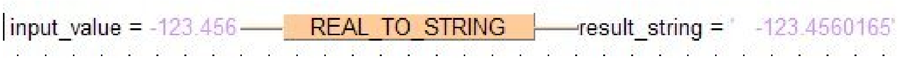
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	-123.4560166	example value
1	VAR	result_string	STRING[15]	''	result: here ' -123.4560166'

The input variable **input_value** of the data type REAL is intialized by the value -123.4560166. The output variable **result_string** is of the data type STRING[15]. It can store a maximum of 15 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body

The **input_value** of the data type REAL is converted into STRING[15]. The converted value is written to **result_string**. When the variable **input_value** = 123.4560166, **result_string** shows '   -123.4560165'.

LD



Example 2: This example illustrates how you create a STRING[7] with 4 positions before and 2 positions after the decimal point out of the data type REAL.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU Header

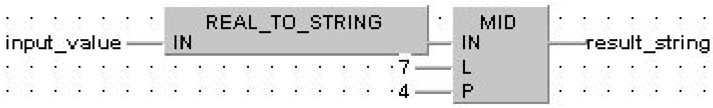
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	-123.4560166	example value
1	VAR	result_string	STRING[7]	''	result: here '-123.45'

In this example, both an input variable **input_value** of the data type REAL and an output variable **result_string** of the data type STRING[7] are declared.

Body In carrying out the operation in question, the standard function MID is attached to the function REAL_TO_STRING. MID creates a central sector in the character string from position P (INT value) with L (INT value) characters.

In the example, the INT constant 7 is entered at the L input of MID, which determines the length of the result string. The INT constant 4 at input P determines the position at which the central sector begins. Out of the variable **input_value** = -123.4560166, the STRING '   -123.4560166' results from the data type conversion. The MID function cuts off the STRING at position 4 and yields the **result_string** '-123.45'.

LD



TIME_TO_STRING

TIME into STRING

Description The function TIME_TO_STRING converts a value of the data type TIME to a value of the data type STRING[20]. In accordance with IEC-1131, the result string is displayed with a short time prefix and without underlines. Possible values for the result string's range are from 'T#000d00h00m00s000ms' to 'T#248d13h13m56s470ms'.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

☞ **When using the data type STRING with small PLCs like FP1 or FP-M, make sure that the length of the result string is equal to or greater than the length of the source string.**

PLC types: **Availability of TIME_TO_STRING (see page 935)**

Data types

Data type	I/O	Function
TIME	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, input and output variables are declared that are used in the function.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	TIME	T#1h30m45s	example value
1	VAR	result_string	STRING[20]	''	result: here 'T#000d01h30m45s000ms'

The input variable **input_value** of the data type TIME is intialized by the value T#1h30m45s. The output variable **result_string** is of the data type STRING[20]. It can store a maximum of 20 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body

The **input_value** of the data type TIME is converted into STRING[20]. The converted value is written to **result_string**. When the variable **input_value** = T#1h30m45s, **result_string** shows 'T#000d01h30m45s000ms'.

LD

ST
```
result_string:=TIME_TO_STRING(input_value);
```

IL
```
LD               input_value
TIME_TO_STRING
ST               result_string
```

Example 2: This example shows how, from an input value of the data type TIME, a TIME STRING[9] with the format 'xxhxxmxxs' is created (only hours, minutes and seconds are output).
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU Header

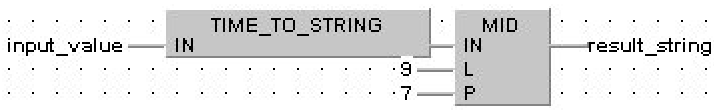
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	TIME	T#1h30m45s	example value
1	VAR	result_string	STRING[9]	''	result: here '01h30m45s'

In this example, both an input variable **input_value** of the data type TIME and an output variable **result_string** of the data type STRING[9] are declared.

Body In carrying out the operation in question, the standard function MID is attached to the function TIME_TO_STRING. MID creates a central sector in the character string from position P (INT value) with L (INT value) characters.

In the example, the INT constant 9 is entered at the L input of MID, which determines the length of the result string. The INT constant 7 at input P determines the position at which the central sector begins. Out of the variable **input_value** = T#1h30m45s, the STRING 'T#000d01h30m45s000ms' results from the data type conversion. The MID function cuts off the STRING at position 7 and yields the **result_string** '01h30m45s'.

LD

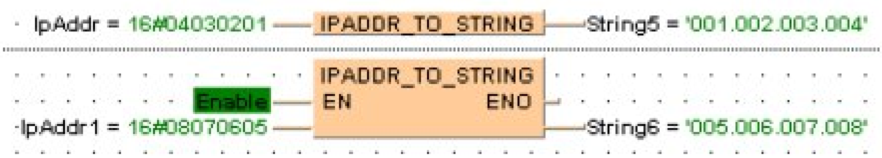


IPADDR_TO_STRING

IP Address to STRING

Description This function converts a binary IP address of the data type DWORD into a STRING in IP address format.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

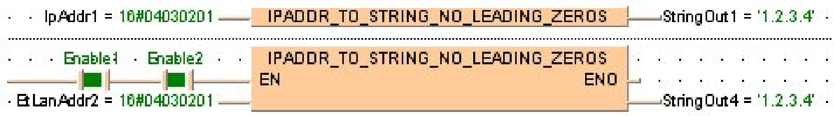
The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and lowest byte of the IP address the first octet. The format of the IP address corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.

IPADDR_TO_STRING_NO_LEADING_ZEROS

IP Address to STRING

Description This function converts a binary IP address of the data type DWORD into a STRING in IP address format.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.4'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

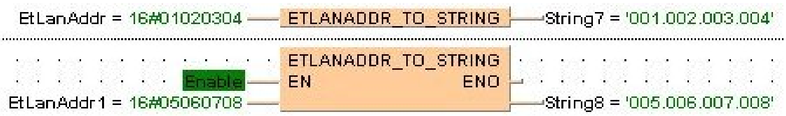
The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and lowest byte of the IP address the first octet. The format of the IP address corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.

ETLANADDR_TO_STRING ETLAN Address to STRING

Description This function converts a binary ETLAN address of the data type DWORD into a STRING in ETLAN address format.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

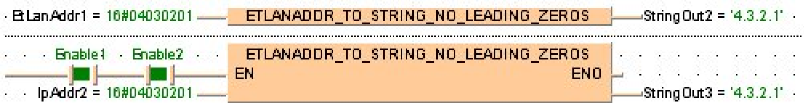
ETLANADDR_TO_STRING_ NO_LEADING_ZEROS

ETLAN Address to STRING

Description This function converts a binary ETLAN address of the data type DWORD into a STRING in ETLAN address format.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.4'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

WORD_TO_BOOL16 WORD to BOOL16

Description This function copies data of the data type WORD at the input to an array with 16 elements of the data type BOOL at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of WORD_TO_BOOL16 (see page 936)**

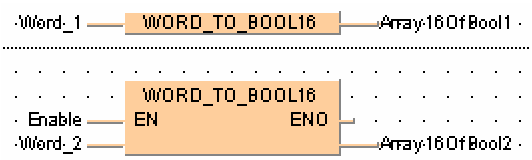
Data types

Data type	**Comment**
WORD	input variable
ARRAY of BOOL	ARRAY with 16 elements

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Word_1	WORD	0
2	VAR	Word_2	WORD	0
3	VAR	Array16OfBool1	ARRAY [0..15] OF BOOL	[16(FALSE)]
4	VAR	Array16OfBool2	ARRAY [0..15] OF BOOL	[16(FALSE)]

Body with and without EN/ENO:



INT_TO_BOOL16 INTEGER to BOOL16

Description This function copies data of the data type INT at the input to an array with 16 elements of the data type BOOL at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_BOOL16 (see page 932)**

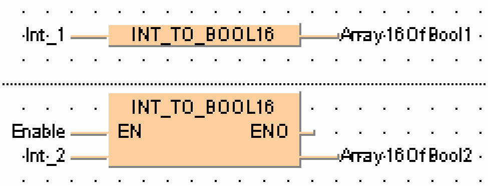
Data types

Data type	Comment
INT	input variable
ARRAY of BOOL	ARRAY with 16 elements

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Int_1	INT	0
2	VAR	Int_2	INT	0
3	VAR	Array16OfBool1	ARRAY [0..15 OF BOOL	[FALSE]
4	VAR	Array16OfBool2	ARRAY [0..15 OF BOOL	

Body with and without EN/ENO:



DWORD_TO_BOOL32 DOUBLE WORD to BOOL32

Description This function copies data of the data type DWORD at the input to an array with 32 elements of the data type BOOL at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DWORD_TO_BOOL32 (see page 924)**

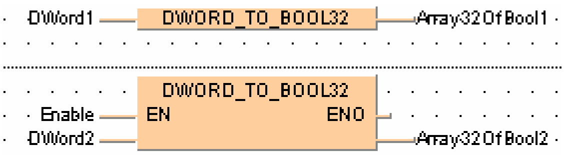
Data types

Data type	Comment
DWORD	input variable
ARRAY of BOOL	ARRAY with 32 elements

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31 OF BOOL	[FALSE]
2	VAR	Array32OfBool2	ARRAY [0..31 OF BOOL	
3	VAR	DWord1	DWORD	0
4	VAR	DWord2	DWORD	0

Body with and without EN/ENO:



DINT_TO_BOOL32 DOUBLE INTEGER to BOOL32

Description This function copies data of the data type DINT at the input to an array with 32 elements of the data type BOOL at the output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_BOOL32 (see page 924)**

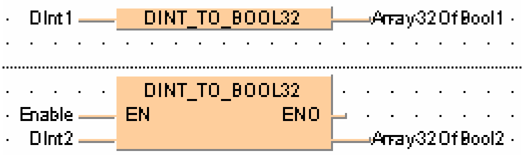
Data types

Data type	Comment
DINT	input variable
ARRAY of BOOL	ARRAY with 32 elements

POU header:

	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31 OF BOOL	[FALSE]
2	VAR	Array32OfBool2	ARRAY [0..31 OF BOOL	
3	VAR	DInt1	DINT	0
4	VAR	DInt2	DINT	0

Body with and without EN/ENO:



WORD_TO_BOOLS

WORD to 16 variables of the data type BOOL

Description This function converts a value of the data type WORD bit-wise to 16 values of the data type BOOL.

The outputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types: **Availability of WORD_TO_BOOLS (see page 936)**

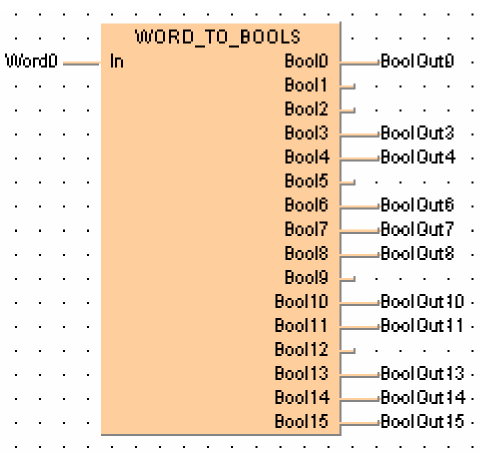
Data types

Variable	Data type	Function
In	WORD	input variable
BOOL0 ... BOOL15	BOOL	16 output variables of the data type BOOL

POU header:

	Class	Identifier	Type	Initial
0	VAR	Word0	WORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE
11	VAR	Bool11	BOOL	FALSE
12	VAR	Bool12	BOOL	FALSE
13	VAR	Bool13	BOOL	FALSE
14	VAR	Bool14	BOOL	FALSE
15	VAR	Bool15	BOOL	FALSE

Body:



DWORD_TO_BOOLS

DOUBLE WORD to 32 variables of the data type BOOL

Description This function converts a values of the data type DWORD bit-wise to 32 values of the data type BOOL.

The outputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types: **Availability of DWORD_TO_BOOLS (see page 924)**

Data types

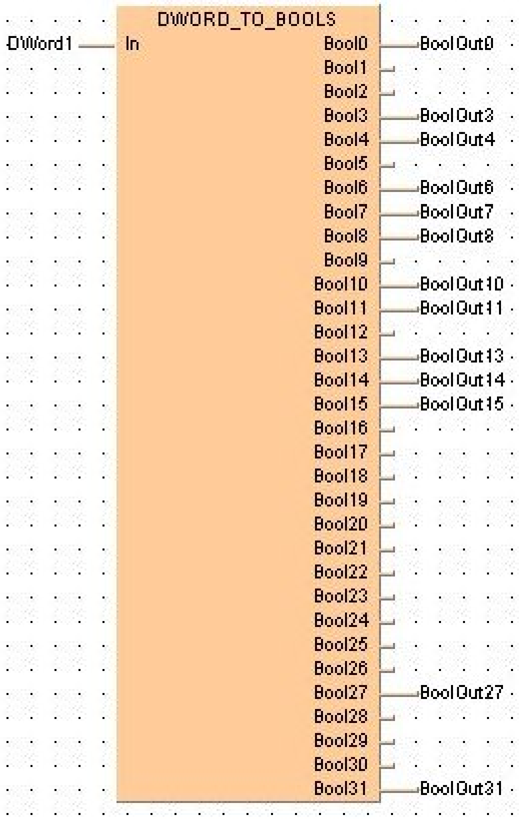
Variable	Data type	Function
In	DWORD	input variable
BOOL0 ... BOOL31	BOOL	32 output variables of the data type BOOL

POU header:

	Class	Identifier	Type	Initial
0	VAR	Dword1	DWORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE

etc. to Bool31

Body:


DWORD_TO_BOOLS
DWord1
In
Bool0
BoolOut0
Bool1
Bool2
Bool3
BoolOut3
Bool4
BoolOut4
Bool5
Bool6
BoolOut6
Bool7
BoolOut7
Bool8
BoolOut8
Bool9
Bool10
BoolOut10
Bool11
BoolOut11
Bool12
Bool13
BoolOut13
Bool14
BoolOut14
Bool15
BoolOut15
Bool16
Bool17
Bool18
Bool19
Bool20
Bool21
Bool22
Bool23
Bool24
Bool25
Bool26
Bool27
BoolOut27
Bool28
Bool29
Bool30
Bool31
BoolOut31

INT_TO_BOOLS

INTEGER to 16 variables of the data type BOOL

Description This function converts a value of the data type INT bit-wise to 16 values of the data type BOOL.

The outputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types: **Availability of INT_TO_BOOLS (see page 932)**

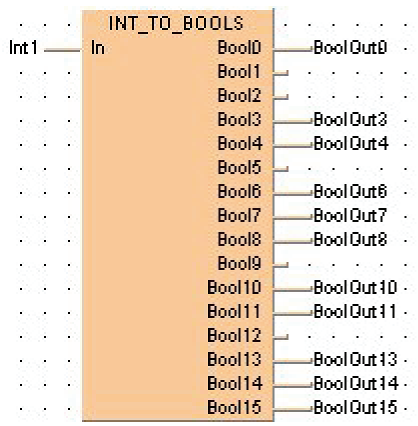
Data types

Variable	Data type	Function
In	INT	input variable
BOOL0 ... BOOL15	BOOL	16 output variables of the data type BOOL

POU header:

	Class	Identifier	Type	Initial
0	VAR	Int1	INT	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE
11	VAR	Bool11	BOOL	FALSE
12	VAR	Bool12	BOOL	FALSE
13	VAR	Bool13	BOOL	FALSE
14	VAR	Bool14	BOOL	FALSE
15	VAR	Bool15	BOOL	FALSE

Body:



DINT_TO_BOOLS

DOUBLE INTEGER to 32 variables of the data type BOOL

Description This function converts a value of the data type INT bit-wise to 32 values of the data type BOOL.

The outputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types: **Availability of DINT_TO_BOOLS** **(see page 924)**

Data types

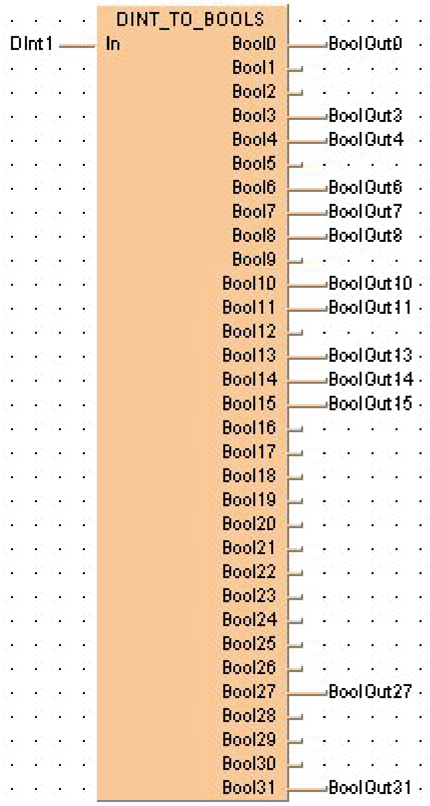
Variable	Data type	Function
In	DINT	input variable
BOOL0 ... BOOL31	BOOL	32 output variables of the data type BOOL

POU header:

	Class	Identifier	Type	Initial
0	VAR	Dint1	DINT	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE

etc. to Bool31

Body:


DINT_TO_BOOLS
DInt1
In
Bool0
BoolOut0
Bool1
Bool2
Bool3
BoolOut3
Bool4
BoolOut4
Bool5
Bool6
BoolOut6
Bool7
BoolOut7
Bool8
BoolOut8
Bool9
Bool10
BoolOut10
Bool11
BoolOut11
Bool12
Bool13
BoolOut13
Bool14
BoolOut14
Bool15
BoolOut15
Bool16
Bool17
Bool18
Bool19
Bool20
Bool21
Bool22
Bool23
Bool24
Bool25
Bool26
Bool27
BoolOut27
Bool28
Bool29
Bool30
Bool31
BoolOut31

INT_TO_BCD

INTEGER into BCD

Description INT_TO_BCD converts a binary value of the data type INTEGER into a BCD value (binary coded decimal integer) of the type WORD in order to be able to output BCD values in word format.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INT_TO_BCD** **(see page 932)**

Since the output variable is of the type WORD and 16 bits wide, the value of the input variable should have a maximum of 4 decimal places and should thus be located between 0 and 9999.

Data types

Data type	I/O	Function
INT	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

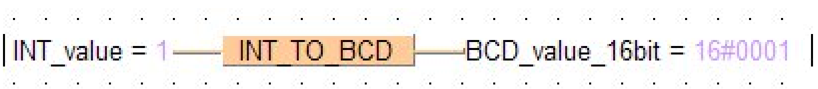
	Class	Identifier	Type	Initial	Comment
0	VAR	BCD_value_16bit	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a BCD value of the data type WORD. The converted value is written into **BCD_value_16bit**.

LD



ST

```
BCD_value_16bit:=INT_TO_BCD(INT_value);
```

DINT_TO_BCD

DOUBLE INTEGER into BCD

Description DINT_TO_BCD converts a value of the data type DINT into a BCD value of the data type DWORD.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DINT_TO_BCD (see page 924)**

The value for the input variable should be between 0 and 999 999 99.

Data types

Data type	I/O	Function
DINT	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	BCD_value_32bit	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a BCD value of the data type DOUBLE WORD. The converted value is written into **BCD_value_32bit**.

LD

ST

```
BCD_value_32bit:=DINT_TO_BCD(DINT_value);
```

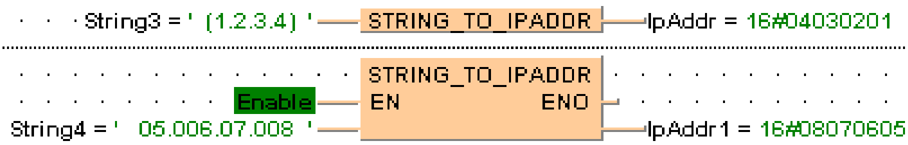
STRING_TO_IPADDR STRING to IP Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD via a sub-programm of approx. 330 steps that is also used in the functions STRING_TO_IPADDR and STRING_TO_ETLANADDR.

See also: STRING_TO_IPADDR_STEPSAVER (see page 189)

Example:



Permissible format:

´[Space]Octet1.Octet2.Octet3.Octet4[Space]´, e.g.: ´ [192.168.206.4] ´

Permissible characters:

Space	All characters except for decimal numbers
Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

PLC types: **Availability of STRING_TO_IPADDR (see page 935)**

- **The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.**
- **If the format is wrong the result is 0.**
- **The conversion is such that the first octet represents the lowest byte of the IP address and the fourth octet the highest byte of the ET-LAN address. The format corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.**

Data types

Data type	Comment
STRING	input variable
DWORD	output variable

STRING_TO_IPADDR _STEPSAVER

STRING (IP-Address Format 00a.0bb.0cc.ddd) to DWORD

Description This function converts a STRING in IP address format into a value of the data type DWORD.

The function uses for approx. 50 steps of generated code the basic instruction F76_A2BIN. The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example:

Permissible format:

´Octet1.Octet2.Octet3.Octet4[Space]´, e.g.: ´ [192.168.206.4] ´

Permissible characters:

Octets 1-4	Decimal numbers "0"-"9", maximal 3 positions, with or without leading zeros in the range 0-255

PLC types: **Availability of STRING_TO_IPADDR_STEPSAVER (see page 935)**

- **If the format is wrong the result is 0.**
- **The conversion is such that the first octet represents the lowest byte of the IP address and the fourth octet the highest byte of the ET-LAN address. The format corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.**

Data types

Data type	Comment
STRING	input variable
DWORD	output variable

STRING_TO_ETLAN ADDR

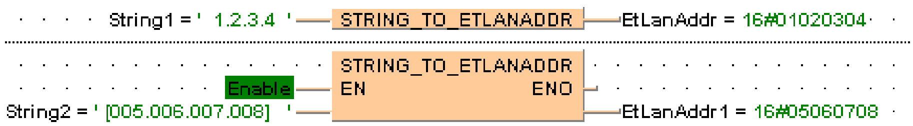
STRING to ETLAN Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD via a sub-programm of approx. 330 steps that is also used in the functions STRING_TO_IPADDR and STRING_TO_ETLANADDR.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example:



Permissible format:

´[Space]Octet1.Octet2.Octet3.Octet4[Space]´, e.g.: ´ [192.168.206.4] ´

Permissible characters:

Space	All characters except for decimal numbers
Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

☞

- **The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.**
- **If the format is wrong the result is 0.**
- **The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.**

STRING_TO_ETLAN ADDR_STEPSAVER

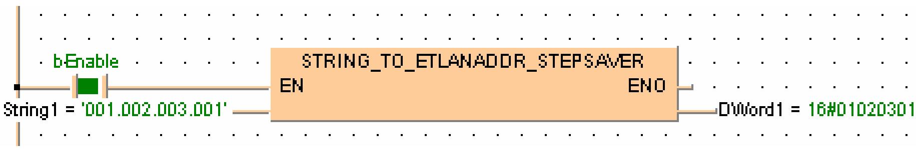
STRING (IP-address format 00a.0bb.0cc.ddd) to ETLAN Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

The function uses for approx. 50 steps of generated code the basic instruction F76_A2BIN. The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example:



Permissible format:

´Octet1.Octet2.Octet3.Octet4[Space]´, e.g.: ´ [192.168.206.4] ´

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

- **If the format is wrong the result is 0.**
- **The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.**

Chapter 8

Selection Instructions

MAX

Maximum value

Description MAX determines the input variable with the highest value.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MAX (see page 933)**

☞ **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever input variable's value is greater

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

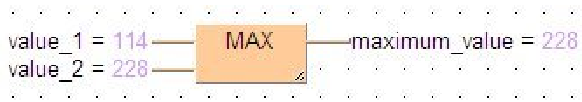
	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	maximum_value	INT	0	all types allowed

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

Value_1 and **value_2** are compared with each other. The maximum value of all input variables is written in **maximum_value**.

LD



ST

```
maximum_value:=MAX(value_1, value_2);
```

MIN

Minimum value

Description MIN detects the input variable with the lowest value.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MIN (see page 934)**

The number of input contacts lies in the range of 2 to 28.

Data types

Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever input variable's value is smallest

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

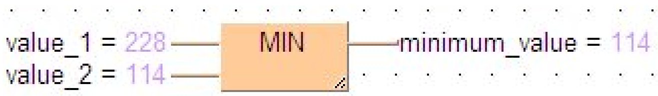
	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	minimum_value	INT	0	all types allowed

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

Value_1 and **value_2** are compared with each other. The lower value of the two is written into **minimum_value**.

LD



ST

```
minimum_value:=MIN(value_1, value_2);
```

LIMIT

Limit value for input variable

Description In LIMIT the 1st input variable forms the lower and the 3rd input variable the upper limit value. If the 2nd input variable is within this limit, it will be transferred to the output variable. If it is above this limit, the upper limit value will be transferred; if it is below this limit the lower limit value will be transferred.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of LIMIT (see page 933)**

Data types

Data type	I/O	Function
all data types	1st input	upper limit
all data types	2nd input	value compared to upper and lower limit
all data types	3rd input	lower limit
all data types	output as input	result, 2nd input value if between upper and lower limit, otherwise the upper or lower limit

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

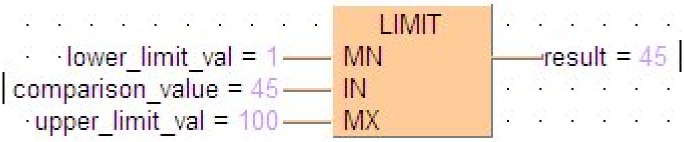
	Class	Identifier	Type	Initial	Comment
0	VAR	lower_limit_val	INT	0	all types allowed
1	VAR	comparison_value	INT	0	all types allowed
2	VAR	upper_limit_val	INT	0	all types allowed
3	VAR	result	INT	0	all types allowed

In this example the input variables (**lower_limit_val, comparison_value** and **upper_val**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

Lower_limit_val and **upper_limit_val** form the range where the **comparison_value** has to be, if it has to be transferred to **result**. If the **comparison_value** is above the **upper_limit_val**, the value of **upper_limit_val** will be transferred to **result**. If it is below the **lower_limit_val**, the value of **lower_limit_val** will be transferred to **result**.

LD



ST

```
result:=LIMIT(MN:=lower_limit_val, IN:=comparison_value,
MX:=upper_limit_val);
```

MUX

Select value from multiple channels

Description The function Multiplexer selects an input variable and writes its value into the output variable. The 1st input variable determines which input variable (IN1or IN2 ...) is to be written into the output variable. The function MUX can be configured for any desired number of inputs.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MUX (see page 934)**

Data types

Data type	I/O	Function
INT	1st input	selects channel for 2nd or 3rd input value to be written to
all data types	2nd input	value 1
all data types	3rd input	value 2
all data types	output as 2nd and 3rd input	result

The 2nd and 3rd input variables must be of the same data type.

- **The difference between the functions MUX and SEL (see page 200) is that in MUX with an integer value you can select between plural channels, and in SEL with a Boolean value only between two channels.**
- **The number of input contacts lies in the range of 2 to 28.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

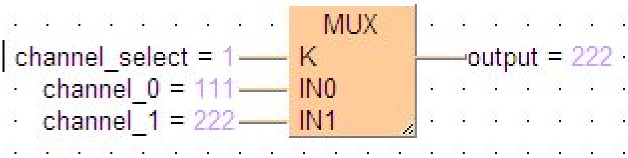
POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	channel_select	INT	0	value '0' to 'n'
1	VAR	channel_0	INT	0	all types allowed
2	VAR	channel_1	INT	0	all types allowed
3	VAR	output	INT	0	all types allowed

In this example the input variables (**channel_select, channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body In **channel_select** you find the integer value (0, 1...n) for the selection of **channel_0** or **channel_1**. The result will be written into **output**.

LD



ST
```
output:=MUX( K:= channel_select , IN0:= channel_0 ,
          IN1:= channel_1 );
```

SEL

Select value from one of two channels

Description With the first input variable (data type BOOL) of SEL you define which input variable is to be written into the output variable. If the Boolean value = 0 (FALSE), the second input variable will be written into the output variable, otherwise the third.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of SEL (see page 934)**

Data types

Data type	I/O	Function
BOOL	1st input	selects channel for 2nd or 3rd input value to be written to
all data types	2nd input	value 1
all data types	3rd input	value 2
all data types	output as 2nd and 3rd input	result

☞ **The difference between the functions SEL and MUX (see page 198) is that in case of SEL a Boolean value serves for the channel selection, and in case of MUX an integral number (INT). Therefore, you can choose between more than two channels with MUX.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

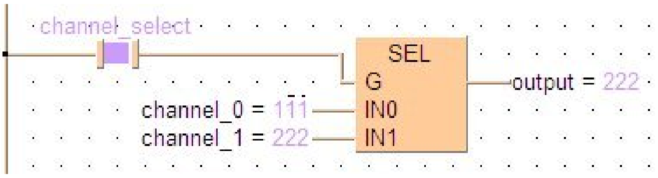
	Class	Identifier	Type	Initial
0	VAR	channel_select	BOOL	FALSE
1	VAR	channel_0	INT	0
2	VAR	channel_1	INT	0
3	VAR	output	INT	0

In this example the input variables (**channel_select, channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body

If **channel_select** has the value 0, **channel_0** will be written into **output**, otherwise **channel_1**.

LD


channel_select
SEL
G
IN0
IN1
output = 222
channel_0 = 111
channel_1 = 222

Chapter 9

String Instructions

LEN — String Length

Description LEN calculates the length of the input string and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of LEN (see page 933)**

☞

- **If the string is longer than the length defined for the input variable (**input_string**) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
STRING	input	input data type
INT	output	length of string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

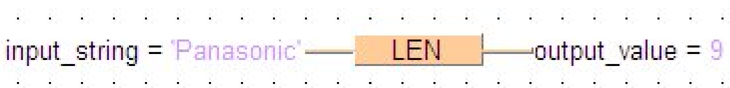
	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[12]	'Panasonic'	sample string
1	VAR	output_value	INT	0	result: here 9

In this example the input variable (**input_string)** has been declared. Instead, you may enter the string (**'Panasonic'**) directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body

The length (9) of **input_string** (**'Panasonic'**) is written into **output_value**.

LD



```
ST   output_value:=LEN(input_value);
```

LEFT

Copy characters from the left

Steps: 8

Description LEFT copies, starting from the left, **n** characters of the string of the first input variable to the output variable. You define the number of characters to be delivered **n** by the second input variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of LEFT (see page 933)**

- **If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable (**output_string**).**
- **If the output string is longer than the length defined for the output variable in the field "Type", only as many characters are copied from the left as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied, from the left
STRING	output	copied string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

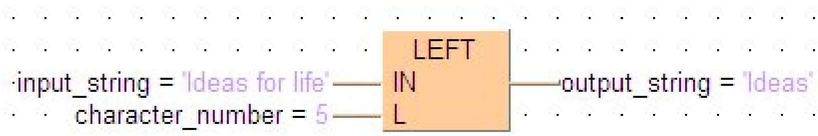
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for l...	sample string
1	VAR	output_string	STRING[5]	''	result: here 'Ideas'
2	VAR	character_number	INT	5	characters to be delivered

In this example the input variables (**input_string** and **character_number**) have been declared. Instead, you may enter the string (**'Ideas for life'**) and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body Starting from the left, **character_number** (5) of **input_string** (**'Ideas for life'**) is copied to **output_string** (**'Ideas'**).

LD



ST

```
output_string:=LEFT(IN:=input_string, L:=character_number);
```

RIGHT

Copy characters from the right

Steps: 8

Description RIGHT copies, starting from the right, n characters of the string of the first input variable to the output variable. You define the number of characters to be delivered n by the second input variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of RIGHT** **(see page 934)**

- **If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable (**output_string**).**
- **If the output string is longer than the length defined for the output variable in the field "Type", only as many characters are copied from the left as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied, from the right
STRING	output	copied string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

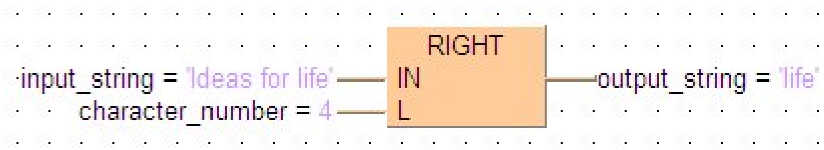
	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for life'	sample string
1	VAR	character_number	INT	4	characters to be delivered
2	VAR	output_string	STRING[4]	''	result here: 'life'

In this example the input variables (**input_string** and **character_number**) have been declared. Instead, you may enter the string (**'Ideas for life'**) and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body

Starting from the right, **character_number** (4) of **input_string** (**‘Ideas for life’**) is copied to **output_string** (**‘life’**).

LD



MID

Copy characters from a middle position **Steps: 10**

Description MID copies, starting from the position **P**, **L** characters of the string of the first input variable to the output variable. You define the number of characters to be delivered **L** by the second and the start position **P** by the third input variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MID (see page 933)**

☞

- **The sum of start position and number of characters to be delivered should not be greater than the input string. If you want to receive for example 5 characters of a 10-character string, starting from position 7, only the last 4 characters are delivered.**
- **If the output string is longer than the length defined for the output variable (**output_string**) in the field "Type", only as many characters are copied from the start position as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied
INT	3rd input	position where copying begins
STRING	output	copied string

Error Flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type" or start position is greater than the input string
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

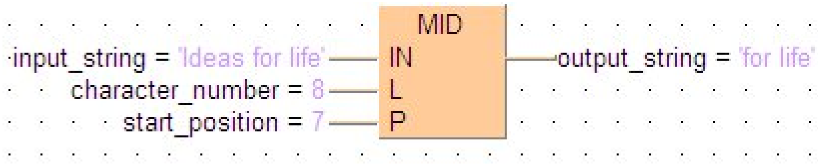
	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[11]	'NAiSControl'	sample string
1	VAR	character_number	INT	5	characters to delivered
2	VAR	start_position	INT	4	postition to start copying
3	VAR	output_string	STRING[5]	''	reslut: here 'SCont'

In this example the input variables (**input_string**, **character_number** and **start_position**) have been declared. Instead, you may enter the string (**'Ideas for life'**), the number of characters to be delivered and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body

Starting from **start_position** (7), **character_number** (8) of **input_string** (**'Ideas for life'**) is copied to **output_string** (**'for life'**).

LD



ST

```
output_string:=MID(IN:=input_string, L:=character_number,
P:=start_position);
```

CONCAT

Concatenate (attach) a string

Description CONCAT concatenates (attaches) the second and the following input strings (IN1 + IN2 + ...) to the first input string and writes the resulting string into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of CONCAT (see page 923)**

- **If the output string is longer than the length defined for the output variable (output_string) in the field "Type", only as many characters are copied, starting from the left, as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help.**

Data types

Data type	I/O	Function
STRING	1st input	beginning input string
STRING	2nd input	string that will be attached to the beginning string
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

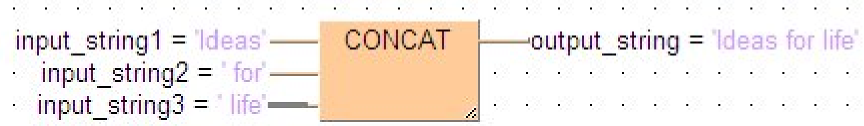
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_string1	STRING[32]	'Ideas'	sample string
1	VAR	input_string2	STRING[32]	' for'	sample string
2	VAR	input_string3	STRING[32]	' life'	sample string
3	VAR	output_string	STRING[32]	''	reslut: here 'Ideas for life'

In this example the input variables (**input_string1, input_string2** and **input_string3**) have been declared. However, you may enter the strings (**'Ideas'**, **' for' and ' life'**) directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

Body **Input_string3** (**‘ life’**) is attached to **input_string2** (**‘ for’**) and this string is attached to **input_string1** (**'Ideas'**). The resulting string (**‘Ideas for life’**) is written into **output_string**.

LD



ST

```
output_string:=CONCAT(input_string1, input_string2,
input_string3);
```

DELETE

Delete characters from a string

Steps: 19

Description DELETE deletes, starting from position **P**, **L** characters from the string of the first input variable. The resulting string is written into the output variable. You define the number of characters to be deleted **L** by the second and the start position **P** by the third input variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DELETE (see page 924)**

☞

- **If the output string is longer than the length defined for the output variable (**output_string**) in the field "Type", only as many characters are copied, starting from the left, as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are deleted
INT	3rd input	position where deletion begins
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

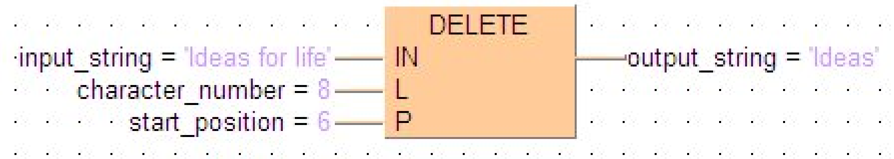
	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for life'	sample string
1	VAR	character_number	INT	8	characters to be deleted
2	VAR	start_position	INT	6	position to start deleting
3	VAR	output_string	STRING[5]	''	result here 'Ideas'

In this example the input variables (**input_string**, **character_number** and **start_position**) have been declared. Instead, you may enter the string (**'Ideas for life'**), the number of characters to be deleted and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body

Starting from **start_position** (**6**), **character_number** (**8**) is deleted from **input_string** (**'Ideas for life'**). The resulting string (**'Ideas')** is written into **output_string**.

LD



ST

```
output_string:=DELETE(input_string, character_number,
start_position);
```

FIND

Find string's position

Description FIND returns the position at which the second input string first occurs in the first input string. The result is written into the output variable. If the second input string does not occur in the first input string, the value ZERO is returned.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of FIND (see page 932)**

- **If the strings are longer than the length defined for the input variables (**input_string_1 **and** input_string_2**) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	string that is searched for in the input string
INT	output	position at which the string searched for is found

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

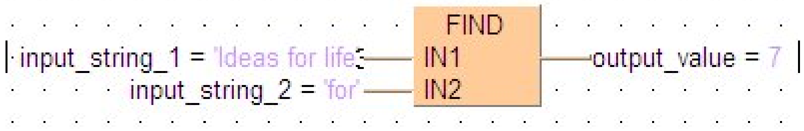
POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_string_1	STRING[15]	'Ideas for l...	sample string
1	VAR	input_string_2	STRING[3]	'for'	searched string
2	VAR	output_value	INT	0	1st position found

In this example the input variables (**input_string_1** and **input_string_2**) have been declared. Instead, you may enter the strings (**'Ideas for life'** and **'for'**) directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

Body **Input_string_2** (**'for'**) is searched in **input_string_1** (**'Ideas for life'**). The position of the first occurrence (**7**) is written into **output_value**.

LD



ST
```
output_value:= FIND(input_string_1, input_string_2);
```

INSERT

Insert characters

Steps: 19

Description INSERT inserts the STRING specified at **IN2** into the STRING specified at **IN1** beginning after the character position **P**. The result is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of INSERT** **(see page 932)**

- **If the strings are longer than the length defined for the input variables (**input_string_1 **and** input_string_2**) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	string to be inserted into input string
INT	3rd input	position at which string is inserted
STRING	output	result string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

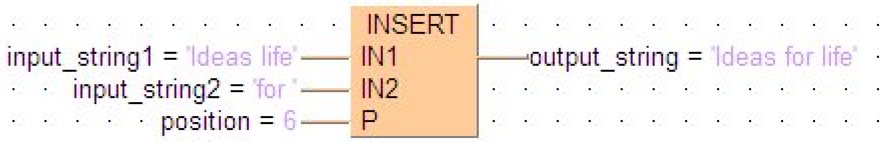
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_string1	STRING[32]	'Ideas life'	sample string
1	VAR	input_string2	STRING[32]	'for '	sample string
2	VAR	position	INT	6	
3	VAR	output_string	STRING[32]	''	reslut: here 'Ideas for life'

Body In this example the input variables **input_string1**, **input_string2** and **position** have been declared. However, you may enter the values directly at the function's input contact pins instead. The STRING values have to be put in inverted commas, both in the POU header and at the contact pins. **input_string2** ('for ') is inserted into **input_string1** ('Ideas life') after character position 6. The result ('Ideas for life') is returned at **output_value**. In the LD example,  (Monitoring) icon was activated while in online mode, hence you can see the results immediately.

LD



ST

```
output_value:=INSERT(IN1:=input_string1, IN2:=input_string2,
P:=6);
```

REPLACE

Replaces characters **Steps: 26**

Description The STRING specified at **IN2** replaces characters in the STRING specified at **IN1**. The number of characters, i.e. the length (L), to be replaced is specified at **L**. The position at which the replacement starts is specified at **P**. The result is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of REPLACE (see page 934)**

- **If the strings are longer than the length defined for the input variables (**input_string_1 **and** input_string_2**) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also table of steps in the online help. (up to 200 steps)**

Data types

Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	replacement string
INT	3rd input	the number of characters in the input string to be replaced
INT	4th input	position at which characters begin to be replaced
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

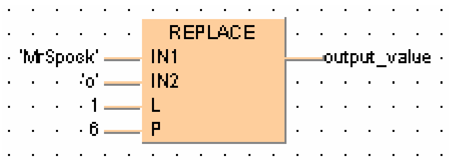
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	output_value	STRING[32]	''	result: 'MrSpook'

Body In this example constant values are entered directly at the function's input contact pins. However, you may declare variables in the POU header. The STRING values have to be put in inverted commas, either in the POU header or at the contact pins. Here the 'c' in the STRING 'MrSpock' has been replaced with an 'o', yielding 'MrSpook'.

LD



Chapter 10

Date and Time Instructions

ADD_TIME

Add TIME

Description ADD_TIME adds the times of the two input variables and writes the sum in the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of ADD_TIME (see page 923)**

Data types

Data type	I/O	Function
TIME	1st input	augend
TIME	2nd input	addend
TIME	output	sum

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	time_value_2	TIME	T#0s	
2	VAR	time_value_3	TIME	T#0s	

In this example the input variables (**time_value_1** and **time_value_2**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body

Time_value_1 and **time_value_2** are added. The result is written into **time_value_3**.

LD

ST

```
time_value_3:=ADD_TIME(time_value_1, time_value_2);
```

SUB_TIME

Subtract TIME

Description SUB_TIME subtracts the value of the second input variable from the value of the first input variable and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of SUB_TIME (see page 935)**

Data types

Data type	I/O	Function
TIME	1st input	minuend
TIME	2nd input	subtrahend
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

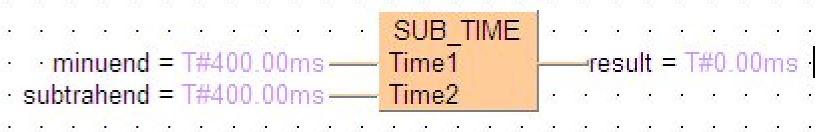
	Class	Identifier	Type	Initial	Comment
0	VAR	minuend	TIME	T#0s	
1	VAR	subtrahend	TIME	T#0s	
2	VAR	result	TIME	T#0s	

In this example the input variables (**minuend** and **subtrahend**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body

Subtrahend is subtracted from **minuend**. The result will be written into **result.**

LD



ST

```
result:= SUB_TIME(minuend, subtrahend);
```

MUL_TIME_INT

Multiply TIME by INTEGER

Description MUL_TIME_INT multiplies the values of the two input variables with each other and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MUL_TIME_INT (see page 934)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
INT	2nd input	multiplicator
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

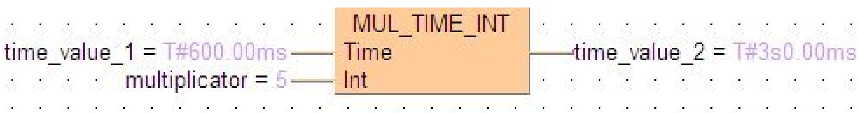
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	multiplier	INT	0	
2	VAR	time_value_2	TIME	T#0s	

In this example the input variables (**time_value_1** and **multiplier**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body

Time_value_1 is multiplied with **multipli**er. The result is written into **time_value_2**.

LD



ST

```
time_value_2:=MUL_TIME_INT(time_value_1, multiplier);
```

MUL_TIME_DINT

Multiply TIME by DOUBLE INTEGER

Description MUL_TIME_DINT multiplies the values of the input variables and writes the result to the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MUL_TIME_DINT (see page 934)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
DINT	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#1s 500ms	
1	VAR	multiplier	DINT	5	
2	VAR	time_value_2	TIME	T#0s	result: T# 7s 500ms

In this example, the input variables **time_value** and **multiplier** have been declared. However, you can write a constant directly at the input contact of the function instead.

Body

time_value_1 is multiplied by **multiplier**. The result is written in **time_value**_2.

LD

ST

```
time_value_2:=MUL_TIME_DINT(time_value_1, multiplier);
```

MUL_TIME_REAL

Multiply TIME by REAL

Description MUL_TIME_REAL multiplies the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of MUL_TIME_REAL (see page 934)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
REAL	2nd input	multiplicator
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

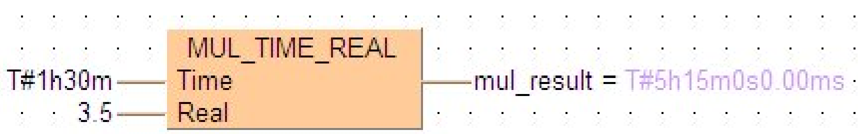
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	mul_result	TIME	T#0s

Body

The constant **T#1h30m** is multiplied by the value **3.5**. The result is written in **mul_result**. By clicking on the  (Monitoring) icon while in the online mode, you can see the result **T#5h15m0s0.00ms** immediately.

LD



ST

```
mul_result:=MUL_TIME_REAL(T#1h30m, 3.5);
```

DIV_TIME_INT

Divide TIME by INTEGER

Description DIV_TIME_INT divides the value of the first input variable by the value of the second input variable and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DIV_TIME_INT (see page 924)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
INT	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	time_value_2	TIME	T#0s	
2	VAR	INT_value	INT	0	

In this example the input variables (**time_value_1** and **INT_value**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body

time_value_1 is divided by **INT_value**. The result is written into **time_value_2**.

LD

ST

```
time_value_2:=DIV_TIME_INT(time_value_1, INT_value);
```

DIV_TIME_DINT

Divide TIME by DOUBLE INTEGER

Description DIV_TIME_DINT divides the value of the first input variable by the value of the second and writes the result into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DIV_TIME_DINT (see page 924)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
DINT	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

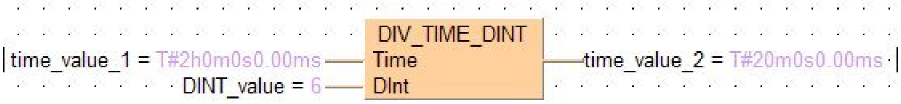
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#2h	
1	VAR	time_value_2	TIME	T#0s	result: T#20m
2	VAR	DINT_value	DINT	6	

In this example, the input variables (**time_value_1**, **DINT_value**) have been declared. However, you can write a constant directly at the input contact of the function instead.

Body **time_value_1** is divided by **DINT_value**. The result is written in **time_value_2**.

LD



ST

```
time_value_2:=DIV_TIME_DINT(time_value_1, INT_value);
```

DIV_TIME_REAL

Divide TIME by REAL

Description DIV_TIME_REAL divides the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

PLC types: **Availability of DIV_TIME_REAL (see page 924)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
REAL	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

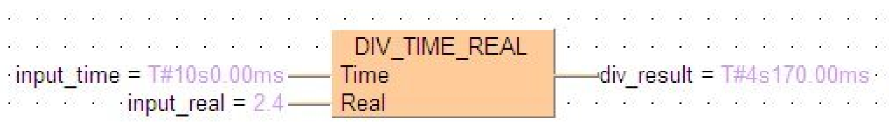
All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial
0	VAR	input_time	TIME	T#10s
1	VAR	input_real	REAL	2.4
2	VAR	div_result	TIME	T#0s

Body

The value of variable **input_time** is divided by the value of the variable **input_real**. The result is written in **div_result**. In this example the input variables have been declared in the POU header. However, you may enter constants directly at the contact pins of the function.

LD



ST

```
div_result:=DIV_TIME_REAL(input_time, input_real);
```

Chapter 11

Bistable Instructions

SR Set/reset

Description The function block SR (set/reset) allows you to both set and reset an output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

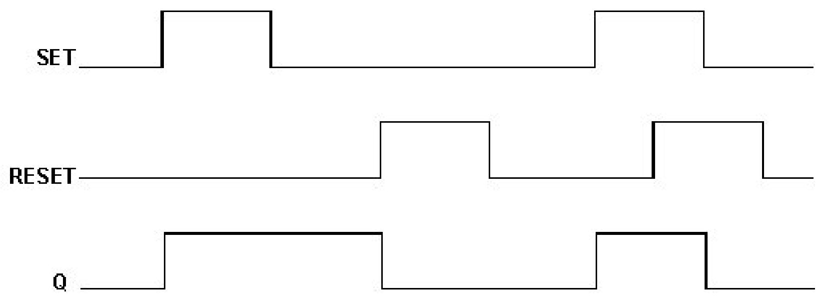
For SR declare the following:

SET (S1)	set The output Q is set for each rising edge at SET
RESET (R)	reset The output Q is reset for each rising edge detected at RESET, except when SET is set (see time chart)
Q (Q1)	signal output is set if a rising edge is detected at SET; is reset if a rising edge is detected at RESET if SET is not set.

☞

- **The names in brackets are the valid parameter names of the ST-editor.**
- **Q is set if a rising edge is detected at both inputs (Set and Reset).**
- **Upon initialising, Q always has the status zero (reset).**

Time Chart:



PLC types: **Availability of SR (see page 935)**

Data types

Data types	I/O	Function
BOOL	1st input	set
BOOL	2nd input	reset
BOOL	output	set or reset depending on inputs

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

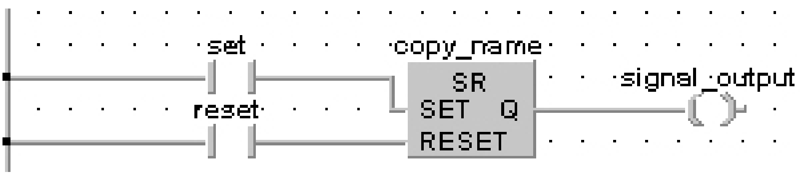
All input and output variables which are used for programming the function block SR are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	SR		under this identifier a copy of the SR function block is saved and a seperate data area is reserved
1	VAR	set	BOOL	FALSE	set input
2	VAR	reset	BOOL	FALSE	reset input
3	VAR	signal_output	BOOL	FALSE	

Body

If **set** is set (status = TRUE), **signal_output** will be set. If only **reset** is set, the **signal_output** will be reset (status = FALSE). If both **set** and **reset** are set, **signal_output** will be set.

LD



ST

```
copy_name( SET:= set, RESET:= reset);
        signal_output:= signal_output;
```

RS Reset/set

Description The function block RS (reset/set) allows you to both reset and set an output.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

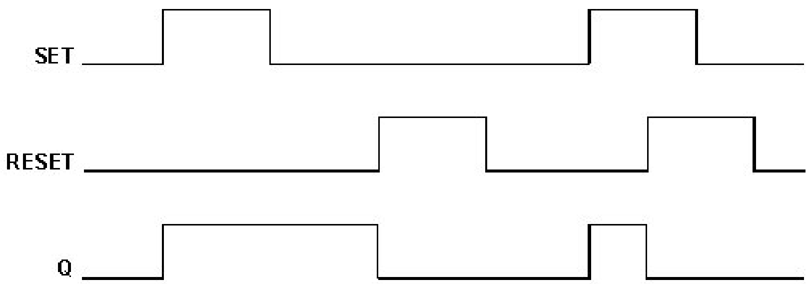
For RS declare the following:

SET (S1)	set The output Q is set for each rising edge at SET if RESET is not set.
RESET (R)	reset The output Q is reset for each rising edge at RESET.
Q (Q1)	signal output is set if a rising edge is detected at SET and if RESET is not set; is reset if a rising edge is detected at RESET.

☞

- **The names in brackets are the valid parameter names of the ST-editor.**
- **Q is reset if a rising edge is detected at both inputs.**

Time Chart:



PLC types: **Availability of RS** **(see page 934)**

Data types

Data types	I/O	Function
BOOL	1st input	set
BOOL	2nd input	reset
BOOL	output	set or reset depending on inputs

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

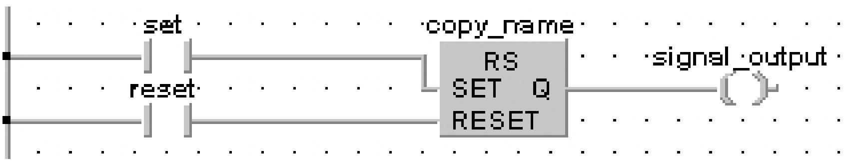
All input and output variables which are used for programming the function block RS are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	RS		under this identifier a copy of the RS function block is saved and a seperate data area is reserved
1	VAR	set	BOOL	FALSE	set input
2	VAR	reset	BOOL	FALSE	reset input
3	VAR	signal_output	BOOL	FALSE	

Body

If **set** is set (status = TRUE) the **signal_output** will be set. If only **reset** is set, the **signal_output** will be reset (status = FALSE). If both **set** and **reset** are set, the **signal_output** will be reset to FALSE.

LD



ST

```
copy_name( SET:= set, RESET:= reset);
        signal_output:= signal_output;
```

Chapter 12

Edge Detection Instructions

R_TRIG

Rising edge trigger

Description The function block R_TRIG (rising edge trigger) allows you to recognize a rising edge at an input.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For R_TRIG declare the following:

CLK **signal input**

the output Q is set for each rising edge at the signal input (clk = clock)

Q **signal output**

is set when a rising edge is detected at CLK.

PLC types: **Availability of R_TRIG (see page 934)**

The output Q of a function block R_TRIG remains set for a complete PLC cycle after the occurrence of a rising edge (status change FALSE -> TRUE) at the CLK input and is then reset in the following cycle.

Data types

Data types	I/O	Function
BOOL	input CLK	detects rising edge for clock
BOOL	output Q	set when rising edge detected

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

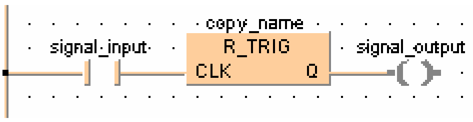
POU Header

All input and output variables which are used for programming the function block R_TRIG are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial
0	VAR	copy_name	R_TRIG	
1	VAR	signal_input	BOOL	FALSE
2	VAR	signal_output	BOOL	FALSE

Body **Signal_output** will be set, if a rising edge is detected at **signal_input.**

LD



ST

```
copy_name( CLK:= signal_input ,
        Q=> signal_output );
```

F_TRIG

Falling edge trigger

Description The function block F_TRIG (falling edge trigger) allows you to recognize a falling edge at an input.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For F_TRIG declare the following:

CLK **signal input**

the output Q is set for each falling edge at the signal input (clk = clock)

Q **signal output**

is set if a falling edge is detected at CLK.

PLC types: **Availability of F_TRIG (see page 932)**

The output Q of a function block F_TRIG remains set for a complete PLC cycle after the occurrence of a falling edge (status change TRUE -> FALSE) at the CLK input and is then reset in the following cycle.

Data types

Data types	I/O	Function
BOOL	input CLK	detects falling edge at input clock
BOOL	output Q	is set if falling edge is detected at input

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block F_TRIG are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial
0	VAR	copy_name	F_TRIG	
1	VAR	signal_input	BOOL	FALSE
2	VAR	signal_output	BOOL	FALSE

Body

Signal_output will be set, if a falling edge is detected at **signal_input.**

LD

```
ST   copy_name( CLK:= signal_input ,
               Q=> signal_output );
```

Chapter 13

Counter Instructions

CTU Up counter

Steps: 31

Description The function block CTU (count up) allows you to program counting procedures.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

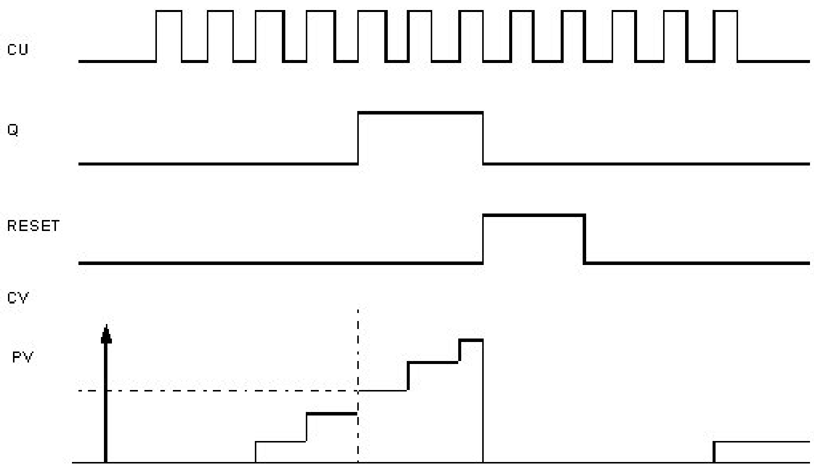
For CTU declare the following:

CU	**clock generator** the value 1 is added to CV for each rising edge at CU, except when RESET is set
RESET (R)	**reset** CV is reset to zero for each rising edge at RESET
PV	**set value** if PV (preset value) is reached, Q is set
Q	**signal output** is set if CV is greater than/equal to PV
CV	**current value** contains the addition result (CV = current value)

☞ **The names in brackets are the valid parameter names of the ST-editor.**

PLC types: **Availability of CTU (see page 924)**

Time Chart:



Data types

Data types	I/O	Function
BOOL	input CU	detects rising edge, adds 1 to CV
BOOL	input RESET	resets CV to 0 at rising edge
INT	input PV	set value
BOOL	output Q	set if CV >= PV
INT	output CV	current value

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

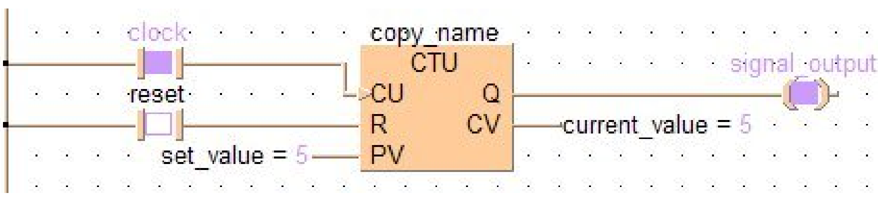
All input and output variables which are used for programming the function block CTU are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTU		under this identifier a copy of the CTU function block is saved and a separate data area is reserved
1	VAR	clock	BOOL	FALSE	upward counter input
2	VAR	reset	BOOL	FALSE	reset input (reset to 0)
3	VAR	set_value	INT	0	default (PV=preset value)
4	VAR	signal_output	BOOL	FALSE	
5	VAR	current_value	INT	0	current counter value (EV=elapsed value)

Body

If **reset** is set (status = TRUE), **current_value** (CV) will be reset. If a rising edge is detected at **clock**, the value 1 will be added to **current_value**. If a rising edge is detected at **clock,** this procedure will be repeated until **current_value** is greater than/equal to **set_value**. Then, **signal_output** will be set.

LD



ST

```
copy_name( CU:= clock, RESET:= reset, PV:= set_value, Q:=
signal_output, CV:= current_value);
```

CTD

Down counter

Steps: 31

Description The function block CTD (count down) allows you to program counting procedures.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For CTD declare the following:

CD — **clock generator input**
the value 1 is subtracted from the current value CV for each rising edge detected at CD, except when LOAD is set or CV has reached the value zero.

LOAD (LD) — **set**
with LOAD the counter state is reset to PV

PV — **preset value**
is the value subjected to subtraction during the first counting procedure

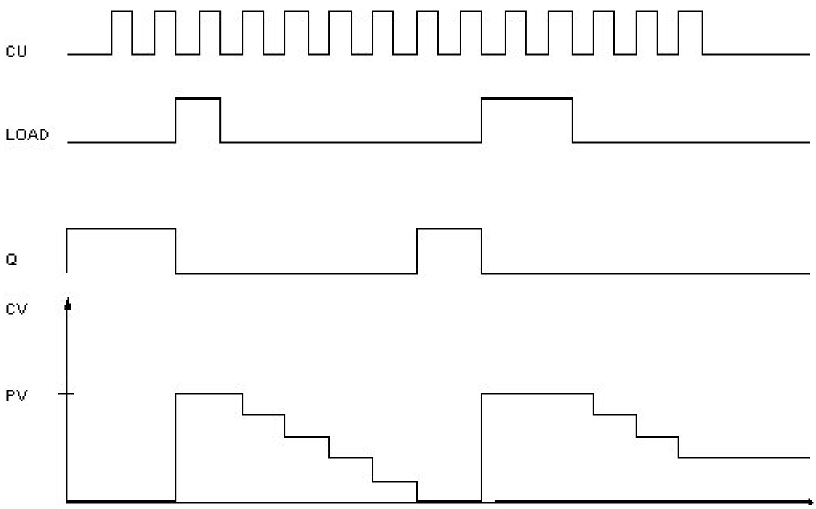
Q — **signal output**
is set if CV = zero

CV — **current value**
contains the current subtraction result (CV = current value)

The names in brackets are the valid parameter names of the ST-editor.

PLC types: **Availability of CTD (see page 924)**

Time Chart:



Data types

Data types	I/O	Function
BOOL	input CD	subtracts 1 from CV at rising edge
BOOL	input LOAD	resets counter to PV
INT	input PV	preset value
BOOL	output Q	signal output, set if CV = 0
INT	output CV	current value

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

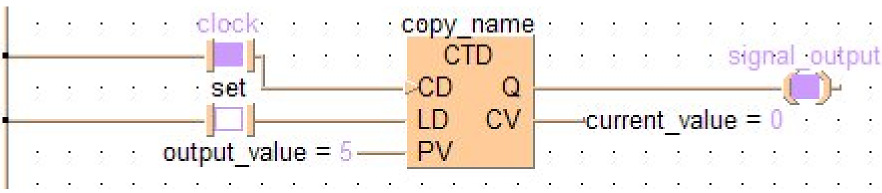
All input and output variables which are used for programming the function block CTD are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTD		under this identifier a copy of the CTD function block is saved and a separate data area is reserved
1	VAR	clock	BOOL	FALSE	downward counter input
2	VAR	set	BOOL	FALSE	set input (set to preset value (PV))
3	VAR	output_value	INT	0	minuend
4	VAR	signal_output	BOOL	FALSE	
5	VAR	current_value	INT	0	current counter value

Body

If **set** is set (status = TRUE), the **preset_value** (PV) is loaded in the **current_value** (CV). The value 1 will be subtracted from the **current_value** each time a rising edge is detected at **clock.** This procedure will be repeated until the **current_value** is greater than/equal to zero. Then, **signal_output** will be set.

LD



ST When programming with structured text,structured text, use CTD as follows:

```
IF set THEN (* first cycle *)
      load:=TRUE;             (* load has to be TRUE,
                                         to set
current_value to output_value *)
      clock:=FALSE;
END_IF;
copy_name(CD:= clock, LOAD:= set, PV:= output_value, Q:=
signal_output, CV:= current_value);
load:=FALSE;      (* now current_value got the right value,
load doesn't need to be *)
                  (* TRUE any longer *);
```

CTUD

Up/down counter

Steps: 66

Description The function block CTUD (count up/down) allows you to program counting procedures (up and down).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

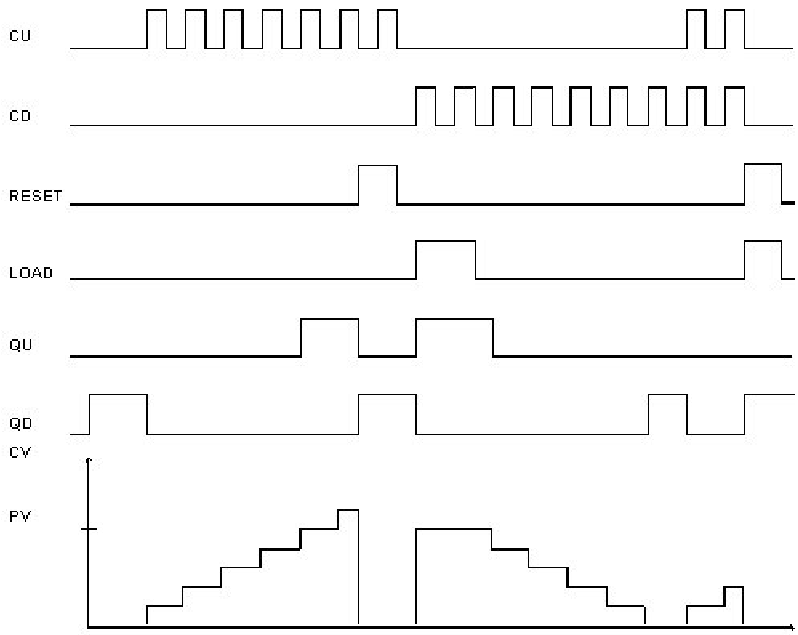
For CTUD declare the following:

CU	**count up**	
	the value 1 is added to the current CV for each rising edge detected at CU, except when RESET and/or LOAD is/are set.	
CD	**count down**	
	the value 1 is subtracted from the current CV for each rising edge detected at CD, except when RESET and/or LOAD is/are set and if CU and CD are simultaneously set. In the latter case, counting will be upwards.	
RESET (R)	**reset**	
	if RESET is set, CV will be reset	
LOAD (LD)	**set**	
	if LOAD is set, PV is loaded to CV. This, however, does not apply if RESET is set simultaneously. In this case, LOAD will be ignored.	
PV	**preset value**	
	defines the preset value which is to be attained with the addition or subtraction (PV = preset value)	
QU	**signal output - count up**	
	is set if CV is greater than/equal to PV	
QD	**signal output - count down**	
	is set if CV = zero	
CV	**current value**	
	is the addition/subtraction result (CV = current value)	

☞ **The names in brackets are the valid parameter names of the ST-editor.**

PLC types: **Availability of CTUD (see page 924)**

Time Chart:



Data types

Data types	I/O	Function
BOOL	input CU	count up
BOOL	input CD	count down
BOOL	input RESET	resets CV if set
BOOL	input LOAD	loads PV to CV
INT	input PV	set value
BOOL	output QU	signal output count up
BOOL	output QD	signal output count down
INT	output CV	current value

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block CTUD are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTUD		under this identifier a copy of the CTUD function block is saved and a separate data area is reserved
1	VAR	up_clock	BOOL	FALSE	upward counter input
2	VAR	down_clock	BOOL	FALSE	downward counter input
3	VAR	reset	BOOL	FALSE	reset input (reset to 0)
4	VAR	set	BOOL	FALSE	set input (set to set_value)
5	VAR	set_value	INT	0	default
6	VAR	output_up	BOOL	FALSE	
7	VAR	output_down	BOOL	FALSE	
8	VAR	current_value	INT	0	current counter value

Body

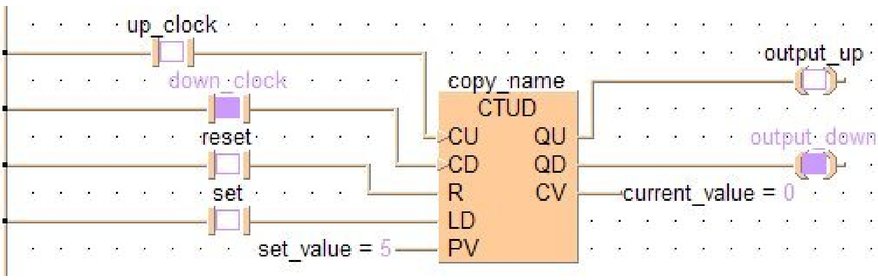
Count up:

If **reset** is set, the **current_value** (CV) will be reset. If up_**clock** is set, the value 1 is added to the **current_value**. This procedure is repeated for each rising edge detected at up_**clock** until the **current value** is greater than/equal to the **set_value**. Then **output_up** is set. The procedure is not conducted, if **reset** and/or **set** is/are set.

Count down:

If **set** is set (status = TRUE), the **set_value** (PV = preset value) will be loaded in the **current_value** (CV). If **down_clock** is set, the value 1 is subtracted from **set_value** at each clock. This procedure is repeated at each clock until the **current_value** is smaller than/equal to zero. Then, **signal_output** is set. The procedure will not be conducted, if **reset** and/or **set** is/are set or if CU and CV are set at the same time. In the latter case, counting will be downwards.

LD



ST

```
copy_name(CU:= up_clock, CD:= down_clock, RESET:= reset,
LOAD:= set, PV:= set_value,
              QU:= output_up, QD:= output_down, CV:=
current_value);
```

Chapter 14

Timer Instructions

TP Timer with defined period

Steps: 14

Description The function block TP allows you to program a pulse timer with a defined clock period.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For TP declare the following:

IN **clock generator**

if a rising edge is detected at IN, a clock is generated having the period defined in PT

PT **clock period**

(16-bit value: 0 - 327.27s, 32-bit value: 0 -21,474,836.47s; resolution 10ms each) a timer having the period **PT** is caused for each rising edge at **IN**. A new rising edge detected at **IN** within the pulse period does not cause a new timer (see time chart, section 2)

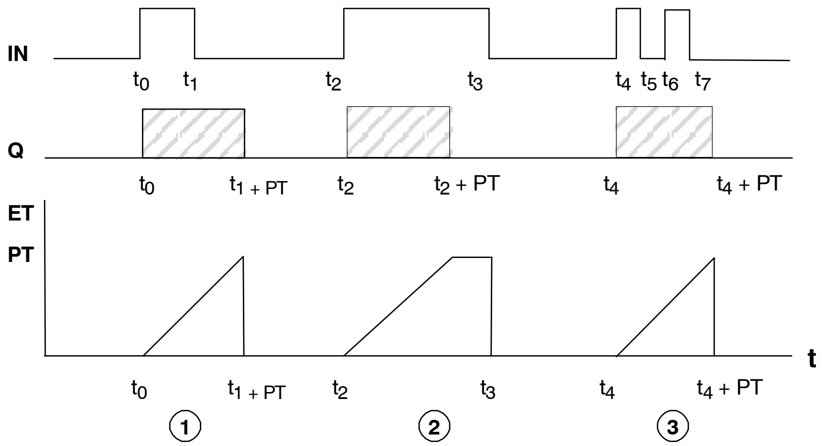
Q **signal output**

is set for the period of PT as soon as a rising edge is detected at **IN**

ET **elapsed time**

contains the elapsed period of the timer. If **PT** = **ET**, **Q** will be reset

Time Chart:



1 + 2 Independent of the turn-on period of the IN signal, a clock is generated at the output Q having a length defined by PT. The function block TP is triggered if a rising edge is detected at the input IN.

3 A rising edge at the input IN does not have any influence during the processing of PT.

PLC types: **Availability of TP (see page 936)**

Data types

Data types	I/O	Function
BOOL	input IN	clock generated according to clock period at rising edge
TIME	input PT	clock period
BOOL	output Q	signal output
TIME	output ET	elapsed time

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

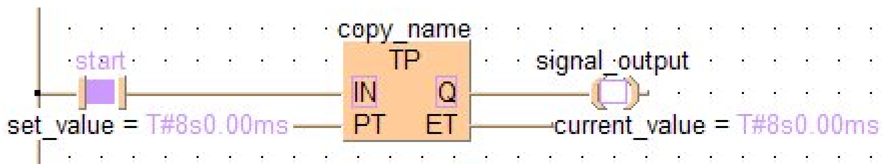
All input and output variables which are used for programming the function block TP are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

	Class	Identifier	Type	Initial
0	VAR	copy_name	TP	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body

If **start** is set (status = TRUE), the clock is emitted at **signal_output** until the **set_value** for the clock period is reached.

LD



ST

```
copy_name( IN:= start, PT:= set_value);
        signal_output := signal_output;
        current_value:= current_value;
```

TON

Timer with switch-on delay

Steps: 7

Description The function block TON allows you to program a switch-on delay.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For TON declare the following:

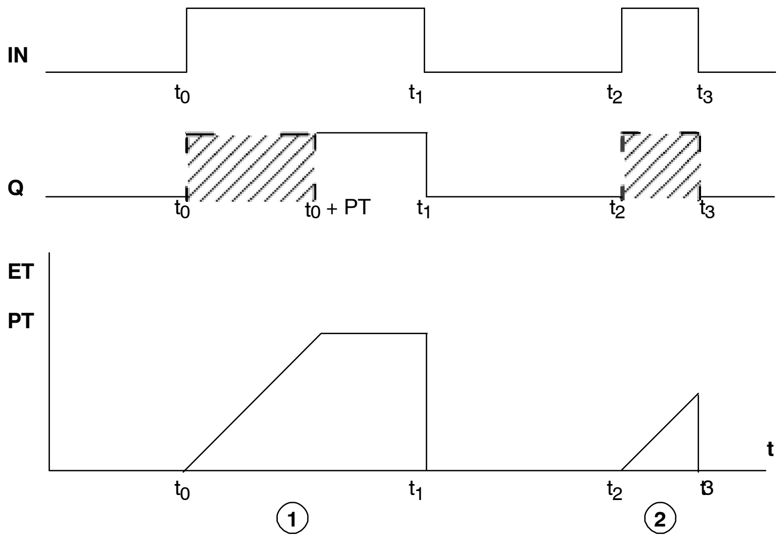
IN **timer ON**
an internal timer is started for each rising edge detected at IN

PT **switch-on delay**
(16-bit value: 0 - 327.27s, 32-bit value: 0 - 21,474,836.47s; resolution 10ms each) the switch-on delay is defined here (PT = preset time)

Q **signal output**
is set if PT = ET

ET **elapsed time**
indicates the current value of the elapsed time

Time Chart:



1. **Q** is set delayed with the time defined in **PT**. Resetting is without any delay.
2. If the input **IN** is only set for the period of the delay time **PT** or even for a shorter period of time (t3 - t2 < PT), **Q** will not be set.

PLC types: **Availability of TON (see page 936)**

Data types

Data types	I/O	Function
BOOL (IN)	input	internal timer starts at rising edge
TIME (PT)	input	switch on delay
BOOL (Q)	output	signal output set if PT = ET
TIME (ET)	output	elapsed time

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

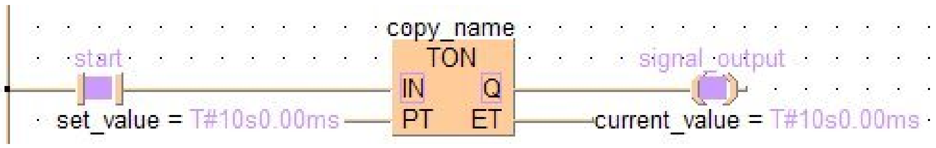
All input and output variables which are used for programming the function block TON are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

	Class	Identifier	Type	Initial
0	VAR	copy_name	TON	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body

If **start** is set (status = TRUE), the input signal is transferred to **signal_output** with a delay by the time period **set_value**.

LD



ST

```
copy_name( IN:= start ,
        PT:= set_value ,
        Q=> signal_output ,
        ET=> current_value );
```

TOF — Timer with switch-off delay

Steps: 23

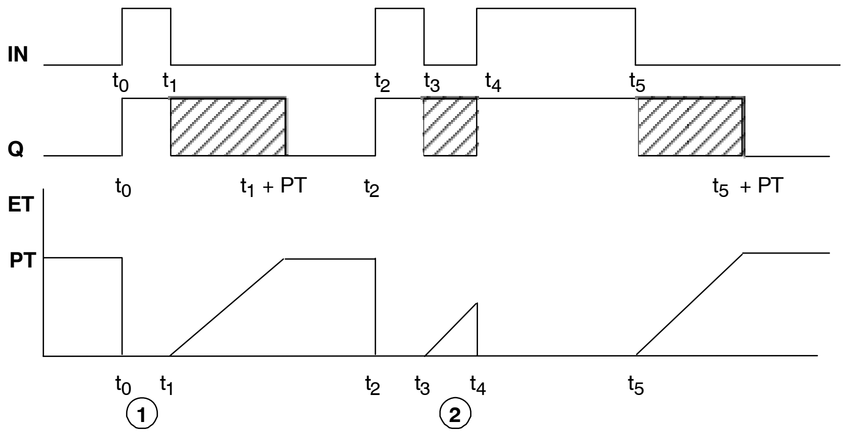
Description The function block TOF allows you to program a switch-off delay, e.g. to switch off the ventilator of a machine at a later point in time than the machine itself.

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

For TON declare the following:

IN	**timer ON**	an internal timer is started if a falling edge is detected at IN. If a rising edge is detected at IN before PT has reached its value, Q will not be switched off (see time chart, section 2)
PT	**switch-off delay**	(16-bit value: 0 - 327.27s, 32-bit value: 0 - 21,474,836.47s; resolution 10ms each) the switch-off delay is defined here (PT = preset time)
Q	**signal output**	is reset if PT = ET
ET	**elapsed time**	represents the current value of the elapsed time

Time Chart:



1 **Q** is switched off with a delay corresponding to the time defined in **PT**. Switching on is carried out without delay.

2 If **IN** (as in the time chart on top for t3 to t4) is set prior to the lapse of the delay time **PT**, **Q** remains set (time chart for t2 to t3).

PLC types: **Availability of TOF (see page 936)**

Data types

Data types	I/O	Function
BOOL (IN)	input	internal timer on a falling edge
TIME (PT)	input	switch off delay
BOOL (Q)	output	signal output reset if PT = ET
TIME (ET)	output	elapsed time

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

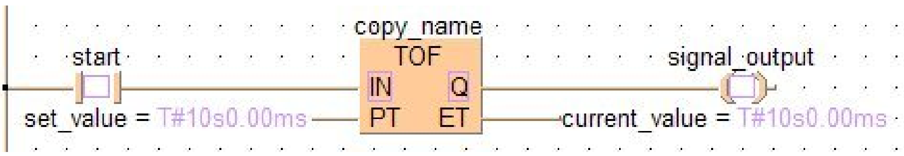
All input and output variables which are used for programming the function block TOF are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

	Class	Identifier	Type	Initial
0	VAR	copy_name	TOF	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body

If **start** is reset, this signal is transferred to **signal_output** with a delay corresponding to the period of time **set_value**.

LD



ST

```
copy_name( IN:= start ,
        PT:= set_value ,
        Q=> signal_output ,
        ET=> current_value );
```

Chapter 15

Data Transfer Instructions

15.1 Data Transfer Within the PLC

In This Section:

F0_MV 16-bit data move

Steps: 5

Description The 16-bit data or 16-bit equivalent constant specified by **s** is copied to the 16-bit area specified by **d**, if the trigger **EN** is in the ON-state.

PLC types: **Availability of F0_MV (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	source 16-bit area
d	INT, WORD	destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#0089

source

bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
s	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1

dest.

bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
d	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1

Destination value in this example: 16#0089

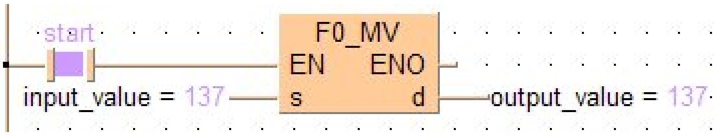
Example In this example the function F0_MV is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	137	contains the source value
2	VAR	output_value	INT	0	the area, where the source value will be copied to. result after a 0->1 leading edge from start: 137

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F0_MV(input_value, output_value);
END_IF;
```

F1_DMV 32-bit data move

Steps: 7

Description The 32-bit data or 32-bit equivalent constant specified by **s** is copied to the 32-bit area specified by **d**, if the trigger **EN** is in the ON-state.

PLC types: **Availability of F1_DMV (see page 925)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source 32-bit area
d	DINT, DWORD	destination 32-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Explanation with example value 16#ACAEE486

source

bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 1 0	1 1 0 0	1 0 1 0	1 1 1 0	1 1 1 0	0 1 0 0	1 0 0 0	0 1 1 0

32-bit area

dest.

bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 1 0	1 1 0 0	1 0 1 0	1 1 1 0	1 1 1 0	0 1 0 0	1 0 0 0	0 1 1 0

Destination value in this example: 16#ACAEE486

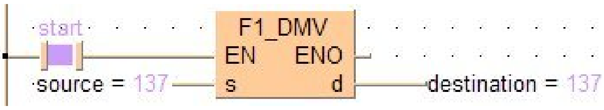
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source	DINT	137	contains the source value
2	VAR	destination	DINT	0	the area, where the source value will be copied to result after a 0->1 leading edge from start: 137

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F1_DMV(source, destination);
END_IF;
```

F2_MVN

16-bit data inversion and move

Steps: 5

Description The 16-bit data or 16-bit equivalent constant specified by **s** is inverted and transferred to the 16-bit area specified by **d** if the trigger **EN** is in the ON-state.

PLC types: **Availability of F2_MVN (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	source 16-bit area to be inverted
d	INT, WORD	destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#5555

source	15 . . 12	11 . . 8	7 . . 4	3 . . 0
bit	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1

dest.	15 . . 12	11 . . 8	7 . . 4	3 . . 0
bit	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0

Each bit is inverted, target value in this example: 16#AAAA

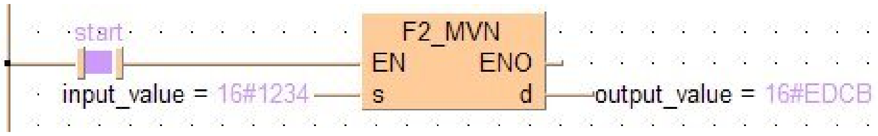
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#1234	this value will be inverted
2	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#EDCB

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F2_MVN(input_value, output_value);
END_IF;
```

F3_DMVN

32-bit data inversion and move

Steps: 7

Description The 32-bit data or 32-bit equivalent constant specified by **s** is inverted and transferred to the 32-bit area specified by **d** if the trigger **EN** is in the ON-state.

PLC types: **Availability of F3_DMVN (see page 925)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source 32-bit area to be inverted
d	DINT, DWORD	destination 32-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Explanation with example value 16#75BCD15

source

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 1 1 1	0 1 0 1	1 0 1 1	1 1 0 0	1 1 0 1	0 0 0 1	0 1 0 1

32-bit area

dest.

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 1 1 1	1 0 0 0	1 0 1 0	0 1 0 0	0 0 1 1	0 0 1 0	1 1 1 0	1 0 1 0

Each bit is inverted, destination value in this example: 16#F8A432EA

Example In this example the function F3_DMVN is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DWORD	16#00001234	this value will be inverted
2	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#FFFFEDCB

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F3_DMVN(input_value, output_value);
END_IF;
```

F7_MV2

Two 16-bit data move

Steps: 7

Description The two 16-bit data or two 16-bit equivalent constants specified by **s1** and **s2** are copied to the 32-bit area specified by **d** when the trigger turns ON.

PLC types: **Availability of F7_MV2 (see page 925)**

☞ **To transfer three 16-bit data types, use either the F190_MV3 (see page 274) or P190_MV3 instruction.**

Data types

Variable	Data type	Function
s1, s2	INT, WORD	source 16-bit area
d	DINT, DWORD	destination 32-bit area

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	WORD	16#ABCD	
2	VAR	input_value_2	WORD	16#1234	
3	VAR	output_value	DWORD	0	result: here 16#1234ABCD

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F7_MV2(input_value1, input_value2, output_value);
END_IF;
```

F8_DMV2 Two 32-bit data move

Steps: 11

Description The function copies two 32-bit data areas specified at inputs **s1** and **s2** to a 32-bit ARRAY with two elements at output **d**.

PLC types: **Availability of F8_DMV2 (see page 925)**

☞ **To copy three 32-bit data, use either the F191_DMV3 (see page 276) or P191_DMV3 instruction.**

Data types

Variable	Data type	Function
s1, s2	DINT, DWORD	source 32-bit area
d	ARRAY **[0..1] of** DINT **or** DWORD	destination, lower 32-bit area of 64-bit area

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

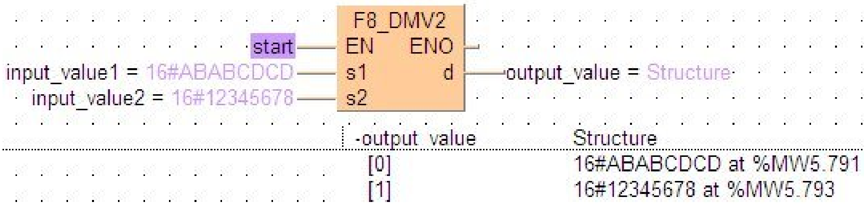
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#ABABCDCD	
2	VAR	input_value_2	DWORD	16#12345678	
3	VAR	output_value	ARRAY [0..1] OF DWORD	[2(0)]	result: here output_value[0] = 16#ABABCDCD output_value[1] = 16#12345678

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST
```
IF start THEN
    F8_DMV2(input_value_1, input_value_2, output_value);
END_IF;
```

F190_MV3 — Three 16-bit data move

Steps: 10

Description The function copies three 16-bit data values at inputs **s1**, **s2** and **s3** to an ARRAY with three elements that is returned at output **d**.

PLC types: **Availability of F190_MV3** **(see page 930)**

☞ **To transfer two 16-bit data types, use either the F7_MV2** **(see page 271)** **or P7_MV2 instruction.**

Data types

Variable	Data type	Function
s1, s2, s3	INT, WORD	source 16-bit area
d	ARRAY **[0..2] of** WORD, INT	destination, lower 16-bit area of 48-bit area

The variables **s1**, **s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1,s2,s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

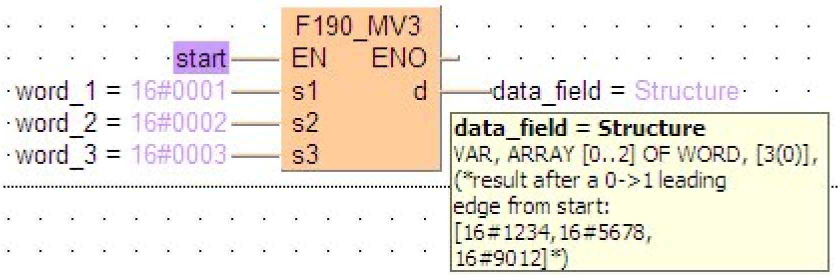
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header — In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
1	VAR	word_1	WORD	1
2	VAR	word_2	WORD	2
3	VAR	word_3	WORD	3
4	VAR	data_field	ARRAY [0..2] OF WORD	[3(0)]

Body — When the variable **start** is set to TRUE, the function is carried out.

LD



ST
```
IF start THEN
    F190_MV3(word_1, word_2, word_3, data_field);
END_IF;
```

F191_DMV3 Three 32-bit data move

Steps: 16

Description The function copies three 32-bit data values at inputs **s1**, **s2** and **s3** to an ARRAY with three elements that is returned at output **d**.

PLC types: **Availability of F191_DMV3 (see page 930)**

To transfer two 32-bit data types, use either the F8_DMV2 (see page 272) or P8_DMV2 instruction.

Data types

Variable	Data type	Function
s1, s2, s3	DINT, DWORD	source 32-bit area
d	ARRAY **[0..2] of** DINT **or** DWORD	destination, lower 32-bit area of 96-bit area

The variables **s1, s2, s3** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1,s2,s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

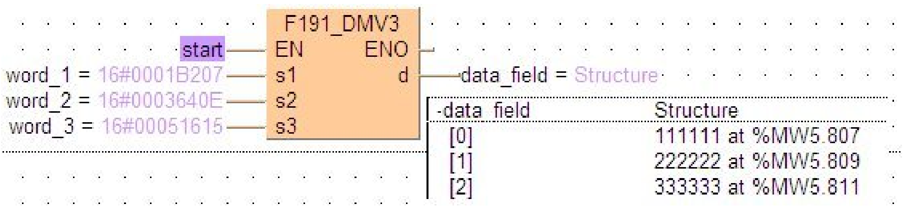
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	word_1	DWORD	111111	
2	VAR	word_2	DWORD	222222	
3	VAR	word_3	DWORD	333333	
4	VAR	data_field	ARRAY [0..2] OF DWORD	[3(0)]	result here: [111111,222222,333333]

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F191_DMV3(word_1, word_2, word_3, data_field);
END_IF;
```

F10_BKMV

Block move

Steps: 7

Description The data block specified by the 16-bit starting area specified by **s1** and the 16-bit ending area specified by **s2** are copied to the block starting from the 16-bit area specified by **d** if the trigger **EN** is in the ON-state.

The operands **s1** and **s2** should be:

- in the same operand
- **s1** ≤ **s2**

Whenever **s1**, **s2** and **d** are in the same data area:

- **s1** = **d**: data will be recopied to the same data area.

source	15 . 12	11 . . 8	7 . 4	3 . . 0
[0]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
[2]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
[3]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
[4]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 1

dest.	15 . 12	11 . . 8	7 . 4	3 . . 0
[0]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
[2]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0

PLC types: **Availability of F10_BKMV (see page 925)**

Data types

Variable	Data type	Function
s1	INT, WORD	starting 16-bit area, source
s2	INT, WORD	ending 16-bit area, source
d	INT, WORD	starting 16-bit area, destination

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Part III F/P Instructions

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

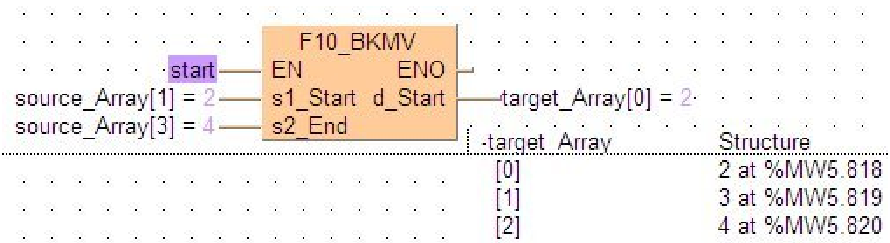
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_Array	ARRAY [0..4] OF INT	[1,2,3,4,5]	
2	VAR	target_Array	ARRAY [0..2] OF INT	[3(0)]	result after a 0->1 leading edge from start: [2,3,4]

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. It moves the data block starting at the 16-bit area specified by **s1** and ending at the 16-bit area specified by **s2** to the 16-bit area specified by **d**.

LD



ST

```
IF start THEN
    F10_BKMV( s1_Start:= source_Array[1],
        s2_End:= source_Array[3],
        d_Start=> target_Array[0]);
END_IF;
```

F10_BKMV_NUMBER Block move by number

Steps: 7

Description The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by **d_Start** if the trigger EN is in the ON-state.

This instruction is a modification of the F10_BKMV (see page 277) generated by the compiler.

Whenever **s1_Start** and **d_Start** are in the same data area:

- s1_Start = d_Start: data will be recopied to the same data area.

PLC types: **Availability of F10_BKMV_NUMBER (see page 925)**

The value for 's2_number' has to be greater than 0.

Data types

Variable	Data type	Function
s1_Start	INT, WORD	starting 16-bit area, source
s2_Number	INT, WORD	number of words to be copied, source
d_Start	INT, WORD	starting 16-bit area, destination

The variables **s1_Start, s2_Number** and **d_Start** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F10_BKMV_NUMBER is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

POU Header

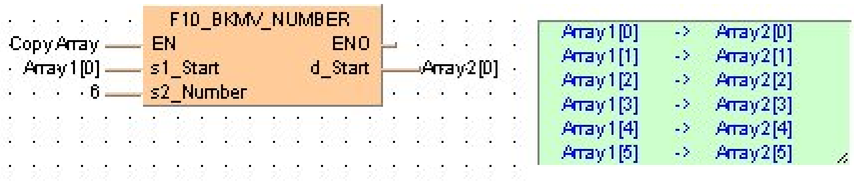
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	Array2	ARRAY [0..5] OF INT	[6(0)]
2	VAR	CopyArray	BOOL	FALSE

Body

When the variable **CopyArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by **d_Start**.

LD


F10_BKMV_NUMBER
CopyArray
EN
ENO
Array1[0]
s1_Start
d_Start
Array2[0]
6
s2_Number
Array1[0] -> Array2[0]
Array1[1] -> Array2[1]
Array1[2] -> Array2[2]
Array1[3] -> Array2[3]
Array1[4] -> Array2[4]
Array1[5] -> Array2[5]

F10_BKMV_OFFSET Block move to an offset from source

Steps: 7

Description

This instruction is a modification of the F10_BKMV (see page 277) generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and 16-bit ending area specified by **s2_End** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger EN is in the ON-state.

Whenever **s1_Start** and **s2_End** are in the same data area:

- d_Offset = 0: data will be recopied to the same data area.

PLC types: **Availability of F10_BKMV_OFFSET (see page 925)**

Data types

Variable	Data type	Function
s1_Start	INT, WORD	starting 16-bit area, source
s2_End	INT, WORD	ending 16-bit area, source
d_Offset	INT, WORD	offset from s1_Start, destination

The variables **s1_Start**, **s2_End** and **d_Offset** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start, s2_End	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_Offset	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example

In this example the function F10_BKMV_OFFSET is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

POU Header

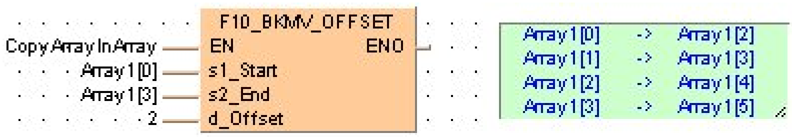
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	CopyArrayInArray	BOOL	FALSE

Body

When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and 16-bit ending area specified by **s2_End** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

LD



F10_BKMV_NUMBER_OFFSET

Block move by number to an offset from source

Steps: 7

Description

This instruction is a modification of the F10_BKMV (see page 277) generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger EN is in the ON-state.

Whenever **d_Offset** = 0: data will be recopied to the same data area.

PLC types: **Availability of F10_BKMV_NUMBER_OFFSET (see page 925)**

☞ **The value for 's2_number' has to be greater than 0.**

Data types

Variable	Data type	Function
s1_Start	INT, WORD	starting 16-bit area, source
s2_Number	INT, WORD	Number of words to be copied, source
d_Offset	INT, WORD	starting 16-bit area, destination

The variables **s1_Start**, **s2_Number** and **d_Offset** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Offset	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example

In this example the function F10_BKMV_NUMBER_OFFSET is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	CopyArrayInArray	BOOL	FALSE

Body

When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

LD


F10_BKMV_NUMBER_OFFSET
CopyArrayInArray
EN
ENO
Array1[0]
s1_Start
4
s2_Number
2
d_Offset
Array1[0] -> Array1[2]
Array1[1] -> Array1[3]
Array1[2] -> Array1[4]
Array1[3] -> Array1[5]

F11_COPY

Block copy

Steps: 7

Description The 16-bit equivalent constant or 16-bit area specified by **s** is copied to all 16-bit areas of the block specified by **d1** and **d2** if the trigger **EN** is in the ON-state.

The operands **d1** and **d2** should be:

- in the same operand
- **d1** $\leq$ **d2**

source	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

dest.	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
[2]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
[3]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
[4]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
[5]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

PLC types: **Availability of F11_COPY (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	source 16-bit area
d1	INT, WORD	starting 16-bit area, destination
d2	INT, WORD	ending 16-bit area, destination

The variables **s, d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F11_COPY is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

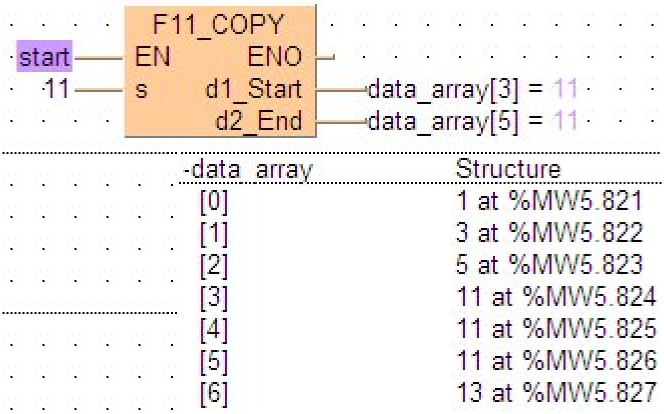
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_array	ARRAY [0..6] OF INT	[1,3,5,7,9,11,13]	result after a 0->1 leading edge from start: [1,3,5,11,11,11,13]

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    (* Copy the value 11 to data_array[3], *)
    (* data_array[4] and data_array[5] *)
    F11_COPY( s:= 11,
         d1_Start=> data_array[3],
         d2_End=> data_array[5]);
END_IF;
```

F15_XCH 16-bit data exchange

Steps: 5

Description The contents in the 16-bit areas specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON-state.

Bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
d1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
d2	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

PLC types: **Availability of F15_XCH (see page 925)**

Data types

Variable	Data type	Function
d1	INT, WORD	16-bit area to be exchanged with **d2**
d2	INT, WORD	16-bit area to be exchanged with **d1**

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F15_XCH is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	INT	17	result after a 0->1 leading edge from start: 24
2	VAR	value_2	INT	24	result after a 0->1 leading edge from start: 17

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F15_XCH(value_1, value_2);
END_IF;
```

F16_DXCH

32-bit data exchange

Steps: 5

Description Two 32-bit data specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON-state.

Bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d1	0 0 0 0	1 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
d2	0 0 0 0	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0

32-bit area

PLC types: **Availability of F16_DXCH (see page 925)**

Data types

Variable	Data type	Function
d1	DINT, DWORD	32-bit area to be exchanged with **d2**
d2	DINT, DWORD	32-bit area to be exchanged with **d1**

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	DINT	17	result after a 0->1 leading edge from start: 24
2	VAR	value_2	DINT	24	result after a 0->1 leading edge from start: 17

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F16_DXCH(value_1, value_2);
END_IF;
```

F17_SWAP Higher/lower byte in 16-bit data exchange

Steps: 3

Description The higher byte (higher 8-bits) and lower bytes (lower 8-bits) of a 16-bit area specified by **d** are exchanged if the trigger **EN** is in the ON-state. 1 byte means 8 bit.

Bit	15	.	.	12	10	.	.	8	7	.	.	4	3	.	.	0
DT770	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1
16#	2				3				4				5			

higher byte (8-bit) lower byte (8-bit)

Bit	15	.	.	12	10	.	.	8	7	.	.	4	3	.	.	0
DT770	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1
16#	4				5				2				3			

PLC types: **Availability of F17_SWAP (see page 925)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area in which the higher and lower bytes are swapped (exchanged)

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

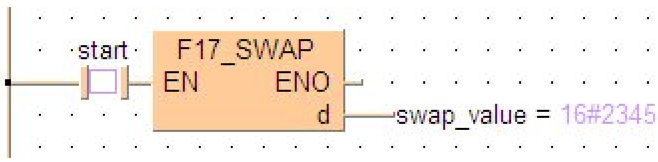
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	swap_value	WORD	16#2345	result after 0->1 leading edge from start: 16#4523

Body When the variable **start** is set to TRUE, the function is executed.

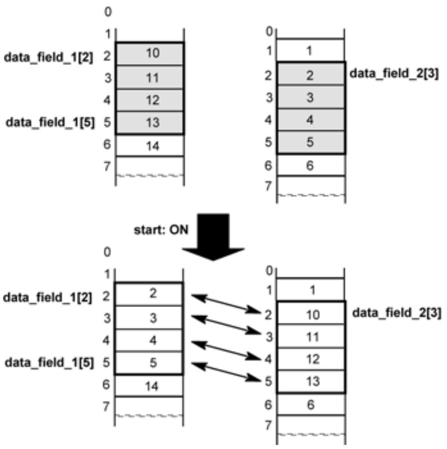
LD



ST
```
IF start THEN
    F17_SWAP(swap_value);
END_IF;
```

F18_BXCH — 16-bit blocked data exchange — Steps: 7

Description The function exchanges one 16-bit data block for another. The beginning of the first data block is specified at output **d1** and its end at output **d2**. Output **d3** specifies the beginning of the second data block.



PLC types: **Availability of F18_BXCH (see page 925)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area of block data 1
d2	INT, WORD	ending 16-bit area of block data 1
d3	INT, WORD	starting 16-bit area of block data 2

Operands

For	Relay				T/C		Register			Constant
d1, d2, d3	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variables at outputs **d1** > **d2** - the data block to be exchanged is larger than the target area.
R9008	%MX0.900.8	for an instant	

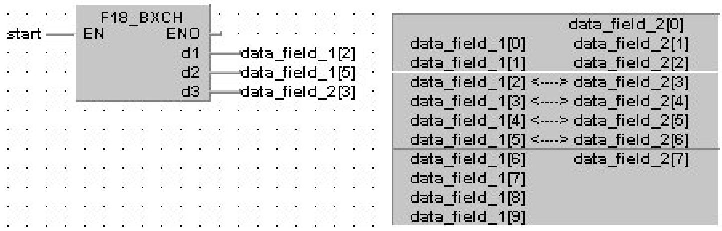
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field_1	ARRAY [0..9] OF INT	[8,9,10,11,12,13,14,15,16,17]	Arbitrarily large data field
2	VAR	data_field_2	ARRAY [0..7] OF INT	[-1,0,1,2,3,4,5,6]	Arbitrarily large data field

Body When the variable **start** is set to TRUE, the function is carried out. It exchanges the data of ARRAY **data_field_1** (from the 2nd to the 5th element) with the data of ARRAY **data_field_2** (from the 3rd element on).

LD



ST

```
IF start THEN
    F18_BXCH(
         d1_Start=> d1[2], d2_End=> d1[4], d3_Start=>
d2[1]);
END_IF;
```

F147_PR **Parallel printout** **Steps: 5**

Description Outputs the ASCII codes for 12 characters stored in the 6-word area specified by **s** via the word external output relay specified by **d** if the trigger **EN** is in the ON-state. If a printer is connected to the output specified by **d**, a character corresponding to the output ASCII code is printed.

Only bit positions 0 to 8 of **d** are used in the actual printout. ASCII code is output in sequence starting with the lower byte of the starting area. Three scans are required for 1 character constant output. Therefore, 37 scans are required until all characters constants are output.

Since it is not possible to execute multiple **F147_PR** instructions in one scan, use print-out flag R9033 to be sure they are not executed simultaneously. If the character constants convert to ASCII code, use of the F95_ASC (see page 640) instruction is recommended.

PLC types: **Availability of F147_PR (see page 929)**

Data types

Variable	Data type	Function
s	INT, WORD	starting 16-bit area for storing 12 bytes (6 words) of ASCII codes (source)
d	WORD	word external output relay used for output of ASCII codes (destination)

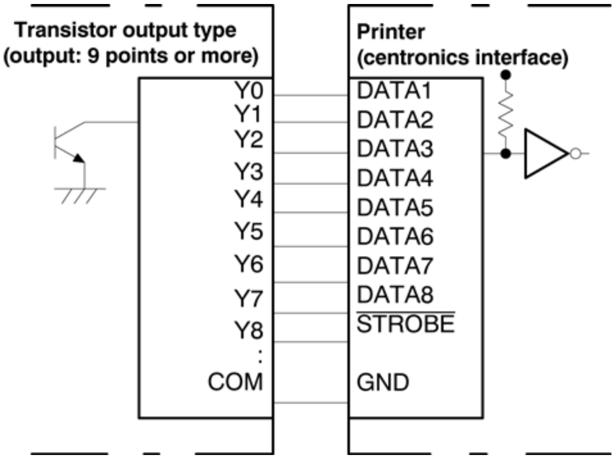
Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	-	-	-	-	-	-	-	-

Error flags

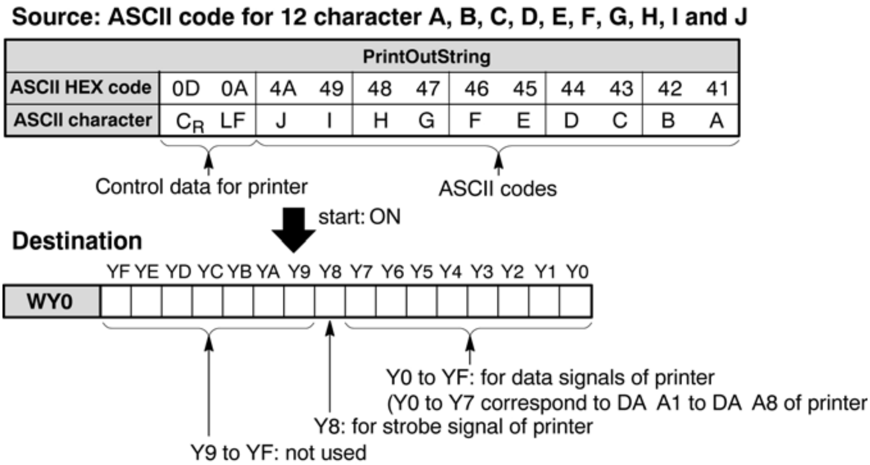
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the ending area for storing ASCII codes exceeds the limit - the trigger of another F147_PR instruction turns on while one F147_PR instruction is being executed
R9008	%MX0.900.8	for an instant	
R9033	%MX0.903.3	permanently	- a F147_PR instruction is being executed

■ Connection example



Example In this example the function F147_PR is programmed in ladder diagram (LD).

The ASCII codes stored in the string **PrintOutString** are output through word external output relay WY0 when trigger **Start** turns on.



GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

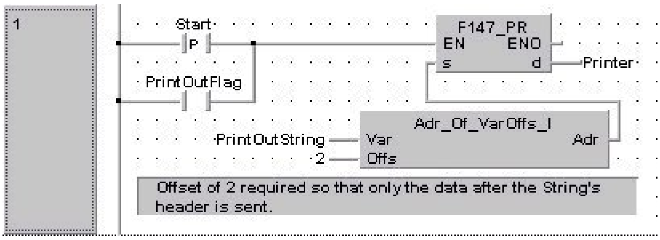
Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Printer	WY0	%QW0	WORD	0
1	VAR_GLOBAL	Print Out Flag	R9033	%MX0.903.3	BOOL	FALSE

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR_EXTERNAL	PrintOutFlag	BOOL	FALSE	
2	VAR	PrintOutString	STRING[12]	'ABCDEFGHIJLR'	$L = line feed $R = carriage return
3	VAR_EXTERNAL	Printer	WORD	0	

Body
LD



ST

```
IF DF(start) OR PrintOutFlag THEN
    F147_PR( Adr_Of_VarOffs( PrintOutString, 2), Printer);
END_IF;
```

15.2 Data Transfer Between PLCs and Modules

In This Section:

F143_IORF **Partial I/O update** **Steps: 5**

Description

- **If** d1 **and** d2 **are variables and not constants, then the compiler automatically accesses the variables' values via the index register.**
- **With input refreshing, WX0 should be specified for** d1 **and** d2**.**
- **With output refreshing, WY0 should be specified for** d1 **and** d2**.**

PLC types: **Availability of F143_IORF (see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting word address
d2	INT, WORD	ending word address

The same type of operand should be specified for **d1** and **d2**.

Operands

For	Relay				T/C		Register			Constant
d1	WX	WY	-	WL	SV	EV	DT	-	FL	dec. or hex.
d2	WX	WY	-	WL	SV	EV	DT	-	FL	dec. or hex.

Example In this example the function F143_IORF is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

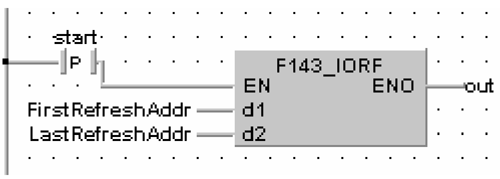
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	FirstRefreshAddr	INT	10	
1	VAR	LastRefreshAddr	INT	10	

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. To update WX10 and WY10 based on the master I/O map configuration, set d1 = 10 and d2 = 10.

LD



ST

F12_EPRD

EEPROM read from memory

Steps: 11

Description Using this instruction data will be copied from EEPROM/ Flash-ROM to the destination area (DT). The copy function is carried out with blocks only. Thus you can not copy single words. The block size and the number of blocks is shown in the table "PLC specific information". Also ensure that there at least 64/ 2048 free data registers (1 block = 64 words/ 2048 words (DTs)) reserved for the destination area.

PLC types: **Availability of F12_EPRD**

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function block (when EN has the state TRUE, the function block will be executed at every PLC scan)
s1	DINT, DWORD	EEPROM start block number
s2	DINT, DWORD	Number of blocks to be read (1 block = 64 words/ 2048 words (DTs))
d	INT, WORD	DT start address for information to be written
ENO	BOOL	When the function block was executed, ENO is set to TRUE. Helpful at cascading of function blocks with EN-functionality

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	-	-	-	-	-	DT	-	-	-

■ PLC specific information

PLC type	FP-Sigma FP-X
ROM	Flash-ROM
Block size (1 block)	2048 words
EEPROM start block number	0 to 15
Number of blocks to be read / written each execution	1 (writing) 1 to 16 (reading)
Write duration (Additional scan time)	< 100ms each block
Read duration (Additional scan time)	9.94µs + (1562.6*number of blocks) µs
Max number of writing events ☞ **Power down, RUN -> PROG mode changes are also counted.**	10,000
Max read times	No limit

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

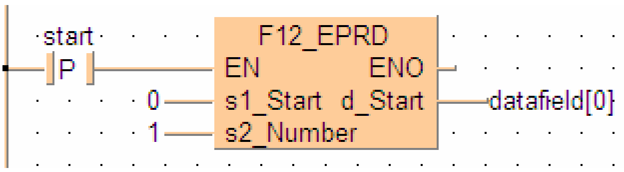
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data field	ARRAY [0..63] OF INT	[64(0)]	data field to be uploaded data from EEPROM

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. The function reads the first block (= 64 words) after start block number 0 from the EEPROM and writes the information into the data fields from data field[0] until data field[63].

LD



P13_EPWT

EEPROM write to memory

Steps: 11

Description Using this instruction data will be copied from the data area (DT) to the EEPROM/ Flash-ROM.

The EEPROM memory is not the same as the hold area. The hold area stores data in real time. Whenever the power shuts down, the hold data is stored in the EEPROM memory. The P13_EPWT instruction sends data into the EEPROM only when the instruction is executed. It also has a limitation of the number of times you can write to it (see table below). You must make sure that the P13_EPWT instruction will not be executed more often than the specified number of writes.

For example, if you execute P13_EPWT with R901A relay (pulse time 0.1s), the EEPROM will become inoperable after 100,000 * 0.1 sec=10,000 sec (2.8 hours). However if you want to hold your profile data such as positioning parameters or any other parameter values that are changed infrequently, you will find this instruction very useful.

PLC types: **Availability of P13_EPWT**

☞ **One of the two input variables 's2' or 'd' has to be assigned constant number value.**

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function block (when EN changes from FALSE to TRUE, the function block will be executed one time)
s1	INT, WORD	DT start address of the block(s) that you want to save
s2	DINT, DWORD	Number of blocks to write (1 block = 64 words/ 2048 words (DTs))
d	DINT, DWORD	EEPROM start block number
ENO	BOOL	When the function block was executed, ENO is set to TRUE. Helpful at cascading of function blocks with EN-functionality

Operands

For	Relay				T/C		Register			Constant
s1	-	-	-	-	-	-	DT	-	-	-
s2, d	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.

■ PLC specific information

PLC type	FP-Sigma FP-X
ROM	Flash-ROM
Block size (1 block)	2048 words
EEPROM start block number	0 to 15
Number of blocks to be read / written each execution	1 (writing) 1 to 16 (reading)

PLC type	FP-Sigma FP-X
Write duration (Additional scan time)	< 100ms each block
Read duration (Additional scan time)	9.94µs + (1562.6*number of blocks) µs
Max number of writing events **Power down, RUN -> PROG mode changes are also counted.**	10,000
Max read times	No limit

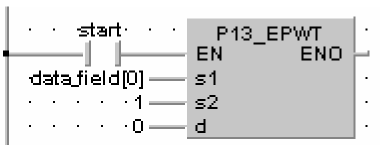
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data field	ARRAY [0..63] OF INT	[1,2,3,4,5,6,7,8,9,10,11,12,52(0)]	data field to be uploaded data from EEPROM

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. The function reads the contents of data field[0] until data field[63] (s2* = 1 => 1 block = 64 words) and writes the information after start block number 0 into the EEPROM.

LD



F150_READ

Data read from intelligent units

Steps: 9

Description Reads data from the shared memory in an intelligent module.

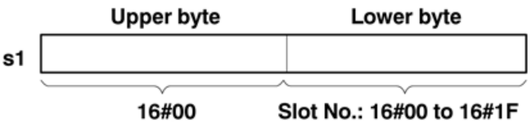
The **n** words of the data stored in the shared memory of the intelligent unit/board specified by **s1** are read from the address specified by **s2**, and are stored in the area specified by **d** of the CPU.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

Intelligent unit without bank

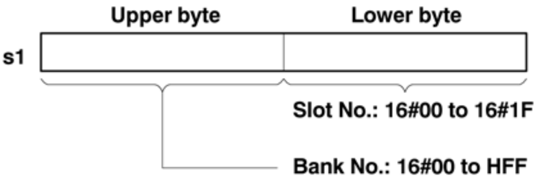
Specifying s1

Specify the slot number in which the target intelligent unit has been installed.



Intelligent unit with bank

Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).



Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092

PLC types: **Availability of F150_READ (see page 929)**

Data types

Variable	Data type	Function
s1	INT, WORD	Specifies the bank/slot number in the shared memory of the intelligent module
s2	INT, WORD	Specifies the starting address in the shared memory of the intelligent module (source data address)
n	INT	Specifies the number of words to be read
d	INT, WORD	Starting address in the CPU for storing data read (destination address)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- s1 exceeds the limit of specified range - the data read exceeds the area of d
R9008	%MX0.900.8	for an instant	

Example

In this example the function F150_READ is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

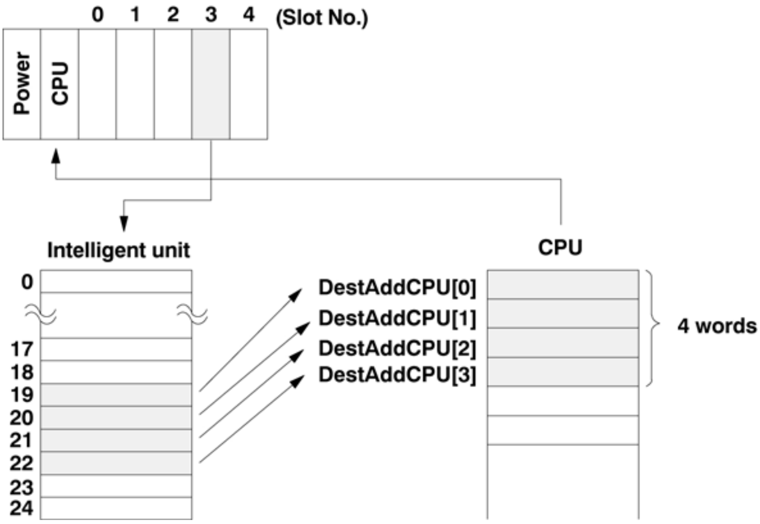
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

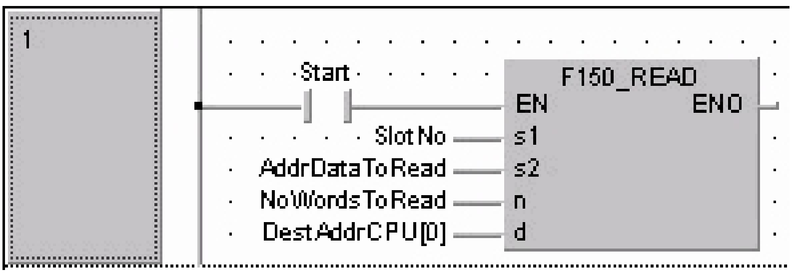
	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	SlotNo	WORD	16#03	
2	VAR	AddrDataToRead	INT	19	Starting address in intelligent unit for data to be read (source)
3	VAR	NoWordsToRead	INT	4	
4	VAR	DestAddrCPU	ARRAY [0..3] OF INT	[4(0)]	Starting address in CPU to store data read

Body

Reads 4 words of data stored in the addresses starting from 19, specified in **AddrDataToRead**, of the intelligent unit's shared memory (located in slot 3). Then it stores them in the array **DestAddrCPU**, when Start turns on.



LD



ST

```
IF start THEN
    F150_READ( s1_BankSlot:= SlotNo,
         s2_Start:= AddrDataToRead,
         n_Number:= NoWordsToRead,
         d_Start:= DestAddrCPU[0]);
END_IF;
```

F151_WRT

Data read from intelligent units

Steps: 9

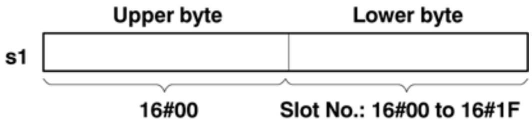
Description Writes data from the shared memory into the memory of an intelligent unit.

Writes **n** words of the initial data from the area specified by **s2** of the CPU to the address specified by **d** of the shared memory of the intelligent unit specified by **s1**.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

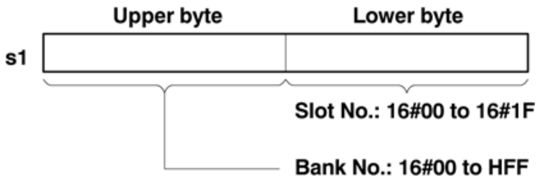
Intelligent unit without bank

Specifying s1 Specify the slot number in which the target intelligent unit has been installed.



Intelligent unit with bank

Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).



Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092

PLC types: **Availability of F151_WRT (see page 929)**

Data types

Variable	Data type	Function
s1	INT, WORD	Specifies the bank/slot number in the shared memory of the intelligent module
s2	INT, WORD	Starting address for data in the shared memory of the CPU
n	INT	Specifies the number of words to be written to the shared memory
d	INT, WORD	Specifies the starting address in the intelligent unit for storing data written (destination address)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- s1 exceeds the limit of specified range - the data read exceeds the area of d
R9008	%MX0.900.8	for an instant	

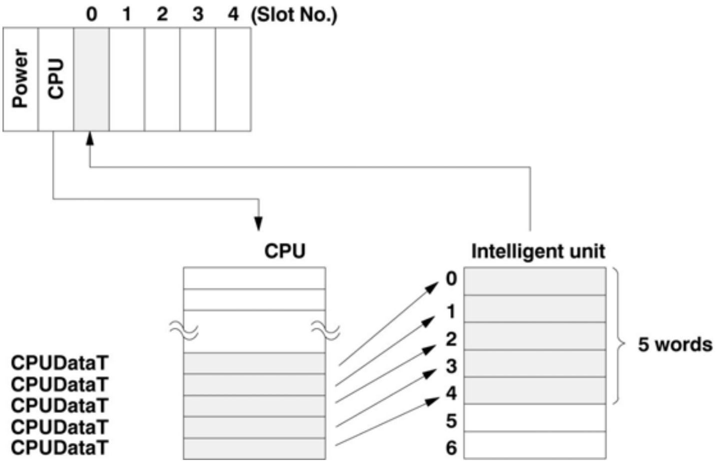
Example In this example the function F151_WRT is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

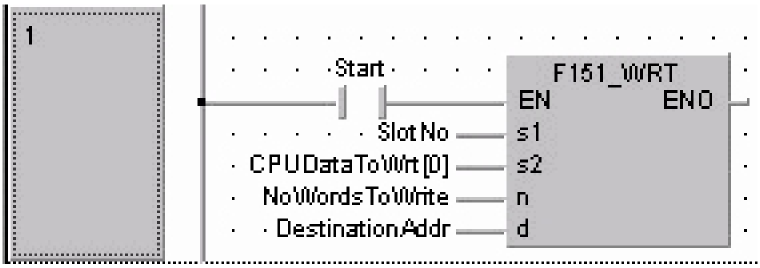
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	SlotNo	WORD	16#00	
2	VAR	CPUDataToWrt	ARRAY [0..4] OF INT	[5,10,15,20,25]	
3	VAR	NoWordsToWrite	INT	5	
4	VAR	DestinationAddr	INT	0	Starting 16-bit address for storing data in the intelligent unit

Body Five words of data defined in **CPUDataToWrt** are written into the addresses starting from 0 to 4 of the intelligent unit's shared memory (located in slot 0) when Start turns on.



LD



ST

```
IF start THEN
    F151_WRT( s1_BankSlot:= SlotNo,
          s2_Start:= CPUDataToWrt[0],
          n_Number:= NoWordsToWrite,
          d_Start:= DestinationAddr);
END_IF;
```

15.3 Data Transfer Between PLCs and Other Devices (via COM Port or Network)

15.3.1 Transmission and Reception of Data via COM Ports

◆ REFERENCE

For detailed information on installation and wiring, please refer to the hardware manuals of the corresponding units.

15.3.1.1 Description of the Communication Modes

Data transmission and reception can be carried out using the following modes:

MEWTOCOL-COM Slave (Computer Link)

MEWTOCOL-COM Slave (Computer Link) is used for communication with a computer or another PLC using the MEWTOCOL protocol. Instructions (command messages) are transmitted to the PLC, and the PLC responds (sends response messages) based on the instructions received.

The proprietary MEWNET protocol called MEWTOCOL-COM is used to exchange data between the computer and the PLC. There are two different communication methods, 1:1 and 1:N communication. A 1:N network is called a C-NET. If the PLC is used as a slave in a 1:N network (C-NET), the system register entry 'COM port unit number' specifies the PLC's station number in the network.

The PLC answers automatically to commands received from the computer, so no program is necessary on the PLC side in order to carry out communication.

Program Controlled Mode

In program controlled serial communication, data is sent and received via the COM ports to and from an external device, e.g. an imagechecker or bar code reader.

The PLC can act as a master or slave using any protocol. The PLC program handles the protocol.

Sending the data (see page 321)

For all PLC types	The instruction F159_MTRN (see page 324) can be used to send the data. IsTransmissionDone (see page 311) can be used to detect the end of the transmission.

Receiving the data (see page 327)

For all PLC types	The instructions IsReceptionDone (see page 312) or IsReceptionDoneByTimeOut (see page 314) should be used to detect the end of the data received. Both instructions should also be used to start the analysis of the data received.

COM Port of the CPUs	Receiving the data is done automatically in a reception buffer (see page 329), which can be configured by the system registers 'COM Port data register starting address for receive buffer' and 'COM Port receive buffer capacity'.
COM Port of the MCUs	It is possible to copy the data received to a reception area using the instruction F161_MRCV (see page 330). This should be done only after IsReceptionDone (see page 312) or IsReceptionDoneByTimeOut (see page 314) has evaluated the data. (Polling the data using F161_MRCV does not work correctly!)

Clear the reception buffer (see page 331) and reset the reception done flags

COM Port of the CPUs	A subsequent execution of the send instruction F159_MTRN (also with NumberOfBytes equal to zero) clears the reception buffer and resets the "reception done flag". The COM port is again ready to receive subsequent data.
COM Port of the MCUs	The use of F161_MRCV also implicitly clears the reception buffer and resets the "reception done flag". The COM port is again ready to receive subsequent data.

PLC Link Mode

In a PLC link, data is shared with all PLCs connected via MEWNET using dedicated internal relays called link relays (L) and data registers called link registers (LD).

If the link relay contact for one PLC goes on, the same link relay also goes on in each of the other PLCs connected to the network. Likewise, if the contents of a link register are rewritten in one PLC, the change is made in the same link register of each of the other PLCs connected to the network.

The status of the link relays and link registers in any one PLC is fed back to all of the other PLCs connected to the network. Hence control of data that needs to be consistent throughout the network, e.g. target production values and type codes, can easily be implemented to coordinate the data, and the data of all units is updated at the same time.

Modbus RTU Master/Slave

The PLC can act as a master or slave using the MODBUS RTU protocol.

It works as a master using the commands F145_MODBUS_WRITE_DATA (see page 333) and F146_MODBUS_READ_DATA (see page 343).

Otherwise the PLC is configured as a slave for communication with a computer or another PLC. The PLC answers automatically to commands received from the master, so no program is necessary on the PLC side in order to carry out communication.

Modbus RTU Slave

MODBUS RTU Slave is used for communication with a computer or another PLC using the MODBUS RTU protocol. Instructions (command messages) are transmitted to the PLC, and the PLC responds (sends response messages) based on the instructions received.

The PLC answers automatically to commands received from the master, so no program is necessary on the PLC side in order to carry out communication.

15.3.1.2 Setting the Communication Parameters

CPU: Setting the communication parameters for the COM ports		
	During PROG mode:	- via system registers (see page 309)
	During RUN time:	- F159 (see page 332) (switch communication mode with 16#8000) - SYS1 (see page 806) with FP-Sigma and FP-X - SYS2 (see page 818) with FP-Sigma and FP-X

Setting the CPU's COM Ports in PROG Mode via System Registers

For a general description on setting the system registers, please refer to setting the system registers.

PROCEDURE

1. **Choose "COM port" under "System Register" from the navigator**
   The number of the system register for the respective settings may vary according to the PLC type used. Please refer to the comment under "Additional Information" for the proper settings.

2. **Set the system register "COM port selection" to "Program controlled"**
   For an example on changing the use of the COM port with the programming software, refer to switching system registers during RUN mode (see page 332). Take into account that FP-Sigma has two COM ports.

   Possible settings of system register "COM port selection":

   - MEWTOCOL-COM Slave
   - Program controlled
   - PLC Link (MEWNET-W0/W)

3. **Set the transmission format**
   Set the transmission format parameter so that the "Transmission Format Setting" in the respective system register matches the external device connected to the COM port.

4. **Set the initial baud rate**
   Set the transmission speed so that the "COM Port Baud Rate Setting" in the respective system register matches the external device connected to the COM port.

Setting the CPU's COM Ports in RUN Mode with SYS1 (FP-Sigma, FP-X)

Please refer to the description of SYS1, communication condition setting (see page 806).

Setting the CPU's COM Ports in RUN Time with SYS2 (FP-Sigma, FP-X)

Please refer to the description of SYS2 (see page 818).

15.3.1.3 Getting the Communication Parameters and Statuses

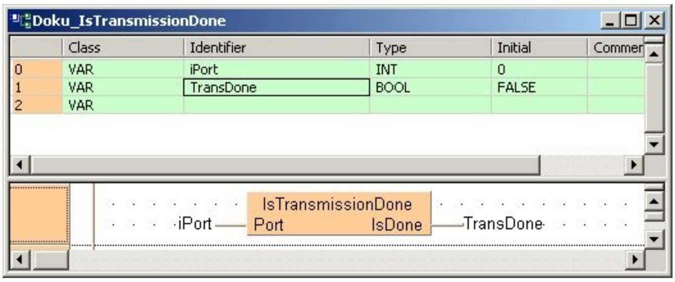
In This Section:

IsTransmissionDone

Returns the value of the "Transmission Done" flag

Description This function returns the value of the "Transmission Done" flag of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	R903F	sys_bIsToolPort TransmissionDone
	1	COM1 port	R9039	sys_bIsComPort1 TransmissionDone
	2	COM2 port	R9049	sys_bIsComPort2 TransmissionDone

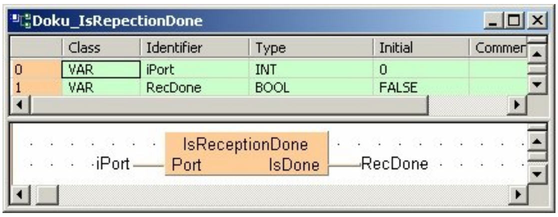
For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 4).

IsReceptionDone

Returns the value of the "Reception Done" flag

Description This function returns the value of the "Reception done flag" of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

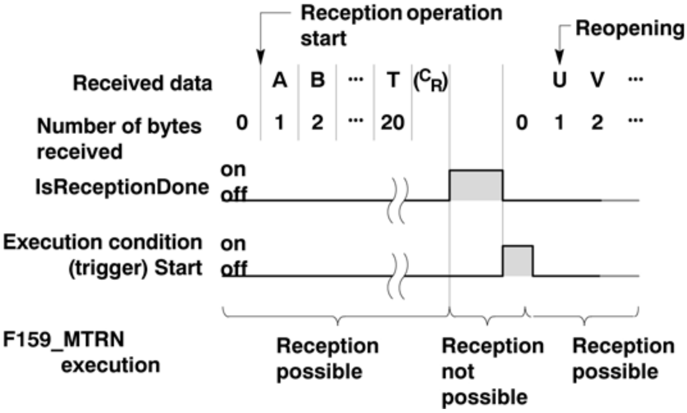
PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	R903E	sys_bIsToolPort ReceptionDone
	1	COM1 port	R9038	sys_bIsComPort1ReceptionDone
	2	COM2 port	R9048	sys_bIsComPort2ReceptionDone

For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

Operation of the IsReceptionDone Flag:

When the "reception done flag" is off and data is sent from an external device, operation will proceed as follows. (After RUN, "reception done flag" is off during the first scan.)

The data received is stored in order in the reception data storage area of reception buffer beginning from the lower byte of the second word of the area. Start and end codes will not be stored.
With each one byte received, the value in the leading address of the reception buffer is incremented by 1.



When an end code is received, the "reception done flag" goes on. After this, no further reception of data is allowed.

To continue receiving data, please refer to clearing the reception buffer (see page 331).

IsReceptionDoneBy Timeout

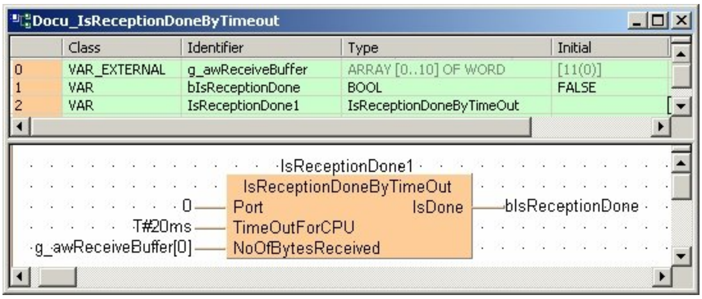
Evaluates a "Reception Done" condition

Description Depending on the PLC type and the input parameter **Port**, this function evaluates a "Reception Done" condition if no end terminator in the received data stream is expected.

If a CPU's COM port is selected, the first word of the ReceiveBuffer (see page 329), which indicates the number of bytes received, is evaluated. If it does not increment within the time specified by the input parameter **TimeOutForCPU**, the output **IsDone** is set.

If the MCU's COM port is selected, the MCU's "reception done (see page 312) flag" is evaluated. The timeout for this COM port must be entered via the MCU dialog or during RUN mode via F159_MWRT_PARA.

Example



This flag is evaluated depending on the PLC type:

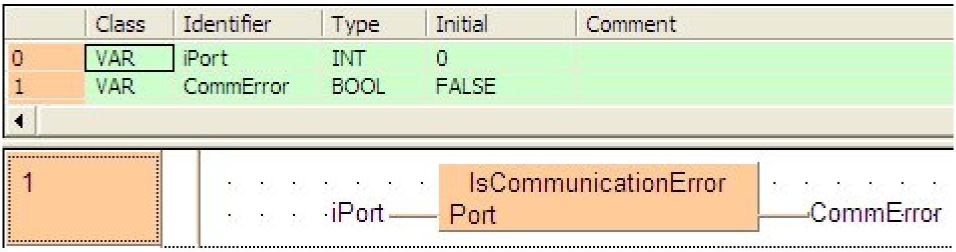
PLC	Port number	Port name	Flag/Condition
FP-Sigma	-	-	NoOfBytesReceived after TimeOutForCPU

IsCommunication Error

Returns the value of the "Communication Error" flag

Description This instruction returns the value of the "Communication Error" flag of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	R900E	sys_bIsToolPortCommunicationError
	1	COM1 port	R9037	sys_bIsComPort1CommunicationError
	2	COM2 port	R9047	sys_bIsComPort2CommunicationError

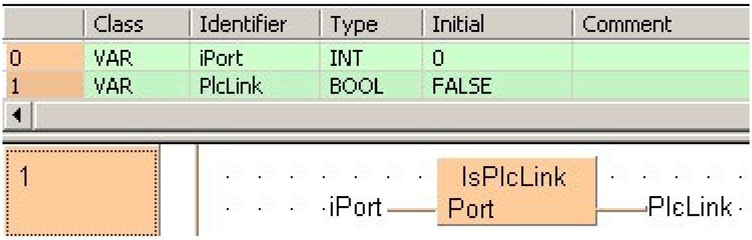
For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

IsPlcLink

Returns the value of the "PLC Link" flag

Description This instruction returns the value of the "PLC Link" flag of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	FALSE	sys_bIsComPort1PlcLink
	1	COM1 port	R9041	sys_bIsComPort1PlcLink
	2	COM2 port	FALSE (* PLC Link is possible but there is no IsPlcLink flag which can be evaluated)	

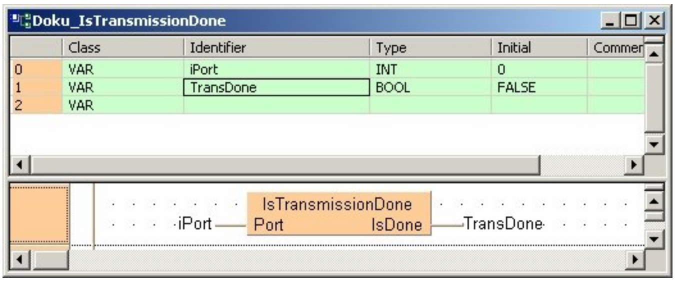
For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

IsProgramControlled

Returns the value of the "Program Controlled" flag

Description This instruction returns the value of the "Program Controlled" flag of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	R9040	sys_bIsToolPortProgramControlled
	1	COM1 port	R9032	sys_bIsComPort1ProgramControlled
	2	COM2 port	R9042	sys_bIsComPort2ProgramControlled

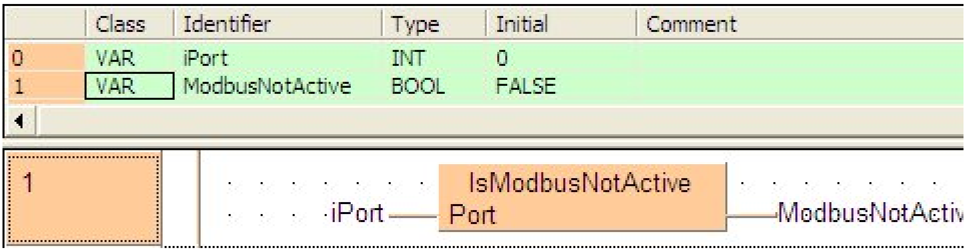
For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

IsModbusNotActive

Returns the value of the "IsModbusNotActive" flag

Description This instruction returns the value of the "Modbus Not Active" flag of the PLC's serial communication interface.

Example



This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	TRUE	-
	1	COM1 port	R9044	sys_bIsComPort1ModbusNotActive
	2	COM2 port	R904A	sys_bIsComPort2ModbusNotActive

For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

IsModbusError

Returns the value of the "Modbus Error" flag

Description This instruction returns the value of the "Modbus Error" flag of the PLC's serial communication interface.

Example

	Class	Identifier	Type	Initial	Comment
0	VAR	iPort	INT	0	
1	VAR	ModbusError	BOOL	FALSE	

This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	FALSE	-
	1	COM1 port	R9045	sys_bIsComPort1ModbusError
	2	COM2 port	R904B	sys_bIsComPort2ModbusError

For detailed information on using system variables, please refer to data transfer to and from special data (see page 4)registers.

15.3.1.4 Getting the Communication Parameters and Statuses in RUN Mode via Special Relays and Special Data Registers from the CPU's COM Ports

Address		Name	Description
COM1	COM2		
R9032	**R9042**	Program controlled mode flag	Turns on when the program controlled communication function is being used. Goes off when the MEWTOCOL-COM Slave or the PLC Link (MEWNET-W0/W) function is being used.
R9037	**R9047**	communication error flag	Goes on if a transmission error occurs during data communication. Goes off when a request is made to send data using the F159_MTRN instruction.
R9038	**R9048**	reception done flag	Turns on when the terminator is received during program controlled serial communication.
R9039	**R9049**	transmission done flag	Goes on when transmission has been completed in program controlled serial communication. Goes off when transmission is requested in program controlled serial communication.
R9041	-	PLC link flag	Turns on while the PLC Link (MEWNET-W0/W) is used.

15.3.1.5 Data Transfer in Program Controlled Mode

For all PLC types and all COM ports (including the MCU's COM port) the following instructions are available:

- Transmission (see page 321) and reception (see page 327) in program controlled mode using the instructions F159_MTRN (see page 324) and F161_MRCV (see page 330)
- IsReceptionDone (see page 312), IsTransmissionDone (see page 311) and IsReceptionDoneByTimeOut (see page 314)

◆ NOTE

F144_TRNS generates different code depending on which PLC type you use. To get PLC-independent code, do not use F144_TRNS or the explicit reception or transmission done flags (R9038...). Instead use F159_MTRN, F161_MRCV, IsReceptionDone, etc.

In program controlled serial communication, data is sent and received via the COM ports to and from an external device, e.g. an imagechecker or bar code reader.

The PLC can act as a master or slave using any protocol. The PLC program handles the protocol.

Sending the data (see page 321)

For all PLC types	The instruction F159_MTRN (see page 324) can be used to send the data. IsTransmissionDone (see page 311) can be used to detect the end of the transmission.

Receiving the data (see page 327)

For all PLC types	The instructions IsReceptionDone (see page 312) or IsReceptionDoneByTimeOut (see page 314) should be used to detect the end of the data received. Both instructions should also be used to start the analysis of the data received.
COM Port of the CPUs	Receiving the data is done automatically in a reception buffer (see page 329), which can be configured by the system registers 'COM Port data register starting address for receive buffer' and 'COM Port receive buffer capacity'.
COM Port of the MCUs	It is possible to copy the data received to a reception area using the instruction F161_MRCV (see page 330). This should be done only after IsReceptionDone (see page 312) or IsReceptionDoneByTimeOut (see page 314) has evaluated the data. (Polling the data using F161_MRCV does not work correctly!)

Clear the reception buffer (see page 331) and reset the reception done flags

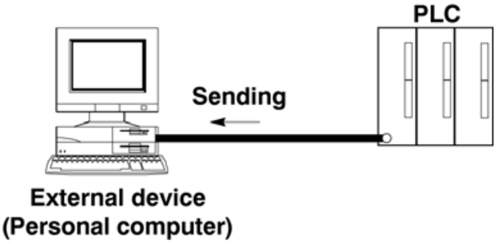
COM Port of the CPUs	A subsequent execution of the send instruction F159_MTRN (also with NumberOfBytes equal to zero) clears the reception buffer and resets the "reception done flag". The COM port is again ready to receive subsequent data.
COM Port of the MCUs	The use of F161_MRCV also implicitly clears the reception buffer and resets the "reception done flag". The COM port is again ready to receive subsequent data.

Transmission

To transmit, write the transmission data to the data table, select it with an F159_MTRN (see page 324) instruction, and execute.

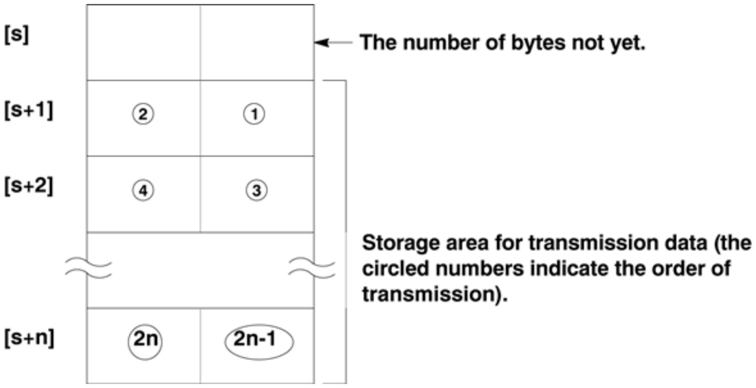
The **n** bytes of the data stored in the data table with the starting area specified by s are transmitted from the COM port or RS232C port to an external device by serial transmission.

A start code and end code can be automatically added before transmission.



Data table for transmission

Data register areas beginning with the area selected by **s** are used as the data table for transmission.

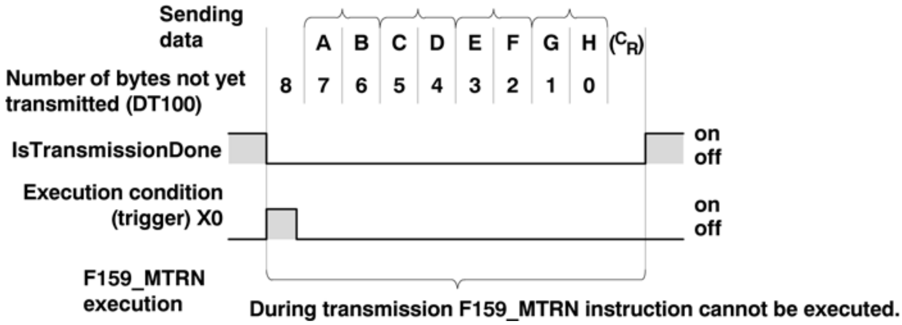


When the F159_MTRN (see page 324) instruction is executed, the number of data bytes not yet transmitted is stored in the starting area of the data table.

Operation:

Write the transmission data to the transmission data storage area selected with **s** (from the second word on). If the "transmission done (see page 311) flag" is on and the execution condition (trigger) for the F159_MTRN (see page 324) instruction turns on, operation will be as follows:

1. **n** is preset in **s** (the number of bytes not yet transmitted). Furthermore, the "reception done (see page 312) flag" is turned off and the number of bytes received is cleared to zero.
2. The data in the data table is transmitted in order from the lower byte.
   - As each byte is transmitted, the value in s (the number of bytes not yet transmitted) decrements by 1.
   - During transmission, the "transmission done flag" flag goes off.
   - If the start code has been set to "STX" in the system registers, the start code will be automatically added to the beginning of the data.
   - There is no restriction on the number of bytes n that can be transmitted. Following the initial area of the data **s**, transmission is possible up to the data range that can be used by the data register.
3. The end code selected is automatically added to the end of the data. (Do not include an end code in the transmission data.)
4. When the specified quantity of data has been transmitted, the value in **s** (the number of bytes not yet transmitted) will be zero and the "transmission done flag" will go on.



☞ **The F159_MTRN instruction cannot be executed and the "transmission done flag" is not turned on unless the CS pin of the COM port (RS232C) is on. If the remote station does not support the CTS signal, be sure to bridge the CS and and RS pins.**

Part III F/P Instructions

F159_MTRN

Serial Data Communication to CPU or MCU Port

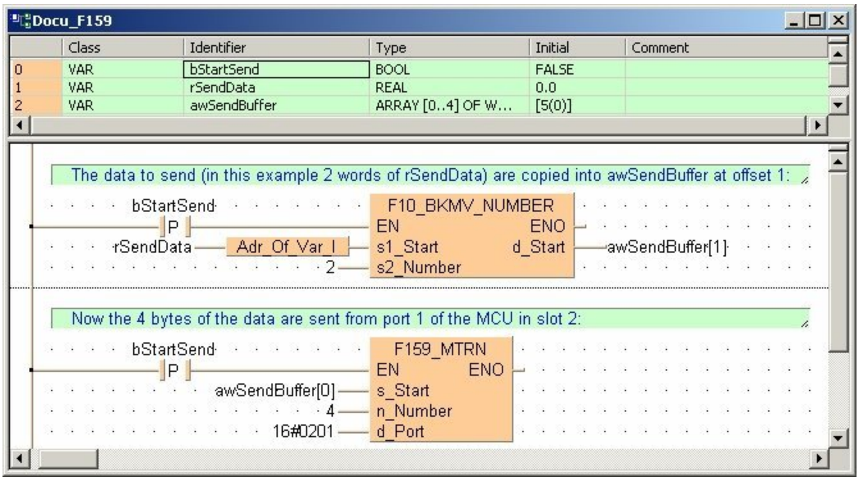
Steps: 7

Description This instruction is used to send data when an external device (computer, measuring instrument, bar code reader, etc.) has been connected to the specified RS232C port. If applied to the CPU's COM port, it also clears the receive buffer (see page 331), resets the "reception done flag" and allows further reception of data.

◆ **REFERENCE**

Please refer to the general description of transmission and reception of data (see page 307).

Example



PLC types: **Availability of F159_MTRN (see page 929)**

Data types

Variable	Data type	Function
s_Start	WORD	First element of the data table
n_Number	INT, WORD	Bytes to send: - Positive value: the terminal code is added in transmission. - Negative value: the terminal code is not added in transmission. - In case of 16#8000, the communication mode of the serial interface specified in transmission is changed.
d_Port	constant	Specification of the slot number and port number of the MCU unit to which the data is transmitted. **FP-X:** 0: Tool port 1: First port on the CPU 2: Second port on the CPU **FP-Sigma:** 1: First port on the CPU 2: Second port on the CPU **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter d_Port will be ignored)

Operands

For	Relay				T/C		Register			Constant
s_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Port	-	WY	WR	WL	SV	EV	DT	LD	FL	-

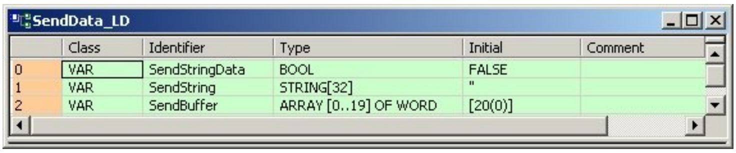
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the specified address using the index modifier exceeds a limit. - the transmitted byte number specified by '**n_Number**' is outside of the specified area. - 16#8000 is designated in the PC (PLC) link mode. **Flags only for the MCU:** - the MCU unit does not exist at the slot no. specified by '**d_Port**'.
R9008	%MX0.900.8	for an instant	

Example In this example the characters of the the string **SendString** are transmitted.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

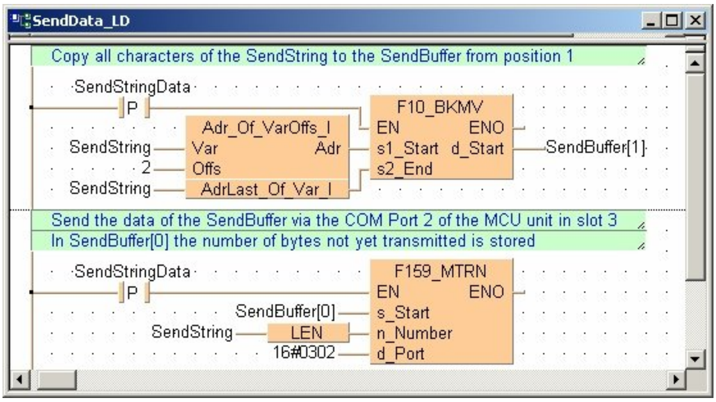

SendData_LD

	Class	Identifier	Type	Initial	Comment
0	VAR	SendStringData	BOOL	FALSE	
1	VAR	SendString	STRING[32]	''	
2	VAR	SendBuffer	ARRAY [0..19] OF WORD	[20(0)]	

Body

When the variable **SendStringData** is set to TRUE, the function F10_BKMV copies the characters from the string **SendString** to the buffer **SendBuffer** beginning at **SendBuffer[1]**. To get the first characters, an offset of 2 has to be added to the header of the string. Make sure that the send buffer is big enough for all the data to be sent. To determine its size you must take into account that two characters of the string **SendString** can be copied into each element of the array **SendBuffer**. **SendBuffer[0]** remains reserved to show the number of bytes not yet transmitted by the instruction F159_MTRN.

LD



ST

```
if (DF(SendStringData)) then
        (* Copy all characters of the SendString to the
SendBuffer from position 1 *)
        F10_BKMV(s1_Start := Adr_Of_VarOffs(Var := SendString,
Offs := 2),
                         s2_End  := AdrLast_Of_Var(SendString),
                         d_Start => SendBuffer[1]);
        (* Send the data of the SendBuffer via the COM Port 2
of the MCU unit in slot 3 *)
        (* In SendBuffer[0] the number of bytes not yet
transmitted is stored *)
        F159_MTRN(s_Start := SendBuffer[0], n_Number :=
LEN(SendString), d_Port := 16#0302);
end_if;
```

Further information:

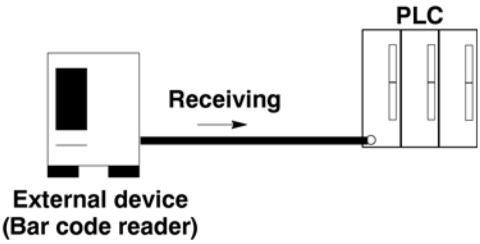
IsTransmissionDone (see page 311)

Reception

Reception is controlled by the "reception done (see page 312) flag". When this flag is off, the data sent to the COM port or RS232C port is stored in the reception buffer (see page 329) selected in system registers 417 and 418. When an F159_MTRN (see page 324) instruction is executed, the "reception done flag" goes off.

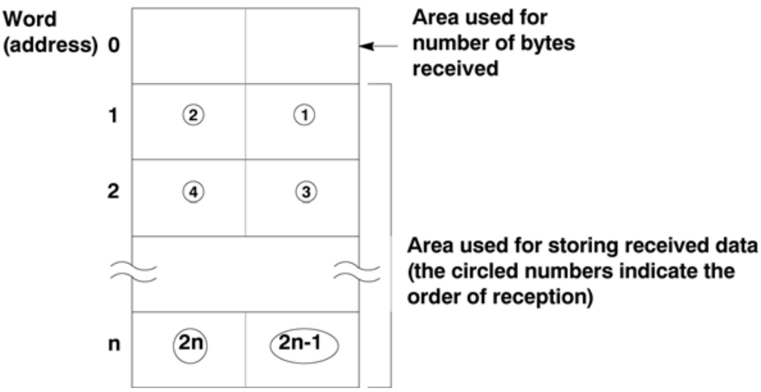
◆ **NOTE**

The number of the system register for the respective settings may vary according to the PLC type used.



Data sent from the external device connected to the COM port or RS232C port will be stored in the data register areas set as the reception buffer in system registers 417 and 418.

Reception buffer



Each time data is received, the amount of data received (number of bytes) is stored as a count in the leading address of the reception buffer. The initial value is zero.

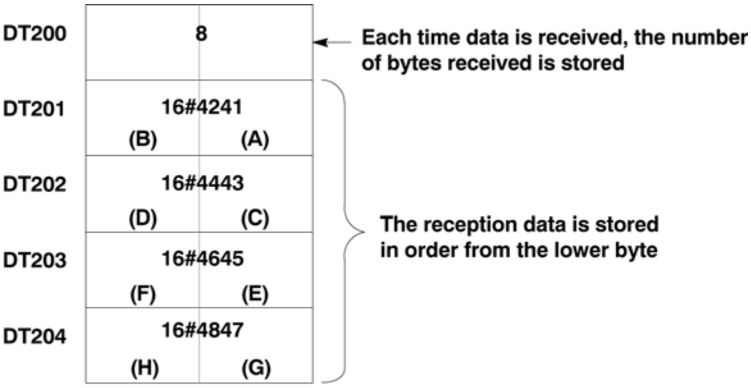
The data received is stored in order in the reception data storage area beginning from the lower byte of the second word of the area.

EXAMPLE

In this example, the eight characters A, B, C, D, E, F, G and H (8 bytes of data) are received from an external device via the COM port of the CPU.

The system register settings for this example are as follows:

- System register 417: 200
- System register 418: 4



Reception buffer when reception is completed

When reception of data from an external device has been completed, the "reception done (see page 312) flag" (in this example R9038) goes on and further reception of data is not allowed.

To continue receiving data please refer to Clearing the Reception Buffer (see page 331).

For repeated reception of data, refer to the following procedure.

1. Receive data
2. Reception completed ("reception done flag" is on, Reception: not allowed)
3. Process received data
4. Execute F159_MTRN instruction ("reception done flag" is off, Reception: enable)
5. Receive further data

Setting the Reception Buffer for the CPU COM Port

NOTE

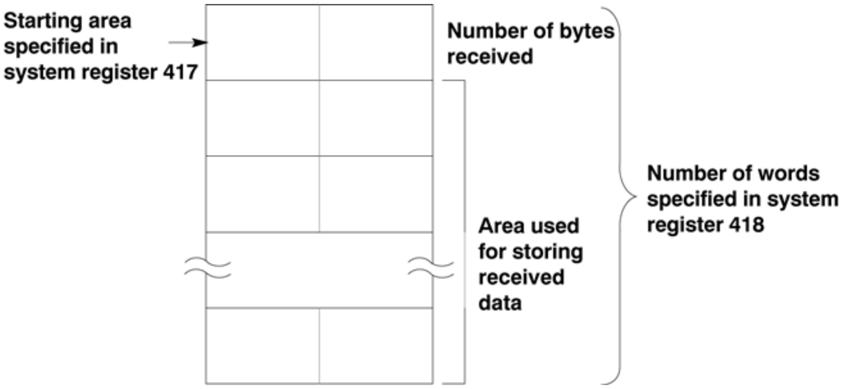
The numbers of the system registers for the respective settings may vary according to the PLC type used.

PROCEDURE

1. **Setting the reception buffer for the CPU's COM port**

   All areas of the data register are initially set for use as the reception buffer. To change the reception buffer, set the starting area number in system register 417 and the size (number of words: max. 1000) in system register 418.

   The reception buffer will be as follows:



2. **Setting the reception buffer for FP-Sigma's COM ports:**

   COM1 port: 416, 417

   COM2 port: 418, 419

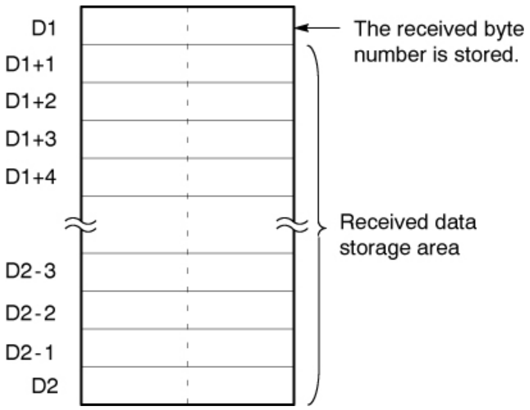
F161_MRCV Read Serial Data from the MCU's COM Port

Steps: 7

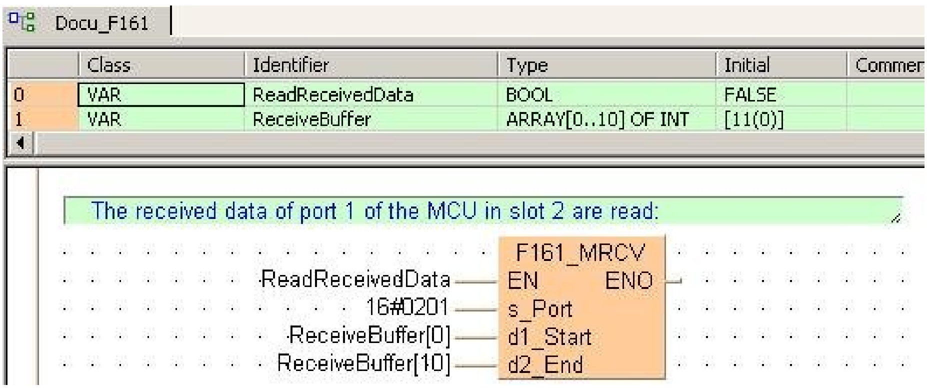
Description The received data from external equipment is copied to the receive buffer.

The number of bytes received is stored in the initial address specified by **d1_start** of the receive buffer. If the data received exceeds the ending address specified by **d2_end**, an operation error is detected. The data which has been received up to the d**2_end** will be stored. It (see page 331) also clears the receive buffer, resets the "reception done flag" and allows further reception of data.

Data table (receive buffer)



Example:



Data types

Variable	Data type	Function
s_Port	Constant	Specification of the slot number (high byte) and port number (low byte) of the MCU unit to which the data is transmitted. 16#xx01: COM1 of the MCU module in slot 16#xx 16#xx02: COM2 of the MCU module in slot 16#xx
d1_Start	ARRAY of INT	Initial address of the receive buffer in which the received data is stored.
d2_End	ARRAY of INT	Ending address of the receive buffer in which the received data is stored.

Operands

For	Relay				T/C		Register			Constant
s_Port	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d2_End	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the specified address using the index modifier exceeds a limit. - the MCU unit does not exist at the slot no. specified by 's_Port'. - the communication port specified by 's_Port' does not exist.
R9008	%MX0.900.8	for an instant	

Clearing the Reception Buffer

■ COM Port of the CPU

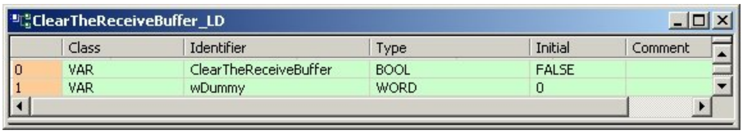
When F159_MTRN (see page 324) is executed, the "reception done (see page 312) flag" is turned off, the received byte number is reset to zero and the buffer can receive new data.

To clear the reception buffer without sending new data, you can execu (see page 324)te F159_MTRN with the number of bytes set to zero.

☞ **You can only execute F159_MTRN with the number of bytes equal to zero for the COM ports of a CPU; otherwise an operation error will occur.**

POU Header

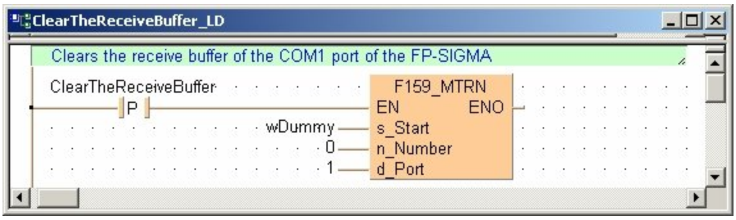
In the POU header, all input and output variables are declared that are used for programming this function.



ClearTheReceiveBuffer_LD

	Class	Identifier	Type	Initial	Comment
0	VAR	ClearTheReceiveBuffer	BOOL	FALSE	
1	VAR	wDummy	WORD	0	

LD Body



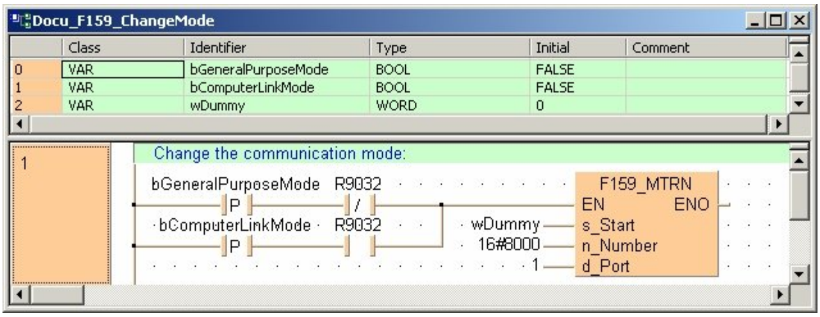
ST Body

```
if (DF(ClearTheReceiveBuffer)) then
      (* Clears the receive buffer of the COM1 port of the
FP-SIGMA *)
      F159_MTRN(s_Start := wDummy, n_Number := 0, d_Port := 1);
end_if;
```

Example In this example the function F159_MWRT is programmed in ladder diagram (LD) and structured text (ST) to toggle between program controlled and MEWTOCOL-COM Slave mode.

POU Header and LD Body

In this progam, the communication mode is changed by the program in RUN mode, depending on whether the communication mode flag R9032 is set or not.



The default vaule of the variable **n_Number** is set to 16#8000. This value changes during the execution of F159_MTRN the settings of the COM port from "Program Controlled [General Purpose]" to "MEWTOCOL-COM Slave [Computer Link]" and vice versa.

ST

```
IF (DF(bGeneralPurposeMode) AND NOT R9032 OR
    DF(bComputerLinkMode) AND R9032)
THEN
    (* Change the communication mode *)
    F159_MTRN( s_Start:= wdummy, n_Number:= 16#8000,
d_port:= 1);
END_IF;
```

Setting the Parameter 'Port'

Data transmission/reception is possible via the following ports of the PLC:

- COM port(s) of the CPU (the FP-Sigma has two COM ports)
- COM ports of the MCU

For all communication instructions (F159_MTRN (see page 324), F161_MRCV (see page 330), IsTransmissionDone (see page 311), IsReceptionDone (see page 312)) in program controlled mode, these ports can be specified using the parameter '**Port'** with the following settings:

PLC Type	'Port'	Explanation
FP-Sigma	1	First COM port on the CPU
	2	Second COM port on the CPU

15.3.1.6 Data Transfer via Modbus RTU Master/Slave Mode (FP-X)

In This Section:

Part III F/P Instructions

F145_MODBUS_WRITE_DATA

Write Data in MODBUS RTU Master/Slave Mode

Steps:

Description

Use this instruction to write data to a slave from a master via the serial port (COM1 or COM2). Both master and slave must be configured in Modbus RTU master/slave mode (see page 308). The slave will automatically be enabled to handle Modbus commands 05, 06, 15 and 16, i.e. you do not need to configure the slave.

The data specfied by **s2_MasterStartAddr** for the master is written to the slave area specified by **d_SlaveStartAddrType** and **d_SlaveStartAddrOffs**. The 2 words in **s1_ControlData** determine whether words or bits are sent to the slave, the slave's unit number and the slave's COM port (1 or 2).

PLC types: **Availability of F145_MODBUS_WRITE_DATA** **(see page 928)**

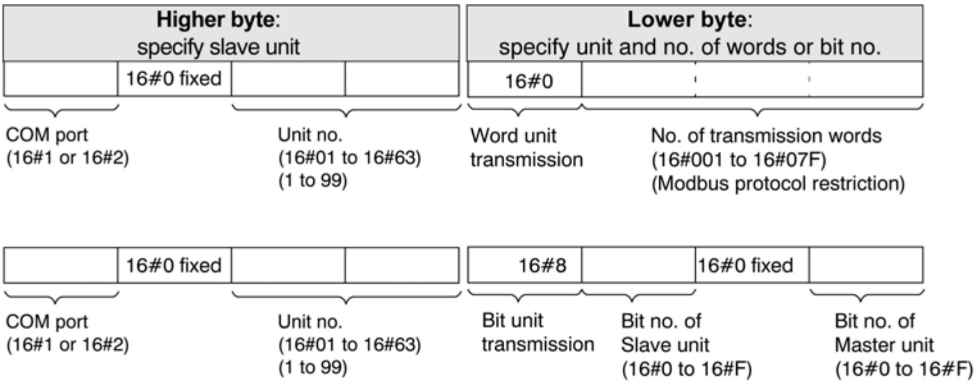
Data types

Variable	Data type	Function
s1_ControlData	DWORD	Stores control data.
s2_MasterStartAddr	ANY	Starting master address that stores the data to be written to the slave.
d_SlaveStartAddrType	ANY16	Address type in the slave to which data is written, e.g. DT, Y, R, WY, etc. The address must be fixed at 0.
d_SlaveStartAddrOffs	ANY16	The offset for the starting slave address whose type is defined by d_SlaveStartAddrType and to which the data is written.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2	WX	WY	WR	WL	-	-	DT	LD	FL	
d	-	WY	WR	-	-	-	DT	-	-	-
d	-	-	-	-	-	-	-	-	-	dec or hex

The control data is specified by s1_ControlData as follows:



1. Specify the transmission unit and transmission method with the lower byte of **s1_ControlData**.
   If data is to be sent in word units, specify the data volume; if it is to be sent in bit units, specify the position of the target bit. (A maximum of 127 (16#7F) word units can be sent because the transmission range allows up to 254 bytes.)
2. Specify the slave unit with the higher byte of **s1_ControlData**.
   Specify the unit number of the slave unit. 16#00 specifies a global transmission (no response). Specify either COM1 or COM2. 16#0 is fixed for the route no.
3. Specify the memory area of the master unit with **s2_MasterStartAddr** in which the data to be sent is stored.
4. Specify the memory area of the slave with **d_SlaveStartAddrType** and **d_SlaveStartAddrOffs** in combination. Specify 0 for the device no. of **d_SlaveStartAddrType**. For example: when **d_SlaveStartAddrType:** DT0 and **d_SlaveStartAddrOffs:** 100 → DT100.

Modbus command

The Modbus command is created according to the operands specified by **s1_ControlData**, **s2_MasterStartAddr** and **d_SlaveStartAddrType**. The following Modbus commands are used: 05 (see page 337) (to write one bit to Y, R), 06 (see page 338) (to write one word to DT), 15 (see page 339) (to write multiple bits to Y, R) and 16 (see page 341) (to write multiple words to DT). When the transmission is executed, 2 bytes of CRC are added to the end after the Modbus command has been created.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- The control data of **s1_ControlData** is a value outside of the specified range. - The number of words specified by **s1_ControlData** causes the area of **s2_MasterStartAddr** or **d_SlaveStartAddrType** to be exceeded when word unit transmission is being used. - **d_SlaveStartAddrType + d_SlaveStartAddrOffs** exceeds the memory type area available. - The Modbus mode has not been specified for the COM port of the control data specified by the higher byte of **s1_ControlData**. - The area of **d_SlaveStartAddrType** is DT in bit unit transmission. - The device no. of **d_SlaveStartAddrType** is not 0.
R9008	%MX0.900.8	permanently	

Precautions during prog.

- It is not possible to execute multiple F145_MODBUS_WRITE_DATA and F146_MODBUS_READ_DATA instructions for the same communication port simultaneously. The program should be set up so that these instructions are executed when the SEND/RECV execution enabled flag (R9044: COM1/R904A: COM2) is ON.

  R9044: COM1 0: Execution inhibited (SEND/RECV instruction being executed)
  R904A: COM2 1: Execution enabled

- The SEND (i.e. F145_MODBUS_WRITE_DATA) instruction only requests that data be sent, but the actual processing takes place when the ED instruction is executed. The SEND/RECV execution end flag (R9045: COM1/R904B: COM2) can be used to check whether or not the transmission has been completed.

  R9045 (COM1) 0: Completed normally
  1: Completed with error (The error code is stored in DT90045.)
  DT90124 (COM1) If the transmission has been completed with an error (R9045 is ON), the contents of the error (error code) are stored.
  R904B (COM2) 0: Completed normally
  1: Completed with error (The error code is stored in DT90125.)
  DT90125 (COM2) If the transmission has been completed with an error (R904B is ON), the contents of the error (error code) are stored.

- For information on the contents of error codes, refer to the FP-X User's Manual or Control FPWIN Pro documentation. If the error code is 16#73, a communication time-out error has occurred. The time-out length can be set from 10.0 ms to 81.9 seconds (in units of 10 ms) using system register 32. The default value is 10 seconds.

Error code	Description
16#73	Time-out: waiting for response

- For global transmission (the transmission performed by specifying 16#00 for the unit no.), the program should be set up so that the transmission is executed after the maximum scan time has elapsed.
- The F145 or F146 instruction cannot be executed if the target address is a special internal relay (from R9000) or a special data register (from DT90000).

Command for Function Code 05

Write Single Bit to Y or R

Description For example, this command transmits a single bit to a specified bit of the slave unit via COM1.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	0	7	8	1	0 fixed	0
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Bit unit transmission	Bit No. of Slave (16#0 to 16#F)		Bit No. of Master (16#0 to 16#F)

- To generate function code 05, bit unit transmission (16#8) must be specified.

s1_ControlData:	16#10078100
s2_MasterStartAddr:	16#0000
d_SlaveStartAddrType:	WY0
d_SlaveStartAddrOffs:	1

Command conversion

Modbus command

1	Slave address	07
2	Function code (16#05)	05
3	Coil No. (H)	00
4	Coil No. (L)	11
5	Setting status (H)	FF
6	Setting status (L)	00
7	CRC16 (H)	DC
8	CRC16 (L)	59

- After the ON or OFF value of bit 0 of s2_MasterStartAddr has been read in the master, this value is set in the slave (ON=FF00, OFF=0000).

Example In this example the function F145 is programmed in ladder diagram (LD).

POU Header

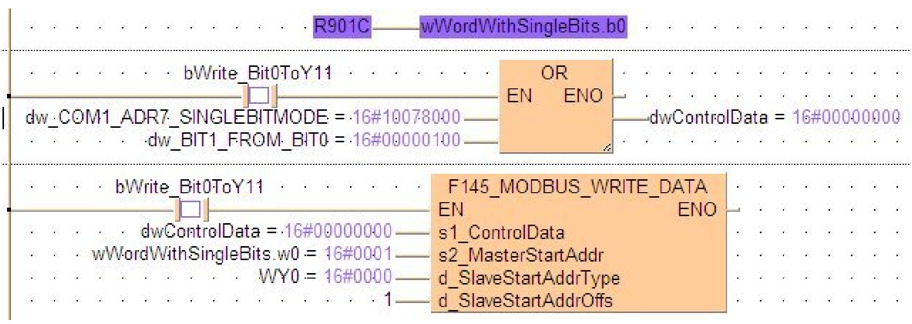
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bWrite_Bit0ToY11	BOOL	FALSE	
1	VAR	wWordWithSingleBits	BOOL16_OVERLAPPING_DUT		
2	VAR	dwControlData	DWORD		
3	VAR_CONSTANT	dw_COM1_ADR7_SINGLEBITMODE	DWORD	16#10078000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_BIT1_FROM_BIT0	DWORD	16#0100	Digit 2: 1=Bit 1 on Remote PLC,

Body

The bit b0 of **wWordWithSingleBits** is written to Y11 (bit no. 1 of word WY1) of slave unit no. 7 (16#7) via COM1.

LD

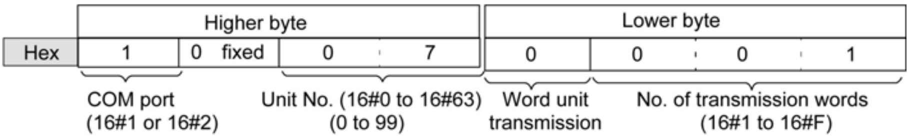


Command for Function Code 06

Write Single Word to DT

Description For example, the command writes a single word to the specified data register of the remote unit via COM1.

s1_ControlData



- To generate function code 06, bit unit transmission (16#8) must be specified and the number of transmission words must be set to 16#1.

s1_ControlData:	16#10070001
s2_MasterStartAddr:	16#1234
d_SlaveStartAddrType:	DT0
d_SlaveStartAddrOffs:	1000

Command conversion

Modbus command

1	Slave address	07
2	Function code (16#06)	06
3	Starting No. (H)	03
4	Starting No. (L)	E8
5	Write data (H)	12
6	Write data (L)	34
7	CRC16 (H)	04
8	CRC16 (L)	AB

- The word data of s2_MasterStartAddr are read and written to DT1000 (16#03E8) in the slave

Example In this example the function F145 is programmed in ladder diagram (LD).

POU Header

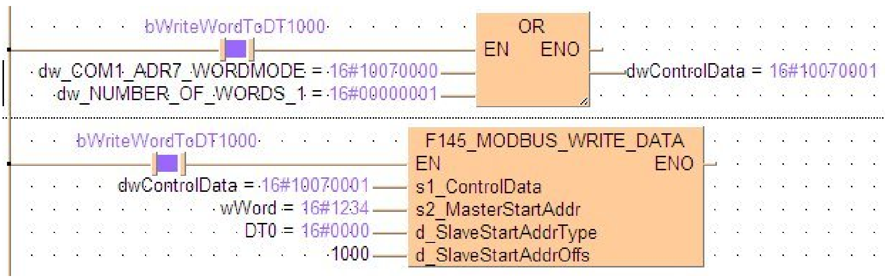
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bWriteWordToDT1000	BOOL	FALSE	
1	VAR	wWord	WORD	16#1234	
2	VAR	dwControlData	DWORD	0	
3	VAR_CONSTANT	dw_COM1_ADR7_WORDMODE	DWORD	16#10070000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_1	DWORD	1	

Body

The value of **wWord** is written to DT1000 of slave unit no. 7 (16#7) via COM1.

LD



Command for Function Code 15

Write Multiple Bits to Y or R

Description For example, the command writes 64 bit values to the specified data area of the slave unit no. 7 via COM1.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	0	7	0	0	0	4
	COM port (16#1 or 16#2)		Unit No. (16#0 to 16#63) (0 to 99)		Word unit transmission	No. of transmission words (16#1 to 16#7F)		

- To generate function code 15, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10070004
s2_MasterStartAddr[0]:	16#3210
s2_MasterStartAddr[1]:	16#7654
s2_MasterStartAddr[2]:	16#BA98
s2_MasterStartAddr[3]:	16#FEDC
d_SlaveStartAddrType:	WY0
d_SlaveStartAddrOffs:	0

Command conversion

Modbus command

1	Slave address	07
2	Function code (16#0F)	0F
3	Starting No. (H)	00
4	Starting No. (L)	00
5	No. of coils to change (H)	00
6	No. of coils to change (L)	40
7	No. of data (No. of bytes)	08
8	Setting data 1	10
9	Setting data 2	32
10	Setting data 3	54
11	Setting data 4	76
12	Setting data 5	98
13	Setting data 6	BA
14	Setting data 7	DC
15	Setting data 8	FE
16	CRC16 (H)	6C
17	CRC16 (L)	B3

- Starting No. is the first coil to be set in the slave.
- No. of coils to change is a hex number (here: 16#0040 = 64). Max. number = 2032 (16#07F0) (Modbus protocol restriction)
- No. of data is the numer of bytes to be written. 8 coils are regarded as 1 byte. Max. No. = 254 (16#FE)

Example In this example the function F145 is programmed in ladder diagram (LD).

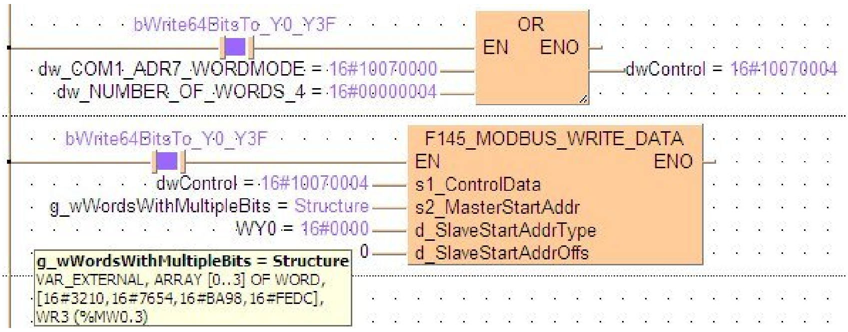
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	g_wWordsWithMultipleBits	ARRAY [0..3] OF WORD	[16#3210,16#7654,16#BA98,16#FEDC]	
1	VAR	bWrite64BitsTo_Y0_Y3F	BOOL	FALSE	
2	VAR	dwControl	DWORD	0	
3	VAR_CONSTANT	dw_COM1_ADR7_WORDMODE	DWORD	16#10070000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_4	DWORD	4	

Body The 64 bit values of **g_wWordsWithMultipleBits** are written to Y0-Y3F (i.e. beginning at bit 0 of word WY0) of slave unit no. 7 (16#7) via COM1.

Part III F/P Instructions

LD

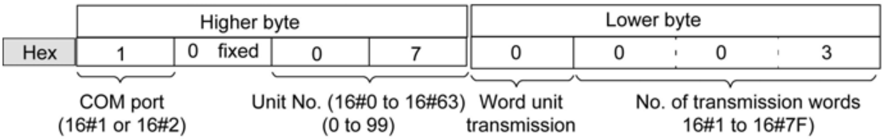

bWrite64BitsTo_Y0_Y3F
OR
EN ENO
dw_COM1_ADR7_WORDMODE = 16#10070000
dw_NUMBER_OF_WORDS_4 = 16#00000004
dwControl = 16#10070004
bWrite64BitsTo_Y0_Y3F
F145_MODBUS_WRITE_DATA
EN ENO
dwControl = 16#10070004
s1_ControlData
g_wWordsWithMultipleBits = Structure
s2_MasterStartAddr
WY0 = 16#0000
d_SlaveStartAddrType
0
d_SlaveStartAddrOffs
g_wWordsWithMultipleBits = Structure
VAR_EXTERNAL, ARRAY [0..3] OF WORD,
[16#3210,16#7654,16#BA98,16#FEDC],
WR3 (%MW0.3)

Command for Function Code 16

Write Multiple Words to DT

Description For example, the command transmits 3 words to the specified data area of slave unit no. 7 via COM1.

s1_ControlData



- To generate function code 16, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10070003
s2_MasterStartAddr[0]:	16#0011
s2_MasterStartAddr[1]:	16#2233
s2_MasterStartAddr[2]:	16#4455
d_SlaveStartAddrType:	DT0
d_SlaveStartAddrOffs:	500

Command conversion

Modbus command

1	Slave adress	07
2	Function code (16#10)	10
3	Starting No. (H)	01
4	Starting No. (L)	F4
5	No. of registers to write (H)	00
6	No. of registers to write (L)	03
7	No. of data (No. of bytes)	06
8	Write data 1 (H)	00
9	Write data 1 (L)	11
10	Write data 2 (H)	22
11	Write data 2 (L)	33
12	Write data 3 (H)	44
13	Write data 3 (L)	55
14	CRC16 (H)	5A
15	CRC16(L)	E7

- Max. No. of registers to write = 127 (16#7F) (Modbus protocol restriction)
- No. of data is the numer of transmission bytes. 2 DT are regarded as 1 byte. Max. No. = 254 (16#FE)

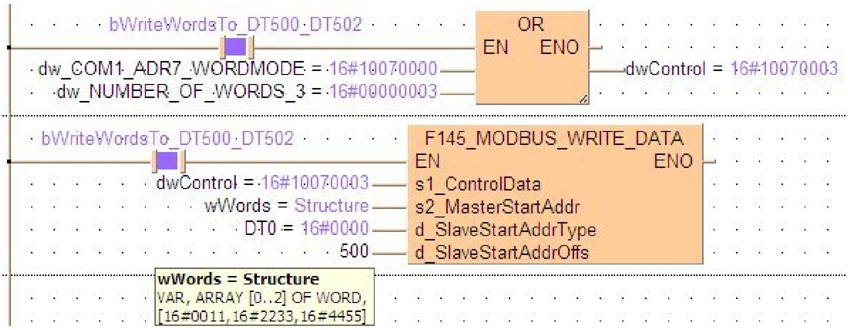
Example In this example the function F145 is programmed in ladder diagram (LD).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	bWriteWordsTo_DT500_DT502	BOOL	FALSE
1	VAR	wWords	ARRAY [0..2] OF WORD	[16#0011,16#2233,16#4455]
2	VAR	dwControl	DWORD	0
3	VAR_CONSTANT	dw_COM1_ADR7_WORDMODE	DWORD	16#10070000
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_3	DWORD	3

Body The 3 values of **wWords** are written to DT500-DT502 of the slave unit no. 7 (16#7) via COM1.

LD


bWriteWordsTo_DT500_DT502
OR
EN ENO
dw_COM1_ADR7_WORDMODE = 16#10070000
dw_NUMBER_OF_WORDS_3 = 16#00000003
dwControl = 16#10070003
bWriteWordsTo_DT500_DT502
F145_MODBUS_WRITE_DATA
EN ENO
dwControl = 16#10070003
s1_ControlData
wWords = Structure
s2_MasterStartAddr
DT0 = 16#0000
d_SlaveStartAddrType
500
d_SlaveStartAddrOffs
wWords = Structure
VAR, ARRAY [0..2] OF WORD,
[16#0011,16#2233,16#4455]

F146_MODBUS_ READ_DATA

Read Data in MODBUS RTU Master/Slave Mode

Steps:

Description

Use this instruction for a master to request data from a slave via the serial port (COM1 or COM2). Both master and slave must be configured in Modbus RTU master/slave mode (see page 308). The slave will automatically be enabled to handle Modbus commands 01, 02, 03 and 04, i.e. you do not need to configure the slave.

The data is read from the memory area of the slave specified by **s2_SlaveStartAddrType** and **s2_SlaveStartAddrOffs**. It is stored in the area of the master specified by **d_MasterStartAddr**. The 2 words in **s1_ControlData** determine whether words or bits are read from the slave, the slave's unit number and the slave's COM port (1 or 2).

PLC types: **Availability of F146_MODBUS_READ_DATA (see page 928)**

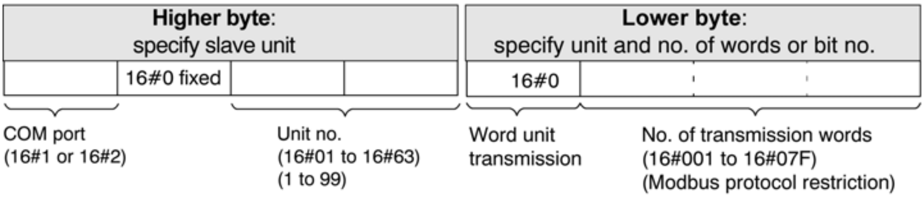
Data types

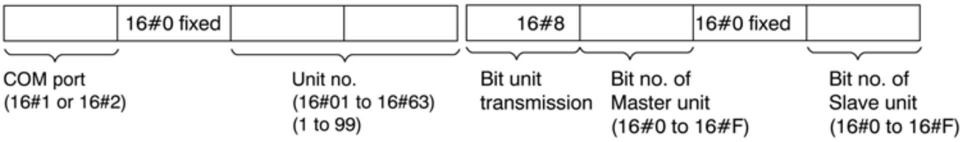
Variable	Data type	Function
s1_ControlData	DWORD	Stores control data.
s2_SlaveStartAddrType	ANY16	Address type in the slave from which data is read, e.g. DT, Y, R, WY, etc. The address must be fixed at 0.
s2_SlaveStartAddrOffs	ANY16	The offset for the starting slave address whose type is defined by s2_SlaveStartAddrType and from which the data is read.
d_MasterStartAddr	ANY	Starting address in the master into which the data read from the slave is stored.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2	WX	WY	WR	WL	-	-	DT	LD	FL	-
s2	-	-	-	-	-	-	-	-	-	dec or hex
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

The control data is specified by s1_ControlData as follows:





1. Specify the transmission unit and transmission method with the lower byte of **s1_ControlData**.
   If data is to be sent in word units, specify the data volume, and if it is to be sent in bit units, specify the position of the target bit. (A maximum of 127 (16#7F) word units can be read because the transmission range allows up to 254 bytes.)
2. Specify the slave unit with the higher byte of **s1_ControlData**.
   Specify the unit number of the slave unit. 16#00 specifies a global transmission (no response). Specify either COM1 or COM2. 16#0 is fixed for the route no.
3. Specify the memory area of the slave unit to be read with **s2_SlaveStartAddrType** and **s2_SlaveStartAddrOffs** in combination. Specify 0 for the device no. of **s2_SlaveStartAddrType**. For example, when **s2_SlaveStartAddrType:** DT0 and **s2_SlaveStartAddrOffs**: 100 → DT100.
4. Specify the area of the master unit with **d_MasterStartAddr** into which the data read is to be stored.

Modbus command

- The Modbus command is created according to the operands specified by **s1_ControlData**, **s2_SlaveStartAddrType**, and **d_MasterStartAddr**. The following Modbus commands are used: 01 (see page 347) (Y, R coil read), 02 (see page 349) (WL, LD read, or X contact read), 03 (see page 351) (DT read) and 04 (WL (see page 353), LD (see page 352) read). When the transmission is executed, 2 bytes of CRC are added to the end after the Modbus command has been created.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- The control data of **s1_ControlData** is a value outside of the specified range. - The number of words specified by **s1_ControlData** causes the area of **s2_SlaveStartAddrType** or **d_MasterStartAddr** to be exceeded when word unit transmission is being used. - **s2_SlaveStartAddrType + s2_SlaveStartAddrOffs** exceeds the memory type area available. - The Modbus mode has not been specified for the COM port of the control data specified by the higher byte of **s1_ControlData**. - The area of **s2_SlaveStartAddrType** is DT, WL and LD in the bit unit transmission. - The device no. of **s2_SlaveStartAddrType** is not 0.
R9008	%MX0.900.8	permanently	

Precautions during prog.

- It is not possible to execute multiple F145_MODBUS_WRITE_DATA and F146_MODBUS_READ_DATA instructions for the same communication port simultaneously. The program should be set up so that these instructions are executed when the SEND/RECV execution enabled flag (R9044: COM1/R904A: COM2) is ON.

  R9044: COM1 0: Execution inhibited (SEND/RECV instruction being executed)
  R904A: COM2 1: Execution enabled

- The SEND (i.e. F145_MODBUS_WRITE_DATA) instruction only requests that data be sent, but the actual processing takes place when the ED instruction is executed. The SEND/RECV execution end flag (R9045: COM1/R904B: COM2) can be used to check whether or not the transmission has been completed.

  R9045 (COM1) 0: Completed normally
  1: Completed with error (The error code is stored in DT90045.)
  DT90124 (COM1) If the transmission has been completed with an error (R9045 is ON), the contents of the error (error code) are stored.
  R904B (COM2) 0: Completed normally
  1: Completed with error (The error code is stored in DT90125.)
  DT90125 (COM2) If the transmission has been completed with an error (R904B is ON), the contents of the error (error code) are stored.

- For information on the contents of error codes, FP-X User's Manual or Control FPWIN Pro documentation. If the error code is 16#73, a communication time-out error has occurred. The time-out length can be set from 10.0 ms to 81.9 seconds (in units of 10 ms) using system register 32. The default value is 10 seconds.

Error code	Description
16#73	Time-out: waiting for response

- The F145 or F146 instruction cannot be executed if the target address is a special internal relay (from R9000) or a special data register (from DT90000).

Command for Function Code 01

Read Single Bit from R or Y

Description For example, when bit Y17 (bit no. 7 of word WY1) is read from the slave unit no. 17 (16#11) via COM1, the value is transmitted to the 5th bit of the variable at input **d_MasterStartAddr**.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	8	5	0	7
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Bit unit transmission	Bit No. of Master (16#0 to 16#F)		Bit No. of Slave (16#0 to 16#F)

- To generate function code 01, bit unit transmission (16#8) must be specified.

s1_ControlData:	16#10118507
s2_SlaveStartAddrType:	WY0
s2_SlaveStartAddrOffs:	1
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#01)	01
3	Starting No. (H)	00
4	Starting No. (L)	17
5	No. of coils to read (H)	00
6	No. of coils to read (L)	01
7	CRC16 (H)	DC
8	CRC16 (L)	59

- Starting No. is the first coil to read in the slave (here: Y17)
- The No. of coils to read must be 1

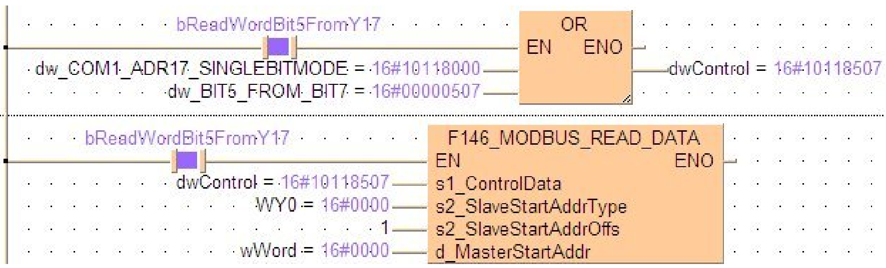
Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bReadWordBit5FromY17	BOOL	FALSE	
1	VAR	wWord	WORD	0	
2	VAR	dwControl	DWORD		
3	VAR	bExampleBit	BOOL	FALSE	
4	VAR_CONSTANT	dw_COM1_ADR17_SINGLEBITMODE	DWORD	16#10118000	Digit 7: 1=COM1,
5	VAR_CONSTANT	dw_BIT5_FROM_BIT7	DWORD	16#0507	Digit 2: 1=Bit 5 on Local PLC,

Body The bit Y17 (i.e. bit 7 of word WY1) of the slave unit at address 17 (16#11) is read and then transmitted to bit 5 of **wWord** of the master via COM1.

LD



Command for Function Code 01_x

Read Multiple Bits from R or Y

Description For example, when 64 bits (4 words) from Y10-Y4F of slave unit no.17 are read via COM1, the value is transmitted to the variable at input **d_MasterStartAddr**.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	0	0	0	4
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Word unit transmission	No. of transmission words (16#001 to 16#07F)		

- To generate function code 01_x, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10110004
s2_SlaveStartAddrType:	WY0
s2_SlaveStartAddrOffs:	1
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#01)	01
3	Starting No. (H)	00
4	Starting No. (L)	10
5	No. of data to read (H)	00
6	No. of data to read (L)	40
7	CRC16 (H)	3E
8	CRC16 (L)	AF

- Starting No. is the first coil to read in the slave (here: Y10).
- No. of data to read is the number of transmission words (here: 4 words = 64 bits = 16#0040)

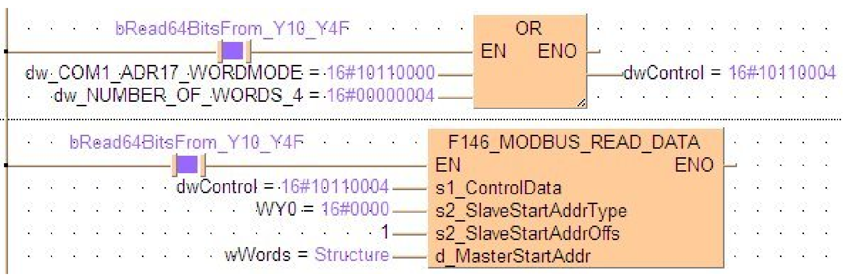
Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bRead64BitsFrom_Y10_Y4F	BOOL	FALSE	
1	VAR	wWords	ARRAY [0..3] OF WORD	[4(0)]	
2	VAR	dwControl	DWORD		
3	VAR	bExampleBit	BOOL	FALSE	
4	VAR_CONSTANT	dw_COM1_ADR17_WORDMODE	DWORD	16#10110000	Digit 7: 1=COM1,
5	VAR_CONSTANT	dw_NUMBER_OF_WORDS_4	DWORD	4	

Body The 64 bit values of Y10-Y4F (4 words each with 16 bits beginning at WY1) of the slave unit no. 17 (16#11) are read and transmitted to **wWords** of the master unit via COM1.

LD



Command for Function Code 02

Read Single Bit from X

Description For example, when X17 (bit no. 7 of word WX1) is read from slave unit no. 17 (16#11) via COM1, the value is transmitted to the 5th bit of the variable at input **d_MasterStartAddr**.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	8	5	0	7
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Bit unit transmission	Bit No. of Master (16#0 to 16#F)		Bit No. of Slave (16#0 to 16#F)

- To generate function code 02, bit unit transmission (16#8) must be specified.

s1_ControlData:	16#10118507
s2_SlaveStartAddrType:	WX0
s2_SlaveStartAddrOffs:	1
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#02)	02
3	Starting No. (H)	00
4	Starting No. (L)	17
5	No. of data to read (H)	00
6	No. of data to read (L)	01
7	CRC16 (H)	0B
8	CRC16 (L)	5E

- Starting No. is the first coil to be read in the slave (here: X17)
- No. of data to read must be 1.

Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header

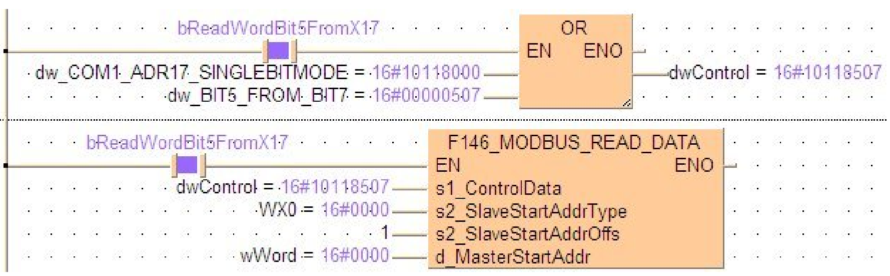
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bReadWordBit5FromX17	BOOL	FALSE	
1	VAR	wWord	WORD	0	
2	VAR	dwControl	DWORD		
3	VAR	bExampleBit	BOOL	FALSE	
4	VAR_CONSTANT	dw_COM1_ADR17_SINGLEBITMODE	DWORD	16#10118000	Digit 7: 1=COM1,
5	VAR_CONSTANT	dw_BIT5_FROM_BIT7	DWORD	16#0507	Digit 2: 1=Bit 5 on Local PLC,

Body

The bit X17 (i.e. bit 7 or word WX1) of slave unit no. 17 (16#11) is read, the value of which is transmitted to bit 5 of **wWord** of the master unit via COM1.

LD



Command for Function Code 02_x

Read Multiple Bits from X

Description For example, when the 64 bits (4 words) from X10 to X4F are read from slave unit no.17 via COM1, and stored in the variable at input **d_MasterStartAddr**.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	0	0	0	4
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Word unit transmission	No. of transmission words (16#001 to 16#07F)		

- To generate function code 02_x, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10110004
s2_SlaveStartAddrType:	WX0
s2_SlaveStartAddrOffs:	1
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#02)	02
3	Starting No. (H)	00
4	Starting No. (L)	10
5	No. of data to read (H)	00
6	No. of data to read (L)	40
7	CRC16 (H)	7A
8	CRC16 (L)	A0

- Starting No. is the first coil to be read in the slave (here: X17)
- No. of data to read is a hex number (here: 16#0040 = 64). Max. number = 2032 (16#07F0) (Modbus protocol restriction)
- No. of data to read is the number of words to read (here: 4 words = 64 bits = 16#0040)

Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header

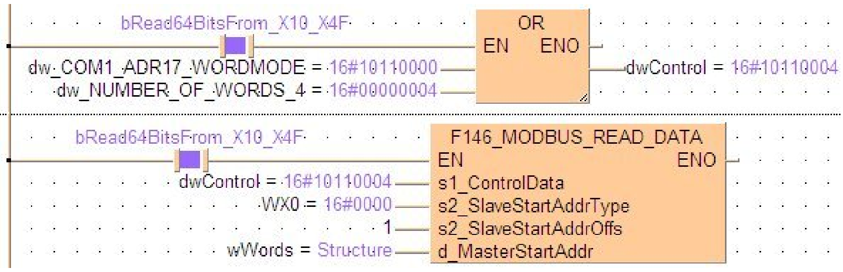
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bRead64BitsFrom_X10_X4F	BOOL	FALSE	
1	VAR	wWords	ARRAY [0..3] OF WORD	[4(0)]	
2	VAR	dwControl	DWORD		
3	VAR	bExampleBit	BOOL	FALSE	
4	VAR_CONSTANT	dw_COM1_ADR17_WORDMODE	DWORD	16#10110000	Digit 7: 1=COM1,
5	VAR_CONSTANT	dw_NUMBER_OF_WORDS_4	DWORD	4	

Body

The 64 bit values X10-X4F (4 words each of 16 bits starting at WX1) of slave unit no. 17 (16#11) are read and stored in **wWords** of the master unit via COM1.

LD



Command for Function Code 03

Read Multiple Words from DT

Description For example, when 6 words from DT500 to DT505 are read from slave unit no. 17, and then transmitted to the variable at input **d_MasterStartAddr** via COM1.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	0	0	0	6
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Word unit transmission	No. of transmission words (16#001 to 16#07F)		

- To generate function code 03, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10110006
s2_SlaveStartAddrType:	DT0
s2_SlaveStartAddrOffs:	500
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Command (16#03)	03
3	Starting No. (H)	01
4	Starting No. (L)	F4
5	No. of data to read (H)	00
6	No. of data to read (L)	06
7	CRC16 (H)	87
8	CRC16 (L)	56

- Starting No. is the first DT register to be read in the slave (here: DT500 = 16#01F4)
- No. of data to read is the number of transmission words (here: 6)

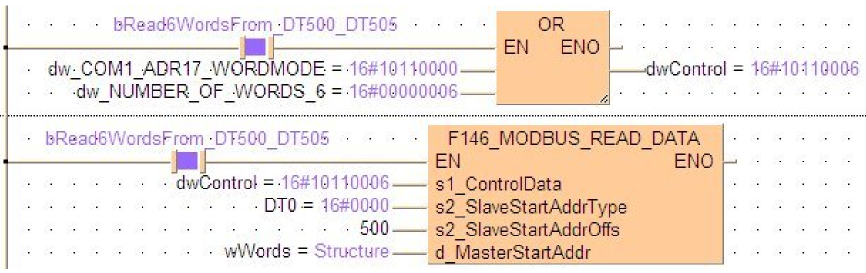
Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bRead6WordsFrom_DT500_DT505	BOOL	FALSE	
1	VAR	wWords	ARRAY [0..5] OF WORD	[6(0)]	
2	VAR	dwControl	DWORD		
3	VAR_CONSTANT	dw_COM1_ADR17_WORDMODE	DWORD	16#10110000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_6	DWORD	6	

Body The 6 word values of DT500-DT505 of the slave unit no. 17 (16#11) are read and transmitted to **wWords** of the master unit via COM1.

LD



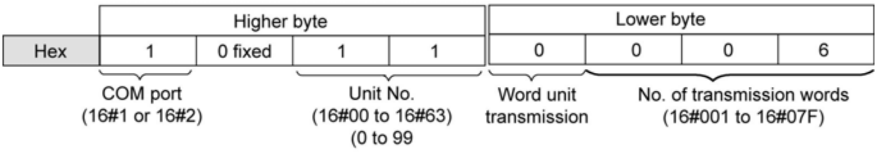
Part III F/P Instructions

Command for Function Code 04

Read Multiple Words from LD

Description For example, when 6 words from LD100 to LD105 are read from slave unit no. 17, and then transmitted to the variable at input **d_MasterStartAddr** via COM1.

s1_ControlData



- To generate function code 04, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10110006
s2_SlaveStartAddrType:	LD0
s2_SlaveStartAddrOffs:	100
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#04)	04
3	Starting No. (H)	08
4	Starting No. (L)	34
5	No. of data to read (H)	00
6	No. of data to read (L)	06
7	CRC16 (H)	32
8	CRC16 (L)	9C

- Starting No. ist the first LD register to be read in the slave. The reference no. for LD being 2000, this number will be added to the address internally. (here: LD100 + 2000 = LD2100 = 16#0834).
- No. of data to read is the number of transmission words (here: 6)

Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header

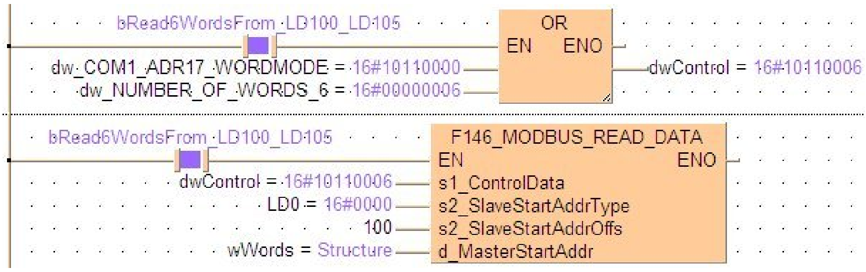
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bRead6WordsFrom_LD100_LD105	BOOL	FALSE	
1	VAR	wWords	ARRAY [0..5] OF WORD	[6(0)]	
2	VAR	dwControl	DWORD		
3	VAR_CONSTANT	dw_COM1_ADR17_WORDMODE	DWORD	16#10110000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_6	DWORD	6	

Body

The 6 words beginning at LD100 in slave unit no. 17 (16#11) are read and transmitted to **wWord** of the master unit via COM1.

LD



Part III F/P Instructions

Command for Function Code 04

Read Multiple Words from WL

Description For example, when 6 words fromWL20 to WL25 are read from slave unit no. 17, and then transmitted to the variable at input **d_MasterStartAddr** via COM1.

s1_ControlData

	Higher byte				Lower byte			
Hex	1	0 fixed	1	1	0	0	0	6
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Word unit transmission	No. of transmission words (16#001 to 16#07F)		

- To generate function code 04, word unit transmission (16#0) must be specified.

s1_ControlData:	16#10110006
s2_SlaveStartAddrType:	WL0
s2_SlaveStartAddrOffs:	20
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#04)	04
3	Starting No. (H)	00
4	Starting No. (L)	14
5	No. of data to read (H)	00
6	No. of data to read (L)	06
7	CRC16 (H)	32
8	CRC16 (L)	9C

- Starting No. ist the first WL register to be read in the slave (here: WL20 = 16#0014).
- No. of data to read is the number of transmission words (here: 6)

Example In this example the function F146 is programmed in ladder diagram (LD).

POU Header

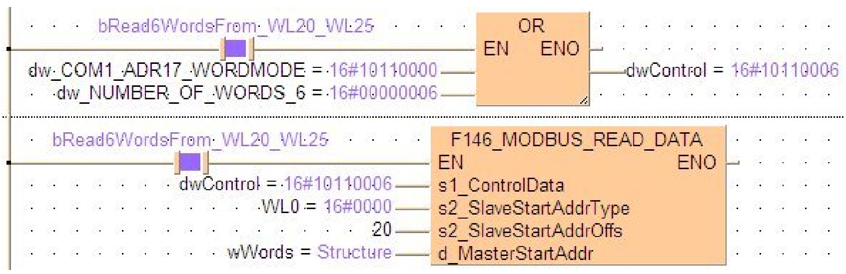
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	bRead6WordsFrom_WL20_WL25	BOOL	FALSE	
1	VAR	wWords	ARRAY [0..5] OF WORD	[6(0)]	
2	VAR	dwControl	DWORD		
3	VAR_CONSTANT	dw_COM1_ADR17_WORDMODE	DWORD	16#10110000	Digit 7: 1=COM1,
4	VAR_CONSTANT	dw_NUMBER_OF_WORDS_6	DWORD	6	

Body

The 6 words beginning at WL20 of slave unit no. 17 (16#11) are read and transmitted to **wWord** of the master unit via COM1.

LD



Part III F/P Instructions

Chapter 16

Arithmetic Instructions

F20_ADD **16-bit addition** Steps: 5

Description The 16-bit equivalent constant or 16-bit area specified by **s** and the 16-bit area specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	1011

+

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0001	0000

Result value 43 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0010	1011

When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F22_ADD2 (see page 360).

PLC types: **Availability of F20_ADD (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	addend
d	INT, WORD	augend and result

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (overflow or underflow).

Example

In this example the function F20_ADD is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

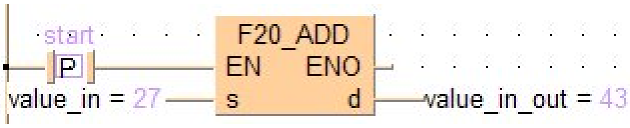
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be added to output_value
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: 43

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F20_ADD(value_in, value_in_out);
END_IF;
```

F21_DADD

32-bit addition

Steps: 7

Description The 32-bit equivalent constant or 32-bit area specified by **s** and the 32-bit data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 32-bit values are treated as double integer values.

Example value 1312896

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 0 0 0

32-bit area

Example value 558144

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0

Result value 1871040 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	1 1 0 0	1 1 0 0	0 0 0 0

☞ **When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F23_DADD2 (see page 362).**

PLC types: **Availability of F21_DADD (see page 925)**

Data types

Variable	Data type	Function
s	DINT, DWORD	addend
d	DINT, DWORD	augend and result

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (overflow or underflow).

Example

In this example the function F21_DADD is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

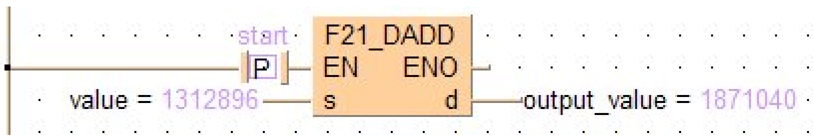
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	1312896	the value, that will b...
2	VAR	output_value	DINT	558144	result after a 0->1 le...

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F21_DADD(value, output_value);
END_IF;
```

F22_ADD2

16-bit addition, destination can be specified

Steps: 7

Description The 16-bit data or 16-bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

+

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

Result value 43 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 1 0	1 0 1 1

PLC types: **Availability of F22_ADD2 (see page 925)**

Data types

Variable	Data type	Function
s1	INT, WORD	augend
s2	INT, WORD	addend
d	INT, WORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (overflow or underflow).

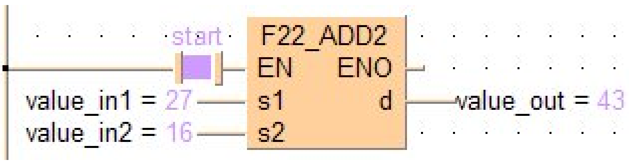
Example In this example the function F22_ADD2 is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	INT	27	
2	VAR	value_in2	INT	16	
3	VAR	value_out	INT	0	result after a 0->1 leading edge from start: 43

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F22_ADD2(value_in1, value_in2, value_out);
END_IF;
```

Part III F/P Instructions

F23_DADD2 — 32-bit addition, destination can be specified — Steps: 11

Description The 32-bit data or 32-bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The added result is stored in **d**. All 32-bit values are treated as double integer values.

Example value 1312896

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 0 0 0

32-bit area

Example value 558144

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0

Result value 1871040 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	1 1 0 0	1 1 0 0	0 0 0 0

PLC types: **Availability of F23_DADD2 (see page 925)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	augend
s2	DINT, DWORD	addend
d	DINT, DWORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (overflow or underflow).

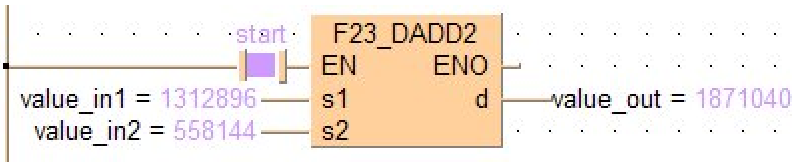
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	DINT	1312896	first summand
2	VAR	value_in2	DINT	558144	second summand
3	VAR	value_out	DINT	0	result after a 0->1 le...

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F23_DADD2(value_in1, value_in2, value_out);
END_IF;
```

F40_BADD 4-digit BCD addition

Steps: 5

Description The 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** and the 16-bit area for 4-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#2111 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 1
16# (BCD)	2	1	1	1

+

Example value 16#0011 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
16# (BCD)	0	0	1	1

Result value 16#2122 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 1 0	0 0 1 0
16# (BCD)	2	1	2	2

When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F41_DBADD (see page 366).

PLC types: **Availability of F40_BADD (see page 926)**

Data types

Variable	Data type	Function
s	WORD	addend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	augend and result, 16-bit area for 4-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (overflow).

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

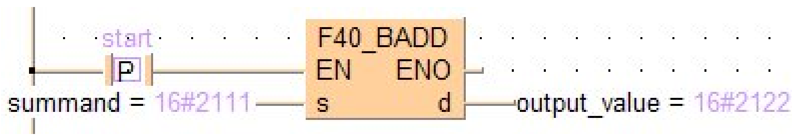
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	WORD	16#2111	this value will be added to the output_value
2	VAR	output_value	WORD	16#0011	result after 0->1 leading edge from start: 16#2122

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F40_BADD(summand, output_value);
END_IF;
```

F41_DBADD 8-digit BCD addition

Steps: 7

Description The 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** and the 8-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#12342000 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0
16# BCD	1	2	3	4	2	0	0	0

32-bit area

Example value 16#00003678 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	0	0	0	0	3	6	7	8

Result value 16#12345678 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	1	2	3	4	5	6	7	8

When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F43_DBADD2 (see page 370).

PLC types: **Availability of F41_DBADD (see page 926)**

Data types

Variable	Data type	Function
s	DWORD	addend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	augend and result, 32-bit area for 8-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (overflow).

Example

In this example the function F41_DBADD is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

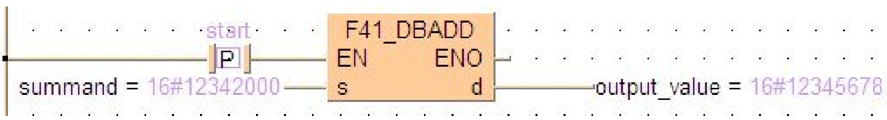
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	DWORD	16#12342000	this value will be added to the output_value
2	VAR	output_value	DWORD	16#00003678	result after 0->1 leading edge from start: 16#12345678

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F41_DBADD(summand, output_value);
END_IF;
```

F42_BADD2

4-digit BCD addition, destination can be specified

Steps: 7

Description The 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#4321 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# (BCD)	4	3	2	1

+

Example value 16#1234 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0
16# (BCD)	1	2	3	4

Result value 16#5555 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
16# (BCD)	5	5	5	5

PLC types: **Availability of F42_BADD2 (see page 926)**

Data types

Variable	Data type	Function
s1	WORD	augend, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	addend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	sum, 16-bit area for 4-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (overflow).

Example

In this example the function F42_BADD2 is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

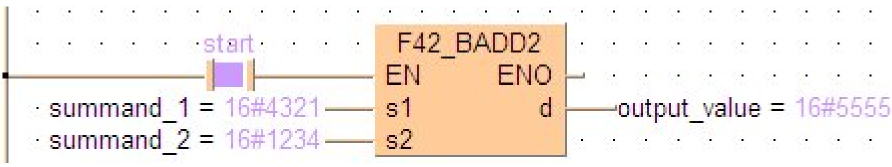
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	WORD	16#4321	first summand
2	VAR	summand_2	WORD	16#1234	second summand
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#5555

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F42_BADD2(summand_1, summand_2, output_value);
END_IF;
```

F43_DBADD2 — 8-digit BCD addition, destination can be specified

Steps: 11

Description The 8-digit BCD equivalent constant or 32-bit area for 8-digit BCD data specified by **s1** and **s2** are added together if the trigger EN is in the ON-state. The result is stored in **d**.

Example value 16#12345678 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	1	2	3	4	5	6	7	8

<— 32-bit area —>

Example value 16#87654321 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

Result value 16#99999999 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1
16# BCD	9	9	9	9	9	9	9	9

PLC types: **Availability of F43_DBADD2** **(see page 926)**

Data types

Variable	Data type	Function
s1	DWORD	augend, 32-bit area for 8-digit BCD data or equivalent constant
s2	DWORD	addend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	sum, 32-bit area for 8-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (overflow).

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

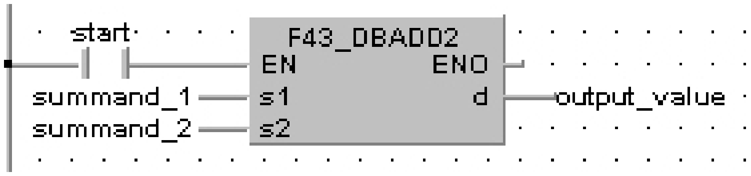
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	DWORD	16#12345678	first summand
2	VAR	summand_2	DWORD	16#87654321	second summand
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#99999999

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F43_DBADD2( summand_1, summand_2, output_value);
END_IF;
```

F35_INC

16-bit increment

Steps: 3

Description Adds "1" to the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 18 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0

PLC types: **Availability of F35_INC (see page 926)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be increased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (overflow).

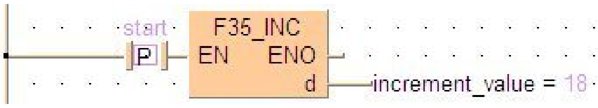
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	INT	17	result after a 0->1 leading edge from start: 18

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F35_INC(increment_value);
END_IF;
```

F36_DINC 32-bit increment

Steps: 3

Description Adds "1" to the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 131081

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 1

← 32-bit area →

Result value 131082 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0

PLC types: **Availability of F36_DINC (see page 926)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit area to be increased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (overflow).

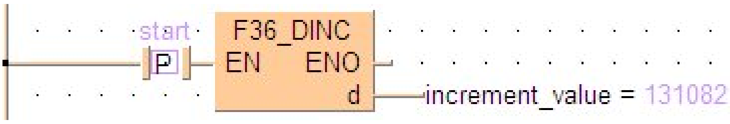
Example In this example the function F36_DINC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DINT	131081	result after a 0->1 leading edge from start: 131082

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F36_DINC(increment_value);
END_IF;
```

F55_BINC 4-digit BCD increment

Steps: 3

Description Adds "1" to the 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#4320 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 0
16# BCD	4	3	2	0

Result value 16#4321 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	4	3	2	1

PLC types: **Availability of F55_BINC (see page 926)**

Data types

Variable	Data type	Function
d	WORD	16-bit area for 4-digit BCD data to be increased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (overflow).

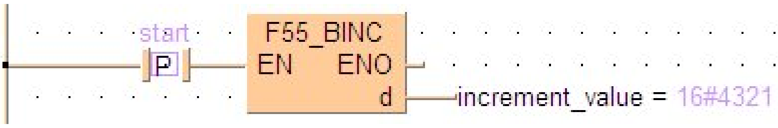
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	WORD	16#4320	result after a 0->1 leding edge from start: 16#4321

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F55_BINC(increment_value);
END_IF;
```

F56_DBINC 8-digit BCD increment

Steps: 3

Description Adds "1" to the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#87654320 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 0
16# BCD	8	7	6	5	4	3	2	0

32-bit area

Result value 16#87654321 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

PLC types: **Availability of F56_DBINC** **(see page 926)**

Data types

Variable	Data type	Function
d	DWORD	32-bit area for 8-digit BCD data to be increased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (overflow).

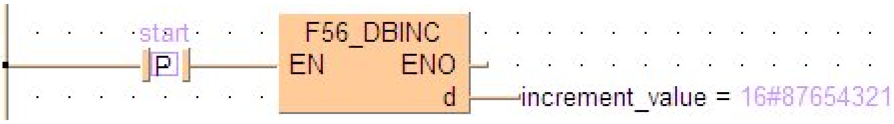
Example In this example the function F56_DBINC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DWORD	16#87654320	result after a 0->1 leding edge from start: 16#87654321

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F56_DBINC(increment_value);
END_IF;
```

F25_SUB 16-bit subtraction

Steps: 5

Description Subtracts the 16-bit equivalent constant or 16-bit area specified by **s** from the 16-bit area specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d** (minuend area). All 16-bit values are treated as integer values.

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

−

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

Result value -11 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 1 1 1	1 1 1 1	1 1 1 1	0 1 0 1

PLC types: **Availability of F25_SUB (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	subtrahend
d	INT, WORD	minuend and result

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (overflow or underflow).

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

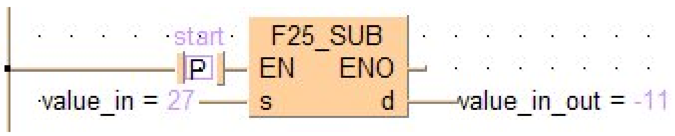
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be subtracted from value_in_out
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: -11

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F25_SUB(value_in, value_in_out);
END_IF;
```

F26_DSUB

32-bit subtraction

Steps: 7

Description Subtracts the 32-bit equivalent constant or 32-bit data specified by **s** from the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d** (minuend area). All 32-bit values are treated as double integer values.

Example value 16778109

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	0 1 1 1	1 1 0 1

32-bit area

Example value 524740

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	0 1 0 0

Result value 16253369 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	1 1 1 1	1 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1	1 0 0 1

PLC types: **Availability of F26_DSUB (see page 926)**

Data types

Variable	Data type	Function
s	DINT, DWORD	subtrahend
d	DINT, DWORD	minuend and result

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (overflow or underflow).

Example In this example the function F26_DSUB is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

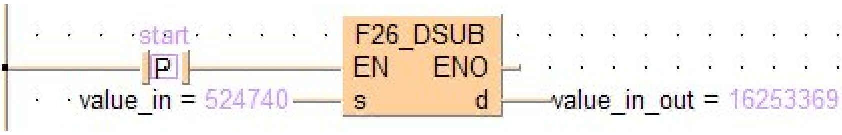
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	DINT	524740	the value, that will b...
2	VAR	value_in_out	DINT	16778109	result after a 0->1 le...

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F26_DSUB(value_in, value_in_out);
END_IF;
```

F27_SUB2

16-bit subtraction, destination can be specified

Steps: 7

Description Subtracts the 16-bit data or 16-bit equivalent constant specified by **s2** from the 16-bit data or 16-bit equivalent constant specified by **s1** if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

Result value 11 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 1

PLC types: **Availability of F27_SUB2 (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	minuend
s2	INT, WORD	subtrahend
d	INT, WORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (overflow or underflow).

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

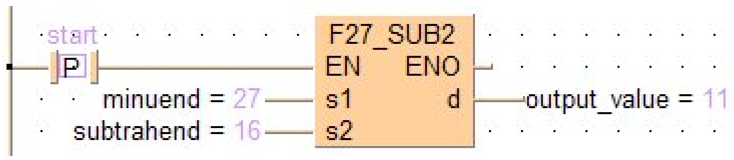
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	INT	27	minuend
2	VAR	subtrahend	INT	16	subtrahend
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 11

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F27_SUB2(minuend, subtrahend, output_value);
END_IF;
```

F28_DSUB2

32-bit subtraction, destination can be specified

Steps: 11

Description Subtracts the 32-bit data or 32-bit equivalent constant specified by **s2** from the 32-bit data or 32-bit equivalent constant specified by **s1** if the trigger is in the ON-state. The result is stored in **d**. All 32-bit values are treated as double integer values.

Example value 16809984

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

← 32-bit area →

Example value 525312

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0	0 0 0 0

Result value 16284672 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	1 1 1 1	1 0 0 0	0 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0

PLC types: **Availability of F28_DSUB2 (see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	minuend
s2	DINT, DWORD	subtrahend
d	DINT, DWORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (overflow or underflow).

Example

In this example the function F28_DSUB2 is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

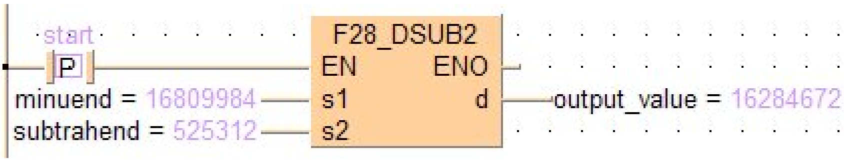
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DINT	16809984	minuent
2	VAR	subtrahend	DINT	525312	subtrahent
3	VAR	output_value	DINT	0	result after a 0->1 le...

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F28_DSUB2(minuend, subtrahend, output_value);
END_IF;
```

Part III F/P Instructions

F45_BSUB

4-digit BCD subtraction

Steps: 5

Description Subtracts the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** from the 16-bit area for 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#2111 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 1
16# (BCD)	2	1	1	1

−

Example value 16#0011 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
16# (BCD)	0	0	1	1

Trigger: ON

Result value 16#2100 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 0	0 0 0 0
16# (BCD)	2	1	0	0

PLC types: **Availability of F45_BSUB (see page 926)**

Data types

Variable	Data type	Function
s	WORD	subtrahend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	minuend and result, 16-bit area for 4-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (overflow).

Example

In this example the function F45_BSUB is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	WORD	16#0011	this value will be subtracted from the output_value
2	VAR	output_value	WORD	16#2111	result after 0->1 leading edge from start: 16#2100

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

```
IF DF(start) THEN
    F45_BSUB(subtrahend, output_value);
END_IF;
```

F46_DBSUB 8-digit BCD subtraction

Steps: 5

Description Subtracts the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** from the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#23210044 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 1 1	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 0	0 1 0 0	0 1 0 0
16# BCD	2	3	2	1	0	0	4	4

32-bit area

−

Example value 16#00210011 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
16# BCD	0	0	2	1	0	0	1	1

Trigger: ON

Result value 16#23000033 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	0 0 1 1
16# BCD	2	3	0	0	0	0	3	3

PLC types: **Availability of F46_DBSUB (see page 926)**

Data types

Variable	Data type	Function
s	DWORD	subtrahend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	minuend and result, 32-bit area for 8-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (overflow).

Example

In this example the function F46_DBSUB is programmed is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	DWORD	16#00210011	this value will be subtracted from the output_value
2	VAR	output_value	DWORD	16#23210044	result after 0->1 leading edge from start: 16#23000033

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

```
IF DF(start) THEN
    F46_DBSUB(subtrahend, output_value);
END_IF;
```

F47_BSUB2

4-digit BCD subtraction, destination can be specified

Steps: 7

Description Subtracts the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s2** from the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#16 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
16# (BCD)	0	0	1	6

Example value 16#4 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
16# (BCD)	0	0	0	4

Trigger: ON

Result value 16#12 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0
16# (BCD)	0	0	1	2

PLC types: **Availability of F47_BSUB2 (see page 926)**

Data types

Variable	Data type	Function
s1	WORD	minuend, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	subtrahend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	result, 16-bit area for 4-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (overflow).

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

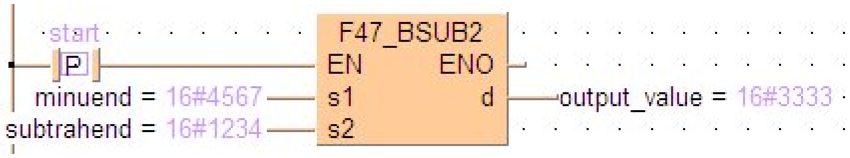
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	WORD	16#4567	minuend
2	VAR	subtrahend	WORD	16#1234	subtrahend
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#3333

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F47_BSUB2(minuend, subtrahend, output_value);
END_IF;
```

F48_DBSUB2

8-digit BCD subtraction, destination can be specified

Steps: 11

Description Subtracts the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s2** from the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s1** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 16#33555588 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 1	0 0 1 0	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1	1 0 0 0	1 0 0 0
16# BCD	3	3	5	5	5	5	8	8

32-bit area

Example value 16#00110022 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 0	0 0 1 0	0 0 1 0
16# BCD	0	0	1	1	0	0	2	2

Trigger: ON

Result value 16#33445566 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 1	0 0 1 1	0 1 0 0	0 1 0 0	0 1 0 1	0 1 0 1	0 1 1 0	0 1 1 0
16# BCD	3	3	4	4	5	5	6	6

PLC types: **Availability of F48_DBSUB2 (see page 926)**

Data types

Variable	Data type	Function
s1	DWORD	minuend, 32-bit area for 8-digit BCD data or equivalent constant
s2	DWORD	subtrahend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	result, 32-bit area for 8-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (overflow).

Example

In this example the function F48_DBSUB2 is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

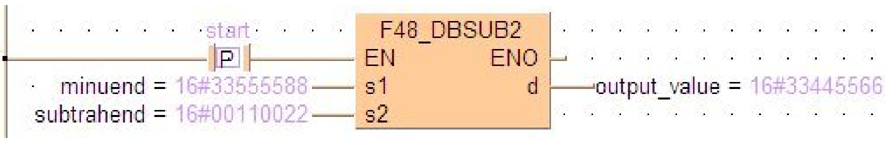
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DWORD	16#33555588	minuent
2	VAR	subtrahend	DWORD	16#00110022	subtrahent
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#33445566

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F48_DBSUB2(minuend, subtrahend, output_value);
END_IF;
```

F37_DEC 16-bit decrement

Steps: 3

Description Subtracts "1" from the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 131081

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 1

32-bit area

Result value 131082 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0

PLC types: **Availability of F37_DEC (see page 926)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be decreased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 16-bit data (underflow).

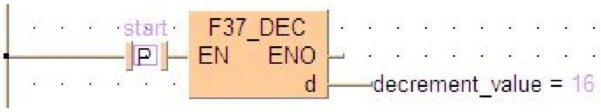
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	INT	17	result after a 0->1 leading edge from start: 16

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST
```
IF DF(start) THEN
    F37_DEC(decrement_value);
END_IF;
```

F38_DDEC

32-bit decrement **Steps: 3**

Description Subtracts "1" to the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 131081

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 1

32-bit area

Result value 131082 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0

PLC types: **Availability of F38_DDEC (see page 926)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit area to be decreased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 32-bit data (underflow).

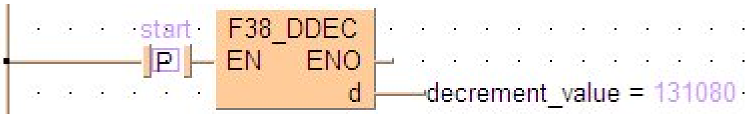
Example In this example the function F38_DDEC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DINT	131081	result after a 0->1 leading edge from start: 131080

Body: When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F38_DDEC(decrement_value);
END_IF;
```

F57_BDEC 4-digit BCD decrement

Steps: 3

Part III F/P Instructions

Description Subtracts "1" from the 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 4322 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 1 0
16# BCD	4	3	2	2

Trigger: ON

Result value 4321 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	4	3	2	1

PLC types: **Availability of F57_BDEC (see page 926)**

Data types

Variable	Data type	Function
d	WORD	16-bit area for BCD data to be decreased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 4-digit BCD data (underflow).

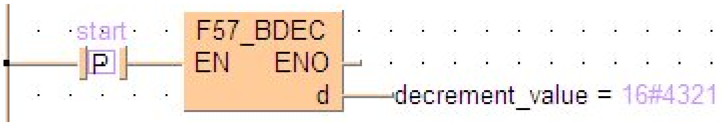
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	WORD	16#4322	result after a 0->1 leading edge from start: 16#4321

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F57_BDEC(decrement_value);
END_IF;
```

F58_DBDEC

8-digit BCD decrement

Steps: 3

Description Subtracts "1" from the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.

Example value 87654322 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 1 0
16# BCD	8	7	6	5	4	3	2	2

32-bit area

Trigger: ON

Result value 87654321 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

PLC types: **Availability of F58_DBDEC (see page 926)**

Data types

Variable	Data type	Function
d	DWORD	32-bit area for BCD data to be decreased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the result exceeds the range of 8-digit BCD data (underflow).

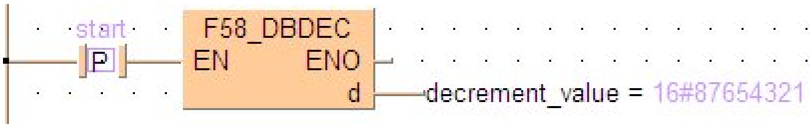
Example In this example the function F58_DBDEC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DWORD	16#87654322	result after a 0->1 leding edge from start: 16#87654321

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F58_DBDEC(decrement_value);
END_IF;
```

F30_MUL

16-bit multiplication, destination can be specified

Steps: 7

Description Multiplies the 16-bit data or 16-bit equivalent constant **s1** and the 16-bit data or 16-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in **d** (32-bit area). All 16-bit values are treated as integer values.

Example value 10

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	1 0 0 0	0 1 0 0	0 0 0 1	0 0 0 1

Result value 170 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0	1 0 1 0

32-bit area

PLC types: **Availability of F30_MUL (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	multiplicand
s2	INT, WORD	multiplier
d	DINT, DWORD	result

The variables **s1, s2** and **d** have to be of the same data type (INT/DINT or WORD/DWORD).

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.

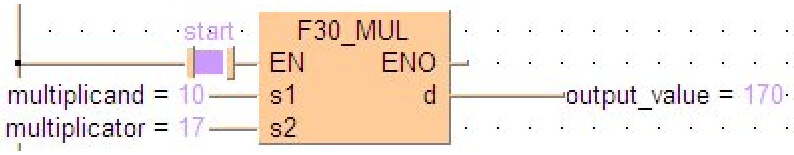
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	INT	10	multiplicand
2	VAR	multiplicator	INT	17	multiplicator
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 170

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F30_MUL(multiplicand, multiplicator, output_value);
END_IF;
```

F31_DMUL

32-bit multiplication, destination can be specified

Steps: 11

Description Multiplies the 32-bit data or 32-bit equivalent constant specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in **d[0]**, **d[1]** (64-bit area). All 32-bit values are treated as double integer values.

PLC types: **Availability of F31_DMUL (see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	multiplicand
s2	DINT, DWORD	multiplier
d	ARRAY **[0..1] of** DINT **or** DWORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

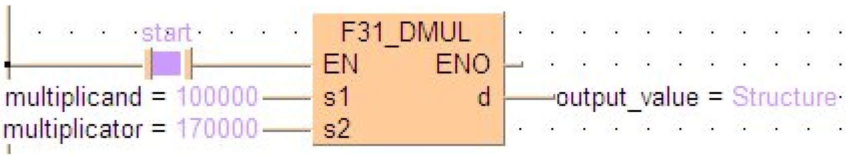
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	multiplicand	DINT	100000	multiplicant
2	VAR	multiplicator	DINT	170000	multiplicator
3	VAR	output_value	ARRAY [0..1] OF DINT	[2(0)]	result after a 0->1 le...

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



Access to the result is possible with output_value[0] and output_value[1].

ST
```
IF start THEN
    F31_DMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F34_MULW

16-bit data multiply (result in 16 bits)

Steps: 7

Description The function multiplies the value specified at input **s1** by the value specified at input **s2**. The result of the function is returned at output **d**. The result at output **d** lies between -32768 and 32767 (i.e. between 16#0 and 16#FFFF). All 16-bit values are treated as integer values.

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 18 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0

PLC types: **Availability of F34_MULW (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	multiplicand
s2	INT, WORD	multiplier
d	INT, WORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the result calculated exceeds the 16-bit area specified at output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	- the result calculated is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

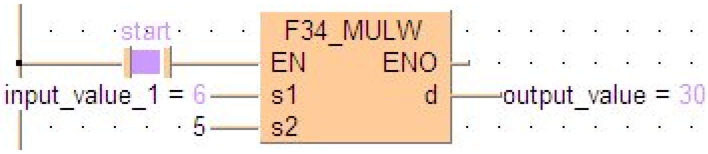
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	INT	6	
2	VAR	output_value	INT	0	result: here 30

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F34_MULW(input_value_1, 5, output_value);
END_IF;
```

F39_DMULD

32-bit data multiply (result in 32 bits)

Steps: 11

Description The function multiplies the value specified at input **s1** by the value specified at input **s2**. The result of the function is returned at output **d**. The result at output 'd' lies between -2147483648 and 2147483647 (i.e. between 16#0 and 16#FFFFFFFF). All 32-bit values are treated as double integer values.

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 18 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0

PLC types: **Availability of F39_DMULD** **(see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	multiplicand
s2	DINT, DWORD	multiplier
d	DINT, DWORD	result

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the result calculated exceeds the 32-bit area specified at output d.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	- the result calculated is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

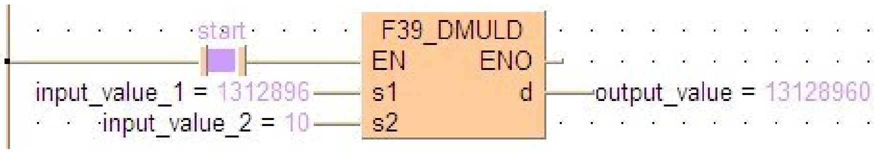
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	input_value_1	DINT	1312896	multiplicant
2	VAR	input_value_2	DINT	10	multiplicator
3	VAR	output_value	DINT	0	result after a 0->1 le...

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F39_DMULD(input_value_1, input_value_2, output_value);
END_IF;
```

F50_BMUL

4-digit BCD multiplication, destination can be specified

Steps: 7

Description Multiplies the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The result is stored in **d** (8-digit area).

Example value 16#20 BCD

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0	0	2	0

×

Example value 16#2 BCD

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0	0	0	2

Result value 16#40 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0	0	0	0	0	0	4	0

← 32-bit area →

PLC types: **Availability of F50_BMUL (see page 926)**

Data types

Variable	Data type	Function
s1	WORD	multiplicand, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	multiplier, 16-bit area for 4-digit BCD data or equivalent constant
d	DWORD	result, 32-bit area for 8-digit BCD data

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	WORD	16#20	multiplicand
2	VAR	multiplicator	WORD	16#2	multiplicator
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start:16#00000040

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F50_BMUL(multiplicand, multiplicator, output_value);
END_IF;
```

Part III F/P Instructions

F51_DBMUL

8-digit BCD multiplication, destination can be 11 specified

Steps: 11

Description Multiplies the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in the ARRAY **d[0]**, **d[1]** (64-digit area).

Example value 16#60008 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0
16# BCD	0	0	0	6	0	0	0	8

32-bit area

+

Example value 16#40002 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
16# BCD	0	0	0	4	0	0	0	2

Result value 16#2400440016 (BCD) if trigger is ON stored in the ARRAY [0..1] of DWORD

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d array[0]	0 0 0 0	0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
16# BCD	0	0	4	4	0	0	1	6

output_array[0]

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d array[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 1 0 0
16# BCD	0	0	0	0	0	0	2	4

output_array[1]

PLC types: **Availability of F51_DBMUL (see page 926)**

Data types

Variable	Data type	Function
s1	DWORD	multiplicand, 32-bit area for 8-digit BCD data or equivalent constant
s1	DWORD	multiplier, 32-bit area for 8-digit BCD data or equivalent constant
d	ARRAY **[0..1] of** DWORD	result

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.

Example

In this example the function F51_DBMUL is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

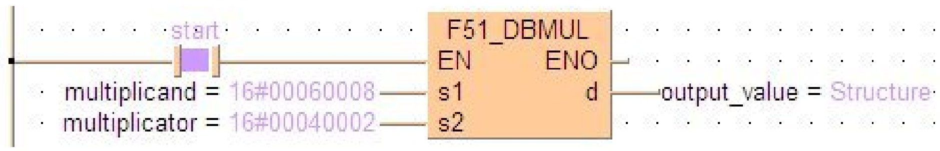
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	DWORD	16#60008	multiplicant
2	VAR	multiplicator	DWORD	16#40002	multiplicator
3	VAR	output_value	ARRAY [0..1] OF DWORD	[2(0)]	result after a 0->1 leading edge from start:[16#00440016, 16#00000024]

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F51_DBMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F32_DIV

16-bit division, destination can be specified

Steps: 7

Description The 16-bit data or 16-bit equivalent constant specified by **s1** is divided by the 16-bit data or 16-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state.

The quotient is stored in **d** and the remainder is stored in the special data register DT9015. All 16-bit values are treated as integer values.

Example value 36

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 1 0	0 1 0 0

÷

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 2 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0

Remainder 2 stored in DT9015/90015

15 . . 12	10 . . 8	7 . . 4	3 . . 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0

PLC types: **Availability of F32_DIV** **(see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	dividend
s2	INT, WORD	divisor
d	INT, WORD	quotient

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculated result is 0.
R9009	%MX0.900.9	for an instant	- the negative minimum value -32768 (16#8000) is divided by -1 (16#FFFF)

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

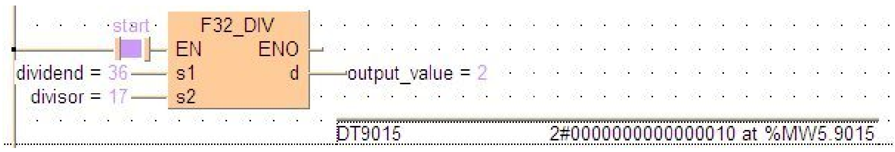
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dividend	INT	36	dividend
2	VAR	divisor	INT	17	divisor
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 2

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F32_DIV(dividend, divisor, output_value);
END_IF;
```

F33_DDIV

32-bit division, destination can be specified

Steps: 11

Description The 32-bit data or 32-bit equivalent constant specified by **s1** is divided by the 32-bit data or 32-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state. The quotient is stored in **d** and the remainder is stored in the special data registers DDT9015. All 32-bit values are treated as double integer values.

Example value 16908416

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0000	0001	0000	0010	0000	0000	1000	0000
	32-bit area							

÷

Example value 589828

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0000	0000	0000	1001	0000	0000	0000	0100

Result value 28 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0000	0000	0000	0000	0001	1100

Remainder 393232

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
	0000	0000	0000	0110	0000	0000	0001	0000
	DT9016/DDT90016				DT9015/DDT90015			

PLC types: **Availability of F33_DDIV** **(see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	dividend
s2	DINT, DWORD	divisor
d	DINT, DWORD	quotient

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages.

POU Header

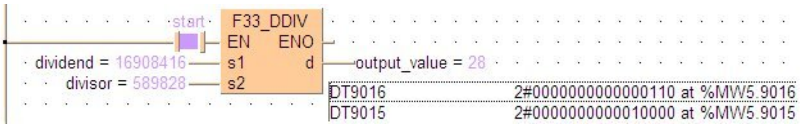
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	DINT	16908416	dividend
2	VAR	divisor	DINT	589828	divisor
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 28

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F33_DDIV(dividend, divisor, output_value);
END_IF;
```

F52_BDIV

4-digit BCD division, destination can be specified

Steps: 7

Description The 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s1** is divided by the 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s2** if the trigger **EN** is in the ON-state.

The quotient is stored in the area specified by **d** and the remainder is stored in special data register DT9015.

Example value 16#0037 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 1 1	0 1 1 1
16# (BCD)	0	0	3	7

÷

Example value 16#0015 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
16# (BCD)	0	0	1	5

Trigger: ON

Result value 16#0002

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
16# (BCD)	0	0	0	2

Remainder 16#0007

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
DT9015	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1
16# (BCD)	0	0	0	7

PLC types: **Availability of F52_BDIV (see page 926)**

Data types

Variable	Data type	Function
s1	WORD	dividend, 16-bit area for BCD data or 4-digit BCD equivalent constant
s2	WORD	divisor, 16-bit area for BCD data or 4-digit BCD equivalent constant
d	WORD	quotient, 16-bit area for BCD data (remainder stored in special data register DT9015/DT90015)

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

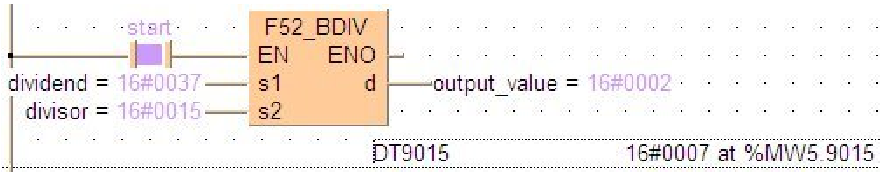
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	WORD	16#0037	dividend
2	VAR	divisor	WORD	16#0015	divisor
3	VAR	output_value	WORD	0	result after 0->1 leading edge from start: 16#0002

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F52_BDIV(dividend, divisor, output_value);
END_IF;
```

F53_DBDIV

8-digit BCD division, destination can be specified

Steps: 11

Description The result is stored in the area specified by **d**, and the remainder is stored in the special data registers DT9016 and DT9015.

Example value 16#00001110 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 0
16# BCD	0	0	0	0	0	1	1	0

32-bit area

÷

Example value 16#0000011 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
16# BCD	0	0	0	0	0	0	1	1

Result value 16#00000100 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0
16# BCD	0	0	0	0	0	1	0	0

Remainder 16#00000010 (BCD) if trigger is ON stored in DT9015 to DT9016 (DDT90015 to DDT90016)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
16# BCD	0	0	0	0	0	0	1	0
	DT9016/DDT90016				DT9015/DDT90015			

PLC types: **Availability of F53_DBDIV (see page 926)**

Data types

Variable	Data type	Function
s1	DWORD	dividend, 32-bit area for BCD data or 8-digit BCD equivalent constant
s2	DWORD	divisor, 32-bit area for BCD data or 8-digit BCD equivalent constant
d	DWORD	quotient, 32-bit area for BCD data (remainder stored in special data register DT9016 and DT9015/DT90016 and DT90015)

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated is 0.

Example

In this example the function F53_DBDIV is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

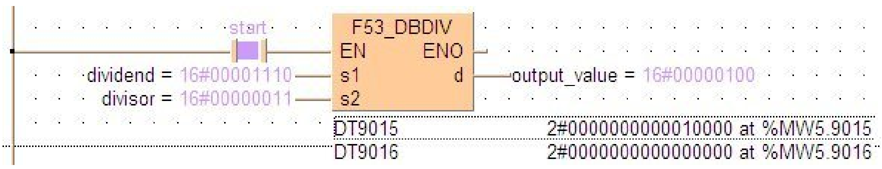
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	DWORD	16#00001110	dividend
2	VAR	divisor	DWORD	16#00000011	divisor
3	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#00000100

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F53_DBDIV(dividend, divisor, output_value);
END_IF;
```

F313_FDIV Floating Point Data Divide

Steps: 14

Description The real number data specified by **s1** is divided by the real number data specified by **s2** when the trigger turns on. The result is stored in **d**.

PLC types: **Availability of F313_FDIV (see page 931)**

- **F313_FDIV cannot be programmed in the interrupt program.**
- **Instead of using F313_FDIV, you can use variables of the type REAL with the more flexible instruction DIV (see page 35).**

Data types

Variable	Data type	Function
s1	REAL	Real number data for dividend.
s2	REAL	Real number data for divisor.
d	REAL	32-bit area for result (destination).

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- data other than real number data is specified in s1 and s2. - the real number data (floating point data) for the divisor specified by s2 is "0.0".
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the result is overflowed.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

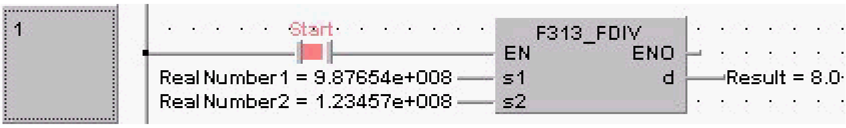
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Result	REAL	0.0	
2	VAR	RealNumber1	REAL	987654321.0	
3	VAR	RealNumber2	REAL	123456789.0	

Body

When the variable **Start** is set to TRUE, the real number entered for the variable **RealNumber1** is divided by the real number entered for **RealNumber2** and the result stored at the address assigned by the compiler to the variable **Result**. The monitor value icon is activated.

LD


1
Start
F313_FDIV
EN
ENO
Real Number1 = 9.87654e+008
s1
d
Result = 8.0
Real Number2 = 1.23457e+008
s2

F70_BCC

Block check code calculation

Steps: 9

Description Calculates the Block Check Code (BCC), which is used to detect errors in message transmission, of **s3** bytes of ASCII data starting from the 16-bit area specified by **s2** according to the calculation method specified by **s1**. The Block Check Code (BCC) is stored in the lower byte of the 16-bit area specified by **d**. (BCC is one byte. The higher byte of **d** does not change.)

s1 specifies the Block Check Code (BCC) calculation method using decimal data as follows:

0: Addition

1: Subtraction

2: Exclusive OR operation

10: Cyclic Redundancy Check (CRC) calculation (only FP10SH Version 3.02 and up)

PLC types: **Availability of F70_BCC (see page 927)**

Data types

Variable	Data type	Function
s1	INT	specifies BCC calculation method: 0 = addition, 1 = subtraction, 2 = exclusive OR operation
s2	INT, WORD	starting 16-bit area to calculate BCC
s3	INT	specifies number of bytes for BCC calculation
d	INT, WORD	16-bit area for storing BCC

Operands

For	Relay				T/C		Register			Constant
s1, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of specified bytes for the target data exceeds the limit of the specified data area.
R9008	%MX0.900.8	for an instant	

Example In this example the function F70_BCC is programmed in ladder diagram (LD) and structured text (ST).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

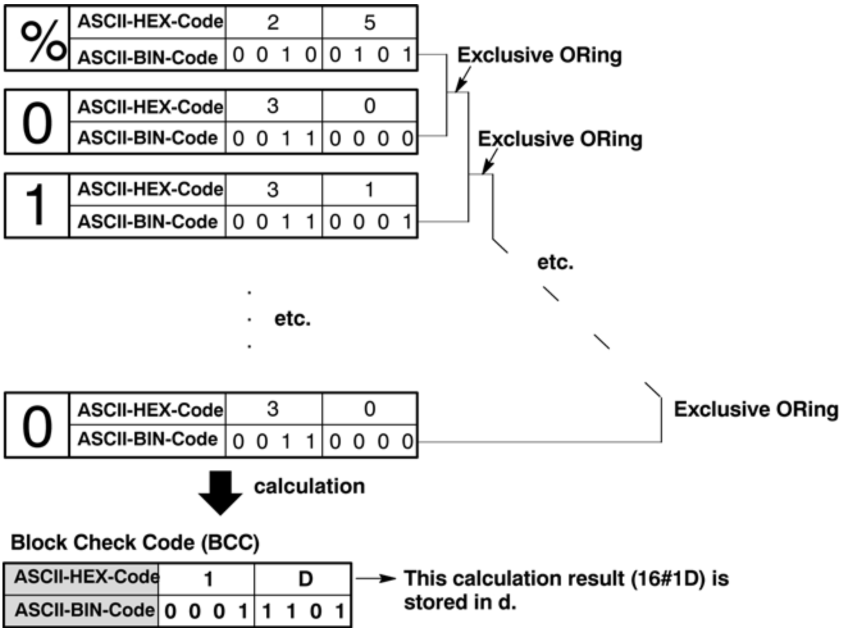
	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	BCC_Calc_Method	INT	2	0 = Addition 1 = Subtraction 2 = Exclusive OR operation
2	VAR	ASCII_String	STRING[32]	'%01#RCSX0000'	
3	VAR	BCC	WORD	0	Result = 16#1D

Body A block check code is performed on the value entered for the variable **ASCII_String** when **Start** becomes TRUE. The exclusive OR operation, which is more suitable when large amounts of data are transmitted, has been chosen for the BCC method.

How the BCC is calculated using the exclusive OR operation:

Exclusive OR operation:

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0



The ASCII BIN code bits of the first two characters are compared with each other to yield an 8-character exclusive OR operation result:

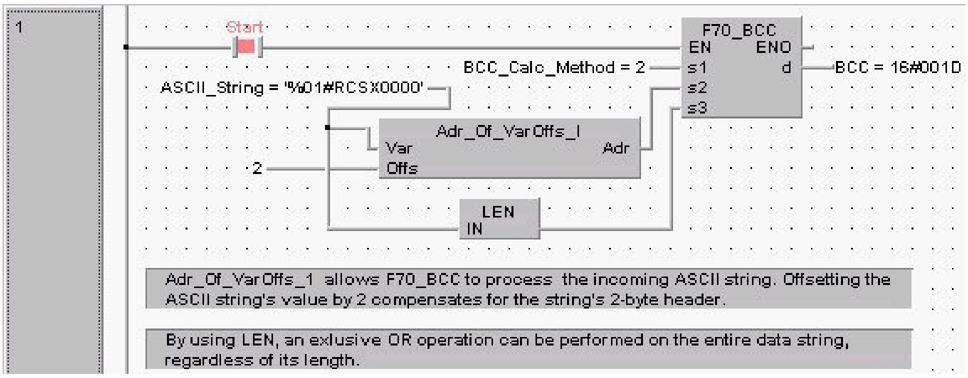
Sign for comparison	ASCII BIN code
%	00100101
0	00110000
Exclusive OR result	00010101

This result is then compared to the ASCII BIN code of the next character, i.e. "1".

Sign for comparison	ASCII BIN code
Exclusive OR result	00010101
1	00110001
Next exclusive OR	00100100

And so on until the final character is reached.

LD



ST

```
IF start THEN
    F70_BCC( s1_Control:= BCC_Calc_Methode,
         s2_Start:= Adr_Of_VarOffs( Var:= ASCII_String,
         Offs:= 2),
         s3_Number:= LEN( ASCII_String),
         d=> BCC);
END_IF;
```

F160_DSQR

32-bit data square root

Steps: 7

Description The square root of the 32-bit data or constant value specified by **s** is calculated if the trigger **EN** is in the ON-state. The result (square root) is stored in **d**.

The figures of the first decimal place and below are disregarded.

Example value 64

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
Binary	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0
Decimal	64							

32-bit area

Trigger: ON

Result value 8

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
Binary	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
Decimal	8							

PLC types: **Availability of F160_DSQR (see page 929)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source, 32-bit area to be calculated
d	DINT, DWORD	square root (decimal places deleted)

The variables **s1** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function F160_DSQR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	70	input_value:=70
2	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 8

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F160_DSQR(input_value, output_value);
END_IF;
```

F87_ABS 16-bit data absolute value

Steps: 3

Description Gets the absolute value of 16-bit data with the sign specified by **d** if the trigger **EN** is in the ON-state.

The absolute value of the 16-bit data with +/- sign is stored in **d**. This instruction is useful for handling data whose sign (+/-) may vary.

PLC types: **Availability of F87_ABS (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area for storing original data and its absolute value

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the 16-bit data is the negative minimum value -32768 (16#8000).
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the 16-bit data is the negative value in the range from -1 to -32767 (16#FFFF to 16#8001).

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	INT	-123	result after a 0->1 leading edge from start: 123

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST
```
IF start THEN
    F87_ABS(abs_value);
END_IF;
```

F88_DABS 32-bit data absolute value

Steps: 3

Description Gets the absolute value of 32-bit data with the sign specified by **d** if the trigger **EN** is in the ON-state. The absolute value of the 32-bit data with sign is stored in **d**. This instruction is useful for handling data whose sign (+/-) may vary.

PLC types: **Availability of F88_DABS (see page 927)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit area for storing original data and its absolute value

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the 32-bit data is the negative minimum value -2147483648 (16#80000000).
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the 32-bit data is the negative value in the range from -1 to -2147483647 (16#FFFFFFFF to 16#80000001).

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	DINT	-123	result after a 0->1 leading edge from start: 123

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

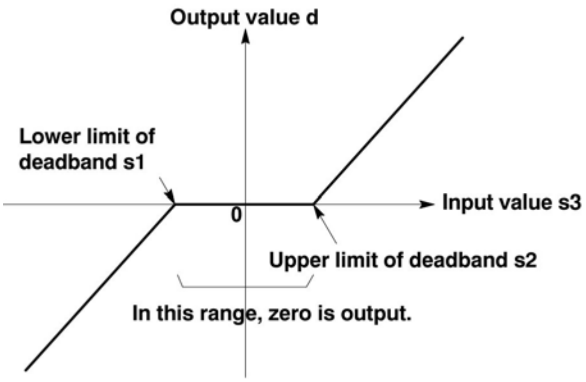
```
IF start THEN
        F88_DABS(abs_value);
END_IF;
```

F287_BAND 16-bit data deadband control

Steps: 10

Description The function compares the input value at input **s3** with a deadband whose lower limit is specified at input **s1** and whose upper limit is specified at **s2**. The result of the function is returned at output **d** as follows:

- If the input value at input **s3** < **s1**, the lower limit at input **s1** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** > **s2**, the upper limit at input **s2** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, 0 is returned as the output value at **d**.



PLC types: **Availability of F287_BAND (see page 931)**

Data types

Variable	Data type	Function
s1	INT, WORD	the area where the lower limit is stored or the lower limit data
s2	INT, WORD	the area where the upper limit is stored or the upper limit data
s3	INT, WORD	the area where the input value is stored or the input value data
d	INT, WORD	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at s1 > s2.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	TRUE	- the input value at s3 is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

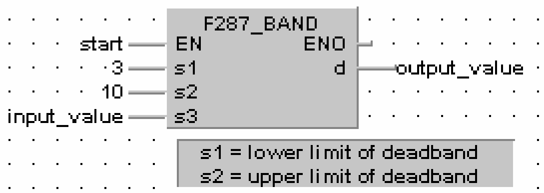
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	12	
2	VAR	output_value	INT	0	result: here 2

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. The constant 3 (lower limit of the deadband) and 10 (upper limit of the deadband) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



ST

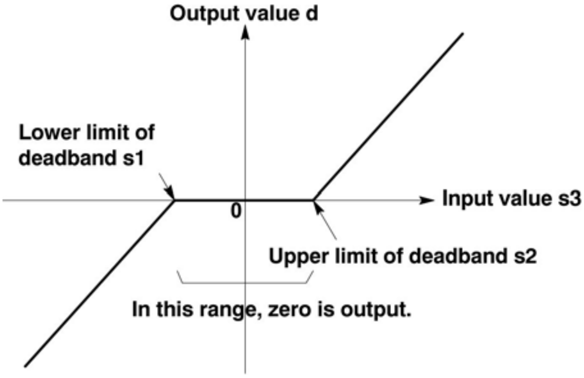
```
IF start THEN
    F287_BAND( 3, 10, input_value, output_value);
END_IF;     (* 3=lower limit of deadband, 10=upper limit of
deadband *)
```

F288_DBAND 32-bit data deadband control

Steps: 10

Description The function compares the input value at input **s3** with a deadband whose lower limit is specified at input **s1** and whose upper limit is specified at **s2**. The result of the function is returned at output **d** as follows:

- If the input value at input **s3** < **s1**, the lower limit at input **s1** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** > **s2**, the upper limit at input **s2** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, 0 is returned as the output value at **d**.



PLC types: **Availability of F288_DBAND (see page 931)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	the area where the lower limit is stored or the lower limit data
s2	DINT, DWORD	the area where the upper limit is stored or the upper limit data
s3	DINT, DWORD	the area where the input value is stored or the input value data
d	DINT, DWORD	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at s1 > s2.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the input value at s3 is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

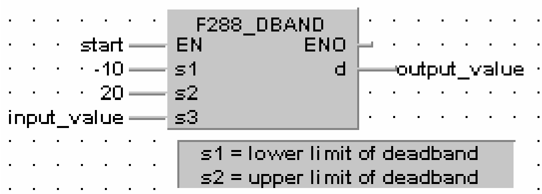
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	-22	
2	VAR	output_value	DINT	0	result: here -12

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. The constant -10 (lower limit of the deadband) and 20 (upper limit of the deadband) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



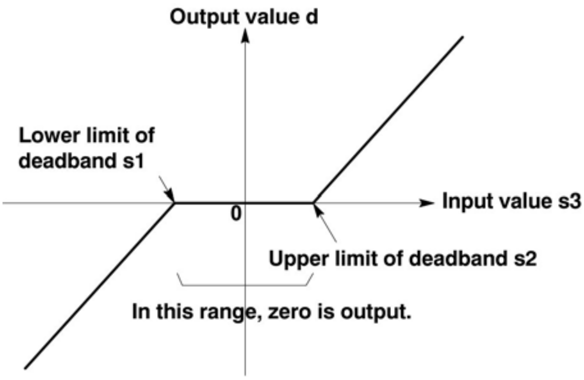
ST

```
IF start THEN
    F288_DBAND( -10, 20, input_value, output_value);
END_IF;     (* 10=lower limit of deadband, 20=upper limit of
deadband *)
```

F348_FBAND Floating point data deadband control

Steps: 17

Description The function compares the input value at input **s3** with a deadband whose lower limit is specified at input **s1** and whose upper limit is specified at **s2**. The result of the function is returned at output **d** as follows:



Comparison between s1 and s2	Flag		
	R900A (> flag)	R900B (= flag)	R900C (< flag)
s1 < s2	off	off	on
s1 ≤ s3 and s2 ≤ s1	off	on	off
s3 < s1	on	off	off

PLC types: **Availability of F348_FBAND (see page 932)**

Data types

Variable	Data type	Function
s1	REAL	the area where the lower limit is stored or the lower limit data
s2	REAL	the area where the upper limit is stored or the upper limit data
s3	REAL	the area where the input value is stored or the input value data
d	REAL	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the values at inputs s1, s2, and s3 are not REAL numbers or the value at s1 > s2.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

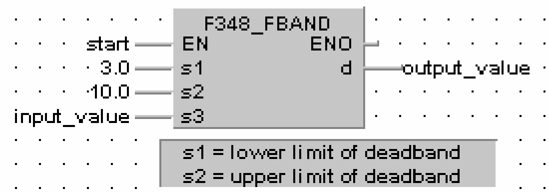
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	12.0	
2	VAR	output_value	REAL	0.0	result: here 2.0

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

The constants 3.0 and 10.0 are assigned to the inputs s1 (lower limit of the deadband) and s3 (upper limit of the deadband). However, you can declare two variables in the POU header and write them in the function in the body at the inputs. When the variable **start** is set to TRUE, the function is carried out. Since the **input_value** = 12.0 is larger than the value of the upper limit of the deadband at s2, the **output_value** = 12.0 -10.0 = 2.0.

LD



ST

```
IF start THEN
    F348_FBAND( s1_Min:= 3.0 ,
         s2_Max:= 10.0 ,
         s3_In:= input_value ,
         d=> output_value )
END_IF;       (* 3.0=lower limit of deadband, 10.0=upper
limit *)
```

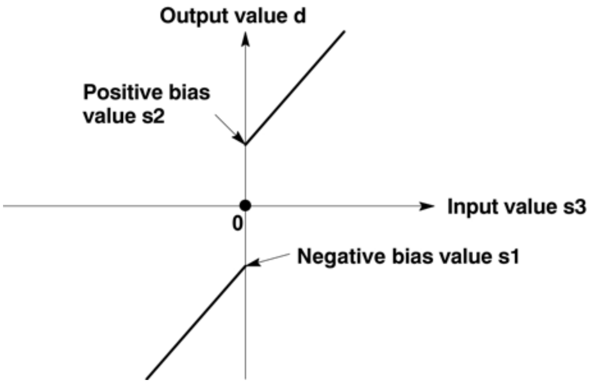
F289_ZONE

16-bit data zone control

Steps: 10

Description The function adds an offset value to the input value at input **s3**. The offset values for the negative and positive areas are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:

- If the input value at input **s3** < 0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0, 0 is returned at the output value to output **d**.
- If the input value at input **s3 >** 0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



PLC types: **Availability of F289_ZONE (see page 931)**

Data types

Variable	Data type	Function
s1	INT, WORD	area where negative bias value is stored or negative bias value data
s2	INT, WORD	area where positive bias value is stored or positive bias value data
s3	INT, WORD	area where input value is stored or input value data
d	INT, WORD	area where output value is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculation results in an overflow or an underflow of output d.
R9009	%MX0.900.9	for an instant	- the input value s3 is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

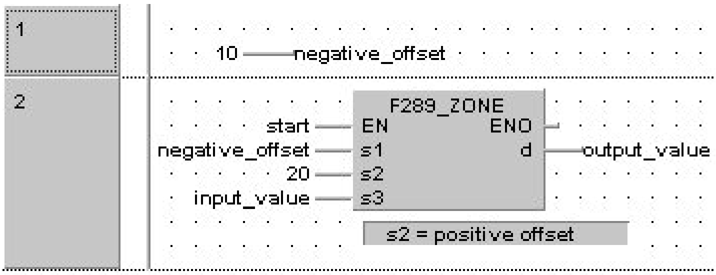
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negative_offset	INT	0	
2	VAR	input_value	INT	-12	
3	VAR	output_value	INT	0	result: here -2

In this example the input variables **input_value** and **negative_offset** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. It adds the corresponding negative offset value = 10 to the negative **input_value** = -12. However, you can declare a variable in the POU header and assign it to the function's input in the body.

LD



ST

```
IF start THEN
    F289_ZONE( negative_offset, 20, input_value,
output_value);
END_IF;      (*negative_offset=neg. offset, 20=pos. offset
*)
```

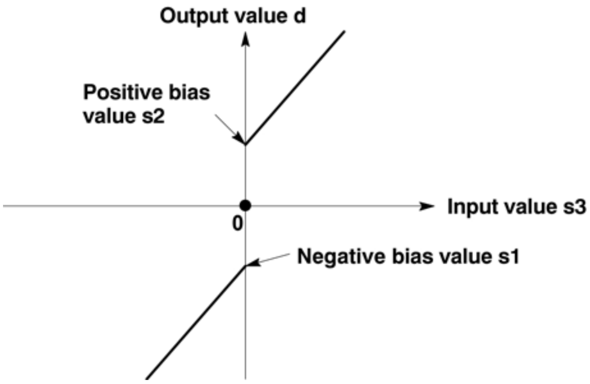
F290_DZONE

32-bit data (double word data) zone control

Steps: 10

Description The function adds an offset value to the input value at input **s3**. The offset value for the negative and positive area are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:

- If the input value at input **s3** < 0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0, 0 is returned at the output value to output **d**.
- If the input value at input **s3 >** 0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



PLC types: **Availability of F290_DZONE (see page 931)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	area where negative bias value is stored or negative bias value data
s2	DINT, DWORD	area where positive bias value is stored or positive bias value data
s3	DINT, DWORD	area where input value is stored or input value data
d	DINT, DWORD	area where output value is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the calculation results in an overflow or an underflow of output d.
R9009	%MX0.900.9	for an instant	- the input value s3 is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

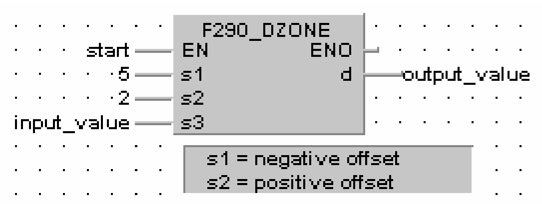
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	18	
2	VAR	output_value	DINT	0	result: here 20

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. It adds the corresponding positive offset value = 2 to the positive input value = 18. The constants 5 (negative offset) and 2 (positive offset) are assigned to inputs s1 and s2 respectively. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



ST

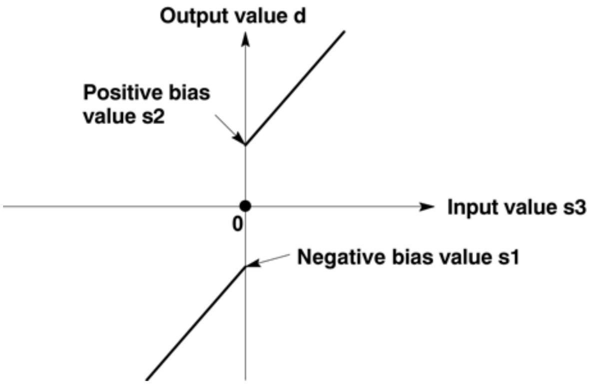
```
IF start THEN
    F290_DZONE( s1_NegBias:= 5,
         s2_PosBias:= 2,
         s3_In:= input_value,
         d=> output_value);
END_IF;      (*5=neg. offset, 2=pos. offset *)
```

F349_FZONE Floating point data zone control

Steps: 17

Description The function adds an offset value to the input value at input **s3**. The offset value for the negative and positive area are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:

- If the input value at input **s3** < 0.0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0.0, 0.0 is returned as the output value to output **d**.
- If the input value at input **s3 >** 0.0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



PLC types: **Availability of F349_FZONE (see page 932)**

Data types

Variable	Data type	Function
s1	REAL	area where negative bias value is stored or negative bias value data
s2	REAL	area where positive bias value is stored or positive bias value data
s3	REAL	area where input value is stored or input value data
d	REAL	area where output value is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the values at inputs s1, s2, and s3 are not REAL numbers.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

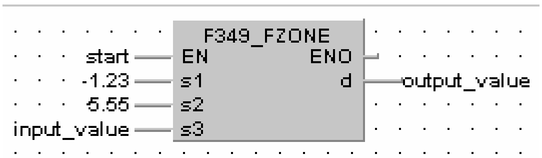
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-10.0	
2	VAR	output_value	REAL	0.0	result: here -11.23

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

The constant -1.23 is assigned to input s1 (negative offset) and the constant 5.55 is assigned to input s2 (positive offset). However, you can declare two variables in the POU header and write them in the function in the body at the inputs. When the variable **start** is set to TRUE, the function is carried out. Since the **input_value** is negative (-10.0), the negative offset -1.23 is added to it. The result here is: **output_value** = -11.23.

LD



ST

```
IF start THEN
    F349_FZONE( s1_NegBias:= -1.23 ,
        s2_PosBias:= 5.55 ,
        s3_In:= input_value ,
        d=> output_value );
END_IF;        (*-1.23=neg. offset, 5.55=pos. offset *)
```

Part III F/P Instructions

F85_NEG

16-bit data two's complement

Steps: 3

Description Takes two's complement of 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Two's complement of the original 16-bit data is stored in **d**.

Two's complement is a number system used to express positive and negative numbers in binary format. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. Two's complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 16-bit data from positive to negative or from negative to positive.

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Decimal data	3			

start: ON

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1
Decimal data	-3			

PLC types: **Availability of F85_NEG (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area for storing original data and its two's complement

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

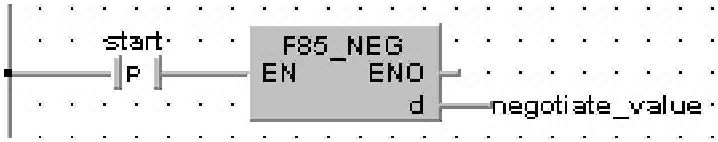
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001111

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F85_NEG(negotiate_value);
END_IF;
```

F86_DNEG

32-bit data two's complement

Steps: 3

Description Takes two's complement of 32-bit data specified by **d** if the trigger **EN** is in the ON-state. Two's complement of the original 32-bit data is stored in **d**.

Two’s complement is a number system used to express positive and negative numbers in binary format. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. Two’s complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 16-bit data from positive to negative or from negative to positive.

Destination	DT1				DT0			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1
Decimal data	-3							

32-bit area

start: ON

Destination	DT1				DT0			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Decimal data	3							

32-bit area

PLC types: **Availability of F86_DNEG (see page 927)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit area for storing original data and its two's complement

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function F86_DNEG is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	DWORD	2#11010001000011000110000011101111	result after a 0->1 leading edge from start: 2#0010111011110011 1001111100010001

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

```
IF DF(start) THEN
    F86_DNEG(negotiate_value);
END_IF;
```

F270_MAX — Maximum value search in 16-bit data table — Steps: 8

Description The function searches for the maximum value and its position in a 16-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The maximum value is returned at output **max** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

PLC types: **Availability of F270_MAX (see page 930)**

Data types

Variable	Data type	Function
s1	INT, WORD	starting area of data table
s2	INT, WORD	ending area of data table
max	INT	specifies maximum value
pos	INT	position where maximum value was found

Operands

For	Relay				T/C		Register			Const.
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
max, pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

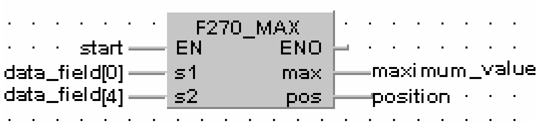
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	maximum_value	INT	0	result: here 6
3	VAR	position	INT	0	result: here 2

Body: When the variable **start** is set to TRUE, the function is carried out. It searches for the maximum value and its position in the **data_field**. The result is here: **maximum_value** = 6 and **position** = 2.

LD



ST

```
IF start THEN
    F270_MAX( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        Max=> maximum_value,
        Pos=> position);
END_IF;
```

F271_DMAX

Maximum value search in 32-bit data table **Steps: 8**

Description The function searches for the maximum value and its position in a 32-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The maximum value is returned at output **max** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

PLC types: **Availability of F271_DMAX (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	starting area of data table
s2	DINT, DWORD	ending area of data table
max	DINT	specifies maximum value
pos	WORD	position where maximum value was found

Operands

For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWF	DWL	DSV	DEV	DDT	DLD	DFL	-
max	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

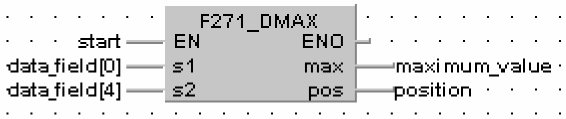
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	maximum_value	DINT	0	result: here 222222
3	VAR	position	INT	0	result: here 2

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the maximum value and its position in the **data_field**. The result is here: **maximum_value** = 222222 and **position** = 2.

LD



ST
```
IF start THEN
    F271_DMAX( s1_Start:= data_field[0],
          s2_End:= data_field[4],
          Max=> maximum_value,
          Pos=> position);
END_IF;
```

F272_MIN Minimum value search in 16-bit data table

Steps: 8

Description The function searches for the minimum value and its position in a 16-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The minimum value is returned at output **min** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

PLC types: **Availability of F272_MIN (see page 930)**

Data types

Variable	Data type	Function
s1	INT, WORD	starting area of data table
s2	INT, WORD	ending area of data table
min	INT	specifies minimum value
pos	INT	position where minimum value was found

Operands

For	Relay				T/C		Register			Const.
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
min, pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

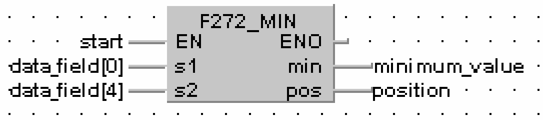
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	minimum_value	INT	0	result: here -3
3	VAR	position	INT	0	result: here 3

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the minimum value and its position in the **data_field**. The result is here: **minimum_value** = -3 and **position** = 3.

LD



ST

```
IF start THEN
    F272_MIN( s1_Start:= data_field[0],
            s2_End:= data_field[4],
            Min=> minimum_value,
            Pos=> position);
END_IF;
```

F273_DMIN

Minimum value search in 32-bit data table **Steps: 8**

Description The function searches for the minimum value and its position in a 32-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The minimum value is returned at output **min** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

PLC types: **Availability of F273_DMIN (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	starting area of data table
s2	DINT, DWORD	ending area of data table
min	DINT	specifies minimum value
pos	INT	position where minimum value was found

Operands

For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
min	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	minimum_value	DINT	0	result: here -333333
3	VAR	position	INT	0	result: here 3

Body

When the variable **start** is set to TRUE, the function is carried out. It searches for the minimum value and its position in the **data_field**. The result is here: **minimum_value** = -333333 and **position** = 3.

LD

ST

```
IF start THEN
    F273_DMIN( s1_Start:= data_field[0],
          s2_End:= data_field[4],
          Min=> minimum_value,
          Pos=> position);
END_IF;
```

F275_MEAN

Total and mean numbers calculation in 16-bit data table

Steps: 8

Description This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 16-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The sum of all elements in the data table is returned at output **sum** and the arithmetic mean of all elements in the data table is returned at output **mean**. The arithmetic mean is rounded off if it is not a whole number.

PLC types: **Availability of F275_MEAN (see page 930)**

Data types

Variable	Data type	Function
s1	INT, WORD	starting area of data table
s2	INT, WORD	ending area of data table
mean	INT	mean of all elements in data table area specified
sum	DINT	sum of all elements in data table area specified

Operands

For	Relay				T/C		Register			Const.
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
mean	-	WY	WR	WL	SV	EV	DT	LD	FL	-
sum	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas are different.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the total value range overflows or underflows the 32-bit range.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

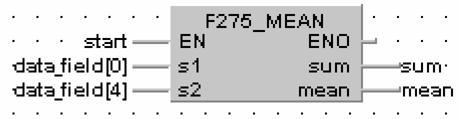
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	sum	DINT	0	result: here 9
3	VAR	mean	INT	0	result: here 1

Body

When the variable **output** is set to TRUE, the function F275_MEAN is carried out. The function calculates the sum of all elements of the data table (sum = 4 + 3 + 8 + (-2) + 1 + (-6) = 8) and writes the result (in this case 8) to the variable **sum.** Additionally, the function calculates the arithmetic mean of all elements of the data table (mean = sum/6 = (4 + 3 + 8 + (-2) + 1 + (-6)) / 6 = 1.333) and writes the roanded-off number (in this case 1) to the variable **mean**.

LD



ST

```
IF start THEN
    F275_MEAN( s1_Start:= data_field[0],
         s2_End:= data_field[4],
         Sum=> sum,
         Mean=> mean);
END_IF;
```

F276_DMEAN

Total and mean numbers calculation in 32-bit data table

Steps: 8

Description This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 32-bit data table.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The sum of all elements in the data table are returned at output **sum** and the arithmetic mean of all elements in the data table are returned at output **mean**. The arithmetic mean is rounded off if it is not already a whole number.

PLC types: **Availability of F276_DMEAN (see page 931)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	starting area of data table
s2	DINT, DWORD	ending area of data table
mean	DINT	mean of all elements in data table area specified
sum	ARRAY **[0..1] of** DINT	sum of all elements in data table area specified

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
mean, sum	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2. - the address areas are different.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the total value range overflows or underflows the 32-bit range.

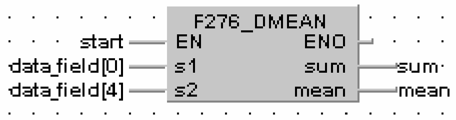
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	sum	ARRAY [0..1] OF DINT	[2(0)]	result: here ARRAY[0] = 16#FFFE4DFF (lower 32 bits) ARRAY[1] = 16#FFFFFFFF (upper 32 bits) corresponds to 16#FFFFFFFFFFFE4DFF = -111105
3	VAR	mean	DINT	0	result: here -22221

Body When the variable **start** is set to TRUE, the function is carried out. The function calculates the sum of all elements of ARRAY **data_field** (sum = 2 + 3 + 222222 + (-333333) + 1 = -111105) and transfers the result to the variable **sum**. In addition, the function calculates the mean (mean = sum/5 = -111105/5 = -22221) and transfers the result to the variable **mean**.

LD



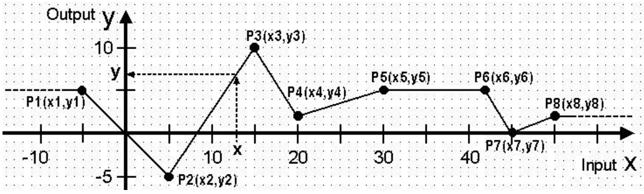
ST

```
IF start THEN
    F276_DMEAN( s1_Start:= data_field[0],
         s2_End:= data_field[4],
         Sum=> sum,
         Mean=> mean);
END_IF;
```

F282_SCAL Linearization of 16-bit data

Steps: 8

Description The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points Pw(xw, yw) and Pw+1(xw+1, yw+1). In this example, w is the nearest reference point whose x value is smaller than the input value x, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **s**.



The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature y in relation to the outside temperature x
- etc.

PLC types: **Availability of F282_SCAL (see page 931)**

Data types

Variable	Data type	Function
x	INT	Output value **x**
xy_data	DUT	The first element of an DUT-type variable that contains the xy value pairs.
y	INT	Output value **y**
EN	BOOL	Activation of the function (when EN = TRUE, the function is executed during each PLC cycle)
ENO	BOOL	ENO is set to TRUE as soon the function is executed. Helpful when cascading function blocks with EN functions.

Operands

For	Relay				T/C		Register			Constant
x	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
y	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of reference points is not between 2 ... 100, or the x values are not in ascending order (x1 < x2 < x3 < ...).
R9008	%MX0.900.8	for an instant	

■ Limitations of the output value y:

If the input value **x** is smaller than the x-coordinate of the first reference point (P1: x < x1), the output **y** is set to the first reference point's y-coordinate (output **y**

= y1, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (P8: x > x8), the output **y** is set to the last reference point's y-coordinate (output **y** = y8, horizontal dashed line in the graphic's upper right corner).

■ DUT for the xy value pairs (reference points P1, P2, ...):

The reference points (P1, P2, ...) are copied to the function via an DUT-type variable that contains the number of reference points and the xy value pairs (number; x1, x2, ...; y1, y2; ...).

Structure of the DUT:

1. Entry: Variable of the data type INT that contains the number of reference points.
   The number of reference points (xy value pairs) can be set anywhere between 2 ... 100. In the graph, eight reference points (P1 ... P8) are used.
2. Entry: Variable of the data type ARRAY (see page 20) [0..z] OF INT that contains the x values.
   Here z represents the place marker for the number of reference points (see entry 1).
3. Entry: Variable of the data type ARRAY (see page 20) [0..z] OF INT that contains the y values.
   Here z represents the place marker for the number of reference points (see entry 1).

■ Important information:

x values

The x values have to be entered in ascending order (x1 < x2 < x3 < ...). If the x values are the same (e.g. x2 = x3 = x4) the reference points P2(x2,y2) and P3(x3,y3) are ignored.

Overflow of the function:

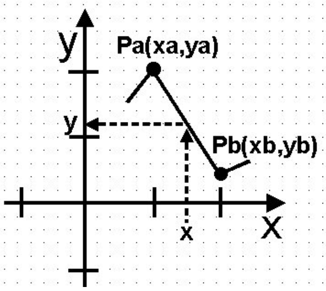
In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

|ya - yb| < 32767

|x - xb| < 32767

|(ya - yb)*(x - xb)| < 32767

|xa - xb| < 32767



Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value y. For example, if at the position x, y = 511,13, the function returns the value 511.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

DUT

In the DUT Pool the number of reference points and the xy value pairs are declared.

Interpolation_8 [DUT]

	Identifier	Type	Initial	Comment
0	referencepoints	INT	8	Eight reference points were initialized since the data fields for the x- and y-values contain maximal values.
1	X_values	ARRAY [1..8] OF INT	[8(0)]	Field 1..8 ->contains 8 x-values (x1, x2, ..., x8)
2	Y_values	ARRAY [1..8] OF INT	[8(0)]	Field 1..8 ->contains 8 y-values (y1, y2, ..., y8)

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

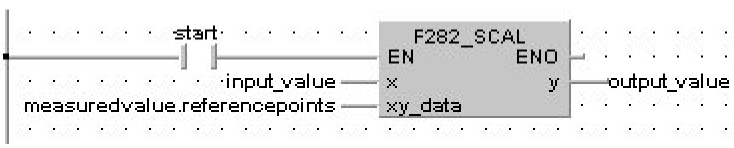
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	0	input value x
2	VAR	measured_value	Interpolation_8	X_values := [-5,5,15,20,30,42,45,50], Y_values := [5,-5,10,2,2(5),0,2]	number of reference points x-values, y-values
3	VAR	output_value	INT	0	output value y

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.X_Values[1]** for x**.**

Body

When the variable **start** is set to TRUE, the function is carried out. For the input value at position x, the output value y is calculated via linear interpolation of the neighboring reference points stored in the variable **measured value**.

LD



ST

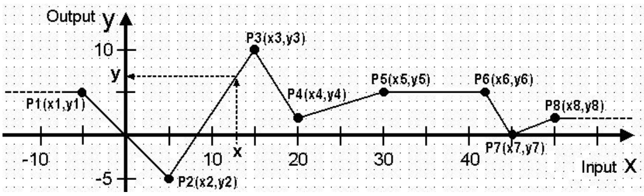
```
IF start THEN
    F282_SCAL(input_value, measured_value.referencepoints,
output_value);
END_IF;
```

F283_DSCAL

Linearization of 32-bit data

Steps: 8

Description The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points Pw(xw, yw) and Pw+1(xw+1, yw+1). In this example, w is the nearest reference point whose x value is smaller than the input value x, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **s**.



The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature y in relation to the outside temperature x
- etc.

PLC types: **Availability of F283_DSCAL (see page 931)**

Data types

Variable	Data type	Function
x	INT	Input value **x**
xy_data	DUT	The first element of a DUT-type variable that contains the xy value pairs.
y	DINT	Output value **y**
EN	BOOL	Activation of the function (when EN = TRUE, the function is executed during each PLC cycle)
ENO	BOOL	ENO is set to TRUE as soon as the function is executed. Helpful when cascading function blocks with EN functions.

Operands

For	Relay				T/C		Register			Constant
x	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
y	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of reference points is not between 2 ... 100, or the x values are not in ascending order (x1 < x2 < x3 < ...).
R9008	%MX0.900.8	for an instant	

■ **Limitations of the output value y:**

If the input value **x** is smaller than the x-coordinate of the first reference point (P1: x < x1), the output **y** is set to the first reference point's y-coordinate (output **y** = y1, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (P8: x > x8), the output **y** is set to the last reference point's y-coordinate (output **y** = y8, horizontal dashed line in the graphic's upper right corner).

■ **DUT for the xy value pairs (reference points P1, P2, ...):**

The reference points (P1, P2, ...) are copied to the function via a DUT-type variable that contains the number of reference points and the xy value pairs (number; x1, x2, ...; y1, y2; ...).

Structure of the DUT:

1. Entry: Variable of the data type INT that contains the number of reference points.
   The number of reference points (xy value pairs) can be anywhere between 2 ... 100. In the graph, eight reference points (P1 ... P8) are used.
2. Entry: Variable of the data type ARRAY (see page 20) [0..z] OF DINT that contains the x values.
   Here z represents the place marker for the number of reference points (see entry 1).
3. Entry: Variable of the data type ARRAY (see page 20) [0..z] OF DINT that contains the x values.
   Here z represents the place marker for the number of reference points (see entry 1).

■ **Important information:**

x values

The x values have to be entered in an ascending order (x1 < x2 < x3 < ...). If the x values are the same (e.g. x2 = x3 = x4) the reference points P2(x2,y2) and P3(x3,y3) are ignored.

Overflow of the function:

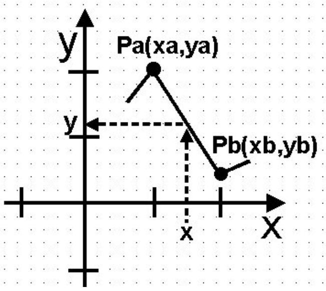
In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

|ya - yb| < 2147483647

|x - xb| < 2147483647

|(ya - yb)*(x - xb)| < 2147483647

|xa - xb| < 2147483647



Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value y. For example, if at the position x, y = 511,13, the function returns the value 511.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

DUT In the DUT Pool, the number of reference points and the xy value pairs are declared.

Interpolation_8d [DUT]

	Identifier	Type	Initial	Comment
0	referencepoints	DINT	8	Eight reference points were initialized since the data fields for the x- and y-values contain maximal values.
1	X_values	ARRAY [1..8] OF DINT	[8(0)]	Field 1..8 ->contains 8 x-values (x1, x2, ..., x8)
2	Y_values	ARRAY [1..8] OF DINT	[8(0)]	Field 1..8 ->contains 8 y-values (y1, y2, ..., y8)

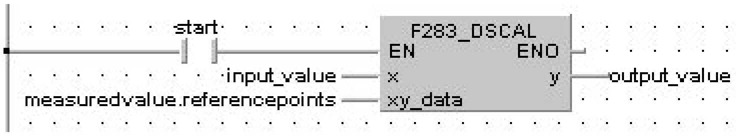
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	0	input value x
2	VAR	measured_value	Interpolation_8d	X_values := [-5,5,15,20,30,42,45,50], Y_values := [5,-5,10,2,2(5),0,2]	number of reference points x-values, y-values
3	VAR	output_value	DINT	0	output value y

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.Y_Values[3]** for y3**.**

Body When the variable **start** is set to TRUE, the function is carried out. For the **input value** at position x, the output value y is calculated via linear interpolation between the neighboring reference points stored in the variable **measured value**.

LD



ST

```
IF start THEN
    F283_DSCAL(input_value, measured_value.referencepoints,
output_value);
END_IF;
```

F96_SRC

Table data search (16-bit search)

Steps: 7

Description

Searches for the value that is the same as **s1** in the block of 16-bit areas specified by **s2** (starting area) through **s3** (ending area) if the trigger **EN** is in the ON-state.

When the search operation is performed, the search results are stored as follows:

- The number of data that is the same as **s1** is transferred to special data register DT9037.
- The position the data is first found in, counting from the starting 16-bit area, is transferred to special data register DT9038.

Be sure that **s2** ≤ **s3**.

PLC types: **Availability of F96_SRC** **(see page 927)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit area or equivalent constant to store the value searched for
s2	INT, WORD	starting 16-bit area of the block
s3	INT, WORD	ending 16-bit area of the block

The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2, s3	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example the function F96_SRC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

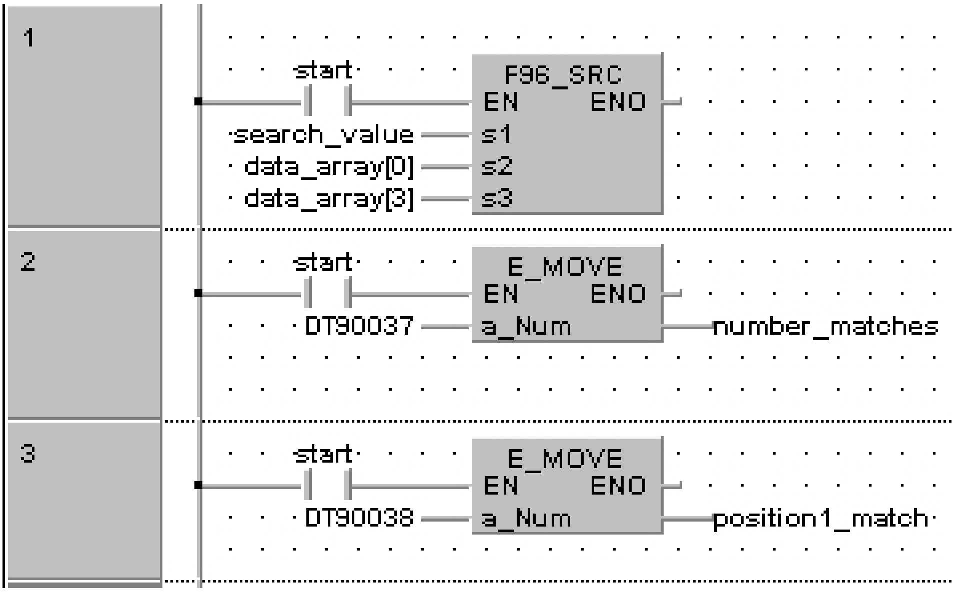
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	search_value	WORD	16#20	specifies the value to search for
2	VAR	data_array	ARRAY [0..3] OF WORD	[16#101,16#2A04,16#20,16#20]	2 matches for 16#20 data_array[2] = 1st match
3	VAR	number_matches	INT	0	
4	VAR	position1_match	INT	0	

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

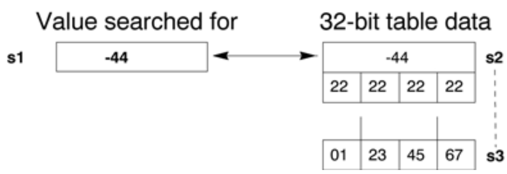
```
IF start THEN
    F96_SRC( s1:= search_value ,
         s2_Start:= data_array[0] ,
         s3_End:= data_array[3] );
    number_matches:=DT90037;
    position_1match:=DT90038;
END_IF;
```

F97_DSRC

32-bit table data search

Steps: 9

Description The function searches for the value specified at input **s1** in a block of 32-bit areas whose beginning is specified at input **s2** and whose end is specified at input **s3**.



The number of data items that match **s1** is stored in special data register DT90037.

The relative position of the first matching data item, counting from the starting 32-bit area **s2**, is stored in special data register DT90038.

PLC types: **Availability of F97_DSRC (see page 927)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit area or equivalent constant to store the value searched for
s2	DINT, DWORD	starting 32-bit area of the block
s3	DINT, DWORD	ending 32-bit area of the block

The adresses of the variables at inputs **s2** and **s3** must be of the same adress type.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s2, s3	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variables at outputs s2 > s3.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Part III F/P Instructions

POU Header

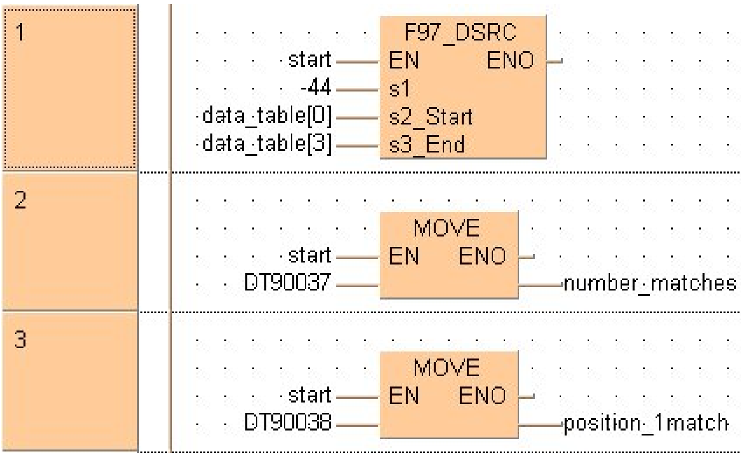
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_table	ARRAY [0..3] OF DINT	[-44,222222,-44,12345]	Arbitrarily large data field
2	VAR	number_matches	INT	0	result: here 2
3	VAR	position_1match	INT	0	result: here 0

Body

When the variable **start** is set to TRUE, the function is carried out. Instead of using an input variable in this example, a constant (-44) is assigned to input s1. The result is stored in special data registers DT90037 and DT90038. The two E_MOVE functions copy the results to the two variables **number_matches** and **position_1match**.

LD



ST

```
IF start THEN
    F97_DSRC( s1:= -44 ,
        s2_Start:= data_table[0] ,
        s3_End:= data_table[3] );
    number_matches:=DT90037;
    position1_match:=DT90038;
END_IF;
```

16.1 Introduction into the FIFO Buffer

The FIFO buffer is a first-in-first-out buffer area realized as a ring buffer. Data is stored in the order in which it is written to the buffer, and then read out in the order stored, starting from the first data item stored. It is convenient for buffering objects in sequential order.

Usage procedure

- The area to be used is defined as the FIFO buffer using the F115_FIFT (see page 474) instruction. (This should be done only once, before reading or writing is done.)
- Data should be written to the buffer using the F117_FIFW (see page 480) instruction, and read out of the buffer using the F116_FIFR (see page 477) instruction.

Writing data

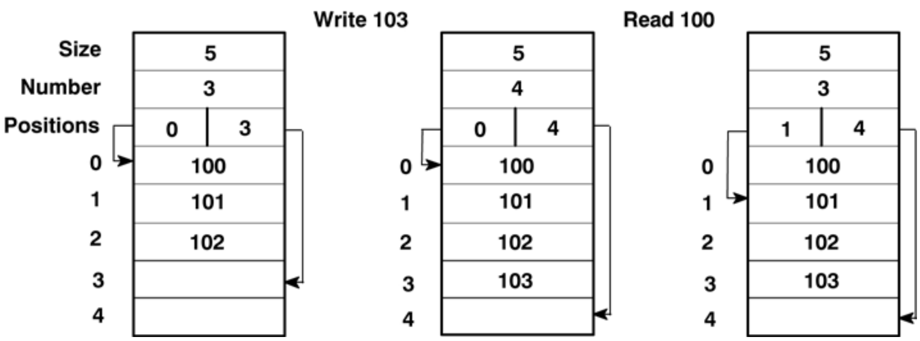
- When data is written, the data items are stored in sequential order, starting from the first data storage area. The writing pointer indicates the next area to which data is to be written. The number of words stored increases by 1.
- If the data storage area becomes full, i.e. the number of words stored is equal to n-1, further data writing is inhibited.

Reading data

- When data is read, data is transferred starting from the first data item stored. The reading pointer indicates the next area from which data is to be read. The number of words stored decreases by 1.
- An error occurs if an attempt is made to read data when the data storage area is empty, the number of words stored is equal to the memory size of the FIFO buffer or is equal to zero.

Data storage area

If data is written while the FIFO buffer is in the status shown below, the data will be stored in the area indicated by 3. The writing pointer moves to 4, i.e. the next data item will be written to 4. If data is read, it will be read from the area indicated by 0. The reading pointer then moves to 1, i.e. the next data item will be read from 1. (For more information on the reading and writing pointer, see F115_FIFT (see page 474)).



F115_FIFT — FIFO buffer area definition — Steps: 5

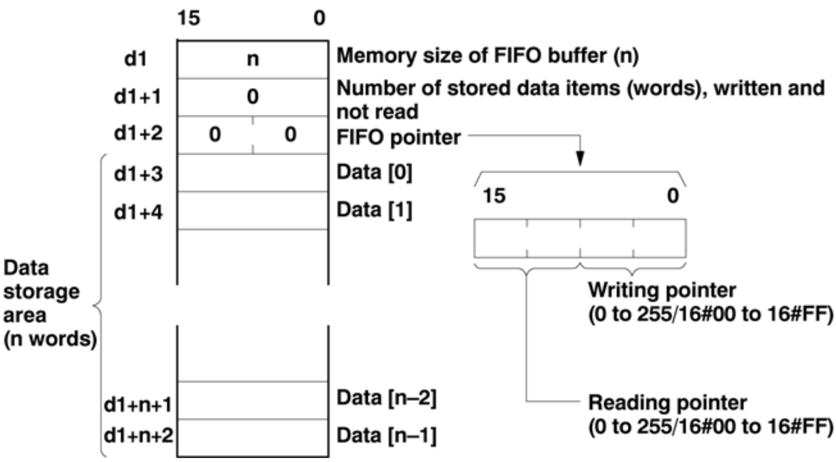
Description F/P115 specifies the starting area d1 for the FIFO (First-In-First-Out) buffer and the memory size n of the FIFO buffer.

n: memory size (number of words (16-bit)) of FIFO buffer, **n** = 1 to 256.

d1: the starting 16-bit area of FIFO buffer

How to use the FIFO buffer (see page 473)

Definition of the area using the FIFT instruction should be carried out only once, before writing to or reading from the FIFO buffer. When the FIFT instruction is executed, the FIFO buffer area is defined as follows:



When the FIFT instruction is executed, the following are stored as default values: d1 = n (the value specified by the FIFT instruction), d1 + 1 = 0, and d1 + 2 = 16#0000.

PLC types: **Availability of F115_FIFT (see page 928)**

Data types

Variable	Data type	Function
n	INT	specifies the memory size of FIFO buffer
d1	INT, WORD	starting 16-bit area of FIFO buffer

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- n = 0 - n > 256 - The area specified by n exceeds the limit
R9008	%MX0.900.8	for an instant	

Example In this example the function F115_FIFT is activated.

DUT The Data Unit Type is created in the DUT Pool. It can be accessed by the POU header after being declared there.

FIFO_n_WORD [DUT]

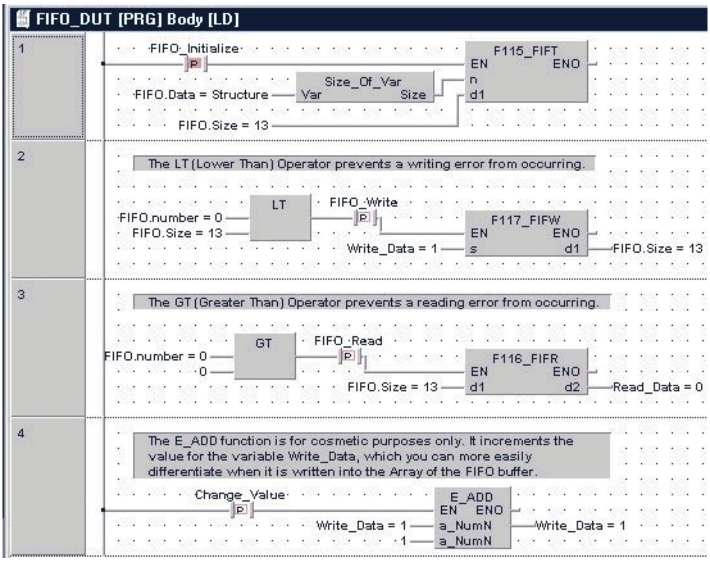
	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	FIFO	FIFO_n_WORD		
1	VAR	Read_Data	INT	0	
2	VAR	Write_Data	INT	1	
3	VAR	FIFO_Initialize	BOOL	FALSE	
4	VAR	FIFO_Write	BOOL	FALSE	
5	VAR	FIFO_Read	BOOL	FALSE	
6	VAR	Change_Value	BOOL	FALSE	

Body When the **FIFT_Initialize** instruction is enabled, the following values are stored as the default values: **FIFO.Size** = 13**; FIFO.Number** = 0; **FIFO.Positions** = 16#0000.

LD



ST

```
IF DF(FIFO_Initalize) THEN
    (* Create the FIFO buffer *)
    F115_FIFT( n_Number:= Size_Of_Var(FIFO.Data), d1_Start:=
FIFO.Size);
    REPEAT
        (* Initialize FIFO buffer with values *)
        Write_Data:=Write_Data+1;
        F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
    UNTIL(FIFO.Number>=FIFO.Size)
    END_REPEAT;
END_IF;

IF DF( FIFO_Write) THEN
    (* Write value of Write_Data to FIFO buffer *)
    (* at leading edge of FIFO_Write *)
    F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
END_IF;

IF DF(FIFO_Read) THEN
    (* Read value from FIFO buffer *)
    (* at leading edge of FIFO_Read *)
    F116_FIFR( d1_Start:= FIFO.Size, d2:= Read_Data);
END_IF;
```

F116_FIFR

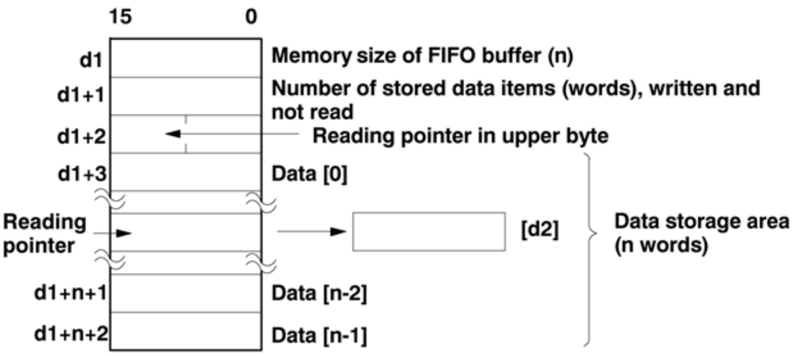
Read from FIFO buffer

Steps: 5

Description F/P116 reads the data **d1** from the FIFO (First-In-First-Out) buffer and stores the data in area specified by **d2**.

How to use the FIFO buffer (see page 473)

Reading of data is done starting from the address specified by the reading pointer when the instruction is executed.



- (0), (n–2) and (n–1) are addresses assigned to the data storage area.
- n is the value specified by the F115_FIFT (see page 474) instruction.

The reading pointer is stored in the upper eight bits of the third word of the FIFO buffer area. The actual address is the value of the leading address in the FIFO buffer area specified by d1 plus 3, plus the value of reading pointer (the value of which only the first byte is a decimal value).

When the reading is executed, 1 is subtracted from the number of stored data items, and the reading pointer is incremented by 1, or reset to zero if the reading pointer pointed to the final element.

PLC types: **Availability of F116_FIFR (see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area of FIFO buffer
d2	INT, WORD	16-bit area for storing data read from FIFO buffer

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Const.
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Part III F/P Instructions

- **An error occurs if this is executed when the number of stored data items is 0 or when the reading pointer is equal to the writing pointer.**
- **Reading is only carried out when the reading pointer is not equal to the writing pointer.**
- **If this is executed when the reading pointer is indicating the final address in the FIFO buffer (the n defined by the FIFO instruction minus 1), the reading pointer is set to 0.**

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	-The size (n) of the FIFO specified by d1 is n = 0, or when n > 256. -The number of stored data items of the FIFO = 0. -The number of stored data items of the FIFO > FIFO size (n). -The final address of the FIFO based on the FIFO size (n) exceeds the area. -The FIFO reading pointer > FIFO size (n). -The FIFO reading pointer is 256 (16#100) or higher after the data has been read.
R9008	%MX0.900.8	for an instant	

Example This example illustrates the FIFO buffer by incorporating the functions F115_FIFT, F116_FIFR and F117_FIFW.

DUT The Data Unit Type is created in the DUT Pool. It can be accessed by the POU header after being declared there.

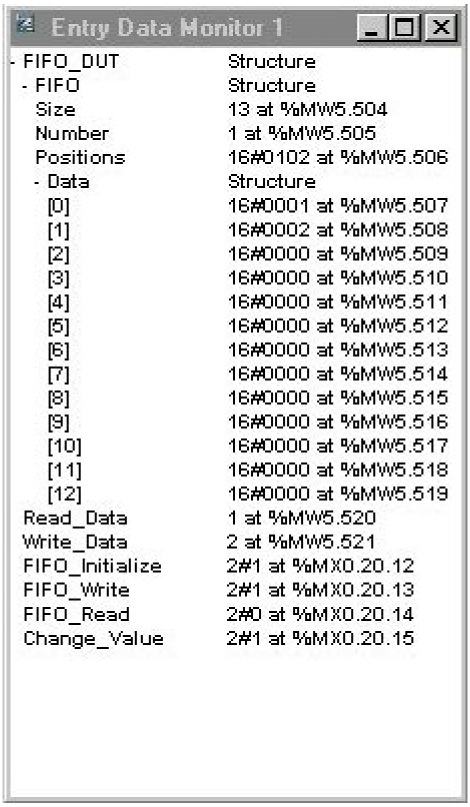
FIFO_n_WORD [DUT]

	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

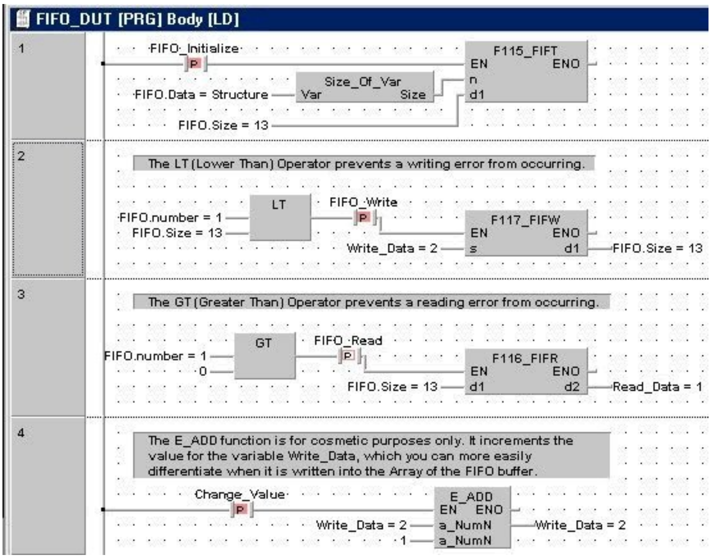
	Class	Identifier	Type	Initial	Comment
0	VAR	FIFO	FIFO_n_WORD		
1	VAR	Read_Data	INT	0	
2	VAR	Write_Data	INT	1	
3	VAR	FIFO_Initialize	BOOL	FALSE	
4	VAR	FIFO_Write	BOOL	FALSE	
5	VAR	FIFO_Read	BOOL	FALSE	
6	VAR	Change_Value	BOOL	FALSE	

Body The example below illustrates the status of the buffer after **FIFO_Write** has been enabled twice and **FIFO_Read** once. When **FIFO_Write** was activated the first time, the value 1 was written into **FIFO.Data[0]**. When **FIFO_Read** was enabled, **Read_Data** then read this value. When **FIFO_Write** was enabled the second time, the Writing pointer was incremented by one and the value 2 written into **FIFO.Data[1]**. see Entry Data Monitor 1


Entry Data Monitor 1

- FIFO_DUT	Structure
- FIFO	Structure
Size	13 at %MW5.504
Number	1 at %MW5.505
Positions	16#0102 at %MW5.506
- Data	Structure
[0]	16#0001 at %MW5.507
[1]	16#0002 at %MW5.508
[2]	16#0000 at %MW5.509
[3]	16#0000 at %MW5.510
[4]	16#0000 at %MW5.511
[5]	16#0000 at %MW5.512
[6]	16#0000 at %MW5.513
[7]	16#0000 at %MW5.514
[8]	16#0000 at %MW5.515
[9]	16#0000 at %MW5.516
[10]	16#0000 at %MW5.517
[11]	16#0000 at %MW5.518
[12]	16#0000 at %MW5.519
Read_Data	1 at %MW5.520
Write_Data	2 at %MW5.521
FIFO_Initialize	2#1 at %MX0.20.12
FIFO_Write	2#1 at %MX0.20.13
FIFO_Read	2#0 at %MX0.20.14
Change_Value	2#1 at %MX0.20.15

LD



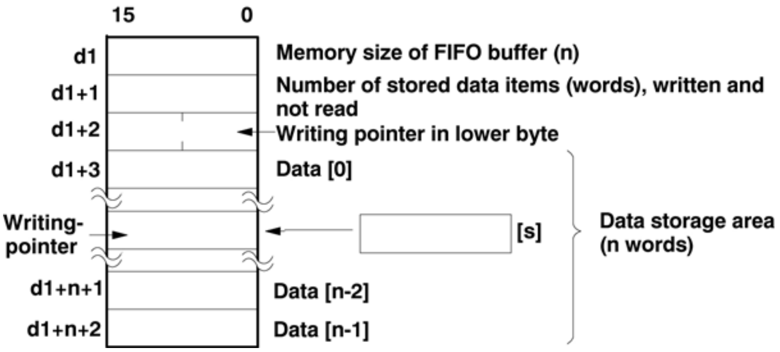
F117_FIFW Write to FIFO buffer

Steps: 5

Description F/P117 writes the data specified by **s** into the FIFO buffer specified by **d1**.

How to use the FIFO buffer (see page 473)

The specified data is written to the address indicated by the writing pointer when the instruction is executed.



- (0), (n-2) and (n-1) are addresses assigned to the data storage area.
- **n** is the value specified by the F115_FIFT (see page 474) instruction.

The writing pointer is stored in the lower eight bits of the third word of the FIFO buffer area, and is indicated by a relative position in the data storage area. The actual address to which data is being written is specified by d1 plus the offset 3 plus the value of the writing pointer (the value of which only the lower byte is a decimal value).

When the writing is executed, 1 is added to the number of stored data items, and the writing pointer is incremented by 1, or reset to zero if the writing pointer pointed to the final element.

PLC types: **Availability of F117_FIFW (see page 928)**

Data types

Variable	Data type	Function
s	INT, WORD	16-bit area or equivalent constant for storing data to write in the FIFO buffer
d1	INT, WORD	starting 16-bit area of FIFO buffer

The variables **s** and **d1** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	-The size (n) of the FIFO specified by d1 is n = 0, or when n > 256. -The number of stored data items of the FIFO = 0. -The number of stored data items of the FIFO > FIFO size (n). -The final address of the FIFO based on the FIFO size (n) exceeds the area. -The FIFO writing pointer > FIFO size (n).
R9008	%MX0.900.8	for an instant	-The FIFO writing pointer is 256 (16#100) or higher after the data has been written.

☞

- **An error occurs if this is executed when the FIFO buffer is full (the number of stored data items = the size n of the FIFO defined by the FIFT instruction). Writing is inhibited.**
- **If this is executed when the writing pointer is indicating the final address in the FIFO buffer (the "n" value defined by the FIFT instruction), the writing pointer will be set to 0.**

Example

This example illustrates the FIFO buffer by incorporating the functions F115_FIFT, F116_FIFR and F117_FIFW.

DUT

The Data Unit Type is created in the DUT Pool. It can be accessed by the POU header after being declared there.

FIFO_n_WORD [DUT]

	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	FIFO	FIFO_n_WORD		
1	VAR	Read_Data	INT	0	
2	VAR	Write_Data	INT	1	
3	VAR	FIFO_Initialize	BOOL	FALSE	
4	VAR	FIFO_Write	BOOL	FALSE	
5	VAR	FIFO_Read	BOOL	FALSE	
6	VAR	Change_Value	BOOL	FALSE	

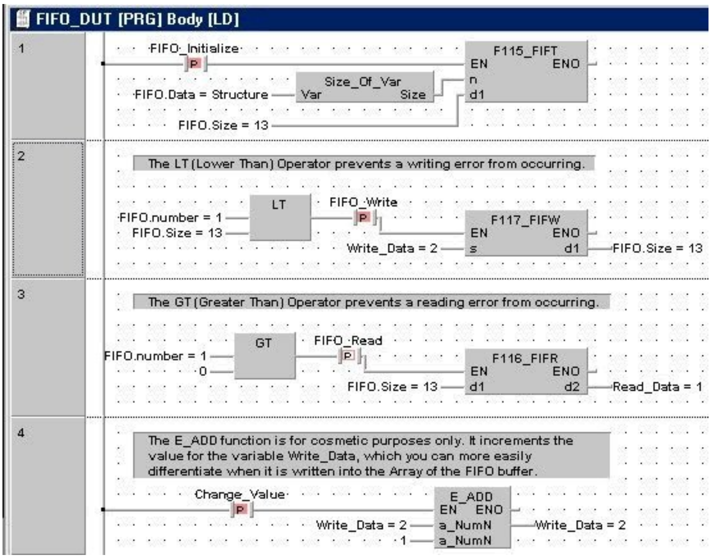
Body

The example below illustrates the status of the buffer after **FIFO_Write** has been enabled twice and **FIFO_Read** once. When **FIFO_Write** was activated the first time, the value 1 was written into **FIFO.Data[0]**. When **FIFO_Read** was enabled, **Read_Data** then read this value. When **FIFO_Write** was enabled the second time, the Writing pointer was incremented by one and the value 2 written into **FIFO.Data[1]**. see Entry Data Monitor 1

Entry Data Monitor 1

- FIFO_DUT	Structure
- FIFO	Structure
Size	13 at %MW5.504
Number	1 at %MW5.505
Positions	16#0102 at %MW5.506
- Data	Structure
[0]	16#0001 at %MW5.507
[1]	16#0002 at %MW5.508
[2]	16#0000 at %MW5.509
[3]	16#0000 at %MW5.510
[4]	16#0000 at %MW5.511
[5]	16#0000 at %MW5.512
[6]	16#0000 at %MW5.513
[7]	16#0000 at %MW5.514
[8]	16#0000 at %MW5.515
[9]	16#0000 at %MW5.516
[10]	16#0000 at %MW5.517
[11]	16#0000 at %MW5.518
[12]	16#0000 at %MW5.519
Read_Data	1 at %MW5.520
Write_Data	2 at %MW5.521
FIFO_Initialize	2#1 at %MX0.20.12
FIFO_Write	2#1 at %MX0.20.13
FIFO_Read	2#0 at %MX0.20.14
Change_Value	2#1 at %MX0.20.15

LD


FIFO_DUT [PRG] Body [LD]
1
FIFO_Initialize
P
F115_FIFT
EN ENO
n
d1
Size_Of_Var
Var Size
FIFO.Data = Structure
FIFO.Size = 13
2
The LT (Lower Than) Operator prevents a writing error from occurring.
LT
FIFO_Write
FIFO.number = 1
FIFO.Size = 13
F117_FIFW
EN ENO
s d1
Write_Data = 2
FIFO.Size = 13
3
The GT (Greater Than) Operator prevents a reading error from occurring.
GT
FIFO_Read
FIFO.number = 1
0
F116_FIFR
EN ENO
d1 d2
FIFO.Size = 13
Read_Data = 1
4
The E_ADD function is for cosmetic purposes only. It increments the value for the variable Write_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.
Change_Value
E_ADD
EN ENO
a_NumN
a_NumN
Write_Data = 2
1
Write_Data = 2

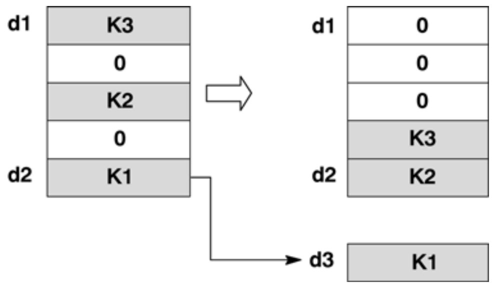
F98_CMPR — Data table shift-out and compress

Steps: 7

Description Shifts out non-zero data stored at the highest address of the table to the specified area and compresses the data in the table to the higher address. The data in the table specified by **d1** and **d2** is rearranged as follows:

- Contents of **d2** (highest address) are shifted out to the area specified by **d3**.

Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.



- Starting area **d1** and ending area **d2** should be the same type of operand.
- Be sure to specify **d1** and **d2** with "**d1** ≤ **d2**".

PLC types: **Availability of F98_CMPR (see page 927)**

Data types

Variable	Data type	Function
d1	ARRAY **of** WORD, INT	starting (lowest) address of data to be compressed
d2	ARRAY **of** WORD, INT	final (highest) address of data to be compressed, data at **d2** is shifted out
d3	WORD, INT	receives data shifted out from **d2**

Operands

For	Relay				T/C		Register			Const.
d1, d2, d3	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- d1 > d2
R9008	%MX0.900.8	for an instant	- d1 and d2 are not in the same memory area

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

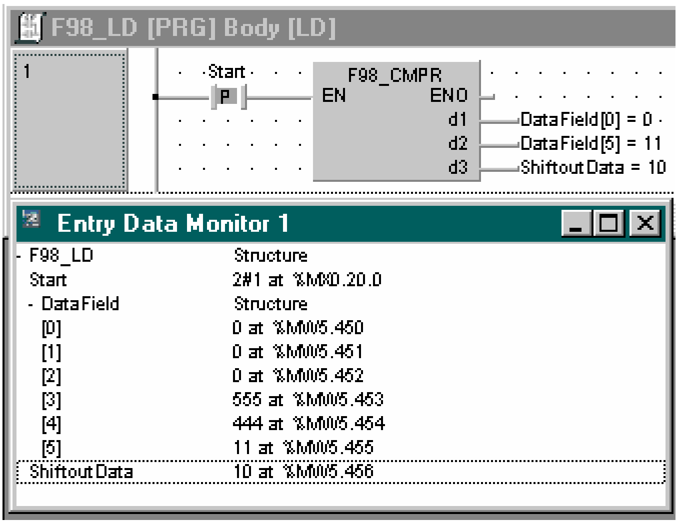
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DataField	ARRAY [0..5] OF INT	[555,444,0,11,0,10]	
2	VAR	ShiftoutData	INT	0	

Body

When the variable **Start** is set to TRUE, the function is carried out. The data in the lower addresses is compressed toward the higher addresses, and the value defined at the highest address, i.e. 10, is shifted out.

LD



Example 2: In combination with the F99_ CMPW/ P99_CMPW instruction, this can be used to construct an optional buffer. (Use a FIFO buffer for non-zero values.)

1. Executing the F99_CMPW/ P99_CMPW instruction
   When data items are written to the first address of the buffer (the area of the specified range), they are stored and accumulated in the buffer in sequential order. The oldest data will be stored in the last address of the buffer.
2. Executing the F98_CMPR/ P98_CMPR instruction
   When the data in the last address of the buffer (the area of the specified range) has been read, data can be extracted in sequential order, starting from the oldest data.

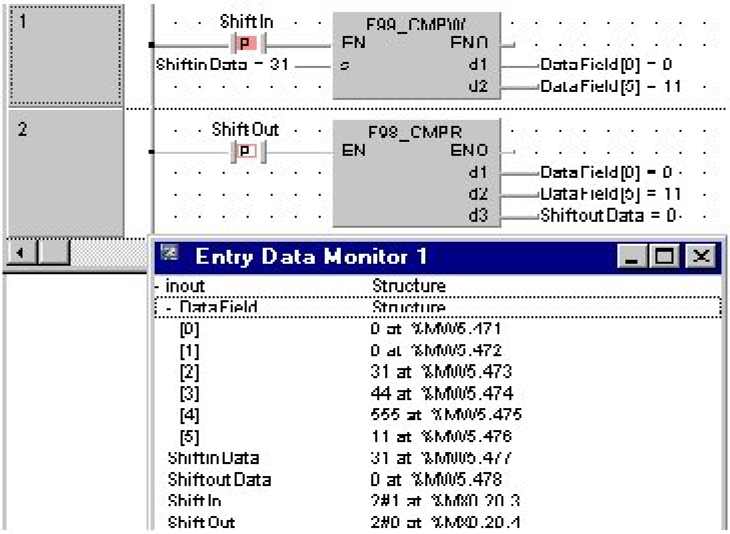
The rest of the data in the buffer is shifted in the direction of the first address, so normally, the oldest data at that point is stored in the last address of the buffer.

POU Header

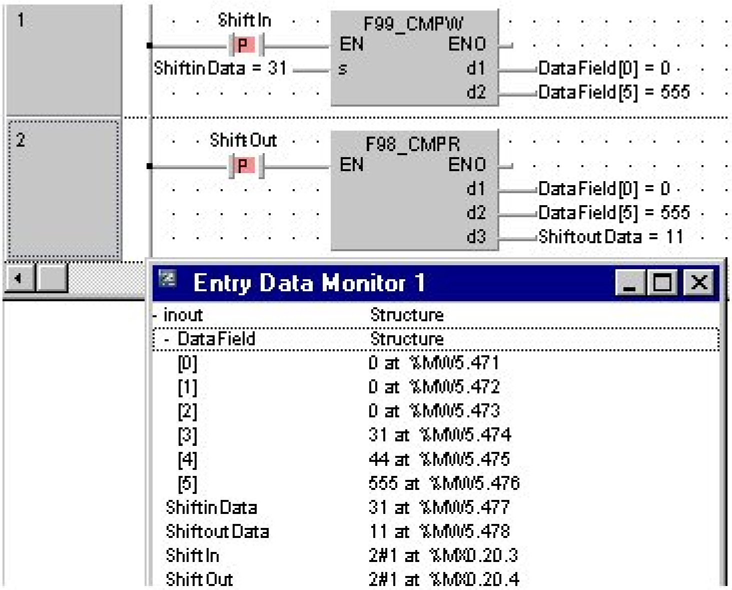
	Class	Identifier	Type	Initial	Comment
0	VAR	DataField	ARRAY [0..5] OF INT	[0,44,0,555,0,11]	
1	VAR	ShiftinData	INT	31	
2	VAR	ShiftoutData	INT	0	
3	VAR	ShiftIn	BOOL	FALSE	
4	VAR	ShiftOut	BOOL	FALSE	

LD

In Step 1 the F99 function is activated, shifting in the value given in the variable **ShiftinData** at s, i.e. 31, and compressing the rest of the data.



In Step 2 the F98 function is activated, and the value defined in the variable at d3, i.e. 11, is shifted out.



F99_CMPW

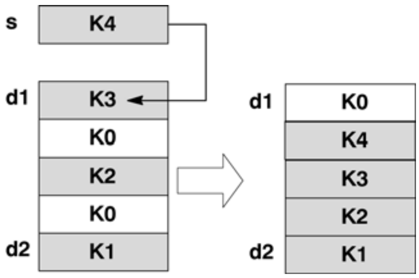
Data table shift-in and compress

Steps: 7

Description Shifts in data to the smallest address of the specified data table and compresses the data in the table toward the higher address. The data in the table specified by **d1** and **d2** is rearranged as follows:

- Data specified by **s** is shifted in to the area specified by **d1** (starting address).

Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.



- Starting area **d1** and ending area **d2** should be the same type of operand.
- Be sure to specify **d1** and **d2** with "**d1** ≤ **d2**".
- If the content of **s** is "0", only a compressed shift is carried out.

PLC types: **Availability of F99_CMPW (see page 927)**

☞ **For an example on how to construct a FIFO buffer using F/P99 and F/P98, see Example 2 from F/P98.**

Data types

Variable	Data type	Function
s	INT, WORD	data to be shifted in
d1	INT, WORD	starting address of area that is compressed into which data from **s** is shifted
d2	INT, WORD	end address of area where data is compressed

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- d1 > d2
R9008	%MX0.900.8	for an instant	- d1 and d2 are not in the same memory area

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

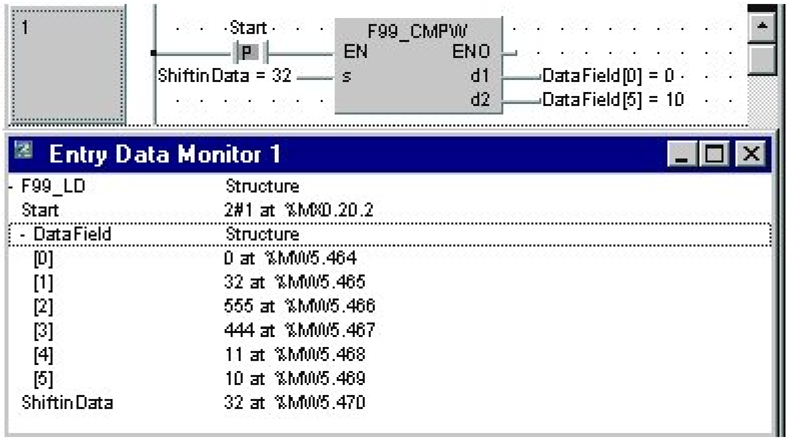
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DataField	ARRAY [0..5] OF INT	[555,444,0,11,0,10]	
2	VAR	ShiftinData	INT	32	

Body

After the variable **Start** is set to TRUE, the value of the variable **ShiftinData**, i.e. 32, at the contact s is shifted into the specified area of the data table, and the data is compressed.

LD



F277_SORT

Sort data in 16-bit data table (in smaller or larger number order)

Steps: 8

Description The function sorts values (with +/- sign) in a data table in ascending or descending order.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. You determine the sorting order at input **s3**.

At input **s3** you can enter the following values:

0 ascending order, i.e. begin with the smallest value

1 descending order, i.e. begin with the largest value

The data are sorted via bubble sort in the order specified according to the value entered at input **s1**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the address of the variable at input **s1 = s2**, no sorting takes place.

PLC types: **Availability of F277_SORT (see page 931)**

Data types

Variable	Data type	Function
s1	INT	starting area of data table to be sorted
s2	INT	ending area of data table to be sorted
s3	INT	specifies sorting order: 0 = ascending, 1 = descending

Operands

For	Relay				T/C		Register			Constant
s1, s2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2 - the address areas of the values at inputs s1 and s2 are different
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

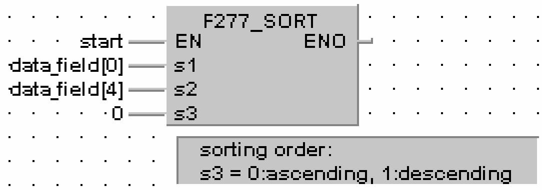
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field result: here [-3,1,2,3,6]

Body

When the variable **start** is set to TRUE, the function is carried out. The constant 0 is specified at input s3, which means the sorting is carried out in an ascending order. However, you can declare a variable in the POU header and write it in the function in the body at input s3.

LD



ST

```
IF start THEN
    F277_SORT( s1_Start:= data_field[0],
            s2_End:= data_field[4],
            s3_Descending:= 0);
END_IF;
```

Part III F/P Instructions

F278_DSORT

Sort data in 32-bit data table (in smaller or larger number order)

Steps: 8

Description The function sorts values (with +/- sign) in a data table in ascending or descending order.

Input **s1** specifies the starting area of the data table, and **s2** specifies the end. You determine the sorting order at input **s3**.

At input **s3** you can enter the following values:

0 ascending order, i.e. begin with the smallest value

1 descending order, i.e. begin with the largest value

The data are sorted via bubble sort in the order specified according to the value entered at input **s1**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the address of the variables at inputs **s1 = s2**, no sorting takes place.

PLC types: **Availability of F278_DSORT (see page 931)**

Although this is a 32-bit instruction, the number of steps is the same as the 16-bit instruction.

Data types

Variable	Data type	Function
s1	DINT	starting area of data table to be sorted
s2	DINT	ending area of data table to be sorted
s3	INT	specifies sorting order: 0 = ascending, 1 = descending

Operands

For	Relay				T/C		Register			Constant
s1, s2	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variable at input s1 > s2 - the address areas of the values at inputs s1 and s2 are different
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

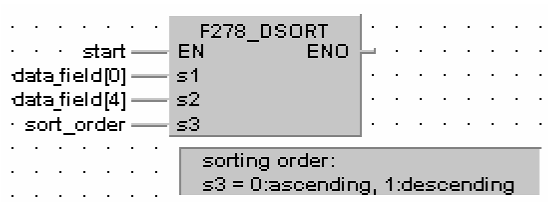
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field result: here [222222,3,2,1,-333333]
2	VAR	sort_order	INT	1	0:ascending, 1:descending

In this example, the input variable **sort_order** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. Since the variable **sort_order** is set to 1, the specified data field in sorted in descending order.

LD



ST

```
IF start THEN
    F278_DSORT( s1_Start:= data_field[0],
            s2_End:= data_field[4],
            s3_Descending:= sort_order);
END_IF;
```

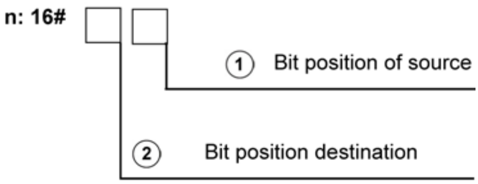
Chapter 17

Bitwise Boolean Instructions

F5_BTM **Bit data move** Steps: 7

Description 1 bit of the 16-bit data or constant value specified by **s** is copied to a bit of the 16-bit area specified by **d** according to the content specified by **n** if the trigger **EN** is in the ON-state. When the 16-bit equivalent constant is specified by **s**, the bit data move operation is performed internally converting it to 16-bit binary expression.

The operand **n** specifies the bit number as follows:



- Bit No. 0 to 3: source bit No. (16#0 to 16#F)
- Bit No. 8 to 11: destination bit No. (16#0 to 16#F)
- Bit No. 12 to 15: invalid

For example, reading from the right, n = 16#C01 would move from bit position one, one bit to bit position 12 (16#C).

PLC types: **Availability of F5_BTM (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	source 16-bit area
n	INT, WORD	specifies source and destination bit positions
d	INT, WORD	destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#8888 and bit at position 2 moves to destination value at bit position 15

source

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

target

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

result

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	0	1	1	1												

Bit at position 15 is exchanged, destination value in this example: 16#7FFF

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

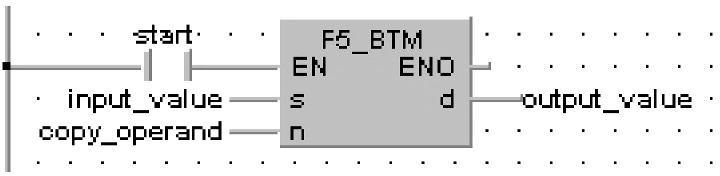
In the POU header, all input and output variables are declared that are used for programming this function.

	Klasse	Bezeichner	Typ	Initial	Kommentar
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1000100010001000	
2	VAR	copy_operand	WORD	16#0F02	digit no.1 and no.3 are invalid, digit no.0 locates the position of the source bit (here: 2), digit no.2 locates the position of the destination bit (here: 15)
3	VAR	output_value	WORD	2#1111111111111111	result after a 0->1 leading edge from start: 2#0111111111111111

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F5_BTM( s:= input_value,
        n:= copy_operand,
        d=> output_value);
END_IF;
```

F6_DGT Digit data move

Steps: 7

Description The hexadecimal digits in the 16-bit data or in the 16-bit equivalent constant specified by **s** are copied to the 16-bit area specified by **d** as specified by **n**.

Digits are units of 4 bits used when handling data. With this instruction, 16-bit data is separated into four digits. The digits are called in order hexadecimal digit 0, digit 1, digit 2 and digit 3, beginning from the least significant four bits:

	16-bit data			
bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 1
	hexadec. digit 3	hexadec. digit 2	hexadec. digit 1	hexadec. digit 0

n specifies the 3 source hexadecimal digit position, the 2 number of digits and the 1 destination hexadecimal digit position to be copied using hexadecimal data as follows:

n: 16# □ □ □

3 Source: Starting hexadecimal digit position

0 Hexadecimal digit 0
1 Hexadecimal digit 1
2 Hexadecimal digit 2
3 Hexadecimal digit 3

2 Number of hexadecimal digits to be copied

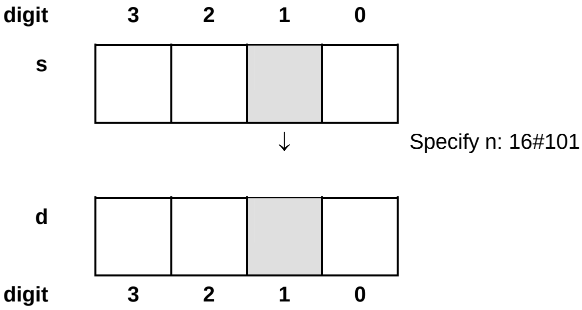
0 Copies 1 hexadecimal digits (4 bits)
1 Copies 2 hexadecimal digits (8 bits)
2 Copies 3 hexadecimal digits (12 bits)
3 Copies 4 hexadecimal digits (16 bits)

1 Destination: Starting hexadecimal digit position

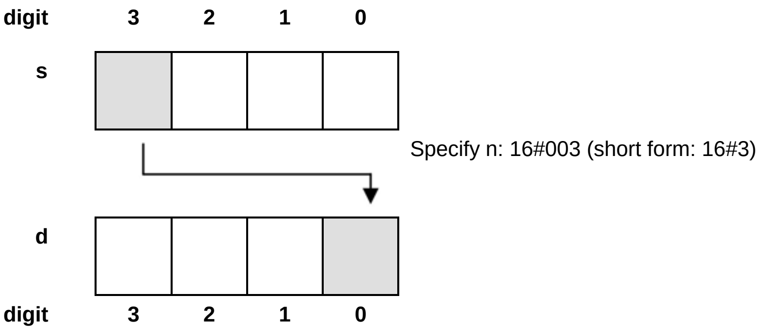
0 Hexadecimal digit 0
1 Hexadecimal digit 1
2 Hexadecimal digit 2
3 Hexadecimal digit 3

Following are some patterns of digit transfer based on the specification of n.

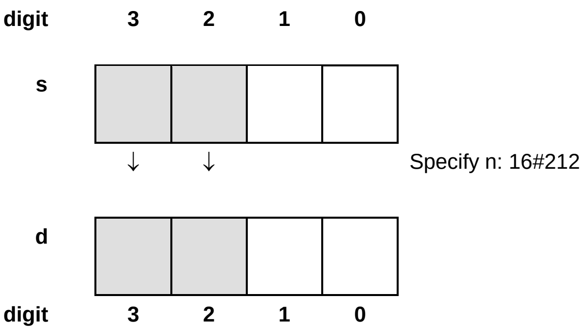
- When hexadecimal digit 1 of the source is copied to hexadecimal digit 1 of the destination:



- When hexadecimal digit 3 of the source is copied to hexadecimal digit 0 of the destination:

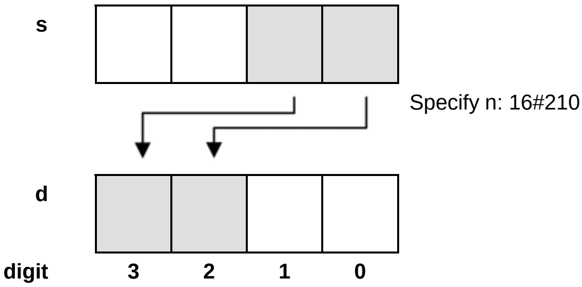


- When multiple hexadecimal digits (hexadecimal digits 2 and 3) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination:

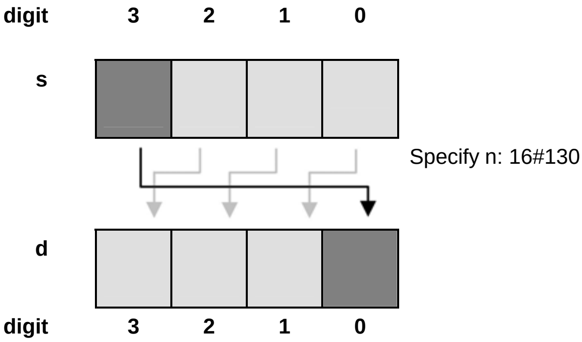


- When multiple hexadecimal digits (hexadecimal digits 0 and 1) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination:

digit 3 2 1 0



- When 4 hexadecimal digits (hexadecimal digits 0 to 3) of the source are copied to 4 hexadecimal digits (hexadecimal digits 0 to 3) of the destination:



PLC types: **Availability of F6_DGT** **(see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	16-bit area source
n	INT, WORD	Specifies source and destination hexadecimal digit position and number of hexadecimal digits
d	INT, WORD	16-bit area destination

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

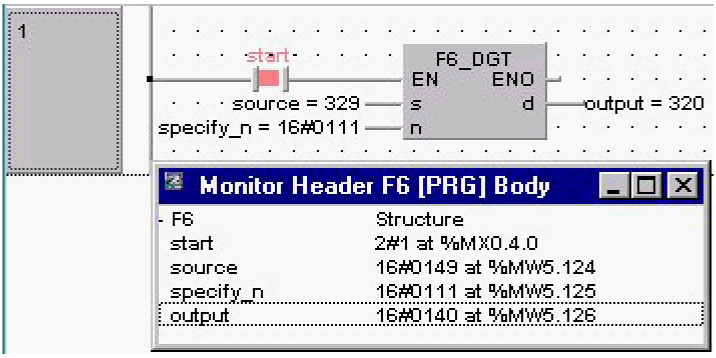
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	
1	VAR	source	INT	329	decimal 329 = hex. 149
2	VAR	specify_n	WORD	16#111	Beginning from the end: 1: first hexadecimal digit is digit 1, i.e. 4 1: copies 2 hexadecimal digits, i.e. 14 1: destination is hexadecimal digit 1
3	VAR	output	INT	0	hex. 140 = decimal 320

Body

When the variable **start** is set to TRUE, the function is executed. The values for **source** and **output** in the Monitor Header of the ladder diagram body have been set to display the hexadecimal value by activating the Hex button in the tool bar.

LD



ST

```
IF start THEN
    F6_DGT( s:= source,
        n:= specify_n,
        d=> output);
END_IF;
```

F65_WAN — 16-bit data AND

Steps: 7

Description Executes AND operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The AND operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the AND operation is performed internally converting it to 16-bit binary expression. You can use this instruction to turn OFF certain bits of the 16-bit data.

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1	0 1 0 0	1 1 0 1	1 0 1 1	1 0 0 1

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s2	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1

start: ON

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 0 0 0	0 0 0 0	1 0 1 1	1 0 0 1

PLC types: **Availability of F65_WAN (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit equivalent constant or 16-bit area
s2	INT, WORD	16-bit equivalent constant or 16-bit area
d	INT, WORD	16-bit area for storing AND operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F65_WAN is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

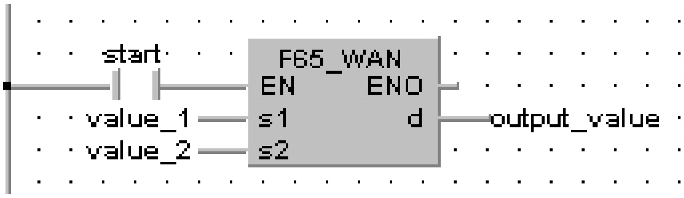
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000010001000

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F65_WAN(value_1, value_2, output_value);
END_IF;
```

Part III F/P Instructions

F66_WOR

16-bit data OR

Steps: 7

Description Executes OR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The OR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the OR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to turn ON certain bits of the 16-bit data.

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	1

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

start: ON

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
d	0	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1

PLC types: **Availability of F66_WOR (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit equivalent constant or 16-bit area
s2	INT, WORD	16-bit equivalent constant or 16-bit area
d	INT, WORD	16-bit area for storing OR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

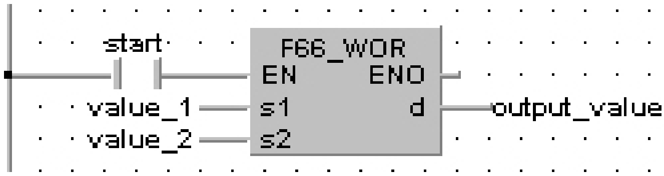
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000011101110

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F66_WOR(value_1, value_2, output_value);
END_IF;
```

F67_XOR

16-bit data exclusive OR

Steps: 7

Description Executes exclusive OR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The exclusive OR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the exclusive OR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to review the number of identical bits in the two 16-bit data.

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1	0 1 0 0	1 1 0 1	1 0 1 1	1 0 0 1

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s2	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1

start: ON

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 1 0 0	1 1 0 1	0 1 0 0	0 1 1 0

PLC types: **Availability of F67_XOR (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit equivalent constant or 16-bit area
s2	INT, WORD	16-bit equivalent constant or 16-bit area
d	INT, WORD	16-bit area for storing XOR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

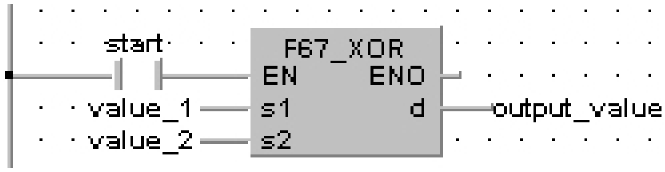
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0011000001100110

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F67_XOR(value_1, value_2, output_value);
END_IF;
```

F68_XNR

16-bit data exclusive NOR

Steps: 7

Description Executes exclusive NOR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The exclusive NOR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the exclusive NOR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to review the number of identical bits in the two 16-bit data.

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	1

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

start: ON

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
d	1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	1

PLC types: **Availability of F68_XNR (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit equivalent constant or 16-bit area
s2	INT, WORD	16-bit equivalent constant or 16-bit area
d	INT, WORD	16-bit area for storing NOR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F68_XNR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

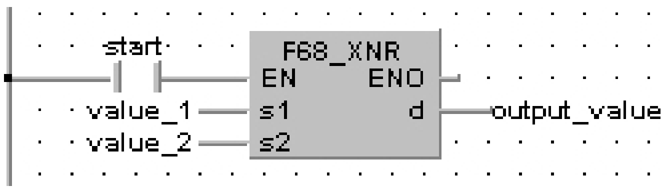
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#1100111110011001

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

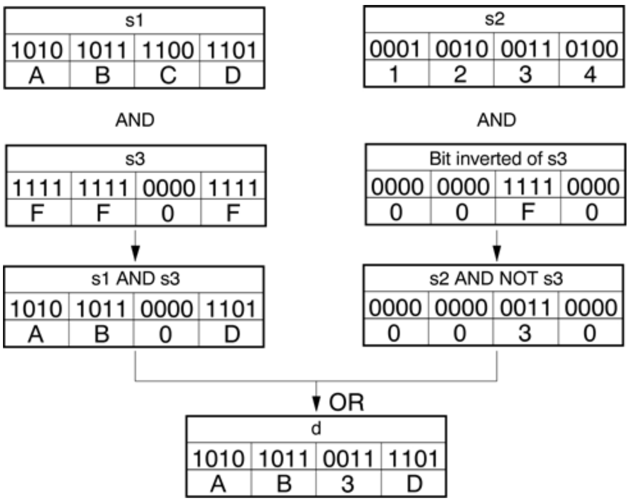
```
IF start THEN
    F68_XNR(value_1, value_2, output_value);
END_IF;
```

F69_WUNI 16-bit data unite

Steps: 9

Description The function combines the two values at inputs **s1** and **s2** with the value at input **s3** by bit-unit processing. The result of the function is returned at output **d**. The data-unite is calculated as follows:

[d] = ([s1] AND [s3]) OR ([s2] AND (NOT[s3]))



When the value at input **s3** = 16#0, the value at input **s2** is returned at output **d**.

When the value at input **s3** = 16#FFFF, the value at input **s2** is returned at output **d**.

PLC types: **Availability of F69_WUNI** **(see page 927)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit equivalent constant or 16-bit area
s2	INT, WORD	16-bit equivalent constant or 16-bit area
s3	INT, WORD	16-bit area that stores master data for combination or 16-bit equivalent constant data
d	INT, WORD	16-bit area for storing calculated result

The variables **s1, s2, s3** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

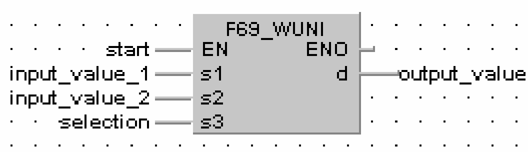
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	WORD	16#ABCD	
2	VAR	input_value_2	WORD	16#1234	
3	VAR	selection	WORD	16#FF0F	selection: 1 selects the bit from input_value_1 0 selects the bit from input_value_2
4	VAR	output_value	WORD	0	result: here 16#AB3D

In this example the input variables **input_value_1, input_value _2** and **selection** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F69_WUNI( s1:= input_value1,
         s2:= input_value2,
         s3_Mask:= selection,
         d=> output_value);
END_IF;
```

F215_DAND 32-bit data AND

Steps: 12

Description The function performs a bit-wise AND operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.

Truth Table:

s1	s2	d
0	0	0
0	1	0
1	0	0
1	1	1

PLC types: **Availability of F215_DAND (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit equivalent constant or 32-bit area
s2	DINT, DWORD	32-bit equivalent constant or 32-bit area
d	DINT, DWORD	32-bit area for storing AND operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#10204468

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF START THEN
    F215_DAND(dint1, dint2, dint3);
END_IF;
```

F216_DOR 32-bit data OR

Steps: 12

Description The function performs a bit-wise OR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.

Truth Table:

s1	s2	d
0	0	0
0	1	1
1	0	1
1	1	1

PLC types: **Availability of F216_DOR (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit equivalent constant or 32-bit area
s2	DINT, DWORD	32-bit equivalent constant or 32-bit area
d	DINT, DWORD	32-bit area for storing OR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

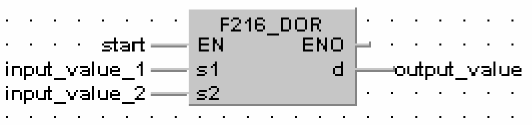
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#92BFDFFF

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F216_DOR(input_value_1, input_value_2, output_value);
END_IF;
```

F217_DXOR 32-bit data XOR

Steps: 12

Description The functions a bit-wise exclusive OR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.

Truth Table:

s1	s2	d
0	0	0
0	1	1
1	0	1
1	1	0

Using this instruction you can check how many bits in the two 32-bit data items are different, for example. At each position in which the bits at inputs **s1** and **s2** are different, a 1 is added in the result.

PLC types: **Availability of F217_DXOR (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit equivalent constant or 32-bit area
s2	DINT, DWORD	32-bit equivalent constant or 32-bit area
d	DINT, DWORD	32-bit area for storing XOR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

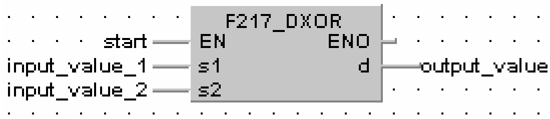
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#829F9B97

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F217_DXOR(input_value_1, input_value_2, output_value);
END_IF
```

F218_DXNR

32-bit data XNR

Steps: 12

Description The function performs a bit-wise exclusive NOR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.

Truth Table:

s1	s2	d
0	0	1
0	1	0
1	0	0
1	1	1

Using this instruction you can check how many bits in the two 32-bit data items are the same. At each position in which the bits at inputs **s1** and **s2** match, a 1 is produced in the result.

PLC types: **Availability of F218_DXNR (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit equivalent constant or 32-bit area
s2	DINT, DWORD	32-bit equivalent constant or 32-bit area
d	DINT, DWORD	32-bit area for storing XNR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated (output d) is 0.

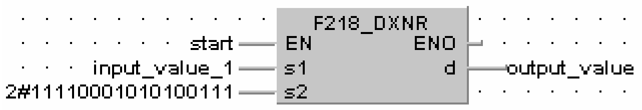
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	2#10101110101011110001	bit combination
2	VAR	output_value	DWORD	0	result: here 2#11111111111101001110111101010001

Body When the variable **output** is set to TRUE, the function F218_DXNR is carried out.

LD



ST

```
IF start THEN
    F218_DXNR(input_value_1, 2#1111000101010100111,
output_value);
END_IF;
```

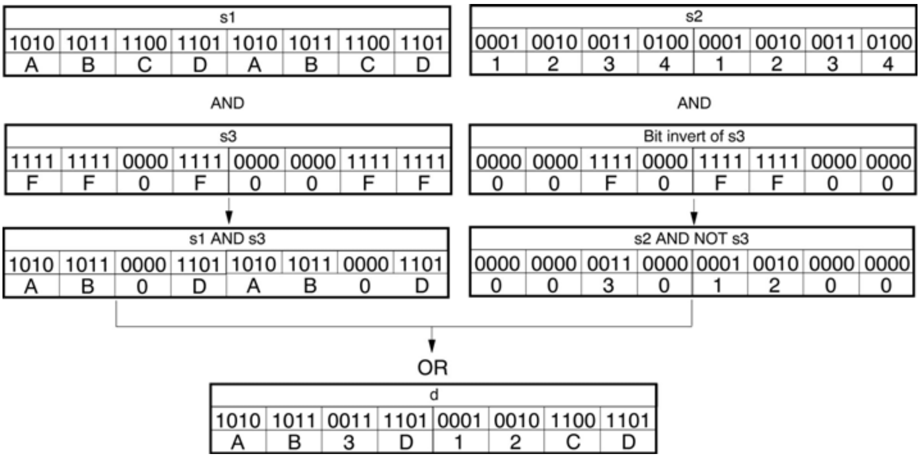
F219_DUNI

32-bit data unites 12

Steps: 16

Description The function combines the two values at inputs **s1** and **s2** bit-wise with the value at input **s3**. The result of the function is returned at output **d**. The data-unite is calculated as follows:

[d] = ([s1] AND [s3]) OR ([s2] AND (NOT[s3]))



When the value at input **s3** = 16#0, then the value at input **s2** is returned at output **d**.

When the value at input **s3** = 16#FFFFFFFF, then the value at input **s1** is returned at output **d**.

PLC types: **Availability of F219_DUNI (see page 930)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit equivalent constant or 32-bit area
s2	DINT, DWORD	32-bit equivalent constant or 32-bit area
s3	DINT, DWORD	32-bit area that stores master data for combination or 32-bit equivalent constant
d	DINT, DWORD	32-bit area for storing result

The variables **s1, s2, s3** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	- the result calculated (output d) is 0.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

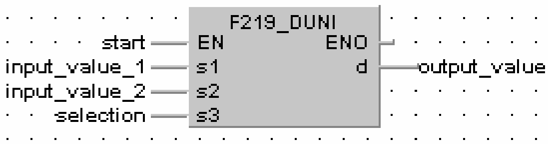
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#ABCDABCD	
2	VAR	input_value_2	DWORD	16#12341234	
3	VAR	selection	DWORD	16#FF0F00FF	selection: 1 selects the bit from input_value_1 0 selects the bit from input_value_2
4	VAR	output_value	DWORD	0	result: here 16#AB3D12CD

In this example the input variables **input_value_1, input_value _2** and **selection** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F219_DUNI( s1:= input_value1,
         s2:= input_value2,
         s3_Mask:= selection,
         d=> output_value);
END_IF;
```

Part III F/P Instructions

F130_BTS 16-bit data bit set

Steps: 5

Description Turns ON the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types: **Availability of F130_BTS** **(see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	specifies bit position to be set

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#101010	result after a 0->1 leading edge from start: 2#101011

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F130_BTS( n:= 0,
         d=> output_value);
END_IF;
```

F131_BTR

16-bit data bit reset

Steps: 5

Description Turns OFF the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types: **Availability of F131_BTR (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	specifies bit position to be reset

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function F131_BTR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#10101	result after a 0->1 leading edge from start: 2#10001

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F131_BTR( n:= 2,
         d=> output_value);
END_IF;
```

F132_BTI 16-bit data bit invert

Steps: 5

Description Inverts [1 (ON) → 0 (OFF) or 0 (OFF) → 1 (ON)] the bit at bit position **n** in the 16-bit data area specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types: **Availability of F132_BTI (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	specify bit position to be inverted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

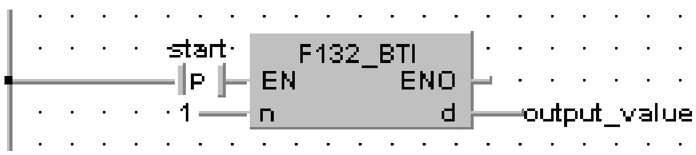
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#111	result after a 0->1 leading edge from start: 2#101

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F132_BTI( n:= 1,
         d=> output_value);
END_IF;
```

F133_BTT

16-bit data test

Steps: 5

Description Checks the state [1 (ON) or 0 (OFF)] of bit position **n** in the 16-bit data specified by **d** if the trigger **EN** is in the ON-state.

The specified bit is checked by special internal relay R900B.

- When specified bit is 0 (OFF), special internal relay R900B (=flag) turns ON.
- When specified bit is 1 (ON), special internal relay R900B (=flag) turns OFF.

n specifies the bit position to be checked in decimal data.
Range of **n**: 0 to 15

PLC types: **Availability of F133_BTT (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	specifies bit position to be tested

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function F133_BTT is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

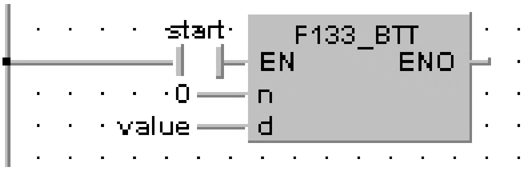
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit0_is_TRUE	BOOL	FALSE	TRUE if bit LSB of value is TRUE else FALSE
2	VAR	value	WORD	2#101	result after a 0->1 leading edge: 2#101 zero-flag (R900B) has state FALSE

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F133_BTT( n:= 0,
        d:= value);
    IF R900B THEN
        bit0_is_TRUE := FALSE;
    ELSE
        bit0_is_TRUE := TRUE;
    END_IF;
END_IF;
```

Part III F/P Instructions

F135_BCU Number of ON bits in 16-bit data

Steps: 5

Description Counts the number of bits in the ON state (1) in the 16-bit data specified by **s** if the trigger **EN** is in the ON-state.

The number of 1 (ON) bits is stored in the 16-bit area specified by **d**.

PLC types: **Availability of F135_BCU (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	source
n	INT	destination area for storing the number of bits in the ON (1) state

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function F135_BCU is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

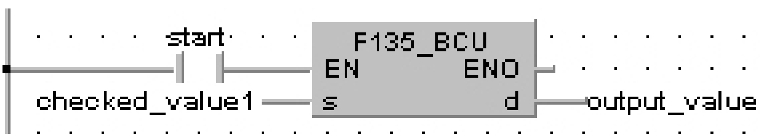
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value1	WORD	2#11011	this value will be checked for ON-bits
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 4

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F135_BCU(checked_value1, output_value);
END_IF;
```

Part III F/P Instructions

F136_DBCU

Number of ON bits in 32-bit data

Steps: 7

Description Counts the number of bits in the ON state (1) in the 32-bit data specified by **s** if the trigger **EN** is in the ON-state.

The number of 1 (ON) bits is stored in the 16-bit area specified by **d**.

PLC types: **Availability of F136_DBCU (see page 928)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source
d	INT	destination area for storing the number of bits in the ON (1) state

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value	DWORD	16#1111FFFF	this value will be checked for ON-bits
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 20

Body When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F136_DBCU(checked_value, output_value);
END_IF;
```

F84_INV

16-bit data invert (one's complement)

Steps: 5

Description Inverts each bit (0 or 1) of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The inverted result is stored in the 16-bit area specified by **d**. This instruction is useful for controlling an external device that uses negative logic operation.

Destination

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
d	0 1 0 1	1 1 1 0	1 0 1 1	1 1 0 1

start: ON

Destination

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
d	1 0 1 0	0 0 0 1	0 1 0 0	0 0 1 0

PLC types: **Availability of F84_INV (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be inverted

Operands

For	Relay				T/C		Register			Const.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

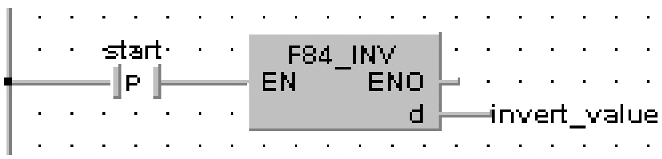
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	invert_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001110

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST
```
IF DF(start) THEN
    F84_INV(invert_value);
END_IF;
```

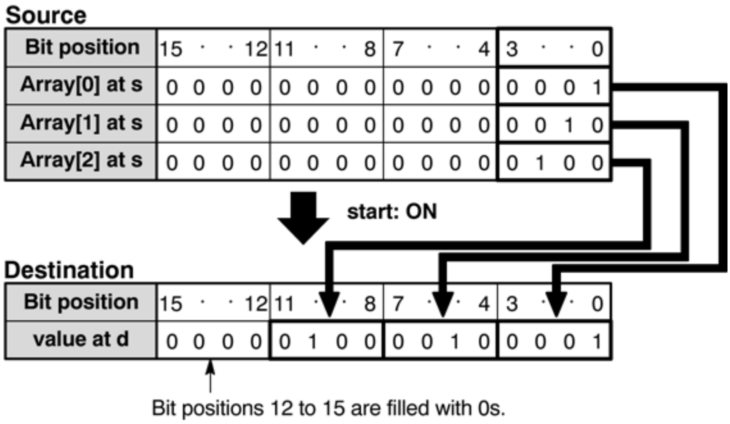
F93_UNIT 16-bit data combine

Steps: 7

Description Extracts each lower 4 bits (bit position 0 to 3) starting with the 16-bit area specified by **s** and combines the extracted data into 1 word if the trigger **EN** is in the ON-state. The result is stored in the 16-bit area specified by **d**.

n specifies the number of data to be extracted. The range of **n** is 0 to 4.

The programming example provided below can be envisioned thus:



PLC types: **Availability of F93_UNIT (see page 927)**

Data types

Variable	Data type	Function
s	WORD	starting 16-bit area to be extracted (source)
n	INT	specifies number of data to be extracted
d	WORD	16-bit area for storing combined data (destination)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the area specified using the index modifier exceeds the limit - the value at n ≥ 5
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

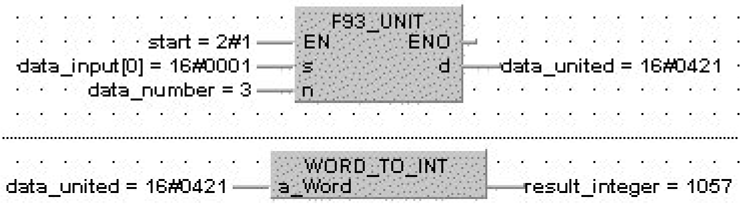
	Class	Identifier	Type	Initial
0	VAR	start	BOOL	TRUE
1	VAR	data_input	ARRAY [0..2] OF WORD	[1,2,4]
2	VAR	data_number	INT	3
3	VAR	data_united	WORD	0
4	VAR	result_integer	INT	0

Body

When the variable **start** is set to TRUE, the function is carried out. The binary values in the illustration on the main help page serve as the array values in **data_input**. In this example, variables are declared in the POU header. However, you may assign constants directly at the input function’s contact pins instead.

LD

In this example,  (Monitoring) was activated so you can see the results immediately.



F94_DIST

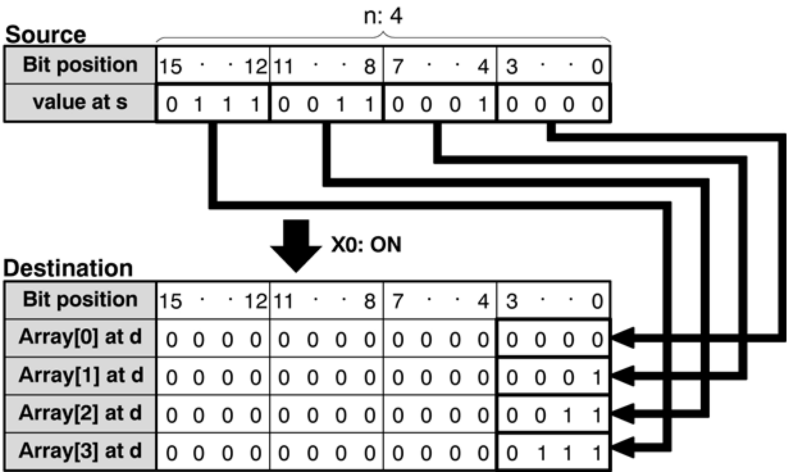
16-bit data distribution

Steps: 7

Description Divides the 16-bit data specified by **s** into 4-bit units and distributes the divided data into the lower 4 bits (bit position 0 to 3) of 16-bit areas starting with **d** if the trigger **EN** is in the ON-state.

n specifies the number of data to be divided. The range of **n** is 0 to 4. When 0 is specified by **n**, this instruction is not executed.

The programming example provided below can be envisioned thus:



PLC types: **Availability of F94_DIST (see page 927)**

Data types

Variable	Data type	Function
s	WORD	16-bit area or equivalent constant to be divided (source)
n	INT	specifies number of data to be divided
d	WORD	starting 16-bit area for storing divided data (destination)

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	-the area specified using the index modifier exceeds the limit -the value at n ≥ 5 the last area for the result exceeds the limit
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

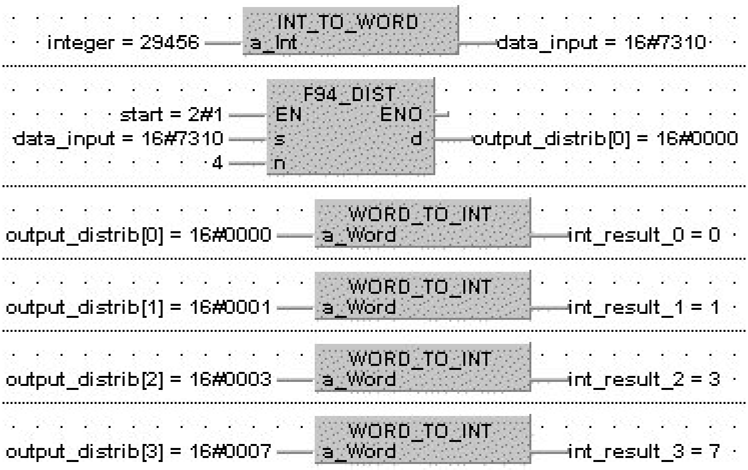
	Class	Identifier	Type	Initial
0	VAR	integer	INT	29456
1	VAR	data_input	WORD	0
2	VAR	start	BOOL	TRUE
3	VAR	output_distrib	ARRAY [0..3] OF WORD	[4(0)]
4	VAR	int_result_0	INT	0
5	VAR	int_result_1	INT	0
6	VAR	int_result_2	INT	0
7	VAR	int_result_3	INT	0

Body

When the variable **start** is set to TRUE, the function is carried out. The binary values in the illustration on main help page serve as the values calculated. In this example, variables are declared in the POU header. Also, a constant value of 4 is assigned directly at the contact pin for n.

LD

In this example,  (Monitoring) was activated so you can see the results immediately.



Chapter 18

Bitshift Instructions

LSR

Left shift register **Steps: 1**

Description Shifts 1 bit of the specified data area (WR) to the left (to the higher bit position). When programming the LSR instruction, be sure to program the data input (DataInput), shift (shiftTrigger) and reset triggers (ReSetTrigger).

DataInput: specifies the state of new shift-in data:

- new shift-in data 1: when the input is ON
- new shift-in data 0: when the input is OFF

shiftTrigger: shifts 1 bit to the left when the leading edge of the trigger is detected

ReSetTrigger: turns all the bits of the data area to 0 if the trigger is in the ON-state

The area available for this instruction is only the word internal relay (WR).

PLC types: **Availability of LSR (see page 933)**

Word internal relay (WR) number range, depends on the free area in the Extras → Options → Compile Options → Address Ranges menu.

Data types

Variable	Data type	Function
DataInput	BOOL	when ON, shift-in data = 1, when OFF, shift-in data = 0
shiftTrigger	BOOL	shifts one bit to the left when ON
ReSetTrigger	BOOL	resets data area to 0 when ON
WR	INT, WORD	specified data area where data shift takes place

Operands

For	Relay				T/C	
DataInput, shiftTrigger, ReSetTrigger	X	Y	R	L	T	C
d	-	-	WR	-	-	-

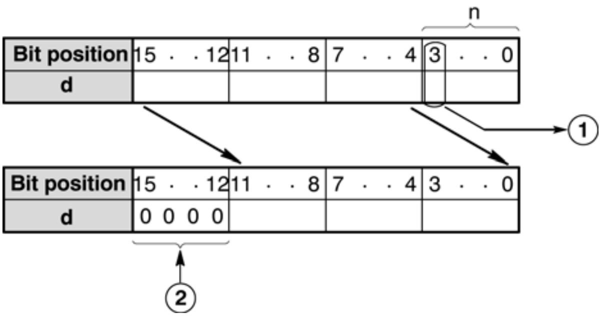
Example

F100_SHR

Right shift of 16-bit data in bit units

Steps: 5

Description Shifts **n** bits of 16-bit data area specified by **d** to the right (to the lower bit position) if the trigger **EN** is in the ON-state.



When **n** bits are shifted to the right, the data in the **n**th bit ① is transferred to special internal relay R9009 (carry-flag) and the higher **n** bits of the 16-bit data area ② specified by **d** are filled with 0s.

PLC types: **Availability of F100_SHR (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be shifted to the right
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

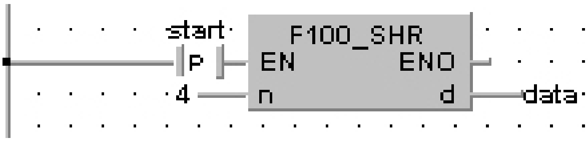
Example In this example the function F100_SHR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

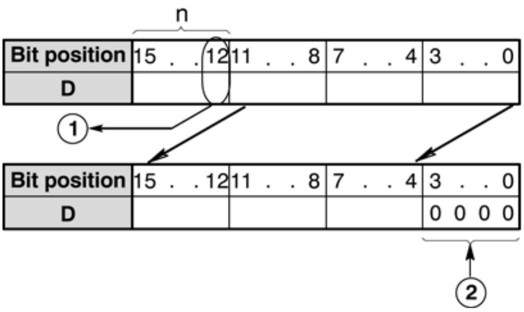
```
IF DF(start) THEN
    F100_SHR( n:= 4 ,
        d=> data );
END_IF;
```

F101_SHL

Left shift of 16-bit data in bit units

Steps: 5

Description Shifts **n** bits of 16-bit data area specified by **d** to the left (to the higher bit position) if the trigger **EN** is in the ON-state.



When **n** bits are shifted to the left, the data in the **n**th bit ① is transferred to special internal relay R9009 (carry-flag) and **n** bits ② starting with bit position 0 are filled with 0s.

PLC types: **Availability of F101_SHL (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be shifted to the left
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

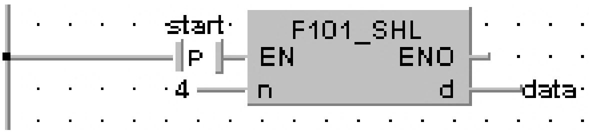
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



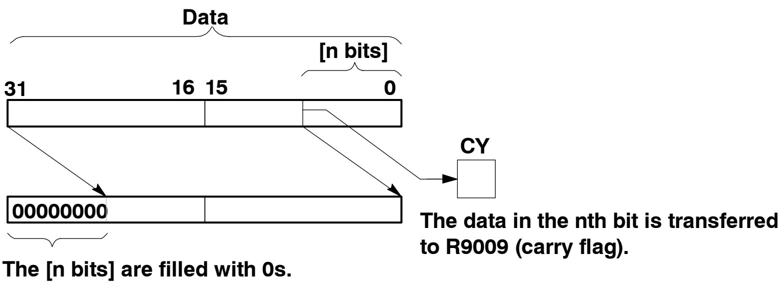
ST

```
IF DF(start) THEN
    F101_SHL( n:= 4,
         d=> data);
END_IF;
```

F102_DSHR

Right shift of 32-bit data in bit units **Steps: 5**

Description The function shifts the value at output **d** to the right. The number of bits at output **d** to be shifted to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. A shifting distance larger than 32 does not make sense, since when **n** = 32 the value at output **d** is already filled with zeros. The bit at position **n** - 1 (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly. When **n** = 0 the content of the carry flag does not change.



PLC types: **Availability of F102_DSHR**

Data types

Variable	Data type	Function
n	INT	number of bits to be shifted (range: 16#0 to 16#FF)
d	DINT, DWORD	32-bit area to be shifted to the right

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position n - 1 has the value 1.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

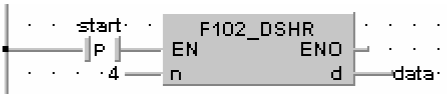
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge from start: 16#01234ABC

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the right. The 4 bits in **data** resulting from the shift are filled with zeros. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



ST

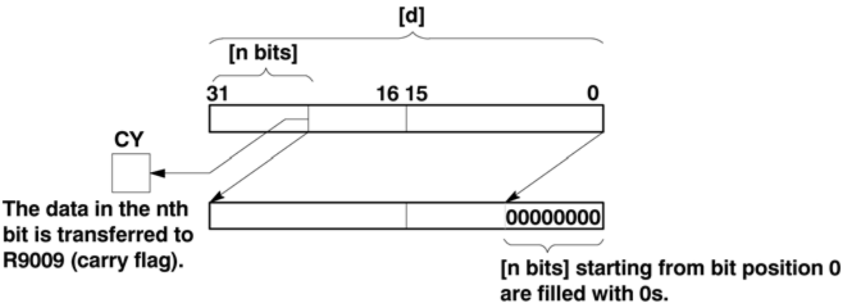
```
IF DF(start) THEN
    F102_DSHR( n:= 4 ,
         d=> data);
END_IF;;
```

F103_DSHL

Left shift of 32-bit data in bit units

Steps: 5

Description The function rotates the value at output **d** to the left. The number of bits at output **d** to be shifted to the left is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. A shifting distance larger than 32 does not make sense, since when **n** = 32 the value at output **d** is already filled with zeros. The bit at position 31 - **n** (the last bit shifted out to the left) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly. When **n** = 0 the content of the carry flag does not change.



PLC types: **Availability of F103_DSHL**

Data types

Variable	Data type	Function
n	INT	number of bits to be shifted (range: 16#0 to 16#FF)
d	DINT, DWORD	32-bit area to be shifted to the left

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position 31 - n has the value 1.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after ar 0->1 leading edge from start: 16#234ABCD0

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the left. The 4 bits in **data** resulting from the shift are filled with zeros. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD

ST

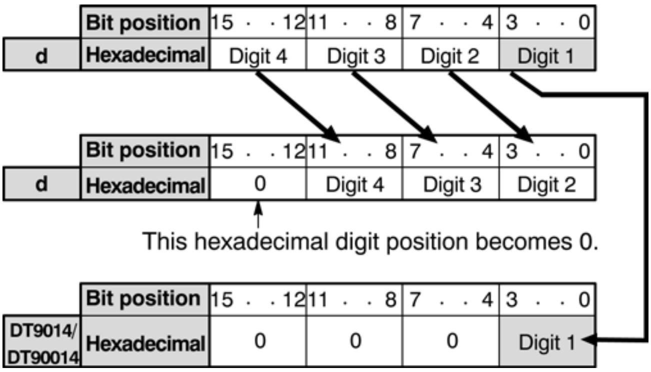
```
IF DF(start) THEN
    F103_DSHL( n:= 4,
         d=> data);
END_IF;
```

F105_BSR

Right shift of one hexadecimal digit (4 bits) of 16-bit data

Steps: 3

Description Shifts one hexadecimal digit (4 bits) of the 16-bit area specified by **d** to the right (to the lower digit position) if the trigger **EN** is in the ON-state.



When one hexadecimal digit (4 bits) is shifted to the right,

- hexadecimal digit position 0 (bit position 0 to 3) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014.
- hexadecimal digit position 3 (bit position 12 to 15) of the 16-bit area specified by **d** becomes 0.
- This instruction is useful when the hexadecimal or BCD data is handled.

PLC types: **Availability of F105_BSR (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be shifted to the right

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F105_BSR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

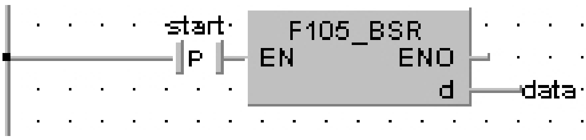
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

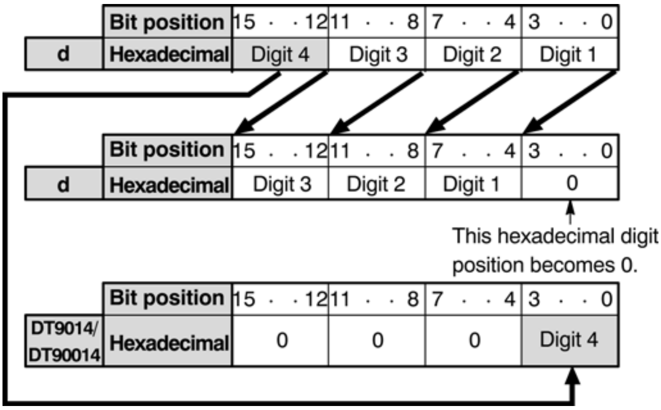
```
IF DF(start) THEN
    F105_BSR(data);
END_IF;
```

F106_BSL

Left shift of one hexadecimal digit (4 bits) of 16-bit data

Steps: 3

Description Shifts one hexadecimal digit (4 bits) of the 16-bit area specified by **d** to the left (to the higher digit position) if the trigger **EN** is in the ON-state.



- When one hexadecimal digit (4 bits) is shifted to the left,
- hexadecimal digit position 3 (bit position 12 to 15) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014.
- hexadecimal digit position 0 (bit position 0 to 3) of the 16-bit area specified by **d** becomes 0.

This instruction is useful when the hexadecimal or BCD data is handled.

PLC types: **Availability of F106_BSL (see page 927)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area to be shifted to the left

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

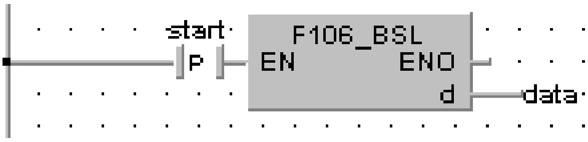
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

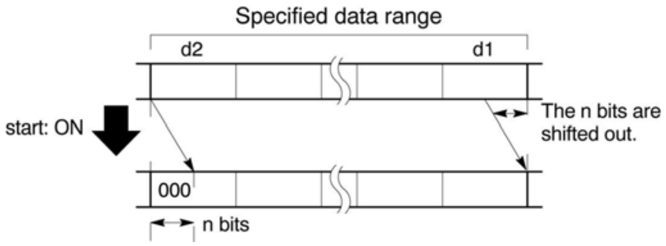
```
IF DF(start) THEN
    F106_BSL(data);
END_IF;
```

F108_BITR

Right shift of multiple bits of 16-bit data range

Steps: 7

Description The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the right. The number of bits by which the data range is to be shifted to the right is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. When input **n** = 16, a shift of one WORD occurs, i.e. the same process takes place as with function F110_WHSL (see page 551).



PLC types: **Availability of F108_BITR (see page 927)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area
n	INT	number of bits to be shifted

The addresses of the variables at inputs **d1** and **d2** have to have the same address type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variables at the outputs d1 > d2 or the value at input is n ≥ 16.
R9008	%MX0.900.8	for an instant	- the address of the variables at the outputs d1 > d2 or the value at input is n ≥ 16.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..2] OF WORD	[16#1234,16#ABCD,16#5678]	Arbitrarily large data field, result: after a 0->1 leading edge of start: data_field[0] = 16#D123 data_field[1] = 16#8ABC data_field[2] = 16#0567
2	VAR	number_bits	INT	4	

In this example, the input variable **number_bits** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the right. The 4 bits in **data_field[2]** resulting from the shift are filled with zeros.

LD

ST

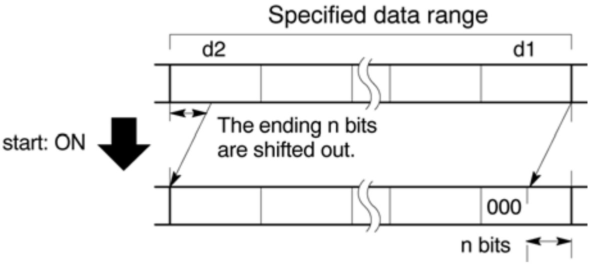
```
IF DF(start) THEN
    F108_BITR( n:=number_bits,
            d1_Start=> data_field[0],
            d2_End=> data_field[2]);
END_IF;
```

F109_BITL

Left shift of multiple bits of 16-bit data range

Steps: 7

Description The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the left. The number of bits by which the data range is to be shifted to the left is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. When input **n** = 16, a shift of one WORD occurs, i.e. the same process takes place as with function F111_WSHL (see page 553).



PLC types: **Availability of F109_BITL (see page 927)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area
n	INT	number of bits to be shifted

The addresses of the variables at inputs **d1** and **d2** have to have the same address type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the address of the variables at the outputs d1 > d2 or the value at input is n ≥ 16.
R9008	%MX0.900.8	for an instant	- the address of the variables at the outputs d1 > d2 or the value at input is n ≥ 16.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..2] OF WORD	[16#1234,16#ABCD,16#5678]	Arbitrarily large data field, result: after a 0->1 leading edge of start: data_field[0] = 16#2340 data_field[1] = 16#BCD1 data_field[2] = 16#678A

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the left. The 4 bits in **data_field[0]** resulting from the shift are filled with zeros. At input **n** the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD

ST

```
IF DF(start) THEN
    F109_BITL( n:=4,
         d1_Start=> data_field[0],
         d2_End=> data_field[2]);
END_IF;
```

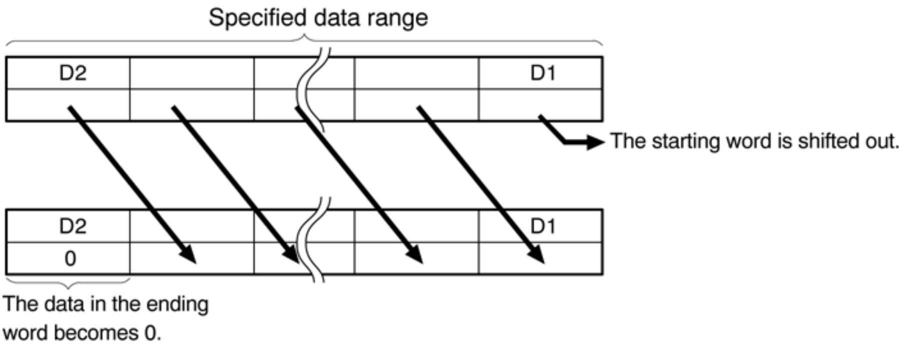
F110_WSHR

Right shift of one word (16 bits) of 16-bit data range

Steps: 5

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower word address) if the trigger **EN** is in the ON-state.

When one word (16 bits) is shifted to the right, the starting word is shifted out and the data in the ending word becomes 0.



d1 and **d2** should be:

- the same type of operand
- **d1** ≤ **d2**

PLC types: **Availability of F110_WSHR (see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

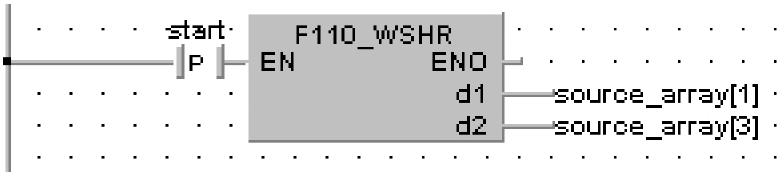
Example In this example the function F110_WSHR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,4,5,0]

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F110_WSHR( d1_Start=> source_array[1],
             d2_End=> source_array[3]);
END_IF;
```

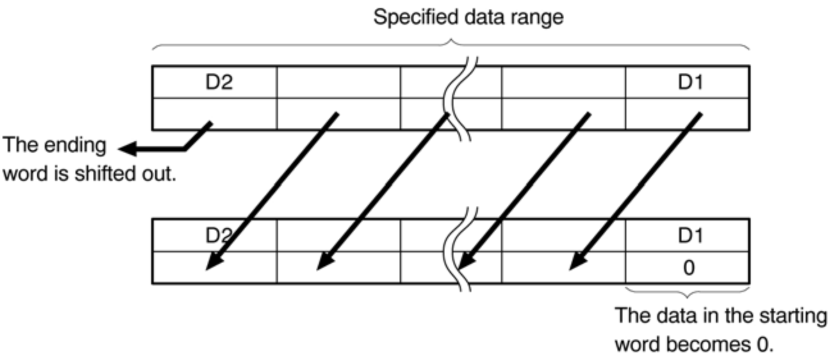
F111_WSHL

Left shift of one word (16 bits) of 16-bit data range

Steps: 5

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher word address) if the trigger **EN** is in the ON-state.

When one word (16 bits) is shifted to the left, the ending word is shifted out and the data in the starting word becomes 0.



d1 and **d2** should be:

- the same type of operand
- **d1** ≤ **d2**

PLC types: **Availability of F111_WSHL (see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

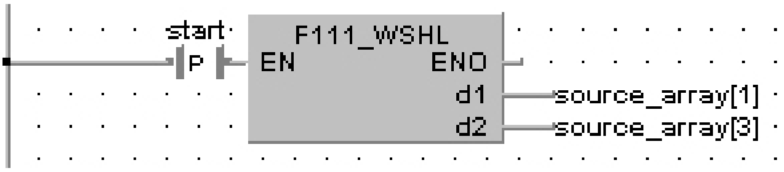
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,0,3,4]

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F111_WSHL( d1_Start=> source_array[1],
         d2_End=> source_array[3]);
END_IF;
```

F112_WBSR

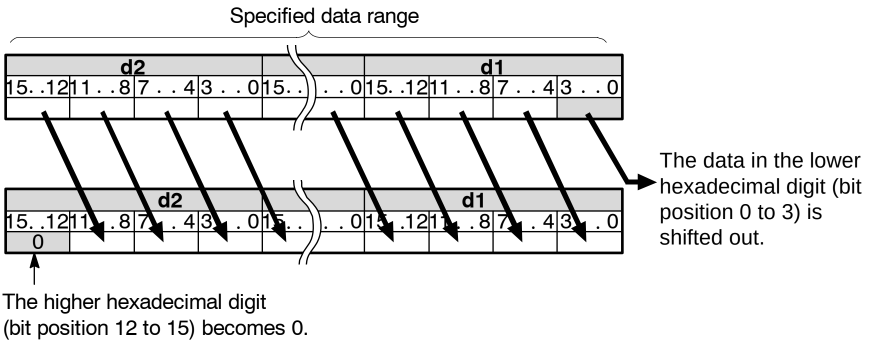
Right shift of one hex. digit (4 bits) of 16-bit 5 data range

Steps: 5

Description Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower digit position) if the trigger **EN** is in the ON-state.

When one hexadecimal digit (4 bits) is shifted to the right:

- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1** is shifted out.
- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2** becomes 0.



d1 and **d2** should be:

- the same type of operand
- **d1** ≤ **d2**

PLC types: **Availability of F112_WBSR** **(see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F112_WBSR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#8901, 16#4567,16#0123]

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

```
IF DF(start) THEN
    F112_WBSR( d1_Start=> source_array[1],
        d2_End=> source_array[3]);
END_IF;
```

F113_WBSL

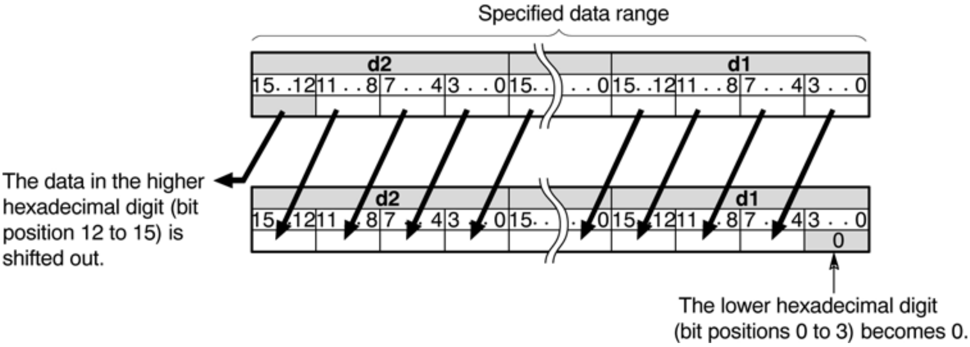
Left shift of one hex. digit (4 bits) of 16-bit data range

Steps: 5

Description Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher digit position) if the trigger **EN** is in the ON-state.

When one hexadecimal digit (4 bits) is shifted to the left,

- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2** is shifted out.
- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1** becomes 0.



d1 and **d2** should be:

- the same type of operand
- **d1** ≤ **d2**

PLC types: **Availability of F113_WBSL (see page 928)**

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

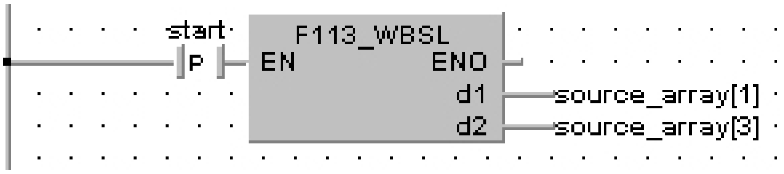
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#0120, 16#6789,16#2345]

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F112_WBSR( d1_Start=> source_array[1],
         d2_End=> source_array[3]);
END_IF;
```

F119_LRSR

LEFT/RIGHT shift register

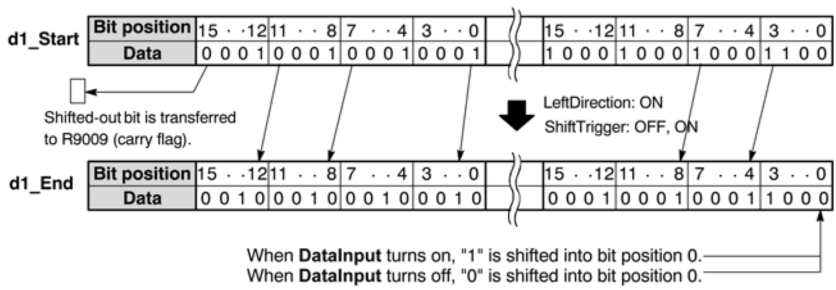
Steps: 5

Description Shifts 1 bit of the 16-bit data range to the left or to the right.

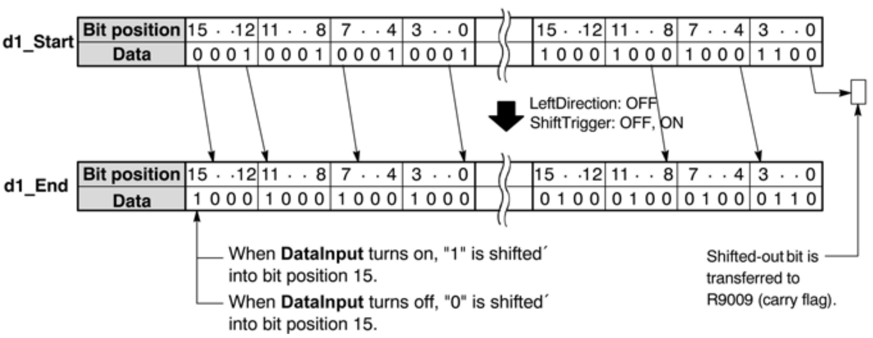
Left/right shift is a shift register which shifts 1 bit of the specified data area to the left (to the higher bit position) or to the right (to the lower bit position).

LeftDirection	Left/right trigger; specifies the direction of the shift-out.	
LeftDirection	= TRUE	shifting out to the left.
LeftDirection	= FALSE	shifting out to the right.
DataInput	Specifies the new shift-in data.	
	New shift-in data = TRUE:	when the data input is in the TRUE-state.
	New shift-in data = FALSE:	when the data input is in the FALSE-state.
ShiftTrigger	Shifts 1 bit to the left or right when the leading edge of the trigger is detected (FALSE → TRUE).	
Reset	Turns all the bits of the data range specified by **d1** and **d2** to 0 if this trigger is in the TRUE-state.	
d1	Start of 16-bit area.	
d2	End of 16-bit area.	
Carry	Shifted-out bit.	

Left shift operation



Right shift operation



PLC types: **Availability of F119_LRSR (see page 928)**

☞

- **The variables 'd1 and d2' have to be of the same data type.**
- **This function does not require a variable at the output "Carry".**

Data types

Variable	Data type	Function
LeftDirection	BOOL	specifies direction of shift, TRUE = left, FALSE = right
DataInput	BOOL	shift-in data, TRUE = 1, FALSE = 0
ShiftTrigger	BOOL	activates shift
Reset	BOOL	resets data in area specified by d1 and d2 to 0
Carry	BOOL	bit shifted out
d1	INT, WORD	starting 16-bit area
d2	INT, WORD	ending 16-bit area

Operands

For	Relay				T/C		Register			Constant
LeftDirection, DataInput, ShiftTrigger, Reset	X	Y	R	L	T	C	-	-	-	-
Carry	-	Y	R	L	T	C	-	-	-	-
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F119_LRSR is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	data_array	ARRAY [0..2] OF INT	[2#00000...	
1	VAR	enable_leftShift	BOOL	FALSE	function shifts left if TRUE,
2	VAR	reset	BOOL	FALSE	if TRUE, the whole array
3	VAR	input	BOOL	TRUE	specifies the new shift-in data
4	VAR	shift_trigger	BOOL	FALSE	activates the function at a 0->1
5	VAR	carry_out_value	BOOL	FALSE	result after a 0->1 leading edge

Body

When the variable **enable_leftShift** is set to TRUE, the function shifts left, else it shifts right.

LD

ST

```
carry_out_value:=F119_LRSR( LeftDirection:=
enable_leftShift,
         DataInput:= input,
         ShiftTrigger:= shift_trigger,
         Reset:= reset,
         d1_Start:= data_array[0],
         d1_End:= data_array[2]);
```

Part III F/P Instructions

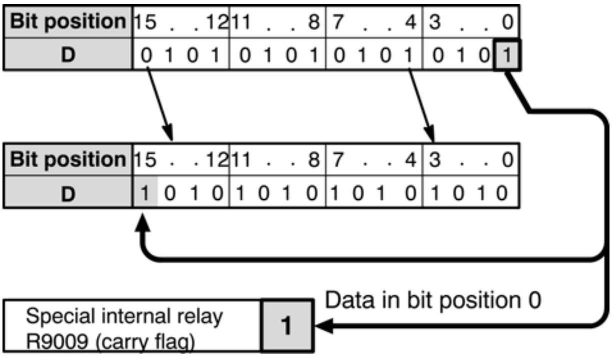
F120_ROR

16-bit data right rotate

Steps: 5

Description Rotates **n** bits of the 16-bit data specified by **d** to the right if the trigger **EN** is in the ON-state.

The following example rotates one bit to the right:



When **n** bits are rotated to the right,

- the data in bit position **n**-1 (**n**th bit starting from bit position 0) is transferred to the special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 0 are shifted out to the right and into the higher bit positions of the 16-bit data specified by **d**.

PLC types: **Availability of F120_ROR (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

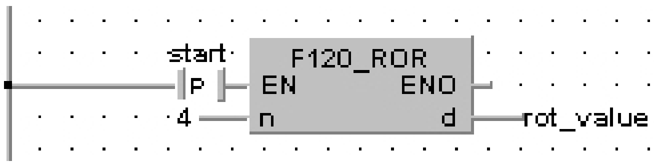
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#4123

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F120_ROR( n:= 4,
         d=> rot_value);
END_IF;
```

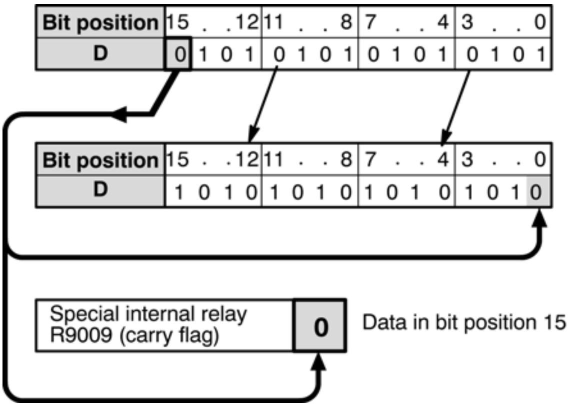
F121_ROL

16-bit data left rotate

Steps: 5

Description Rotates **n** bits of the 16-bit data specified by **d** to the left if the trigger **EN** is in the ON-state.

The following example rotates one bit to the left:



When **n** bits are rotated to the left,

- the data in bit position 16-**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 15 are shifted out to the left and into the lower bit positions of the 16-bit data specified by **d**.

PLC types: **Availability of F121_ROL (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function F121_ROL is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2341

Body When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

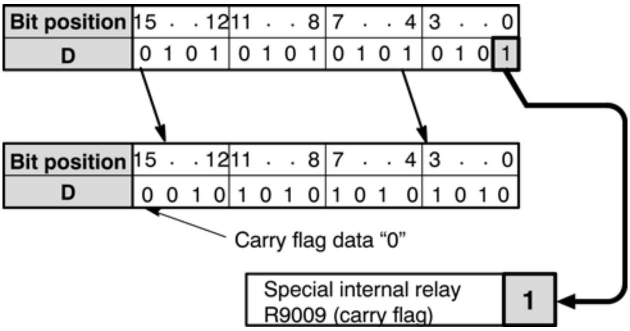
```
IF DF(start) THEN
    F121_ROL( n:= 4,
        d=> rot_value);
END_IF;
```

F122_RCR 16-bit data right rotate with carry-flag data

Steps: 5

Description Rotates **n** bits of the 16-bit data specified by **d** including the data of carry-flag to the right if the trigger **EN** is in the ON-state.

This example rotates one bit to the right:



When **n** bits with carry-flag data are rotated to the right,

- the data in bit position **n**-1 (**n**th bit starting from bit position 0) are transferred to special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 0 are shifted out to the right and carry-flag data and **n**-1 bits starting from bit position 0 are subsequently shifted into the higher bit positions of the 16-bit data specified by **d**.

PLC types: **Availability of F122_RCR (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

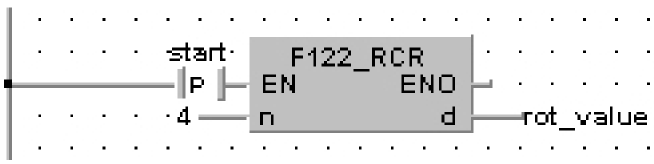
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#8123 (!) (carry flag)

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
    F122_RCR( n:= 4,
            d=> rot_value);
END_IF;
```

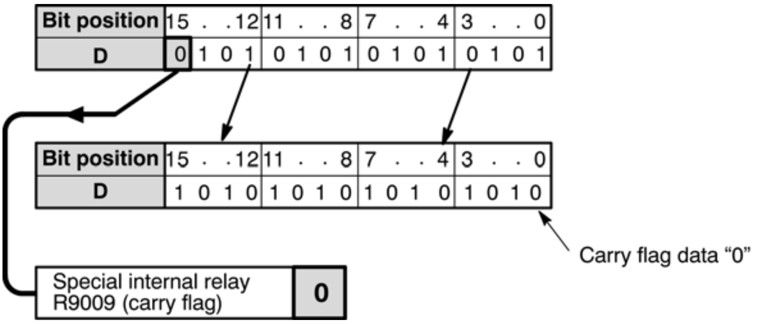
F123_RCL

16-bit data left rotate with carry-flag data

Steps: 5

Description Rotates **n** bits of the 16-bit data specified by **d** including the data of carry-flag to the left if the trigger **EN** is in the ON-state.

This example rotates one bit to the left:



When **n** bits with carry-flag data are rotated to the left,

the data in bit position 16-**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry-flag).

n bits starting from bit position 15 are shifted out to the left and carry-flag data and **n**-1 bits starting from bit position 15 are shifted into lower bit positions of the 16-bit data specified by **d**.

PLC types: **Availability of F123_RCL (see page 928)**

Data types

Variable	Data type	Function
d	INT, WORD	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function F125_RCL is programmed is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340 (!) (carry flag)

Body

When the variable **start** changes from FALSE to TRUE, the function is executed.

LD

ST

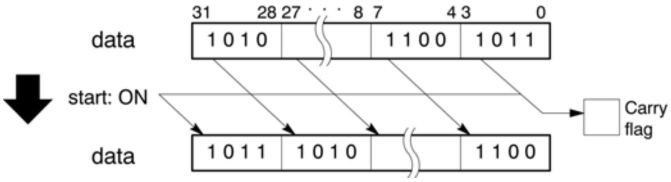
```
IF DF(start) THEN
    F123_RCL( n:= 4,
            d=> rot_value);
END_IF;
```

F125_DROR

32-bit data right rotate

Steps: 5

Description The function rotates the value at output **d** to the right. The number of bits at output **d** to be rotated to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Right rotate means that the bits shifted out of bit position 0 (LSB) are shifted via bit position 31 (MSB) into the value at output **d.** When input **n** = 0, no rotation takes place. When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 32 produces the same result as when **n** = 0; **n** = 33 the same as **n** = 1. The bit at position **n - 1** (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly.



PLC types: **Availability of F125_DROR** **(see page 928)**

Data types

Variable	Data type	Function
n	INT	number of bits to be rotated (range: 0 to 255)
d	DINT, DWORD	32-bit area

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position n - 1 of d has the value 1.

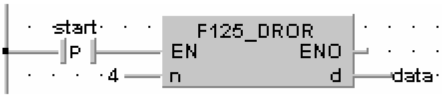
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16#D1234ABC

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It rotates 4 bits (corresponds to one position in a hexadecimal representation) to the right. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



ST

```
IF DF(start) THEN
    F125_DROR( n:= 4,
         d=> data);
END_IF;
```

F126_DROL

32-bit data left rotate

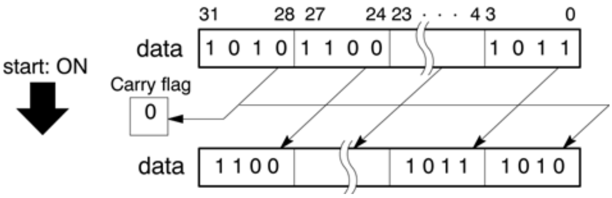
Steps: 5

Description The function rotates the value at output d to the left. The number of bits at output d to be rotated to the left is specified by the value assigned at input n. This shift can lie between 0 and 255 (only the lower value byte of n is effective).Left rotate means that the bits shifted out of bit position 31 (MSB) are shifted via bit position 0 (LSB) into the value at output d.

When input n = 0, no rotation takes place.

When at input n > 32, the same result is achieved as with a number n < 32: e.g. n = 33 produces the same result as when n = 0; n = 34 the same as n = 1.

The bit at position 32 - n (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly.



PLC types: **Availability of F126_DROL (see page 928)**

Data types

Variable	Data type	Function
n	INT	number of bits to be rotated (range: 0 to 255)
d	DINT, DWORD	32-bit area

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position 32 - n of d has the value 1.

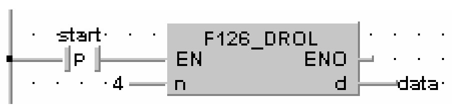
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of output: 16#234ABCD1

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It rotates 4 bits (corresponds to one position in a hexadecimal representation) to the left. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



ST

```
IF DF(start) THEN
    F126_DROL( n:= 4,
         d=> data);
END_IF;
```

Part III F/P Instructions

F127_DRCR

32-bit data right rotate with carry flag data

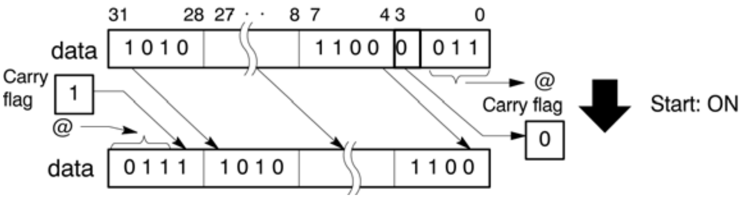
Steps: 5

Description The function rotates the value at output **d** via the carry flag to the right. The number of bits at output **d** to be rotated to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective).

The bit value at bit position **n** - 1 is stored in the carry flag. The function shifts out **n** bits from bit 0 to the right, and then along with the inverted carry flag first, continues via bit 31 into the higher bit positions. Position 32 - **n** now has the inverted value of the carry flag.

When input **n** = 0, no rotation occurs and the carry flag remains unchanged.

When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 33 produces the same result as when **n** = 0; **n** = 34 the same as **n** = 1.



PLC types: **Availability of F127_DRCR** **(see page 928)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit data area
n	INT	number of bits to be rotated (range: 0 to 255)

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position n - 1 has the value 1.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

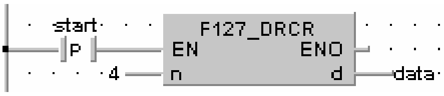
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16#A1234ABC

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. In this example the constant (4) is assigned to the function at input n. You may, however, declare a variable in the POU header instead.

LD



ST

```
IF DF(start) THEN
    F127_DRCR( n:= 4,
        d=> data);
END_IF;
```

F128_DRCL

32-bit data right rotate with carry flag data

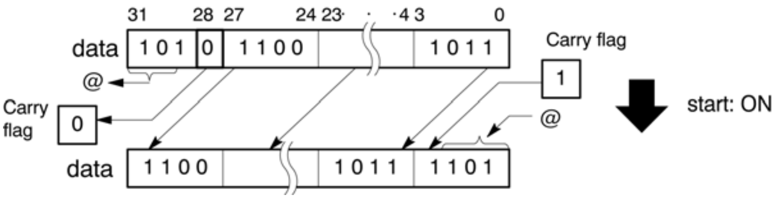
Steps: 5

Description The function rotates the value at output **d** via the carry flag to the left. The number of bits at output **d** to be rotated to the left is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective).

The bit value at bit position 32 - **n** is stored in the carry flag. The function shifts out **n** bits to the left via bit 31 (MSB), and then along with the inverted carry flag first, continues via bit 0 (LSB) into the storage range. Position **n** - 1 now has the inverted value of the carry flag.

When input **n** = 0, no rotation occurs and the carry flag remains unchanged.

When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 33 produces the same result as when **n** = 0; **n** = 34 the same as **n** = 1.



PLC types: **Availability of F128_DRCL (see page 928)**

Data types

Variable	Data type	Function
d	DINT, DWORD	32-bit area
n	INT	number of bits to be rotated (range: 0 to 255)

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	- the bit at position 32 - n has the value 1.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

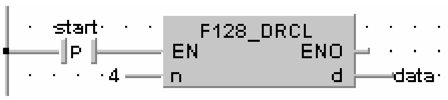
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16#234ABCD0

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. In this example the constant (4) is assigned to the function at input n. You may, however, declare a variable in the POU header instead.

LD



ST

```
IF DF(start) THEN
    F128_DRCL( n:= 4,
        d=> data);
END_IF;
```

Chapter 19

Comparison Instructions

F60_CMP — 16-bit data compare

Steps: 5

Description Compares the 16-bit data specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON-state. The compare operation result is stored in special internal relays R9009, R900A to R900C.

Data	Comparison between s1 and s2	Flags			
		R900A (>flag)	R900B (=flag)	R900C (<flag)	R9009 (carry-flag)
16-bit data with sign	s1<s2	OFF	OFF	ON	#
	s1=s2	OFF	ON	OFF	OFF
	s1>s2	ON	OFF	OFF	#
16-bit data without sign	s1<s2	#	OFF	#	ON
	s1=s2	OFF	ON	OFF	OFF
	s1>s2	#	OFF	#	OFF

turns ON or OFF depending on the conditions

PLC types: **Availability of F60_CMP (see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit area or 16-bit equivalent constant to be compared
s2	INT, WORD	16-bit area or 16-bit equivalent constant to be compared

The variables **s1** and **s2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

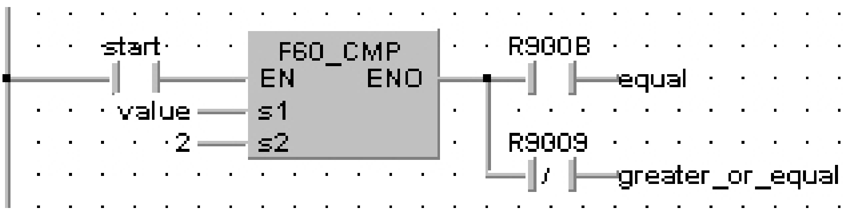
Example In this example the function F60_CMP is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	INT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on the status of R900B (= flag)
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST
```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
    F60_CMP(value, 2);
    IF R900B THEN
        equal := TRUE;
    END_IF;
    IF NOT(R9009) THEN
        greater_or_equal:= TRUE;
    END_IF;
END_IF;
```

F61_DCMP

32-bit data compare

Steps: 9

Description Compares the 32-bit data or 32-bit equivalent constant specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON-state. The compare operation result is stored in special internal relays R9009, R900A to R900C.

Data	Comparison between s1 and s2	Flags			
		R900A (>flag)	R900B (=flag)	R900C (<flag)	R9009 (carry-flag)
16-bit data with sign	s1<s2	OFF	OFF	ON	#
	s1=s2	OFF	ON	OFF	OFF
	s1>s2	ON	OFF	OFF	#
16-bit data without sign	s1<s2	#	OFF	#	ON
	s1=s2	OFF	ON	OFF	OFF
	s1>s2	#	OFF	#	OFF

turns ON or OFF depending on the conditions

PLC types: **Availability of F61_DCMP (see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit area or 32-bit equivalent constant to be compared
s2	DINT, DWORD	32-bit area or 32-bit equivalent constant to be compared

The variables **s1** and **s2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Example In this example the function F61_DCMP is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

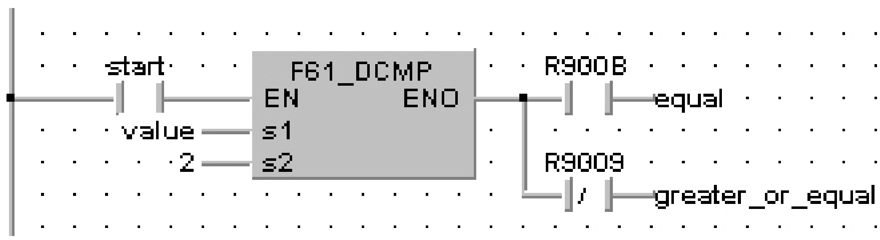
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on the status of R900B (= flag)
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
    F61_DCMP(value, 2);
    IF R900B THEN
        equal:= TRUE;
    END_IF;
    IF NOT(R9009) THEN
        greater_or_equal:= TRUE;
    END_IF;
END_IF;
```

F62_WIN 16-bit data band compare

Steps: 7

Description Compares the 16-bit equivalent constant or 16-bit data specified by **s1** with the data band specified by **s2** and **s3** if the trigger **EN** is in the ON-state. This instruction checks that **s1** is in the data band between **s2** (lower limit) and **s3** (higher limit), larger than **s3**, or smaller than **s2**. The compare operation considers +/- sign. Since the BCD data is also treated as 16-bit data with sign, we recommend using BCD data within the range of 0 to 7999 to avoid confusion. The compare operation result is stored in special relays R900A, R900B, and R900C.

Comparison between s1, s2 and s3	**Flags**		
	R900A (>flag)	**R900B (=flag)**	**R900C (<flag)**
s1 < s2	OFF	OFF	ON
s2 ≤ s1 ≤ s3	OFF	ON	OFF
s1 > s3	ON	OFF	OFF

PLC types: **Availability of F62_WIN** **(see page 926)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit area or 16-bit equivalent constant to be compared
s2	INT, WORD	lower limit, 16-bit area or 16-bit equivalent constant
s3	INT, WORD	upper limit, 16-bit area or 16-bit equivalent constant

The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

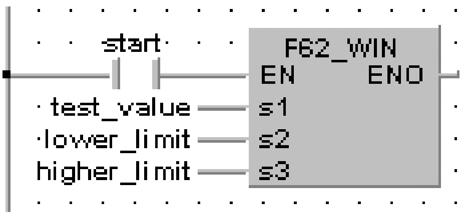
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	INT	35	this value will be compared with the data band specified by lower_limit and higher_limit; result after 0->1 leading edge from start: R900A and R900C are OFF, R900B is ON
2	VAR	lower_limit	INT	0	lower limit
3	VAR	higher_limit	INT	100	higher limit

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F62_WIN( s1_In:= test_value,
         s2_Min:= lower_limit,
         s3_Max:= higher_limit);
END_IF;
```

F63_DWIN 32-bit data band compare

Steps: 13

Description Compares the 32-bit equivalent constant or 32-bit data specified by **s1** with the data band specified by **s2** and **s3** if the trigger **EN** is in the ON-state. This instruction checks that **s1** is in the data band between **s2** (lower limit) and **s3** (higher limit), larger than **s3**, or smaller than **s2**. The compare operation considers +/- sign. Since the BCD data is also treated as 32-bit data with sign, we recommend using BCD data within the range of 0 to 79999999 to avoid confusion. The compare operation result is stored in special relays R900A, R900B, and R900C.

Comparison between s1, s2 and s3	Flags		
	R900A (>flag)	R900B (=flag)	R900C (<flag)
s1 < s2	OFF	OFF	ON
s2 ≤ s1 ≤ s3	OFF	ON	OFF
s1 > s3	ON	OFF	OFF

PLC types: **Availability of F63_DWIN** **(see page 926)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit area or 32-bit equivalent constant to be compared
s2	DINT, DWORD	lower limit, 32-bit area or 32-bit equivalent constant
s3	DINT, DWORD	upper limit, 32-bit area or 32-bit equivalent constant

The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Example In this example the function F63_DWIN is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

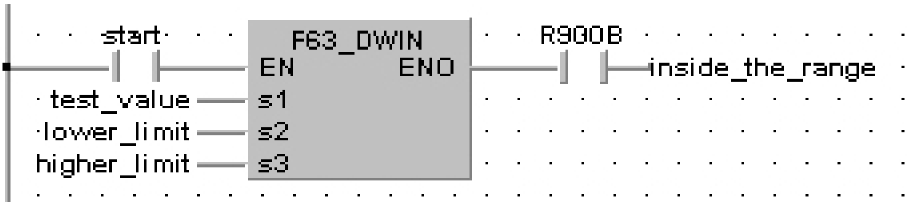
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	DINT	35	this value will be compared with the data band specified by lower_limit and higher_limit; result after 0->1 leading edge from start: R900A and R900C are OFF, R900B is ON
2	VAR	lower_limit	DINT	0	lower limit
3	VAR	higher_limit	DINT	100	higher limit
4	VAR	inside_the_range	BOOL	FALSE	

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
inside_the_range:= FALSE;
IF start THEN
    F63_DWIN( s1_In:= test_value,
         s2_Min:= lower_limit,
         s3_Max:= higher_limit);
    IF R900B THEN
        inside_the_range:= TRUE;
    END_IF;
END_IF;
```

F64_BCMP

Block data compare

Steps: 7

Description Compares the contents of data block specified by **s2** with the contents of data block specified by **s3** according to the contents specified by **s1** if the trigger **EN** is in the ON-state.

s1 specifications

16#	1	0	0 4
	⇑	⇑	⇑
	A	B	C

A = Starting byte position of data block specified by **s3**

1: Starting from higher byte

0: Starting from lower byte

B = Starting byte position of data block specified by **s2**

1: Starting from higher byte

0: Starting from lower byte

C = Number of bytes to be compared

range: 16#01 to 16#99 (BCD)

The compare operation result is stored in the special internal relay R900B. When **s2 = s3**, the special internal relay is in the ON-state.

PLC types: **Availability of F64_BCMP** **(see page 926)**

☞ **The flag R900B used for the compare instruction is renewed each time a compare instruction is executed. Therefore the program that uses R900B should be just after F64_BCMP.**

Data types

Variable	Data type	Function
s1	WORD	control code specifying byte positions and number of bytes to be compared
s2	INT, WORD	starting 16-bit area to be compared to s3
s3	INT, WORD	starting 16-bit area to be compared to s2

The variables **s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

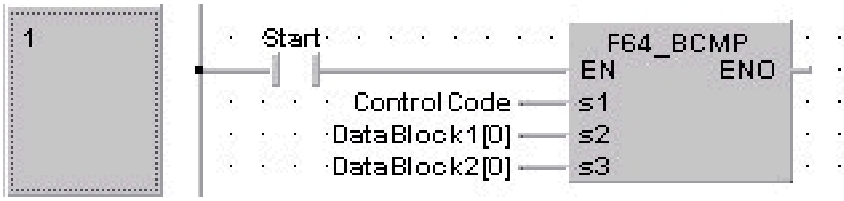
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	ControlCode	WORD	16#1106	s2 starting from upper byte s3 starting from upper byte compare 6 bytes
2	VAR	DataBlock1	ARRAY [0..5] OF INT	[6(1234)]	
3	VAR	DataBlock2	ARRAY [0..5] OF INT	[6(1234)]	

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F64_BCMP( s1_Control:= ControlCode,
         s2_Start:= DataBlock1[0],
         s3_Start:= DataBlock2[0]);
END_IF;
```

F346_FWIN **Floating point data band compare** **Steps: 14**

Description The function compares a data band whose upper and lower limits are specified at inputs **s2** and **s3** with a value that is entered at input **s1**. The result is returned as follows:

- If the value at **s1** is smaller than the value at **s2** (lower limit of the data band), the < special internal relay (R900C) is set to TRUE.
- If the value at **s1** is larger than the value at **s3** (upper limit of the data band), the > special internal relay (R900A) is set to TRUE. The < flag (R900C) and the = flag (R900B) are set to FALSE.
- If the value at **s1** is within the data band values set at **s2** and **s3**, the = special internal relay (R900B) is set for an instant. The < flag (R900C) and the > flag (R900A) are set to FALSE.

PLC types: **Availability of F346_FWIN (see page 932)**

Data types

Variable	Data type	Function
s1	REAL	REAL number data to be compared to s2 and s3
s2	REAL	lower limit
s3	REAL	upper limit

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the values at inputs s1, s2, and s3 are not REAL numbers or if the value at s2 > s3.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

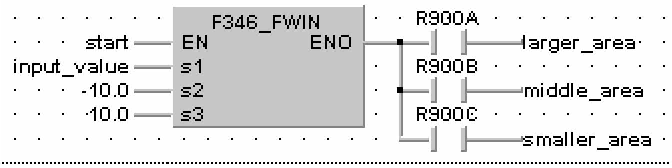
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	3.111	
2	VAR	larger_area	BOOL	FALSE	result: here FALSE
3	VAR	middle_area	BOOL	FALSE	result: here TRUE
4	VAR	smaller_area	BOOL	FALSE	result: here FALSE

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

The constants -10.0 and 10.0 are assigned to the inputs s2 (upper limit) and s3 (lower limit). You may, however, declare two variables in the POU header instead. When the variable **start** is set to TRUE, the function is carried out. The values of special internal relays R900A (> flag), R900B (= flag) and R900C (< flag) are transferred to the variables **larger_area, middle_area** and **smaller_area**. Since the **output_value** = 3.111 is within the range of the limits set (-10.0 to 10.0), the = relay and hence the variable **middle_area** are set to TRUE.

LD



ST

```
input_value:=3.111;
IF start THEN
    F346_FWIN( s1_In:= input_value , s2_Min:= -10.0 ,
s3_Max:= 10.0 );
END_IF;(* -10.0 =lower limit, 10.0 upper limit *)

IF R900A THEN
    larger_area:=TRUE;
END_IF;
IF R900B THEN
    middle_area:=TRUE;
END_IF;
IF R900C THEN
    smaller_area:=TRUE;
END_IF;
```

F373_DTR

16-bit data revision detection

Steps: 6

Description The function detects changes in a value at input s by comparing it with its former value that is stored at output d. If the new input value at s does not coincide with the old value, the function assigns the new value to output d. To signal the change, the carry flag R9009 is set simultaneously.

PLC types: **Availability of F373_DTR (see page 932)**

☞ **The status of the carry flag is updated at each execution of the instruction. Therefore, programs that use the carry flag should utilize it immediately after F373_DTR is executed.**

Data types

Variable	Data type	Function
s	INT, WORD	16-bit area for detecting data changes
d	INT, WORD	area where data of previous execution is stored.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	to TRUE	- the input value at s has changed in comparison to the former value.

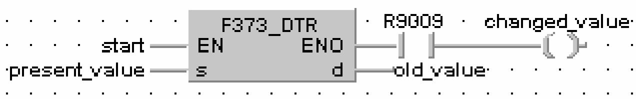
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	present_value	INT	0	value whose status should be monitored
2	VAR	old_value	INT	0	dummy value for storing the former present value
3	VAR	changed_value	BOOL	FALSE	signal for changing present value

Body When the variable **start** is set to TRUE, the function is carried out. If the input value **present_value** has changed in comparison to the output value **old_value** the carry flag R9009 is set. The status of the carry flag is then assigned to the variable **changed_value**.

LD



ST

```
IF start THEN
    F373_DTR(present_value, old_value);
    IF R9009 THEN
        changed_value:=TRUE;
    END_IF;
END_IF;
```

F374_DDTR 32-bit data revision detection

Steps: 6

Description The function detects changes in a value at input s by comparing it with its former value that is stored at output d. If the new input value at s does not coincide with the old value, the function assigns the new value to output d. To signal the change, the carry flag R9009 is set simultaneously.

PLC types: **Availability of F374_DDTR** **(see page 932)**

☞ **The status of the carry flag is updated at each execution of the instruction. Therefore, programs that use the carry flag should utilize it immediately after F374_DDTR is executed.**

Data types

Variable	Data type	Function
s	DINT, DWORD	32-bit area for detecting data changes
d	DINT, DWORD	32-bit area where data of previous execution is stored

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

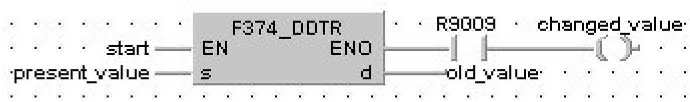
No.	IEC address	Set	If
R9009	%MX0.900.9	to TRUE	- the input value at s has changed in comparison to the former value.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	present_value	DINT	0	value whose status should be monitored
2	VAR	old_value	DINT	0	dummy value for storing the former present value
3	VAR	changed_value	BOOL	FALSE	signal for changing present value

Body When the variable **start** is set to TRUE, the function is carried out. If the input value **present_value** has changed in comparison to the output value **old_value** the carry flag R9009 is set. The status of the carry flag is then assigned to the variable **changed_value**.

LD 

ST

```
IF start THEN
    F374_DDTR(present_value, old_value);
    IF R9009 THEN
        changed_value:=TRUE;
    END_IF;
END_IF;
```

19.1 Further Comparison Instructions

If you need information on one of the following comparison instructions, please refer to the corresponding standard operators in the online help:

ST=	AN=	OR=	STD=	AND=	ORD=
ST<>	AN<>	OR<>	STD<>	AND<>	ORD<>
ST>	AN>	OR>	STD>	AND>	ORD>
ST>=	AN>=	OR>=	STD>=	AND>=	ORD>=
ST<	AN<	OR<	STD<	AND<	ORD<
ST<=	AN<=	OR<=	STD<=	AND<=	ORD<=

Chapter 20

Conversion Instructions

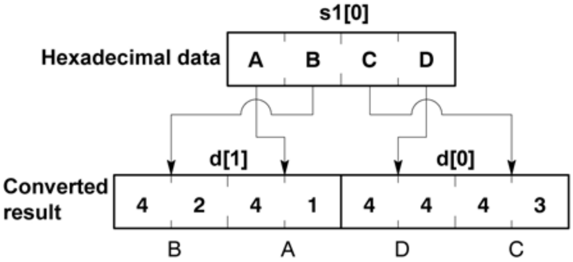
F71_HEX2A

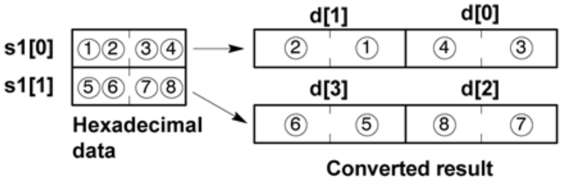
HEX -> ASCII conversion

Steps: 7

Description Converts the data of **s2** bytes starting from the 16-bit area specified by **s1** to ASCII codes that express the equivalent hexadecimals if the trigger **EN** is in the ON-state. The number of bytes to be converted is specified by **s2**. The converted result is stored in the area starting with the 16-bit area specified by **d**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.





ASCII HEX codes to express hexadecimal characters:

Hexadecimal number	ASCII HEX code
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39
A	16#41
B	16#42
C	16#43
D	16#44
E	16#45
F	16#46

PLC types: **Availability of F71_HEX2A (see page 927)**

Data types

Variable	Data type	Function
s1	INT, WORD	starting 16-bit area for hexadecimal number (source)
s2	INT	specifies number of source data bytes to be converted
d	WORD	starting 16-bit area for storing ASCII code (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the byte number specified by s2 exceeds the area specified by s1 - the calculated result exceeds the area specified by d. - the data specified by s2 is recognized as "0".
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

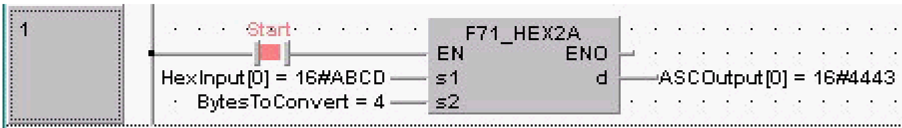
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	HexInput	ARRAY [0..1] OF WORD	[16#abcd,16#ef]	
2	VAR	BytesToConvert	INT	4	3 bytes will be converted
3	VAR	ASCOutput	ARRAY [0..3] OF WORD	[4(0)]	3 bytes hex. require 6 bytes for ASCII code ARRAY[3] will be filled with two zero characters = 16#3030

Body

When the variable **Start** is set to true, the number of data bytes given in **BytesToConvert** in **HexInput** is converted to ASCII code and stored in **ASCOutput**. Note that two characters that make up one byte are interchanged when stored. One Monitor Header shows the Hex values, and the other the ASCII values.

LD



Monitor Header f71 [PRG]...

- f71	Structure
Start	2#1 at %MX0.4.10
- HexInput	Structure
[0]	16#ABCD at %MW5.133
[1]	16#00EF at %MW5.134
BytesToConvert	4 at %MW5.135
- ASCOutput	Structure
[0]	16#4443 at %MW5.136
[1]	16#4241 at %MW5.137
[2]	16#4645 at %MW5.138
[3]	16#3030 at %MW5.139

Monitor Header f71 [PRG]...

- f71	Structure
Start	2#1 at %MX0.4.10
- HexInput	Structure
[0]	16#ABCD at %MW5.133
[1]	46#00EF at %MW5.134
BytesToConvert	4 at %MW5.135
- ASCOutput	Structure
[0]	CD at %MW5.136
[1]	AB at %MW5.137
[2]	EF at %MW5.138
[3]	00 at %MW5.139

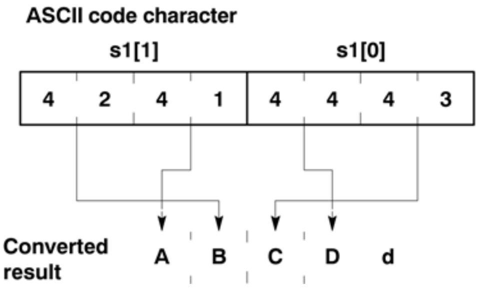
ST
```
IF start THEN
    F71_HEX2A( s1_Start:= HexInput[0],
         s2_Number:= BytesToConvert,
         d_Start=> ASCOutput[0]);
END_IF;
```

F72_A2HEX ASCII -> HEX conversion

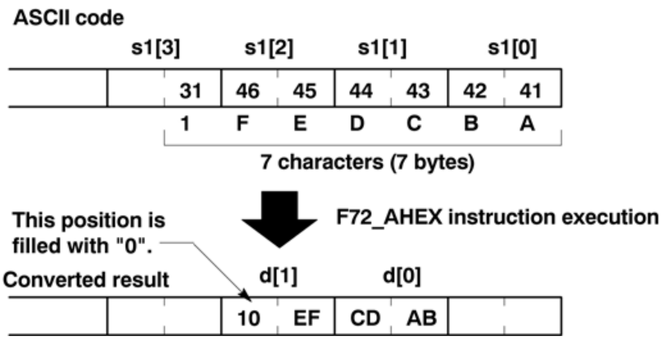
Steps: 7

Description Converts the ASCII codes that express the hexadecimal characters starting from the 16-bit area specified by **s1** to hexadecimal numbers if the trigger **EN** is in the ON-state. **s2** specifies the number of ASCII (number of characters) to be converted. The converted result is stored in the area starting from the 16-bit area specified by **d**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to a hexadecimal number, the data length will thus be half the length of the ASCII code source data.

The data for two ASCII code characters is converted to two numeric digits for one word. When this takes place, the characters of the upper and lower bytes are interchanged. Four characters are converted as one segment of data.



Converted results are stored in byte units. If an odd number of characters is being converted, "0" will be entered for bits 0 to 3 of the final data (byte) of the converted results. Conversion of odd number of source data bytes:



Hexadecimal characters and ASCII codes:

ASCII HEX code	Hexadecimal number
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9
16#41	A
16#42	B
16#43	C
16#44	D
16#45	E
16#46	F

PLC types: **Availability of F72_A2HEX (see page 927)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	INT, WORD	starting 16-bit area for storing converted data (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of bytes specified by s2 exceeds the area specified by s1. - the converted result exceeds the area specified by d. - the data specified by s2 is recognized as "0". - ASCII code, not a hexadecimal number (0 to F), is specified.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

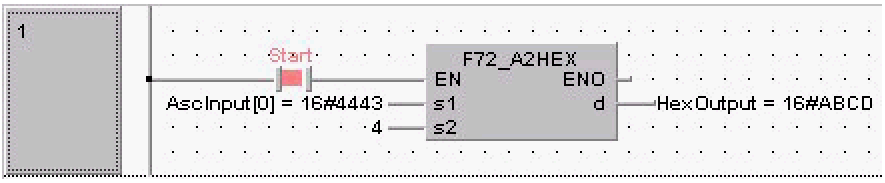
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	AscInput	ARRAY [0..1] OF WORD	[16#4443,16#4241]	16#4443 = CD (ASCII) 16#4241 = AB (ASCII)
2	VAR	HexOutput	WORD	0	Result = ABCD Upper- and lower-byte data interchanged

Body

When the variable **Start** is set to TRUE, the function is executed. In this example, the value for s2, i.e. the number of bytes to be converted from ASCII code to hexadecimal code, is entered directly at the contact pin.

LD



ST

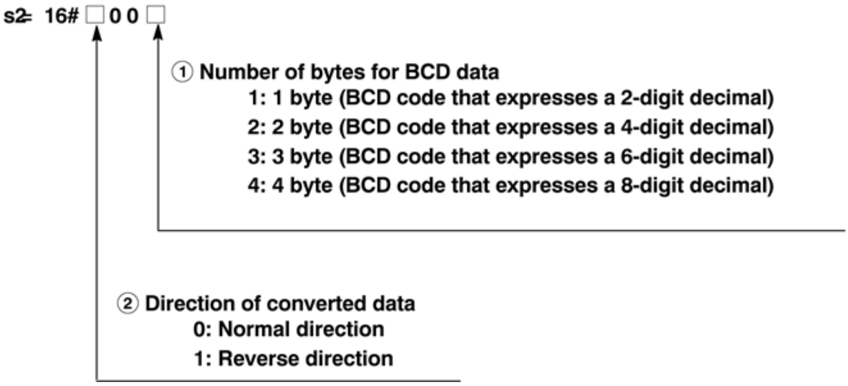
```
IF start THEN
    F72_A2HEX( s1_Start:= AscInput[0],
         s2_Number:= 4,
         d_Start=> HexOutput);
```

F73_BCD2A

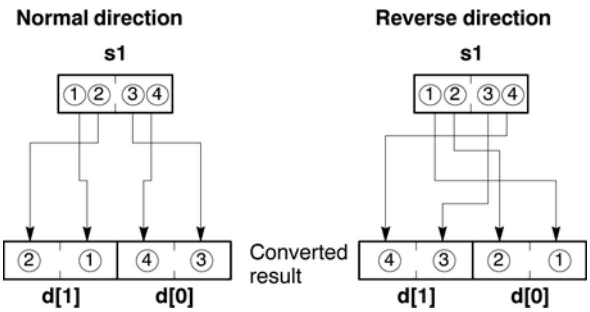
BCD -> ASCII conversion

Steps: 7

Description Converts the BCD code starting from the 16-bit area specified by **s1** to the ASCII code that expresses the equivalent decimals according to the contents specified by **s2** if the trigger **EN** is in the ON-state. **s2** specifies the number of source data bytes and the direction of converted data (normal/reverse).



The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data:



The converted result is stored in the area specified by **d**. ASCII code requires 8 bits (one byte) to express one BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the BCD source data.

ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

PLC types: **Availability of F73_BCD2A** **(see page 927)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for BCD data (source)
s2	INT, WORD	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16-bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the data specified by s1 is not BCD data. - the number of bytes specified by s2 exceeds the area specified by s1. - the converted result exceeds the area specified by d. - the data specified by s2 is recognized as "0". - the number of bytes specified by s2 is more than 16#4.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

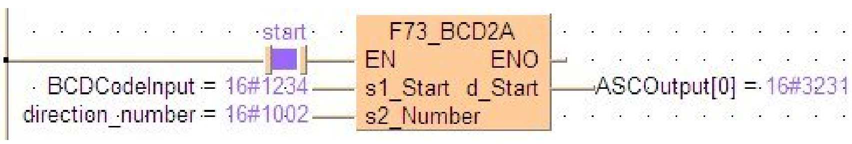
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	BCDCodeInput	WORD	16#1234	
2	VAR	direction_number	WORD	16#1002	specifies the operation:
3	VAR	ASCOutput	ARRAY [0..1] OF WORD	[2(0)]	result after a 0->1 leading

Body

When the variable **Enable** is set to TRUE, the function is executed. In this example, the variable **direction_number** specifies that from the input variable **BCDCodeInput**, 2 bytes will be converted in the reverse direction and stored in **ASCOutput**.

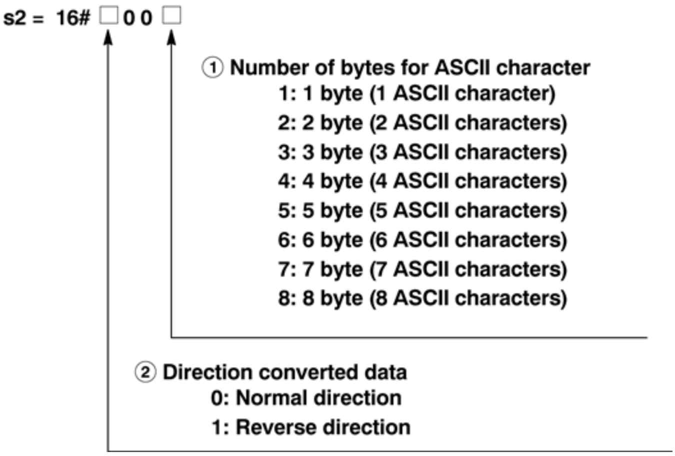
LD



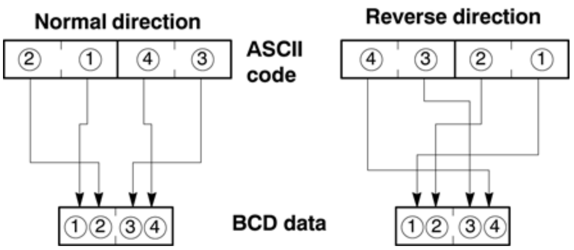
ST
```
IF start THEN
    F73_BCD2A( s1_Start:= BCDCodeInput ,
         s2_Number:= direction_number ,
         d_Start=> ASCOutput[0] );
END_IF;
```

F74_A2BCD ASCII -> BCD conversion Steps: 9

Description Converts the ASCII codes that express the decimal characters starting from the 16-bit area specified by **s1** to BCD if the trigger **EN** is in the ON-state. **s2** specifies the number of source data bytes and the direction of converted code source data.

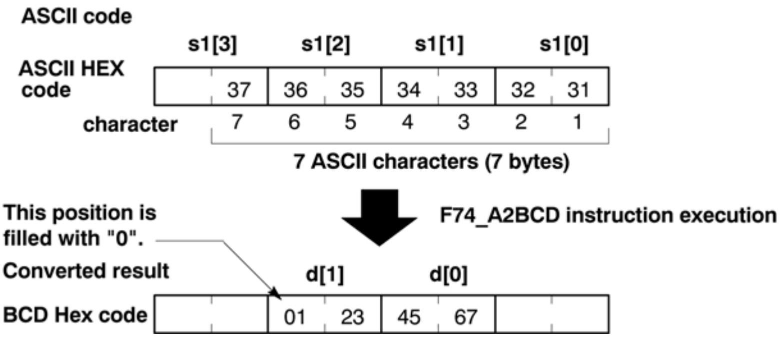


Four characters are converted as one segment of data:



The converted result is stored in byte units in the area starting from the 16-bit area specified by **d**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to a BCD number, the data length will thus be half the length of the ASCII code source data.

If an odd number of characters is being converted, "0" will be entered for bit position 0 to 3 of the final data (byte) of the converted results if data is sequenced in the normal direction, and "0" will be entered for bit position 4 to 7 if data is being sequenced in the reverse direction:



ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

PLC types: **Availability of F74_A2BCD (see page 927)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for storing ASCII code (source)s
s2	INT, WORD	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16-bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- ASCII code not corresponding to decimal numbers (0 to 9) is specified. - the number of bytes specified by **s2** exceeds the area specified by **s1**. - the converted result exceeds the area specified by **d**. - the data specified by **s2** is recognized as "0". - the number of bytes for ASCII characters in **s2** is more than 16#8.
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

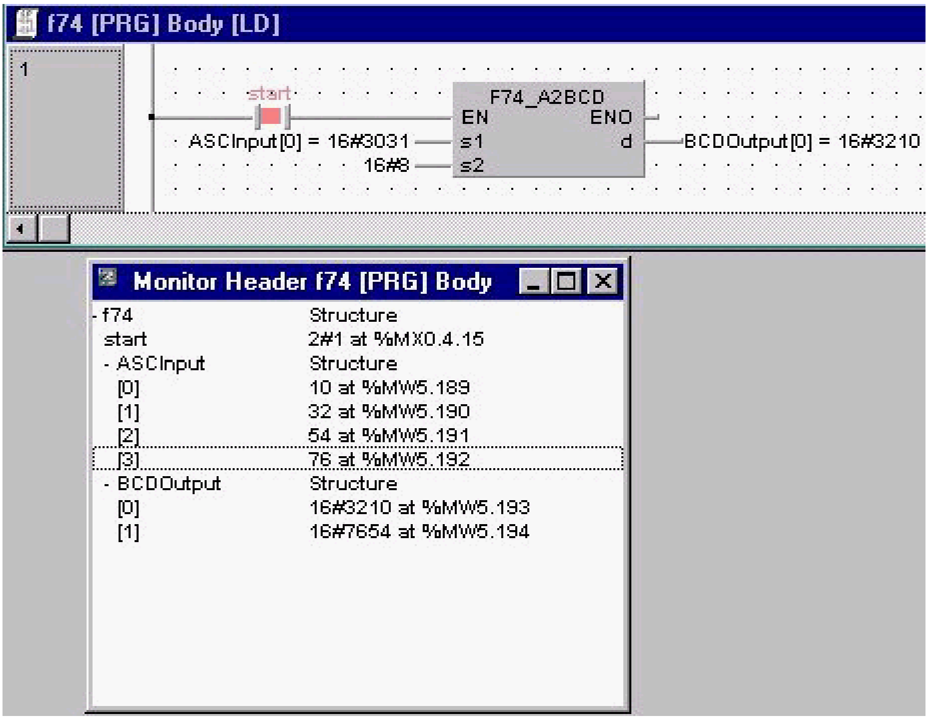
f74 [PRG] Header

	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	ASCInput	ARRAY [0..3] OF WORD	[16#3031,16#3233,16#3435,16#3637]
2	VAR	BCDOutput	ARRAY [0..1] OF WORD	[2(0)]

Body

When the variable **start** is set to TRUE, the function is executed. For the variable at s1, you never need define an ARRAY with more than four elements because 8 ASCII characters require 8 bytes of memory and the function cannot convert more than 8 bytes. In this example, the value for s2 is entered directly at the contact pin.

LD



ST

```
IF start THEN
    F74_A2BCD( s1_Start:= ASCInput[0] ,
        s2_Number:= 16#8 ,
        d_Start=> BCDOutput[0] );
END_IF;
```

F75_BIN2A

16-bit BIN -> ASCII conversion

Steps: 7

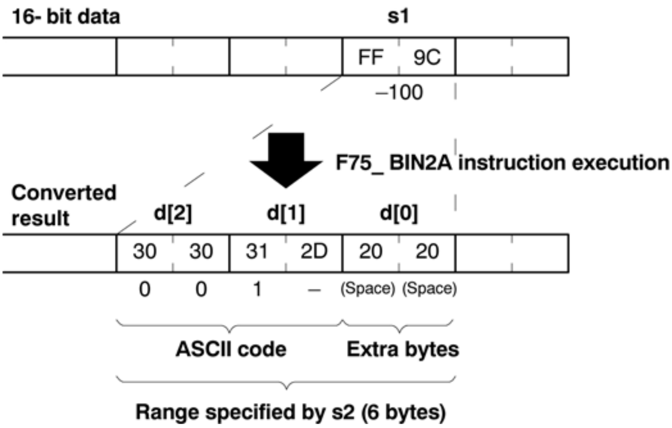
Description Converts the 16-bit data specified by **s1** to ASCII codes that express the equivalent decimal value. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. Specify the number of bytes in decimal number in **s2**. (This specification cannot be made with BCD data.)

- If a positive number is converted, the "+" sign is not converted.
- When a negative number is converted, the "-" sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data, the ASCII code for "SPACE" (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction towards the final address, so the position of the ASCII code may change, depending on the size of the data storage area.

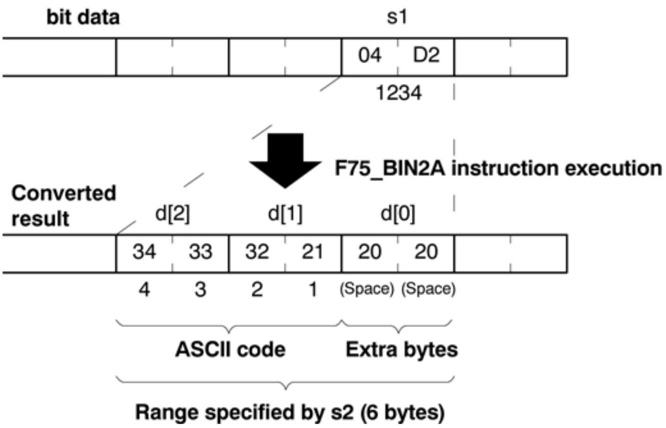
- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

The following illustrations show conversions from 16-bit decimal data to ASCII codes.

When a negative number is converted:



When a positive number is converted



Decimal characters to express ASCII HEX code:

Decimal characters	ASCII HEX code
SPACE	16#20
-	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

PLC types: **Availability of F75_BIN2A** **(see page 927)**

Data types

Variable	Data type	Function
s1	INT, WORD	16-bit area to be converted (source)
s2	INT	specifies number of bytes used to express destination data (ASCII codes)
d	WORD	16-bit area for storing ASCII codes (destination)

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of bytes specified by **s2** exceeds the area specified by **d**. - the data specified by **s2** is recognized as "0". - the converted result exceeds the area specified by **d**. - the number of bytes of converted result exceeds the number of bytes specified by **s2**.
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

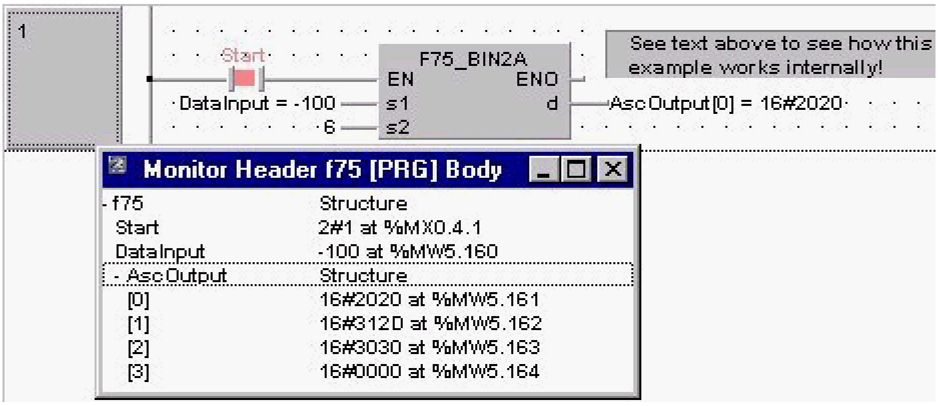
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DataInput	INT	-100	
2	VAR	AscOutput	ARRAY [0..3] OF WORD	[4(0)]	

Body

When the variable **Start** is set to TRUE, the function is executed. This programming example is based on the example for the conversion of a negative number outlined above. The monitor value icon is activated for both the LD and IL bodies; the monitor header icon is activated for the LD body.

LD



ST

```
IF start THEN
    F75_BIN2A( s1:= DataInput ,
         s2_Number:= 6 ,
         d_Start=> ASCOutput[0] );
END_IF;
```

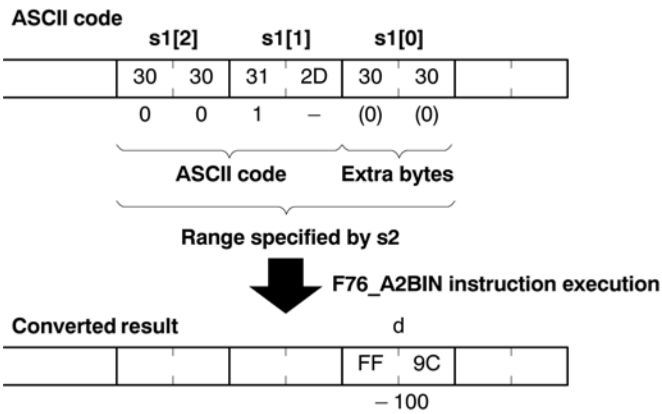
F76_A2BIN ASCII -> 16-bit BIN conversion

Steps: 7

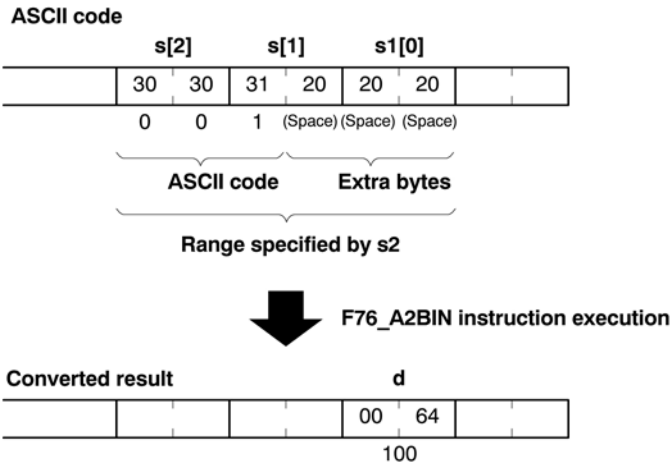
Description Converts the ASCII codes that express the decimal digits, starting from the 16-bit area specified by **s1** to 16-bit data as specified by **s2**. The converted result is stored in the area specified by **d**. **s2** specifies the number of source data bytes to be converted using decimal number. (This specification cannot be made with BCD data.)

- The ASCII codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required for the data you want to convert, place "0" (ASCII HEX code: 16#30) or "SPACE" (ASCII HEX code: 16#20) into the extra bytes.
- ASCII codes with signs (such as +: 16#2B and -: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII code indicating a negative number



Example of converting an ASCII code indicating a positive number



ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	-
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

PLC types: **Availability of F76_A2BIN** **(see page 927)**

Data types

Variable	Data type	Function
s1	WORD	16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	INT, WORD	16-bit area for storing converted data (destination)

Part III F/P Instructions

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of bytes specified by s2 exceeds the area specified by s1. - the data specified by s2 is recognized as "0". - the converted result exceeds the 16-bit area specified by d. - ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, -, and SPACE) is specified.
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

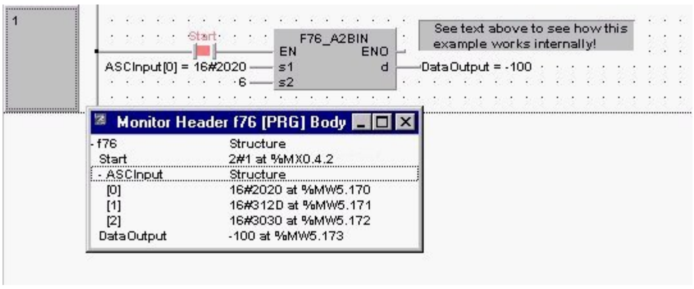
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	ASCInput	ARRAY [0..2] OF WORD	[16#2020,16#3120,16#3030]
2	VAR	DataOutput	INT	0

Body

When the variable **Start** is set the TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number on the main page of F76_A2BIN.

LD



ST

```
IF start THEN
    F76_A2BIN( s1_Start:= ASCInput[0] ,
        s2_Number:= 6 ,
        d=> DataOutput );
END_IF;
```

F77_DBIN2A

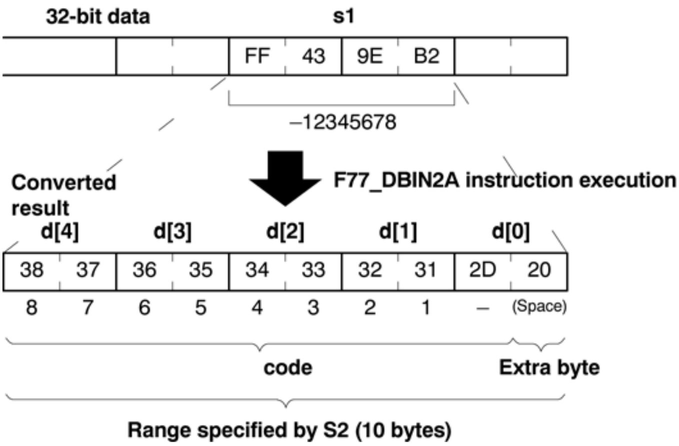
32-bit BIN -> ASCII conversion

Steps: 11

Description Converts the 32-bit data specified by **s1** to ASCII code that expresses the equivalent decimals. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. **s2** specifies the number of bytes used to express the destination data using decimal.

- When a positive number is converted, the "+" sign is not converted.
- When a negative number is converted, the "-" sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data, the ASCII code for "SPACE" (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction of the last address, so the position of the ASCII code may change depending on the size of the data storage area.
- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

Example of converting a negative number from 32–bit decimal format to ASCII codes



Decimal characters to express ASCII HEX code:

Part III F/P Instructions

Decimal characters	ASCII HEX code
SPACE	16#20
+	16#2B
-	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

PLC types: **Availability of F77_DBIN2A (see page 927)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	32-bit data area to be converted (source)
s2	INT	specifies number of bytes to express destination data (ASCII codes)
d	WORD	16-bit area for storing ASCII codes (destination)

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of bytes specified by **s2** exceeds the area specified by **d**. - the data specified by **s2** is recognized as "0". - the converted result exceeds the area specified by **d**. - the number of bytes of converted result exceeds the number of bytes specified by **s2**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

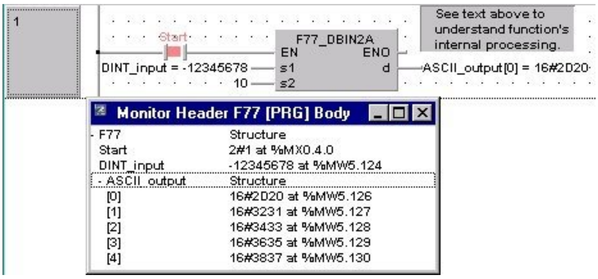
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DINT_input	DINT	-12345678	
2	VAR	ASCII_output	ARRAY [0..4] OF WORD	[5(0)]	

Body When the variable **Start** is set to TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2.

LD



ST

```
IF start THEN
    F77_DBIN2A( s1:= DINT_input ,
         s2_Number:= 10 ,
         d_Start=> ASCII_output[0] );
END_IF;
```

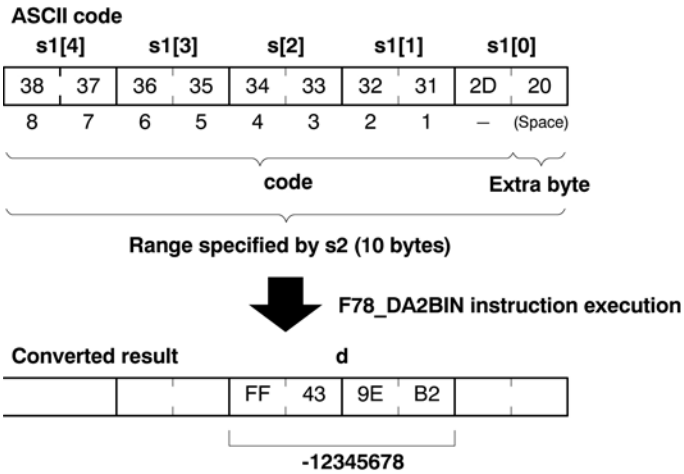
F78_DA2BIN ASCII -> 32 bit BIN conversion

Steps: 11

Description Converts ASCII code that expresses the decimal digits, starting from the 16-bit area specified by **s1** to 32-bit data as specified by **s2**. The converted result is stored in the area starting from the 16-bit area specified by **d**. **s2** specifies the number of bytes used to express the destination data using decimals.

- The ASCII codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required by the data you want to convert, place "0" (ASCII HEX code: 16#30) or "SPACE" (ASCII HEX code: 16#20) in the extra bytes.
- ASCII codes with signs (such as +: 16#2B and -: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII code indicating a negative number



ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	-
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

PLC types: **Availability of F78_DA2BIN** **(see page 927)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	DINT, DWORD	area for 32-bit data storage (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the number of bytes specified by s2 exceeds the area specified by s1. - the data specified by s2 is recognized as "0". - the converted result exceeds the area specified by d. - the converted result exceeds the 32-bit area. - ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, -, and SPACE) is specified.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

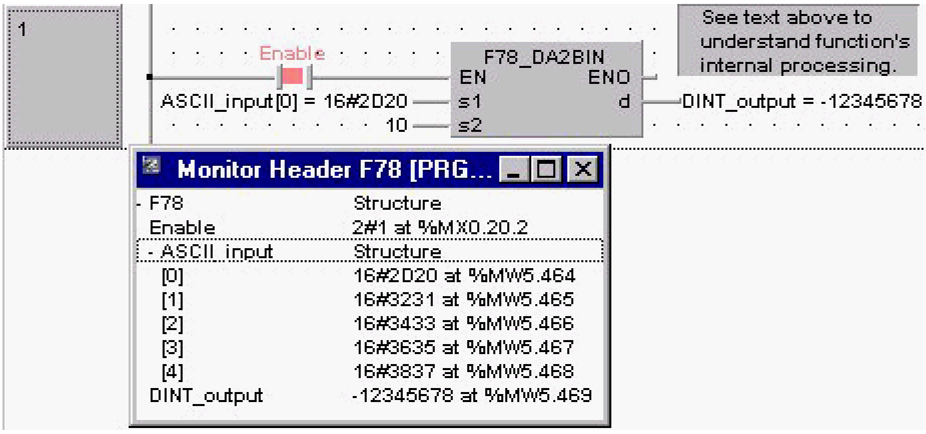
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	ASCII_input	ARRAY [0..4] OF WORD	[16#2D20,16#3:	For values, see Monitor Header
2	VAR	DINT_output	DINT	0	

Body When the variable **Enable** is set to TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number outlined above.

LD



ST

```
IF start THEN
    F78_DA2BIN( s1_Start:= ASCII_input[0] ,
        s2_Number:= 10 ,
        d=> DINT_output );
END_IF;
```

F80_BCD

16-bit BIN -> 4-digit BCD conversion

Steps: 5

Description Converts the 16-bit binary data specified by **s** to the BCD code that expresses 4-digit decimals if the trigger **EN** is in the ON-state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 9999 (270F hex).

Source [s]: 16

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
Decimal	16			

Conversion (to BCD code)

Destination [d]: 16#16 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
BCD Hex code	0	0	1	6

PLC types: **Availability of F80_BCD** **(see page 927)**

Data types

Variable	Data type	Function
s	INT, WORD	binary data (source), range: 0 to 9999
d	WORD	16-bit area for 4-digit BCD code (destination)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- 16-bit binary data outside the range of 0 (16#0) to 9999 (16#270F) is converted
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	Decimal Input	INT	16	
2	VAR	BCD_output	WORD	0	

Body When the variable **Enable** is set to TRUE, the function is executed. The decimal value in **DecimalInput** is converted to a BCD hexadecimal value and stored in the variable **BCD_output**.

LD

ST

```
IF Enable THEN
    F80_BCD(DecimalInput, BCD_output);
END_IF;
```

F81_BIN

4-digit BCD -> 16-bit BIN conversion

Steps: 5

Description Converts the BCD code that expresses 4-digit decimals specified by **s** to 16-bit binary data if the trigger **EN** is in the ON-state. The converted result is stored in the area specified by **d**.

Source [s]: 16#15 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
BCD code	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
BCD Hex code	0	0	1	5

Conversion (to binary data)

Destination [d]: 15

position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
Decimal	15			

PLC types: **Availability of F81_BIN (see page 927)**

Data types

Variable	Data type	Function
s	WORD	16-bit area for 4-digit BCD data (source)
d	INT, WORD	16-bit area for storing 16-bit binary data (destination)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the data specified by s is not BCD data.
R9008	%MX0.900.8	for an instant	

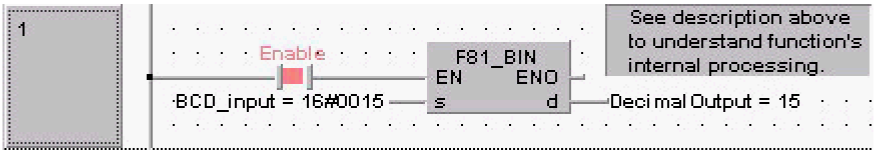
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	BCD_input	WORD	16#0015	
2	VAR	Decimal Output	INT	0	

Body When the variable **Enable** is set to TRUE, the function is executed. The BCD value assigned to the variable **BCD_input** is converted to a decimal value and stored in the variable **DecimalOutput**. The monitor value icon is activated for both the LD and IL bodies.

LD



ST

```
IF Enable THEN
    F81_BIN(BCD_Input, DecimalOutput);
END_IF;
```

F82_DBCD

32-bit BIN -> 8-digit BCD conversion

Steps: 7

Description Converts the 32-bit binary data specified by **s** to the BCD code that expresses 8-digit decimals if the trigger **EN** is in the ON-state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 99,999,999 (5F5E0FF hex).

Source (s): 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							
	32-bit area							

Destination (d): 16#72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

PLC types: **Availability of F82_DBCD (see page 927)**

Data types

Variable	Data type	Function
s	DINT, DWORD	binary data (source), range: 0 to 99,999,999
d	DWORD	32-bit area for 8-digit BCD code (destination)

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- 32-bit data specified by s outside the range of 0 (16#0) to 99999999 (16#5F5E0FF) is converted.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

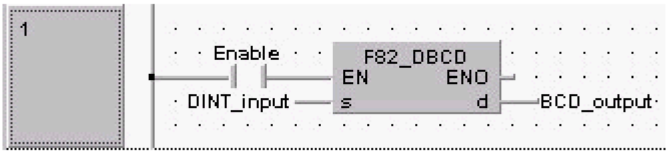
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	DINT_input	DINT	72811730	
2	VAR	BCD_output	DWORD	0	

Body

When the variable **Enable** is set to TRUE, the function is executed. The decimal value in **DINT_input** is converted to a BCD hexadecimal value and stored in the variable **BCD_output**. You may also assign a decimal, binary (prefix 2#), or hexadecimal (prefix 16#) value directly at the contact pin for s.

LD



ST

```
IF Enable THEN
    F82_DBCD(DINT_input, BCD_output);
END_IF;
```

F83_DBIN

8-digit BCD -> 32-bit BIN conversion

Steps: 7

Description Converts the BCD code that expresses 8-digit decimals specified by **s** to 32-bit binary data if the trigger **EN** is in the ON-state. The converted result is stored in the area specified by **d**.

Source (s): 16#72811730 (BCD)

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

32-bit area

Destination (d): 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							

PLC types: **Availability of F83_DBIN (see page 927)**

Data types

Variable	Data type	Function
s	DWORD	area for 8-digit BCD data (source)
d	DINT, DWORD	32-bit area for storing 32-bit data (destination)

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the data specified by **s** is not BCD data.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

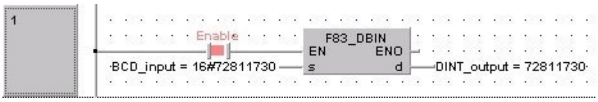
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	BCD_input	DWORD	16#72811730	
2	VAR	DINT_output	DINT	0	

Body

When the variable **Enable** is set to TRUE, the function is executed. The BCD value assigned to the variable **BCD_input** is converted to a decimal value and stored in the variable **DINT_output**.

LD



ST

```
IF Enable THEN
    F83_DBIN(BCD_input, DINT_Output);
END_IF;
```

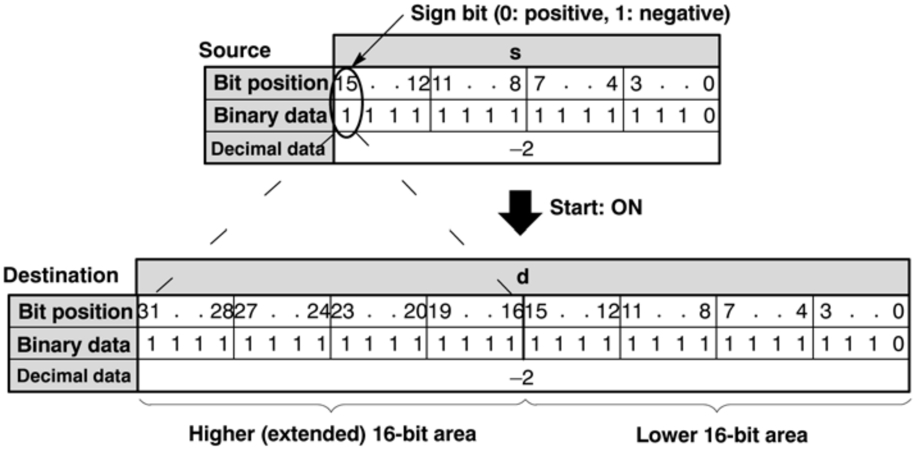
F89_EXT

16-bit data sign extension, INT -> DINT

Steps: 3

Description 16-bit data is converted to 32-bit data without signs and values being changed. F89 copies the sign bit of the 16-bit data specified in **s** to all the bits of the higher 16-bit area (extended 16-bit area) in **d**.

If the sign bit (bit position 15) of the 16-bit data specified by **s** is 0, all higher 16 bits in the variable assigned to **d** will be 0. If the sign bit of **s** is 1, the higher 16 bits of **d** will be 1.



PLC types: **Availability of F89_EXT** **(see page 927)**

Data types

Variable	Data type	Function
s	INT, WORD	16-bit source data area, bit 15 is sign bit
d	DINT, DWORD	32-bit destination area, s copied to lower 16 bits, higher 16 bits filled with sign bit of s

Operands

For	Relay				T/C		Register			Constant
s	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function F89_EXT is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

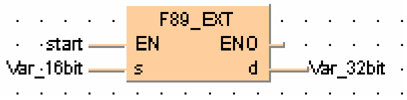
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	Var_16bit	INT	0	16bit value
2	VAR	Var_32bit	DINT	0	32bit value

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F89_EXT(Var_16bit, Var_32bit);
END_IF;
```

F90_DECO

Decode hexadecimal -> bit state

Steps: 7

Description Decodes the contents of 16-bit data specified by **s** according to the contents of **n** if the trigger EN is in the ON-state. The decoded result is stored in the area starting with the 16-bit area specified by **d**.

n specifies the starting bit position and the number of bits to be decoded using hexadecimal data:

- Bit No. 0 to 3: number of bits to be decoded
- Bit No. 8 to 11: starting bit position to be decoded

(The bits No. 4 through No. 7 and No. 12 through No. 15 are invalid.)

e.g. when **n** = 16#0404, four bits beginning at bit position four are decoded.

Relationship between number of bits and occupied data area for decoded result:

Number of bits to be decoded	Data area required for the result	Valid bits in the area for the result
1	1-word	2-bit*
2	1-word	4-bit*
3	1-word	8-bit*
4	1-word	16-bit
5	2-word	32-bit
6	4-word	64-bit
7	8-word	128-bit
8	16-word	256-bit

*Invalid bits in the data area required for the result are set to 0.

PLC types: **Availability of F90_DECO (see page 927)**

Data types

Variable	Data type	Function
s	INT, WORD	source 16-bit area or equivalent constant to be decoded
n	INT, WORD	control data to specify the starting bit position and number of bits to be decoded
d	INT, WORD	starting 16-bit area for storing decoded data (destination)

The variables **s, n** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F90_DECO is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1100011000011110	
2	VAR	specify_n	WORD	16#0003	specifies decoding
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000001000000

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F90_DECO( s:= input_value ,
        n:= specify_n ,
        d=> output_value );
END_IF;
```

F91_SEGT 16-bit data 7-segment decode

Steps: 3

Description Converts the 16-bit equivalent constant or 16-bit data specified by **s** to 4-digit data for 7-segment indication if the trigger **EN** is in the ON-state. The converted data is stored in the area starting with the 16-bit area specified by **d**. The data for 7-segment indication occupies 8 bits (1 byte) to express 1 digit.

PLC types: **Availability of F91_SEGT (see page 927)**

Data types

Variable	Data type	Function
s	INT, WORD	16-bit area or equivalent constant to be converted to 7-segment indication (source)
d	DINT, DWORD	32-bit area for storing 4-digit data for 7-segment indication (destination)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#A731	
2	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#77274F06

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

```
IF start THEN
    F91_SEGT(input_value, output_value);
END_IF;
```

F92_ENCO

Encode bit state -> hexadecimal

Steps: 7

Description Encodes the contents of data specified by **s** according to the contents of **n** if the trigger **EN** is in the ON-state. The encoded result is stored in the 16-bit area specified by **d** starting with the specified bit position. Invalid bits in the area specified for the encoded result are set to 0.

n specifies the starting bit position of destination data **d** and the number of bits to be decoded using hexadecimal data:

Bit No. 0 to 3	number of bits to be encoded
Bit No. 8 to 11	starting bit position of destination data to be encoded

(The bits No. 4 through No. 7 and No. 12 through No. 15 are invalid.)

- **Put at least one bit into the area to be checked to avoid an error message from the PLC.**
- **When several bits are set, the uppermost bit is evaluated.**

PLC types: **Availability of F92_ENCO (see page 927)**

Data types

Variable	Data type	Function
s	INT, WORD	starting 16-bit area to be encoded (source)
n	INT, WORD	control data to specify the starting bit position and number of bits to be encoded
d	INT, WORD	16-bit area for storing encoded data (destination)

The variables **s, n** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F92_ENCO is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

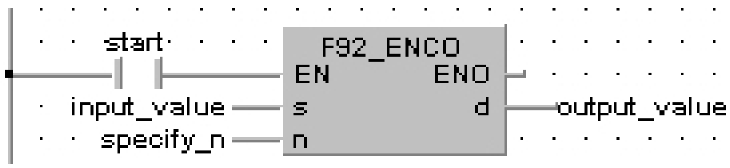
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#0000000001000000	
2	VAR	specify_n	WORD	16#0003	specifies the encodation
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000000000110

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F92_ENCO( s:= input_value ,
         n:= specify_n ,
         d=> output_value );
END_IF;
```

F95_ASC

12 Character -> ASCII transfer

Steps: 15

Description Converts the character constants specified by **s** to ASCII code. The converted ASCII code is stored in 6 words starting from the 16-bit area specified by **d**.

[s]

Character constants	A B C 1 2 3 0 D E F

[d]

Data register	d[5]		d[4]		d[3]		d[2]		d[1]		d[0]	
ASCII HEX code	2 0	4 6	4 5	4 4	2 0	3 0	3 3	3 2	3 1	4 3	4 2	4 1
ASCII character		F	E	D		0	3	2	1	C	B	A

SPACE

If the number of character constants specified by s is less than 12, the ASCII code 16#20 (SPACE) is stored in the extra destination area, e.g. s = ’12345’, d[0] = 3231, d[1] = 3433, d[2] = 2034, d[3] - d[5] = 2020.

PLC types: **Availability of F95_ASC (see page 927)**

Data types

Variable	Data type	Function
s	constant, no variables possible	Character constants, max. 12 letters (source).
d	WORD	Starting 16-bit area for storing 6-word ASCII code (destination).

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	-	-	-	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the last area for ASCII code exceeds the limit (6 words: six 16-bit areas).
R9008	%MX0.900.8	for an instant	

ASCII Hex-Code

b3	b2	b1	b0		b7							
					b6	0	0	0	0	1	1	1	1
					b5	0	0	1	1	0	0	1	1
					b4	0	1	0	1	0	1	0	1
				ASCII HEX code	Most significant digit	0	1	2	3	4	5	6	7
0	0	0	0	Least significant digit	0	NUL	DEL	SPACE	0	@	P		p
0	0	0	1		1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0		2	STX	DC2	”	2	B	R	b	r
0	0	1	1		3	ETX	DC3	#	3	C	S	c	s
0	1	0	0		4	EOT	DC4	$	4	D	T	d	t
0	1	0	1		5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0		6	ACK	SYN	&	6	F	V	f	v
0	1	1	1		7	BEL	ETB	’	7	G	W	g	w
1	0	0	0		8	BS	CAN	(	8	H	X	h	x
1	0	0	1		9	HT	EM	)	9	I	Y	i	y
1	0	1	0		A	LF	SUB	*	:	J	Z	j	z
1	0	1	1		B	VT	ESC	+	;	K	[	k	{
1	1	0	0		C	FF	FS	,	<	L	\	l	「
1	1	0	1		D	CR	GS	–	=	M	]	m	}
1	1	1	0		E	SO	RS	.	>	N	^	n	~
1	1	1	1		F	SI	US	/	?	O	_	o	DEL

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

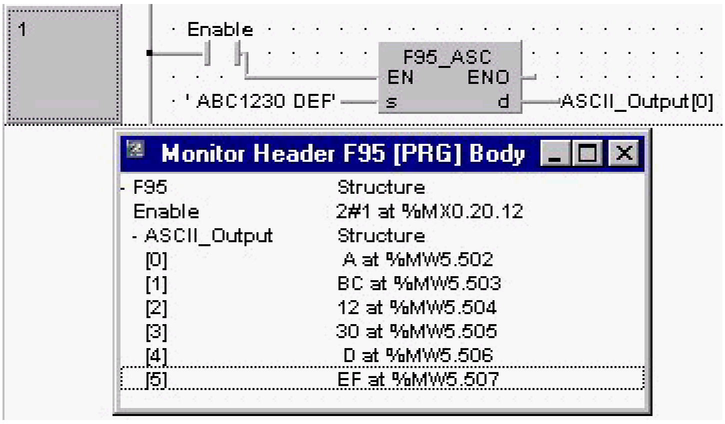
	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	ASCII_Output	ARRAY [0..5] OF WORD	[6(0)]	

Body

When the variable **Enable** is enabled, the character constants entered at the input s are converted to ASCII code and stored in the variable **ASCII_Output**.

Part III F/P Instructions

LD



ST

```
IF Enable THEN
    F95_ASC( s:= 'ABC1230 DEF' ,
        d_Start=> ASCII_Output[0] );
END_IF;
```

Part III F/P Instructions

F235_GRY

16-bit data -> 16-bit Gray code

Steps: 6

Description The function converts a value at input **s** to a gray code value. The result of the conversion is returned at output **d**.

PLC types: **Availability of F235_GRY (see page 930)**

Data types

Variable	Data type	Function
s	INT, WORD	source data to be converted
d	INT, WORD	destination for storing gray codes

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	23	
2	VAR	output_value	INT	0	result: here 28

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F235_GRY(input_value, output_value);
END_IF;
```

F236_DGRY 32-bit data -> 32-bit Gray code

Steps: 8

Description The function converts a value at input **s** to a gray code value. The result of the conversion is returned at output **d**.

PLC types: **Availability of F236_DGRY** **(see page 930)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source data to be converted
d	DINT, DWORD	destination for storing gray code

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	12345678	
2	VAR	output_value	DINT	0	result: here 14832105

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F236_DGRY(input_value, output_value);
END_IF;
```

F237_GBIN 16-bit Gray code -> 16-bit binary data

Steps: 6

Description The function converts a gray-code value at input **s** to binary data. The result of the conversion is returned at output **d**.

PLC types: **Availability of F237_GBIN** **(see page 930)**

Data types

Variable	Data type	Function
s	INT, WORD	source area to gray code
d	INT, WORD	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

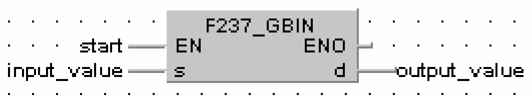
	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	input_value	INT	28	
2	VAR	output_value	INT	0	result: here 23

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F237_GBIN(input_value, output_value);
END_IF;
```

F238_DGBIN

32-bit Gray code -> 32-bit binary data

Steps: 8

Description The function converts a gray-code value at input **s** to binary data. The result of the conversion is returned at output **d**.

PLC types: **Availability of F238_DGBIN (see page 930)**

Data types

Variable	Data type	Function
s	DINT, DWORD	source area for gray code
d	DINT, DWORD	destination area for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	14832105	
2	VAR	output_value	DINT	0	result: here 12345678

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F238_DGBIN(input_value, output_value);
END_IF;
```

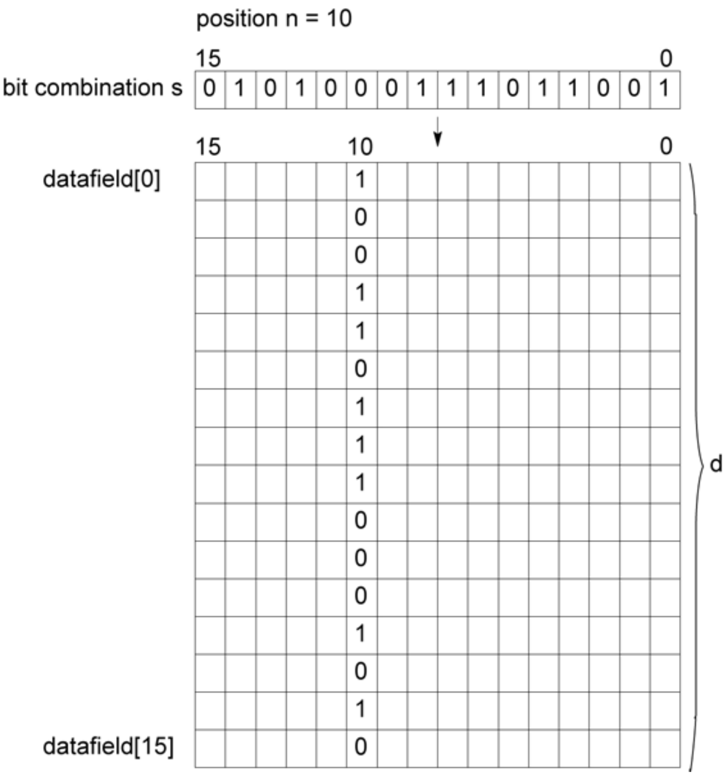
F240_COLM

Bit line to bit column conversion

Steps: 8

Description The function creates a bit column out of a value given at input **s** that is returned within an ARRAY at output **d**. The position of the column in the ARRAY is specified at input **n**. The value assigned at **n** can be between 0 and 15.

The bits of the ARRAY that are not overwritten by the input value (input **s**) are not effected.



PLC types: **Availability of F240_COLM (see page 930)**

Data types

Variable	Data type	Function
s	INT, WORD	source
n	INT, WORD	specifies bit position
d	ARRAY **[0..15] of** INT **or** WORD	destination area that will be rewritten with bit column

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the bit position specified at input n is not between 0 and 15 - the conversion operation results in an overflow of the address area at output d.
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

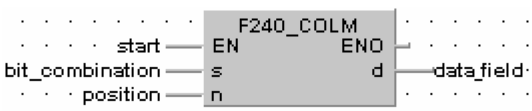
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit_combination	WORD	16#FFFF	
2	VAR	position	INT	15	acceptable values 0.. 15
3	VAR	data_field	ARRAY [0..15] OF WORD	[16(0)]	result: bit 16 (highest-value bit) of the array's elements is set to 1 (TRUE)

In this example **bit_combination** and **position** are declared as input variables. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

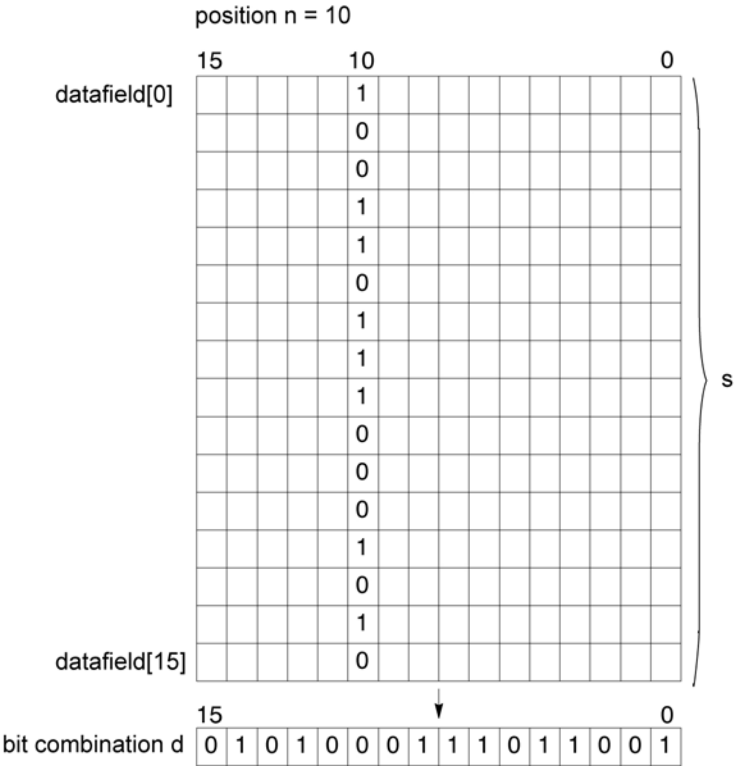
```
IF start THEN
    F240_COLM( s:= bit_combination,
         n:= position,
         d=> data_field );
END_IF;
```

F241_LINE

Bit column to bit line conversion

Steps: 8

Description The function converts a bit column out of an ARRAY at input **s** and returns it at output **d**. The position at which the conversion takes place is specified at input **n**. The value assigned at input **n** should be between 0 and 15.



PLC types: **Availability of F241_LINE (see page 930)**

Data types

Variable	Data type	Function
s	ARRAY **[0..15] of** INT **or** WORD	source area where bit column will be read
n	INT, WORD	specifies bit position
d	INT, WORD	destination area for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the bit position specified at input n is not between 0 and 15 - an overflow of the address area at input s occurs.
R9008	%MX0.900.8	for an instant	

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

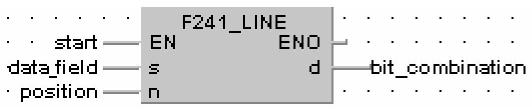
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit_combination	WORD	0	result: here 16#FFFF
2	VAR	position	INT	15	acceptable values 0.. 15
3	VAR	data_field	ARRAY [0..15] OF WORD	[16(16#8000)]	highest value bit of the array's elements is set to 1 (TRUE)

In this example **bit_combination** and **position** are declared as input variables. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
    F241_LINE( s:= data_field ,
           n:= position ,
           d=> bit_combination );
END_IF;
```

F327_INT

Floating point data -> 16-bit integer data (the largest integer not exceeding the floating point data)

Steps: 8

Description The function converts a floating point data at input **s** in the range -32767.99 to 32767.99 into integer data (including +/- sign). The result of the function is returned at output **d**.

The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

PLC types: **Availability of F327_INT (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
s2	INT, WORD	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input **s** is not a REAL number, or the converted result exceeds the 16-bit area at output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	- the result is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1.234	
2	VAR	output_value	INT	0	result: here -2

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. It converts the floating point value -1.234 into the whole number value -2, which is transferred to the variable **output_value** at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD

ST

```
IF start THEN
    F327_INT(input_value, output_value);
END_IF;
```

F328_DINT

Floating point data -> 32-bit integer data (the largest integer not exceeding the floating point data)

Steps: 8

Description The function converts a floating point data at input **s** in the range -2147483000 to 214783000 into integer data (including +/- sign). The result of the function is returned at output **d**.

The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

PLC types: **Availability of F328_DINT (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
d	DINT, DWORD	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input **s** is not a REAL number, or the converted result exceeds the 32-bit area of output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	- the result is 0.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1234567.89	
2	VAR	output_value	DINT	0	result: here -1234568

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. It converts the floating point value -1234567.89 into the whole number value -1234568, which is transferred to the variable **output_value** at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD

ST

```
IF start THEN
    F328_DINT(input_value, output_value);
END_IF;
```

F333_FINT

Rounding the first decimal point down

Steps: 8

Description The function rounds down the decimal part of the real number data and returns it at output **d**.

The converted whole-number value at output **d** is always less than or equal to the floating-point value at input **s**:

- If a positive floating-point value is at the input, a positive pre-decimal point value is returned at the output.
- If a negative floating-point value is at the input, the next smallest pre-decimal point value is returned at the output.
- If the negative floating-point value has only zeros after the decimal point, its pre-decimal point position is returned.

PLC types: **Availability of F333_FINT (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input **s** is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

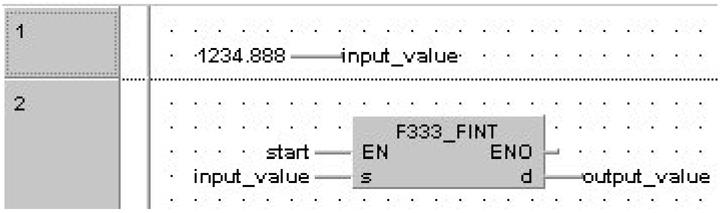
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here 1234.000

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body

The value 1234.888 is assigned to the variable **input_value.** When the variable **start** is set to TRUE, the function is carried out. It rounds down the **input_value** after the decimal point and returns the result (here: 1234.000) at the variable **output_value**.

LD



ST

```
input_value:=1234.888;
IF start THEN
    F333_FINT(input_value, output_value);
END_IF;
```

F334_FRINT

Rounding the first decimal point off **Steps: 8**

Description The function rounds off the decimal part of the real number data and returns it at output **d**.

If the first post-decimal digit is between 0..4, the pre-decimal value is rounded down. If the first post-decimal digit is between 5..9, the pre-decimal value is rounded up.

PLC types: **Availability of F334_FRINT (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input **s** is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	1234.567	
2	VAR	output_value	REAL	0.0	result: here 1235.000

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. It rounds off the **input_value** = 1234.567 after the decimal point and returns the result (here: 1235.000) at the variable **output_value**.

LD

ST

```
IF start THEN
    F334_FRINT(input_value, output_value);
END_IF;
```

F335_FSIGN

Floating point data sign changes (negative/positive conversion)

Steps: 8

Description The function changes the sign of the floating point value at input **s** and returns the result at output **d**.

PLC types: **Availability of F335_FSIGN (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

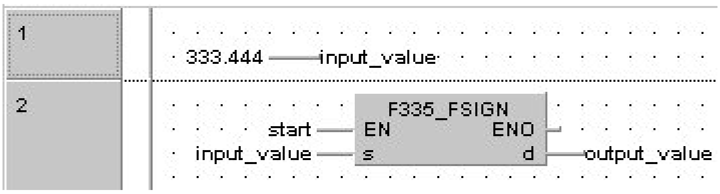
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here -333.444

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body

The value 333.4 is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function is carried out. The **output_value** is then -333.4.

LD



ST

```
input_value:=333.444;
IF start THEN
    F335_FSIGN(input_value, output_value);
END_IF;
```

F337_RAD

Conversion of angle units (Degrees -> Radians)

Steps: 8

Description The function converts the value of an angle entered at input **s** from degrees to radians and returns the result at output **d**.

PLC types: **Availability of F337_RAD (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source angle data (degrees), 2 words
d	REAL	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	180.0	angle in °
2	VAR	output_value	REAL	0.0	angle in radians result: here 3.14159

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F337_RAD(input_value, output_value);
END_IF;
```

F338_DEG

Conversion of angle units (Radians -> Degrees)

Steps: 8

Description The function converts the value of an angle entered at input **s** from radians to degrees and returns the result at output **d**.

PLC types: **Availability of F338_DEG (see page 932)**

Data types

Variable	Data type	Function
s	REAL	source angle data (radians), 2 words
d	REAL	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	- the result is 0.
R9009	%MX0.900.9	for an instant	- the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	3.14159	angle in radians
2	VAR	output_value	REAL	0.0	angle in ° result: here 180.0

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

ST

```
IF start THEN
    F338_DEG(input_value, output_value);
END_IF;
```

Chapter 21

Selection Instructions

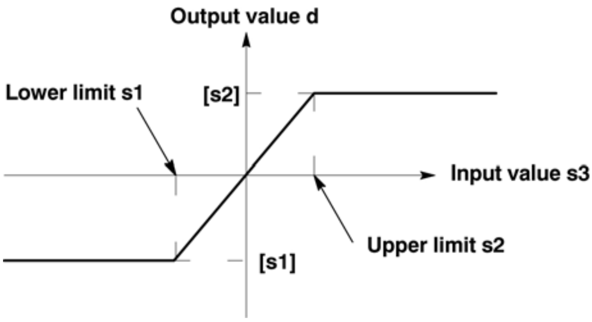
F285_LIMT — 16-bit data upper and lower limit control

Steps: 10

Description The function compares the input value at input **s3** with a lower and an upper limit. The lower limit is specified at input **s1**, and the upper limit at input **s2**. The result of the function is returned at output **d** as follows.

- If the input value at **s3** < **s1**, the lower limit at input **s1** is returned at output **d**.
- If the input value at **s3** < **s2**, the upper limit at input **s2** is returned at output **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, the input value **s3** is returned unchanged at output **d**.

If you want to control the output value solely via the upper value **s2**, set -32768 or 16#8000 for the lower limit **s1**. To perform lower limit control only, set 32767 or 16#7FFF for the upper limit **s2**.



PLC types: **Availability of F285_LIMT (see page 931)**

Data types

Variable	Data type	Function
s1	INT, WORD	the area where the lower limit is stored or the lower limit data
s2	INT, WORD	the area where the upper limit is stored or the upper limit data
s3	INT, WORD	the area where the input value is stored or the input value data
d	INT, WORD	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at s1 > s2.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- the result of processing is between the upper and lower limits.

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

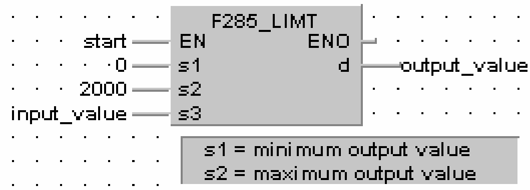
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	2222	
2	VAR	output_value	INT	0	result: here 2000

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. The constant 0 (lower limit) and 2000 (upper limit) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



ST

```
IF start THEN
    F285_LIMT( 0, 2000, input_value, output_value);
END_IF;     (* 0=lower limit, 2000=upper limit *)
```

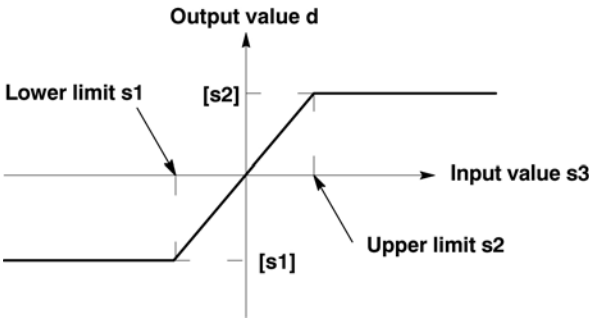
F286_DLIMT

32-bit data upper and lower limit control **Steps: 10**

Description The function compares the input value at input **s3** with a lower and an upper limit. The lower limit is specified at input **s1**, and the upper limit at input **s2.** The result of the function is returned at output **d** as follows:

- If the input value at **s3** < **s1**, the lower limit at input **s1** is returned at output **d**.
- If the input value at **s3** < **s2**, the upper limit at input **s2** is returned at output **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, the input value **s3** is returned unchanged at output **d**.

If you want to control the output value solely via the upper value **s2**, set -2147483648 or 16#80000000 for the lower limit **s1.** To perform lower limit control only, set 2147483647 or 16#7FFFFFFF the upper limit **s2.**



PLC types: **Availability of F286_DLIMT (see page 931)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	the area where the lower limit is stored or the lower limit data
s2	DINT, DWORD	the area where the upper limit is stored or the upper limit data
s3	DINT, DWORD	the area where the input value is stored or the input value data
d	DINT, DWORD	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the value at s1 > s2.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- the result of processing is between the upper and lower limits.

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

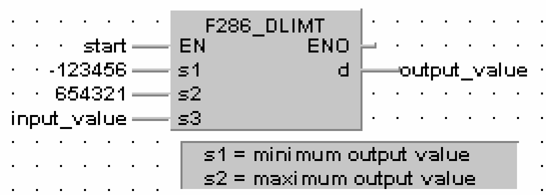
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	0	
2	VAR	output_value	DINT	0	

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. The constant -123456 (lower limit) and 654321 (upper limit) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



ST

```
IF start THEN
    F286_DLIMT( 123456, 654321, input_value, output_value);
END_IF;       (* 123456= lower limit, 654321=upper limit *)
```

Chapter 22

Date and Time Instructions

F138_TIMEBCD_TO_SECBCD

h:min:s -> s conversion

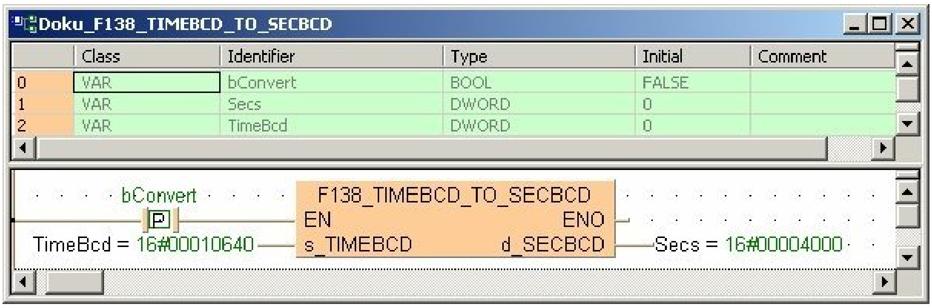
Steps: 7

Description

Converts the hours, minutes, and seconds data stored in the 32-bit area specified by **s** to seconds data if the trigger **EN** is in the ON-state.

The converted seconds data is stored in the 32-bit area specified by **d**. All hours, minutes, and seconds data to convert and the converted seconds data is BCD. The max. data input value is 9,999 hours, 59 minutes and 59 seconds, which will be converted to 35,999,999 seconds in BCD format.

Example



PLC types: **Availability of F138_TIMEBCD_TO_SECBCD (see page 928)**

Data types

Variable	Data type	Function
s_TIMEBCD	DWORD	source area for storing hours, minutes and seconds data
d_SECBCD	DWORD	destination area for storing converted seconds data

Operands

For	Relay				T/C		Register			Const.
s_TIMEBCD	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d_SECBCD	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

F139_SECBCD_TO_TIMEBCD

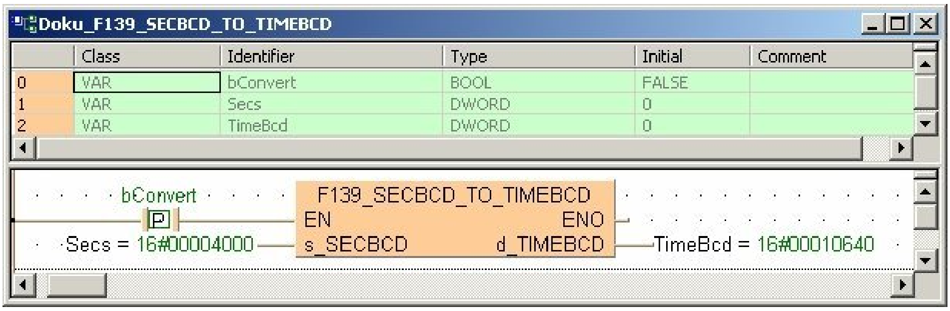
s -> h:min:s conversion

Steps: 5

Description Converts the second data stored in the 32-bit area specified by **s** to hours, minutes, and seconds data if the trigger **EN** is in the ON-state.

The converted hours, minutes, and seconds data is stored in the 32-bit area specified by **d**. The seconds prior to conversion and the hours, minutes, and seconds after conversion are all BCD data. The maximum data input value is 35,999,999 seconds, which is converted to 9,999 hours, 59 minutes and 59 seconds.

Example



PLC types: **Availability of F139_SECBCD_TO_TIMEBCD (see page 928)**

Data types

Variable	Data type	Function
s_SECBCD	DWORD	source area for storing seconds data
d_TIME_BCD	DWORD	destination area for storing converted hours, minutes and seconds data

Operands

For	Relay				T/C		Register			Const.
s_SECBCD	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d_TIME_BCD	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

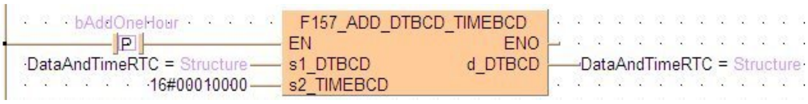
F157_ADD_DTBCD_TIMEBCD

Time addition

Steps: 9

Description The date/clock data (3 words) specified by **s1_DTBCD** and the time data (2 words) specified by **s2_TIMEBCD** are added together if the trigger **EN** is in the ON-state. The result is stored in the area (3 words, same format as **s1_DTBCD**) specified by **d_DTBCD**. This instruction handles all data in BCD format.

Symbol



You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056 for FP2/2SH and FP10/10S/10SH) for the operand d_DTBCD. These registers are factory built-in calendar timer values. To change the built-in calendar timer value, first store the added result in other memory areas and transfer them to the special data registers using SET_RTC_DTBCD (see page 677) instruction.

Example 1: clock/calendar data in DTBCD format	**DUT Member**	**Result**
August 1, 1992, Time: 14:23:31 (hours:minutes:seconds)	MinSec	16#2331 (minutes/seconds)
	DayHour	16#0114 (day/hour)
	YearMon	16#9208 (year/month)
Example 2: time data in TIMEBCD format		
32 hours; 50 minutes; and 45 seconds		16#00325045 hex (hours/minutes/seconds)

PLC types: **Availability of F157_ADD_DTBCD_TIMEBCD (see page 929)**

Data types

Variable	Data type	Function
s1_DTBCD	DTBCD	augend, time and date, values in BCD format
s2_TIMEBCD	DWORD	addend, 32-bit area for storing time data in BCD format
d_DTBCD	DTBCD	sum in BCD format

Operands

For	Relay				T/C		Register			Const.
s1_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_TIMEBCD	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

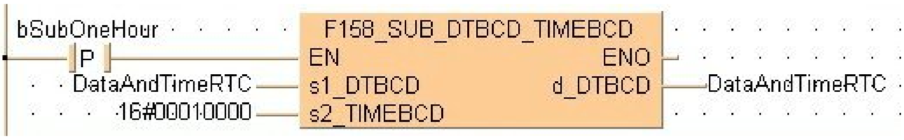
F158_SUB_DTBCD_TIMEBCD

Time subtraction

Steps: 9

Description Subtracts time data (2 words) specified by **s2_TIMEBCD** from the date/clock data (3 words) specified by **s1_DTBCD** if the trigger **EN** is in the ON-state. The result is stored in the area (3 words, same format than **s1_DTBCD**) specified by **d_DTBCD**. All the data used in this instruction are handled in form of BCD.

Symbol



You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056 for FP2/2SH and FP10/10S/10SH) for the operand d_DTBCD. These registers are factory built-in calendar timer values. To change the built-in calendar timer value, first store the subtraction result in other memory areas and transfer them to the special data registers using SET_RTC_DTBCD (see page 677) instruction.

Example 1: clock/calendar data in DTBCD format	DUT Member	Result
August 1, 1992, Time: 14:23:31 (hour:minutes:seconds)	MinSec	16#2331 (minutes/seconds)
	DayHour	16#0114 (day/hour)
	YearMon	16#9208 (year/month)
Example 2: time data in TIMEBCD format		
32 hours; 50 minutes; and 45 seconds		16#00325045 hex (hours/minutes/seconds)

PLC types: **Availability of F158_SUB_DTBCD_TIMEBCD (see page 929)**

Data types

Variable	Data type	Function
s1_DTBCD	DTBCD	minuend, time and date, values in BCD format
s2_TIMEBCD	DWORD	subtrahend, 32-bit area for storing time data in BCD format
d_DTBCD	DTBCD	result in BCD format

Operands

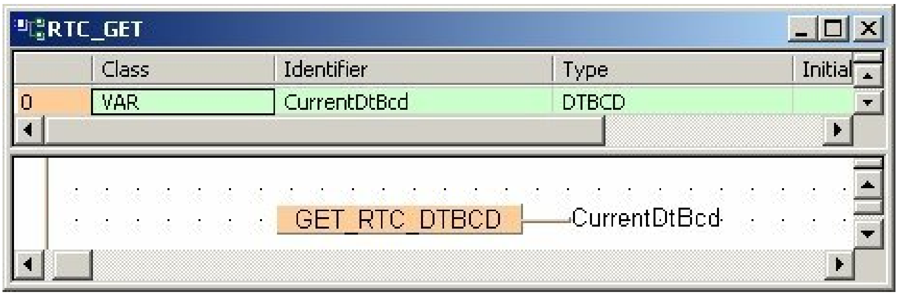
For	Relay				T/C		Register			Const.
s1_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_TIMEBCD	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

GET_RTC_DTBCD Read Real-Time Clock

Description Use this PLC independent instruction to read the real-time clock data from the PLC. When the instruction is carried out, the values from the special data registers DT90054 to DT90056 (DT9054 to DT9056) are transferred to the data unit type DTBCD. You can also use the system variables to set the RTC. For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 4).

If you require an enable input (EN) and an enable output (ENO): Insert the EN/ENO instruction by selecting **[Insert with EN/ENO]** from the OP/FUN/FB selection in the LD, FBD and IL editors. To facilitate reusing the Enable (E_) instruction, it will then appear as such under "Recently used" in the pop-up menu.

Example:



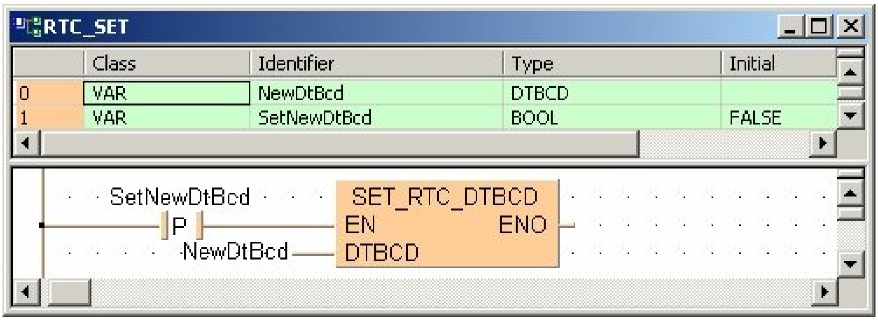
SET_RTC_DTBCD

Set the Real-Time Clock

Steps: 3

Description Use this PLC independent instruction to write date and time data in BCD format (DTBCD) to the real-time clock. When the variable **SetNewDtBcd** is set to TRUE, the values from the data unit type DTBCD are transferred to the special data registers DT90054 to DT90056 (DT9054 to DT9056) and the value 16#8000 is written to the special data register DT90058 (DT9058) to set the real-time clock of the PLC. You can also use the system variables to set the RTC. For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 4).

Example



Chapter 23

Bistable Instructions

KEEP

Serves as a relay with set and reset inputs

Steps: 1

Description KEEP serves as a relay with set and reset points.

When the **SetTrigger** turns ON, output of the specified relay goes ON and maintains its condition. Output relay goes OFF when the **ResetTrigger** turns ON. The output relay's ON state is maintained until a **ResetTrigger** turns ON regardless of the ON or OFF states of the **SetTrigger**. If the **SetTrigger** and **ResetTrigger** turn ON simultaneously, the **ResetTrigger** is given priority.

PLC types: **Availability of KEEP (see page 933)**

Data types

Variable	Data type	Function
Set Trigger	BOOL	sets Address output, i.e. turns in ON
Reset Trigger	BOOL	resets Address output, i.e. turns it OFF
Address	BOOL	specifed relay whose status (set or reset) is kept

Operands

For	Relay				T/C		Register			Constant
Set Trigger, Reset Trigger	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

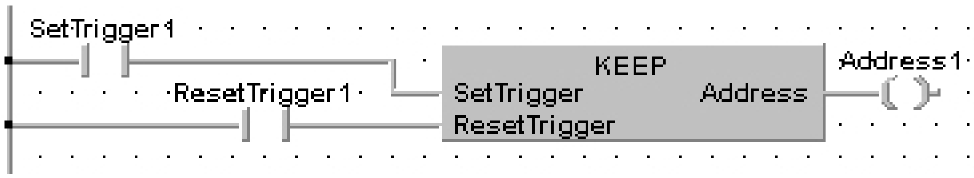
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are required for programming the function are declared in the POU header.

	Class	Identifier	Type	Initial	Comment
0	VAR	SetTrigger1	BOOL	FALSE	Set output
1	VAR	ResetTrigger1	BOOL	FALSE	Reset output
2	VAR	Address1	BOOL	FALSE	Output

Body LD



ST

```
Address1:=KEEP(SetTrigger1, ResetTrigger1);
```

SET

SET, RESET

Steps: 3

Description SET: When the execution conditions have been satisfied, the output is turned on, and the on status is retained.

RST: When the execution conditions have been satisfied, the output is turned off, and the off status is retained.

- You can use relays with the same number as many times as you like with the **SET** and **RST** instructions. (Even if a total check is run, this is not handled as a syntax error.)
- When the **SET** and **RST** instructions are used, the output changes with each step during processing of the operation.
- To output a result while operation is still in progress, use a partial I/O update instruction (**F143**).
- The output destination of a **SET** instruction is held even during the operation of an **MC** instruction.
- The output destination of a **SET** instruction is reset when the mode is changed from RUN to PROG. or when the power is turned off, except when a hold type internal relay is specified as the output destination.
- Placing a **DF** instruction (or specifying a rising edge in LD) before the **SET** and **RST** instructions ensures that the instruction is only executed at a rising edge.

Relays:

- Relays can be turned off using the **RST** instruction.
- Using the various relays with the **SET** and **RST** instructions does not result in double output.
- It is not possible to specify a pulse relay (P) as the output destination for a **SET** or **RST** instruction.

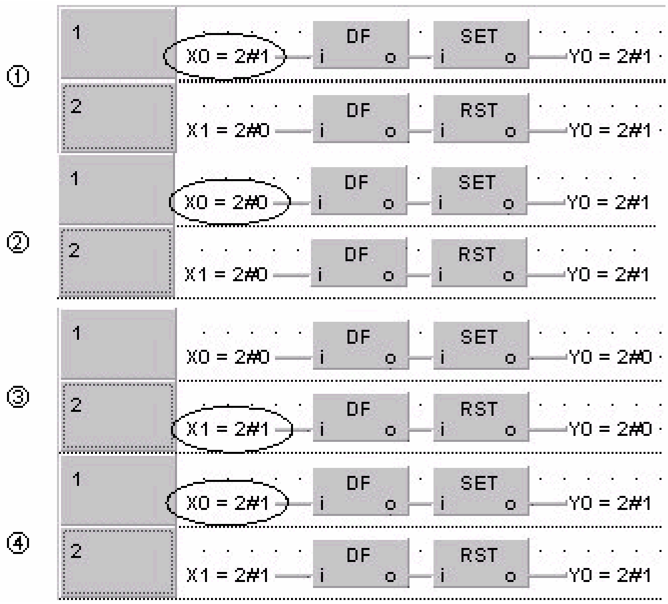
Operands

For	Relay				T/C		Register			Constant
SET **RST**	-	Y	R	L	-	-	-	E	-	-

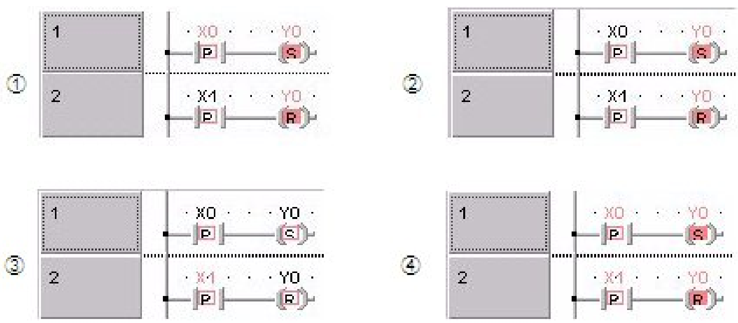
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list). Since addresses are assigned directly using FP addresses, no POU header is necessary.

Body Using the DF command or specifying a rising edge refines the program by making the programming step valid for one scan only:
(1) When the input X0 is activated, the output Y0 is set.
(2) When the input X0 is turned off, the output Y0 remains set.
(3) When the input X1 is activated, the output Y0 is reset.
(4) When the input X0 is reactivated, the output Y0 is set.

FBD



LD In ladder diagram, specify a rising edge in the contact and SET or RESET in the coil:



ST

```
(*TRUE and FALSE are assigned to Y0*)
IF DF(X0) THEN
      Y0:= TRUE;
END_IF;

IF DF(X1) THEN
      Y0:= FALSE;
END_IF;
```

Chapter 24

Edge Detection Instructions

DF Leading edge differential

Steps: 1

Description **DF** is a leading edge differential instruction. The **DF** instruction executes and turns ON output **o** for a singular scan duration if the trigger **i** changes from an OFF to an ON state.

PLC types: **Availability of DF (see page 924)**

Data types

Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

Example In this example the function DF is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

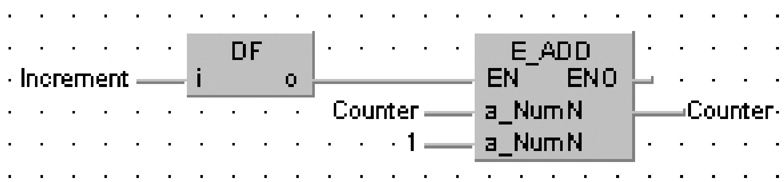
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Increment	BOOL	FALSE	
1	VAR	Counter	INT	0	

Body

Each rising edge at the input **Increment** increments the counter.

LD



ST

When programming with structured text, enter the following:

```
IF DF(Increment) THEN
    Counter:=Counter+1;
END_IF;
```

DFN

Trailing edge differential

Steps: 1

Description The **DFN** instruction executes and turns ON output **o** for a single scan duration if the trigger **i** changes from an ON to an OFF state.

PLC types: **Availability of DFN (see page 924)**

Data types

Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

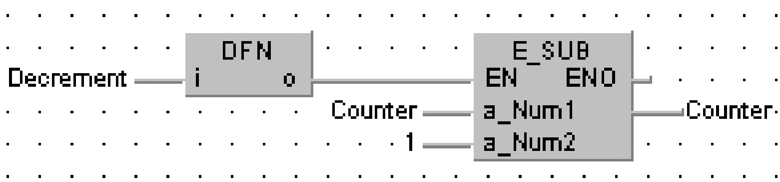
Example In this example the function DFN is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Decrement	BOOL	FALSE	
1	VAR	Counter	INT	0	

Body Each falling edge at the input **Decrement** decrements the couter.

LD



ST When programming with structured text, enter the following:

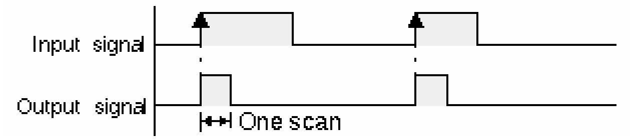
```
IF DFN(Decrement) THEN
    Counter:=Counter-1;
END_IF;
```

DFI

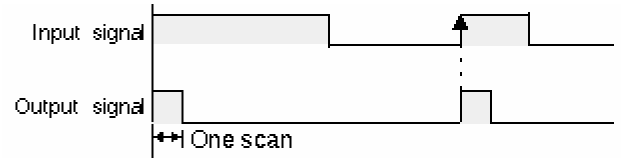
Leading edge differential (initial execution type)

Steps: 1

Description When a leading edge of the input signal (input **i**) is detected, this function changes the status of the output signal (output **o**) to TRUE for the duration of the scan.



Detection of the input signal's leading edge is also assured at the first scan.



You may use an unlimited number of DFI functions.

If the input signal = TRUE already when the system is turned on and this signal should not be interpreted as the first leading edge, the DF function must be used instead.

PLC types: **Availability of DFI (see page 924)**

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. (Specific basic JUMP and LOOP instructions are: MC to MCE instruction, JP to LBL instruction, F19 (SJP) to LBL instruction, LOOP to LBL instruction.

Data types

Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	input_value	BOOL	FALSE
1	VAR	output_value	BOOL	FALSE

LD

ST

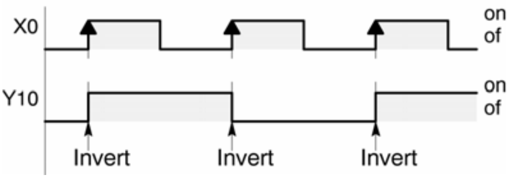
```
output_value:=DFI(input_value);
```

ALT — Alternative out

Description The function inverts the output condition (output **o**) each time the leading edge of the input signal (input **i**) is detected.

When the mode is changed from PROG to RUN or the power is turned on in RUN mode while the input signal is TRUE, a leading edge will not be detected for the first scan.

Time chart



PLC types: **Availability of ALT (see page 923)**

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. (Specific basic JUMP and LOOP instructions are: MC to MCE instruction, JP to LBL instruction, F19_SJP to LBL instruction, LOOP to LBL instruction.

Data types

Variable	Data type
input	BOOL
output	BOOL

Operands

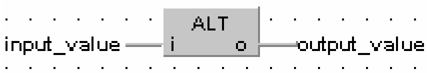
For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	FALSE	
1	VAR	output_value	BOOL	FALSE	

LD



ST

```
output_value:=(ALT(input_value));
```

Chapter 25

Counter Instructions

CT_FB

Down Counter

Steps: 3

Description Counters realized with the CT_FB function block are down counters. The count area **SV** (set value) is 1 to 32767.

For the CT_FB function block declare the following:

Count **count contact**
each time a rising edge is detected at Count, the value 1 is subtracted from the elapsed value **EV** until the value 0 is reached

Reset **reset contact**
each time a rising edge is detected at Reset, the value 0 is assigned to **EV** and the signal output **C** is reset; each time a falling edge is detected at Reset, the value at **SV** is assigned to **EV**

SV **set value**
value of **EV** after a reset procedure

C **signal output**
is set when **EV** becomes 0

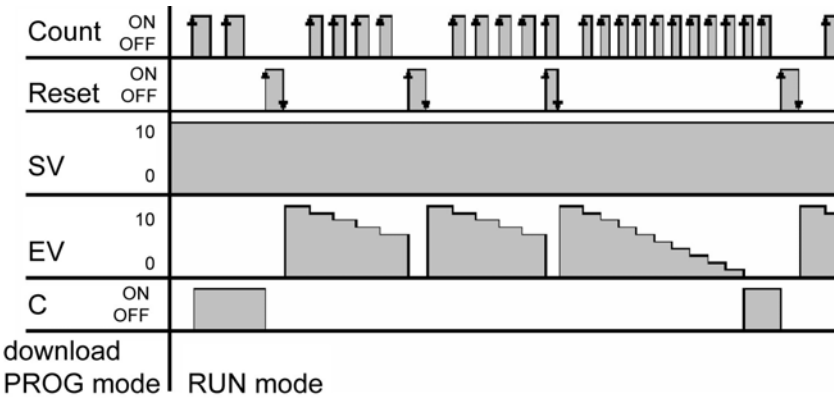
EV **elapsed value**
current counter value

PLC types: **Availability of CT_FB (see page 924)**

Data types

Variable	Data type	Function
Count	BOOL	count contact (down)
Reset	BOOL	reset contact
SV	INT, WORD	set value
C	BOOL	set when EV = 0
EV	INT, WORD	elapsed value

Time Chart



☞

- **In order to work correctly, the CT_FB function block needs to be reset each time before it is used.**
- **The number of available counters is limited and depends on the settings in the system registers 5 and 6. The compiler assigns a NUM* address to every counter instance. The addresses are assigned counting downwards, starting at the highest possible address.**
- **The basic CT (see page 695) function (down counter) uses the same NUM* address area (Num* input). In order to avoid errors (address conflicts), the CT function and the CT_FB function block should not be used together in a project.**

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

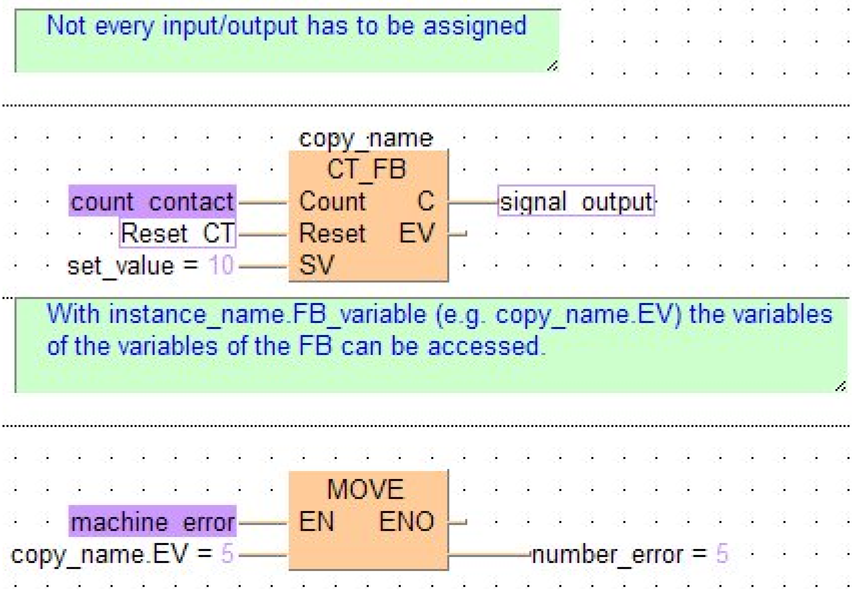
POU Header

All input and output variables which are used for programming the function block CT_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.

	Class	Identifier	Type	Initial
0	VAR	copy_name	CT_FB	
1	VAR	set_value	INT	10
2	VAR	signal_output	BOOL	FALSE
3	VAR	count_contact	BOOL	FALSE
4	VAR	Reset_CT	BOOL	FALSE
5	VAR	machine_error	BOOL	FALSE
6	VAR	number_error	INT	0

Body This example uses variables. You may also use constants for the input variables. Each rising edge detected at **count_contact** the value 1 is subtracted from **set_value**. **Signal_output** is set to TRUE if set_value becomes zero.

LD



CT Counter

Description Decrements a preset counter. The function has the following parameters: Count, Reset, Num*, SV, and C. Their functions are listed in the Data types table below.

When the **Reset** input is on, the set value (**SV**) is reset to the value assigned to it. The set value can be set to a decimal constant from 0 to 32767.

When the **Count** input changes from off to on, the set value begins to decrement. When this value reaches 0, the counter output (**C**) turns on.

If the **Count** input and **Reset** input both turn on at the same time, the **Reset** input is given priority.

If the **Count** input rises and the **Reset** input falls at the same time, the count input is ignored and preset is executed.

PLC types: **Availability of CT (see page 924)**

This function does not require a variable at the output "C".

Data types

Variable	Data type	Function
Count	BOOL	subtracts 1 from the set value each time it is activated
Reset	BOOL	resets the counter when it is ON
Num*	decimal constant	number assigned to the counter (see System Register 5)
SV	INT, WORD	set value is the number the counter starts subtracting from
C	BOOL	the counter turns on when it reaches the SV

Operands

For	Relay				T/C		Register			Constant
Count	X	Y	R	L	T	C	-	-	-	-
Reset	X	Y	R	L	T	C	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
C	-	Y	R	L	-	-	-	-	-	-
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

■ Details about points of Down Counter CT:

Type	Number of points	Nos. that can be used
FP-Sigma	24 points	1000 to 1023

The number of counter points can be changed using System Register 5. The number of points can be increased up to 1,024 with the FP-Sigma. Be aware that increasing the number of counter points decreases the number of timer points.

For all models there is a hold type, in which the counter status is retained even if the power supply is turned off, or if the mode is switched from RUN to PROG, and a non-hold type, in which the counter is reset under these conditions. System register 6 can be used to specify a non-hold type.

■ Set Value and Elapsed Value area

At the fall time when the reset input goes from on to off, the value of the set value area (SV) is preset in the elapsed value area (EV).

When the reset input is on, the elapsed value is reset to 0.

When the count input changes from off to on, the set value begins to decrement, and when the elapsed value reaches 0, the counter contact Cn (n is the counter number) turns on.

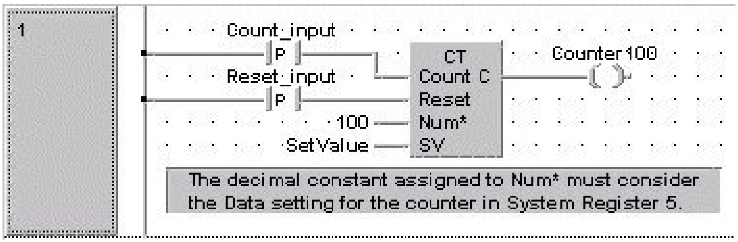
Example In this example the function CT is programmed in ladder diagram (LD) and structured text (ST).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	Count_input	BOOL	FALSE	Listed in Global Variable List with IEC Address %IX0.0
1	VAR_EXTERNAL	Reset_input	BOOL	FALSE	Listed in Global Variable List with IEC Address %IX0.1 Resets SetValue
2	VAR	SetValue	INT	10	Decrements by one each time Count_input is activated
3	VAR	Counter100	BOOL	FALSE	Turns on when Count_input has been activated 10 times

Body The set value SV is set to 10 when **Reset_input** is activated. Each time **Count_input** is activated, the value of SV decreases by 1. When this value reaches 0, **Counter100** turns on. Num* is assigned the counter number, which must be equal to or greater to the number assigned to Data in System Register 5.

LD



ST

```
Counter100:=CT( Count:= Count_input ,
          Reset:= Reset_input ,
          Num:= 100 ,
          SV:= Setvalue );
              (* Num*, 100 in this example, must be a
constant *)
```

F118_UDC UP/DOWN counter

Steps: 5

Description UD_Trig: DOWN counting if the trigger is in the OFF state. UP counting if the trigger is in the ON state.

Cnt_Trig: Adds or subtracts one count at the leading edge of this trigger.

Rst_Trig: The condition is reset when this signal is on.

The area for the elapsed value d becomes 0 when the leading edge of the trigger is detected (OFF → ON). The value in s is transferred to d when the trailing edge of the trigger is detected (ON → OFF).

s: Preset (Set) value or area for Preset (Set) value.

d: Area for count (elapsed) value.

PLC types: **Availability of F118_UDC (see page 928)**

Data types

Variable	Data type	Function
UD_Trig	BOOL	sets counter to count up (ON) or down (0FF)
Cnt_Trig	BOOL	starts counter
Rst_Trig	BOOL	resets counter
s	INT, WORD	16-bit area or equivalent constant for counter preset value
d	INT, WORD	16-bit area for counter elapsed value

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
UD_Trig, Cnt-Trig, Rst_Trig	X	Y	R	L	T	C	-	-	-	-
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function F118_UDC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

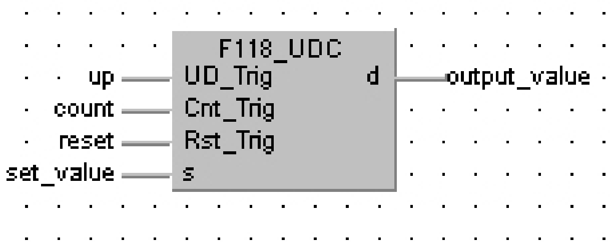
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	up	BOOL	FALSE	declares, if the counter counts up or down
1	VAR	count	BOOL	FALSE	at a rising edge on count the counter counts
2	VAR	reset	BOOL	FALSE	resets the counter to set_value if TRUE
3	VAR	set_value	INT	0	the starting value
4	VAR	output_value	INT	0	the actual value

Body

A rising edge at the input **Cnt_Trig** activates the counter. The boolean variable at the input **UD_Trig** sets the direction of the counter (TRUE = up, FALSE =down). TRUE at the input **Rst_Trig** resets the counter to the starting value.

LD



ST

When programming with structured text, enter the following:

```
output_value:=F118_UDC( UD_Trig:= up, Cnt_Trig:= count,
Rst_Trig:= reset, s:= set_value);
(* output_value contains the count value *)
```

Chapter 26

High Speed Counter and Pulse Output Instructions

F0_MV High-speed counter control

Steps: 5

Description This instruction controls the software reset, disabling of the counter and stops pulse outputs.

PLC types: **Availability of F0_MV (see page 925)**

Data types

Variable	Data type	Function
s	INT, WORD	specifies high-speed counter operation
d	INT, WORD	controls high-speed counter operation at specified address, DT9052/DT90052 (DT90052 for FP0 T32-CP, FP2/2SH and FP10/10S/10SH)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

Nr.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the value of **s** exceeds the limit of specified range.
R9008	%MX0.900.8	for an instant	

Example The following provides generic examples and explanations of F0_MV when used to control high-speed counter functions.

- Perform software reset 16#1(0001)
- Count disable 16#2(0010)
- Stop pulse output 16#8(1000)
- Turn off pulse output and reset elapsed value 16#9(1001)

Enter the control code into the area DT9052/DT90052 of the corresponding channel.

16#0 (0000):

- Software reset operation is not performed.
- Count inputs are accepted.
- Reset input X2 enabled.

Operation This instruction is used when performing the following operations while using the high-speed counter:

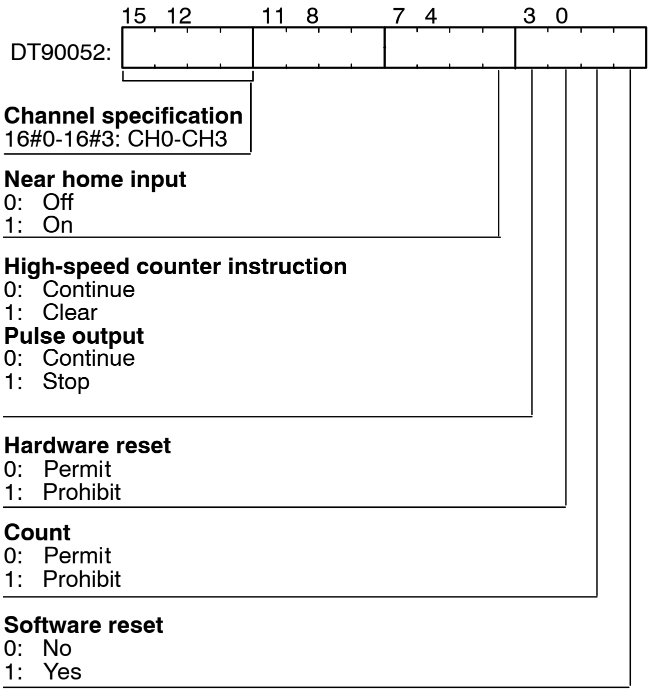
- Performing a software reset.
- Disabling the count.
- Temporarily disabling the hardware reset by the external input X2 and X5.
- Stopping the positioning and pulse outputs.
- Clearing the controls executed with the high-speed counter instructions F166 (see page 717), F167 (see page 720), F171 (see page 723), F172 (see page 732), F173 (see page 736) and F174 (see page 739).
- Setting the near home input during home return operations for decelerating the speed of movement.

When a control code is programmed once, it is saved until the next time it is programmed.

Precautions during prog.

- The hardware reset disable is only effective when using reset inputs (X2 and X5).
- Count disable and software reset during home return operations does not allow near home processing.
- To enable near home processing during home return operations, bit 4 of special data register DT90052 must be set to 1. This bit is saved. Therefore, the near home bit should be reset to 0 right after setting it.

Control code (specify with a hex constant)



Operations of control code:

Software reset operation (bit position 0 of DT90052)

Count input control operation (bit position 1 of DT90052)

Hardware reset control operation (bit position 2 of DT90052)

Control of high-speed counter instructions (bit position 3 of DT90052)

Near home input instructions (bit position 4 of DT90052)

Channel specification (bit positions 12 - 15 of DT90052)

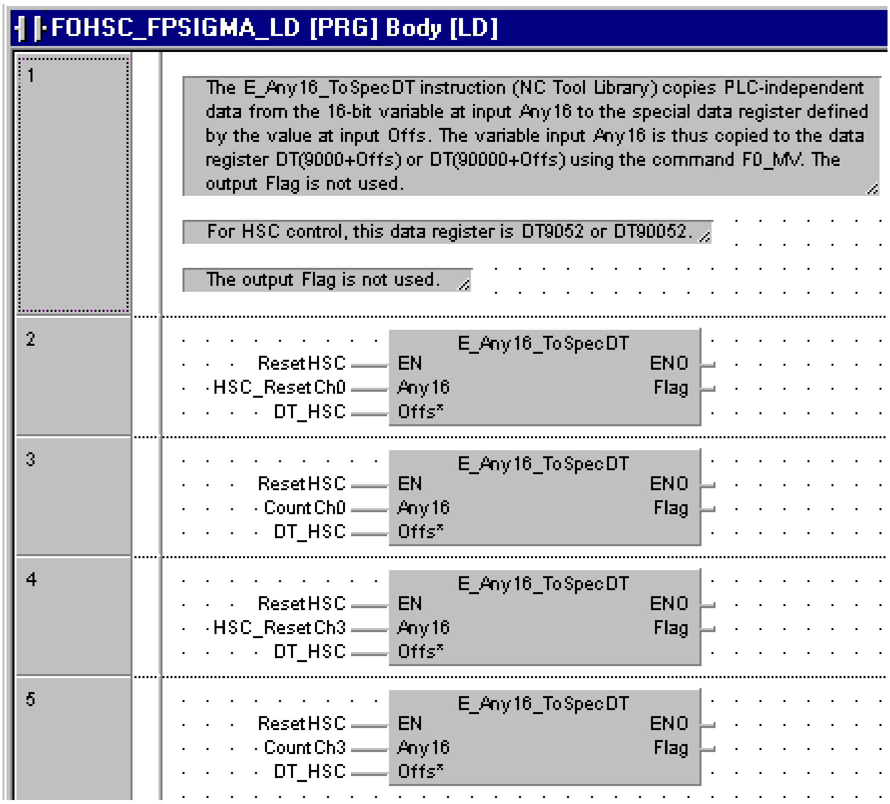
Example In this example the function F0_MV is programmed in ladder diagram (LD).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	ResetHSC	BOOL	FALSE	
1	VAR	HSC_ResetCh0	WORD	16#0001	Resets channel 0
2	VAR	CountCh0	WORD	16#0000	Ch0 starts counting
3	VAR	HSC_ResetCh3	WORD	16#3001	Resets channel 3
4	VAR	CountCh3	WORD	16#3000	Ch3 starts counting
5	VAR_CONSTANT	DT_HSC	INT	52	Copies value into DT90052

LD

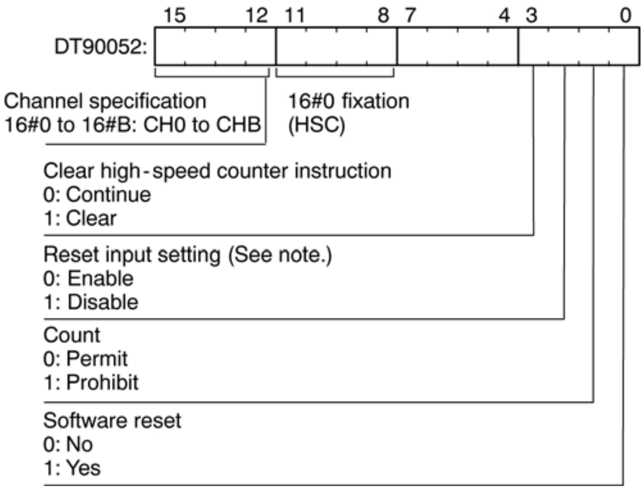


26.1.1.1 Setting the Control Code for High-Speed Counter with FP-X

The area DT90052 for writing channels and control codes is allocated as shown below.

Control codes written with an F0_MV instruction are stored by channel in special registers DT90190 to DT90193.

High-speed counter and pulse output controls flag area of FP-X:



◆ **NOTE**

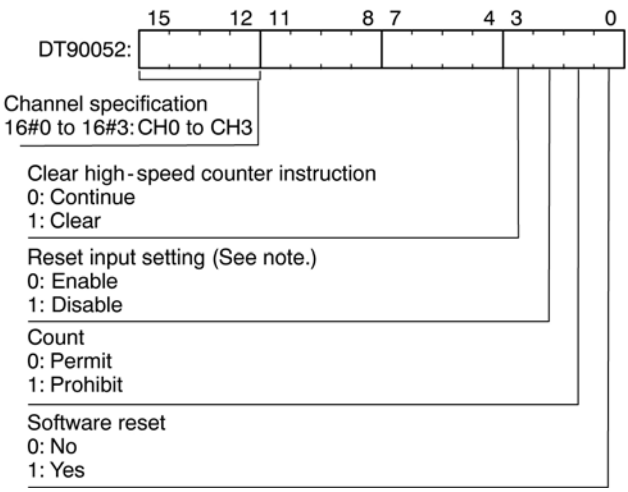
At the reset input setting, you set whether the reset input (X2 or X5), which was assigned by the system register high-speed counter setting, will be enabled or disabled.

26.1.1.2 Setting the Control Code for High-Speed Counter with FP-Sigma

The area DT90052 for writing channels and control codes is allocated as shown below.

Control codes written with an F0_MV instruction are stored by channel in special registers DT90190 to DT90193.

High-speed counter and Pulse output controls flag area of FP-Sigma:



Note:

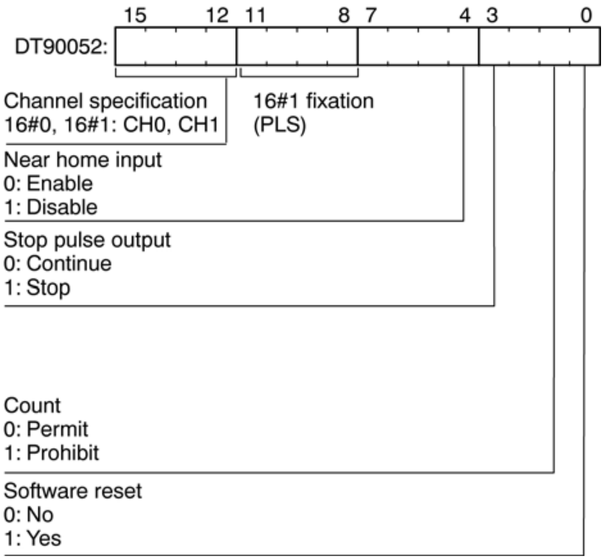
At the reset input setting, you set whether the reset input (X2 or X5), which was assigned by the system register high-speed counter setting, will be enabled or disabled.

26.1.1.3 Setting the Control Code for Pulse Output with FP-X

The area DT90052 for writing channels and control codes is allocated as shown below.

Control codes written with an F0_MV instruction are stored by channel in special registers DT90372 to DT90373.

High-speed counter and pulse output controls flag area of FP-X:

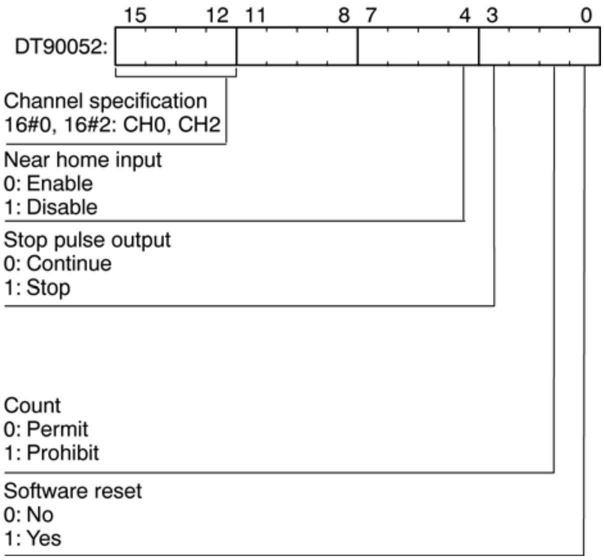


26.1.1.4 Setting the Control Code for Pulse Output with FP-Sigma

The area DT90052 for writing channels and control codes is allocated as shown below.

Control codes written with an F0_MV instruction are stored by channel in special registers DT90190 to DT90192.

High-speed counter and pulse output controls flag area of FP-Sigma:



26.1.2 Reading the Elapsed Value and Setting the Target Values

26.1.2.1 Elapsed Values and Target Values for FP-X

FP-X			Address	System variable
Control Code			DT90052	sys_w_HSC_PLS_ControlFlags
High-speed counter channel no.	**CH0**	Monitoring active:	R9110	sys_b_HSC_CH0_IsActive
		Monitoring value:	DT90360	sys_w_HSC_CH0_ControlFlags
		Elapsed value:	DDT90300	sys_di_HSC_CH0_ElapsedValue
		Target value:	DDT90302	sys_di_HSC_CH0_TargetValue
	CH1	Monitoring active:	R9111	sys_b_HSC_CH1_IsActive
		Monitoring value:	DT90361	sys_w_HSC_CH1_ControlFlags
		Elapsed value:	DDT90304	sys_di_HSC_CH1_ElapsedValue
		Target value:	DDT90306	sys_di_HSC_CH1_TargetValue
	CH2	Monitoring active:	R9112	sys_b_HSC_CH2_IsActive
		Monitoring value:	DT90362	sys_w_HSC_CH2_ControlFlags
		Elapsed value:	DDT90308	sys_di_HSC_CH2_ElapsedValue
		Target value:	DDT90310	sys_di_HSC_CH2_TargetValue
	CH3	Monitoring active:	R9113	sys_b_HSC_CH3_IsActive
		Monitoring value:	DT90363	sys_w_HSC_CH3_ControlFlags
		Elapsed value:	DDT90312	sys_di_HSC_CH3_ElapsedValue
		Target value:	DDT90314	sys_di_HSC_CH3_TargetValue
	CH4	Monitoring active:	R9114	sys_b_HSC_CH4_IsActive
		Monitoring value:	DT90364	sys_w_HSC_CH4_ControlFlags

FP-X			Address	System variable
		Elapsed value:	DDT90316	sys_di_HSC_CH4_ElapsedValue
		Target value:	DDT90318	sys_di_HSC_CH4_TargetValue
	CH5	Monitoring active:	R9115	sys_b_HSC_CH5_IsActive
		Monitoring value:	DT90365	sys_w_HSC_CH5_ControlFlags
		Elapsed value:	DDT90320	sys_di_HSC_CH5_ElapsedValue
		Target value:	DDT90322	sys_di_HSC_CH5_TargetValue
	CH6	Monitoring active:	R9116	sys_b_HSC_CH6_IsActive
		Monitoring value:	DT90366	sys_w_HSC_CH6_ControlFlags
		Elapsed value:	DDT90324	sys_di_HSC_CH6_ElapsedValue
		Target value:	DDT90326	sys_di_HSC_CH6_TargetValue
	CH7	Monitoring active:	R9117	sys_b_HSC_CH7_IsActive
		Monitoring value:	DT90367	sys_w_HSC_CH7_ControlFlags
		Elapsed value:	DDT90328	sys_di_HSC_CH7_ElapsedValue
		Target value:	DDT90330	sys_di_HSC_CH7_TargetValue
	CH8	Monitoring active:	R9118	sys_b_HSC_CH8_IsActive
		Monitoring value:	DT90368	sys_w_HSC_CH8_ControlFlags
		Elapsed value:	DDT90332	sys_di_HSC_CH8_ElapsedValue
		Target value:	DDT90334	sys_di_HSC_CH8_TargetValue
	CH9	Monitoring active:	R9119	sys_b_HSC_CH9_IsActive
		Monitoring value:	DT90369	sys_w_HSC_CH9_ControlFlags
		Elapsed value:	DDT90336	sys_di_HSC_CH9_ElapsedValue
		Target value:	DDT90338	sys_di_HSC_CH9_TargetValue
	CHA	Monitoring active:	R911A	sys_b_HSC_CHA_IsActive
		Monitoring value:	DT90370	sys_w_HSC_CHA_ControlFlags
		Elapsed value:	DDT90340	sys_di_HSC_CHA_ElapsedValue
		Target value:	DDT90342	sys_di_HSC_CHA_TargetValue
	CHB	Monitoring active:	R911B	sys_b_HSC_CHB_IsActive
		Monitoring value:	DT90371	sys_w_HSC_CHB_ControlFlags
		Elapsed value:	DDT90344	sys_di_HSC_CHB_ElapsedValue
		Target value:	DDT90346	sys_di_HSC_CHB_TargetValue
Pulse output channel no.	CH0	Monitoring active:	R911C	sys_b_PLS_CH0_IsActive
		Monitoring value:	DT90372	sys_w_PLS_CH0_ControlFlags
		Elapsed value:	DDT90348	sys_di_PLS_CH0_ElapsedValue
		Target value:	DDT90350	sys_di_PLS_CH0_TargetValue
	CH2	Monitoring active:	R911D	sys_b_PLS_CH2_IsActive
		Monitoring value:	DT90373	sys_w_PLS_CH2_ControlFlags
		Elapsed value:	DDT90352	sys_di_PLS_CH2_ElapsedValue
		Target value:	DDT90354	sys_di_PLS_CH2_TargetValue

26.1.2.2 Elapsed Values and Target Values for FP-Sigma

These target values are set implicitly by the corresponding F instructions.

FP-Σ			**Address**	**System variable**
Control Code			DT90052	sys_w_HSC_PLS_ControlFlags
High-speed counter channel no.	**CH0**	Monitoring active:	R903A	sys_b_HSC_CH0_IsActive
		Monitoring value:	DT90190	sys_w_HSC_CH0_ControlFlags
		Elapsed value:	DDT90044	sys_di_HSC_CH0_ElapsedValue
		Target value:	DDT90046	sys_di_HSC_CH0_TargetValue
	CH1	Monitoring active:	R903B	sys_b_HSC_CH1_IsActive
		Monitoring value:	DT90191	sys_w_HSC_CH1_ControlFlags
		Elapsed value:	DDT90048	sys_di_HSC_CH1_ElapsedValue
		Target value:	DDT90050	sys_di_HSC_CH1_TargetValue
	CH2	Monitoring active:	R903C	sys_b_HSC_CH2_IsActive
		Monitoring value:	DT90192	sys_w_HSC_CH2_ControlFlags
		Elapsed value:	DDT90200	sys_di_HSC_CH2_ElapsedValue
		Target value:	DDT90202	sys_di_HSC_CH2_TargetValue
	CH3	Monitoring active:	R903D	sys_b_HSC_CH3_IsActive
		Monitoring value:	DT90193	sys_w_HSC_CH3_ControlFlags
		Elapsed value:	DDT90204	sys_di_HSC_CH3_ElapsedValue
		Target value:	DDT90206	sys_di_HSC_CH3_TargetValue
Pulse output channel no.	**CH0**	Monitoring active:	R903A	sys_b_PLS_CH0_IsActive
		Monitoring value:	DT90190	sys_w_PLS_CH0_ControlFlags
		Elapsed value:	DDT90044	sys_di_PLS_CH0_ElapsedValue
		Target value:	DDT90046	sys_di_PLS_CH0_TargetValue
	CH2	Monitoring active:	R903C	sys_b_PLS_CH2_IsActive
		Monitoring value:	DT90192	sys_w_PLS_CH2_ControlFlags
		Elapsed value:	DDT90200	sys_di_PLS_CH2_ElapsedValue
		Target value:	DDT90202	sys_di_PLS_CH2_TargetValue

F162_HC0S

High-speed counter output set

Steps: 7

Description Sets the value specified by **s** as the target value of the high-speed counter if the trigger **EN** is in the ON-state. When the elapsed value (DT9045 and DT9044) of the high-speed counter matches the target value (DT9047 and DT9046), the external output relay specified by **d** turns ON. You can use 8 external output relays (Y0 to Y7).

The target value is stored in special data registers DT9047 and DT9046 when the **F162_HC0S** instruction is executed, and it is cleared when the elapsed value of the high-speed counter matches the target value.

Use 24-bit binary data with sign data for the target value of HSC (FF800000 hex to 007FFFFF hex / -8,388,608 to 8,388,607).

Special internal relay R903A turns ON and stays ON while the **F162_HC0S** instruction is executed and it is cleared when the elapsed value of the high-speed counter reaches the target value.

Even if the reset operation of the high-speed counter is performed after executing the **F162_HC0S** instruction, the target value setting is not cleared until the elapsed value of the high-speed counter reaches the target value.

To reset the external output relay, which is set ON by the **F162_HC0S** instruction, use the F163_HCOR (see page 713) instruction.

You can use the same external output relay specified by the **F162_HC0S** instruction in other parts of program. The system does not register a duplicate use of the same output.

While special internal relay R903A is in ON state, no other high-speed counter instructions **F162_HC0S**, F163_HCOR (see page 713), F164_SPDO (see page 715), F165_CAMO (see page 716) can be executed.

PLC types: **Availability of F162_HC0S (see page 929)**

Data types

Variable	Data type	Function
s	DINT, DWORD	area or equivalent constant for storing target value of high-speed counter
d	BOOL	available external output relay: Y0 to Y7

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	Y	-	-	-	-	-	-	-	-

Example In this example the function F162_HC0S is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	BOOL	FALSE	will be set, when high speed counter reaches 100

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

When programming with structured text, enter the following:

```
IF start THEN
    F162_HC0S( s_Target:= 100 ,
         d=> output_value );
END_IF;
```

F163_HC0R **High-speed counter output reset** **Steps: 7**

Description Sets the value specified by **s** as target value of the high-speed counter if the trigger **EN** is in the ON-state. When the elapsed value (DT9045 and DT9044) of the high-speed counter matches the target value (DT9047 and DT9046), the external output relay specified by **d** turns OFF. You can use 8 external output relays (Y0 to Y7).

When the **F163 (HC0R)** instruction is executed, the target value is stored in special data registers DT9047 and DT9046 and it is cleared when the elapsed value of the high-speed counter matches the target value.

Use 24-bit binary data with sign data for the target value of HSC (FF800000 hex to 007FFFFF hex / -8,388,608 to 8,388,607).

Once the **F163 (HC0R)** instruction is executed, special internal relay R903A turns ON and stays ON. It is cleared when the elapsed value of the high-speed counter reaches the target value.

Even if the reset operation of the high-speed counter is performed after executing the **F163 (HC0R)** instruction, the target value setting is not cleared until the elapsed value of the high-speed counter reaches the target value.

You can use the same external output relay specified by the **F163 (HC0R)** instruction in other parts of program. The system does not register a duplicate use of the same output.

While special internal relay R903A is in ON state, no other high-speed counter instructions F162_HC0S (see page 711), **F163 (HC0R)**, F164_SPD0 (see page 715), F165_CAM0 (see page 716) can be executed.

PLC types: **Availability of F163_HC0R (see page 929)**

Data types

Variable	Data type	Function
s	DINT, DWORD	area or equivalent constant for storing target value of high-speed counter
d	BOOL	available external output relay: Y0 to Y7

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	Y	-	-	-	-	-	-	-	-

Example In this example the function F163_HC0R is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOO	FALSE	activates the function
1	VAR	output_value	BOO	FALSE	will be set, when high speed counter reaches 100

Body

When the variable **start** is set to TRUE, the function is executed.

LD

ST

When programming with structured text, enter the following:

```
IF start THEN
    F163_HC0R( s_Target:= 100 ,
         d=> output_value );
END_IF;
```

F164_SPD0 Pulse output control; Pattern output control

Steps: 3

Description Outputs the pattern of the pulse corresponding to the elapsed value of the HSC.

When the executing condition is ON and the HSC control-flag (R903A) is OFF, this instruction starts operation. This instruction executes pattern output or pulse output corresponding to the data of the data table registered at the data register specified by **s**.

You can use pulse output for positioning with a pulse motor and pattern output for controlling an inverter. When you execute pulse output with this instruction, input the pulse of Y7 directly to HSC or input the encoder output pulse. When you execute pattern output, input the encoder output pulse to HSC. Specify using system register No. 400 whether you will use HSC or not.

It is not possible to execute this instruction without the following settings: input condition to detect a rising edge (0/1), and the HSC flag (R903A) must be reset. The output coils of pattern output are within the 8 outputs Y0 to Y7. The output coil of pulse output is Y7 only. Select either pattern outputs or pulse outputs by the content of the first word of the data table. When you input 0 for one word of the first address (all bits are 0), it will be the pulse output. When you execute pattern output, an error occurs unless the No. of the control steps is 1 to F or unless the No. of control points is 1 to 8. An error occurs when the first target value is not FF800000 to 7FFFFF. An error does not occur when the first target value on and after the second one are not FF800000 to 7FFFFF. The operation, however, is stopped and R903A turns OFF. When the frequency data is "0", pulse output ends. It will also end if the area is exceeded during its execution.

PLC types: **Availability of F164_SPD0 (see page 929)**

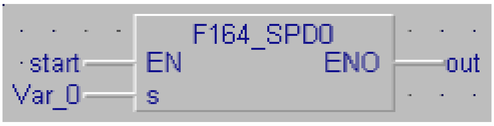
Data types

Variable	Data type	Function
s	INT, WORD	starting 16-bit area for storing control data

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DT	-	-	-

Example Below is an example of a ladder diagram (LD) body for the instruction.



F165_CAM0 Can control

Steps: 5

Description Converts ON/OFF of output specified in the table corresponding to the elapsed value of HSC.

This instruction controls up to 8 cam control outputs (Y0 to Y7), corresponding to the ON/OFF target value of each coil on the table, which is for the control of cam position specified by **s**. The target value is within the range of 0 to 8388607 (i.e. 23 bits of data, 16#00000001 to 16#007FFFFF).

If you execute cam control, you have to specify the operating mode as addition counter.

(If it is not addition counter, you will not be able to execute the control properly.) The target value is 32 steps maximum with FP1-C16, 64 steps maximum with FP1-C24/C40.

If you cancel hard reset, soft reset, and control maximum value you can set the initial pattern at output, set the elapsed value to 0 and restart Cam control. You can output the initial pattern at the start of control. However, you cannot clear the elapsed value to 0.

PLC types: **Availability of F165_CAM0 (see page 930)**

Data types

Variable	Data type	Function
s_Control	INT, WORD	starting 16-bit area for storing control data

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DT	-	-	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

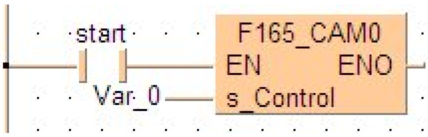
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	Var_0	WORD	0

Body

When the variable **start** is set to TRUE, the function is executed.

LD

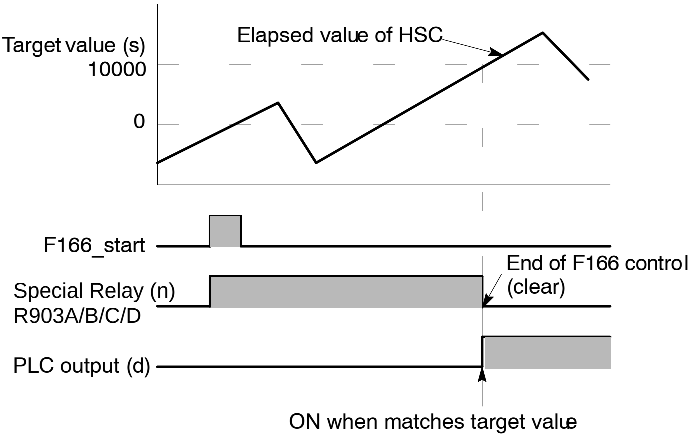


F166_HC1S

Sets Output of High-Speed Counter (4 channels)

Steps: 11

Description If the trigger EN of the instruction F166 has the status TRUE, pulses at the high-speed counter (HSC) will be counted. If the elapsed value of the HSC equals the target value **s**, an interrupt will be executed and the output relay **d** of the PLC will be set. In addition to this the special relay for the HSC **n** (R903A/B/C/D) will be reset and F166 is deactivated.



If the high-speed counter is reset (reset input of HSC from 0 to 1, see system register 400/401 in the project navigator) before the elapsed value has reached the target value **s**, the elapsed value will be reset to zero. F166 remains active and counting restarts at zero. The duplicate use of an external output relay in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected. While the special relay(s) R903A/B/C/D is/are in ON state no other high-speed counter instructions can be executed. FP0 and FP-Sigma provide 4 HSC channels. The channel number is specified by **n** (0 to 3).

n values for		**0**	**1**	**2**	**3**
FP-Sigma	**Elapsed value register**	DDT90044	DDT90048	DDT90200	DDT90204
	Target value register	DDT90046	DDT90050	DDT90202	DDT90206
	Used channel	CH0 of HSC0	CH1 of HSC0	CH0 of HSC1	CH1 of HSC1
	ON during execution	R903A	R903B	R903C	R903D

s values	**dec**	**hex**
FP-Sigma	2,147,483,468	16#80000000
	...	...
	-2,147,483,647	16#7FFFFFFF

d values	value	output
FP-X	0 to 671	Y0 to Y29F

PLC types: **Availability of F166_HC1S** **(see page 930)**

Data types

Variable	Data type	Function
n	DINT, DWORD	the channel no. of the high-speed counter that corresponds to the matching output (n: 0 to 3) For the FP-X: n: 16#0 to 16#B
s	DINT, DWORD	the high-speed counter target value data or the starting address of the area that contains the data
d	BOOL	the output coil that is turned on when the values match (Yn, n: 0 to 7) For the FP-X: Yn, n: 0 to 29F

Operands

For	Relay				T/C		Register			Constant
n	-	-	-	-	-	-	-	-	-	dec. or hex.
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	- if index is too high
R9008	%MX0.900.8	ON	- parameter s exceeds the valid range

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

Global Variables

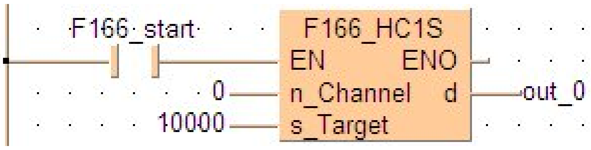
	Class	Identifier	FP Address	IEC Address	Type	Initial	Comment
0	VAR_GLOBAL	out_0	Y0	%QX0.0	BOOL	FALSE	output Y0 of PLC

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	F166_start	BOOL	FALSE	F166 start condition
1	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC

Body When the variable **F166_start** is set to TRUE, the function is executed.

LD



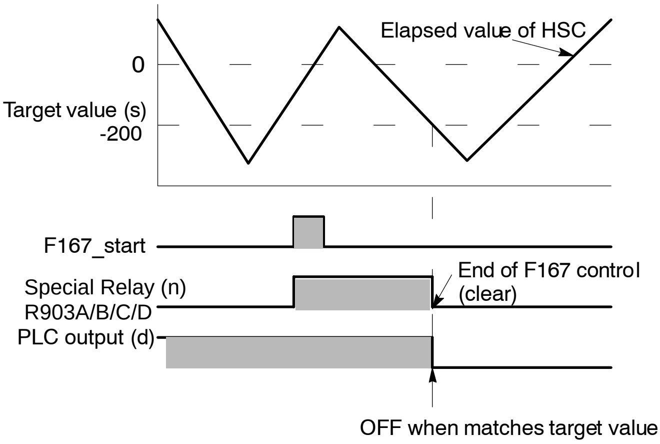
☞ **Assign a number to the input variable (e.g. Monitor → Monitor Header, click the variable, enter the value, press <Enter>) or replace the input variables by numbers.**

F167_HC1R

Resets Output of High-Speed Counter (4 channels)

Steps: 11

Description If the trigger EN of the instruction F167 has the status TRUE, pulses at the high-speed counter (HSC) will be counted. If the elapsed value of the HSC equals the target value **s**, an interrupt will be executed and the output relay **d** of the PLC will be reset. In addition to this the special relay for the HSC **n** (R903A/B/C/D) will be reset and F167 is deactivated.



If the high-speed counter is reset (reset input of HSC from 0 to 1, see system register 400/401 in the project navigator) before the elapsed value has reached the target value **s**, the elapsed value will be reset to zero. F167 remains active and counting restarts at zero. The duplicate use of an external output relay **d** in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected. While the special relay(s) R903A/B/C/D is/are in ON state no other high-speed counter instructions can be executed. FP0 and FP-Sigma provide 4 HSC channels. The channel number is specified by **n** (0 to 3).

n values for		**0**	**1**	**2**	**3**
FP-Sigma	**Elapsed value register**	DDT90044	DDT90048	DDT90200	DDT90204
	Target value register	DDT90046	DDT90050	DDT90202	DDT90206
	Used channel	CH0 of HSC0	CH1 of HSC0	CH0 of HSC1	CH1 of HSC1
	ON during execution	R903A	R903B	R903C	R903D

s values	**dec**	**hex**
FP-Sigma	2,147,483,468	16#80000000
	...	...
	-2,147,483,647	16#7FFFFFFF

d values	value	output
FP-X	0 to 671	Y0 to Y29F

PLC types: **Availability of F167_HC1R (see page 930)**

Data types

Variable	Data type	Function
n	DINT, DWORD	the channel no. of the high-speed counter that corresponds to the matching output (n: 0 to 3) For the FP-X: n: 16#0 to 16#B
s	DINT, DWORD	the high-speed counter target value data or the starting address of the area that contains the data
d	BOOL	the output coil that is turned on when the values match (Yn, n: 0 to 7) For the FP-X: Yn, n: 0 to 29F

Operands

For	Relay				T/C		Register			Constant
n	-	-	-	-	-	-	-	-	-	dec. or hex.
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	- if index is too high
R9008	%MX0.900.8	ON	- parameter s exceeds the valid range

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

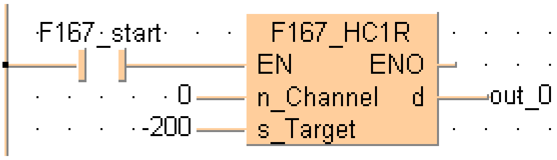
	Identifier	Address	Type	Initial	Comment
0	out_0	%QX0.0	BOOL	FALSE	output Y0 of PLC

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	F167_start	BOOL	FALSE	F167 start condition

Body When the variable **F167_start** is set to TRUE, the function is executed.

LD



Assign a number to the input variable (e.g. Monitor → Monitor Header, click the variable, enter the value, press <Enter>) or replace the input variables by numbers.

F171_SPDH

Pulse Output Instruction for Trapezoidal Control and Home Return with Channel Specification

Steps: 5

Description This instruction outputs pulses from the specified channel (CH0 or CH2) according to the specified parameters. You can use this instruction for:

- Trapezoidal control (see page 724)
- Home position return (see page 729)

- **When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".**
- **If you perform a rewrite during RUN when pulse output is taking place, more pulses than the setting may be output.**
- **The high-speed counter control flag also changes during scanning.**

PLC types: **Availability of F171_SPDH (see page 930)**

Data types

Variable	Data type	Function
s	DUT	Starting address of area containing the data table
n	decimal constant	Channel 0 or 2 for pulse output

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- n is a value other than 0 or 2 - the data for the control code, Fmin, Fmax (and the target value for trapezoidal control) are outside the specification range - Fmin > Fmax
R9008	%MX0.900.8	for an instant	

Precautions during Prog.

- When the control code (lower order) is 16#20 to 16#27, the home input is enabled after near home input regardless of whether deceleration has ended or is still in progress.
- When the control code (lower order) is 16#30 to 16#37, the home input is only enabled following near home input after deceleration to the initial speed has been completed.
- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration begins.
- If both the normal program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- If the specified value for the deviation counter clear signal is outside the specification range, it will be corrected to a value within the range.

Trapezoidal Control

Pulses are output from the specified channel (CH0 or CH2) when the corresponding control flag turns off and the internal relay turns on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	**Pulse/direction**
CH0	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
FP-Σ: CH2	Y3	CW	Pulse
FP-X: CH1	Y4	CCW	Direction

The control code, initial speed, maximum speed, acceleration/deceleration time, and target value are specified by creating a DUT (Data Unit Type) variable.

The frequency is changed using the specified acceleration/deceleration time from the initial speed to the maximum speed. During deceleration (normally 30 steps), the frequency is changed based on the same slope as during acceleration.

If the frequency is set to 50 kHz or more, specify a duty of 1/4 (25%).

Table of areas used

	Channel no.	Control flag	Elapsed value area	Target value area
FP-Sigma	**CH0**	R903A	DDT90044	DDT90046
	CH2	R903C	DDT90200	DDT90202
FP-X	**CH0**	R911C	DDT90348	DDT90350
	CH1	R911D	DDT90352	DDT90354

Operation modes

Incremental position control

Outputs the pulses set with the target value.

Selected mode Target value	CW/CCW	Pulse + direction Forward off Reverse on	Pulse + direction Forward on Reverse off	HSC counting method
Positive	Pulse output from CW	Pulse output when direction output is off	Pulse output when direction output is on	Incremental
Negative	Pulse output from CCW	Pulse output when direction output is on	Pulse output when direction output is off	Decremental

Absolute position control

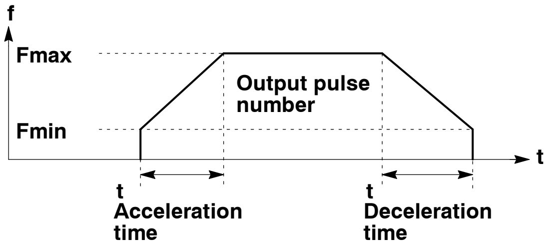
Outputs a number of pulses equal to the difference between the set target value and the current value.

Selected mode Target value	CW/CCW	Pulse + direction Forward off Reverse on	Pulse + direction Forward on Reverse off	HSC counting method
Target value greater than current value	Pulse output from CW	Pulse output when direction output is off	Pulse output when direction output is on	Incremental
Target value less than current value	Pulse output from CCW	Pulse output when direction output is on	Pulse output when direction output is off	Decremental

Precautions during programming

If both the regular program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

Data Unit Type settings



	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest WORD fixed to 0000 Lowest WORD = Control Code
1	Fmin	DINT	0	Initial frequency (Hz)
2	Fmax	DINT	0	Target frequency (Hz)
3	AccelTime	DINT	0	Time between Fmin and Fmax in ms
4	TargetValue	DINT	0	Target value, number of pulses
5	Termination	DINT	0	End of data table

DUT element 0: Control code (specify with a hex. constant)

16# □□□□□□□□

0: Fixed

Number of acceleration/deceleration steps

0: 30 steps
1: 60 steps (can only be specified for ve r. 2.0 or higher)

Duty (ON width)

0: Duty 1/2 (50%)
1: Duty 1/4 (25%)

Frequency rang e

0: 1.5 Hz to 9.8 kHz
1: 48 Hz to 100 kHz
2: 191 Hz to 100 kHz

Position control mode and output metho d

00: Incremental CW/CCW
02: Incremental pulse and direction (forward off/reverse on)
03: Incremental pulse and direction (forward on/reverse off)
10: Absolute CW/CCW
12: Absolute pulse and direction (forward off/reverse on)
13: Absolute pulse and direction (forward on/reverse off)

DUT element 1, 2: Frequency (Hz)

1.5 Hz to 9.8 kHz [1 to 9800 (units: Hz)]
(Maximum error near 9.8 kHz approximately -0.9 kHz)

48 Hz to 100 kHz [48 to 100000 (units: Hz)]
(Maximum error near 100 kHz approximately -3 kHz)

191 Hz to 100 kHz [191 to 100000 (units: Hz)]
(Maximum error near 100 kHz approximately -0.8 kHz)
The minimum frequency is 1.5 Hz. Therefore setting 1 will specify 1.5 Hz.

Specify the initial frequency to 30 kHz or less.

DUT element 3: Acceleration/deceleration time (ms)

With 30 steps: K30 to K32767

With 60 steps: K36 to K32767

DUT element 4: Target value

-2147483648 to 2147483647 (16#80000000 to 16#7FFFFFFF)

Example In this example the function F171_SPDH is programmed in ladder diagram (LD).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

DUT A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

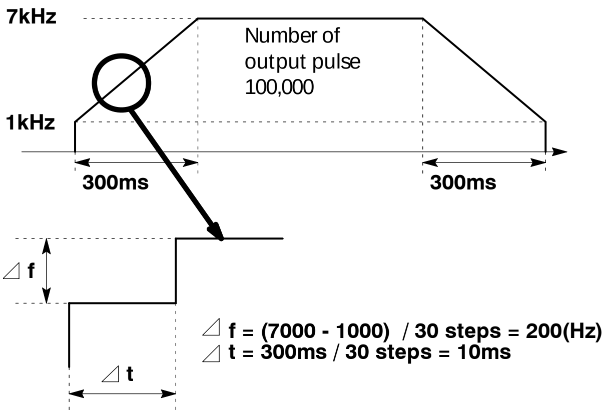
MotorData [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest WORD fixed to 0000 Lowest WORD = Control Code
1	Fmin	DINT	0	Initial frequency (Hz)
2	Fmax	DINT	0	Target frequency (Hz)
3	AccelTime	DINT	0	Time between Fmin and Fmax in ms
4	TargetValue	DINT	0	Target value, number of pulses
5	Termination	DINT	0	End of data table

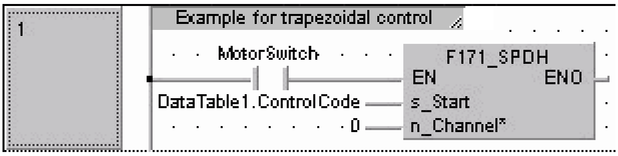
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTE	MotorSwitch	BOOL	FALSE	
1	VAR	DataTable1	MotorDat	ControlCode := 16#110C Fmin := 1000, Fmax := 7000, AccelTime := 300, TargetValue := 100000	For ControlCode (16#1100): Lowest WORD: 1 = Duty 25% 1 = Frequency 48 to 100 kHz 00 = Incremental CW/CCW

Body The parameters defined in the DUT will be executed as illustrated below.


7kHz
1kHz
Number of
output pulse
100,000
300ms
300ms
⊿ f
⊿ t
⊿ f = (7000 - 1000) / 30 steps = 200(Hz)
⊿ t = 300ms / 30 steps = 10ms

LD


1
Example for trapezoidal control
MotorSwitch
F171_SPDH
EN
ENO
DataTable1.ControlCode
s_Start
0
n_Channel*

Home Position Return

Precautions During Programming (see page 724)

Pulses are output from the specified channel (CH0 or CH2) when the corresponding control flag turns off and the internal relay turns on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	**Pulse/direction**
CH0	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
	Y2	Deviation counter clear	
FP-Σ: CH2	Y3	CW	Pulse
FP-X: CH1	Y4	CCW	Direction
	Y5	Deviation counter clear	

The control code, initial speed, maximum speed, acceleration/deceleration time, and deviation counter reset output time are specified by creating a DUT (Data Unit Type) variable.

The frequency is changed using the specified acceleration/deceleration time from the initial speed to the maximum speed. During deceleration (normally 30 steps), the frequency is changed based on the same slope as during acceleration.

If the frequency is set to 50 kHz or more, specify a duty of 1/4 (25%).

Table of areas used

	Channel no.	Control flag	Elapsed value area	Target value area	Near home	Home input
FP-Sigma	CH0	R903A	DT90044, 90045	DT90046, 90047	DT90052 bit4	X2
	CH2	R903C	DT90200, 90201	DT90202, 90203	DT90052 bit4	X5
FP-X	CH0	R911C	DT90348, DT90349	DT90350, DT90351	DT90052 bit4	X2
	CH1	R911D	DT90352, DT90353	DT90354, DT90355	DT90052 bit4	X5

Operation modes

Pulses are output continually until home input (X2 or X5) occurs. To decelerate at near home, set the bit4 of special data register DT90052 to off → on → off when near home input occurs. The value in the elapsed value area during a home position return differs from the current value. When the return is completed, the elapsed value changes to 0.

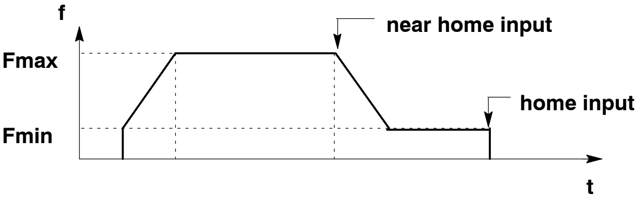
There are two different operating modes:

- Type I Home Return

  The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.

- Type II Home Return

  In this mode, the home input is effective only after deceleration (started by near home input) has been completed.



Example In this example the function F171_SPDH is programmed in ladder diagram (LD).

GVL In the Global Variable List you define variables that can be accessed by all POUs in the project.

Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

DUT A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

MotorDataHome [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest WORD fixed to 0000 Lowest WORD = Control Code
1	Fmin	DINT	0	Initial frequency (Hz)
2	Fmax	DINT	0	Target frequency (Hz)
3	AccelTime	DINT	0	Time between Fmin and Fmax in ms
4	DeviationSignal	DINT	0	Deviation counter clear signal in ms

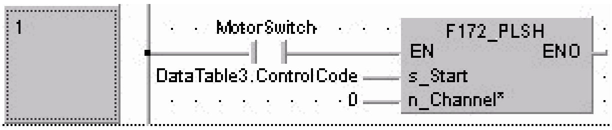
POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EX	MotorSwitch	BOOL	FALSE	
1	VAR	DataTable2	MotorDataHom	ControlCode := 16#1121 Fmin := 1000, Fmax := 7000, AccelTime := 100	For ControlCode (16#1121) 1 = 25% duty 1 = 48 to 100 kHz 21 = Home position return type I CW

Body

The parameters defined in the DUT will be executed as illustrated below.



F172_PLSH

Pulse output instruction with channel specification (JOG operation)

Steps: 5

Description Pulses are output from the specified channel (CH0 or CH2) when the corresponding control flag is off and the execution condition is on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	Pulse/direction
CH0	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
FP-Σ: CH2 FP-X: CH1	Y3	CW	Pulse
	Y4	CCW	Direction

By specifying either incremental counting or decremental counting in the control code, this instruction can be used as an instruction for JOG operations.

The frequency can be changed each time a scan is carried out. It cannot be changed, however, when the control code is in the midst of executing an instruction.

If the frequency is set to 50 kHz or more, specify a duty of 1/4 (25%).

Table of areas used

	Channel no.	Control flag	Elapsed value area	Target value area
FP-Sigma	CH0	R903A	DDT90044	DDT90046
	CH2	R903C	DDT90200	DDT90202
FP-X	CH0	R911C	DDT90348	DDT90350
	CH1	R911D	DDT90352	DDT90354

◆ **NOTES**

- When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".
- If a rewrite is executed during RUN while the system is operating, pulse output stops while the program is being rewritten.
- If the same notation is being used for both the ordinary program and the interrupt program, make sure they are not both executed at the same time.
- The high-speed counter control flag can be changed while a scan is in progress.

- If a value outside the specified range is written for the frequency area while the instruction is being executed, the frequency that is output will be adjusted to either minimum or maximum. An operation error occurs when execution of the instruction starts.

DUT Settings

MotorDataJog [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest WORD fixed to 0000 Lowest WORD = Control Code
1	Frequency	DINT	0	Frequency (Hz)

DUT element 0: Control code (specify with a hex constant)

16# □□□□□□□□

0: Fixed

Target value settin g
0: Mode with no target value
1: Target value match stop mode
(can only be specified for ve r. 2.0 or higher)

Duty (ON width)
0: Duty 1/2 (50%)
1: Duty 1/4 (25%)

Frequency rang e
0: 1.5 Hz to 9.8 kHz
1: 48 Hz to 100 kHz
2: 191 Hz to 100 kHz

Output method

00:	No counting	CW
01:	No counting	CCW
10:	Incremental counting	CW
12:	Incremental counting	Direction output off
13:	Incremental counting	Direction output on
21:	Decremental counting	CCW
22:	Decremental counting	Direction output off
23:	Decremental counting	Direction output on

DUT element 1: Frequency (Hz)

1.5 Hz to 9.8 kHz [1 to 9800 (units: Hz)]
(Maximum error near 9.8 kHz approximately -0.9 kHz)

48 Hz to 100 kHz [48 to 100000 (units: Hz)]
(Maximum error near 100 kHz approximately -3 kHz)

191 Hz to 100 kHz [191 to 100000 (units: Hz)]
(Maximum error near 100 kHz approximately -0.8 kHz)
The minimum frequency is 1.5 Hz. Therefore setting 1 will specify 1.5 Hz.

DUT element 2: Target value (absolute value) - only V2.0 or higher

Designate the target value setting in the range indicated below. If an out of range value is designated, the number of pulses output will be different than the designated value. The target value setting is ignored in the no count mode (0 = "Mode with no target value").

Output method	**Range of target values which can be designated**
Incremental counting	Designate a value larger than the current value.
Decremental counting	Designate a value smaller than the current value.

PLC types: **Availability of F172_PLSH (see page 930)**

Data types

Variable	Data type	Function
s	DUT	Starting address of area containing the data table
n*	decimal constant	Channel 0 or 2 for pulse output

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- n is a value other than 0 or 2 - the data for the control code or frequency are outside the specification range
R9008	%MX0.900.8	for an instant	

Example In this example the function F172_PLSH is programmed in ladder diagram (LD).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

DUT A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

MotorDataJog [DUT]

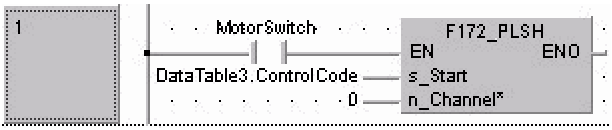
	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest WORD fixed to 0000 Lowest WORD = Control Code
1	Frequency	DINT	0	Frequency (Hz)

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTEI	MotorSwitch	BOOL	FALSE	
1	VAR	DataTable3	MotorDataJog	ControlCode := 16#1121 Frequency := 60000	ControlCode 16#1121= 1 = 25% duty 1= 48 to 100 kHz 21 = decremental counting CCW Frequency = 60 kHz

LD



F173_PWMH

Pulse output instruction with channel specification (PWM output)

Steps: 5

Description When the corresponding control flag is off and the execution condition is in the on state, a PWM pulse is output from the specified channel (CH0 or CH2) is obtained. The pulses are output while the execution condition is on.

The data table shown below, indicating the frequency and duty, is created and the values are specified by the user program.

The duty, particularly when it is close to the minimum or maximum value, may be off from the specified ratio, depending on the load voltage and load current.

The duty can be changed for each separate scan. Control codes, however, cannot be changed while an instruction is being executed.

Table of areas used

	Channel no.	Output	Output method
FP-Sigma	**Ch0**	Y0	R903A
	Ch2	Y3	R903C
FP-X	**Ch0**	Y0	R911C
	Ch1	Y3	R911D

◆ **NOTES**

- When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".
- If a rewrite is executed during RUN while the system is operating, pulse output stops while the program is being rewritten.
- If the same notation is being used for both the ordinary program and the interrupt program, make sure they are not both executed at the same time.
- The high-speed counter control flag can be changed while a scan is in progress.
- If a value outside the specified range is written for the frequency area while the instruction is being executed, the frequency that is output will be adjusted to either minimum or maximum. An operation error occurs when execution of the instruction starts.

■ Data table settings

Offset 0	Control code	(ARRAY of INT [0])
Offset 1	Duty	(ARRAY of INT [1])

Offset 0: control code

Resolution of 1000

Setting	Frequency (Hz)	Timing (ms)
0	1.5	666.7
1	2.0	502.5
2	4.1	245.7
3	6.1	163.9
4	8.1	122.9
5	9.8	102.4
6	19.5	51.2
7	48.8	20.5
8	97.7	10.2
9	201.6	5.0
10	403.2	2.5
11	500.0	2.0
12	694.4	1.4
13	1.0 k	1.0
14	1.3 k	0.8
15	1.6 k	0.6
16	2.1 k	0.5
17	3.1 k	0.3
18	6.3 k	0.2
19	12.5 k	0.1

Resolution of 100

Setting	Frequency (Hz)	Timing (ms)
20	15.6 k	0.06
21	20.8 k	0.05
22	25.0 k	0.04
23	31.3 k	0.03
24	41.7 k	0.02

Offset 1: duty

If the control code is 0 to 19, the duty is 0 to 999 (0.0% to 99.9%).
If the control code is 20 to 24, the duty is 0 to 990 (0% to 99%).
Values are specified in units of 1% (10). Digits below the decimal point are rounded off.

PLC types: **Availability of F173_PWMH (see page 930)**

Data types

Variable	Data type	Function
s	ARRAY **[0..1]** of INT or WORD	Contains settings for the control code and duty
n	decimal constant	Channel 0 or 2 that corresponds to the pulse

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DT	-	-	-
n	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- n is a value other than 0 or 2 - the value set for frequency is outside the specified range - a value higher than 100% is specified for the duty
R9008	%MX0.900.8	for an instant	

Example In this example the function F173_PWMH is programmed in ladder diagram (LD).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

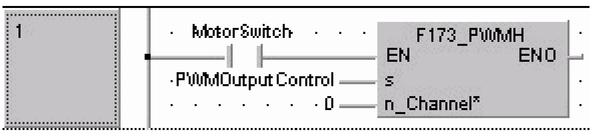
Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTE	MotorSwitch	BOOL	FALSE	
1	VAR	PWMOutputContr	ARRAY [0..1] OF INT	[11,900]	11 = 500.0 Hz, 2.0 ms 900 = 90.0% duty

Body



F174_SP0H

Pulse output instruction, table control with channel specification

Steps: 5

Description This instruction outputs pulses from the specified channel (0 or 2) according to the specified parameters.

The pulse output control mode is selected by settings in the contents of the 32-bit areas specified by **s**, which includes:

- the control code
- frequencies
- target values

The pulse output operation starts at the first frequency specified by the contents of offset 2 when the trigger turns on.

When the elapsed value of the high-speed counter agrees with the target value, specified by the contents of offset 4, the output pulse frequency changes from the initial output pulse frequency to the next output pulse frequency.

Then the PLC executes the next (nth) pulse output specified by the nth frequency of offset 2+(offset n-1)*4 and the nth target value at offset 4+(offset n-1)*4, and so on.

When the frequency 0 is specified, this is regarded as the final frequency and the pulse output operation stops.

NOTES

- **When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".**
- **If you perform a rewrite during RUN when pulse output is taking place, more pulses than the setting may be output.**
- **The high-speed counter control flag also changes during scanning.**

PLC types: **Availability of F174_SP0H (see page 930)**

Data types

Variable	Data type	Function
s	DUT	Starting address of area containing the data table
n	decimal constant	Channel 0 or 2 for pulse output

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- n is a value other than 0 or 2 - the data for the control code or frequency 1 are outside the setting range
R9008	%MX0.900.8	for an instant	

Table shaped control

There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	Pulse/direction
CH0	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
FP-Σ: CH2 FP-X: CH1	Y3	CW	Pulse
	Y4	CCW	Direction

The control code, frequencies, and target values are specified by creating a DUT (Data Unit Type) variable.

If the frequency is set to 50 kHz or more, specify a duty of 1/4 (25%).

Table of areas used

	Channel no.	Control flag	Elapsed value area	Target value area
FP-Sigma	CH0	R903A	DDT90044	DDT90046
	CH2	R903C	DDT90200	DDT90202
FP-X	CH0	R911C	DDT90348	DDT90350
	CH1	R911D	DDT90352	DDT90354

■ Operation modes

Incremental position control

Outputs the pulses set with the target value.

Mode			0	1	2	3	4	5	Counter
Target	**CH0**	**CH2**	CW	CCW	Pulse + direction				
					forward OFF	reverse ON	forward ON	reverse OFF	
Positive	Y0 Y1	Y3 Y4	pulses OFF	OFF pulses	pulses OFF	pulses ON	pulses ON	pulses OFF	Incremental
Negative	Y0 Y1	Y3 Y4	OFF pulses	pulses OFF	pulses ON	pulses OFF	pulses OFF	pulses ON	Decremental

Absolute position control

Outputs a number of pulses equal to the difference between the set target value and the current value.

Mode			0	1	2	3	4	5	Counter
Target value	**CH0**	**CH2**	CW	CCW	Pulse + direction				
					forward OFF	reverse ON	forward ON	reverse OFF	
> current value	Y0 Y1	Y3 Y4	pulses OFF	OFF pulses	pulses OFF	pulses ON	pulses ON	pulses OFF	Incremental
< current value	Y0 Y1	Y3 Y4	OFF pulses	pulses OFF	pulses ON	pulses OFF	pulses OFF	pulses ON	Decremental

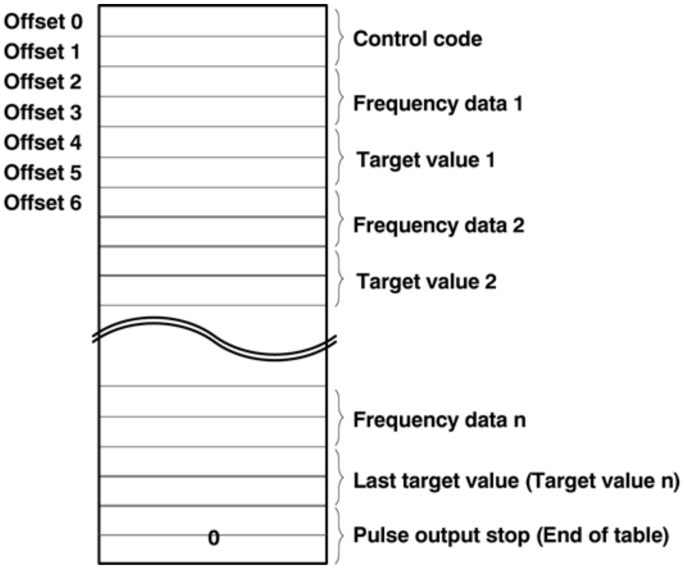
Precautions during programming

- The high-speed counter control flag R903A (R903C) is on from the time that the execution condition for the F174_SP0H instruction has gone on until the pulse output stops.
- During the time that the high-speed counter control flag R903A (R903C) is on, the high-speed counter and pulse output instructions F166 to F174 which use the same control flag, cannot be executed.
- An operation error occurs if a value that is not within the allowable range is specified for the control code or for frequency 1. (If the data for frequency 1 is 0, the operation is terminated without anything being executed.)
- Pulse output is stopped if the frequency of the second or a subsequent stage is specified as 0 or as a value outside the allowable range.
- If the table pointer exceeds the data register DT area during pulse output, pulse output control stops and the high-speed counter control flag R903A (R903C) goes off.
- When the F174_SP0H instruction is executed, the channel CH0 target value areas (DT90046 and DT90047) and the CH2 target value areas (DT90202 and DT90203) are not used.
- Always make sure that the target values are specified within the ranges indicated on the following page. If a value outside the allowable range is specified, the number of pulses output will be different from the specified value.

- If a periodic interrupt or high-speed counter value interrupt program is run, or the PLC link function is used at the same time, a frequency of 80 kHz or less should be used.

Data Unit Type settings

Overview of the settings for the data table



F174_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest word fixed to 0000 Lowest word = control code
1	Frequency1	DINT	0	
2	TargetValue1	DINT	0	
3	Frequency2	DINT	0	
4	TargetValue2	DINT	0	
5	Frequency3	DINT	0	
6	TargetValue3	DINT	0	
7	Termination	DINT	0	End of data table

DUT element 0: Control code (specify with a hex. constant)

16# □□□ □□□ □□

Upper word
0: Fixed

Duty (ON width)
0: Duty 1/2 (50%)
1: Duty 1/4 (25%)

Frequency range
0: 1.5 Hz to 9.8 kHz
1: 48 Hz to 100 kHz
2: 191 Hz to 100 kHz

control mode

0: Incremental	Specifies the amount of travel (number of pulses)
1: Absolute	Specifies the target value (absolute value)

Output method

0: CW	(incremental counting)
1: CCW	(decremental counting)
2: Pulse and direction (forward of	(incremental counting)
3: Pulse and direction (reverse on)	(decremental counting)
4: Pulse and direction (forward on)	(incremental counting)
5: Pulse and direction (reverse of	(decremental counting)

DUT element 1, 3, 5, 7, etc.: Frequency (Hz)

1.5 Hz to 9.8 kHz [1 to 9800 (units: Hz)]
The minimum frequency is 1.5 Hz. Therefore setting 1 will specify 1.5 Hz.

48 Hz to 80 kHz [48 to 80000 (units: Hz)]
(Maximum error near 80 kHz approximately -2 kHz)

191 Hz to 80 kHz [191 to 100000 (units: Hz)]
(Maximum error near 80 kHz approximately -0.5 kHz)

DUT element 2, 4, 6, 8, etc.: Target value

-2147483648 to 2147483647 (16#80000000 to 16#7FFFFFFF)

Incremental:	Specify move value (pulse): positive value when counter increases negative value when counter decreases
Absolute:	Specify target value

Final DUT element: End of Table

Set 0 at the final address of the DUT to stop pulse output.

Example In this example the function F174_SP0H is programmed in ladder diagram (LD).

GVL In the Global Variable List, you define variables that can be accessed by all POUs in the project.

Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

DUT A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

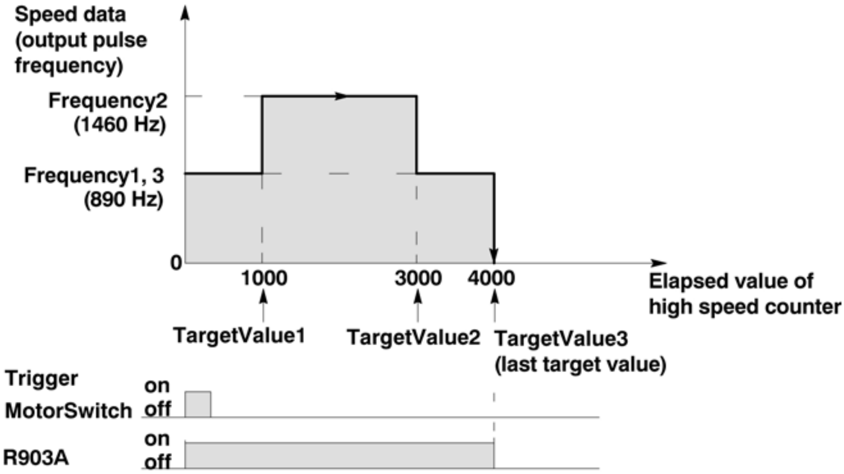
F174_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest word fixed to 0000 Lowest word = control code
1	Frequency1	DINT	0	
2	TargetValue1	DINT	0	
3	Frequency2	DINT	0	
4	TargetValue2	DINT	0	
5	Frequency3	DINT	0	
6	TargetValue3	DINT	0	
7	Termination	DINT	0	End of data table

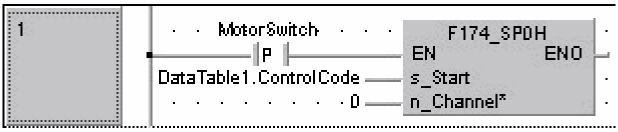
POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	MotorSwitch	BOOL	FALSE	
1	VAR	DataTable1	F174_DUT	ControlCode := 16#1200, Frequency1 := 800, TargetValue1 := 1000, Frequency2 := 1460, TargetValue2:= 3000, Frequency3 := 800 TargetValue3 := 4000	Control Code: 1=Duty 25% 2=Range 101Hz to 80kHz 0=Incremental 0=CW(counter increases)

Body The parameters defined in the DUT will be executed as illustrated in the following time chart:


Speed data
(output pulse
frequency)
Frequency2
(1460 Hz)
Frequency1, 3
(890 Hz)
0
1000
3000
4000
Elapsed value of
high speed counter
TargetValue1
TargetValue2
TargetValue3
(last target value)
Trigger
MotorSwitch
on
off
R903A
on
off

LD


1
MotorSwitch
P
F174_SP0H
EN
ENO
DataTable1.ControlCode
s_Start
0
n_Channel*

F175_SPSH_LINEAR Pulse output (Linear interpolation)

Steps: 5

Description

Precautions during programming (see page 747)

Pulses are output from channel CH0 and CH2, in accordance with the parameters specified in the DUT PULSE_LINEAR, so that the path to the target position forms a straight line. The DUT is predefined in the FP Library.

Pulses are output from channel CH0 (X-axis) and CH2 (Y-axis) when the corresponding control flag is off and the execution conditions are on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	**Pulse/direction**
CH0 (for X-axis)	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
FP-Σ: CH2 **FP-X: CH1** **(for Y-axis)**	Y3	CW	Pulse
	Y4	CCW	Direction

- **When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".**
- **If you perform a rewrite during RUN when pulse output is taking place, more pulses than the setting may be output.**

PLC types: **Availability of F175_SPSH_LINEAR (see page 930)**

Data types

Variable	Data type	Function
s	PULSE_LINEAR (DUT)	Contains all data for the instruction to be executed.
n*	Constant	Must always be zero.

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n*	-	-	-	-	-	-	-	-	-	dec. or hex

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- "n" is anything other than 0. - the DUT data is outside the specification range. - the initial speed 'InitialSpeed' (offset 2, 3) > maximum speed 'MaximumSpeed' (offset 4, 5)
R9008	%MX0.900.8	for an instant	- the maximum speed 'MaximumSpeed' (offset 4, 5) > 100kHz

Table of areas used

	Channel no.	Control flag	Elapsed value area	Target value area
FP-Sigma	**CH0**	R903A	DDT90044	DDT90046
	CH2	R903C	DDT90200	DDT90202
FP-X	**CH0**	R911C	DDT90348	DDT90350
	CH1	R911D	DDT90352	DDT90354

26.1.3 Precautions during programming

CAUTION

- The execution conditions for this instruction must be set permanently. When the execution conditions are off, pulse output stops.
- Designate settings for the target value or movement distance so they are within the following range:
  -8,388,608 to +8,388,607
- When using in combination with other positioning instructions like F171_SPDH (see page 723), the target value in these instructions must also be within the above range.
- With this instruction, the component speed is calculated once per scan, and movement is done in an arc shape while performing correction. As the execution conditions must always be set, use in combination with a constant scan or periodical interrupt program.
- The constant scan time or the periodical interrupt should be 10 to 20 times the cycle of the specified frequency.

◆ **EXAMPLE**

If you specify a frequency of 5 kHz (0.2ms cycle), then the set time should be 2ms to 4ms.

When the scan time is shorter than ten times the cycle, we recommend using the constant scan function. When it is longer, we recommend using the periodical interrupt function.

If both the regular program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

If you make the current position equal the target value when specifying the center position setting method, a circle drawing operation will result.

When using in application requiring precision, check the actual machine.

Example In this example the function F175_SPSH_LINEAR is programmed in ladder diagram (LD).

DUT The following DUT PULSE_LINEAR is predefined in the library "System Lib".

PULSE_LINEAR [DUT]

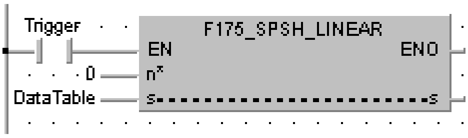
	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	IN: Control code
1	InitialSpeed	DINT	0	IN: Composite speed initial Fmin (Hz) [1.5Hz to 100kHz]
2	MaximumSpeed	DINT	0	IN: Composite speed maximum Fmax (Hz) [1.5Hz to 100kHz]
3	AccelDecelTime	DINT	0	IN: Acceleration/deceleration time T (ms) [0ms to 32767ms]
4	TargetValue_X	DINT	0	IN: X-axis (CH0) target value (movement distance) [-8388608 to 8388607]
5	TargetValue_Y	DINT	0	IN: Y-axis (CH2) target value (movement distance) [-8388608 to 8388607]
6	InitialSpeed_X	REAL	0	OUT: X-axis (CH0) component speed initial Fmin (Hz)
7	MaximumSpeed_X	REAL	0	OUT: X-axis (CH0) component speed maximum Fmax (Hz)
8	InitialSpeed_Y	REAL	0	OUT: Y-axis (CH2) component speed initial Fmin (Hz)
9	MaximumSpeed_Y	REAL	0	OUT: Y-axis (CH2) component speed maximum Fmax (Hz)
10	FreqRange_X	INT	0	OUT: X-axis (CH0) frequency range [0,1,2]
11	FreqRange_Y	INT	0	OUT: Y-axis (CH2) frequency range [0,1,2]
12	AccelDecelStepsX	INT	0	OUT: X-axis (CH0) acceleration/deceleration steps [0 to 60]
13	AccelDecelStepsY	INT	0	OUT: Y-axis (CH2) acceleration/deceleration steps [0 to 60]

IN: Control code
Bit 12 Duty: FALSE=25%, TRUE=50%
Bit 4: FALSE=Incremental, TRUE=Absolut
Bits 0,1: 0=CW/CCW, 2=PLS+SIGN(forward OFF), 3=PLS+SIGN(forward ON)

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Trigger	BOOL	FALSE	
1	VAR	DataTable	PULSE_LINEAR	ControlCode := 16#1121, InitialSpeed := 0, MaximumSpeed := 80000, AccelDecelTime := 30000, TargetValue_X := 8000000, TargetValue_Y := 5000000	

LD



F176_SPCH_CENTER Pulse output (Arc interpolation)

Steps: 5

Description Precautions during programming (see page 747)

Pulses are output from channel CH0 and CH2, in accordance with the parameters specified in the DUT PULSE_ARC_CENTER (see page 751), so that the path to the target position forms an arc. Both DUTs are predefined in the library "System Lib".

Pulses are output from the channel CH0 (X-axis) and CH2 (Y-axis) when the corresponding control flag turns off and the execution condition (trigger) turns on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	**Pulse/direction**
CH0 (for X-axis)	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
CH2 (for Y-axis)	Y3	CW	Pulse
	Y4	CCW	Direction

☞

- **When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".**
- **If you perform a rewrite during RUN when pulse output is taking place, more pulses than the setting may be output.**
- **Bit number 8 of the control code specifies the method used. Using PULSE_ARC_CENTER this bit is set automatically by the compiler.**

PLC types: **Availability of F176_SPCH_CENTER (see page 930)**

Data types

Variable	Data type	Function
s	DUT PULSE_ARC_CENTER (see page 751)	Contains all data for the instruction to be executed.
n*	Constant	Must always be "0".

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- "n" is anything other than 0. - the DUT data is outside the specification range. - incremental mode is designated and the value of "current value + movement distance" is outside the range -8388608 to +8388607. - absolute mode is designated and the target value is outside the range -8388608 to +8388607. - Center position O = Target position E - Center position O = Current position S
R9008	%MX0.900.8	for an instant	

Table of areas used

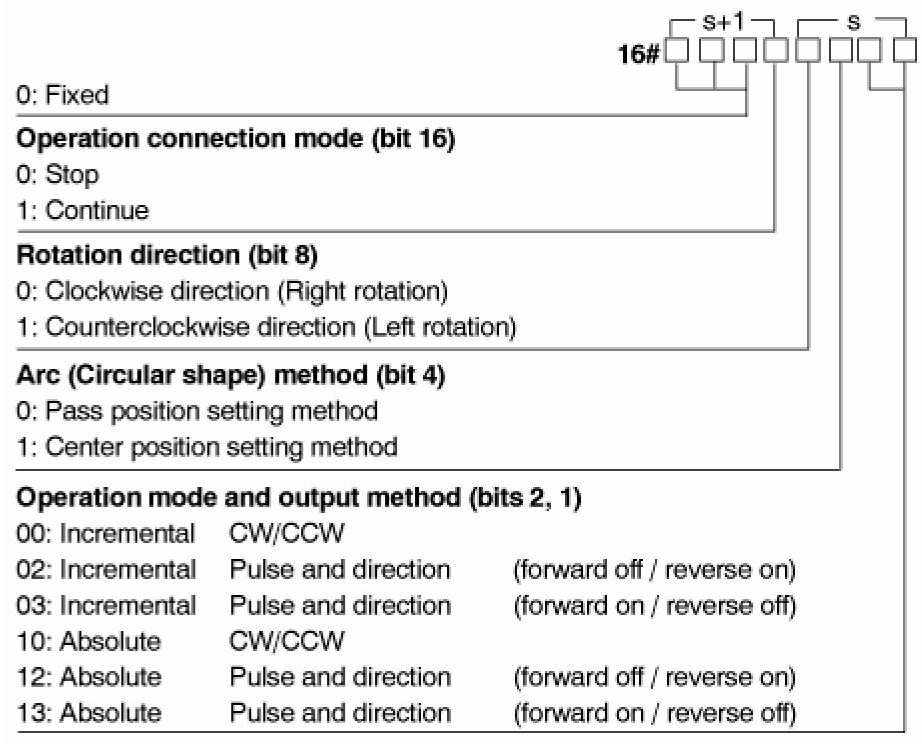
Channel no.	Control flag	Elapsed value area	Target value area
CH0	R903A	DT90044, DT90045	DT90046, DT90047
CH2	R903C	DT90200, DT90201	DT90202, DT90203

DUT PULSE_ARC_CENTER

This DUT specifies the control code, composite speed, target position and center position.

Offset	Name of DUT element	Meaning	Units	Range
Setting area				
0 1	ControlCode	Control code (see page 752)	Hz	
2 3	Speed	Composite speed (Frequency) Fv	Hz	100 Hz to 20 kHz [100 to 20000]
4 5	TargetPos_X	X-axis (CH0) Target position	pulses	-8388608 to 8388607
6 7	TargetPos_Y	Y-axis (CH2) Target position	pulses	-8388608 to 8388607
8 9	CenterPos_X	X-axis (CH0) Center position	pulses	-8388608 to 8388607
10 11	CenterPos_Y	Y-axis (CH2) Center position	pulses	-8388608 to 8388607
Operation result storage area				
12 13	Radius	Radius	pulses	

■ Control code of PULSE_ARC_CENTER (PASS)



■ Control code explanations

Bit 16: Operation connection mode

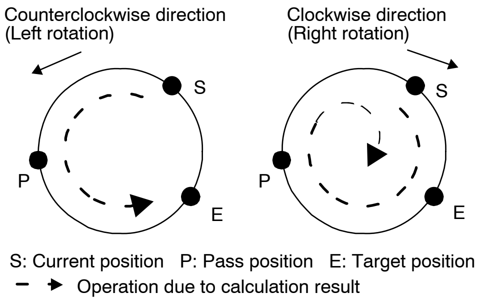
Stop:
When stop (0) is specified, pulse output will stop when the target position is reached.

Continue:

When continue (1) is specified after arc interpolation action begins, the arc interpolation data table is overwritten. The following arc interpolation begins when the first arc interpolation that was started up finishes (target position reached). To finish, specify stop (0) for this flag (operation connection mode) after the last arc interpolation action has started.

Bit 8: Rotation direction

Pulses are output according to the designated direction. Operation differs, as indicated below, depending on the pass position and rotation direction setting.

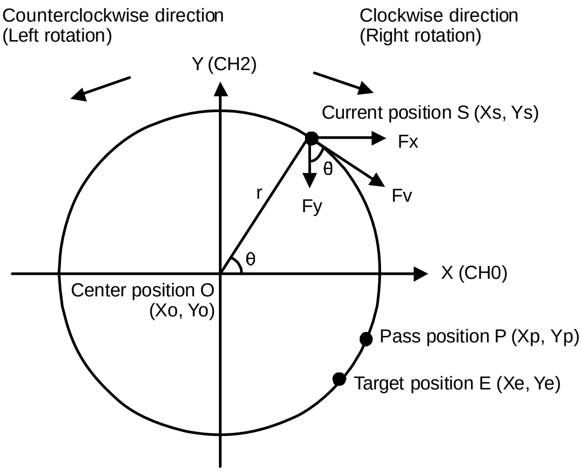


Bit 4: Arc (Circular shape) method

Pass position setting method:

The center position and the radius of the arc are calculated by specifying the pass and target positions for the current position.

Center position setting method:
The radius of the arc is calculated by specifying the center and target positions for the current position.



Let CH0 be the X–axis, and CH2 be the Y–axis.

Fv:	Composite speed	O (Xo, Yo):	Center point (Center position)
Fx:	X–axis component speed	S (Xs, Ys):	Start point (Current position)
Fy:	Y–axis component speed	P (Xp, Yp):	Pass point (Pass position)
r:	Radius	E (Xe, Ye):	End point (Target position)

$$Fx = Fv \sin\theta = Fv \frac{|Ye - Yo|}{r} \qquad Fy = Fv \cos\theta = Fv \frac{|Xe - Xo|}{r}$$

Example In this example the function F176_SPCH_CENTER is programmed in ladder diagram (LD).

DUT The following DUT PULSE_ARC_CENTER is predefined in the library "System Lib".

PULSE_ARC_CENTER [DUT]

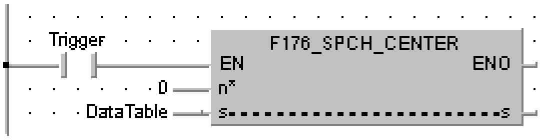
	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	16#00000100	IN: Control mode
1	Speed	DINT	0	IN: Composite speed initial Fmin (Hz) [100Hz to 20kHz]
2	TargetPos_X	DINT	0	IN: X-axis (CH0) target position [-8388608 to 8388607]
3	TargetPos_Y	DINT	0	IN: Y-axis (CH2) target position [-8388608 to 8388607]
4	CenterPos_X	DINT	0	IN: X-axis (CH0) center position [-8388608 to 8388607]
5	CenterPos_Y	DINT	0	IN: Y-axis (CH2) center position [-8388608 to 8388607]
6	Radius	DINT	0	OUT: Radius

IN: Control mode
Bit 16: FALSE=Stop, TRUE=Continue
Bit 12: FALSE=CW (right), TRUE=CCW (left)
Bit 8: FALSE=Pass, TRUE=Center
Bit 4: FALSE=Incremental, TRUE=absolut
Bits 0,1: 0=CW/CCW, 2=PLS+SIGN(forward OFF), 3=PLS+SIGN(forward ON)

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Trigger	BOOL	FALSE	
1	VAR	DataTable	PULSE_ARC_CENTER	ControlCode := 16#1121, Speed := 15000, TargetPos_X := 8000000, TargetPos_Y := 5000000, CenterPos_X := 8300000, CenterPos_Y := 5500000	

LD



F176_SPCH_PASS Pulse output (Arc interpolation)

Steps:

Description Precautions during programming (see page 747)

Pulses are output from channel CH0 and CH2, in accordance with the parameters specified in the DUT PULSE_ARC_PASS (see page 756), so that the path to the target position forms an arc. Both DUTs are predefined in the library "System Lib".

Pulses are output from the channel CH0 (X-axis) and CH2 (Y-axis) when the corresponding control flag turns off and the execution condition (trigger) turns on. There are two different output methods to control positioning: CW/CCW and pulse/direction. CW/CCW uses one pulse output to specify a forward rotation and one pulse output to specify a reverse rotation. Pulse/direction uses one pulse output to specify the speed and one pulse output to specify the direction of rotation with on/off signals. Use the control code to set the pulse output method.

Channel no.	Output	Output method	
		CW/CCW	**Pulse/direction**
CH0 (for X-axis)	Y0	CW (clockwise)	Pulse
	Y1	CCW (counter-clockwise)	Direction
CH2 (for Y-axis)	Y3	CW	Pulse
	Y4	CCW	Direction

☞

- **When using this instruction, set the HSC channels in system registers 400 and 401 to "Unused".**
- **If you perform a rewrite during RUN when pulse output is taking place, more pulses than the setting may be output.**
- **Bit number 8 of the control code specifies the method used. Using PULSE_ARC_PASS (see page 756) this bit is reset automatically by the compiler.**

PLC types: **Availability of F176_SPCH_PASS (see page 930)**

Data types

Variable	Data type	Function
s	DUT PULSE_ARC_PASS (see page 756)	Contains all data for the instruction to be executed.
n*	Constant	Must always be "0".

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	DDT	-	-	-
n*	-	-	-	-	-	-	-	-	-	dec. or hex

Error flags

<table>
<tr><th>No.</th><th>IEC address</th><th>Set</th><th>If</th></tr>
<tr><td>R9007</td><td>%MX0.900.7</td><td>permanently</td><td rowspan="2">- "n" is anything other than 0.
- the DUT data is outside the specification range.
- incremental mode is designated and the value of "current value + movement distance" is outside the range -8388608 to +8388607.
- absolute mode is designated and the target value is outside the range -8388608 to +8388607.
- Current position S ≈ Target position E
- Current position S ≈ Pass position P
- Pass position P ≈ Target position E
- Current position S, Pass position P and Target position E approximate a straight line.</td></tr>
<tr><td>R9008</td><td>%MX0.900.8</td><td>for an instant</td></tr>
</table>

DUT PULSE_ARC_PASS

The DUT PULSE_ARC_PASS (see page 756) specifies the control code, composite speed, target position and pass position.

Setting area				
Offset	**Name of DUT element**	**Meaning**	**Units**	**Range**
0 **1**	ControlCode	Control code (see page 752)	Hz	
2 **3**	Speed	Composite speed (Frequency) Fv	Hz	100 Hz to 20 kHz [100 to 20000]
4 **5**	TargetPos_X	X-axis (CH0) Target position	pulses	-8388608 to 8388607
6 **7**	TargetPos_Y	Y-axis (CH2) Target position	pulses	-8388608 to 8388607
8 **9**	PassPos_X	X-axis (CH0) Pass position	pulses	-8388608 to 8388607
10 **11**	PassPos_Y	Y-axis (CH2) Pass position	pulses	-8388608 to 8388607
Operation result storage area				
12 **13**	Radius	Radius	pulses	
14 **15**	CenterPos_X	X-axis (CH0) Center position	pulses	
16 **17**	CenterPos_Y	Y-axis (CH2) Center position	pulses	

For more detailed information, please refer to the control code explanations (see page 752)

Example In this example the function F176_SPCH_PASS is programmed in ladder diagram (LD).

DUT The following DUT PULSE_ARC_PASS is predefined in the library "System Lib".

PULSE_ARC_PASS [DUT]

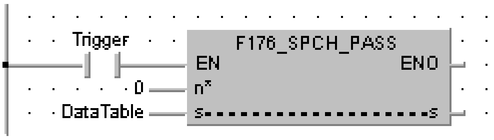
	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	16#00000000	IN: Control code
1	Speed	DINT	0	IN: Composite speed initial Fmin (Hz) [100Hz to 20kHz]
2	TargetPos_X	DINT	0	IN: X-axis (CH0) target position [-8388608 to 8388607]
3	TargetPos_Y	DINT	0	IN: Y-axis (CH2) target position [-8388608 to 8388607]
4	PassPos_X	DINT	0	IN: X-axis (CH0) pass position [-8388608 to 8388607]
5	PassPos_Y	DINT	0	IN: Y-axis (CH2) pass position [-8388608 to 8388607]
6	Radius	DINT	0	OUT: Radius
7	CenterPos_X	DINT	0	OUT: X-axis (CH0) center position
8	CenterPos_Y	DINT	0	OUT: Y-axis (CH2) center position

IN: Control code
Bit 16: FALSE=Stop, TRUE=Continue
Bit 12: FALSE=CW (right), TRUE=CCW (left)
Bit 8: FALSE=Pass, TRUE=Center
Bit 4: FALSE=Incremental, TRUE=absolut
Bits 0,1: 0=CW/CCW, 2=PLS+SIGN(forward OFF), 3=PLS+SIGN(forward ON)

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	Trigger	BOOL	FALSE	
1	VAR	DataTable	PULSE_ARC_PASS	ControlCode := 16#1021, Speed := 18000, TargetPos_X := 8000000, TargetPos_Y := 5000000, PassPos_X := 8000000, PassPos_Y := 600000, CenterPos_X := 8300000, CenterPos_Y := 5500000	

LD



Part III F/P Instructions

Chapter 27

Timer Instructions

TM_1ms_FB

Timer for 1ms intervals (0 to 32.767s)

Steps: 3-4

Description This timer for 0.001s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.

For the TM_1ms_FB function block declare the following:

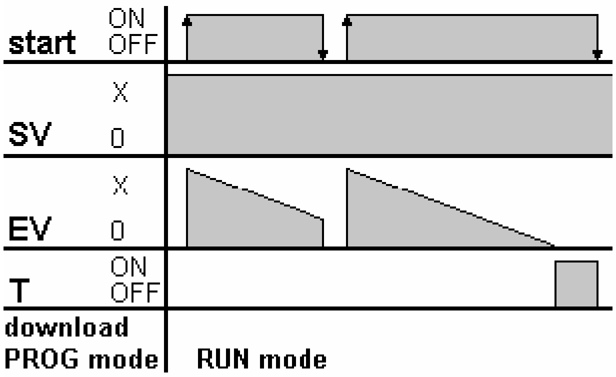
start — **start contact**
each time a rising edge is detected, the set value SV is copied to the elapsed value EV and the timer is started

SV — **set value**
the defined ON-delay time (0 to 32.767s)

T — **timer contact**
is set when the time defined at SV has elapsed, this means when EV becomes 0

EV — **elapsed value**
count value from which 1 is subtracted every 0.001s while the timer is running

Time Chart:



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types: **Availability of TM_1ms_FB (see page 935)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block TM_1ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_1ms_FB		
2	VAR	Start_Contact	BOOL	FALSE	
3	VAR	Alarm_Relay_1	BOOL	FALSE	
4	VAR	Alarm_Relay_2	BOOL	FALSE	

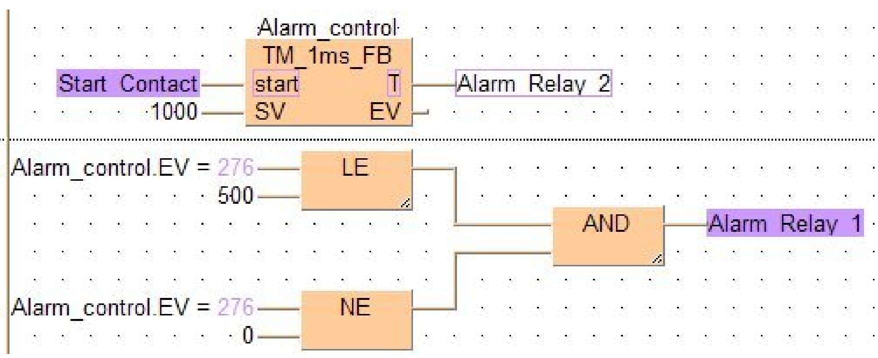
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1ms. When **EV** reaches the value 0 (after 1 second as SV = 1000 with the timer type TM_1ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 500 (after 0.5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



```
ST   Alarm_Control( start:= Start_Contact ,
             SV:= 1000,
             T=> Alarm_Relay_2 ,
             EV=> Alarm_Control.EV );
     (*The ON-delay time is 1000ms*)
     Alarm_Relay_1:= Alarm_Control.EV <= 500 & Alarm_Control.EV
     <> 0;
     (*Alarm_Relay_1 is set to TRUE after 500ms*)
```

TM_10ms_FB

Timer for 10ms intervals (0 to 327.67s)

Steps: 3-4

Description This timer for 0.01s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.

For the TM_10ms_FB function block declare the following:

start — **start contact**

each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV — **set value**

the defined ON-delay time (0 to 327.67s)

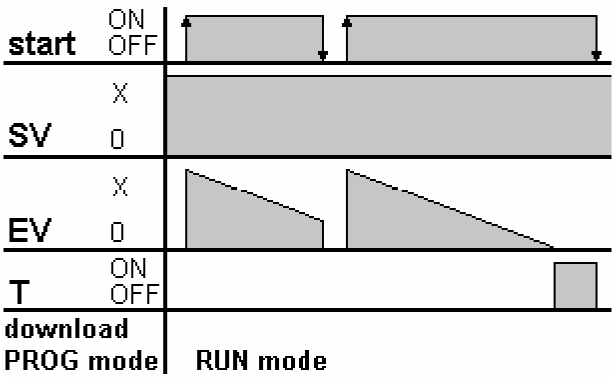
T — **timer contact**

is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV — **elapsed value**

count value from which 1 is subtracted every 0.01s while the timer is running

Time Chart:



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types: **Availability of TM_10ms_FB (see page 936)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block TM_10ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.

Class	Identifier	Type	Initial	Comment
VAR	Alarm_Control	TM_10ms_FB		
VAR	Start_Contact	BOOL	FALSE	
VAR	Alarm_Relay_1	BOOL	FALSE	
VAR	Alarm_Relay_2	BOOL	FALSE	

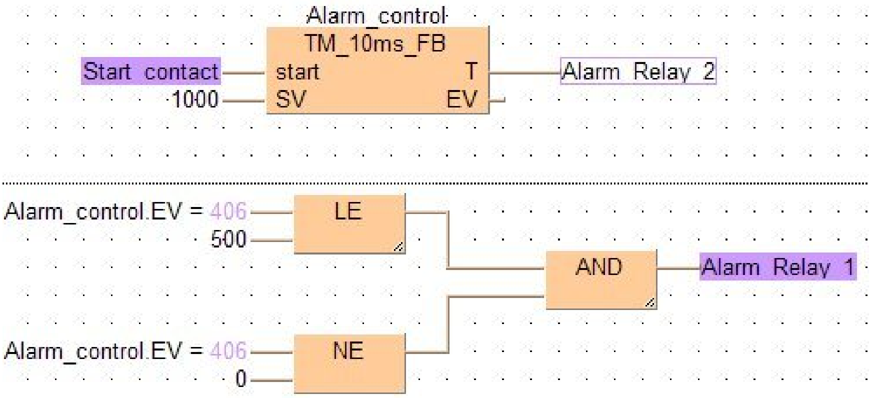
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 10ms. When **EV** reaches the value 0 (after 10 second as SV = 1000 with the timer type TM_10ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 500 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



ST

```
Alarm_Control( start:= Start_Contact ,
        SV:= 1000,
        T=> Alarm_Relay_2 ,
        EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)

Alarm_Relay_1:= Alarm_Control.EV <= 500 & Alarm_Control.EV
<> 0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_100ms_FB

Timer for 100ms intervals (0 to 3276.7s)

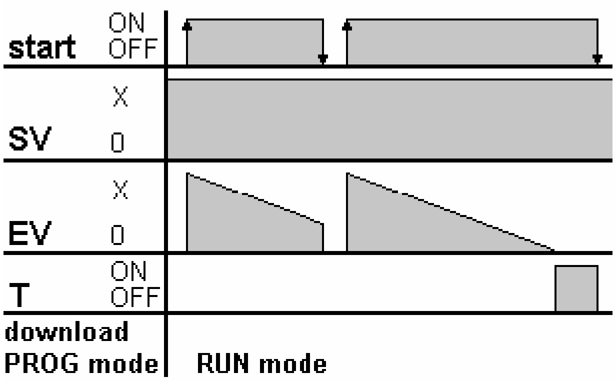
Steps: 3-4

Description This timer for 0.1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.

For the TM_100ms_FB function block declare the following:

start	**start contact** each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started
SV	**set value** the defined ON-delay time (0 to 3276.7s)
T	**timer contact** is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0
EV	**elapsed value** count value from which 1 is subtracted every 0.1s while the timer is running

Time Chart:



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types: **Availability of TM_100ms_FB (see page 936)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block TM_100ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_100ms_FB		
2	VAR	Start_Contact	BOOL	FALSE	
3	VAR	Alarm_Relay_1	BOOL	FALSE	
4	VAR	Alarm_Relay_2	BOOL	FALSE	

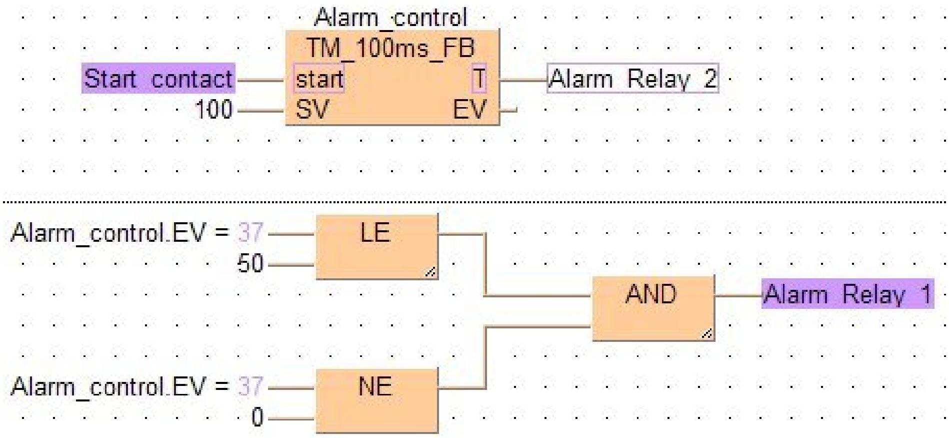
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 100ms. When **EV** reaches the value 0 (after 10 seconds as **SV** = 100 with the timer type TM_100ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 50 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



ST
```
Alarm_Control( start:= Start_Contact ,
        SV:= 100,
        T=> Alarm_Relay_2 ,
        EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)
Alarm_Relay_1:= Alarm_Control.EV <= 50 & Alarm_Control.EV <>
0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_1s_FB

Timer for 1s intervals (0 to 32767s)

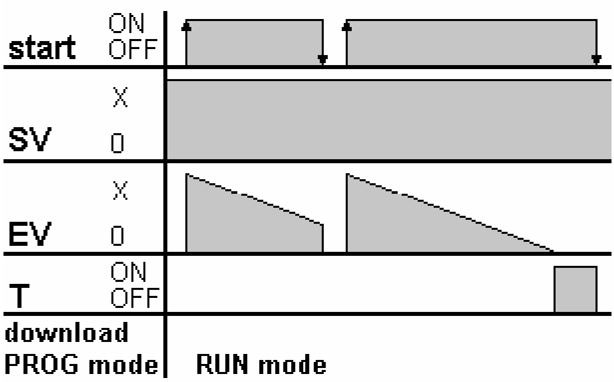
Steps: 4-5

Description This timer for 1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.

For the TM_1s_FB function block declare the following:

start	**start contact** each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started
SV	**set value** the defined ON-delay time (0 to 32767s)
T	**timer contact** is set when the time defined at SV has elapsed, this means when **EV** becomes 0
EV	**elapsed value** count value from which 1 is subtracted every 1s while the timer is running

Time Chart:



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types: **Availability of TM_1s_FB (see page 935)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

All input and output variables which are used for programming the function block TM_1s_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.

	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_1s_FB		
1	VAR	Start_Contact	BOOL	FALSE	
3	VAR	Alarm_Relay_1	BOOL	FALSE	
4	VAR	Alarm_Relay_2	BOOL	FALSE	

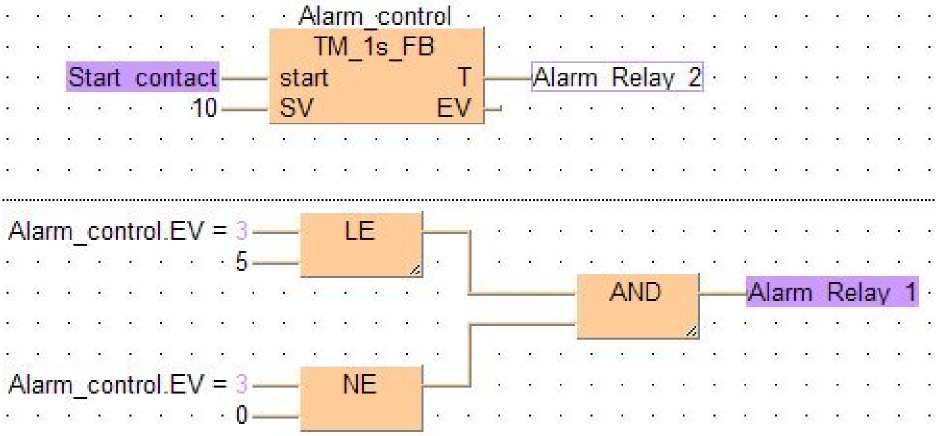
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1s. When **EV** reaches the value 0 (after 10 seconds as **SV** = 10 with the timer type TM_1s_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 5 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



ST

```
Alarm_Control( start:= Start_Contact ,
        SV:= 10,
        T=> Alarm_Relay_2 ,
        EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)
Alarm_Relay_1:= Alarm_Control.EV <= 5 & Alarm_Control.EV <>
0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_1ms

Timer for 1ms intervals (0 to 32.767s)

Steps: 3-4

Description The TM_1ms instruction sets the ON-delay timer for 0.001s units (0 to 32.767s).

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/2 of units (0.5ms) + scan time

Example:

150ms time set value and 8ms PLC scan time

Upper limit = 150 - 0 + 8 = 158ms
Lower limit = 150 -0.5 +8 = 157.5ms

The result is a timing operation from 157.5ms to 158ms.

PLC types: **Availability of TM_1ms (see page 935)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer contact
SV	INT, WORD	timer address in system registers 5 and 6
T	BOOL	set value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

☞

- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_1ms_FB (see page 760).

TM_10ms

Timer for 10ms intervals (0 to 327.67s)

Steps: 3-4

Description The TM_10ms instruction sets the ON-delay timer for 0.01 s units (0 to 327.67s).

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/4 of units (2.5ms) + scan time

Example:

150ms time set value and 8ms PLC scan time

Upper limit = 150 - 0 + 8 = 158ms
Lower limit = 150 -2.5 +8 = 155.5ms

The result is a timing operation from 155.5ms to 158ms.

PLC types: **Availability of TM_10ms (see page 936)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_10ms_FB (see page 763).

TM_100ms

Timer for 100ms intervals (0 to 3276.7s)

Steps: 3-4

Description The TM_100ms instruction sets the ON-delay timer for 0.1s units (0 to 3276.7s).

The **TM** instruction is a down type preset timer.

The area used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value **-** 0 to 1/4 of units (25ms) + scan time

Example:

1500ms time set value and 8ms PLC scan time

Upper limit = 1500 - 0 + 8 = 1508ms
Lower limit = 1500 -25 +8 = 1483ms

The result is a timing operation from 1483ms to 158ms.

PLC types: **Availability of TM_100ms (see page 936)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_100ms_FB (see page 766).

TM_1s

Timer for 1s intervals (0 to 32767s)

Steps: 4-5

Description The TM_1s instruction sets the ON-delay timer for 1s units (0 to 32767s).

The area used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/4 of units (250ms) + scan time

Example:

150s time set value and 8ms PLC scan time

Upper limit = 150000 - 0 + 8 = 150008ms
Lower limit = 150000 -250 +8 = 149758ms

The result is a timing operation from 149758ms to 158ms.

PLC types: **Availability of TM_1s (see page 935)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_1s_FB (see page 769).

F137_STMR

Timer 16-bit

Steps: 5

Description The auxiliary timer instruction **F137_STMR** is a down type timer. The formula of the timer-set time is 0.01 sec. * set value **s** (time can be set from 0.01 to 327.67 sec.). If you use the special internal relay R900D as the timer contact, be sure to program it at the address immediately after the instruction.

Timer operation:

- If the trigger **EN** of the auxiliary timer instruction (STMR) is in the ON-state, the constant or value specified by **s** is transferred to the area specified by **d**.
- During the timing operation, the time is subtracted from the value in the area specified by **d**.
- The output ENO turns ON when the value in the area specified by **d** becomes 0.

PLC types: **Availability of F137_STMR (see page 928)**

Data types

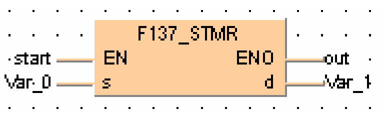
Variable	Data type	Function
s	INT, WORD	16-bit area or equivalent constant for timer set value
d	INT, WORD	16-bit area for timer elapsed value

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example



F183_DSTM

Timer 32-bit

Steps: 7

Description The F183 instruction activates an upward counting 32-bit timer which works on-delayed. The smallest counting unit is 0.01s. During execution of F183 (start = TRUE), elapsing time is added to the elapsed value **d**. The timer output will be enabled when the elapsed value **d** equals the set value **s**. If the start condition start is set to FALSE, execution will be interrupted and the elapsed value **d** will be reset to zero. The set value **s** can be changed during execution of F183.

The delay time of the timer can be calculated using the following formula: (Set Value **s**) * (0.01s) = on-delay

PLC types: **Availability of F183_DSTM (see page 930)**

Data types

Variable	Data type	Function
s	DINT, DWORD	set value, range 0 to 2147483647
d	DINT, DWORD	elapsed value, range 0 to 2147483647

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	DWY	DWR	-	DSV	DEV	DDT	-	-	-

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU header

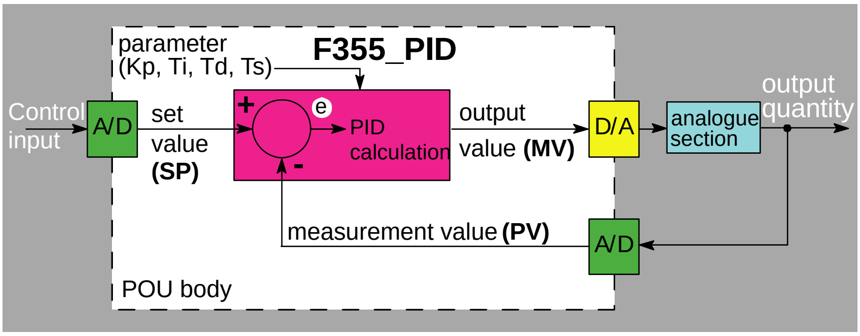
	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	SetValue	DINT	1000	10 seconds
2	VAR	TimerOutput	BOOL	FALSE	turns on when 10s have elapsed
3	VAR	ElapsedValue	DINT	0	

LD

Chapter 28

Process Control Instructions

28.1 Explanation of the Operation of the PID Instuctions



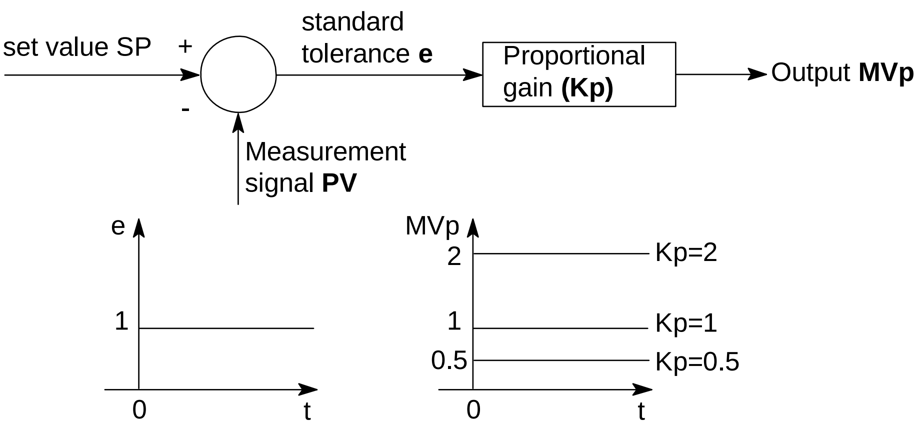
The above POU body represents the standard control loop. The control input is determined by the user (e.g. desired room temperature of 22°C). After the A/D conversion the set value (SP) is entered as the input value for the PID processing instruction. The measured value (PV) (e.g. current room temperature) is normally transmitted via a sensor and entered as the input value for the PID processor. F355_PID calculates the standard tolerance e from the set value and the measured value (e = set value - measured value). With the parameters given (proportional gain Kp, integral time Ti, ...) a new output value (MV) is calculated in increments set by the control cycle Ts. This result is then applied to the actuator (e.g. a fan that regulates room temperature) after the D/A conversion. The analog section represents the system's actuator, e.g. heater and temperature regulation of a room.

A PID operation consists of three components:

1. Proportional part (P part)

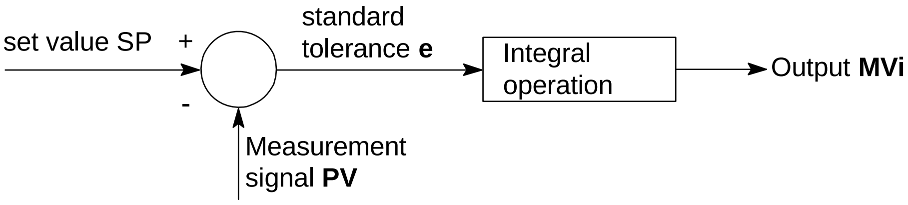
A proportional part generates an output that is proportional to the input. The proportional gain Kp determines by how much the input value is increased or decreased.
A proportional part can be a simple electric resistor or a linear amplifier, for example.

The P part displays a relatively large maximum overshot, a long settling time and a constant standard tolerance.

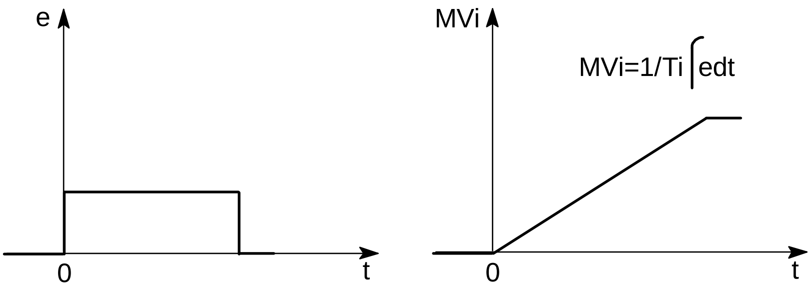


2. Integral part (I part)

An integral part produces an output quantity that corresponds to the time integral and input quantity (area of the input quantity). The integral time thus evaluates the output quantity MVi. The integral part can be a quantity scale of a tank that is filled by a volume flow, for example. Because of the slow reaction time of the integral part, it has a larger maximum overshot than the P component, but no constant standard tolerance.

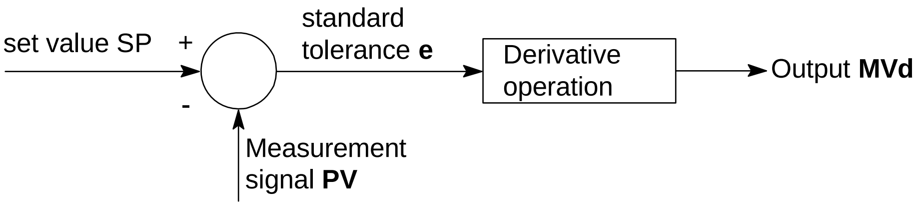


Example Input quantity **e** and the output quantity **MVi** produced.

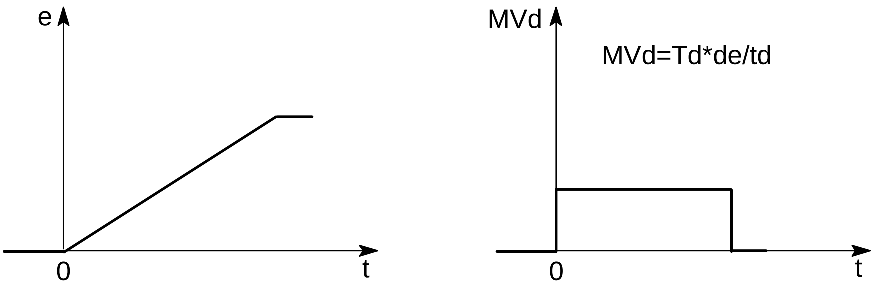


3. Derivative part (D part)

The derivative part produces an output quantity that corresponds to the time derivation of the input quantity. The derivative time corresponds to the weighting of the derived input quantity. A derivative component can be an RC-bleeder (capacitor hooked up in series and resistance in parallel), for example.

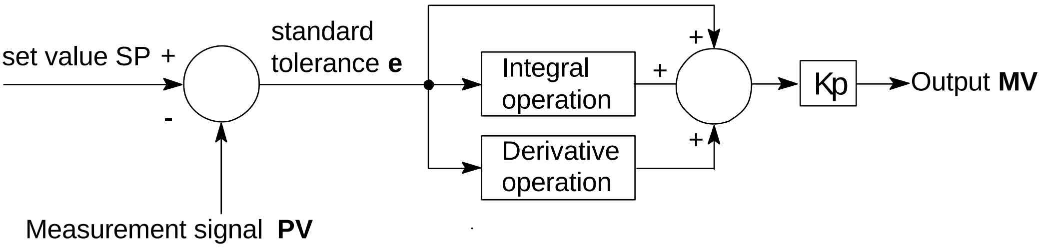


Example Input quantity **e** and the output quantity **MVd** produced.



4. PID controller

A PID controller is a combination of a P component, an I component and a D component. When the parameters Kp, Ti and Td are optimally adjusted, a PID controller can quickly control and maintain a quantity at a predetermined set value.



Reference equations for calculating the controller output MV

The following equations are used to calculate the controller output MV under the following conditions:

In general:
The output value at time period **n** is calculated from the previous output value (n-1) and the change in the output value in this time interval.

$$MV_n = MV_{n-1} + \Delta MV$$

Reverse operation PI-D Control = 16#X000

$$\Delta MV = Kp \times \left[\left(e_n - e_{n-1}\right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

Part III F/P Instructions

Forward operation PI-D Control = 16#X001

$$\Delta MV = Kp \times \left[\left(e_n - e_{n-1} \right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = \left(\eta\beta - 1 \right) D_{n-1} + \beta \left(PV_n - PV_{n-1} \right)$$

$$\eta = \frac{1}{8} \left(constant \right)$$

$$\beta = \frac{Td}{\left(Ts + \eta Td \right)}$$

Reverse operation I-PD Control = 16#X002

$$\Delta MV = Kp \times \left[\left(PV_{n-1} - PV_n \right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = \left(\eta\beta - 1 \right) D_{n-1} + \beta \left(PV_{n-1} - PV_n \right)$$

$$\eta = \frac{1}{8} \left(constant \right)$$

$$\beta = \frac{Td}{\left(Ts + \eta Td \right)}$$

Forward operation I-PD Control = 16#X003

$$\Delta MV = Kp \times \left[\left(PV_n - PV_{n-1} \right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = \left(\eta\beta - 1 \right) D_{n-1} + \beta \left(PV_n - PV_{n-1} \right)$$

$$\eta = \frac{1}{8} \left(constant \right)$$

$$\beta = \frac{Td}{\left(Ts + \eta Td \right)}$$

PID processing instructions:

- PID_FB_DUT (see page 802)
- PID_FB (see page 799)
- F355_PID_DUT (see page 788)

F355_PID_DUT — PID processing instruction

Steps: 4

Description The PID processing instruction is used to regulate a process (e.g. a heater) given a measured value (e.g. temperature) and a predetermined output value (e.g. 20°C).

The function calculates a PID algorithm whose parameters are determined in a data table in the form of an ARRAY with 30 elements that is entered at input **s**.

The required data table PID_DUT_31 contains the following parameters:

Parameter	Data type	Function
Control	WORD	Control mode
SP	INT	Set point value
PV		Process value
MV		Manipulated value
LowerLimit		Output lower limit
UpperLimit		Output upper limit
Kp		Proportional gain
Ti		Integral time
Td		Derivative time
Ts		Control cycle
AT_Progress		Auto-tuning progress
Dummies	ARRAY (see page 20) [11 .. 30] OF WORD	are utilized internally by the PID controller

PLC types: **Availability of F355_PID_DUT (see page 932)**

Data types

Variable	Data type	Function
s	PID_DUT_31	Detailed explanation of parameters

Operands

For	Relay				T/C		Register			Constant
s	-	-	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the parameter settings are outside the allowed range.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Global Variable List

In the global variable list, all values of global inputs and outputs are declared that are used for programming this function.

Global Variables

	Class	Identifier	FP ...	IEC_Address	Type	Initial	A...	Comment
0	VAR_GLOBAL	EnableAutoTuning	X0	%IX0.0	BOOL	FALSE		Switch Autotuning on
1	VAR_GLOBAL	Set_Value_SP	WX4	%IW4	INT	0		A/D CH0
2	VAR_GLOBAL	Process_Value_PV	WX5	%IW5	INT	0		A/D CH1
3	VAR_GLOBAL	Output_Value_MV	WY4	%QW4	INT	0		D/A

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

PID_Control_IL

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	EnableAutoTuning	BOOL	FALSE	Switch Autotuning on
1	VAR_EXTERNAL	Set_Value_SP	INT	0	A/D CH0
2	VAR_EXTERNAL	Process_Value_PV	INT	0	A/D CH1
3	VAR_EXTERNAL	Output_Value_MV	INT	0	D/A
4	VAR	PidParameters	PID_DUT_31		PID Parameters

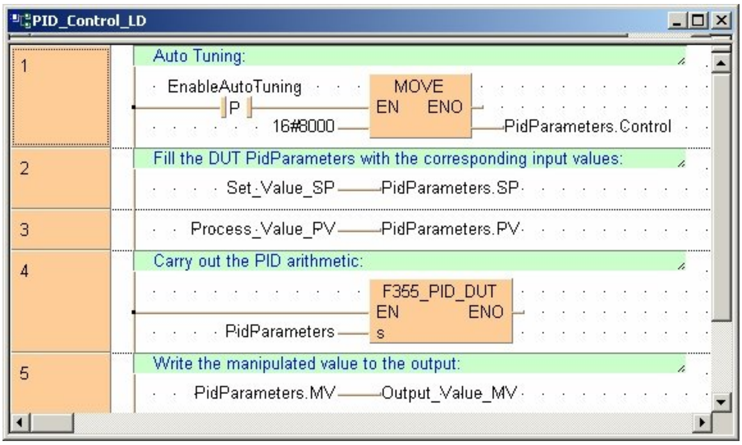
In the initialization of the variable PidParameters of the data type PID_DUT_31 , the upper limit of the controller output is set to 4000. The proportional gain Kp is initially set at 80 (8), Ti and Td at 200 (20s) and the control cycle Ts at 100 (1s).

Body

The standard function MOVE copies the value 16#8000 to the member Control of the DUT PidParameters when the variable EnableAutoTuning is set from FALSE to TRUE (i.e. activates the control mode auto-tuning in the function F355_PID_DUT).
The variables **Set_Value_SP** and **Process_Value_PV** are assigned to the members SP and PV of the DUT PidParameters. They receive their values from the A/D converter CH0 and CH1.

Because the F355_PID_DUT function block has an EN output connected directly to the power rail, the function is carried out when the PLC is in RUN mode. The calculated controller output stored the member MV of the DUT PidParameters is assigned to the variable **Output_Value_MV**. Its value is returned via a D/A converter from the PLC to the output of the system.

LD



ST

```
(* Auto Tuning: *)
if DF(EnableAutoTuning) then
      PidParameters.Control:=16#8000;
end_if;

(* Fill the DUT PidParameters with the corresponding input
values: *)
PidParameters.SP:=Set_Value_SP;
PidParameters.PV:=Process_Value_PV;

(* Carry out the PID arithmetic: *)
F355_PID_DUT(PidParameters);

(* Write the manipulated value to the output: *)
Output_Value_MV:=PidParameters.MV;
```

F356_PID_PWM

Easy PID processing instruction

Steps: 10

Description PID processing is performed to keep the processing value (PV) as close as possible to the set point value (SP). In contrast to F355_PID_DUT (see page 788), this instruction enables a PWM output (on-off output). Auto-tuning is also available to automatically calculate the PID control data Kp, Ti and Td.

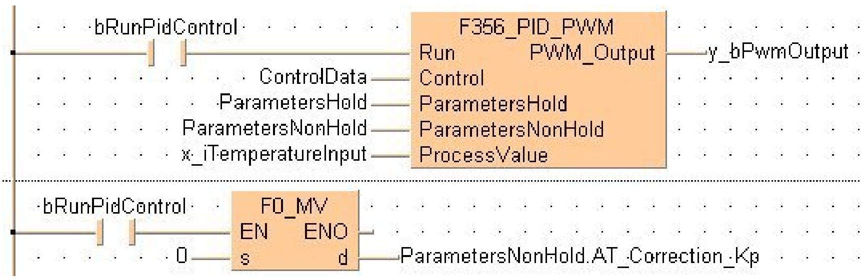
Abbreviations used when describing PID processing

Abbreviation	What it stands for	Also know as
PV	Process value	Measured value
SP	Set point value	Target value or set value
MV	Manipulated value	Output value
Ts	**T**ime, **s**ampling	Sampling time, Cycle time
Ti	**T**ime, **i**ntegral	Integral time
Td	**T**ime, **d**erivative	Derivative time
Kp	Proportional gain	-
AT	Auto-tuning	-

Precautions on prog.

1. When the input at **Run** is executed, the data in the argument **ParametersNonHold** is initialized.
   If you want a value in the DUT to use non-default values, write the values into the DUT using a MOVE instruction, for example, which must be triggered continuously by a TRUE condition.
2. F356_PID_PWM must be executed once and only once per scan. Therefore, do not execute F356_PID_PWM in interrupt programs or loops.
3. Do not turn off the execution condition during PID processing. Otherwise, PID processing will be disabled.
4. If you do not want parallel PWM output cycles, e.g. to control multiple objects, delay the start-up times accordingly, e.g. by employing a timer instruction.

Example



PLC types: **Availability of F356_PID_PWM (see page 932)**

The period (cycle) of the pulsed output is the sampling time Ts (the frequency of the pulsed output is 1/Ts) and the duty is the MV in 0.01% units, e.g. MV = 10000 means a duty of 100%.

Data types

Variable	Data type	Function
Run	BOOL	Start condition
Control	F356_Control_DUT (see page 794)	Control data
Parameters Hold	F356_Parameters_Hold_DUT (see page 795)	PID control parameters
Parameters NonHold	F356_Parameters_NonHold_DUT (see page 796)	Manipulated (output) value (MV), additional control mode area, auto-tuning related area and working area
ProcessValue	INT	Process value (-30000 to 30000)
PWM_Output (see note)	BOOL	Pulsed width output (optional instead of manipulated (output) value MV)

Operands

For	Relay				T/C		Register			Constant
Control	-	WY	WR	WL	SV	EV	DT	LD	FL	-
Parameters	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
Process Values	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any parameter of F356_Parameters_NonHold_DUT is out of the setting range
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	- the area specified with **UpperLimit** or **LowerLimit** is out of the valid range

Detailed information:

Control conditions: F356_Parameters_Hold_DUT (see page 795)

Target value (SP) and the control parameters: F356_Parameters_NonHold_DUT (see page 796)

■ Additional notes on auto-tuning

- The members **AT_Progress** in F356_Parameters_NonHold_DUT (see page 796) and **b1_AT_Complete** in F356_Control_DUT (see page 794) are cleared at the leading edge of the auto-tuning signal.
- When auto-tuning has completed successfully, the member **b1_AT_Complete** in F356_Control_DUT (see page 794) is set, and the auto-tuning done code is stored in the member **AT_Progress** in F356_Parameters_NonHold_DUT (see page 796).
- When auto-tuning is aborted, the parameters of Kp, Ti and Td are not changed.

Example

In this example, the same POU header is used for all programming languages.

Global Variable List

In the global variable list, all values of global inputs and outputs are declared that are used for programming this function.

Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	x_iTemperatureInput	WX2	%IW2	INT	0
1	VAR_GLOBAL	y_bPwmOutput	Y0	%QX0.0	BOOL	FALSE
2	VAR_GLOBAL					

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	x_iTemperatureInput	INT	0
1	VAR_EXTERNAL	y_bPwmOutput	BOOL	FALSE
2	VAR	bStartAutoTuning	BOOL	FALSE
3	VAR	bRunPidControl	BOOL	FALSE
4	VAR	ControlData	F356_Control_DUT	
5	VAR	ParametersHold	F356_Parameters_Hold_DUT	
6	VAR	ParametersNonHold	F356_Parameters_NonHold_DUT	

Body

Specify the member SP (set point value) of F356_Parameters_Hold_DUT (see page 795) before operation.

When **bRunPidControl** turns on, the work area specified with the F356_Parameters_NonHold_DUT (see page 796) will be initialized. However, only the member **MV** (manipulated value) can be held depending on the status of the flag **b2_HoldMV** of F356_Control_DUT (see page 794).

The default control conditions are:

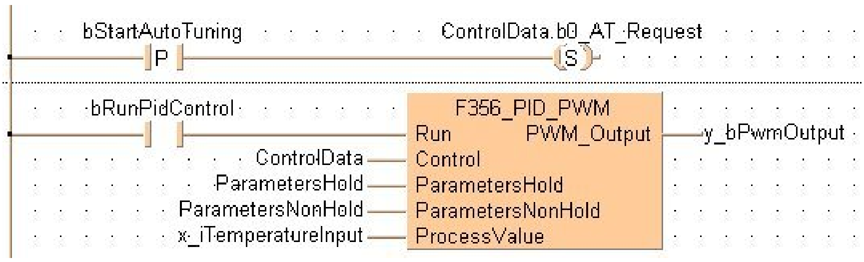
- operation cycle = 1 sec
- proportional-derivative type reverse operation (heating)
- PWM resolution = 1000.

PID control starts from the next scan, and PWM output is executed for **PWM_Output**.

If the member flag **b0_AT_Request** of **ControlData**, a DUT with overlapping elements, is set, auto-tuning begins. When auto-tuning has completed successfully, the member flag **b1_AT_Complete** of **ControlData** is set and Kp, Ti and Td are set for the PID control. If **bRunPidControl** is still on, it will change to PID control automatically and the PWM output will be executed.

If the execution condition bRunPidControl has turned off during PID control, PWM_Output also turns off. However, only the member MV (manipulated value) can be held depending on the status of the flag b2_HoldMV of F356_Control_DUT (see page 794).

LD



ST

```
(* Auto Tuning: *)
if DF(bStartAutoTuning) then
      ControlData.b0_AT_Request:=TRUE;
end_if;
y_bPwmOutput:=F356_PID_PWM(   Run := bRunPidControl,
            Control := ControlData,
            ParametersHold := ParametersHold,
            ParametersNonHold := ParametersNonHold,
            ProcessValue := x_iTemperatureInput);
```

28.1.1 F356_Control_DUT

This data type, a DUT with overlapping elements, is predefined in the System Library and is used by the function F356_PID_PWM (see page 791).

	Identifier	Type	Initial	Comment
0	w0	WORD		Word 0
1	b0_AT_Request	BOOL		Word 0 - Bit 0: Auto-tuning reque...
2	b1_AT_Complete	BOOL		Word 0 - Bit 1: Auto-tuning has c...
3	b2_HoldMV	BOOL		Word 0 - Bit 2: Hold the output M...

We recommend specifying the non-hold type area.

Identifier	Description
w0	Since this is a DUT with overlapping elements, the BOOL members occupy the same data areas as the WORD member **w0**. Therefore by using **w0** you can simultaneously access all bits.

Identifier	Description
b0_AT_Request (bit 0)	When set, auto- tuning is requested. This bit is reset with the instruction F356_PID_PWM when auto-tuning is complete. Reset this bit to cancel auto-tuning. When not set, PID control will be executed.
b1_AT_Complete (bit 1)	When set, auto-tuning has been completed successfully.
b2_HoldMV (bit 2)	When set, the manipulated value output is held by switching F356_PID_PWM (see page 791) from off to on.
Bits 3 to F	Are reserved and normally 0.

28.1.2 F356_Parameters_Hold_DUT

This data type is predefined in the System Library and is used by the function F356_PID_PWM (see page 791).

	Identifier	Type	Initial	Comment
0	SP	INT	0	Set point value [range from -30000 to 30000]
1	Kp	INT	0	Proportional gain in unit 0.1 [range from 1 to 9999 (from 0.1 to 999.9)]
2	Ti	INT	0	Integral time in unit 0.1s [range from 1 to 30000 (from 0.1s to 3000s)]
3	Td	INT	0	Derivative time in unit 0.1s [range from 1-10000 (from 0.1s to 1000s)]

This DUT specifies the control parameter (4 words). We recommend allocating the area used by this data type to the hold-type operation memory.

Variable	Comment	Range
SP	Set point value.	-30000 to 30000
Kp	Stores proportional gain (KP). After auto-tuning has been completed, it is automatically set.	1 to 9999
Ti	Stores integral time (TI). After auto- tuning has been completed, it is automatically set.	0 to 30000
Td	Stores derivative time (TD). After auto-tuning has been completed, it is automatically set.	1 to 10000

If the parameters Kp, Ti and Td are all 0 when PID operation has started, they are initialized at 1,1 and 0, respectively, and operation continues. If any parameter for Kp, Ti, or Td is out of range when auto-tuning has started, they are initialized at 1, 1 and 0, respectively, and auto-tuning continues.

28.1.3 F356_Parameters_NonHold_DUT

This data type is predefined in the System Library and is used by the function F356_PID_PWM (see page 791).

	Identifier	Type	Initial	Comment
0	MV	INT	0	Manipulated value, output value [range from -10000 to 10000, default 0]
1	LowerLimit	INT	0	Output lower limit [range from -10000, default 0]
2	UpperLimit	INT	0	Output upper limit [range up to 10000, default 10000]
3	PV_Band_WithFullOutput	INT	0	no pid control and full 100% output if the PV is in this band [range from 0...
4	Ts	INT	0	Sampling time in unit 0.01s [range from 1 to 6000, default 100 (1s)]
5	ControlMode	INT	0	0: Derivative - Reverse (Heating) (default value)
6	AT_Bias	INT	0	Bias value for autotuning [default 0]
7	AT_Correction_Kp	INT	0	Correction data of autotuning result Kp in unit % [range from 50% to 50...
8	AT_Correction_Ti	INT	0	Correction data of autotuning result Ti in unit % [range from 50% to 500...
9	AT_Correction_Td	INT	0	Correction data of autotuning result Td in unit % [range from 50% to 50...
10	AT_Progress	INT	0	Auto-tuning progress from 0 to 5 [default 0]
11	WorkingArea	ARRAY [11..29] OF INT		Working area (19 dummy words fill up to 30 words)

This DUT specifies the manipulated value (MV) and the control parameters (4 words).

Variable	Comment		Default value	Range
MV	Stores the manipulated (output) value		0	-10000 to 10000
LowerLimit	Sets the lower limit value of manipulated value (MV)		0	min. -10000
UpperLimit	Sets the upper limit value of the manipulated value (MV)		10000	max. 10000
PV_Band_WithFullOutput	No PID control is performed and the output is set to 100% until the defined level (0% - 80%) of the set point value has been reached.		0	0% to 80%
Ts	Sets sampling time (TS). Setting unit = 10ms, default value = 1sec. The sampling time is also the cycle time of the pulsed output.		100	1 to 6000
ControlMode	0	Derivative type (PI-D) reverse, e.g. heating	0	0 to 3
	1	Derivative type (PI-D) forward, e.g. cooling		
	2	Proportional-derivative type (I-PD) reverse, e.g. heating		
	3	Proportional-derivative type (I-PD) forward, e.g. cooling		
AT_Bias	Sets bias value for performing auto-tuning		0	0
AT_Correction_Kp	Sets correction data of auto-tuning result (KP)		100	50% to 500%
AT_Correction_Ti	Sets correction data of auto-tuning result (Ti)		100	50% to 500%
AT_Correction_Td	Sets correction data of auto-tuning result (Td)		100	50% to 500%
AT_Progress	Stores the status while auto-tuning is being performed		0	0 to 5
WorkingArea	Working area of up to 30 words for PID processing and auto-tuning processing		0	

When the execution condition has turned on, the operation work area is initialized.

◆ NOTE

The default value is written when the execution condition turns on. MV is only output in the range between the upper limit value and lower limit value.

Detailed information on the setting method:

PV_Band_WithFullOutput

Define at what percent of the set point value PID control should start. Below this level, output is 100%.

For example, you have set **PV_Band_WithFullOutput** to 80% and the actual processing value (i.e. measured value) is only 50% of the set point value. In this case the output will be set to 100% and remain at 100% until the processing value reaches 80% of the set point value, at which point PID control will start.

By choosing a greater or lesser percentage, you determine how quickly the set point value is reached.

Fine adjustment of auto-tuning

When auto-tuning has completed, the parameters for Kp, Ti and Td are stored in the members of F356_Parameters_Hold_DUT (see page 795). For fine adjustment, you can now correct the result of auto-tuning with the parameters **AT_Correction_Kp**, **AT_Correction_Ti** and **AT_Correction_Td**.

◆ EXAMPLE

Set **AT_Correction_Kp** to 200 (i.e. 200%): perform auto-tuning to correct Kp to double its value.

Set **AT_Correction_Ti** to 125 (i.e. 125%): perform auto-tuning to correct Ti to 1.25 times its value.

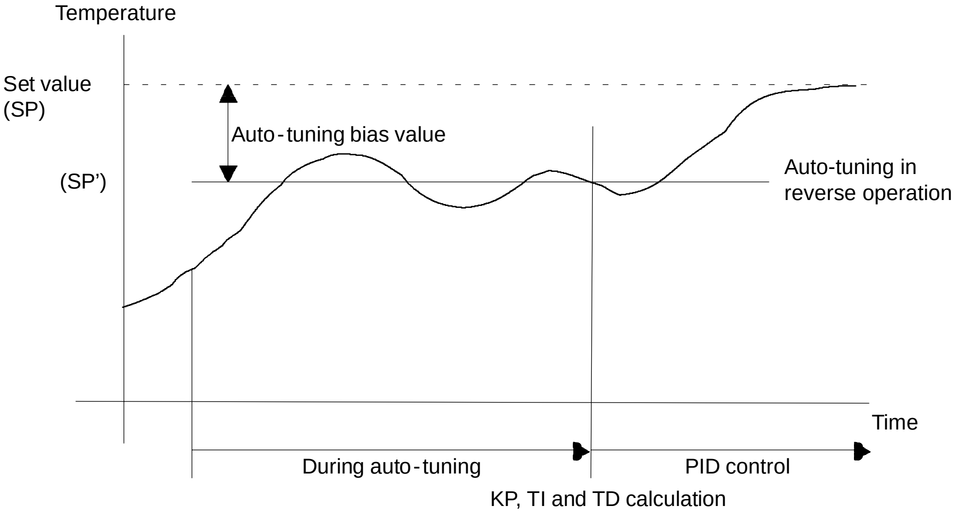
Set **AT_Correction_Td** to 75 (i.e. 75%): perform auto-tuning to correct Td to 0.75 times value.

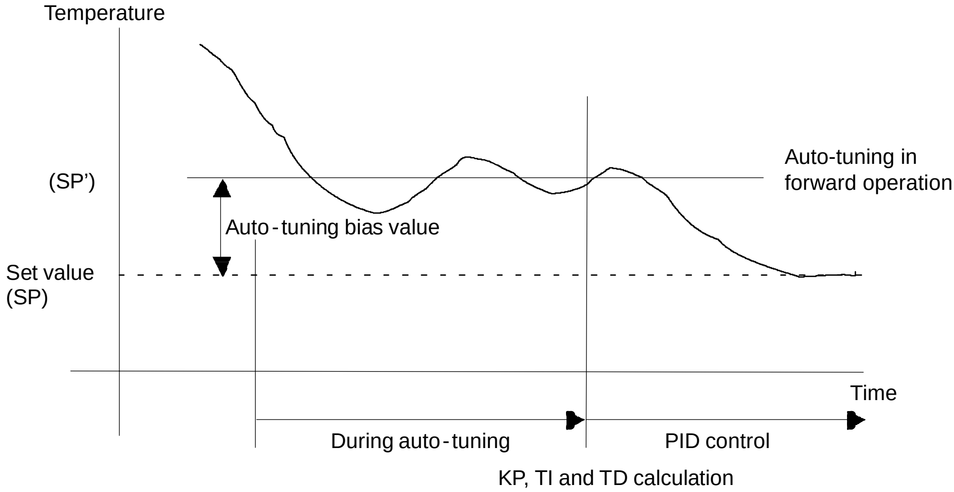
Auto-tuning bias value

In reverse operation, auto-tuning is executed with (set point value (SP) - auto-tuning bias value) as a temporary set point value (SP’).

It is used to control excessive temperature rise during auto-tuning.

In forward operation, auto-tuning is executed with (set point value (SP) + auto-tuning bias value) as a temporary set point value (SP').





◆ NOTE

Even if auto-tuning starts when the processing value (PV) is close to the set point value (SP), auto-tuning is peformed with the above SP'.

PID_FB

PID processing instruction

Description This implementation allows you to set the parameters of F355_PID directly using arguments:

Data types

Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
Automatic	BOOL	FALSE: Manual setting of MV possible TRUE: Automatic PID controlled MV
Forward		FALSE: Reverse operation (heating) TRUE: Forward operation (cooling)
I_PD		FALSE: derivative control (PI-D) TRUE: proportional derivative control (I-PD)
SP	INT	Set point value, Range 0-10000
PV		Process value, Range 0-10000
Kp		Proportional gain, Range: 1-9999, Unit: 0.1
Ti		Integral time, Range: 1-30000, Unit: 0.1s
Td		Derivative time, Range: 1-10000, Unit: 0.1s
Ts		Sampling time, Range: 1-6000, Unit: 0.01s
LowerLimit		Output lower limit, Range: 0-10000
UpperLimit		Output upper limit, Range: 1-10000
Input/output variable (VAR_IN_OUT):		
MV	Manipulated value	

- **Autotuning is not possible using PID_FB. For this, use PID_FB_DUT (see page 802).**
- **The value for MV can be assigned externally either when the program is initialized or when the value of Automatic is FALSE.**
- **In order to achieve maximum resolution and minimum dead time beyond LowerLimit and UpperLimit, their values should, if possible, cover the entire range of 0-10000.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Global Variable List

In the global variable list all global input and output values are declared that are used to program the function. The addresses are depending on the respective PLC-Type.

Global Variables

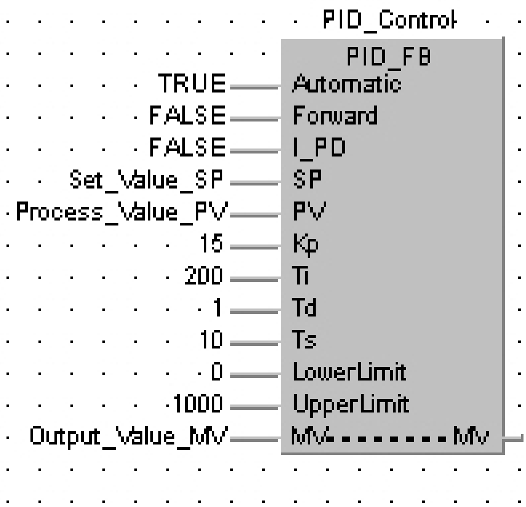
	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Set_Value_SP			INT	0
1	VAR_GLOBAL	Process_V...			INT	0
2	VAR_GLOBAL	Output_Val...			INT	0

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	Set_Value_SP	INT	0	
1	VAR_EXTERNAL	Process_Value_PV	INT	0	
2	VAR_EXTERNAL	Output_Value_MV	INT	0	
3	VAR	PID_Control	PID_FB		

Body LD



```
ST   PID_Control( Automatic:= TRUE,
             Forward:= FALSE,
             I_PD:= FALSE,
             SP:= Set_Value_SP,
             PV:= Process_Value_PV,
             Kp:= 15,
             Ti:= 200,
             Td:= 1,
             Ts:= 10,
             LowerLimit:= 0,
             UpperLimit:= 1000,
             MV:= Output_Value_MV);
```

PID_FB_DUT

PID processing instruction

Description This implementation allows you to access the F355_PID instruction via the structure PID_DUT. This structure is defined in System Libraries / FP Library / DUTs as follows:

Parameter	Function	Range	Unit
Control:	Control mode		
SP:	Set Point value	0-10000	
PV:	Process Value	0-10000	
Kp:	Proportional gain	1-9999	0.1
Ti:	Integral time	1-30000	0.1s
Td:	Derivative time	1-10000	0.1s
Ts:	Sampling time	1-6000	0.01s
LowerLimit:	Output lower limit	0-10000	
UpperLimit:	Output upper limit	1-10000	
MV:	Manipulated output Value	0-10000	

Data types

Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
Automatic	BOOL	FALSE: Manual setting of MV possible TRUE: Automatic PID controlled MV
Input/Output variable (VAR_IN_OUT):		
PidDut	PID_DUT	

☞

- **You may not enter the DUT PID_DUT a second time under DUTs of the current project.**
- **The value for MV can be assigned externally either when the program is initialized or when the value of Automatic is FALSE.**
- **In order to achieve maximum resolution and minimum dead time beyond LowerLimit and UpperLimit, these values should, if possible, cover the entire range of 0-10000.**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

Global Variable List

In the global variable list all global input and output values are declared that are used to program the function. The addresses are depending on the respective PLC-Type.

Global Variables

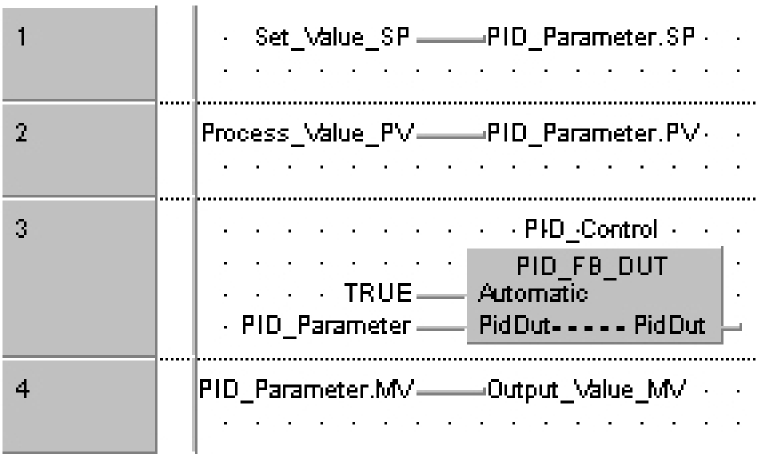
	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Set_Value_SP			INT	0
1	VAR_GLOBAL	Process_V...			INT	0
2	VAR_GLOBAL	Output_Val...			INT	0

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	Set_Value_SP	INT	0	
1	VAR_EXTERNAL	Process_Value_PV	INT	0	
2	VAR_EXTERNAL	Output_Value_MV	INT	0	
3	VAR	PID_Parameter	PID_DUT	UpperLimit := 1000,Kp := 15, Ti := 200,Td := 1,Ts := 10	
4	VAR	PID_Control	PID_FB_DUT		

Body LD



ST

```
PID_Parameter.SP := Set_Value_SP;
PID_Parameter.PV := Process_Value_PV;
PID_Control( Automatic:= TRUE,
        PidDut:= PID_Parameter);
Output_Value_MV := PID_Parameter.MV;
```

Chapter 29

System Register Instructions

SYS1 — Change PLC system setting

Steps: 13

Description The description for SYS1 is divided into the following sections:

- Communication condition setting (see page 806)
- Password setting (see page 810)
- Interrupt setting (see page 811)
- PLC link time setting (see page 813)
- RS485 response time control (see page 815)

PLC types: **Availability of SYS1** **(see page 935)**

Communication condition setting for the COM ports of the CPU

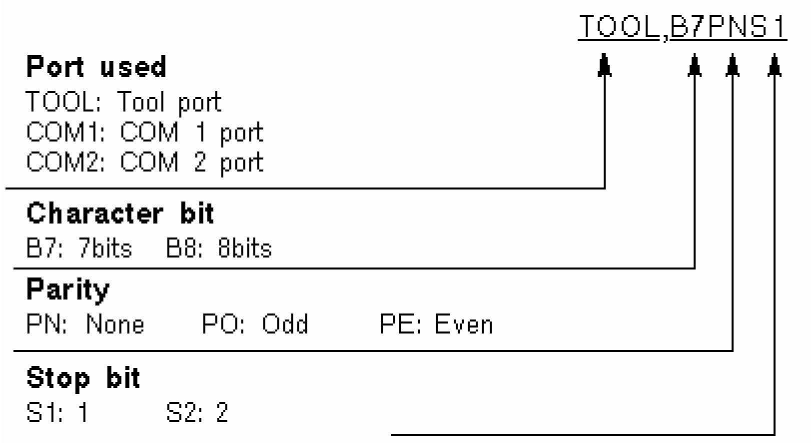
This changes the communication conditions for the COM port or Tool port based on the contents specified by the character constant.

The communication conditions for the port specified by the first keyword are changed to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Contents that can be changed include the following:

1. Communication format
2. Baud rate
3. Unit No.
4. Header and Terminator
5. RS (Request to Send) control

Keyword setting 1. Communication format (Shared by the Tool, COM 1 and COM 2 ports)



2. Baud rate (Shared by the Tool, COM 1 and COM 2 ports)

TOOL,19200

Port used

TOOL: Tool port
COM1: COM 1 port
COM2: COM 2 port

Baud rate

2400: 2,400 bps	4800: 4,800 bps
9600: 9,600 bps	19200: 19,200 bps
38400: 38,400 bps	57600: 57,600 bps
115200: 115,200 bps	

3. Unit No. (Shared by the Tool, COM 1 and COM 2 ports)

TOOL,No1

Port used

TOOL: Tool port
COM1: COM 1 port
COM2: COM 2 port

Unit No.

No1 to No99: No. 1 to No. 99

4. Header and Terminator (Shared by the COM 1 and COM 2 ports)

COM1,STX

Port used

COM1: COM 1 port
COM2: COM 2 port

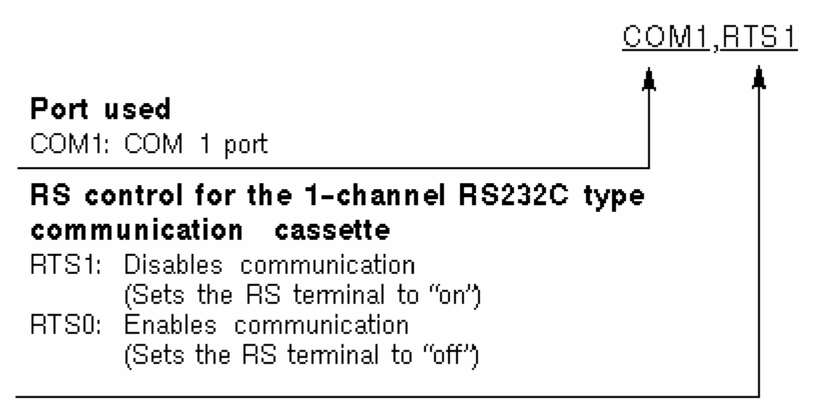
Header

STX: STX
NOSTX: no STX

Terminator

ETX: ETX
CR: CR
CRLF: CR + LF
NOTERM: None

5. RS (Request to Send) control (COM 1 port only)



Precautions during prog.

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- Because the system register settings are changed, a verification error may occur in some cases if verification is carried out with the tools.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any character other than a keyword is specified - no comma is between the first and second keywords - small letters of the alphabet are used to specify the keyword (except for numbers used to specify unit no.) - no communication cassette has been installed when COM1 or COM2 has been set - the setting of the unit no. setting switch is anything other than 0 when COM1 or COM2 has been set and the unit no. is being changed
R9008	%MX0.900.8	for an instant	- the unit no. set using this instruction is anything other than a value between 1 and 99 - the baud rate or transmission format for COM1 has been changed when the PLC link mode is specified for COM1 - the baud rate or transmission format is changed while the Tool port, COM port 1, or COM port 2 is being initialized using MODEM - the communication mode is set to anything other than the general communication mode when header and terminator have been set - any communication cassette other than the 1-channel RS232C type communication cassette is installed when using RS control - the specified unit no. is larger than the largest unit no. specified by the system register when the COM 1 port is in the PLC link mode

Example

In this example the function SYS1 is programmed in ladder diagram (LD).

POU Header

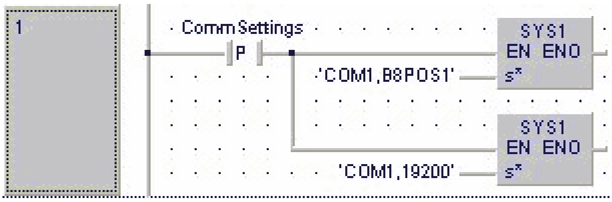
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	CommSettings	BOOL	FALSE	Sets transmission format to: Character bit 8, Parity: Odd Stop bit: 1 Baud rate: 19,200 bps

Body

When **CommSettings** turns on, the transmission format and baud rate for the COM1 port are set as follows: Character bit: 8, Parity: Odd; Stop bit: 1; Baud rate: 19,200 bps.

LD



☞ **The values entered at s* will be right aligned automatically by the compiler.**

Password Setting

This changes the password specified by the controller, based on the contents specified by the character constant.

This changes the password specified by the controller to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Keyword setting for 4-digit hexadecimal password

PASS,ABCD

PASS:Fixed

Password (Example: To set the password to "ABCD")

Keyword setting for 8-digit alphanumeric password (for the FP-X)

The FP-X also supports 8-digit alphanumeric passwords. Enter 'PAS,FP-X␣v␣3'. Spaces at the end of the password are not significant.

PAS,FP-X v 3

PAS : Fixed

Password (Example: To set the password to "FP-X v 3")

Precautions during prog.

- When this instruction is executed, writing to the internal F-ROM takes approximately 100ms.
- If the specified password is the same as the password that has already been written, the password is not written to the F-ROM.
- We recommend using differential execution with this instruction.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any character other than a keyword is specified - no comma is between the first and second keywords - small letters of the alphabet are used to specify the keyword - the data specified for the password setting is any character other than 0 to 9 or A to F, or the specified data consists of other than four digits.
R9008	%MX0.900.8	for an instant	

Part III F/P Instructions

Example In this example the function SYS1 is programmed in ladder diagram (LD).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	ChangePassword	BOOL	FALSE	

Body

When **ChangePassword** turns on, the controller password is changed to "ABCD".

LD

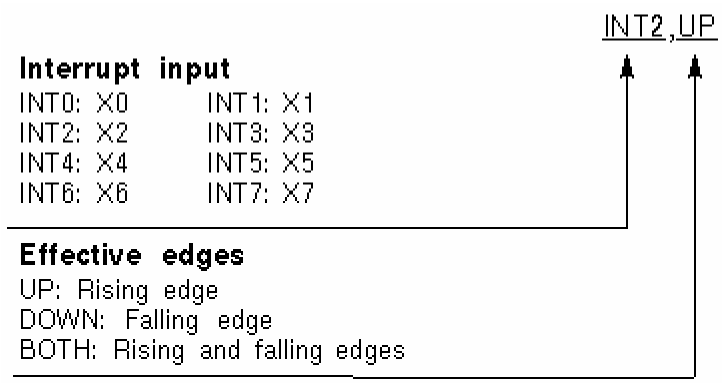
☞ **The values entered at s* will be right aligned automatically by the compiler.**

Interrupt Setting

This sets the interrupt input based on the contents specified by the character constant.

This sets the input specified by the first keyword as the interrupt input, and changes the input conditions to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Keyword setting



For the FP-X you can set INT0 to INT13.

Precautions during prog.

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- When UP or DOWN has been specified, the contents of the system registers change in accordance with the specification, so a verification error may occur in some cases, when the program is verified. When BOTH has been specified, the contents of the system registers do not change.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any character other than a keyword is specified - no comma is between the first and second keywords
R9008	%MX0.900.8	for an instant	- small letters of the alphabet are used to specify the keyword

Example In this example the function SYS1 is programmed in ladder diagram (LD).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	InterruptX1	BOOL	FALSE	

Body When **ChangePassword** turns on, the controller password is changed to "ABCD".

LD

☞ **The values entered at s* will be right aligned automatically by the compiler.**

PLC Link Time Setting

This sets the system setting time when a PLC link is used, based on the contents specified by the character constant.

The conditions specified by the first keyword are set as the time specified by the second keyword. The first and second keywords are separated by a comma.

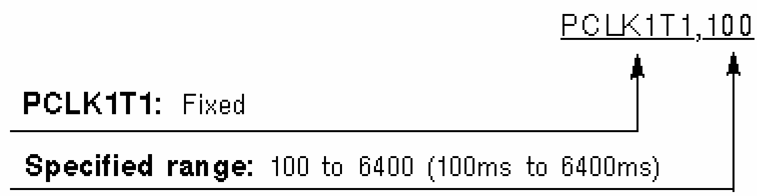
The setting for the link entry waiting time is set if the transmission cycle time is shortened when there are stations that have not joined the link. (Stations that have not joined the link: Stations that have not been connected between the first station and the station with the largest number, or stations for which the power supply has not been turned on.)

The error detection time setting for the transmission assurance relay is set if the time between the power supply being turned off at one station and the transmission assurance relay being turned off at a different station is to be shortened.

Keyword setting

1. Link entry wait time

2. Error detection time for transmission assurance relay



Precautions

Precautions during programming

- The program should be placed at the beginning of all PLCs being linked, and the same values specified.
- This instruction should be specified in order to set special internal relay R9014 as the differential execution condition.
- The setting contents of the system registers are not affected by this instruction being executed.
- Separate first and second keywords with a comma "," and do not use spaces.

Precautions when setting the link entry wait time

- This should be specified such that the value is at least twice that of the largest scan time of all the PLCs that are linked.
- If a short value has been specified, there may be some PLCs that are not able to join the link even though the power supply for that PLC has been turned on.
- If there are any stations that have not joined the link, the setting should not be changed, even if the link transmission cycle time is longer as a result. (The default value is 400 ms.)

Precautions when setting the error detection time for the transmission assurance relay

- This should be specified such that the value is at least twice that of the largest transmission cycle time of all the PLCs that are linked.
- If a short value has been specified, there is a possibility that the transmission assurance relay will malfunction.
- The setting should not be changed, even if the detection time for the transmission assurance relay is longer than the result. (The default value is 6400ms.)

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any character other than a keyword is specified - no comma is between the first and second keywords - small letters of the alphabet are used to specify the keyword - the specified value is outside the specified range
R9008	%MX0.900.8	for an instant	

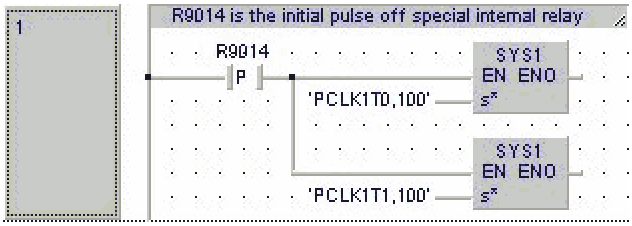
Example Below is an example of a ladder diagram (LD) body for the instruction. Because FP addresses and strings are entered directly instead of using variables, no POU header is required.

Body When R9014 turns on when a PLC link is being used, the link entry wait time and the error detection times for transmission assurance relay are set as follows:

Link entry wait time: 100ms

Error detection time for transmission assurance relay: 100ms.

LD



The values entered at s* will be right aligned automatically by the compiler.

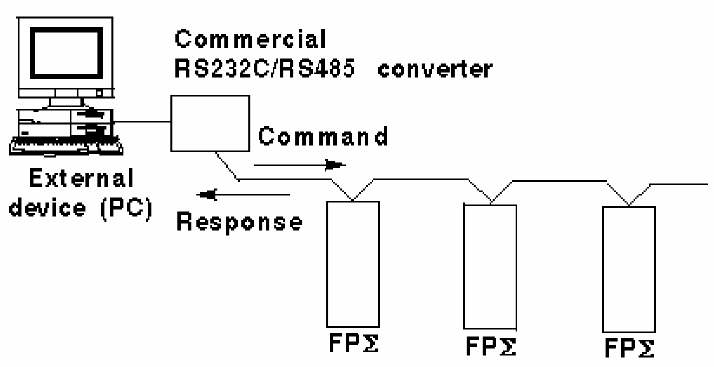
RS485 Response Time Control This changes the communication conditions based on the RS485 of the COM port or Tool port, in response to the contents specified by the character constant.

The port response time specified by the first keyword is delayed based on the contents specified by the second keyword. This instruction is used to delay the response time on the PLC side until the state is reached in which commands can be sent by an external device and responses can be received from the PLC.

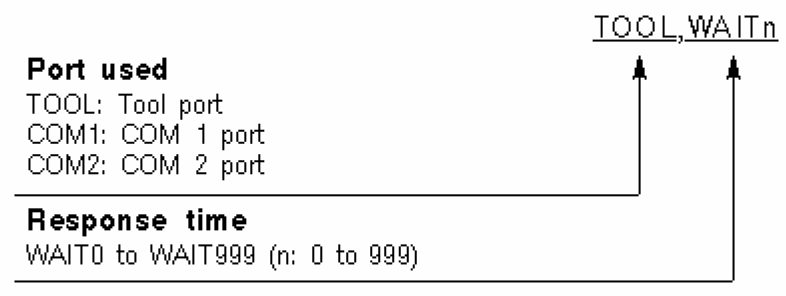
The first and second keywords are separated by a comma.

Usage example

When a commercial RS232C/RS485 converter is being used to carry out communication between a personal computer and the FPΣ, this instruction is used to return the PLC response after switching of the enable signal has been completed on the converter side.



Keyword setting



If the communication mode has been set to the computer link mode, the set time is the scan time x n (n: 0 to 999).

If the communication mode has been set to the PLC link mode, the set time is n µs (n: 0 to 999).

If n = 0, the delay time set by this instruction will be set to "None".

Precautions during Prog.

- This instruction is valid only if the setting on the controller side has been set to the computer link mode or the PLC link mode. It is invalid in the general communication mode.
- Executing this instruction does not change the settings in the system registers.
- We recommend using differential execution with this instruction.
- When the power supply to the PLC is off, the settings set by this instruction are cleared. (The set value will become 0.) If the mode is switched to the PROG. mode after the instruction has been executed, however, the settings will be retained.
- If a commercial RS232C/RS485 converter is being used in the PLC link mode, this instruction should be programmed in all of the stations (PLCs) connected to the link.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- any character other than a keyword is specified - no comma is between the first and second keywords - small letters of the alphabet are used to specify the keyword - no communication cassette has been installed when COM1 or COM2 has been set
R9008	%MX0.900.8	for an instant	

Example

In this example the function SYS1 is programmed in ladder diagram (LD).

POU Header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	ChangeResponseT	BOOL	FALSE	Changes response time of RS485

Body

When **ChangeResponseT** turns on, the response time for COM port 1 is delayed by 2µs.

LD

☞ **The values entered at s* will be right aligned automatically by the compiler.**

SYS2 Change System Register Settings for PC Link Area

Steps: 7

Description

While the PLC is in RUN mode, SYS2 changes the settings for the specified system registers. **s_Start** contains the new values for those system registers defined between **d_Start*** and **d_End***.

You can change the values in system registers 40 - 47 (with the FP-X also 50 - 57), PC link area.

Precautions during prog.

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- A value between 40 and 47 should be specified for **d_Start*** or **d_End***. Also, the values should always be specified in such a way that **d_Start*** ≤ **d_End***.
- The values of the system registers change, so a verification error may occur when the program is verified.

PLC types: **Availability of SYS2 (see page 935)**

Data types

Variable	Data type	Function
s_Start	INT, WORD	Contains new values for the system registers defined by remaining two variables.
d_Start*	constant	First system register (between 40-47) to receive new value.
d_End*	constant	Last system register (between 40-47) to receive new value.

Operands

For	Relay				T/C		Register			Constant
s_Start	-	-	-	-	-	-	DT	-	-	-
d_Start*	-	-	-	-	-	-	-	-	-	dec. or hex.
d_End*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007 R9008	%MX0.900.7 %MX0.900.8	permanently for an instant	- d1 > d2 - the specified value is outside the ranges specified for the various system registers setting values

Example In this example the function SYS2 is programmed in ladder diagram (LD).

DUT

A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

LINK_AREAS [DUT]

	Identifier	Type	Initial	Comment
0	RelayArea	INT	0	System register 40
1	RegisterArea	INT	0	System register 41
2	RelaySendStart	INT	0	System register 42
3	RelaySendSize	INT	0	System register 43
4	RegisterSendStat	INT	0	System register 44
5	RegisterSendSize	INT	0	System register 45

POU Header

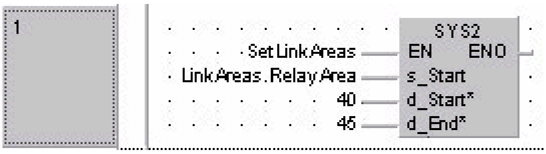
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	LinkAreas	LINK_AREAS	RelayArea := 64, RegisterArea := 128, RelaySendStart := 50, RelaySendSize := 10, RegisterSendStat := 100, RegisterSendSize := 10	
1	VAR	SetLinkAreas	BOOL	FALSE	

Body

Changes the values for the PC link area system registers 40 through 45 as defined in **LinkAreas** when **SetLinkAreas** turns on.

LD



Chapter 30

Special Instructions

F140_STC

Carry-flag set

Steps: 1

Description Special internal relay R9009 (carry-flag) goes ON if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g. F122_RCR (see page 566) and F123_RCL (see page 568) instructions).

PLC types: **Availability of F140_STC (see page 928)**

Example In this example the function F140_STC is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header

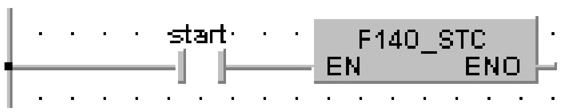
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function; result after a leading edge from start: carry-flag (R9009) will be set ON

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F140_STC();
END_IF;
```

F141_CLC Carry-flag reset

Steps: 1

Description Special internal relay R9009 (carry-flag) goes OFF if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g. F122_RCR (see page 566) and F123_RCL (see page 568) instructions).

PLC types: **Availability of F141_CLC (see page 928)**

Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header

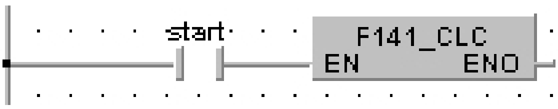
In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function; result after a leading edge from start: carry-flag (R9009) will be set OFF

Body

When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F141_CLC();
END_IF;
```

F148_ERR Self-diagnostic error set

Steps: 3

Description The error No. specified by **n*** is placed into special data register DT9000.

At the same time, the self-diagnostic error-flag R9000 is set and ERROR LED on the CPU is turned ON.
The contents of the error flag R9000 can be read and checked using Control FPWIN Pro (**Monitor** → **Display Special Relays and Registers** → **Basic Error Messages**).

The error No., special data register DT9000, can be read and checked using Control FPWIN Pro (**Monitor** → **Display Special Relays and Registers** → **Basic Error Messages**).

When **n*** = 0, the error is reset. (only for operation continue errors, **n*** = 200 to 299.)

The ERROR LED is turned OFF and the contents of special data register DT9000 are cleared with 0.

Error number areas:

When **n*** = 100 to 199, the operation is halted.

When **n*** = 200 to 299, the operation is continued.

PLC types: **Availability of F148_ERR (see page 929)**

Data types

Variable	Data type	Function
n*	constant	self-diagnostic error code number, range: 0 and 100 to 299

Operands

For	Relay				T/C		Register			Constant
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- **n** exceeds the limit.
R9008	%MX0.900.8	permanently	

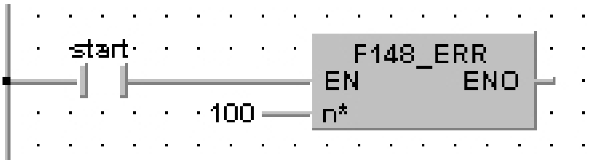
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    (* Sets the self-diagnostic error 100 *)
    (* The ERROR/ALARM LED of the PLC is on,
    and operation stops. *)
    F148_ERR(100);
END_IF;
```

F149_MSG

Message display

Steps: 13

Description This instruction is used for displaying the message on the FP Programmer II screen. After executing the **F149_MSG** instruction, you can see the message specified by **s** on the FP Programmer II screen.

When the **F149_MSG** instruction is executed, the message-flag R9026 is set and the message specified by **s** is set in special data registers DT9030 to DT9035. Once the message is set in special data registers, the message cannot be changed even if the **F149_MSG** instruction is executed again. You can clear the message with the FP Programmer II.

PLC types: **Availability of F149_MSG (see page 929)**

Data types

Variable	Data type	Function
s	STRING(12)	message to be displayed

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	-	-	-	character

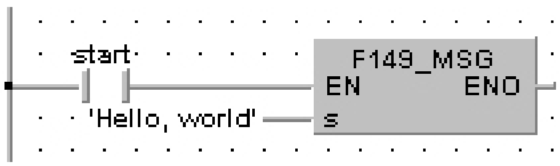
Example In this example the function F149_MSG is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU Header: In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body: When the variable **start** is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
    F149_MSG('Hello, world');
END_IF;
```

Chapter 31

Program Execution Control Functions

MC

Master control relay

Steps: 2

Description Executes the program between the master control relay **MC** and master control relay end MCE (see page 829) instructions of the same number **Num*** only if the trigger **EN** is in the ON-state

When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions is not executed.

A master control instruction **(MC** and **MCE)** pair may also be programmed in between another pair of master control instructions. This construction is called "nesting"**.**

The constant number **Num*** that must correspond to **MC** number, both of which delimit a "nested" program that is not executed.

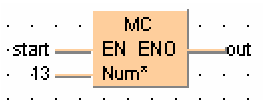
- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types: **Availability of MC (see page 933)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to MCE number, both of which delimit a "nested" program that is not executed

Example



MCE

Master control relay end

Steps: 2

Description Executes the program between the master control relay MC (see page 828) and master control relay end **MCE** instructions of the same number **Num*** only if the trigger **EN** is in the ON-state.

When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions is not executed.

A master control instruction **(MC** and **MCE)** pair may also be programmed in between another pair of master control instructions. This construction is called "nesting".

The constant number **Num*** that must correspond to **MC** number, both of which delimit a "nested" program that is not executed.

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types: **Availability of MCE (see page 933)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to MC number, both of which delimit a "nested" program that is not executed

Example In this example, the progamming language Instruction List (IL) is used.

IL

```
LD      start   (* EN = start; Starting signal for the MC/MCE function. *)
MC      1       (* 1 = Num* *)
                (* ... *)
                (* Execute or execute not this program part. *)
                (* ... *)
MCE     1       (* 1 = Num* *)
```

JP — Jump to label

Steps: 2

Description The JP (Jump to Label) instruction skips to the Label (LBL (see page 832)) function that has the same number **Num*** as the **JP** function when the predetermined trigger **EN** is in the ON-state.

The **JP** function will skip all instructions between a **JP** and an **LBL** of the same number. When the **JP** instruction is executed, the execution time of the skipped instructions is not included in the scan time. Two or more **JP** functions with the same number **Num*** can be used in a program. However, no two **LBL** instructions may be identically numbered. **LBL** instructions are specified as destinations of **JP**, LOOP (see page 831) and F19_SJP instructions.

One **JP** and **LBL** instruction pair can be programmed between another pair. This construction is called nesting.

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types: **Availability of JP (see page 933)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to LBL number, this "nested" program is jumped over

Example In this example, the programming language Instruction List (IL) is used.

IL

```
LD    start    (* EN = start; Starting signal for the JP function. *)
JP    1        (* Num* = 1 (Address of Label) *)
```

LOOP

Loop to label **Steps: 4**

Description LOOP (Loop to Label) instruction skips to the LBL (see page 832) instruction with the same number **Num*** as the **LOOP** instruction and repeats execution of what follows until the data of a specified operand becomes "0".

The **LBL** instructions are specified as destination of the **LOOP** instruction. It is not possible to specify two or more **LBL** instructions with the same number **Num*** within a program. If the set value **s** in the data area is "0" from the beginning, the **LOOP** instruction is not executed (ignored).

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types: **Availability of LOOP (see page 933)**

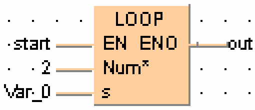
Data types

Variable	Data type	Function
s	INT, WORD	Set value
Num*	constant	Constant number that must correspond to LBL number, this "nested" program is looped until the variable at **s** reaches 0

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

Example



LBL

Label for the JP- and LOOP-instruction

Steps: 1

Description The **LBL** (Label for the JP and LOOP) instruction skips to the **LBL** instruction with the same number **Num*** as the JUMP (see page 830) instruction if the predetermined trigger **EN** is in the ON-state.

Skips to the **LBL** instruction with the same number **Num*** as the (see page 831)LOOP instruction and repeats execution of what follows until the data of a specified operand becomes "0".

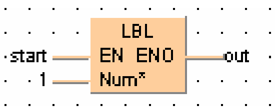
- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types: **Availability of LBL (see page 933)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to JP, LOOP or F19 label number

Example



ICTL Interrupt Control

Description The **ICTL** instruction sets all interrupts to enable or disable. Each time the ICTL instruction is executed, it is possible to set parameters like the type and validity of interrupt programs. Settings can be specified by **s1** and **s2**.

- **s1** 16-bit equivalent constant or 16-bit area for interrupt control setting
- **s2** 16-bit equivalent constant or 16-bit area for interrupt trigger condition setting

The number of interrupt programs available is:

- **16** interrupt module initiated interrupt programs (INT 0 to INT 15)
- **8** advanced module (special modules, like positioning,...) initiated interrupt programs (INT 16 to INT 23)

Be sure to use ICTL instructions so that they are executed once at the leading edge of the ICTL trigger using the DF instruction.

Two or more ICTL instructions can have the same trigger.

Bit	15 .. 8	7 .. 0
s1 16#	**Selection of control function** **00**: Interrupt "enable/disable" control **01**: Interrupt trigger reset control	**Interrupt type selection** **00**: Interrupt module initiated interrupt (INT 0-15) **01**: Advanced module initiated interrupt (INT 16-23) **02**: Time-initiated interrupt (INT 24)
s2 2#	Bit 0: **0** Interrupt program 0 disabled Bit 0: **1** Interrupt program 0 enabled Bit 1: **0** Interrupt program 1 disabled ... Bit 15: **1** Interrupt program 15 enabled Example: s2 = 2#0000000000001010	

- **The current enable/disable status of each interrupt module initiated interrupt can be checked by monitoring the special data register DT90025.**
- **The current enable/disable status of each non-interrupt module initiated interrupt can be checked by monitoring the special data register DT90026.**
- **The current interrupt interval of the time-interrupt can be checked by monitoring the special data register DT90027.**
- **If a program is written into an interrupt task, the interrupt concerned will be enabled automatically during the initialization routine when starting the program.**
- **With the ICTL instruction an interrupt task can be enabled or disabled by the program.**

PLC types: **Availability of ICTL** **(see page 932)**

Data types

Variable	Data type	Function
s1	INT, WORD	Interrupt control data setting
s2	INT, WORD	Interrupt condition setting

Operands

For	Relay				T/C		Register			Constant
s1, s2	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

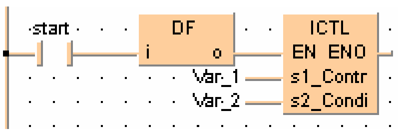
Example In this example, the same POU header is used for all programming languages. Please refer to the online help for an example using IL (instruction list).

POU Header In the POU header, all input and output variables are declared that are used for programming this function.

```
  Class  Identifier Type   Initial  Comment
0 VAR    Var_1      WORD   16#0002  Input parameter s1
1 VAR    Var_2      WORD   10       Input parameter s2
2 VAR    start      BOOL   FALSE    Enable signal
```

Body The interval for executing INT 24 program is specified as 100 ms (10ms time base selected) when the leading edge of start is detected.

LD



Chapter 32

Appendix Programming Information

32.1 FP TOOL Library

The NC TOOL Library contains advanced address, information and copy functions available for all PLCs to make programming easier. Below please find a selection of these functions. For more detailed information and examples, see Online help.

Program can be adversely effected!
These functions can cause substantial problems by accessing incorrect memory areas if they are not used in the sense they were meant for. Especially other parts of the program can be adversely effected.

Name	Function
Addresses Instructions	
Adr_Of_Var	Address of a variable at the input/output of a FP function
AdrLast_Of_Var	Address of a variable at the input/output of a FP function
Adr_Of_VarOffs	Address of a variable with offset at the input/output of a FP function
Pointers Instructions	
AdrDT_Of_Offs	DT address from the address offset for the input/output of a FP function
AdrFL_Of_Offs_I	FL address from the address offset for the input/output of a FP function
AreaOffs_OfVar	Yields memory area and address offset of a variable (with Enable)
Is_AreaDT	Yields TRUE if the memory area of a variable is a DT area (with Enable)
Is_AreaFL	Yields TRUE if the memory area of a variable is a FL area (with Enable)
AreaOffs_ToVar	Copies the content of an address specified by memory area and address offset to a variable (with Enable)
Var_ToAreaOffs	Copies the value of a variable to an address specified by memory area and address offset to a variable (with Enable)
Size Information Instructions	
Size_Of_Var	Yields the size of a variable in words (with Enable)
Elem_OfArray1D	Yields the number of elements in an array (with Enable)
Elem_OfArray2D	Yields the number of elements of the 1st and 2nd dimension of an array (with Enable)
Elem_OfArray3D	Yields the number of elements of the 1st, 2nd and 3rd dimension of an array (with Enable)
Additional Copy Instructions	
	This functions are allowed to be compiled because of the down-compatibility to lower versions but cannot be selected in the OP/FUN/FB dialog anymore.
Any16_ToBool16	Replaced from version 5 onwards by the function INT_TO_BOOL16 or WORD_TO_BOOL16.
Bool16_ToAny16	Replaced from version 5 onwards by the function BOOL16_TO_INT or BOOL16_TO_WORD.
Any32_ToBool32	Replaced from version 5 onwards by the function DINT_TO_BOOL32 or DWORD_TO_BOOL32.

Name	Function
Bool32_ToAny32	Replaced from version 5 onwards by the function BOOL32_TO_DINT or BOOL32_TO_DWORD.
Any16_ToSpecDT	Replaced from version 5 onwards by the function INT_TO_SDT or WORD_TO_SDT.
SpecDT_ToAny16	Replaced from version 5 onwards by the function SDT_TO_INT or SDT_TO_WORD.
Any32_ToSpecDT	Replaced from version 5 onwards by the function DINT_TO_SDDT or DWORD_TO_SDDT.
SpecDT_ToAny32	Replaced from version 5 onwards by the function SDDT_TO_DINT or SDDT_TO_DWORD.
SFC Control Instructions	
Instructions that control all SFC programs simultaneously	
StartStopAllSfcs	Stops and restarts all Sequential Function Chart (SFC) programs
StartStopAllSfcsAndInitData	
A function that reveals the status of all SFCs	
AllSfcsStopped	Indicates whether all Sequential Function Chart (SFC) programs were stopped
Instructions that control a specific SFC	
StartStopSfc	Stops and restarts a specific Sequential Function Chart (SFC) program
StartStopSfcAndInitData	
ControlSfc	Controls a specific Sequential Function Chart (SFC) program
ControlSfcAndData	
ActivateStepsOfStoppedSfc	Continues a Sequential Function Chart (SFC) program that has been stopped
Instructions that reveal the statuses of a specific SFC	
SfcStopped	Indicates whether a specific Sequential Function Chart (SFC) program was stopped
SfcTransitionsInhibited	Indicates whether the transitions of a specific Sequential Function Chart (SFC) program are locked
SfcRunning	Indicates whether a certain Sequential Function Chart (SFC) program is running
SfcOutputsReset	Indicates whether the inputs of a specific Sequential Function Chart (SFC) program have been reset

32.2 Floating Point Instructions

The floating point F/P instructions are designed specifically for applications that require variables of the data type REAL. Most of these can be replaced by the more flexible IEC commands. By doing so you will reduce the number of commands with which you need to be familiar.

The following floating point instructions are described in detail in this manual because they are not easily duplicated with IEC instructions: F327_INT (see page 651), F328_DINT (see page 653), F333_FINT (see page 655), F334_FRINT (see page 657), F335_FSIGN (see page 659), F337_RAD (see page 661) and F338_DEG (see page 663).

For details and examples on the other floating point instructions, see Online help. For quick reference, please refer to the table below.

Name	Function	Equivalent IEC function
F309_FMV	Constant floating point data move	E_MOVE
F310_FADD	Floating point data add	E_ADD
F311_FSUB	Floating point data subtract	E_SUB
F312_FMUL	Floating point data multiply	E_MUL
F313_FDIV	Floating point data divide	E_DIV
F314_FSIN	Floating point Sine operation	E_SIN
F315_FCOS	Floating point Cosine operation	E_COS
F316_FTAN	Floating point Tangent operation	E_TAN
F317_ASIN	Floating point Arcsine operation	E_ASIN
F318_ACOS	Floating point Arccosine operation	E_ACOS
F319_ATAN	Floating point Arctangent operation	E_ATAN
F320_LN	Floating point data natural logarithm	E_LN
F321_EXP	Floating point data exponent	E_EXP
F322_LOG	Floating point data logarithm	E_LOG
F323_PWR	Floating point data power	E_EXPT
F324_FSQR	Floating point data square root	E_SQRT
F325_FLT	16-bit integer → Floating point data	E_INT_TO_REAL
F326_DFLT	32-bit integer → Floating point data	E_DINT_TO_REAL
F329_FIX	Floating point data → 16-bit integer Rounding the first decimal point down	E_TRUNC_TO_INT
F330_DFIX	Floating point data → 32-bit integer Rounding the first decimal point down	E_TRUNC_TO_DINT
F331_ROFF	Floating point data → 16-bit integer Rounding the first decimal point off	E_REAL_TO_INT
F332_DROFF	Floating point data → 32-bit integer Rounding the first decimal point off	E_REAL_TO_DINT
F336_FABS	Floating point data absolute	E_ABS
F345_FCMP	Floating point data compare	E_GE, E_GT, E_EQ, E_LE, E_LT, E_NE
F347_FLIMT	Floating point data upper and lower limit control	E_LIMIT

32.3 Relays, Memory Areas and Constants

32.3.1 Relays, Memory Areas and Constants for FP-Sigma

Item		Number of points	Memory area available for use		Function
			F/P	IEC	
Relay	**External input relay (see note 1)**	1184	X0-X73F	%IX0.0- %IX73.15	Turns on or off based on external input.
	External output relay (see note 1)	1184	Y0-Y73F	%QX0.0- %QX73.15	Externally outputs on or off state.
	Internal relay (see note 2)	1568	R0-R97F	%MX0.0- %MX0.97.15	Turns on or off only within a program.
	Link relay (see note 2)	1024	L0-L63F	%MX7.0.0- %MX7.63.15	Shared relay used for PLC link.
	Timer (see notes 2 and 3)	1024	T0-T1007/ C1008-C1023	%MX1.0- %MX1.1007/ %MX2.1008- %MX2.1023	Goes on when the timer reaches the specified time. Corresponds to the timer number.
	Counter **(see notes 2 and 3)**	1024	C1008-C1023/ T0-T1007	%MX2.1008- %MX2.1023/ %MX1.0- %MX1.1007	Goes on when the timer increments. Corresponds to the timer number.
	Special internal relay	176	R9000-R910F	%MX0.900.0- %MX0.910.15	Turns on or off based on specific conditions. Used as a flag.
Memory area (words)	**External input relay (see note 1)**	74 words	WX0-WX73	%IW0- %IW73	Code for specifying 16 external input points as one word (16 bits) of data.
	External output relay (see note 1)	74 words	WY0-WY73	%QW0- %QW73	Code for specifying 16 external output points as one word (16 bits) of data.
	Internal relay (see note 2)	98 words	WR0-WR97	%MW0.0- %MW0.97	Code for specifying 16 internal relay points as one word (16 bits) of data.
	Link relay	64 words	WL0-WL63	%MW7.0- %MW7.63	Code for specifying 16 link relay points as one word (16 bits) of data.
	Data register (see note 2)	32765 words	DT0-DT32764	%MW5.0- %MW5.32764	Data memory used in a program. Data is handled in 16-bit units (one word).
	Link data register (see note 2)	128 words	LD0-LD127	%MW8.0- %MW8.127	A shared data memory which is used within the PLC link. Data is handled in 16-bit units (one word).
	Timer/counter set value area (see note 2)	1024 words	SV0-SV1023	%MW3.0- %MW3.1023	Data memory for storing a target value of a timer and an initial value of a counter. Stores by timer/counter number.

Item		Number of points	Memory area available for use		Function
			F/P	IEC	
	External input relay (see note 1)	1184	X0-X73F	%IX0.0-%IX73.15	Turns on or off based on external input.
	Timer/counter elapsed value area (see note 2)	1024 words	EV0-EV1023	%MW4.0-%MW4.1023	Data memory for storing the elapsed value during operation of a timer/counter. Stores by timer/counter number.
	Special data register	260 words	DT90000-DT90259	%MW5.90000-%MW5.90259	Data memory for storing specific data. Various settings and error codes are stored.
	Index register	14 words	I0-ID	%MW6.0-%MW6.14	Can be used as an address of memory area and constants modifier.
Memory area (double word) (see note 4)	**External input relay (see note 1)**	16 double words	DWX0-DWX30	%ID0-%ID30	Code for specifying 32 external input points as a double word (32 bits) of data.
	External input relay (see note 1)	37 double words	DWX0-DWX73	%ID0-%ID73	Code for specifying 32 external input points as a double word (32 bits) of data.
	External output relay (see note 1)	16 double words	DWY0-DWY30	%QD0-%QD30	Code for specifying 32 external output points as double word (32 bits) of data.
	External output relay (see note 1)	37 double words	DWY0-DWY73	%QD0-%QD73	Code for specifying 32 external output points as double word (32 bits) of data.
	Internal relay (see note 2)	49 double words	DWR0-DWR96	%MD0.0-%MD0.96	Code for specifying 32 internal relay points as double word (32 bits) of data.
	Link relay	32 double words	DWL0-DWL62	%MD7.0-%MD7.62	Code for specifying 32 link relay points as double word (32 bits) of data.
	Data register (see note 2)	16382 double words	DDT0-DDT32763	%MD5.0-%MD5.32763	Data memory used in a program. Data is handled in 32-bit units (double word).
	Link data register (see note 2)	64 double words	DLD0-DLD126	%MD8.0-%MD8.126	A shared data memory which is used within the PLC link. Data is handled in 32-bit units (double word).
	Timer/counter set value area (see note 2)	512 double words	DSV0-DSV1022	%MD3.0-%MD3.1022	Data memory for storing a target value of a timer and an initial value of a counter. Stores by timer/counter number.
	Timer/counter elapsed value area (see note 2)	512 double words	DEV0-DEV1022	%MD4.0-%MD4.1022	Data memory for storing the elapsed value during operation of a timer/counter. Stores by timer/counter number.
	Special data register	130 double words	DDT90000-DDT90258	%MD5.90000-%MD5.90258	Data memory for storing specific data. Various settings and error codes are stored.

Item		Number of points	Memory area available for use		Function
			F/P	IEC	
	External input relay (see note 1)	1184	X0-X73F	%IX0.0-%IX73.15	Turns on or off based on external input.
	Index register	7 double words	DI0-DIC	%MD6.0-%MD6.13	Can be used as an address of memory area and constants modifier.

Item		Range available for use	
		F/P	IEC
Con-stant	**Decimal con-stants (integer type)**	K-32768 to K32767 (for 16-bit operation)	-32768 to 32767 (for 16-bit operation)
		K-2147483648 to K2147483647 (for 32-bit operation)	-2147483648 to 2147483647 (for 32-bit operation)
	Hexadecimal constants	H0 to HFFFF (for 16-bit operation)	16#0 to 16#FFFF (for 16-bit operation)
		H0 to HFFFFFFFF (for 32-bit operation)	16#0 to 16#FFFFFFFF (for 32-bit operation)
	Decimal con-stants (monore-fined real number)	F-3.402823 10^{38} to F-1.175494 10^{-38}	-3.402823E38 to -1.17549410E-38
		F1.175494 10^{-38} to F3.402823 10^{38}	1.17549410E-38 to 3.402823E38

NOTES

1. **The number of points noted above is the number reserved as the calculation memory. The actual number of points available for use is determined by the hardware configuration.**

2. **If no battery is used, only the fixed area is backed up (counters 16 points: C1008 to C1023, internal relays 128 points: R900 to R97F, data registers: DT32710 to DT32764). When the optional battery is used, data can be backed up. Areas to be held and not held can be specified using the system registers.**

3. **The points for the timer and counter can be changed by the setting of system register 5. The number given in the table are the numbers when system register 5 is at its default setting.**

32.3.2 Relays, Memory Areas and Constants for FP-X

Relays

Item	Number of points available	Function
External input (X) **(see note 1)**	1760 points (X0 to X109F)	Turns on or off based on external input.
External output (Y) **(see note 1)**	1760 points (Y0 to Y109F)	Externally outputs on or off state
Internal relay (R) **(see note 2)**	4096 points (R0 to R255F)	Relay which turns on or off only within program.
Link relay (L) **(see note 2)**	2048 points (L0 to L127F)	This relay is a shared relay used for PLC link.
Timer (T) **(see note 2)**	1024 points (T0 to T1007/C1008 to C1023) (see note 3)	This goes on when the timer reaches the specified time. It corresponds to the timer number.
Counter (C) **(see note 2)**		This goes on when the timer increments. It corresponds to the timer number.
Special internal relay (R)	192 points (R9000 to R911F)	Relay which turns on or off based on specific conditions and is used as a flag.

Memory areas

Item	Range of memory area available C14, C30/C60		Function
External input (WX) **(see note 1)**	110 words (WX0 to WX109)		Code for specifying 16 external input points as one word (16 bits) of data.
External output (WY) **(see note 1)**	110 words (WY0 to WY109)		Code for specifying 16 external output points as one word (16 bits) of data.
Internal relay (WR) **(see note 2)**	256 words (WR0 to WR255)		Code for specifying 16 internal relay points as one word (16 bits) of data.
Link relay (WL)	128 words (WL0 to WL127)		Code for specifying 16 link relay points as one word (16 bits) of data.
Data register (DT) **(see note 2)**	12285 words (DT0 to DT12284)	32765 words (DT0 to DT32764)	Data memory used in program. Data is handled in 16-bit units (one word).
Link register (LD) **(see note 2)**	256 words (LD0 to LD255)		This is a shared data memory which is used within the PLC link. Data is handled in 16-bit units (one word).
Timer/Counter set value area (SV) **(see note 2)**	1024 words (SV0 to SV1023)		Data memory for storing a target value of a timer and setting value of a counter. Stores by timer/counter number
Timer/Counter elapsed value area (EV) **(see note 2)**	1024 words (EV0 to EV1023)		Data memory for storing the elapsed value during operation of a timer/counter. Stores by timer/counter number.

Item	Range of memory area available C14, C30/C60	Function
Special data register (DT)	374 words (DT90000 to DT90373)	Data memory for storing specific data. Various settings and error codes are stored.
Index register (I)	14 words (I0 to ID)	Register can be used as an address of memory area and constants modifier.

Constants

Item	Range of memory area available
Decimal constants (Integer type) (K)	K-32, 768 to K32, 767 (for 16-bit operation)
	K-2, 147, 483, 648 to K2, 147, 483, 647 (for 32-bit operation)
Hexadecimal constants (H)	H0 to HFFFF (for 16-bit operation)
	H0 to HFFFFFFFF (for 32-bit operation)
Decimal constants (Floating point type) (F)	F-1.175494 x 10^{-38} to F-3.402823 x 10^{38}
	F 1.175494 x 10^{-38} to F 3.402823 x 10^{38}

NOTES

1. **The number of points noted is the number reserved for the calculation memory. The actual number of points available for use depends on the hardware configuration.**

2. **If no batter is used, only the fixed area is backed up (counters 16 points: C1008 to C1023, internal relays 128 points: R2470 to R255F, data registers 55 words, C14: DT12230 to DT12284, C30/C60: DT32710 to DT32764). Writing is available up to 10000 times. When the optional battery is used, all area can be backed up. Areas to be held and not held can be specified using the system registers. If an area is held and the backup battery is not installed, the data may be corrupted as it is not cleared to 0 when the power is turned on. If the battery goes dead, the data in the hold area may likewise be corrupted.**

3. **The points for the timer and counter can be changed via system register 5. The numbers given in the table are the default settings for system register 5.**

32.4 System Registers

System registers are used to set values (parameters) which determine operation ranges and functions used. Set values based on the use and specifications of your program. There is no need to set system registers for functions which will not be used.

32.4.1 Precautions When Setting System Registers

System register settings are effective from the time they are set.

However, MEWNET-W0 PLC link settings, input settings, Tool and COM port communication settings become effective when the mode is changed from PROG to RUN. With regard to the modem connection setting, when the power is turned off and on or when the mode is changed from PROG to RUN, the controller sends a command to the modem which enables it for reception.

When the initialization operation is performed, all system register values (parameters) set will be initialized.

32.4.2 Types of System Registers

Hold/non-hold type settings (system registers 5 to 8, 10, 12 and 14)

The values for the timer and counter can be specified by using system register no. 5 to specify the first number of the counter. System registers no. 6 to no. 8, no. 10, no. 12, and no. 14 are used to specify the area to be held when a battery is used.

Operation mode settings for errors (system registers 4, 20, 23 and 26)

Set the operation mode when errors such as battery error, duplicated use of output, I/O verification error and operation error occur.

Time settings (system registers 31 to 34)

Set time-out error detection time and constant scan time.

MEWNET-W0 PLC link settings (system registers 40 to 45, and 47)

These settings are for using link relays and link registers in MEWNET-W0 PLC link communication. Note that PLC link is not the default setting.

Input settings (system register 400 to 403)

When using the high-speed counter function, pulse catch function or interrupt function, set the operation mode and the input number to be used for the function.

Tool and COM port communication settings (system registers 410 to 419)

Set these registers when the Tool port, and COM1 and COM2 ports are to be used for computer link, general-purpose serial communication, PLC link, and modem communication. Note that the default setting is computer link mode.

◆ NOTE

The default setting is computer link mode. With FPG-COM4, the transmission speed setting for the RS485 port (COM2) set in the system registers and using the DIP switch on the communication cassette must be the same.

32.4.3 Checking and Changing System Registers

◆ PROCEDURE

1. **Double-click “PLC” in the project navigator**
2. **Double-click “System Registers”**
3. **To change a set value, write the new value as indicated in the system register table**
4. **Online → Online mode**
5. **Online → Download Program Code and PLC Configuration**
   This downloads the project and system registers.

To download system registers only: **Online → PLC Configuration**, select “System Registers”, choose [Download to PLC].

32.4.4 Table of System Registers for FP-Sigma

Item	No.	Name	Default value	Descriptions	
Hold/ Non-hold 1	**5**	Starting number setting for counter	1008	0 to 1024	
	6	Hold type area starting number setting for timer and counter	1008	0 to 1024	• These settings are effective if the optional backup battery is installed • If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.
	7	Hold type area starting number setting for internal relays	90	0 to 98	
	8	Hold type area starting number setting for data registers	32710	0 to 32765	
	14	Hold or non-hold setting for step ladder process	Non-hold	Hold/Non-hold	
Hold/ Non-hold 2	**10**	Hold type area starting number for PLC link relays	64	0 to 64	
	12	Hold type area starting number for PLC link registers	128	0 to 128	
Action on error	**20**	Disable or enable setting for duplicated output	Yes	Fixed	
	23	Operation setting when an I/O verification error occurs	Stop	Stop/Continuation of operation	
	26	Operation setting when an operation error occurs	Stop	Stop/Continuation of operation	
	4	Alarm Battery Error (Operating setting when battery error occurs)	Disabled	Disabled: When a battery error occurs, a self-diagnostic error is not issued and the ERROR/ALARM LED does not light. Enabled: When a battery error occurs, a self-diagnostic error is issued and the ERROR/ALARM LED lights.	
Time setting	**31**	Wait time setting for multi-frame communication	6500.0 ms	10 to 81900 ms	
	34	Constant value settings for scan time	0.0 ms	0: Normal scan 0 to 350 ms: Scans once each specified time interval.	
PLC link setting	**40**	Range of link relays used for PLC link	0	0 to 64 words	
	41	Range of link data registers used for PLC link	0	0 to 128 words	
	42	Starting number for link relay transmission	0	0 to 63	
	43	Link relay transmission size	0	0 to 64 words	
	44	Starting number for link data register transmission	0	0 to 127	
	45	Link data register transmission size	0	0 to 127 words	
	47	Maximum unit number setting for MEWNET-W0 PLC link	16	1 to 16	

Item	No.	Name	Default value	Descriptions	
High-speed coun-ter	**400**	High-speed counter operation mode settings (X0 to X2) (see notes 1, 2, 4)	CH0: Do not set input X0 as high-speed counter	CH0	Do not set input X0 as high-speed counter. Two-phase input (X0, X1) Two-phase input (X0, X1), reset input (X2) Incremental input (X0) Incremental input (X0), reset input (X2) Decremental input (X0) Decremental input (X0), reset input (X2) Incremental/decremental input (X0, X1) Incremental/decremental input (X0, X1), reset input (X2) Incremental/decremental control input (X0, X1) Incremental/decremental control input (X0, X1), reset input (X2)
			CH1: Do not set input X1 as high-speed counter	CH1	Do not set input X1 as high-speed counter. Incremental input (X1) Incremental input (X1), reset input (X2) Decremental input (X1) Decremental input (X1), reset input (X2)
	401	High-speed counter operation mode settings (X3 to X5) (see notes 1, 2, 4)	CH2: Do not set input X3 as high-speed counter	CH2	Do not set input X3 as high-speed counter. Two-phase input (X3, X4) Two-phase input (X3, X4), reset input (X5) Incremental input (X3) Incremental input (X3), reset input (X5) Decremental input (X3) Decremental input (X3), reset input (X5) Incremental/decremental input (X3, X4) Incremental/decremental input (X3, X4), reset input (X5) Incremental/decremental control input (X3, X4) Incremental/decremental control input (X3, X4), reset input (X5)
			CH3: Do not set input X4 as high-speed counter	CH3	Do not set input X4 as high-speed counter. Incremental input (X4) Incremental input (X4), reset input (X5) Decremental input (X4) Decremental input (X4), reset input (X5)
Inter-rupt input	**402**	Pulse catch input settings (see notes 3, 4)	Not set	X0 X1 X2 X3 X4 X5 X6 X7 Specify the input contacts used as pulse catch input.	

Item	No.	Name	Default value	Descriptions
	403	Interrupt input settings (see notes 3, 4)	Not set	X0 X1 X2 X3 X4 X5 X6 X7 Specify the input contacts used as interrupt input. X0 X1 X2 X3 X4 X5 X6 X7 Specify the effective interrupt edge. (When set: on → off is valid.)
Tool port setting	**410**	Unit no. setting	1	1 to 99
	412	Selection of modem connection	Disabled	Enabled/Disabled
	413	Communication format setting	Data length: 8 bits, Parity check: “with, odd” Stop bit: 1 bit	Enter the settings for the various items. Data length: 7bits/8bits Parity check: none/with odd/with even Stop bit: 1bit/2bits
	415	Communication speed (Baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
COM.1 port setting	**410**	Unit no. setting	1	0 to 99
	412	Communication mode setting	MEWTO-COL-COM Slave	MEWTOCOL-COM Slave Program controlled communication PLC Link
		Selection of modem connection	Disabled	Enabled/Disabled
	413	Communication format setting	Data length: 8 bits, Parity check: “with, odd” Stop bit: 1 bit	Enter the settings for the various items. Data length: 7bits/8bits Parity check: none/with odd/with even Stop bit: 1bit/2bits The following setting is valid only when the communication mode specified by system register 412 has been set to "Program controlled communication". Terminator: CR/CR+LF/None Header: STX not exist/STX exist
	415	Communication speed (Baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
	416	Starting address for received buffer of program controlled communication mode	0	0 to 32764
	417	Buffer capacity setting for data received of program controlled communication mode	2048	0 to 2048
COM.2	**411**	Unit no. setting	1	1 to 99

Item	No.	Name	Default value	Descriptions
port setting	**412**	Communication mode setting	MEWTO-COL-COM Slave	MEWTOCOL-COM Slave Program controlled communication
		Selection of modem connection	Disabled	Enabled/Disabled
	414	Communication format setting	Data length: 8 bits, Parity check: “with, odd” Stop bit: 1 bit	Enter the settings for the various items. Data length: 7bits/8bits Parity check: none/with odd/with even Stop bit: 1bit/2bits The following setting is valid only when the communication mode specified by system register 412 has been set to "Program controlled communication". Terminator: CR/CR+LF/None Header: STX not exist/STX exist
	415	Communication speed (Baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
	418	Starting address for received buffer of program controlled communication mode	2048	0 to 32764
	419	Buffer capacity setting for data received of program controlled communication mode	2048	0 to 2048

NOTES

1. **If the operation mode is set to two-phase, incremental/decremental, or incremental/decremental control, the setting for CH1 is invalid in part 2 of system register 400 and the setting for CH3 is invalid in part 2 of system register 401.**

2. **If reset input settings overlap, the CH1 setting takes precedence in system register 400 and the CH3 setting takes precedence in system register 401.**

3. **The settings for pulse catch and interrupt input can only be specified in system registers 402 and 403.**

4. **If system registers 400 to 403 have been set simultaneously for the same input relay, the following precedence order is effective:**
   1. High-speed counter
   2. Pulse catch
   3. Interrupt input.
   This means, the counter keeps counting even after an interrupt. However, the response time of the high-speed counter is about 100 µs, that of the pulse catch input is about 200 µs. Therefore, the interrupt is recognized quickly enough.

5. **The communication format in a PLC link is fixed at the following settings: Data length 8 bits, odd parity, stop bit 1. The communication speed (baud rate) is fixed at 115,200 bps. Other system register settings will be ignored. With FPG-COM4, the transmission speed setting for the RS485 port (COM2) set in the system registers and using the DIP switch on the communication cassette must be the same.**

32.4.5 Table of System Registers for FP-X

Hold/Non-hold

Address	Name	Default value	Description	
5	Starting number setting for counter	1008	0 to 1024	These settings are effective if the optional backup battery is installed. If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.
6	Hold type area starting number setting for timer and counter	1008	0 to 1024	
7	Hold type area starting number setting for internal relays	248	0 to 256	
8	Hold type area starting number setting for data registers	C14: 12230 C30, C60: 32710	0 to 32765	
14	Hold or non-hold setting for step ladder process	Non-hold	Hold/ Non-hold	
4	Previous value is held for a leading edge detection instruction (DF instrucion) with MC	Hold	Hold/ Non-hold	
10	Hold type area starting number for PLC W0-0 link relays	64	0 to 64	
11	Hold type area starting number for PLC W0-1 link relays	128	64 to 128	
12	Hold type area starting number for PLC W0-0 link registers	128	0 to 128	
13	Hold type area starting number for PLC W0-1 link registers	256	128 to 256	

Action on error

Address	Name	Default value	Description
20	Disable or enable setting for duplicated output	Disabled	Disabled/Enabled
23	Operation setting when an I/O verification error occurs	Stop	Stop/Continuation of operation
26	Operation setting when an operation error occurs	Stop	Stop/Continuation of operation
4	Alarm battery error (Operating setting when battery error occurs)	Disabled	Disabled: When a battery error occurs, a self-diagnostic error is not issued and the ERROR/ALARM LED does not flash.

Address	Name	Default value	Description
			Enabled: When a battery error occurs, a self-diagnostic error is issued and the ERROR/ALARM LED flashes.

Time setting

Address	Name	Default value	Description
31	Wait time setting for multi-frame communication	6500.0ms	10 to 81900ms
32	Timeout setting for SEND/RECV, RMRD/RMWT commands	10000.0ms	10 to 81900ms
34	Constant value settings for scan time	Normal scan	0: Normal scan 0 to 350 ms: Scans once each specified time interval

PLC W0-0 setting

Address	Name	Default value	Description
40	Range of link relays used for PLC link	0	0 to 64 words
41	Range of link data registers used for PLC link	0	0 to 128 words
42	Starting number for link relay transmission	0	0 to 63
43	Link relay transmission size	0	0 to 64 words
44	Starting number for link data register transmission	0	0 to 127
45	Link data register transmission size	0	0 to 127 words
46	PLC link switch flag	Normal	Normal/reverse
47	Maximum unit number setting for MEWNET-W0 PLC link	16	1 to 16

PLC W0-1 setting

Address	Name	Default value	Description
50	Range of link relays used for PLC link	0	0 to 64 words
51	Range of link data registers used for PLC link	0	0 to 128 words
52	Starting number for link relay transmission	64	64 to 127
53	Link relay transmission size	0	0 to 64 words
54	Starting number for link data register transmission	128	128 to 255
55	Link data register transmission size	0	0 to 127 words
57	Maximum unit number setting for MEWNET-W0 PC(PLC) link	16	1 to 16

Pulse I/O cassette (AFPX-PLS)

Address	**Name**	**Default value**	**Description**	
400	High-speed counter operation mode settings (X100 to X102)	CH8: Do not set input X100 as high-speed counter	CH8	Do not set input X100 as high-speed counter. Two-phase input (X100, X101) Two-phase input (X100, X101), Reset input (X102) Incremental input (X100) Incremental input (X100), Reset input (X102) Decremental input (X100) Decremental input (X100), Reset input (X102) Incremental/decremental input (X100, X101) Incremental/decremental input (X100, X101), Reset input (X102) Incremental/decremental control input (X100, X101) Incremental/decremental control input (X100, X101), Reset input (X102)
		CH9: Do not set input X101 as high-speed counter	CH9	Do not set input X101 as high-speed counter. Incremental input (X101) Incremental input (X101), Reset input (X102) Decremental input (X101) Decremental input (X101), Reset input (X102)
	Pulse output operation mode	CH0: Use output as normal output.	CH0	Use output as normal output. Use output Y100 to Y102 as pulse output. Use output Y100 as PWM output.
401	High-speed counter operation mode settings (X200 to X202)	CHA: Do not set input X200 as high-speed counter	CHA	Do not set input X200 as high-speed counter. Two-phase input (X200, X201) Two-phase input (X200, X201), Reset input (X202) Incremental input (X200) Incremental input (X200), Reset input (X202) Decremental input (X202) Decremental input (X202), Reset input (X202) Incremental/decremental input (X200, X201) Incremental/decremental input (X200, X201), Reset input (X202) Incremental/decremental control (X200, X201) Incremental/decremental control (X200, X201), Reset input (X202)
		CHB: Do not set input X201 as high-speed counter	CHB	Does not set input X201 as high-speed counter. Incremental input (X201) Incremental input (X201), Reset input (X202) Decremental input (X201) Decremental input (X201), Reset input (X202)
	Pulse output operation mode	CH1: Use output as normal output.	CH1	Use output as normal output. Use output Y200 to Y202 as pulse output. Use output Y200 as PWM output.

◆ **NOTES**

- **If the operation mode is set to two-phase, incremental/decremental, or incremental/decremental control, the setting for CH9 is invalid in system register 400 and the setting for CHB is invalid in system register 401.**

- **If reset input settings overlap, the CH9 setting takes precedence in system register 400 and the CHB setting takes precedence in system register 401.**
- **CHA, CHB and CH1 input signals in system register 401 are the signals when the pulse I/O cassette (AFPX-PLS) is installed in the cassette mounting part 2.**
- **If the operation mode setting for the pulse output CH0 and CH1 is carried out, it cannot be used as normal output.**
  When the operation mode for the pulse output CH0 is set to 1, the reset input setting for the high-speed counter CH8 and CH9 is invalid.
  When the operation mode for the pulse output CH1 is set to 1, the reset input setting for the high-speed counter CHA and CHB is invalid.

High-speed counter, interrupt inputs

Address	Name	Default value	Description	
402	High-speed counter operation mode settings	CH0: Do not set input X0 as high-speed counter	CH0	Do not set input X0 as high-speed counter. Incremental input (X0) Decremental input (X0) Two-phase input (X0, X1)
		CH1: Do not set input X1 as high-speed counter	CH1	Do not set input X1 as high-speed counter. Incremental input (X1) Decremental input (X1) Two-phase input (X0, X1)
		CH2: Do not set input X2 as high-speed counter	CH2	Do not set input X2 as high-speed counter. Incremental input (X2) Decremental input (X2) Two-phase input (X2, X3)
		CH3: Do not set input X3 as high-speed counter	CH3	Do not set input X3 as high-speed counter. Incremental input (X3) Decremental input (X3) Two-phase input (X2, X3)
		CH4: Do not set input X4 as high-speed counter	CH4	Do not set input X4 as high-speed counter. Incremental input (X4) Decremental input (X4) Two-phase input (X3 X4)
		CH5: Do not set input X5 as high-speed counter	CH5	Do not set input X5 as high-speed counter. Incremental input (X5) Decremental input (X5) Two-phase input (X4, X5)
		CH6: Do not set input X6 as high-speed counter	CH6	Do not set input X6 as high-speed counter. Incremental input (X6) Decremental input (X6) Two-phase input (X5, X6)
		CH7: Do not set input X7 as high-speed counter	CH7	Do not set input X7 as high-speed counter. Incremental input (X7) Decremental input (X7) Two-phase input (X6, X7)

Address	Name	Default value	Description
403	Pulse catch input settings	Not set	X0 X1 X2 X3 X4 X5 X6 X7 Internal input X100 X101 X102 X200 X201 X202 Pulse I/O cassette Select whether to enable the contact for pulse catch input.
404	Interrupt input settings	Not set	X0 X1 X2 X3 X4 X5 X6 X7 Internal input X100 X101 X102 X200 X201 X202 Pulse I/O cassette Select whether to enable the contact for interrupt input.
405	Effective interrupt edge setting for internal input	Leading edge	X0 X1 X2 X3 X4 X5 X6 X7 Leading edge X0 X1 X2 X3 X4 X5 X6 X7 Trailing edge Select whether the input should be activated at a leading edge, trailing edge or both.
406	Effective interrupt edge setting for pulse I/O cassette input	Leading edge	X100 X101 X102 X200 X201 X202 Leading edge X100 X101 X102 X200 X201 X202 Trailing edge Select whether the input should be activated at a leading edge, trailing edge or both.

◆ **NOTES**

- **For counting two-phase input, only CH0, CH2, CH4 and CH6 can be used. When two-phase input is specified for CH0, CH2, CH4 and CH6, the settings for CH1, CH3, CH5 and CH7 corresponding to each CH no. are ignored. However, specify the same setting for those channels.**
- **The settings for pulse catch and interrupt input can only be specified in system registers 403 and 404.**
- **If system register 400 to 404 have been set simultaneously for the same input relay, the following precedence order is effective:**
  1. High-speed counter
  2. Pulse catch
  3. Interrupt input
  <Example>
  When the high-speed counter is being used in the addition input mode, even if input X0 is specified as an interrupt input or as pulse catch input, those settings are invalid, and X0 functions as counter input for the high-speed counter.

Tool port settings

Address	Name	Default value	Description
410	Unit no. setting	1	1 to 99
412	Communication mode setting	Computer link	Computer link General-purpose communications Modbus RTU
	Selection of modem connection	Disabled	Enabled/Disabled
413	Communication format setting	Data length bit: 8 bits Parity check: “with odd” Stop bit: 1 bit	Enter the settings for the various items. Data length bit: 7 bits/8 bits Parity check: none/with odd/with even Stop bit: 1 bit/2 bits The following setting is valid only when the communication mode specified by system register 412 has been set to “General-purpose serial communication”. Terminator: CR/CR+LF/None Header: No STX/STX
415	Communication speed (baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
420	Starting address for received buffer of general (serial data) communication mode	0	0 to 32764
421	Buffer capacity setting for data received of general (serial data) communication mode	2048	0 to 2048

COM 1 port settings

Address	Name	Default value	Description
410	Unit no. setting	1	1 to 99
412	Communication mode setting	Computer link	Computer link General-purpose serial communication PC(PLC) link Modbus RTU
	Selection of modem connection	Disabled	Enabled/Disabled

Address	Name	Default value	Description
413	Communication format setting	Data length bit: 8 bits Parity check: Odd Stop bit: 1 bit	Enter the settings for the various items. Data length bit: 7 bits/8 bits Parity check: none/with odd/with even Stop bit: 1 bit/2 bits The following setting is valid only when the communication mode specified by system register 412 has been set to "General-purpose serial communication". Terminator: CR/CR+LF/None Header: No STX/STX
415	Communication speed (Baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
416	Starting address for received buffer of general (serial data) communication mode	0	0 to 32764
417	Buffer capacity setting for data received of general (serial data) communication mode	2048	0 to 2048

◆ **NOTE**

The communication format for PLC link is fixed at: data length 8 bits, odd parity, stop bit 1, communication speed (baud rate) 15200 bps.

COM 2 port settings

Address	Name	Default value	Description
411	Unit no. setting	1	1 to 99
412	Communication mode setting	Computer link	Computer link General-purpose serial communication PLC link Modbus RTU
	Selection of modem connection	Disabled	Enabled/Disabled
	Selection of port	Built-in USB	Built-in USB Communication cassette

Address	Name	Default value	Description
414	Communication format setting	Data length bit: 8 bits Parity check: “with odd” Stop bit: 1 bit	Enter the settings for the various items. Data length bit: 7 bits/8 bits Parity check: none/odd/even Stop bit: 1 bit/2 bits The following setting is valid only when the communication mode specified by system register 412 has been set to “General-purpose serial communication”. Terminator: CR/CR+LF/None Header: No STX/STX
415	Communication speed (baud rate) setting	9600 bps	2400 bps 4800 bps 9600 bps 19200 bps 38400 bps 57600 bps 115200 bps
416	Starting address for received buffer of general (serial data) communication mode	2048	0 to 32764
417	Buffer capacity setting for data received of general (serial data) communication mode	2048	0 to 2048

NOTES

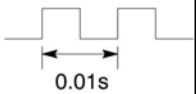
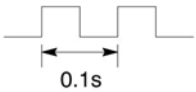
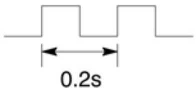
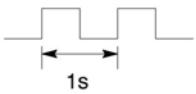
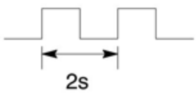
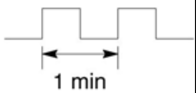
- **The communication format for PLC link is fixed at: data length 8 bits, odd parity, stop bit 1, communication speed (baud rate) 15200 bps.**
- **Use the system registers to select the USB port for C30 and C60. The USB port is the default setting for the COM2 port of C30 and C60. The communication speed for the USB port is 115.2 kbps no matter what the baud rate setting in system register 415 is.**
  The setting for no. 412 must be changed to communication cassette to use the COM2 port of the communication cassette. The COM2 port of the USB port and the communication cassette cannot be used at the same time.

32.5 Special Internal Relays

32.5.1 Special Internal Relays for FP-Sigma

The special internal relays turn on and off under special conditions. The ON and OFF states are not output externally. Writing is not possible with a programming tool or an instruction.

Relay No.: Matsushita IEC	Name	Description
R9000 %MX0.900.0	Self-diagnostic error flag	Turns on when a self-diagnostic error occurs. The content of self-diagnostic error is stored in DT90000.
R9001 %MX0.900.1	Not used	————
R9002 %MX0.900.2	Not used	————
R9003 %MX0.900.3	Not used	————
R9004 %MX0.900.4	I/O verification error flag	Turns on when an I/O verification error occurs.
R9005 %MX0.900.5	Backup battery error flag (non-hold)	Turns on for an instant when a backup battery error occurs.
R9006 %MX0.900.6	Backup battery error flag (hold)	Turns on and keeps the on state when a backup battery error occurs. Once a battery error has been detected, this is held even after recovery has been made. It goes off if the power supply is turned off, or if the system is initialized.
R9007 %MX0.900.7	Operation error flag (hold)	Turns on and keeps the on state when an operation error occurs. The address where the error occurred is stored in DT90017. (Indicates the first operation error which occurred.)
R9008 %MX0.900.8	Operation error flag (non-hold)	Turns on for an instant when an operation error occurs. The address where the operation error occurred is stored in DT90018. The contents change each time a new error occurs.
R9009 %MX0.900.9	Carry flag	This is set if an overflow or underflow occurs in the calculation results, and as a result of a shift system instruction being executed.
R900A %MX0.900.10	> flag	Turns on for an instant when the compared results become larger in the comparison instructions.
R900B %MX0.900.11	= flag	Turns on for an instant, - when the compared results are equal in the comparison instructions. - when the calculated results become 0 in the arithmetic instructions.
R900C %MX0.900.12	< flag	Turns on for an instant when the compared results become smaller in the comparison instructions".
R900D %MX0.900.13	Auxiliary timer instruction flag	Turns on when the set time elapses (set value reaches 0) in the timing operation of the F137_STMR/F183_DSTM auxiliary timer instruction. This flag turns off when the trigger for auxiliary timer instruction turns off.
R900E %MX0.900.14	Tool port communication error	Turns on when a communication error at the Tool port has occurred.

Relay No.: Matsushita IEC	**Name**	**Description**	
R900F **%MX0.900.15**	Constant scan error flag	Turns on when the scan time exceeds the time specified in system register 34 during constant scan execution. This goes on if 0 has been set using system register 34.	
R9010 **%MX0.901.0**	Always on relay	Always on.	
R9011 **%MX0.901.1**	Always off relay	Always off.	
R9012 **%MX0.901.2**	Scan pulse relay	Turns on and off alternately at each scan	
R9013 **%MX0.901.3**	Initial (on type) pulse relay	Goes on for only the first scan after operation (RUN) has been started, and goes off for the second and subsequent scans.	
R9014 **%MX0.901.4**	Initial (off type) pulse relay	Goes off for only the first scan after operation (RUN) has been started, and goes on for the second and subsequent scans.	
R9015 **%MX0.901.5**	Step ladder initial pulse relay (on type)	Turns on for an instant only in the first scan of the process the moment the step ladder process is opened.	
R9016 **%MX0.901.6**	Not used	———————	
R9017 **%MX0.901.7**	Not used	———————	
R9018 **%MX0.901.8**	0.01 s clock pulse relay	Repeats on/off operations in 0.01s cycles. (ON : OFF = 0.005s : 0.005s)	
R9019 **%MX0.901.9**	0.02 s clock pulse relay	Repeats on/off operations in 0.02s cycles. (ON : OFF = 0.01s : 0.01s)	
R901A **%MX0.901.10**	0.1 s clock pulse relay	Repeats on/off operations in 0.1s cycles. (ON : OFF = 0.05s : 0.05s)	
R901B **%MX0.901.11**	0.2 s clock pulse relay	Repeats on/off operations in 0.2s cycles. (ON : OFF = 0.1s : 0.1s)	
R901C **%MX0.901.12**	1 s clock pulse relay	Repeats on/off operations in 1s cycles. (ON : OFF = 0.5s : 0.5s)	
R901D **%MX0.901.13**	2 s clock pulse relay	Repeats on/off operations in 2s cycles. (ON : OFF = 1s : 1s)	
R901E **%MX0.901.14**	1 min clock pulse relay	Repeats on/off operations in 1 min cycles. (ON : OFF = 30s : 30s)	
R901F **%MX0.901.15**	Not used	———————	
R9020 **%MX0.902.0**	RUN mode flag	Turns off while the mode selector is set to PROG. Turns on while the mode selector is set to RUN.	

Relay No.: Matsushita IEC	Name	Description
R9021 %MX0.902.1	Not used	———————
R9022 %MX0.902.2	Not used	———————
R9023 %MX0.902.3	Not used	———————
R9024 %MX0.902.4	Not used	———————
R9025 %MX0.902.5	Not used	———————
R9026 %MX0.902.6	Message flag	Turns on while the F149_MSG instruction is executed.
R9027 %MX0.902.7	Not used	———————
R9028 %MX0.902.8	Not used	———————
R9029 %MX0.902.9	Forcing flag	Turns on during forced on/off operation for input/output relay and timer/counter contacts.
R902A %MX0.902.10	Interrupt enable flag	Turns on while the external interrupt trigger is enabled by the ICTL instruction.
R902B %MX0.902.11	Interrupt error flag	Turns on when an interrupt error occurs.
R902C %MX0.902.12	Not used	———————
R902D %MX0.902.13	Not used	———————
R902E %MX0.902.14	Not used	———————
R902F %MX0.902.15	Not used	———————
R9030 %MX0.903.0	Not used	———————
R9031 %MX0.903.1	Not used	———————
R9032 %MX0.903.2	COM port 1 communication mode flag	Turns on when program controlled communication is being used. Goes off when MEWTOCOL-COM Slave or PLC Link function is being used.
R9033 %MX0.903.3	Print instruction execution flag	Off: Printing is not executed. On: Execution is in progress.
R9034 %MX0.903.4	Run overwrite complete flag	Goes on for only the first scan following completion of a rewrite during RUN operation.
R9035 %MX0.903.5	Not used	———————
R9036 %MX0.903.6	Not used	———————
R9037 %MX0.903.7	COM port 1 communication error flag	Goes on if a transmission error occurs during data communication. Goes off when a request is made to send data, using the F159_MTRN instruction.
R9038 %MX0.903.8	COM port 1 reception done flag during general-purpose serial communication	Turns on when the terminator is received during program controlled communication.

Relay No.: Matsushita IEC	Name		Description
R9039 %MX0.903.9	COM port 1 transmission done flag during general-purpose serial communication		Goes on when transmission has been completed in program controlled communication. Goes off when transmission is requested in program controlled communication.
R903A %MX0.903.10	High-speed counter control flag	CH0	Turns on while the high-speed counter instructions F166_HC15, F167_HC1R and the pulse output instructions F171_SPDH to F176_PWMH are executed.
R903B %MX0.903.11	High-speed counter control flag	CH1	Turns on while the high-speed counter instructions F166_HC15, F167_HC1R and the pulse output instructions F171_SPDH to F176_PWMH are executed.
R903C %MX0.903.12	High-speed counter control flag	CH2	Turns on while the high-speed counter instructions F166_HC15, F167_HC1R and the pulse output instructions F171_SPDH to F176_PWMH are executed.
R903D %MX0.903.13	High-speed counter control flag	CH3	Turns on while the high-speed counter instructions F166_HC15, F167_HC1R and the pulse output instructions F171_SPDH to F176_PWMH are executed.
R903E %MX0.903.14	Not used		—
R903F %MX0.903.15	Not used		—
R9040 %MX0.904.0	Not used		—
R9041 %MX0.904.1	COM port 1 PLC link flag		Turns on while the PLC Link function is used.
R9042 %MX0.904.2	COM port 2 communication mode flag		Goes on when program controlled communication is used. Goes off when MEWTOCOL is used.
R9043 to R9046 %MX0.904.3 to %MX0.904.6	Not used		—
R9047 %MX0.904.7	COM port 2 communication error flag		Goes on if a transmission error occurs during data communication. Goes off when a request is made to send data using the F159_MTRN instruction.
R9048 %MX0.904.8	COM port 2 reception done flag during general purpose communication		Turns on when the terminator is received during program controlled communication.
R9049 %MX0.904.9	COM port 2 transmission done flag during general purpose communication		Goes on when transmission has been completed in program controlled communication. Goes off when transmission is requested in program controlled communication.
R904A to R904D %MX0.904.10 to %MX0.904.13	Not used		—
R904E %MX0.904.14	Circular interpolation control flag		This flag is set when circular interpolation instruction F176_PWMH is run. This state is maintained until the target value is achieved. While this flag is set, other positioning instructions (F171_SPDH to F176_PWMH) cannot be run.

Relay No.: Matsushita IEC	Name	Description	
R904F %MX0.904.15	Target value overwrite flag	This flag is ON for one scan when the circular interpolation instruction F176 starts. (When the circular interpolation instruction F176 is executed in the regular interrupt program, the relay is ON during the set time.)	
R9050 %MX0.905.0	MEWNET-W0 PLC link transmission error flag	When using MEWNET-W0 - turns on when a transmission error occurs in a PLC link. - turns on when there is an error in the PLC link area settings.	
R9051 to R905F %MX0.905.1 to %MX0.905.15	Not used	——	
R9060 %MX0.906.0	MEWNET-W0 PLC link transmission assurance relay	Unit no. 1	Turns on when unit no. 1 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9061 %MX0.906.1		Unit no. 2	Turns on when unit no. 2 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9062 %MX0.906.2		Unit no. 3	Turns on when unit no. 3 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9063 %MX0.906.3		Unit no. 4	Turns on when unit no. 4 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9064 %MX0.906.4		Unit no. 5	Turns on when unit no. 5 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9065 %MX0.906.5		Unit no. 6	Turns on when unit no. 6 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9066 %MX0.906.6		Unit no. 7	Turns on when unit no. 7 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9067 %MX0.906.7		Unit no. 8	Turns on when unit no. 8 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error is occurs, or when not in PLC Link mode.
R9068 %MX0.906.8		Unit no. 9	Turns on when unit no. 9 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9069 %MX0.906.9		Unit no. 10	Turns on when unit no. 10 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R906A %MX0.906.10		Unit no. 11	Turns on when unit no. 11 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R906B %MX0.906.11		Unit no. 12	Turns on when unit no. 12 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R906C %MX0.906.12		Unit no. 13	Turns on when unit no. 13 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.

Relay No.: Matsushita IEC	Name	Description	
R906D **%MX0.906.13**		Unit no. 14	Turns on when unit no. 14 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R906E **%MX0.906.14**		Unit no. 15	Turns on when unit no. 15 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R906F **%MX0.906.15**		Unit no. 16	Turns on when unit no. 16 is communicating properly in PLC Link mode. Turns off when operation is stopped, when an error occurs, or when not in PLC Link mode.
R9070 **%MX0.907.0**	MEWNET-W0 PLC link operation mode relay	Unit no. 1	Turns on when unit no. 1 is in RUN mode. Turns off when unit no. 1 is in PROG mode.
R9071 **%MX0.907.1**		Unit no. 2	Turns on when unit no. 2 is in RUN mode. Turns off when unit no. 2 is in PROG mode.
R9072 **%MX0.907.2**		Unit no. 3	Turns on when unit no. 3 is in RUN mode. Turns off when unit no. 3 is in PROG mode.
R9073 **%MX0.907.3**		Unit no. 4	Turns on when unit no. 4 is in RUN mode. Turns off when unit no. 4 is in PROG mode.
R9074 **%MX0.907.4**		Unit no. 5	Turns on when unit no. 5 is in RUN mode. Turns off when unit no. 5 is in PROG mode.
R9075 **%MX0.907.5**		Unit no. 6	Turns on when unit no. 6 is in RUN mode. Turns off when unit no. 6 is in PROG mode.
R9076 **%MX0.907.6**		Unit no. 7	Turns on when unit no. 7 is in RUN mode. Turns off when unit no. 7 is in PROG mode.
R9077 **%MX0.907.7**		Unit no. 8	Turns on when unit no. 8 is in RUN mode. Turns off when unit no. 8 is in PROG mode.
R9078 **%MX0.907.8**		Unit no. 9	Turns on when unit no. 9 is in RUN mode. Turns off when unit no. 9 is in PROG mode.
R9079 **%MX0.907.9**		Unit no. 10	Turns on when unit no. 10 is in RUN mode. Turns off when unit no. 10 is in PROG mode.
R907A **%MX0.907.10**		Unit no. 11	Turns on when unit no. 11 is in RUN mode. Turns off when unit no. 11 is in PROG mode.
R907B **%MX0.907.11**		Unit no. 12	Turns on when unit no. 12 is in RUN mode. Turns off when unit no. 12 is in PROG mode.
R907C **%MX0.907.12**		Unit no. 13	Turns on when unit no. 13 is in RUN mode. Turns off when unit no. 13 is in PROG mode.
R907D **%MX0.907.13**		Unit no. 14	Turns on when unit no. 14 is in RUN mode. Turns off when unit no. 14 is in PROG mode.
R907E **%MX0.907.14**		Unit no. 15	Turns on when unit no. 15 is in RUN mode. Turns off when unit no. 15 is in PROG mode.
R907F **%MX0.907.15**		Unit no. 16	Turns on when unit no. 16 is in RUN mode. Turns off when unit no. 16 is in PROG mode.

32.5.2 Special Internal Relays for FP-X

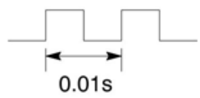
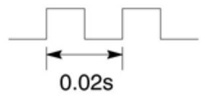
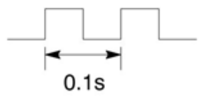
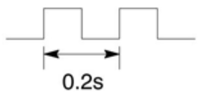
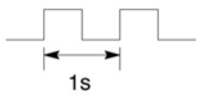
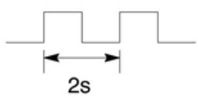
The special internal relays turn on and off under special conditions. The ON and OFF states are not output externally. Writing is not possible with a programming tool or an instruction.

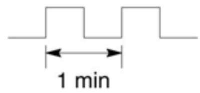
WR900

Relay no. FP address IEC	Name	Description
R9000 %MX0.900.0	Self-diagnostic error flag	Turns on when a self-diagnostic error occurs. The content of self-diagnostic error is stored in DT90000.
R9001 %MX0.900.1	Not used	—
R9002 %MX0.900.2	Application cassette I/O error flag	Turns on when an error is detected in the I/O type application cassette.
R9003 %MX0.900.3	Application cassette abnormal error flag	Turns on when an error is detected in the application cassette.
R9004 %MX0.900.4	I/O verification error flag	Turns on when an I/O verification error occurs.
R9005 %MX0.900.5	Backup battery error flag (non-hold)	Turns on for an instant when a backup battery error occurs.
R9006 %MX0.900.6	Backup battery error flag (hold)	Turns on when a backup battery error occurs. Once a battery error has been detected, this is held even after recovery has been made. It goes off if the power supply is turned off, or if the system is initialized.
R9007 %MX0.900.7	Operation error flag (hold)	Turns on and keeps the on state when an operation error occurs. The address where the error occurred is stored in DT90017. (indicates the first operation error which occurred).
R9008 %MX0.900.8	Operation error flag (non-hold)	Turns on for an instant when an operation error occurs. The address where the operation error occurred is stored in DT90018. The contents change each time a new error occurs.
R9009 %MX0.900.9	Carry flag	This is set if an overflow or underflow occurs in the calculation results, and as a result of a shift system instruction being executed.
R900A %MX0.900.10	> Flag	Turns on for an instant when the compared results become larger in the comparison instructions.
R900B %MX0.900.11	= Flag	Turns on for an instant when the compared results are equal in the comparison instructions. when the calculated results become 0 in the arithmetic instructions.
R900C %MX0.900.12	< Flag	Turns on for an instant when the compared results become smaller in the comparison instructions.
R900D %MX0.900.13	Auxiliary timer instruction flag	Turns on when the set time elapses (set value reaches 0) in the timing operation of the F137_STMR/F183_DSTM auxiliary timer instruction. The flag turns off when the trigger for auxiliary timer instruction turns off.
R900E %MX0.900.14	Tool port communication error	Turns on when a communication error at Tool port has occurred.

Relay no. FP address IEC	Name	Description
R900F **%MX0.900.15**	Constant scan error flag	Turns on when scan time exceeds the time specified in system register 34 during constant scan execution. This goes on if 0 has been set using system register 34.

WR901

Relay no. FP address IEC	Name	Description	
R9010 **%MX0.901.0**	Always on relay	Always on.	
R9011 **%MX0.901.1**	Always off relay	Always off.	
R9012 **%MX0.901.2**	Scan pulse relay	Turns on and off alternately at each scan.	
R9013 **%MX0.901.3**	Initial (on type) pulse relay	Goes on for only the first scan after operation (RUN) has been started, and goes off for the second and subsequent scans.	
R9014 **%MX0.901.4**	Initial (off type) pulse relay	Goes off for only the first scan after operation (RUN) has been started, and goes on for the second and subsequent scans.	
R9015 **%MX0.901.5**	Step ladder initial pulse relay (on type)	Turns on for only the first scan of a process after the boot at the step ladder control.	
R9016 **%MX0.901.6**	Not used	—	
R9017 **%MX0.901.7**	Not used	—	
R9018 **%MX0.901.8**	0.01 s clock pulse relay	Repeats on/off operations in 0.01 sec. cycles. (ON : OFF = 0.005s : =.005s)	
R9019 **%MX0.901.9**	0.02 s clock pulse relay	Repeats on/off operations in 0.02 s. cycles. (ON : OFF = 0.01s : 0.01s)	
R901A **%MX0.901.10**	0.1 s clock pulse relay	Repeats on/off operations in 0.1 s. cycles. (ON : OFF = 0.05s : 0.05s)	
R901B **%MX0.901.11**	0.2 s clock pulse relay	Repeats on/off operations in 0.2 s. cycles. (ON : OFF = 0.1s : 0.1s)	
R901C **%MX0.901.12**	1 s clock pulse relay	Repeats on/off operations in 1 s. cycles. (ON : OFF = 0.5s : 0.5s)	
R901D **%MX0.901.13**	2 s clock pulse relay	Repeats on/off operations in 2 s. cycles. (ON : OFF = 1s : 1s)	

Relay no. FP address IEC	Name	Description
R901E **%MX0.901.14**	1 min clock pulse relay	Repeats on/off operations in 1 min. cycles. (ON : OFF = 30s : 30s) 
R901F **%MX0.901.15**	Not used	—

WR902

Relay no. FP address IEC	Name	Description
R9020 **%MX0.902.0**	RUN mode flag	Turns off while the mode selector is set to PROG. Turns on while the mode selector is set to RUN.
R9021 **%MX0.902.1**	Not used	—
R9022 **%MX0.902.2**	Not used	—
R9023 **%MX0.902.3**	Not used	—
R9024 **%MX0.902.4**	Not used	—
R9025 **%MX0.902.5**	Not used	—
R9026 **%MX0.902.6**	Message flag	Turns on while the F149_MSG instruction is executed.
R9027 **%MX0.902.7**	Not used	—
R9028 **%MX0.902.8**	Not used	—
R9029 **%MX0.902.9**	Forcing flag	Turns on during forced on/off operation for input/output relay timer/counter contacts.
R902A **%MX0.902.10**	Interrupt enable flag	Turns on while the external interrupt trigger is enabled by the ICTL instruction.
R902B **%MX0.902.11**	Interrupt error flag	Turns on when an interrupt error occurs.
R902C **%MX0.902.12**	Not used	—
R902E **%MX0.902.14**	Not used	—
R902F **%MX0.902.15**	Not used	—

WR903

Relay no. FP address IEC	Name	Description
R9030 **%MX0.903.0**	Not used	—

Relay no. FP address IEC	Name	Description
R9031 %MX0.903.1	Not used	—
R9032 %MX0.903.2	COM1 port mode flag	Turns on when the general purpose communication function is being used Goes off when any function other than the general purpose communication function is being used.
R9033 %MX0.903.3	PR instruction flag	Off: Printing is not executed. On: Execution is in progress.
R9034 %MX0.903.4	Editing in RUN mode flag	Goes on for only the first scan following completion of a rewrite during the RUN operation.
R9035 %MX0.903.5	Not used	—
R9036 %MX0.903.6	Not used	—
R9037 %MX0.903.7	COM1 port communication error flag	Goes on if a transmission error occurs during data communication. Goes off when a request is made to send data, using the F159_MTRN instruction.
R9038 %MX0.903.8	COM1 port reception done flag during general- purpose serial communication	Turns on when the terminator is received during general - purpose serial communication.
R9039 %MX0.903.9	COM1 port transmission done flag during general purpose serial communication	Goes on when transmission has been completed in general purpose serial communication. Goes off when transmission is requested in general purpose serial communication.
R903A %MX0.903.10	Not used	—
R903B %MX0.903.11	Not used	—
R903C %MX0.903.12	Not used	—
R903D %MX0.903.13	Not used	—
R903E %MX0.903.14	TOOL port reception done flag during general purpose communication	Turns on when the terminator is received during general purpose serial communication.
R903F %MX0.903.15	TOOL port transmission done flag during general purpose serial communication	Goes on when transmission has been completed in general purpose serial communication. Goes off when transmission is requested in general purpose serial communication.

◆ NOTE

R9030 to R903F can change during 1 scan.

WR904

Relay no. FP address IEC	Name	Description
R9040 %MX0.904.0	TOOL port mode flag	Goes on when the general purpose serial communication is used. Goes off when the MEWTOCOL is used.
R9041 %MX0.904.1	COM1 port PLC link flag	Turns on while the PLC link function is used.
R9042 %MX0.904.2	COM2 port communication mode flag	Goes on when the general purpose serial communication is used. Goes off when the MEWTOCOL is used.
R9043 %MX0.904.3	Not used	—
R9044 %MX0.904.4	COM1 port SEND/RECV instruction execution flag	Monitors whether the F145_SEND or F146_RECV instructions can be executed or not for the COM1 port. **Off:** Neither of the instructions can be executed, i.e. one is already being executed. **On:** One of the above mentioned instructions can be executed.
R9045 %MX0.904.5	COM1 port SEND/RECV instruction execution end flag	Monitors if an abnormality has been detected during the execution of the F145_SEND or F146_RECV instructions for the COM1 port: **Off:** No abnormality detected. **On:** An abnormality detected. (communication error). The error code is stored in DT90124.
R9046 %MX0.904.6	Not used	—
R9047 %MX0.904.7	COM2 port communication error flag	Goes on if a transmission error occurs during data communication. Goes off when a request is made to send data, using the F159_MTRN instruction.
R9048 %MX0.904.8	COM2 port reception done flag during general purpose communication	Turns on when the terminator is received during general purpose serial communication.
R9049 %MX0.904.9	COM2 port transmission done flag during general purpose communication	Goes on when transmission has been completed in general purpose serial communication. Goes off when transmission is requested in general purpose communication.
R904A %MX0.904.10	COM2 port SEND/RECV instruction execution flag	Monitors whether the F145_SEND or F146_RECV instructions can be executed or not for the COM2 port. **Off:** Neither of the instructions can be executed, i.e. one is already being executed. **On:** One of the above mentioned instructions can be executed.
R904B %MX0.904.11	COM2 port SEND/RECV instruction execution end flag	Monitors if an abnormality has been detected during the execution of the F145_SEND or F146_RECV instructions for the COM2 port: **Off:** No abnormality detected. **On:** An abnormality detected. (communication error). The error code is stored in DT90125.
R904C to R904F %MX0.904.12 to %MX0.904.15	Not used	—

NOTE

R9040 to R904F can change during 1 scan.

WR905

Relay no. FP address IEC	Name	Description
R9050 **%MX0.905.0**	MEWNET-W0 PLC link transmission error flag	When using MEWNET-W0 Turns on when a transmission error occurs at PLC link. Turns on when there is an error in the PLC link area settings.
R9051 to R905F **%MX0.905.1 to %MX0.905.15**	Not used	—

WR906: MEWNET-W0 PLC link 0 transmission assurance relays

Relay no. FP address IEC	Unit no.	Description
R9060 **%MX0.906.0**	Unit no. 1	Turns on when the unit no. is communicating properly in PLC link mode. Turns off when operation is stopped, when an error occurs, or when not in the PLC link mode.
R9061 **%MX0.906.1**	Unit no. 2	
R9062 **%MX0.906.2**	Unit no. 3	
R9063 **%MX0.906.3**	Unit no. 4	
R9064 **%MX0.906.4**	Unit no. 5	
R9065 **%MX0.906.5**	Unit no. 6	
R9066 **%MX0.906.6**	Unit no. 7	
R9067 **%MX0.906.7**	Unit no. 8	
R9068 **%MX0.906.8**	Unit no. 9	
R9069 **%MX0.906.9**	Unit no. 10	
R906A **%MX0.906.10**	Unit no. 11	
R906B **%MX0.906.11**	Unit no. 12	
R906C **%MX0.906.12**	Unit no. 13	
R906D **%MX0.906.13**	Unit no. 14	
R906E **%MX0.906.14**	Unit no. 15	
R906F **%MX0.906.15**	Unit no. 16	

WR907: MEWNET-W0 PLC link 0 operation mode relays

Relay no. FP address IEC	Unit no.	Description
R9070 **%MX0.907.0**	Unit no. 1	Turns on when the unit no. is in RUN mode. Turns off when the unit no. is in PROG. mode.
R9071 **%MX0.907.1**	Unit no. 2	
R9072 **%MX0.907.2**	Unit no. 3	
R9073 **%MX0.907.3**	Unit no. 4	
R9074 **%MX0.907.4**	Unit no. 5	
R9075 **%MX0.907.5**	Unit no. 6	
R9076 **%MX0.907.6**	Unit no. 7	
R9077 **%MX0.907.7**	Unit no. 8	
R9078 **%MX0.907.8**	Unit no. 9	
R9079 **%MX0.907.9**	Unit no. 10	
R907A **%MX0.907.10**	Unit no. 11	
R907B **%MX0.907.11**	Unit no. 12	
R907C **%MX0.907.12**	Unit no. 13	
R907D **%MX0.907.13**	Unit no. 14	
R907E **%MX0.907.14**	Unit no. 15	
R907F **%MX0.907.15**	Unit no. 16	

WR908: MEWNET-W0 PLC link 1 transmission assurance relays

Relay no. FP address IEC	Unit no.	Description
R9080 %MX0.908.0	Unit no. 1	Turns on when the unit no. is communicating properly in PLC link mode. Turns off when operation is stopped, when an error occurs, or when not in the PLC link mode.
R9081 %MX0.908.1	Unit no. 2	
R9082 %MX0.908.2	Unit no. 3	
R9083 %MX0.908.3	Unit no. 4	
R9084 %MX0.908.4	Unit no. 5	
R9085 %MX0.908.5	Unit no. 6	
R9086 %MX0.908.6	Unit no. 7	
R9087 %MX0.908.7	Unit no. 8	
R9088 %MX0.908.8	Unit no. 9	
R9089 %MX0.908.9	Unit no. 10	
R908A %MX0.908.10	Unit no. 11	
R908B %MX0.908.11	Unit no. 12	
R908C %MX0.908.12	Unit no. 13	
R908D %MX0.908.13	Unit no. 14	
R908E %MX0.908.14	Unit no. 15	
R908F %MX0.908.15	Unit no. 16	

WR909: MEWNET-W0 PLC link 1 operation mode relays

Relay no. FP address IEC	Unit no.	Description
R9090 **%MX0.909.0**	Unit no. 1	Turns on when the unit no. is in RUN mode. Turns off when the unit no. is in PROG. mode.
R9091 **%MX0.909.1**	Unit no. 2	
R9092 **%MX0.909.2**	Unit no. 3	
R9093 **%MX0.909.3**	Unit no. 4	
R9094 **%MX0.909.4**	Unit no. 5	
R9095 **%MX0.909.5**	Unit no. 6	
R9096 **%MX0.909.6**	Unit no. 7	
R9097 **%MX0.909.7**	Unit no. 8	
R9098 **%MX0.909.8**	Unit no. 9	
R9099 **%MX0.909.9**	Unit no. 10	
R909A **%MX0.909.10**	Unit no. 11	
R909B **%MX0.909.11**	Unit no. 12	
R909C **%MX0.909.12**	Unit no. 13	
R909D **%MX0.909.13**	Unit no. 14	
R909E **%MX0.909.14**	Unit no. 15	
R909F **%MX0.909.15**	Unit no. 16	

WR910

Relay no. FP address IEC	Name	Description
R9100 to R910F %MX0.910.0 to %MX0.910.15	Not used	—

WR911

Relay no. FP address IEC	Control flag name	Description
R9110 %MX0.911.0	HSC-CH0	Turns on while the F166_HC1S and F167_HC1R instructions are executed. Turns off when the F166_HC1S and F167_HC1R instructions are completed.
R9111 %MX0.911.1	HSC-CH1	
R9112 %MX0.911.2	HSC-CH2	
R9113 %MX0.911.3	HSC-CH3	
R9114 %MX0.911.4	HSC-CH4	
R9115 %MX0.911.5	HSC-CH5	
R9116 %MX0.911.6	HSC-CH6	
R9117 %MX0.911.7	HSC-CH7	
R9118 %MX0.911.8	HSC-CH8	
R9119 %MX0.911.9	HSC-CH9	
R911A %MX0.911.10	HSC-CHA	
R911B %MX0.911.11	HSC-CHB	
R911C %MX0.911.12	PLS-CH0	Turns on while pulses are being output by the F171_SPDH, F172_PLSH, F173_PWMH and F174_SP0H instructions.
R911D %MX0.911.13	PLS-CH1	
R911E %MX0.911.14	Not used	—
R911F %MX0.911.15	Not used	—

32.6 Special Data Registers

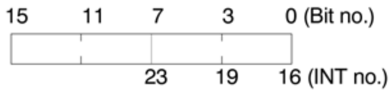
32.6.1 Special Data Registers for FP-Sigma

The special data registers are one word (16-bit) memory areas which store specific information.

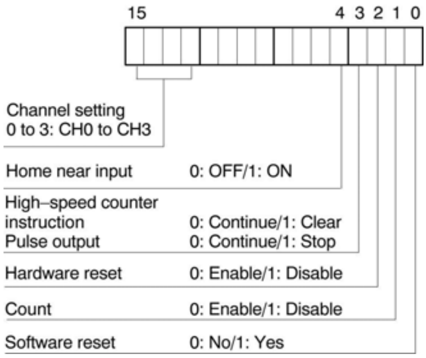
(A: Available, N/A: Not available)

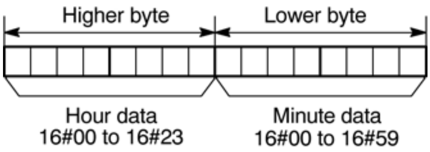
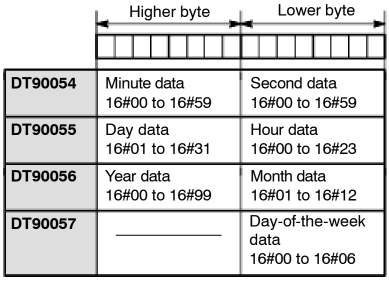
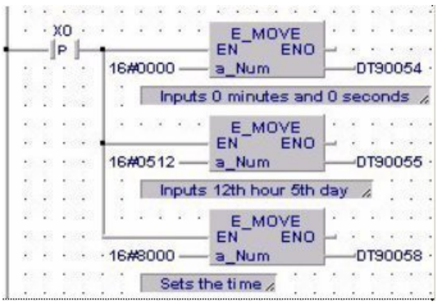
FP Address IEC Address	Name	Description	Read	Write
DT90000 **%MW5.90000**	Self-diagnostic error code	The self-diagnostic error code is stored here when a self-diagnostic error occurs.	A	N/A
DT90001 **%MW5.90001**	Not used	———————	N/A	N/A
DT90002 **%MW5.90002**	Position of abnormal I/O unit for FPΣ left side expansion	When an error occurs at an FPΣ expansion I/O unit, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (unit no.) on: error, off: normal	A	N/A
DT90003 **%MW5.90003**	Not used	———————	N/A	N/A
DT90004 **%MW5.90004**	Not used	———————	N/A	N/A
DT90005 **%MW5.90005**	Not used	———————	N/A	N/A
DT90006 **%MW5.90006**	Position of abnormal intelligent unit for FPΣ left side expansion	When an error condition is detected in an intelligent unit, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (unit no.) on: error, off: normal	A	N/A
DT90007 **%MW5.90007**	Not used	———————	N/A	N/A
DT90008 **%MW5.90008**	Not used	———————	N/A	N/A
DT90009 **%MW5.90009**	Communication error flag for COM 2	Stores the error contents when using COM port 2. Bit 0: Overrun error Bit 1: Framing error Bit 2: Parity error	A	N/A

FP Address IEC Address	Name	Description	Read	Write
DT90010 **%MW5.90010**	Position of I/O verify error unit for FP0 right side expansion	When the state of installation of an FP0 expansion I/O unit has changed since the power was turned on, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (unit no.) on: error, off: normal	A	N/A
DT90011 **%MW5.90011**	Position of I/O verify error unit for FPΣ left side expansion	When the state of installation of an FP0 expansion I/O unit has changed since the power was turned on, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (unit no.) on: error, off: normal	A	N/A
DT90012 **%MW5.90012**	Not used	————	N/A	N/A
DT90013 **%MW5.90013**	Not used	————	N/A	N/A
DT90014 **%MW5.90014**	Operation auxiliary register for data shift instruction	One shift-out hexadecimal digit is stored in bit positions 0 to 3 when the data shift instruction **F105_BSR** or **F106_BSL** is executed. The value can be read and written by executing the **F0_MV** instruction.	A	N/A
DT90015 **%MW5.90015**	Operation auxiliary register for division instruction	The divided remainder (16-bit) is stored in DT90015 when the division instruction **F32_%** or **F52_B%** instruction is executed. The divided remainder (32-bit) is stored in DT90015 and DT90016 when the division instruction **F33_D%** or **F53_DB%** is executed. The value can be read and written by executing the **F0_MV** instruction.	A	N/A
DT90016 **%MW5.90016**	Operation auxiliary register for division instruction	The divided remainder (16-bit) is stored in DT90015 when the division instruction **F32_%** or **F52_B%** instruction is executed. The divided remainder (32-bit) is stored in DT90015 and DT90016 when the division instruction **F33_D%** or **F53_DB%** is executed. The value can be read and written by executing the **F0_MV** instruction.	A	N/A
DT90017 **%MW5.90017**	Operation error address (hold type)	After commencing operation, the address where the first operation error occurred is stored. Monitor the address using decimal display.	A	N/A
DT90018 **%MW5.90018**	Operation error address (non-hold type)	The address where an operation error occurred is stored. Each time an error occurs, the new address overwrites the previous address. At the beginning of a scan, the address is 0. Monitor the address using decimal display.	A	N/A

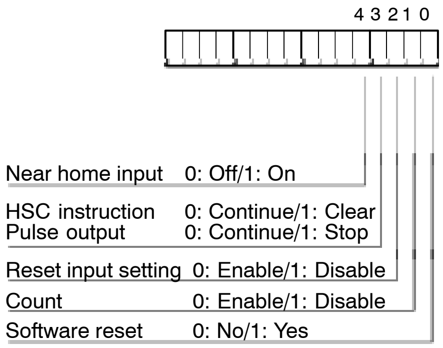
FP Address IEC Address	Name	Description	Read	Write
DT90019 **%MW5.90019**	2.5ms ring counter	The data stored here is increased by one every 2.5ms. (16#0 to 16#FFFF) Difference between the values of the two points (absolute value) × 2.5ms = elapsed time between the two points.	A	N/A
DT90020 **%MW5.90020**	Not used	———————	N/A	N/A
DT90021 **%MW5.90021**	Not used	———————	N/A	N/A
DT90022 **%MW5.90022**	Scan time (current value) (see note)	The current scan time is stored here. The scan time is calculated using the formula: Scan time (ms) = stored data (decimal) × 0.1ms Example: 50 indicates 5ms.	A	N/A
DT90023 **%MW5.90023**	Scan time (minimum value) (see note)	The minimum scan time is stored here. The scan time is calculated using the formula: Scan time (ms) = stored data (decimal) × 0.1ms Example: 50 indicates 5ms.	A	N/A
DT90024 **%MW5.90024**	Scan time (maximum value) (see note)	The maximum scan time is stored here. The scan time is calculated using the formula: Scan time (ms) = stored data (decimal) × 0.1ms Example: 125 indicates 12.5ms.	A	N/A
	Scan time display is only possible in RUN mode and shows the operation cycle time. (In PROG mode, the scan time for the operation is not displayed.) The maximum and minimum values are cleared each time the mode is switched from RUN to PROG.			
DT90025 **%MW5.90025**	Mask condition monitoring register for interrupts (INT 0 to 7)	The mask conditions of interrupts using the ICTL instruction is stored here. Monitor using binary display.  0: interrupt disabled (masked) 1: interrupt enabled (unmasked)	A	N/A
DT90026 **%MW5.90026**	Not used	———————	N/A	N/A
DT90027 **%MW5.90027**	Periodical interrupt interval (INT 24)	The value set by the **ICTL** instruction is stored. 0: periodical interrupt is not used 1 to 3000: 0.5ms to 1.5s or 10ms to 30s	A	N/A
DT90028 **%MW5.90028**	Not used	———————	N/A	N/A
DT90029 **%MW5.90029**	Not used	———————	N/A	N/A
DT90030 **%MW5.90030**	Message 0	The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90031 **%MW5.90031**	Message 1	The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A

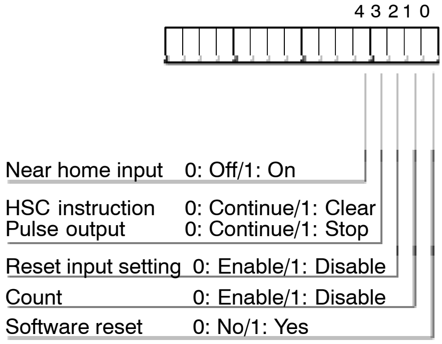
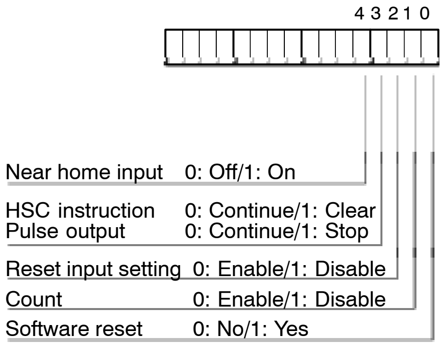
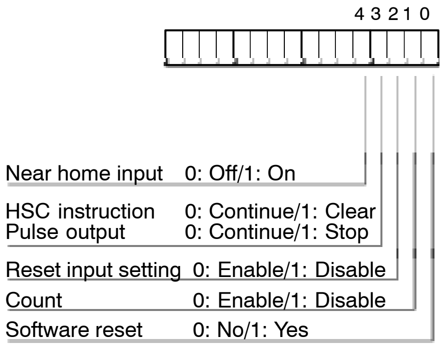
FP Address IEC Address	Name		Description	Read	Write
DT90032 **%MW5.90032**	Message 2		The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90033 **%MW5.90033**	Message 3		The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90034 **%MW5.90034**	Message 4		The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90035 **%MW5.90035**	Message 5		The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90036 **%MW5.90036**	Not used		———	N/A	N/A
DT90037 **%MW5.90037**	Operation auxiliary register for search instruction **F96_SRC**		The number of data that match the searched data is stored here when the **F96_SRC** instruction is executed.	A	N/A
DT90038 **%MW5.90038**	Operation auxiliary register for search instruction **F96_SRC**		The position of the first matching data is stored here when the **F96_SRC** instruction is executed.	A	N/A
DT90039 **%MW5.90039**	Not used		———	N/A	N/A
DT90040 **%MW5.90040**	Potentiometer (volume) input V0		The potentiometer value (0 to 1000) is stored here. This value can be used in analog timers and other applications by using the program to read this value to a data register. V0→DT90040 V1→DT90041	A	N/A
DT90041 **%MW5.90041**	Potentiometer (volume) input V1		The potentiometer value (0 to 1000) is stored here. This value can be used in analog timers and other applications by using the program to read this value to a data register. V0→DT90040 V1→DT90041	A	N/A
DT90042 **%MW5.90042**	———		Used by the system.	N/A	N/A
DT90043 **%MW5.90043**	———		Used by the system.	N/A	N/A
DT90044 **%MW5.90044**	High-speed counter elapsed value	For CH0	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A
DT90045 **%MW5.90045**	High-speed counter elapsed value	For CH0	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A
DT90046 **%MW5.90046**	High-speed counter target value	For CH0	The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here. Target values have been preset for the various instructions to be used when the high-speed counter related instruction **F166_HC1S, F167_HC1R, F171_SPDH, F172_SPSH, F174_SP0H, F175_SPSH,** or **F176_SPCH** is executed. The value can be read by executing an instruction.	A	N/A

<table>
<tr><th>FP Address
IEC Address</th><th colspan="2">Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90047
%MW5.90047</td><td>High-speed counter target value</td><td>For CH0</td><td>The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here.
Target values have been preset for the various instructions to be used when the high-speed counter related instruction F166_HC1S, F167_HC1R, F171_SPDH, F172_SPSH, F174_SP0H, F175_SPSH, or F176_SPCH is executed. The value can be read by executing an instruction.</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90048
%MW5.90048</td><td>High-speed counter elapsed value area</td><td>For CH1</td><td>The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.</td><td>A</td><td>A</td></tr>
<tr><td>DT90049
%MW5.90049</td><td>High-speed counter elapsed value area</td><td>For CH1</td><td>The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.</td><td>A</td><td>A</td></tr>
<tr><td>DT90050
%MW5.90050</td><td>High-speed counter target value area</td><td>For CH1</td><td>The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here.
Target values have been preset for the various instructions to be used when the high-speed counter related instruction F166_HC1S or F167_HC1R is executed. The value can be read by executing an instruction.</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90051
%MW5.90051</td><td>High-speed counter target value area</td><td>For CH1</td><td>The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here.
Target values have been preset for the various instructions to be used when the high-speed counter related instruction F166_HC1S or F167_HC1R is executed. The value can be read by executing an instruction.</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90052
%MW5.90052</td><td colspan="2">High-speed counter and pulse output control flag</td><td>Used to reset the high-speed counter, disable counting, continue or clear the high-speed counter instruction. This register can be set by executing an instruction
Control code setting:

</td><td>N/A</td><td>A</td></tr>
</table>

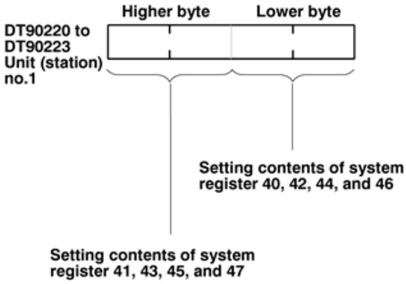
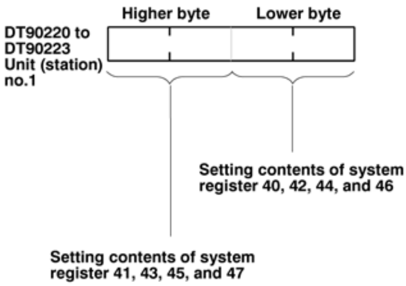
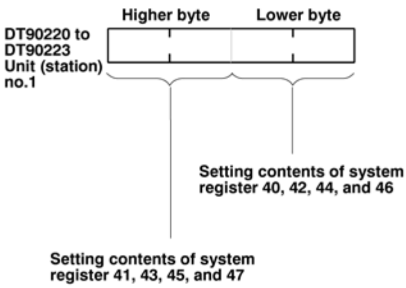
FP Address IEC Address	Name	Description	Read	Write
DT90053 %MW5.90053	Clock/calendar monitor (hour/minute)	Hour and minute data of the clock/calendar are stored here. This data is read-only data, it cannot be overwritten. 	A	N/A
DT90054 %MW5.90054	Clock/calendar setting (minute/second)	The year, month, day, hour, minute, second, and day-of-the-week data for the calendar timer is stored. The built-in calendar timer will operate correctly through the year 2099 and supports leap years. The calendar timer can be set by writing a value using a programming tool software or a programming instruction (see example for DT90058). 	A	A
DT90055 %MW5.90055	Clock/calendar setting (day/hour)		A	A
DT90056 %MW5.90056	Clock/calendar setting (year/month)		A	A
DT90057 %MW5.90057	Clock/calendar setting (day-of-the-week)		A	A
DT90058 %MW5.90058	Clock/calendar time setting	By setting the highest bit of DT90058 to 1, time and date become that written to DT90054 to DT90057. After the time has been set, DT90058 is cleared to 0. **Example:** Set the time to 12:00:00 on day 5 when X0 turns ON.  If you changed the values of DT90054 to DT90057 using the programming tool, it is not necessary to set DT90058. Please refer to the section on time setting for more information.	A	A
DT90059 %MW5.90059	Serial communication error code	Error code is stored here when a communication error occurs.	N/A	N/A

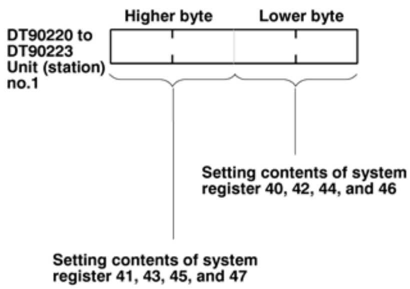
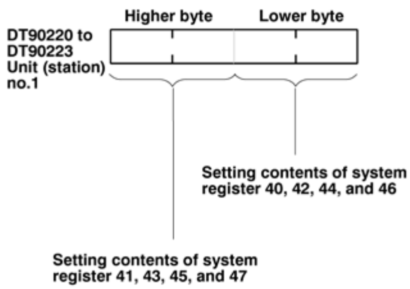
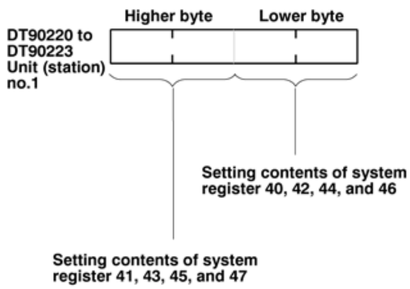
FP Address IEC Address	Name	Description	Read	Write
DT90060 to DT90122 %MW5.90060 to %MW5.90122	Step ladder process (0 to 999)	Indicates the startup condition of the step ladder process. When the process starts, the bit corresponding to the process number turns on. Monitor using binary display. **Example:** 	A	A
DT90123 to DT90125 %MW5.90123 to %MW5.90125	Not used	————	N/A	N/A
DT90126 %MW5.90126	Forced Input/ Output unit no.	Used by the system.	N/A	N/A
DT90127 to DT90139 %MW5.90127 to %MW5.90139	Not used	————	N/A	N/A
DT90140 %MW5.90140	MEWNET-W0 PLC link status	The number of times the receiving operation is performed.	A	N/A
DT90141 %MW5.90141	MEWNET-W0 PLC link status	The current interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90142 %MW5.90142	MEWNET-W0 PLC link status	The minimum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90143 %MW5.90143	MEWNET-W0 PLC link status	The maximum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90144 %MW5.90144	MEWNET-W0 PLC link status	The number of times the sending operation is performed.	A	N/A
DT90145 %MW5.90145	MEWNET-W0 PLC link status	The current interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90146 %MW5.90146	MEWNET-W0 PLC link status	The minimum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90147 %MW5.90147	MEWNET-W0 PLC link status	The maximum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90148 to DT90155 %MW5.90148 to %MW5.90155	Not used	————	N/A	N/A
DT90156 %MW5.90156	MEWNET-W0 PLC link status	Area used for measurement of receiving interval.	A	N/A
DT90157 %MW5.90157	MEWNET-W0 PLC link status	Area used for measurement of sending interval.	A	N/A
DT90158 %MW5.90158	Not used	————	N/A	N/A

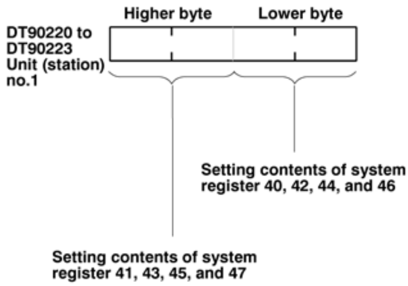
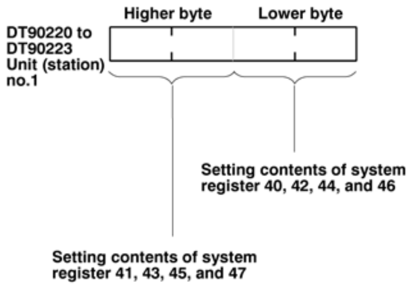
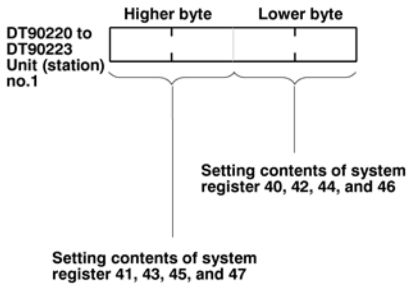
FP Address IEC Address	Name	Description	Read	Write
DT90159 %MW5.90159	Not used	—	N/A	N/A
DT90160 %MW5.90160	MEWNET-W0 PLC link unit no.	Stores the unit no. of a PLC link	A	N/A
DT90161 %MW5.90161	MEWNET-W0 PLC link error flag	Stores the error contents of a PLC link	A	N/A
DT90162 to DT90169 %MW5.90162 to %MW5.90169	Not used	—	N/A	N/A
DT90170 %MW5.90170	MEWNET-W0 PLC link status	Duplicated destination for PLC inter-link address	A	N/A
DT90171 %MW5.90171	MEWNET-W0 PLC link status	Counts how many times a token is lost.	A	N/A
DT90172 %MW5.90172	MEWNET-W0 PLC link status	Counts how many times two or more tokens are detected.	A	N/A
DT90173 %MW5.90173	MEWNET-W0 PLC link status	Counts how many times a signal is lost.	A	N/A
DT90174 %MW5.90174	MEWNET-W0 PLC link status	No. of times undefined commands have been received	A	N/A
DT90175 %MW5.90175	MEWNET-W0 PLC link status	No. of times sum check errors have occurred during reception	A	N/A
DT90176 %MW5.90176	MEWNET-W0 PLC link status	No. of times format errors have occurred in received data	A	N/A
DT90177 %MW5.90177	MEWNET-W0 PLC link status	No. of times transmission errors have occurred	A	N/A
DT90178 %MW5.90178	MEWNET-W0 PLC link status	No. of times procedural errors have occurred	A	N/A
DT90179 %MW5.90179	MEWNET-W0 PLC link status	No. of times overlapping master units have occurred	A	N/A
DT90180 to DT90189 %MW5.90180 to %MW5.90189	Not used	—	N/A	N/A
DT90190 %MW5.90190	High-speed counter control code monitor for CH0	This monitors the data specified in DT90052.  	A	N/A

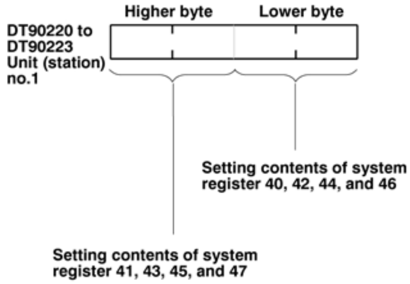
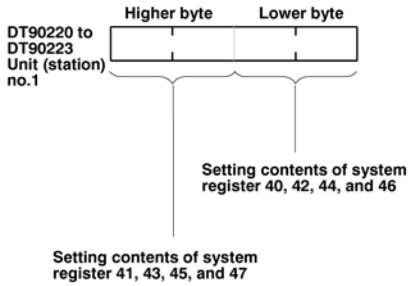
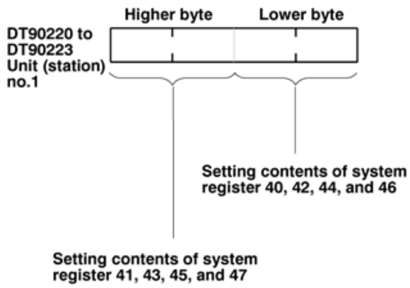
FP Address IEC Address	Name		Description	Read	Write
DT90191 **%MW5.90191**	High-speed counter control code monitor for CH1		This monitors the data specified in DT90052.  		
DT90192 **%MW5.90192**	High-speed counter control code monitor for CH2		This monitors the data specified in DT90052.  		
DT90193 **%MW5.90193**	High-speed counter control code monitor for CH3		This monitors the data specified in DT90052.  		
DT90194 to DT90199 **%MW5.90194 to %MW5.90199**	Not used		———	N/A	N/A
DT90200 **%MW5.90200**	High-speed counter elapsed value	For CH2	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A
DT90201 **%MW5.90201**	High-speed counter elapsed value	For CH2	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A

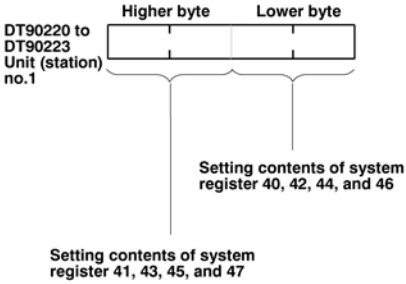
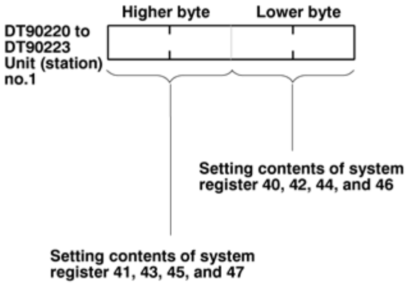
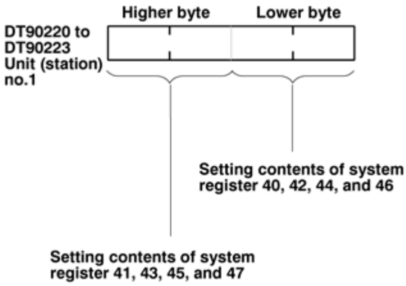
FP Address IEC Address	Name		Description	Read	Write
DT90202 %MW5.90202	High-speed counter target value	For CH2	The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high-speed counter related instruction **F166_HC1S, F167_HC1R, F171_SPDH, F172_SPSH, F174_SP0H, F175_SPSH,** or **F176_SPCH** is executed. The value can be read by executing an instruction.	A	N/A
DT90203 %MW5.90203	High-speed counter target value	For CH2	The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high-speed counter related instruction **F166_HC1S, F167_HC1R, F171_SPDH, F172_SPSH, F174_SP0H, F175_SPSH,** or **F176_SPCH** is executed. The value can be read by executing an instruction.	A	N/A
DT90204 %MW5.90204	High-speed counter elapsed value	For CH3	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A
DT90205 %MW5.90205	High-speed counter elapsed value	For CH3	The elapsed value (32-bit data) for the high-speed counter is stored here. The value can be read and written by executing an instruction.	A	A
DT90206 %MW5.90206	High-speed counter target value	For CH3	The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high-speed counter related instruction **F166_HC1S** or **F167_HC1R** is executed. The value can be read by executing an instruction.	A	N/A
DT90207 %MW5.90207	High-speed counter target value	For CH3	The target value (32-bit data) of the high-speed counter specified by the high-speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high-speed counter related instruction **F166_HC1S** or **F167_HC1R** is executed. The value can be read by executing an instruction.	A	N/A
DT90208 to DT90218 %MW5.90208 to %MW5.90218	Not used		——————	N/A	N/A
DT90219 %MW5.90219	Unit no. (station no.) selection for DT90220 to DT90251		0: Unit no. (station no.) 1 to 8, 1: Unit no. (station no.) 9 to 16	A	N/A

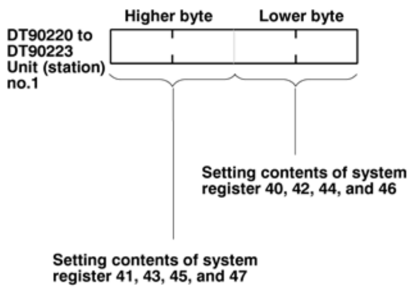
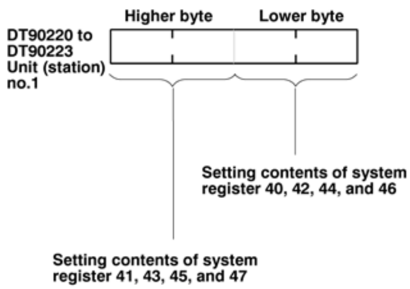
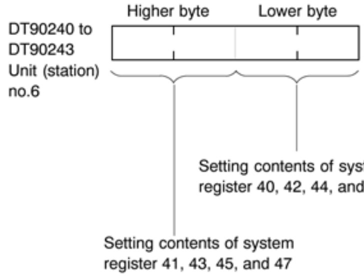
FP Address IEC Address	Name		Description	Read	Write
DT90220 **%MW5.90220**	PLC link unit (station) no. 1 or 9	System register 40 and 41	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A
DT90221 **%MW5.90221**	PLC link unit (station) no. 1 or 9	System register 42 and 43	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A
DT90222 **%MW5.90222**	PLC link unit (station) no. 1 or 9	System register 44 and 45	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90223 **%MW5.90223**	PLC link unit (station) no. 1 or 9	System register 46 and 47	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90224 **%MW5.90224**	PLC link unit (station) no. 2 or 10	System register 40 and 41	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90225 **%MW5.90225**	PLC link unit (station) no. 2 or 10	System register 42 and 43	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90226 **%MW5.90226**	PLC link unit (station) no. 2 or 10	System register 44 and 45	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90227 **%MW5.90227**	PLC link unit (station) no. 2 or 10	System register 46 and 47	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90228 **%MW5.90228**	PLC link unit (station) no. 3 or 11	System register 40 and 41	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90229 %MW5.90229	PLC link unit (station) no. 3 or 11	System register 42 and 43	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A
DT90230 %MW5.90230	PLC link unit (station) no. 3 or 11	System register 44 and 45	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A
DT90231 %MW5.90231	PLC link unit (station) no. 3 or 11	System register 46 and 47	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0  	A	N/A

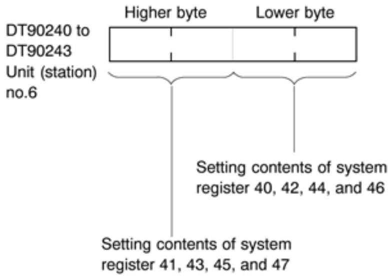
FP Address IEC Address	Name		Description	Read	Write
DT90232 **%MW5.90232**	PLC link unit (station) no. 4 or 12	System register 40 and 41	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90233 **%MW5.90233**	PLC link unit (station) no. 4 or 12	System register 42 and 43	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90234 **%MW5.90234**	PLC link unit (station) no. 4 or 12	System register 44 and 45	The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

<table>
<tr><th>FP Address
IEC Address</th><th colspan="2">Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90235
%MW5.90235</td><td>PLC link unit (station) no. 4 or 12</td><td>System register 46 and 47</td><td>The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0

</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90236
%MW5.90236</td><td>PLC link unit (station) no. 5 or 13</td><td>System register 40 and 41</td><td>The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0

</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90237
%MW5.90237</td><td>PLC link unit (station) no. 5 or 13</td><td>System register 42 and 43</td><td>The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0

</td><td>A</td><td>N/A</td></tr>
</table>

<table>
<tr><th>FP Address
IEC Address</th><th colspan="2">Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90238
%MW5.90238</td><td>PLC link unit (station) no. 5 or 13</td><td>System register 44 and 45</td><td>The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below.

Example:

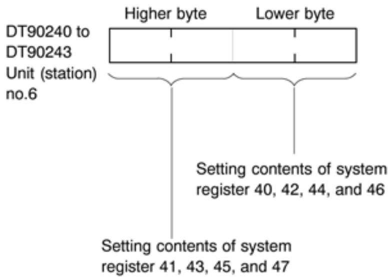
When DT90219 is 0



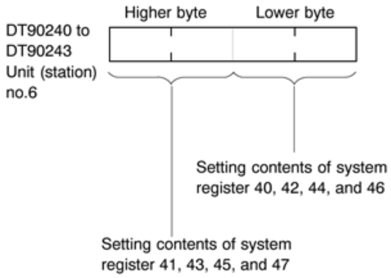
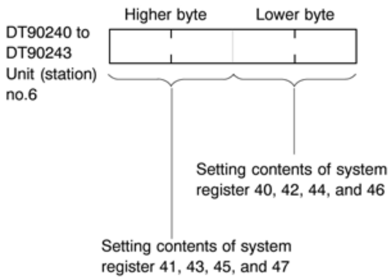
</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90239
%MW5.90239</td><td>PLC link unit (station) no. 5 or 13</td><td>System register 46 and 47</td><td>The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below.

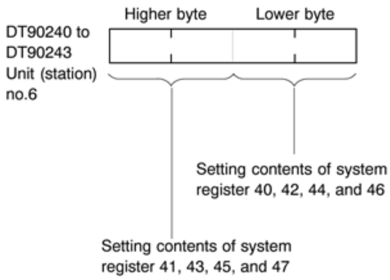
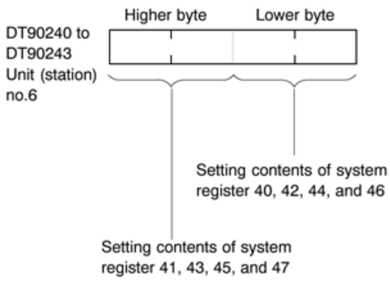
Example:

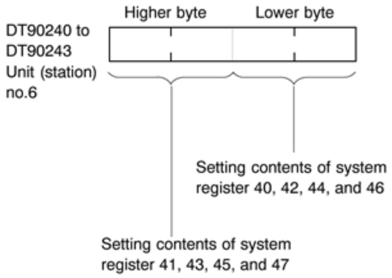
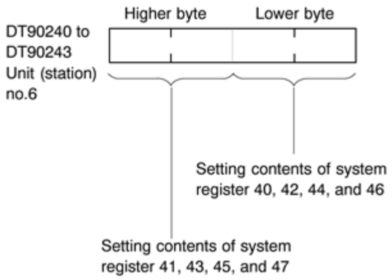
When DT90219 is 0

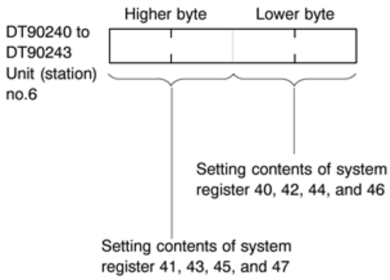
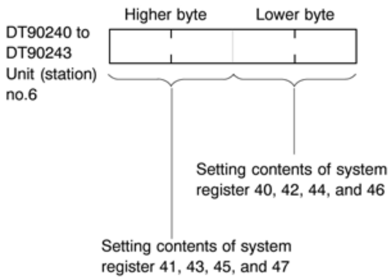


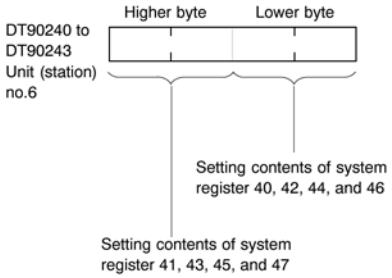
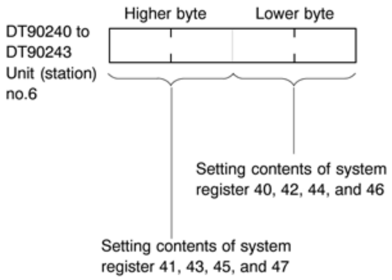
</td><td>A</td><td>N/A</td></tr>
</table>

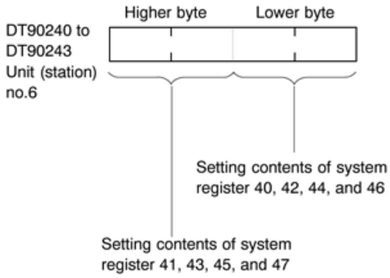
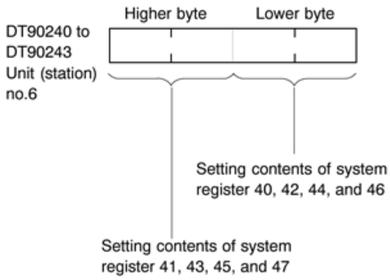
FP Address **IEC Address**	**Name**		**Description**	**Read**	**Write**
DT90240 **%MW5.90240**	PLC link unit (station) no. 6 or 14	System register 40 and 41	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90241 **%MW5.90241**	PLC link unit (station) no. 6 or 14	System register 42 and 43	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90242 **%MW5.90242**	PLC link unit (station) no. 6 or 14	System register 44 and 45	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90243 **%MW5.90243**	PLC link unit (station) no. 6 or 14	System register 46 and 47	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90244 **%MW5.90244**	PLC link unit (station) no. 7 or 15	System register 40 and 41	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90245 **%MW5.90245**	PLC link unit (station) no. 7 or 15	System register 42 and 43	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

<table>
<tr><th>FP Address
IEC Address</th><th colspan="2">Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90246
%MW5.90246</td><td>PLC link
unit (station) no. 7 or 15</td><td>System register 44 and 45</td><td>The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0

</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90247
%MW5.90247</td><td>PLC link
unit (station) no. 7 or 15</td><td>System register 46 and 47</td><td>The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0

</td><td>A</td><td>N/A</td></tr>
</table>

FP Address IEC Address	Name		Description	Read	Write
DT90248 **%MW5.90248**	PLC link unit (station) no. 8 or 16	System register 40 and 41	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90249 **%MW5.90249**	PLC link unit (station) no. 8 or 16	System register 42 and 43	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A

FP Address IEC Address	Name		Description	Read	Write
DT90250 **%MW5.90250**	PLC link unit (station) no. 8 or 16	System register 44 and 45	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90251 **%MW5.90251**	PLC link unit (station) no. 8 or 16	System register 46 and 47	The contents of the system register settings pertaining to the PLC interlink function for the various unit numbers are stored as shown below. **Example:** When DT90219 is 0 	A	N/A
DT90252 **to DT90255** **%MW5.90252** **to** **%MW5.90255**	Not used		———	N/A	N/A
DT90256 **%MW5.90256**	Unit no. (station no.) switch monitor for COM port		Used by the system.	N/A	N/A

32.6.2 Special Data Registers for FP-X

Special data registers are one word (16-bit) memory areas which store specific information.

(A: Available, N/A: Not available)

FP Address IEC Address	Name	Description	Read	Write
DT90000 **%MW5.90000**	Self-diagnostic error code	The self-diagnostic error code is stored here when a self-diagnostic error occurs.	A	N/A
DT90001 **%MW5.90001**	Not used	—	N/A	N/A
DT90002 **%MW5.90002**	Position of abnormal I/O board for application cassette	When an error occurs at the I/O board for the application cassette, the bit corresponding to the board will turn on. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (expansion no.) on: error, off: normal	A	N/A
DT90003 **%MW5.90003**	Not used	—	N/A	N/A
DT90004 **%MW5.90004**	Not used	—	N/A	N/A
DT90005 **%MW5.90005**	Not used	—	N/A	N/A
DT90006 **%MW5.90006**	Position of abnormal application cassette	When an error occurs at the intelligent board for the application cassette, the bit corresponding to the board will turn on. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (expansion no.) on: error, off: normal	A	N/A
DT90007 **%MW5.90007**	Not used	—	N/A	N/A
DT90008 **%MW5.90008**	Not used	—	N/A	N/A
DT90009 **%MW5.90009**	Communication error flag for COM 2	Stores the error contents when using COM 2 port.	A	N/A
DT90010 **%MW5.90010**	Extension I/O verify error unit	When the state of installation of FP-X expansion I/O unit has changed since the power was turned on, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 6 5 4 3 2 1 0 (bit no.) 7 6 5 4 3 2 1 0 (unit no.) on: error, off: normal	A	N/A

FP Address IEC Address	Name	Description	Read	Write
DT90011 **%MW5.90011**	Add-on cassette verify error unit	When the state of installation of an FP-X add-on cassette has changed since the power was turned on, the bit corresponding to the unit no. will turn on. Monitor using binary display. 15 11 7 3 2 1 0 (bit no.) 3 2 1 0 (expansion no.) on: error, off: normal	A	N/A
DT90012 **%MW5.90012**	Not used	—	N/A	N/A
DT90013 **%MW5.90013**	Not used	—	N/A	N/A
DT90014 **%MW5.90014**	Operation auxiliary register for data shift instruction	One shift-out hexadecimal digit is stored in bit positions 0 to 3 when the data shift instruction, **F105_BSR** or **F106_BSL** is executed. The value can be read and written by executing **F0_MV** instruction.	A	A
DT90015 **%MW5.90015**	Operation auxiliary register for division instruction	The divided remainder (16-bit) is stored in DT90015 when the division instruction **F32_%** or **F52_B%** instruction is executed. The divided remainder (32-bit) is stored in DT90015 and DT90016 when the division instruction **F33_D%** or **F53_DB%** is executed. The value can be read and written by executing the **F0_MV** instruction.	A	A
DT90016 **%MW5.90016**			A	A
DT90017 **%MW5.90017**	Operation error address (hold type)	After commencing operation, the address where the first operation error occurred is stored. Monitor the address using decimal display.	A	N/A
DT90018 **%MW5.90018**	Operation error address (non-hold type)	The address where an operation error occurred is stored. Each time an error occurs, the new address overwrites the previous address. At the beginning of a scan, the address is 0. Monitor the address using decimal display.	A	N/A
DT90019 **%MW5.90019**	2.5ms ring counter (see note)	The data stored here is increased by one every 2.5ms. (H0 to HFFFF) Difference between the values of the two points (absolute value) x 2.5ms = elapsed time between the two points.	A	N/A
DT90020 **%MW5.90020**	10µs ring counter (see note)	The data stored here is increased by one every 10.24µs. (H0 to HFFFF) Difference between the values of the two points (absolute value) x 10.24µs = elapsed time between the two points. **Note:** The exact value is 10.24µs.	A	N/A
DT90021 **%MW5.90021**	Not used	—	N/A	N/A

◆ **NOTE**

It is renewed once at the beginning of each one scan.

(A: Available, N/A: Not available)

FP Address IEC Address	Name	Description	Read	Write
DT90022 **%MW5.90022**	Scan time (current value) (see note)	The current scan time is stored here. The scan time is calculated using the formula: Scan time (ms) = stored data (decimal) x 0.1ms Example: 50 indicates 5ms.	A	N/A
DT90023 **%MW5.90023**	Scan time (minimum value) (see note)	The minimum scan time is stored here. Scan time is calculated using the formula: Scan time (ms) = stored data (decimal) x 0.1 ms Example: K50 indicates 5 ms.	A	N/A
DT90024 **%MW5.90024**	Scan time (maximum value) (see note)	The maximum scan time is stored here. The scan time is calculated using the formula: Scan time (ms) = stored data (decimal) x 0.1ms Example: 125 indicates 12.5ms.	A	N/A
DT90025 **%MW5.90025**	Mask condition monitoring register for interrupts (INT 0 to 13)	The mask conditions of interrupts using the **ICTL** instruction is stored here. Monitor using binary display. 15 13 11 7 3 0 (Bit no.) 13 11 7 3 0 (INT no.) 0: interrupt disabled (masked) 1: interrupt enabled (unmasked)	A	N/A
DT90026 **%MW5.90026**	Not used	—	N/A	N/A
DT90027 **%MW5.90027**	Periodical interrupt interval (INT24)	The value set by the **ICTL** instruction is stored. 0: periodical interrupt is not used 1 to 3000: 0.5ms to 1.5s or 10ms to 30s	A	N/A
DT90028 **%MW5.90028**	Not used	—	N/A	N/A
DT90029 **%MW5.90029**	Not used	—	N/A	N/A
DT90030 **%MW5.90030**	Message 0	The contents of the specified message are stored in these special data registers when the **F149_MSG** instruction is executed.	A	N/A
DT90031 **%MW5.90031**	Message 1			
DT90032 **%MW5.90032**	Message 2			
DT90033 **%MW5.90033**	Message 3			
DT90034 **%MW5.90034**	Message 4			
DT90035 **%MW5.90035**	Message 5			
DT90036 **%MW5.90036**	Not used	—	N/A	N/A

◆ **NOTE**

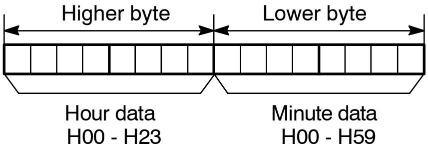
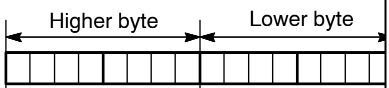
Scan time display is only possible in RUN mode and shows the operation cycle time. (In PROG mode, the scan time for the operation is not displayed.) The

maximum and minimum values are cleared each time the mode is switched from RUN to PROG.

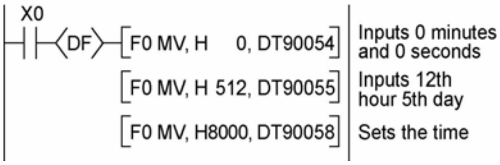
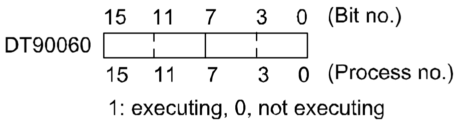
(A: Available, N/A: Not available)

FP Address IEC Address	Name	Description	Read	Write
DT90037 %MW5.90037	Operation auxiliary register for search instruction **F96_SRC**	The number of data that match the searched data is stored here when the **F96_SRC** instruction is executed.	A	N/A
DT90038 %MW5.90038	Operation auxiliary register for search instruction **F96_SRC**	The position of the first matching data is stored here when the **F96_SRC** instruction is executed.	A	N/A
DT90039 %MW5.90039	Not used	—	N/A	N/A
DT90040 %MW5.90040	Potentiometer (volume) input V0	The potentiometer value (0 to 1000) is stored here. This value can be used in analog timers and other applications by using the program to read this value to a data register. V0→DT90040 V1→DT90041	A	N/A
DT90041 %MW5.90041	Potentiometer (volume) input V1			
DT90042 %MW5.90042	Potentiometer (volume) input V2	For C60 only: The potentiometer value (0 to 1000) is stored here. This value can be used in analog timers and other applications by using the program to read this value to a data register. V0→DT90042 V1→DT90043	A	N/A
DT90043 %MW5.90043	Potentiometer (volume) input V3			
DT90044 %MW5.90044	Used by system	Used by the system.	A	A
DT90045 %MW5.90045	Not used	—	N/A	N/A
DT90046 %MW5.90046	Not used	—	N/A	N/A
DT90047 %MW5.90047	Not used	—	N/A	N/A
DT90048 %MW5.90048	Not used	—	N/A	N/A
DT90049 %MW5.90049	Not used	—	N/A	N/A
DT90050 %MW5.90050	Not used	—	N/A	N/A
DT90051 %MW5.90051	Not used	—	N/A	N/A

(A: Available, N/A: Not available)

<table>
<tr><th>FP Address
IEC Address</th><th>Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90052
%MW5.90052</td><td>High-speed counter and pulse output control flag</td><td>Used to reset the high-speed counter, disable counting, continue or clear high-speed counter instruction.
High-speed counter control code setting
Pulse output control code setting</td><td>N/A</td><td>A</td></tr>
<tr><td>DT90053
%MW5.90053</td><td>Clock/calendar monitor (hour/minute)</td><td>Hour and minute data of the clock/calendar are stored here. This data is read-only data. It cannot be overwritten.

</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90054
%MW5.90054</td><td>Clock/calendar setting (minute/second)</td><td rowspan="4">The year, month, day, hour, minute, second, and day-of-the-week data for the calendar timer is stored. The built-in calendar timer will operate correctly through the year 2099 and supports leap years. The calendar timer can be set by writing a value using a programming tool software or a programming instruction (see example for DT90058).

<table><tr><td>DT90054</td><td>Minute
H00 - H59</td><td>Second
H00 - H59</td></tr><tr><td>DT90055</td><td>Day
H01 - H31</td><td>Hour
H00 - H23</td></tr><tr><td>DT90056</td><td>Year
H00 - H99</td><td>Month
H01 - H12</td></tr><tr><td>DT90057</td><td>—</td><td>Day-of-the-week
H00 - H06</td></tr></table></td><td rowspan="4">A</td><td rowspan="4">A</td></tr>
<tr><td>DT90055
%MW5.90055</td><td>Clock/calendar setting (day/hour)</td></tr>
<tr><td>DT90056
%MW5.90056</td><td>Clock/calendar setting (year/month)</td></tr>
<tr><td>DT90057
%MW5.90057</td><td>Clock/calendar setting (day-of-the-week)</td></tr>
</table>

(A: Available, N/A: Not available)

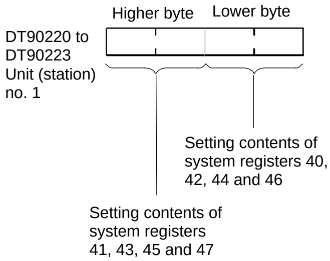
FP Address IEC Address	Name	Description	Read	Write
DT90058 **%MW5.90058**	Clock/calendar time setting and 30 seconds correction register	By setting the highest bit of DT90058 to 1, the time will be set according to the values written to DT90054 to DT90057. After the time is set, DT90058 is cleared to 0. **FPWIN Pro:** You can conveniently set the real-time clock using the **SET_RTC_DTBCD** instruction (see online help). **FPWIN GR:** Use the F0 (MV) instruction to set the corresponding data registers. **Example**: Set the time to 12:00:00 on the 5th day when X0 turns on.  **Note:** If the values of DT90054 to DT90057 are changed with the programming tool software, the time will be set when the new values are written. Therefore, it is unnecessary to write to DT90058. **When correcting times of less than 30 seconds** By setting the lowest bit of DT90058 to 1, the "seconds" value will be rounded off to the nearest minute. 	A	A
DT90059 **%MW5.90059**	Serial communication error code	Error code is stored here when a communication error occurs.	N/A	N/A
DT90060 to DT90122 **%MW5.90060 to %MW5.90122**	Step ladder process (0 to 999)	Indicates the startup condition of the step ladder process. When the process starts, the bit corresponding to the process number turns on. Monitor using binary display. **Example:** 		
DT90123 **%MW5.90123**	Not used	—	N/A	N/A

(A: Available, N/A: Not available)

FP Address IEC Address	Name	Description	Read	Write
DT90124 **%MW5.90124**	SEND/RECV end code for COM1 port	For details, refer to the programming manual or online help for the F145 and F146 instructions.	N/A	N/A
DT90125 **%MW5.90125**	SEND/RECV end code for COM2 port	For details, refer to the programming manual or online help for the F145 and F146 instructions.	N/A	N/A
DT90126 **%MW5.90126**	Forced ON/OFF operating station display	Used by the system	N/A	N/A
DT90127 to DT90139 **%MW5.90127 to %MW5.90139**	Not used	—	N/A	N/A
DT90140 **%MW5.90140**	MEWNET-W0 PLC link 0 status	The number of times the receiving operation is performed.	A	N/A
DT90141 **%MW5.90141**	MEWNET-W0 PLC link 0 status	The current interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90142 **%MW5.90142**	MEWNET-W0 PLC link 0 status	The minimum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90143 **%MW5.90143**	MEWNET-W0 PLC link 0 status	The maximum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90144 **%MW5.90144**	MEWNET-W0 PLC link 0 status	The number of times the sending operation is performed.	A	N/A
DT90145 **%MW5.90145**	MEWNET-W0 PLC link 0 status	The current interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90146 **%MW5.90146**	MEWNET-W0 PLC link 0 status	The minimum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90147 **%MW5.90147**	MEWNET-W0 PLC link 0 status	The maximum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90148 **%MW5.90148**	MEWNET-W0 PLC link 1 status	The number of times the receiving operation is performed.	A	N/A
DT90149 **%MW5.90149**	MEWNET-W0 PLC link 1 status	The current interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90150 **%MW5.90150**	MEWNET-W0 PLC link 1 status	The minimum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90151 **%MW5.90151**	MEWNET-W0 PLC link 1 status	The maximum interval between two receiving operations: value in the register × 2.5ms	A	N/A
DT90152 **%MW5.90152**	MEWNET-W0 PLC link 1 status	The number of times the sending operation is performed.	A	N/A
DT90153 **%MW5.90153**	MEWNET-W0 PLC link 1 status	The current interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90154 **%MW5.90154**	MEWNET-W0 PLC link 1 status	The minimum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90155 **%MW5.90155**	MEWNET-W0 PLC link 1 status	The maximum interval between two sending operations: value in the register × 2.5ms	A	N/A
DT90156 **%MW5.90156**	MEWNET-W0 PLC link 0 status	Area used for measurement of receiving interval.	A	N/A

FP Address IEC Address	Name	Description	Read	Write
DT90157 %MW5.90157	MEWNET-W0 PLC link 0 status	Area used for measurement of sending interval.	A	N/A
DT90158 %MW5.90158	MEWNET-W0 PLC link 1 status	Area used for measurement of receiving interval.	A	N/A
DT90159 %MW5.90159	MEWNET-W0 PLC link 1 status	Area used for measurement of sending interval.	A	N/A
DT90160 %MW5.90160	MEWNET-W0 PLC link 0 unit no.	Stores the unit no. of a PLC link	A	N/A
DT90161 %MW5.90161	MEWNET-W0 PLC link 0 error flag	Stores the error contents of a PLC link	A	N/A
DT90162 to DT90169 %MW5.90162 to %MW5.90169	Not used	—	N/A	N/A
DT90170 %MW5.90170	MEWNET-W0 PLC link 1 status	Duplicated destination for PLC inter-link address.	A	N/A
DT90171 %MW5.90171	MEWNET-W0 PLC link 1 status	Counts how many times a token is lost.	A	N/A
DT90172 %MW5.90172	MEWNET-W0 PLC link 1 status	Counts how many times two or more tokens are detected.	A	N/A
DT90173 %MW5.90173	MEWNET-W0 PLC link 1 status	Counts how many times a signal is lost.	A	N/A
DT90174 %MW5.90174	MEWNET-W0 PLC link 1 status	No. of times undefined commands have been received	A	N/A
DT90175 %MW5.90175	MEWNET-W0 PLC link 1 status	No. of times sum check errors have occurred during reception.	A	N/A
DT90176 %MW5.90176	MEWNET-W0 PLC link 1 status	No. of times format errors have occurred in received data.	A	N/A
DT90177 %MW5.90177	MEWNET-W0 PLC link 1 status	No. of times transmission errors have occurred.	A	N/A
DT90178 %MW5.90178	MEWNET-W0 PLC link 1 status	No. of times procedural errors have occurred.	A	N/A
DT90179 %MW5.90179	MEWNET-W0 PLC link 1 status	No. of times overlapping master units have occurred.	A	N/A
DT90180 to DT90218 %MW5.90180 to %MW5.90218	Not used	—	N/A	N/A

(A: Available, N/A: Not available)

<table>
<tr><th>FP Address
IEC Address</th><th colspan="2">Name</th><th>Description</th><th>Read</th><th>Write</th></tr>
<tr><td>DT90219
%MW5.90219</td><td colspan="2">Unit no. (station no.) selection for DT90220 to DT90251</td><td>0: Unit no. (station no.) 1 to 8,
1: Unit no. (station no.) 9 to 16</td><td>A</td><td>N/A</td></tr>
<tr><td>DT90220
%MW5.90220</td><td rowspan="4">PLC link unit (station) no. 1 or 9</td><td>System registers 40 and 41</td><td rowspan="20">The contents of the system register settings pertaining to the PLC inter-link function for the various unit numbers are stored as shown below.
Example:
When DT90219 is 0


System register 46 (see "Table of System Registers for FP-X" on page 850) of each respective station determines which block of data is transferred:
Normal: PLC link 0, as defined by system registers 40 - 45 and 47.
Reverse: PLC link 1, as defined by system registers 50 - 55 and 57.</td><td rowspan="20">A</td><td rowspan="20">A</td></tr>
<tr><td>DT90221
%MW5.90221</td><td>System registers 42 and 43</td></tr>
<tr><td>DT90222
%MW5.90222</td><td>System registers 44 and 45</td></tr>
<tr><td>DT90223
%MW5.90223</td><td>System registers 46 and 47</td></tr>
<tr><td>DT90224
%MW5.90224</td><td rowspan="4">PLC link unit (station) no. 2 or 10</td><td>System registers 40 and 41</td></tr>
<tr><td>DT90225
%MW5.90225</td><td>System registers 42 and 43</td></tr>
<tr><td>DT90226
%MW5.90226</td><td>System registers 44 and 45</td></tr>
<tr><td>DT90227
%MW5.90227</td><td>System registers 46 and 47</td></tr>
<tr><td>DT90228
%MW5.90228</td><td rowspan="4">PLC link unit (station) no. 3 or 11</td><td>System registers 40 and 41</td></tr>
<tr><td>DT90229
%MW5.90229</td><td>System registers 42 and 43</td></tr>
<tr><td>DT90230
%MW5.90230</td><td>System registers 44 and 45</td></tr>
<tr><td>DT90231
%MW5.90231</td><td>System registers 46 and 47</td></tr>
<tr><td>DT90232
%MW5.90232</td><td rowspan="4">PLC link unit (station) no. 4 or 12</td><td>System registers 40 and 41</td></tr>
<tr><td>DT90233
%MW5.90233</td><td>System registers 42 and 43</td></tr>
<tr><td>DT90234
%MW5.90234</td><td>System registers 44 and 45</td></tr>
<tr><td>DT90235
%MW5.90235</td><td>System registers 46 and 47</td></tr>
<tr><td>DT90236
%MW5.90236</td><td rowspan="4">PLC link unit (station) no. 5 or 13</td><td>System registers 40 and 41</td></tr>
<tr><td>DT90237
%MW5.90237</td><td>System registers 42 and 43</td></tr>
<tr><td>DT90238
%MW5.90238</td><td>System registers 44 and 45</td></tr>
<tr><td>DT90239
%MW5.90239</td><td>System registers 46 and 47</td></tr>
</table>

(A: Available, N/A: Not available)

FP Address IEC Address	Name		Description	Read	Write
DT90240 %MW5.90240	PLC link unit (station) no. 6 or 14	System registers 40 and 41	See previous table.	A	A
DT90241 %MW5.90241		System registers 42 and 43			
DT90242 %MW5.90242		System registers 44 and 45			
DT90243 %MW5.90243		System registers 46 and 47			
DT90244 %MW5.90244	PLC link unit (station) no. 5 or 15	System registers 40 and 41			
DT90245 %MW5.90245		System registers 42 and 43			
DT90246 %MW5.90246		System registers 44 and 45			
DT90247 %MW5.90247		System registers 46 and 47			
DT90248 %MW5.90248	PLC link unit (station) no. 8 or 16	System registers 40 and 41			
DT90249 %MW5.90249		System registers 42 and 43			
DT90250 %MW5.90250		System registers 44 and 45			
DT90251 %MW5.90251		System registers 46 and 47			
DT90252 to DT90256 %MW5.90252 to %MW5.90256	Not used		—	N/A	N/A

Concerning the special data registers for high-speed counting, DT90300 to DT90347 and pulse I/O, DT90348 to DT90355.

- They are all available for reading and writing.
- In FPWIN Pro, you have several convenient programming methods available to access the target or elapsed value areas, including assignment operation with system variables. Please refer to the online help for details.
- For FPWIN GR, use the F1 (DMV) instruction to write to the elapsed value. Use the F166 (HC1S) and F167 (HC1R) instructions to write to the target value area.

FP Address IEC Address	Name			Description
DT90300 %MW5.90300	Elapsed value area	Lower words	HSC-CH0	Counting area for input (X0) or (X0, X1) of the main unit.
DT90301 %MW5.90301		Higher words		
DT90302 %MW5.90302	Target	Lower words		The target value is set when instructions

FP Address IEC Address	**Name**			**Description**
DT90303 %MW5.90303	value area	Higher words		F166_HC1S and F167_HC1R are executed.
DT90304 %MW5.90304	Elapsed value area	Lower words	HSC-CH1	Counting area for input (X1) of the main unit.
DT90305 %MW5.90305		Higher words		
DT90306 %MW5.90306	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90307 %MW5.90307		Higher words		
DT90308 %MW5.90308	Elapsed value area	Lower words	HSC-CH2	Counting area for input (X2) or (X2, X3) of the main unit.
DT90309 %MW5.90309		Higher words		
DT90310 %MW5.90310	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90311 %MW5.90311		Higher words		
DT90312 %MW5.90312	Elapsed value area	Lower words	HSC-CH3	Counting area for input (X3) of the main unit.
DT90313 %MW5.90313		Higher words		
DT90314 %MW5.90314	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90315 %MW5.90315		Higher words		
DT90316 %MW5.90316	Elapsed value area	Lower words	HSC-CH4	Counting area for input (X4) or (X4, X5) of the main unit.
DT90317 %MW5.90317		Higher words		
DT90318 %MW5.90318	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90319 %MW5.90319		Higher words		

FP Address IEC Address	Name			Description
DT90320 %MW5.90320	Elapsed value area	Lower words	HSC-CH5	Counting area for input (X5) of the main unit.
DT90321 %MW5.90321		Higher words		
DT90322 %MW5.90322	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90323 %MW5.90323		Higher words		
DT90324 %MW5.90324	Elapsed value area	Lower words	HSC-CH6	Counting area for input (X6) or (X6, X7) of the main unit.
DT90325 %MW5.90325		Higher words		
DT90326 %MW5.90326	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90327 %MW5.90327		Higher words		
DT90328 %MW5.90328	Elapsed value area	Lower words	HSC-CH7	Counting area for input (X7) of the main unit.
DT90329 %MW5.90329		Higher words		
DT90330 %MW5.90330	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90331 %MW5.90331		Higher words		
DT90332 %MW5.90332	Elapsed value area	Lower words	HSC-CH8	Counting area for input (X0) or (X0, X1) of the main unit.
DT90333 %MW5.90333		Higher words		
DT90334 %MW5.90334	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90335 %MW5.90335		Higher words		
DT90336 %MW5.90336	Elapsed value area	Lower words	HSC-CH9	Counting area for input (X1 of the pulse I/O cassette.
DT90337 %MW5.90337		Higher words		
DT90338 %MW5.90338	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90339 %MW5.90339		Higher words		
DT90340 %MW5.90340	Elapsed value area	Lower words	HSC-CHA	Counting area for input (X3) or (X3, X4) of the pulse I/O cassette.
DT90341 %MW5.90341		Higher words		
DT90342 %MW5.90342	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90343 %MW5.90343		Higher words		

FP Address IEC Address	Name			Description
DT90344 **%MW5.90344**	Elapsed value area	Lower words	HSC-CHB	Counting area for input (X4) of the pulse I/O cassette.
DT90345 **%MW5.90345**		Higher words		
DT90346 **%MW5.90346**	Target value area	Lower words		The target value is set when instructions F166_HC1S and F167_HC1R are executed.
DT90347 **%MW5.90347**		Higher words		
DT90348 **%MW5.90348**	Elapsed value area	Lower words	PLS-CH0	Counting area for output (Y0, Y1) of the pulse I/O cassette.
DT90349 **%MW5.90349**		Higher words		
DT90350 **%MW5.90350**	Target value area	Lower words		The target value is set when instructions F171_SPDH, F172_PLSH, F174_SP0H and F175_SPSH are executed.
DT90351 **%MW5.90351**		Higher words		
DT90352 **%MW5.90352**	Elapsed value area	Lower words	PLS-CH1	Counting area for output (Y3, Y4) of the pulse I/O cassette.
DT90353 **%MW5.90353**		Higher words		
DT90354 **%MW5.90354**	Target value area	Lower words		The target value is set when instructions F171_SPDH, F172_PLSH, F174_SP0H and F175_SPSH are executed.
DT90355 **%MW5.90355**		Higher words		

(A: Available, N/A: Not available)

FP Address IEC Address	Name		Description	Read	Write
DT90356 to DT90359 %MW5.90356 to %MW5.90359	Not used.		—	N/A	N/A
DT90360 %MW5.90360	Control flag monitor area	HSC-CH0	When HSC control is executed and data is written to DT90052, the setting value for the target CH is stored in each CH.	A	N/A
DT90361 %MW5.90361		HSC-CH1			
DT90362 %MW5.90362		HSC-CH2			
DT90363 %MW5.90363		HSC-CH3			
DT90364 %MW5.90364		HSC-CH4			
DT90365 %MW5.90365		HSC-CH5			
DT90366 %MW5.90366		HSC-CH6			
DT90367 %MW5.90367		HSC-CH7			
DT90368 %MW5.90368		HSC-CH8			
DT90369 %MW5.90369		HSC-CH9			
DT90370 %MW5.90370		HSC-CHA			
DT90371 %MW5.90371		HSC-CHB			
DT90372 %MW5.90372		PLS-CH0			
DT90373 %MW5.90373		PLS-CH1			

32.7 Error Codes

32.7.1 General Information about Errors

32.7.1.1 FP-Series PLCs and ERROR Display

FP-Series PLCs' LEDs display errors in different ways.

Model	Display		Behavior
FP1, FP-M, FP2, FP3, FP10SH	LED	ERROR.	Continually lit
FP , FP0, FP-X	LED	ERROR/ALARM	Flashes/continually lit
FP-e	Screen display	ERR.	Continually lit

32.7.1.2 MEWTOCOL-COM Transmission Errors

These are error codes from a PC or other computer device that occur during an abnormal response when communicating with a PLC using MEWTOCOL-COM.

32.7.2 Table of Syntax Check Error

In FPWIN Pro, syntax errors are detected by the compiler and are therefore not critical.

Error code	Name	Operation status	Description and steps to take
E1	Syntax error	Stops	A program with a syntax error has been written. Change to PROG. mode and correct the error.
E2 (* Note)	Duplicated output error	Stops	Two or more OT(Out) instructions and KP(Keep) instructions are programmed using the same relay. Change to PROG. mode and correct the program so that one relay is not used for two or more OT instructions and KP instructions. Or, set the duplicated output to “enable (K1)” in system register 20.
E3	Not paired error	Stops	For instructions which must be used in a pair such as jump (JP and LBL), one instruction is either missing or in an incorrect position. Change to PROG. mode and enter the two instructions which must be used in a pair in the correct positions.
E4	Parameter mismatch error	Stops	An instruction has been written which does not agree with system register settings. For example, the number setting in a program does not agree with the timer/counter range setting. Change to PROG. mode, check the system register settings, and change so that the settings and the instruction agree.

Error code	Name	Operation status	Description and steps to take
E5 (* Note)	Program area error	Stops	An instruction which must be written to a specific area (main program area or subprogram area) has been written to a different area (for example, a subroutine SUB to RET is placed before an ED instruction). Change to PROG. mode and enter the instruction into the correct area.
E6	Compile memory full error (Available PLC: FPΣ/FP-X/ FP2SH/FP10SH)	Stops	The program stored in the FPΣ/FP2SH/FP10SH is too large to compile in the program memory. Change to PROG. mode and reduce the total number of steps for the program.
E7	High-level instruction type error (Available PLC: FPΣ/FP-X/ FP2/FP2SH/FP3/ FP10SH)	Stops	In the program, high-level instructions, which execute in every scan and at the leading edge of the trigger, are programmed to be triggered by one contact [e.g., F0 (MV) and P0 (PMV) are programmed using the same trigger continuously]. Correct the program so that the high-level instructions executed in every scan and only at the leading edge are triggered separately.
E8	High-level instruction operand error	Stops	There is an incorrect operand in an instruction which requires a specific combination operands (for example, the operands must all be of a certain type). Enter the correct combination of operands.
E9	No program error (Available PLC: FP2SH/FP10SH)	Stops	Program may be damaged. Try to send the program again.
E10	Rewrite during RUN syntax error	Continues	When inputting with the programming tool software, a deletion, addition or change of order of an instruction (ED, LBL, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU.

◆ **NOTE**

This error is also detected if you attempt to execute a rewrite containing a syntax error during RUN. In this case, nothing will be written to the CPU and operation will continue.

32.7.3 Table of Self-Diagnostic Errors

Not all errors apply to all PLCs.

E20 - E39

Error code	Name	Operation status	Description and steps to take
E20	CPU error	Stops	Probably a hardware abnormality. Please contact your dealer.
E21 E22 E23 E24 E25	RAM error	Stops	Probably an abnormality in the internal RAM. Please contact your dealer.
E26	User's ROM error	Stops	FP2, FP2SH, FP3, FP10SH: ROM is not installed. There may be a problem with the installed ROM. - ROM contents are damaged - Program size stored on the ROM is larger than the capacity of the ROM Check the contents of the ROM
			FP-X: If the master memory cassette is mounted, the master memor cassette may be damaged. Remove the master memory, and check whether the ERROR turns off. If the ERROR turned off, rewrite the master memory as its contents are damaged, and use it again. If the ERROR does not turn off, please contact your dealer.
			FP0, FP-e, FPΣ, FP1 C14, C16: Probably an abnormality in the built-in ROM. Please contact your dealer.
			All FP-Ms and FP1 C24, C40, C56, and C72: Probably an abnormality in the memory unit or master memory unit. Program the memory unit or master memory unit again and try to operate. If the same error is detected, try to operate with another memory unit or master memory unit.
E27	Intelligent unit installation error	Stops	Intelligent units installed exceed the limitations (i.e. 4 or more link units). Turn off the power and re-configure intelligent units referring to the hardware manual.
E28	System register error	Stops	Probably an abnormality in the system register. Check the system register setting or initialize the system registers.
E29	Configuration parameter error	Stops	A parameter error was detected in the MEWNET-W2 configuration area. Set a correct parameter.
E30	Interrupt error 0	Stops	Probably a hardware abnormality. Please contact your dealer.

Error code	Name	Operation status	Description and steps to take
E31	Interrupt error 1	Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
E32	Interrupt error 2	Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
			There is no interrupt program for an interrupt which occurred. Check the number of the interrupt program and change it to agree with the interrupt request.
E33	Multi-CPU data unmatch error	CPU2 stops	This error occurs when a FP3/FP10SH is used as CPU2 for a multi-CPU system. Please contact your dealer.
E34	I/O status error	Stops	An abnormal unit is installed. Check the contents of special data register DT9036/DT90036 and locate the abnormal unit. Then turn off the power and replace the unit with a new one.
E35	MEWNET-F (remote I/O) slave illegal unit error	Stops	A unit, which cannot be installed on the slave station of the MEWNET-F link system, is installed on the slave station. Remove the illegal unit from the slave station.
E36	MEWNET-F limitation error	Stops	The number of slots or I/O points used for MEWNET-F exceeds the limitation. Re-configure the system so that the number of slots and I/O points is within the specified range.
E37	MEWNET-F I/O mapping error	Stops	I/O overlap or I/O setting that is over the range is detected in the allocated I/O and MEWNET-F I/O map. Re-configure the I/O map correctly.
E38	MEWNET-F slave I/O mapping error	Stops	I/O mapping for remote I/O terminal boards, remote I/O terminal units and I/O link unit is not correct. Re-configure the I/O map for slave stations according to the I/O points of the slave stations.
E39	IC memory card read error	Stops	When reading in the program from the IC memory card (due to automatic reading because of the dip switch 3 setting or program switching due to F14 (PGRD) instruction): - IC memory card is not installed. - There is no program file or it is damaged. - Writing is disabled. - There is an abnormality in the AUTOEXEC.SPG file. - Program size stored on the card is larger than the capacity of the CPU. Install an IC memory card that has the program properly recorded and execute the read once again.

E40 and above

Error code	Name	Operation status	Description and steps to take
E40	I/O error	Selectable	With FP3/FP10SH, communication error in the MEWNET-TR system has occurred. For all other PLCs an abnormality in an I/O unit has been detected. Check the contents of special data registers DT9002 and DT9003/DT90002 and DT90003 and the erroneous MEWNET-TR master unit or abnormal I/O unit (also expansion unit or application cassette). Then check the unit. Selection of operation status using system register 21: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E41	Intelligent unit error	Selectable	An abnormality in an intelligent unit. Check the contents of special data registers DT9006 and DT9007/DT90006 and DT90007 and locate the abnormal intelligent unit. Then check the unit referring to its manual. Selection of operation status using system register 22: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E42	I/O unit verify error	Selectable	I/O unit wiring condition has changed compared to that at time of power-up. Check the contents of special data registers DT9010 and DT9011/DT90010 and DT90011 and locate the erroneous unit. Then check the unit and correct the wiring. Selection of operation status using system register 23: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E43	System watching dog timer error	Selectable	Scan time required for program execution exceeds the setting of the system watchdog timer. Check the program and modify it so that FP2SH/FP10SH can execute a scan within the specified time. Selection of operation status using system register 24: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E44	Slave station connecting time error for MEWNET-F system	Selectable	The time required for slave station connection exceeds the setting of the system register 35. Selection of operation status using system register 25: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)

Error code	Name	Operation status	Description and steps to take
E45	Operation error	Selectable	Operation became impossible when a high-level instruction was executed. Check the contents of special data registers DT9017 and DT9018/DT90017 and DT90018 to find the program address where the operation error occurred. Then correct the program. Refer to the explanation of operation error and the instruction. Selection of operation status using system register 26: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E46	Remote I/O communication error	Selectable	**MEWNET-F communication error:** A communication abnormally was caused by a transmission cable or during the power-down of a slave station. Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the slave condition. Selection of operation status using system register 27: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP) **S-Link communication error (with FP0-SL1 unit only):** When one of the S-LINK errors (ERR1, 3 or 4) has been deteced,error code E46 (remote I/O (S-LINK) communication error) is stored. Selection of operation status using system register 27: - to continue operation, set K1 (CONT) - to stop operation, set K0 (STOP)
E47	MEWNET-F attribute error	Selectable	MEWNET-F communication error A communication abnormally was caused by a transmission cable or during the power-down of a slave station. Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the communication condition. Selection of operation status using system register27: - to continue operation,set K1 - to stop operation, set K0
E50	Backup battery error	Continues	The voltage of the backup battery lowered or the backup battery of CPU is not installed. Check the installation of the backup battery and then replace battery if necessary. By setting the system register 4 in K0 (NO), you can disregard this error. However, the BATT. LED turns on.
E51	MEWNET-F terminal station error	Continues	Terminal station settings were not properly performed.Check stations at both ends of the communication path, and set them in the terminal station using the dip switches.
E52	MEWNET-F I/O update synchronous error	Continues	Set the INITIALIZE/TEST selector to the INITIALIZE position while keeping the mode selector in the RUN position. If the same error occurs after this, please contact your dealer.
E53	Multi-CPU registration error	Continues	Abnormality was detected when the multi-CPU system was used. Please contact your dealer.

Error code	Name	Operation status	Description and steps to take
E54	IC memory card backup battery error	Continues	The voltage of the backup battery for the IC memory card is getting low. The BATT. LED does not turn on. Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.)
E55	IC memory card backup battery error	Continues	The voltage of the backup battery for IC memory card is getting low. The BATT. LED does not turn on. Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.)
E56	Incompatible IC memory card error	Continues	The IC memory card installed is not compatible with FP2SH/FP10SH. Replace the IC memory card compatible with FP2SH/FP10SH.
E57	No unit for the configuration	Continues	MEWNET-W2 The MEWNET-W2 link unit is not installed in the slot specified using the configuration data. Either install a unit in the specified slot or change the parameter.
E100 to E199	Self- diagnostic error set by F148 (ERR)/ P148 (PERR) instruction	Stops	The self-diagnostic error specified by the F148 (ERR)/P148 (PERR) instruction is occurred. Take steps to clear the error condition according to the specification you chose.
E200 to E299		Continues	

32.7.4 MEWTOCOL-COM Error Codes

Error code	Name	Description
!21	NACK error	Link system error
!22	WACK error	Link system error
!23	Unit no. overlap	Link system error
!24	Transmission format error	Link system error
!25	Link unit hardware error	Link system error
!26	Unit no. setting error	Link system error
!27	No support error	Link system error
!28	No response error	Link system error
!29	Buffer closed error	Link system error
!30	Time-out error	Link system error
!32	Transmission impossible error	Link system error
!33	Communication stop	Link system error
!36	No destination error	Link system error
!38	Other communication error	Link system error
!40	BCC error	A transfer error occurred in the data received.

Error code	Name	Description
!41	Format error	A formatting error in the command received was detected.
!42	No support error	A non-supported command was received.
!43	Multiple frames procedure error	A different command was received when processing multiple frames.
!50	Link setting error	A non-existing route number was specified. Verify the route number by designating the transmission station.
!51	Transmission time-out error	Transmission to another device is not possible because the transmission buffer is full.
!52	Transmit disable error	Transmission processing to another device is not possible (link unit runaway, etc.).
!53	Busy error	Processing of command received is not possible because of multiple frame processing or because command being processed is congested.
!60	Parameter error	Content of specified parameter does not exist or cannot be used.
!61	Data error	There was a mistake in the contact, data area, data number designation, size designation, range, or format designation.
!62	Registration over error	Operation was done when number of registrations was exceeded or when there was no registration.
!63	PC mode error	PC command that cannot be processed was executed during RUN mode.
!64	External memory error	An abnormality occurred when loading RAM to ROM/IC memory card. There may be a problem with the ROM or IC memory card. When loading, the specified contents exceeded the capacity. Write error occurs. -ROM or IC memory card is not installed. -ROM or IC memory card does not conform to specifications -ROM or IC memory card board is not installed.
!65	Protect error	A program or system register write operation was executed when the protect mode (password setting or DIP switch, etc.) or ROM operation mode was being used.
!66	Address error	There was an error in the code format of the address data. Also, when exceeded or insufficient address data, there was a mistake in the range designation.
!67	No program error and no data error	Cannot be read because there is no program in the program area or the memory contains an error. Or, reading of non-registered data was attempted.
!68	Rewrite during RUN error	When inputting with programming tool software, editing of an instruction (ED, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU.
!70	SIM over error	Program area was exceeded during a program write process.
!71	Exclusive access control error	A command that cannot be processed was executed at the same time as a command being processed.

32.8 MEWTOCOL-COM Communication Commands

Command name	Code	Description
Read contact area	RC (RCS) (RCP) (RCC)	Reads the on and off status of contacts. Specifies only one point. Specifies multiple contacts. Specifies a range in word units.
Write contact area	WC (WCS) (WCP) (WCC)	Turns contacts on and off. Specifies only one point. Specifies multiple contacts. Specifies a range in word units.
Read data area	RD	Reads the contents of a data area.
Write data area	WD	Writes data to a data area.
Read timer/counter set value area	RS	Reads the value set for a timer/counter.
Write timer/counter set value area	WS	Writes a timer/counter setting value.
Read timer/counter elapsed value area	RK	Reads the timer/counter elapsed value.
Write timer/counter elapsed value area	WK	Writes the timer/counter elapsed value.
Register or Reset contacts monitored	MC	Registers the contact to be monitored.
Register or Reset data monitored	MD	Registers the data to be monitored.
Monitoring start	MG	Monitors a registered contact or data.
Preset contact area (fill command)	SC	Embeds the area of a specified range in a 16-point on and off pattern.
Preset data area (fill command)	SD	Writes the same contents to the data area of a specified range.
Read system register	RR	Reads the contents of a system register.
Write system register	WR	Specifies the contents of a system register.
Read the status of PLC	RT	Reads the specifications of the programmable controller and error codes if an error occurs.
Remote control	RM	Switches the operation mode of the programmable controller.
Abort	AB	Aborts communication.

32.9 Hexadecimal/Binary/BCD

Decimal	Hexadecimal	Binary data	BCD data (Binary Coded Decimal)
0	0000	0000 0000 0000 0000	0000 0000 0000 0000
1	0001	0000 0000 0000 0001	0000 0000 0000 0001
2	0002	0000 0000 0000 0010	0000 0000 0000 0010
3	0003	0000 0000 0000 0011	0000 0000 0000 0011
4	0004	0000 0000 0000 0100	0000 0000 0000 0100
5	0005	0000 0000 0000 0101	0000 0000 0000 0101
6	0006	0000 0000 0000 0110	0000 0000 0000 0110
7	0007	0000 0000 0000 0111	0000 0000 0000 0111
8	0008	0000 0000 0000 1000	0000 0000 0000 1000
9	0009	0000 0000 0000 1001	0000 0000 0000 1001
10	000A	0000 0000 0000 1010	0000 0000 0001 0000
11	000B	0000 0000 0000 1011	0000 0000 0001 0001
12	000C	0000 0000 0000 1100	0000 0000 0001 0010
13	000D	0000 0000 0000 1101	0000 0000 0001 0011
14	000E	0000 0000 0000 1110	0000 0000 0001 0100
15	000F	0000 0000 0000 1111	0000 0000 0001 0101
16	0010	0000 0000 0001 0000	0000 0000 0001 0110
17	0011	0000 0000 0001 0001	0000 0000 0001 0111
18	0012	0000 0000 0001 0010	0000 0000 0001 1000
19	0013	0000 0000 0001 0011	0000 0000 0001 1001
20	0014	0000 0000 0001 0100	0000 0000 0010 0000
21	0015	0000 0000 0001 0101	0000 0000 0010 0001
22	0016	0000 0000 0001 0110	0000 0000 0010 0010
23	0017	0000 0000 0001 0111	0000 0000 0010 0011
24	0018	0000 0000 0001 1000	0000 0000 0010 0100
25	0019	0000 0000 0001 1001	0000 0000 0010 0101
26	001A	0000 0000 0001 1010	0000 0000 0010 0110
27	001B	0000 0000 0001 1011	0000 0000 0010 0111
28	001C	0000 0000 0001 1100	0000 0000 0010 1000
29	001D	0000 0000 0001 1101	0000 0000 0010 1001
30	001E	0000 0000 0001 1110	0000 0000 0011 0000
31	001F	0000 0000 0001 1111	0000 0000 0011 0001
.	.	.	.
.	.	.	.
.	.	.	.
63	**003F**	0000 0000 0011 1111	0000 0000 0110 0011
.	.	.	.
.	.	.	.
.	.	.	.
255	**00FF**	0000 0000 1111 1111	0000 0010 0101 0101
.	.	.	.
.	.	.	.
.	.	.	.
9999	**270F**	0010 0111 0000 1111	1001 1001 1001 1001

32.10 ASCII Codes

									b7								
									b6	0	0	0	0	1	1	1	1
									b5	0	0	1	1	0	0	1	1
									b4	0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		ASCII HEX code	Most significant digit 0	1	2	3	4	5	6	7
				0	0	0	0	Least significant digit	0	NUL	DEL	SPACE	0	@	P		p
				0	0	0	1		1	SOH	DC1	!	1	A	Q	a	q
				0	0	1	0		2	STX	DC2	”	2	B	R	b	r
				0	0	1	1		3	ETX	DC3	#	3	C	S	c	s
				0	1	0	0		4	EOT	DC4	$	4	D	T	d	t
				0	1	0	1		5	ENQ	NAK	%	5	E	U	e	u
				0	1	1	0		6	ACK	SYN	&	6	F	V	f	v
				0	1	1	1		7	BEL	ETB	’	7	G	W	g	w
				1	0	0	0		8	BS	CAN	(	8	H	X	h	x
				1	0	0	1		9	HT	EM	)	9	I	Y	i	y
				1	0	1	0		A	LF	SUB	*	:	J	Z	j	z
				1	0	1	1		B	VT	ESC	+	;	K	[	k	{
				1	1	0	0		C	FF	FS	,	<	L	\	l	「
				1	1	0	1		D	CR	GS	–	=	M	]	m	}
				1	1	1	0		E	SO	RS	.	>	N	^	n	~
				1	1	1	1		F	SI	US	/	?	O	_	o	DEL

32.11 Availability of All Instructions on All PLC Types

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
ABS	x	x	x	x	
ACOS	x	x	x	x	
ActivateStepsOfStoppedSfc	x	x	x	x	
ADD	x	x	x	x	
ADD_TIME	x	x	x	x	
Adr_Of_Var	x	x	x	x	
Adr_Of_VarOffs	x	x	x	x	
AdrDT_Of_Offs	x	x	x	x	
AdrFL_Of_Offs	x	x	x	x	
AdrLast_Of_Var	x	x	x	x	
AllSfcsStopped	x	x	x	x	
ALT	x	x	x	x	
AND	x	x	x	x	
AreaOffs_OfVar	x	x	x	x	
AreaOffs_ToVar	x	x	x	x	
ASIN	x	x	x	x	
ATAN	x	x	x	x	
BCD_TO_DINT	x	x	x	x	
BCD_TO_INT	x	x	x	x	
BOOL_TO_DINT	x	x	x	x	
BOOL_TO_DWORD	x	x	x	x	
BOOL_TO_INT	x	x	x	x	
BOOL_TO_STRING	x	x	x	x	
BOOL_TO_WORD	x	x	x	x	
BOOL16_TO_INT	x	x	x	x	
BOOL16_TO_WORD	x	x	x	x	
BOOL32_TO_DINT	x	x	x	x	
BOOL32_TO_DWORD	x	x	x	x	
BOOLS_TO_DINT	x	x	x	x	
BOOLS_TO_DWORD	x	x	x	x	
BOOLS_TO_INT	x	x	x	x	
BOOLS_TO_WORD	x	x	x	x	
BRK	-	-	-	-	1
CONCAT	x	x	x	x	
ControlSfc	x	x	x	x	

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
ControlSfcAndData	x	x	x	x	
COS	x	x	x	x	
CRC16	x	x	x	x	
CT	x	x	x	x	
CT_FB	x	x	x	x	3
CTD	x	x	x	x	31
CTU	x	x	x	x	31
CTUD	x	x	x	x	66
DELETE	x	x	x	x	19
DF	x	x	x	x	1
DFI	x	x	x	x	1
DFN	x	x	x	x	1
DINT_TO_BCD	x	x	x	x	
DINT_TO_BOOL	x	x	x	x	
DINT_TO_BOOL32	x	x	x	x	
DINT_TO_BOOLS	x	x	x	x	
DINT_TO_DWORD	x	x	x	x	
DINT_TO_INT	x	x	x	x	
DINT_TO_REAL	x	x	x	x	
DINT_TO_SDDT	x	x	x	x	
DINT_TO_STRING	x	x	x	x	
DINT_TO_STRING _LEADING_ZEROS	x	x	x	x	
DINT_TO_TIME	x	x	x	x	
DINT_TO_WORD	x	x	x	x	
DIV	x	x	x	x	
DIV_TIME_DINT	x	x	x	x	
DIV_TIME_INT	x	x	x	x	
DIV_TIME_REAL	x	x	x	x	
DWORD_TO_BOOL	x	x	x	x	
DWORD_TO_BOOL32	x	x	x	x	
DWORD_TO_BOOLS	x	x	x	x	
DWORD_TO_DINT	x	x	x	x	
DWORD_TO_INT	x	x	x	x	
DWORD_TO_SDDT	x	x	x	x	
DWORD_TO_STRING	x	x	x	x	
DWORD_TO_TIME	x	x	x	x	
DWORD_TO_WORD	x	x	x	x	
Elem_OfArray1D	x	x	x	x	

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
Elem_OfArray2D	x	x	x	x	
Elem_OfArray3D	x	x	x	x	
EQ	x	x	x	x	
ETLANADDR_TO_STRING_NO_LEADING_ZEROS	x	x	x	x	
ETLANADDR_TO_STRING	x	x	x	x	
EXP	x	x	x	x	
EXPT	x	x	x	x	
F0_MV	x	x	x	x	5
F1_DMV	x	x	x	x	7
F2_MVN	x	x	x	x	5
F3_DMVN	x	x	x	x	7
F4_GETS	-	-	-	-	
F5_BTM	x	x	x	x	7
F6_DGT	x	x	x	x	7
F7_MV2	x	x	x	x	7
F8_DMV2	x	x	x	x	11
F10_BKMV	x	x	x	x	7
F10_BKMV_NUMBER	x	x	x	x	7
F10_BKMV_NUMBER_OFFSET	x	x	x	x	7
F10_BKMV_OFFSET	x	x	x	x	7
F11_COPY	x	x	x	x	7
F12_ICRD	-	-	-	-	11
F12_EPRD	x	x	x	x	11
F13_ICWT	-	-	-	-	11
F14_PGRD	-	-	-	-	3
F15_XCH	x	x	x	x	5
F16_DXCH	x	x	x	x	5
F17_SWAP	x	x	x	x	3
F18_BXCH	x	x	x	x	7
F19_SJP	-	-	-	-	3
F20_ADD	x	x	x	x	5
F21_DADD	x	x	x	x	7
F22_ADD2	x	x	x	x	7
F23_DADD2	x	x	x	x	11
F25_SUB	x	x	x	x	5

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F26_DSUB	x	x	x	x	7
F27_SUB2	x	x	x	x	7
F28_DSUB2	x	x	x	x	11
F30_MUL	x	x	x	x	7
F31_DMUL	x	x	x	x	11
F32_DIV	x	x	x	x	7
F33_DDIV	x	x	x	x	11
F34_MULW	x	x	x	x	7
F35_INC	x	x	x	x	3
F36_DINC	x	x	x	x	3
F37_DEC	x	x	x	x	3
F38_DDEC	x	x	x	x	3
F39_DMULD	x	x	x	x	11
F40_BADD	x	x	x	x	5
F41_DBADD	x	x	x	x	7
F42_BADD2	x	x	x	x	7
F43_DBADD2	x	x	x	x	11
F45_BSUB	x	x	x	x	5
F46_DBSUB	x	x	x	x	5
F47_BSUB2	x	x	x	x	7
F48_DBSUB2	x	x	x	x	11
F50_BMUL	x	x	x	x	7
F51_DBMUL	x	x	x	x	11
F52_BDIV	x	x	x	x	7
F53_DBDIV	x	x	x	x	11
F55_BINC	x	x	x	x	3
F56_DBINC	x	x	x	x	3
F57_BDEC	x	x	x	x	3
F58_DBDEC	x	x	x	x	3
F60_CMP	x	x	x	x	5
F61_DCMP	x	x	x	x	9
F62_WIN	x	x	x	x	7
F63_DWIN	x	x	x	x	13
F64_BCMP	x	x	x	x	7
F65_WAN	x	x	x	x	7
F66_WOR	x	x	x	x	7
F67_XOR	x	x	x	x	7
F68_XNR	x	x	x	x	7

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F69_WUNI	x	x	x	x	9
F70_BCC	x	x	x	x	9
F71_HEX2A	x	x	x	x	7
F72_A2HEX	x	x	x	x	7
F73_BCD2A	x	x	x	x	7
F74_A2BCD	x	x	x	x	9
F75_BIN2A	x	x	x	x	7
F76_A2BIN	x	x	x	x	7
F77_DBIN2A	x	x	x	x	11
F78_DA2BIN	x	x	x	x	11
F80_BCD	x	x	x	x	5
F81_BIN	x	x	x	x	5
F82_DBCD	x	x	x	x	7
F83_DBIN	x	x	x	x	7
F84_INV	x	x	x	x	5
F85_NEG	x	x	x	x	3
F86_DNEG	x	x	x	x	3
F87_ABS	x	x	x	x	3
F88_DABS	x	x	x	x	3
F89_EXT	x	x	x	x	3
F90_DECO	x	x	x	x	7
F91_SEGT	x	x	x	x	3
F92_ENCO	x	x	x	x	7
F93_UNIT	x	x	x	x	7
F94_DIST	x	x	x	x	7
F95_ASC	x	x	x	x	15
F96_SRC	x	x	x	x	7
F97_DSRC	x	x	x	x	9
F98_CMPR	x	x	x	x	7
F99_CMPW	x	x	x	x	7
F100_SHR	x	x	x	x	5
F101_SHL	x	x	x	x	5
F102_DSHR	x	x	x	x	5
F103_DSHL	x	x	x	x	5
F105_BSR	x	x	x	x	3
F106_BSL	x	x	x	x	3
F108_BITR	x	x	x	x	7
F109_BITL	x	x	x	x	7

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ 12k	FP-Σ 32k	Steps
F110_WSHR	x	x	x	x	5
F111_WSHL	x	x	x	x	5
F112_WBSR	x	x	x	x	5
F113_WBSL	x	x	x	x	5
F115_FIFT	x	x	x	x	5
F116_FIFR	x	x	x	x	5
F117_FIFW	x	x	x	x	5
F118_UDC	x	x	x	x	5
F119_LRSR	x	x	x	x	5
F120_ROR	x	x	x	x	5
F121_ROL	x	x	x	x	5
F122_RCR	x	x	x	x	5
F123_RCL	x	x	x	x	5
F125_DROR	x	x	x	x	5
F126_DROL	x	x	x	x	5
F127_DRCR	x	x	x	x	5
F128_DRCL	x	x	x	x	5
F130_BTS	x	x	x	x	5
F131_BTR	x	x	x	x	5
F132_BTI	x	x	x	x	5
F133_BTT	x	x	x	x	5
F135_BCU	x	x	x	x	5
F136_DBCU	x	x	x	x	7
F137_STMR	x	x	x	x	5
F138_TIMEBCD_TO_SECBCD	x	x	x	x	7
F139_SECBCD_TO_TIMEBCD	x	x	x	x	5
F140_STC	x	x	x	x	1
F141_CLC	x	x	x	x	1
F142_WDT	-			-	3
F143_IORF	x	x	x	x	5
F144_TRNS	x	x	x	x	
F145_SEND	-	-	-	-	9
F145_MODBUS_WRITE_DATA	x	x	-	x	
F146_MODBUS_READ_DATA	x	x	-	x	
F146_RECV	-	-	-	-	9

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F147_PR	x	x	x	x	5
F148_ERR	x	x	x	x	3
F149_MSG	x	x	x	x	13
F150_READ	x	x	x	x	9
F151_WRT	x	x	x	x	9
F152_RMRD	-	-	-	-	9
F153_RMWT	-	-	-	-	9
F154_MCAL	-	-	-	-	3
F154_MCAL_DUT	-	-	-	-	
F155_SMPL	-	-	-	-	1
F156_STRG	-	-	-	-	1
F157_ADD_DTBCD_TIMEBCD	x	x	x	x	9
F158_SUB_DTBCD_TIMEBCD	x	x	x	x	9
F159_MTRN explicitly supported by FP-Sigma and FP2/FP2SH Ver. 1.40 or later for other PLCs: F159_MTRN will be compiled to F144_TRNS	x	x	x	x	7
F159_MWRT_PARA for FP2/FP2SH Ver. 1.40 or later	-	-	-	-	7
F160_DSQR	x	x	x	x	7
F161_MRCV for FP2/FP2SH Ver. 1.40 or later for other PLCs: F161_MRCV will be ignored by the compiler	-	x	x	x	7
F161_MRD_PARA for FP2/FP2SH Ver. 1.40 or later	-	-	-	-	7
F161_MRD_STATUS for FP2/FP2SH Ver. 1.40 or later	-	-	-	-	7
F162_HC0S	-	-	-	-	7
F163_HC0R	-	-	-	-	7
F164_SPD0	-	-	-	-	3

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F165_CAM0	-	-	-	-	3
F166_HC1S	x	x	x	x	11
F167_HC1R	x	x	x	x	11
F168_SPD1	-	-	-	-	5
F169_PLS	-	-	-	-	5
F170_PWM	-	-	-	-	5
F171_SPDH	x	x	x	x	5
F172_PLSH	x	x	x	x	5
F173_PWMH	x	x	x	x	5
F174_SP0H	x	x	x	x	5
F175_SPSH_LINEAR	x	x	x	x	5
F176_SPCH_CENTER	x	-	x	x	5
F176_SPCH_PASS	x	-	x	x	
F180_SCR	-	-	-	-	9
F180_SCR_DUT	-	-	-	-	9
F181_DSP	-	-	-	-	3
F183_DSTM	x	x	x	x	7
F190_MV3	x	x	x	x	10
F191_DMV3	x	x	x	x	16
F215_DAND	x	x	x	x	12
F216_DOR	x	x	x	x	12
F217_DXOR	x	x	x	x	12
F218_DXNR	x	x	x	x	12
F219_DUNI	x	x	x	x	16
F230_DTBCD_TO_SEC	-	x	-	-	
F231_SEC_TO_DTBCD	-	x	-	-	
F235_GRY	x	x	x	x	6
F236_DGRY	x	x	x	x	8
F237_GBIN	x	x	x	x	6
F238_DGBIN	x	x	x	x	8
F240_COLM	x	x	x	x	8
F241_LINE	x	x	x	x	8
F270_MAX	x	x	x	x	8
F271_DMAX	x	x	x	x	8
F272_MIN	x	x	x	x	8
F273_DMIN	x	x	x	x	8
F275_MEAN	x	x	x	x	8

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F276_DMEAN	x	x	x	x	8
F277_SORT	x	x	x	x	8
F278_DSORT	x	x	x	x	8
F282_SCAL FP2: only from CPU version 1.06 onwards FP2SH/10SH: only from CPU version 3.04 onwards	x	x	x	x	8
F283_DSCAL	x	x	x	x	8
F285_LIMT	x	x	x	x	10
F286_DLIMT	x	x	x	x	10
F287_BAND	x	x	x	x	10
F288_DBAND	x	x	x	x	10
F289_ZONE	x	x	x	x	10
F290_DZONE	x	x	x	x	10
F300_BSIN	-	-	-	-	6
F301_BCOS	-	-	-	-	6
F302_BTAN	-	-	-	-	6
F303_BASIN	-	-	-	-	6
F304_BACOS	-	-	-	-	6
F305_BATAN	-	-	-	-	6
F309_FMV	x	x	x	x	8
F310_FADD	x	x	x	x	14
F311_FSUB	x	x	x	x	14
F312_FMUL	x	x	x	x	14
F313_FDIV	x	x	x	x	14
F314_SIN	x	x	x	x	10
F315_COS	x	x	x	x	10
F316_TAN	x	x	x	x	10
F317_ASIN	x	x	x	x	10
F318_ACOS	x	x	x	x	10
F319_ATAN	x	x	x	x	10
F320_LN	x	x	x	x	10
F321_EXP	x	x	x	x	10
F322_LOG	x	x	x	x	10
F323_PWR	x	x	x	x	14
F324_FSQR	x	x	x	x	10
F325_FLT	x	x	x	x	6

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
F326_DFLT	x	x	x	x	8
F327_INT	x	x	x	x	8
F328_DINT	x	x	x	x	8
F329_FIX	x	x	x	x	8
F330_DFIX	x	x	x	x	8
F331_ROFF	x	x	x	x	8
F332_DROFF	x	x	x	x	8
F333_FINT	x	x	x	x	8
F334_FRINT	x	x	x	x	8
F335_FSIGN	x	x	x	x	8
F336_FABS	x	x	x	x	8
F337_RAD	x	x	x	x	8
F338_DEG	x	x	x	x	8
F345_FCMP	x	x	x	x	10
F346_FWIN	x	x	x	x	14
F347_FLIMT	x	x	x	x	17
F348_FBAND	x	x	x	x	17
F349_FZONE	x	x	x	x	17
F350_FMAX	-	-	-	-	8
F351_FMIN	-	-	-	-	8
F352_FMEAN	-	-	-	-	8
F353_FSORT	-	-	-	-	8
F354_FSCAL	x	x	-	x	
F355_PID_DUT	x	x	x	x	4
F356_PID_PWM	x	x	-	x	10
F373_DTR	x	x	x	x	6
F374_DDTR	x	x	x	x	6
F_TRIG	x	x	x	x	
FIND	x	x	x	x	
GE	x	x	x	x	
GET_RTC_DTBCD	x	x	x	x	
GT	x	x	x	x	
ICTL	x	x	x	x	
INSERT	x	x	x	x	19
INT_TO_BCD	x	x	x	x	
INT_TO_BOOL	x	x	x	x	
INT_TO_BOOL16	x	x	x	x	
INT_TO_BOOLS	x	x	x	x	

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
INT_TO_DINT	x	x	x	x	
INT_TO_DWORD	x	x	x	x	
INT_TO_REAL	x	x	x	x	
INT_TO_SDT	x	x	x	x	
INT_TO_STRING	x	x	x	x	
INT_TO_STRING _LEADING_ZEROS	x	x	x	x	
INT_TO_TIME	x	x	x	x	
INT_TO_WORD	x	x	x	x	
IPADDR_TO_STRING	x	x	x	x	
IPADDR_TO_STRING_ NO _LEADING_ZEROS	x	x	x	x	
Is_AreaDT	x	x	x	x	
Is_AreaFL	x	x	x	x	
IsCommunicationError	x	x	x	x	
IsModbusError	x	x	x	x	
IsModbusNotActive	x	x	x	x	
IsPlcLink	x	x	x	x	
IsProgramControlled	x	x	x	x	
IsReceptionDone	x	x	x	x	
IsReceptionDoneByTimeOut	x	x	x	x	
IsTransmissionDone	x	x	x	x	
JP	x	x	x	x	2
KEEP	x	x	x	x	1
LBL	x	x	x	x	1
LE	x	x	x	x	
LEFT	x	x	x	x	8
LEN	x	x	x	x	
LIMIT	x	x	x	x	
LN	x	x	x	x	
LOG	x	x	x	x	
LOOP	x	x	x	x	4
LSR	x	x	x	x	1
LT	x	x	x	x	
MAX	x	x	x	x	
MC	x	x	x	x	2
MCE	x	x	x	x	2
MID	x	x	x	x	10

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
MIN	x	x	x	x	
MOD	x	x	x	x	
MOVE	x	x	x	x	
MUL	x	x	x	x	
MUL_TIME_DINT	x	x	x	x	
MUL_TIME_INT	x	x	x	x	
MUL_TIME_REAL	x	x	x	x	
MUX	x	x	x	x	
NE	x	x	x	x	
NOT	x	x	x	x	
OR	x	x	x	x	
P13_EPWT	x	x	x	x	11
PID_FB	x	x	x	x	
PID_FB_DUT	x	x	x	x	
R_TRIG	x	x	x	x	
REAL_TO_DINT	x	x	x	x	
REAL_TO_INT	x	x	x	x	
REAL_TO_STRING	x	x	x	x	
REAL_TO_TIME	x	x	x	x	
REPLACE	x	x	x	x	26
RIGHT	x	x	x	x	8
ROL	x	x	x	x	
ROR	x	x	x	x	
RS	x	x	x	x	
SDDT_TO_DINT	x	x	x	x	
SDDT_TO_DWORD	x	x	x	x	
SDT_TO_INT	x	x	x	x	
SDT_TO_WORD	x	x	x	x	
SEL	x	x	x	x	
SET_RTC_DTBCD	x	x	x	x	3
SfcOutputsReset	x	x	x	x	
SfcRunning	x	x	x	x	
SfcStopped	x	x	x	x	
SfcTransitionsInhibited	x	x	x	x	
SHL	x	x	x	x	
SHR	x	x	x	x	
SIN	x	x	x	x	
Size_Of_Var	x	x	x	x	

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
SQRT	x	x	x	x	
SR	x	x	x	x	
StartStopAllSfcs	x	x	x	x	
StartStopAllSfcsAndInitData	x	x	x	x	
StartStopSfc	x	x	x	x	
StartStopSfcAndInitData	x	x	x	x	
STRING_TO_DINT	x	x	x	x	
STRING_TO_DINT_ STEPSAVER	x	x	x	x	
STRING_TO_DWORD	x	x	x	x	
STRING_TO_DWORD_ STEPSAVER	x	x	x	x	
STRING_TO_ETLANADDR	x	x	x	x	
STRING_TO_ETLANADDR _STEPSAVER	x	x	x	x	
STRING_TO_INT	x	x	x	x	
STRING_TO_INT_ STEPSAVER	x	x	x	x	
STRING_TO_IPADDR	x	x	x	x	
STRING_TO_IPADDR_ STEPSAVER	x	x	x	x	
STRING_TO_REAL	x	x	x	x	
STRING_TO_WORD	x	x	x	x	
STRING_TO_WORD_ STEPSAVER	x	x	x	x	
SUB	x	x	x	x	
SUB_TIME	x	x	x	x	
SYS1	x	x	x	x	13
SYS2	x	x	x	x	7
TAN	x	x	x	x	
TIME_TO_DINT	x	x	x	x	
TIME_TO_DWORD	x	x	x	x	
TIME_TO_INT	x	x	x	x	
TIME_TO_REAL	x	x	x	x	
TIME_TO_STRING	x	x	x	x	
TIME_TO_WORD	x	x	x	x	
TM_1ms	x	x	x	x	3-4
TM_1ms_FB	x	x	x	x	3-4
TM_1s	x	x	x	x	4-5
TM_1s_FB	x	x	x	x	4-5

Instruction x available - not available	FP-X 12k	FP-X 32k	FP-Σ, 12k	FP-Σ, 32k	Steps
TM_10ms	x	x	x	x	3-4
TM_10ms_FB	x	x	x	x	3-4
TM_100ms	x	x	x	x	3-4
TM_100ms_FB	x	x	x	x	3-4
TOF	x	x	x	x	23
TON	x	x	x	x	7
TP	x	x	x	x	14
TRUNC_TO_DINT	x	x	x	x	
TRUNC_TO_INT	x	x	x	x	
Var_ToAreaOffs	x	x	x	x	
WORD_TO_BOOL	x	x	x	x	
WORD_TO_BOOL16	x	x	x	x	
WORD_TO_BOOLS	x	x	x	x	
WORD_TO_DINT	x	x	x	x	
WORD_TO_DWORD	x	x	x	x	
WORD_TO_INT	x	x	x	x	
WORD_TO_SDT	x	x	x	x	
WORD_TO_STRING	x	x	x	x	
WORD_TO_TIME	x	x	x	x	
XOR	x	x	x	x	

Index

A

B

C

D

E

F

G

R

Record of Changes

Manual No.	Date	Description of Changes
ACGM0132V1.0END	NOV. 2001	First edition
ACGM0132V1.1END	APR. 2002	Matsushita instructions for FP-Sigma added (F174, F175, F176) and SYS1, SYS2 updated.
ACGM0132V1.2END	JULY 2002	Chapter "High-Speed-Counter Special Instructions" updated and corrected
		Appendix "Programming Information" updated
ACGM0132V2.0	MAR. 2006	Update for release of FPWIN Pro V5.2
		Company name change
		New PLC types: FP-Sigma 32k, FP-X
		Grafical explanations completed

GLOBAL NETWORK

North America | Europe | Asia Pacific | China | Japan

Europe

- **Headquarters** — **Panasonic Electric Works Europe AG**
  Rudolf-Diesel-Ring 2, 83607 Holzkirchen, Germany, Tel. (08024) 648-0, Fax (08024) 648-111, www.panasonic-electric-works.com
- **Austria** — **Panasonic Electric Works Austria GmbH**
  Josef Madersperger Straße 2, A-2362 Biedermannsdorf, Austria, Tel. (02236) 26846, Fax (02236) 46133, www.panasonic-electric-works.at
- **Benelux** — **Panasonic Electric Works Sales Western Europe B. V.**
  De Rijn 4, (Postbus 211), 5684 PJ Best, (5680 AE Best), Netherlands, Tel. (0499) 37 27 27, Fax (0499) 37 21 85, www.panasonic-electric-works.nl
- **Czech Republic** — **Panasonic Electric Works Czech s.r.o**
  Prumyslová 1, 34815 Planá, Tel. (0374) 79 99 90, Fax (0374) 79 99 99, www.panasonic-electric-works.cz
- **France** — **Panasonic Electric Works Sales Western Europe B. V. French Branch Office**
  B.P. 44, F-91371 Verrières le Buisson CEDEX, France, Tél. 01 60 13 57 57, Fax 01 60 13 57 58, www.panasonic-electric-works.fr
- **Germany** — **Panasonic Electric Works Deutschland GmbH**
  Rudolf-Diesel-Ring 2, 83607 Holzkirchen, Germany, Tel. (08024) 648-0, Fax (08024) 648-555, www.panasonic-electric-works.de
- **Ireland** — **Panasonic Electric Works UK Ltd. Irish Branch Office**
  Dublin, Republic of Ireland, Tel. (01) 4600969, Fax (01) 4601131, www.panasonic-electric-works.ie
- **Italy** — **Panasonic Electric Works Italia s.r.l.**
  Via del Commercio 3-5 (Z.I. Ferlina), I-37012 Bussolengo (VR), Italy, Tel. (045) 675 27 11, Fax (045) 6 70 04 44, www.panasonic-electric-works.it
- **Nordic Countries** — **Panasonic Electric Works Nordic AB**
  Sjöängsvägen 10, 19272 Sollentuna, Sweden, Tel. (+46) 8 59 47 66 80, Fax (+46) 8 59 47 66 90, www.panasonic-electric-works.se
- **Portugal** — **Panasonic Electric Works Portugal España S.A. Portuguese Branch Office**
  Avda Adelino Amaro da Costa 728 R/C J, 2750-277 Cascais, Portugal, Tel. (351) 21 481 25 20, Fax (351) 21 481 25 29, www.panasonic-electric-works.es
- **Spain** — **Panasonic Electric Works España S.A.**
  Parque Empresarial Barajas, San Severo, 20, 28042 Madrid, Spain, Tel. (91) 329 38 75, Fax (91) 329 29 76, www.panasonic-electric-works.es
- **Switzerland** — **Panasonic Electric Works Schweiz AG**
  Grundstrasse 8, CH-6343 Rotkreuz, Switzerland, Tel. (041) 799 70 50, Fax (041) 799 70 55, www.panasonic-electric-works.ch
- **UK** — **Panasonic Electric Works UK Ltd.**
  Sunrise Parkway, Linford Wood East, Milton Keynes, MK14 6LF, England, Tel. (01908) 231 555, Fax (01908) 231 599, www.panasonic-electric-works.co.uk

North & South America

- **USA** — **PEW Corporation of America Head Office USA**
  629 Central Avenue, New Providence, N.J. 07974, USA, Tel. 1-908-464-3550, Fax 1-908-464-8513

Asia

- **China** — **Panasonic Electric Works (China) Co., Ltd.**
  2013, Beijing Fortune, Building 5, Dong San Huan Bei Lu, Chaoyang District, Beijing, China, Tel. 86-10-6590-8646, Fax 86-10-6590-8647
- **Hong Kong** — **Panasonic Electric Works (Hong Kong) Co., Ltd.**
  Rm1601, 16/F, Tower 2, The Gateway, 25 Canton Road, Tsimshatsui, Kowloon, Hong Kong, Tel. (852) 2956-3118, Fax (852) 2956-0398
- **Japan** — **Matsushita Electric Works, Ltd.**
  1048 Kadoma, Kadoma-shi, Osaka 571-8686, Japan, Tel. 06-6908-1050, Fax 06-6908-5781, www.mew.co.jp/e-acg/
- **Singapore** — **Panasonic Electric Works Asia Pacific Pte. Ltd.**
  101 Thomson Road, #25-03/05, United Square, Singapore 307591,Tel. (65) 6255-5473, Fax (65) 6253-5689