SIEMENS

SIMATIC

Standard Software for S7-300 and S7-400
Standard Functions Part 2

Reference Manual

03/2000
Edition 03

Safety guidelines

This manual contains notices which you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of hazard.

Danger

indicates an imminently hazardous situation that, if not avoided, will result in death or serious injury. Danger is limited to the most extreme situations.

Warning

indicates that death, severe personal injury, or substantial property damage can result.You must take proper precautions.
Warning is used for personal injury and also for property damage.

Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken. Caution is also used for property-damage-only accidents.

Note

draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Qualified personnel

This device should only be set up and operated in conjunction with this manual.

Only **qualified personnel** should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground and to tag equipment, systems and circuits in accordance with established safety practices and standards.

Correct usage

Note the following:

Warning

This device may only be used for the applications described in the catalog or technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored and set up carefully and correctly, and operated and maintained as recommended.

Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are registered trademarks of SIEMENS AG.

Third parties using for their own purposes any other names in this document which refer to trademarks might infringe upon the rights of the trademark owners.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

Disclaimer of Liability

We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

Preface

Purpose

This manual provides descriptions and examples of S7 functions FCs and function blocks FBs in ladder logic (LAD) representation. These FCs and FBs are available to program your S7-300/S7-400 programmable logic controller (PLC). This manual is intended as a reference to provide the necessary information for each function.

Where to Find the S7 Functions

The FCs and FBs described in this manual are located in the Standard Library of STEP 7. Using your STEP 7 file manager, you can copy the FCs and FBs you need to your destination program directory. First make certain that your program does not contain any FCs or FBs with the same number as the ones you want to copy from the library. If you have FCs or FBs with matching numbers, you must renumber either your program FCs or FBs, or the one(s) you want to copy to your program.

Audience

This manual is intended for engineers, programmers, and maintenance personnel who have a general knowledge of programmable logic controllers.

How to Use This Manual

This manual groups the FCs and FBs into the following functional areas:

- Bit logic functions (Chapter 1)
- Table functions (Chapter 2)
- Shift functions (Chapter 3)
- Move functions and function blocks (Chapter 4)
- Timer functions and function blocks (Chapter 5)
- Convert functions and function blocks (Chapter 6)
- Floating Point Math functions (Chapter 7)
- Compare function blocks (Chapter 8)
- The glossary provides an alphabetical listing of definitions of key terms and expressions that are applicable to ladder logic programming.

Each chapter describes FCs and FBs that you can add to your standard set of instructions to provide additional programming flexibility. Each FC or FB is listed with the full name, the abbreviation, and the FC or FB number. Each FC or FB is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table giving the declaration, data type, valid memory areas, and the description for each parameter.
- Error Information — errors that would prevent the FC or FB from being executed successfully.
- Example — a figure consisting of a graphic representation of the FC or FB with example parameters and the results of the execution.

Overview of the STEP 7 Documentation Set

This manual is a part of the STEP 7 documentation package that consists of the manuals listed in the following table.

Title	Content
Working with STEP 7, Getting Started	The *manual* offers a basic introduction to the methodology of the structure and programming of an S7-300/S7-400. It is especially suited to first-time users of an S7 programmable control system.
Programming with STEP 7 Manual	This manual provides basic information on the structure of the operating system and of a user program of an S7 CPU. The first-time user of an S7-300 or S7-400 should use this manual to acquire an overview of the programming methodology and to use it to base their user program design on.
System Software for S7-300 and S7-400 System and Standard Functions Reference Manual	The S7 CPUs have integrated system functions and organization blocks included with their operating system, which you can use when programming. The manual provides you with an overview of the system functions, organization blocks, and loadable standard functions available in S7, and – in the form of reference information – detailed interface descriptions for their use in your user program.
Configuring Hardware and Communication Connections STEP 7 Manual	The *STEP 7 Manual* explains the main usage and the functions of the STEP 7 automation software. As a first-time user of STEP 7 and as an experienced user of STEP 5, this manual will provide you with an overview of the procedures used to configure, program, and start up an S7-300/S7-400. While you are working with the software you can access a range of on-line help topics which offer detailed support on using the software.
From S5 to S7 Converter Manual	You will need the *Converting STEP 5 Programs Manual* if you want to convert existing STEP 5 programs to run them on S7 CPUs. The manual provides an overview of the procedures and usage of the Converter; you can find a detailed description of the converter functions in the on-line help. You will also find the interface descriptions for the converted S7 functions available in the on-line help.
Statement List, Ladder Logic, SCL[1] Reference Manuals	The manuals for the programming language packages Statement List, Ladder Logic, and SCL (Sequential Control Language) contain both the user's guide and the reference description of the programming language or representation type. You only require one language type for programming an S7-300/S7-400, but you can mix the languages within a project, if required. If you are using a language for the first time, it is recommended that you use the manual to learn about the methodology of creating a program in the chosen language first. While you are working with the software you can access a range of on-line help topics which offer detailed support on using the respective editors/compilers.

Title	Content
S7-GRAPH[1] , S7-HiGraph[1], CFC[1] Manuals	The languages S7-GRAPH, S7-HiGraph, and CFC (Continuous Function Chart) offer additional methods of programming blocks in the form of sequential controls, state graphs, or charts. The manuals contain both the user's guide and the reference description of the programming language. If you are using a language for the first time, it is recommended that you use the manual to learn about the methodology of creating a program in the chosen language first. While you are working with the software you can access a range of on-line help topics which offer detailed support on using the respective editors/compilers (with the exception of HiGraph).

1 Optional package for system software for S7-300/S7-400

Other Manuals

The various S7-300 and S7-400 CPUs, the S7-300 and S7-400 modules, and the instructions of the CPU are described in the following manuals:

- For the S7-300 programmable logic controller, refer to the manuals: Hardware and Installation (CPU Data, Module Data) and the Instruction List.
- For the S7-400 programmable logic controller, refer to the manuals: Hardware and Installation (CPU Data, Module Data) and the Instruction List.

You can find additional information in the on-line help.

Additional Assistance

If you have any questions not answered in this or one of the other STEP 7 manuals, if you need information on ordering additional documentation or equipment, or if you need information on training, please contact your Siemens distributor or sales office.

List of Functions and Function Blocks

The following functions and function blocks are described in this manual:

Function or Function Block	Number	Page
Software Timer On Delay—Retentive (TONR)	FC80	5-2
Indirect Block Move (IBLKMOV)	FC81	4-2
Reset Range of Outputs (RSET)	FC82	1-2
Set Range of Outputs (SET)	FC83	1-6
Add to Table (ATT)	FC84	2-2
First In/First Out Unload Table (FIFO)	FC85	2-4
Table Find (TBL_FIND)	FC86	2-6
Last In/First Out Unload Table (LIFO)	FC87	2-9
Table (TBL)	FC88	2-11
Move Table to Word (TBL_WRD)	FC89	2-13
Word Shift Register (WSR)	FC90	3-2
Word to Table (WRD_TBL)	FC91	2-15
Bit Shift Register (SHRB)	FC92	3-4
Seven Segment Decoder (SEG)	FC93	6-2
ASCII to Hex (ATH)	FC94	6-4
Hex to ASCII (HTA)	FC95	6-6
Encode Binary Position (ENCO)	FC96	6-8
Decode Binary Position (DECO)	FC97	6-9
Ten’s Complement (BCDCPL)	FC98	6-10
Sum Number of Bits (BITSUM)	FC99	6-11
Reset Range of Immediate Outputs (RSETI)	FC100	1-4
Set Range of Immediate Outputs (SETI)	FC101	1-8
Standard Deviation (DEV)	FC102	7-2
Correlated Data Table (CDT)	FC103	2-17
Table to Table (TBL_TBL)	FC104	2-19
Scaling Values (SCALE)	FC105	6-12
Unscaling Values (UNSCALE)	FC106	6-14
Lead/Lag Algorithm (LEAD_LAG)	FB80	6-16
Discrete Control Alarm Timer (DCAT)	FB81	5-4
Motor Control Alarm Timer (MCAT)	FB82	5-7
Index Matrix Compare (IMC)	FB83	8-2
Scan Matrix Compare (SMC)	FB84	8-6
Event Maskable Drum (DRUM)	FB85	5-10
Pack Data (PACK)	FB86	4-4

Contents

Bit Logic Functions

1

This chapter describes the bit logic functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility.

1.1 Reset Range of Outputs (RSET): FC82

Description

The Reset Range of Outputs (RSET) function resets the signal state of each bit in a specified range to 0 if the MCR bit is 1. If the MCR bit is 0, the signal state of each bit in the range remains unchanged. The number of bits in the range to be reset is specified by N, and the starting point of the range is pointed to by S_BIT.

Parameters

Table 1-1 describes the Reset Range of Outputs (RSET) parameters.

Table 1-1 Reset Range of Outputs (FC82) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
S_BIT	Input	Pointer*	I, Q, M, D	Points to the first bit in the range
N	Input	INT	I, Q, M, D, L, P or constant	Number of bits in the range to be reset

* Double word pointer format for area-crossing register indirect addressing

Error Information

If the S_BIT pointer references the I/O external input and output memory area (P memory), the signal state of each bit in the range remains unchanged and the signal state of ENO is set to 0.

Example

Figure 1-1 shows how the RSET instruction works. If the signal state of input I 0.0 is 1 (activated) and the MCR bit is 1, the RSET function is executed. In this example, S_BIT points to the first bit at location M0.0. The N parameter specifies 10 bits to be reset. After the instruction is executed, the signal state of each of the 10 bits in the range M0.0 through M1.1 is reset to 0.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

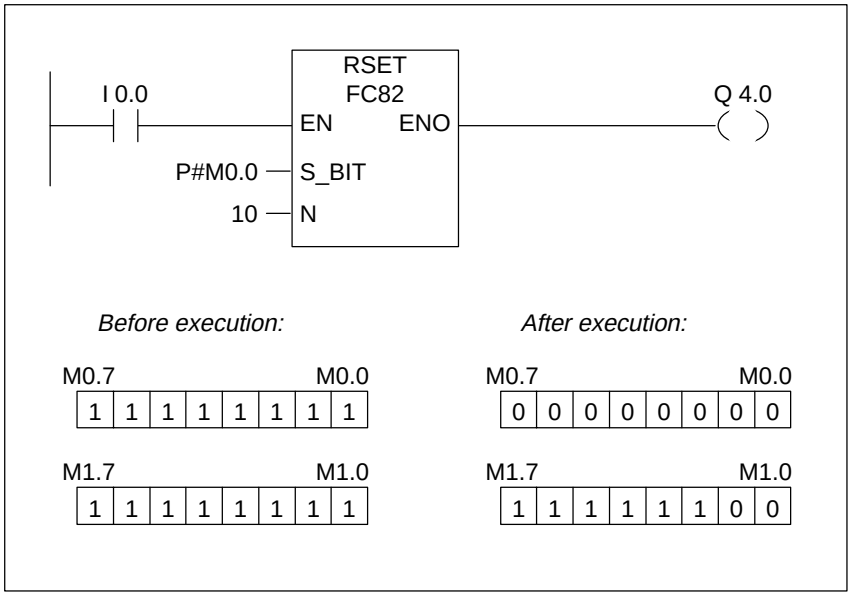


Figure 1-1 Reset Range of Outputs (RSET)

1.2 Reset Range of Immediate Outputs (RSETI): FC100

Description

The Reset Range of Immediate Outputs (RSETI) function resets the signal state of a range of bytes to 0 if the MCR bit is 1. If the MCR bit is 0, the signal state of each byte in the range remains unchanged. S_BYTE points to the first byte in the range, and N specifies the size of the range. The size of the range is expressed by specifying the number of bits in the range. For example, to specify a range of 2 bytes, you would enter 16 (16 bits) for the value of N.

Note

The value of N must be a multiple of eight (for example, 8, 16, 24, etc.).

The S_BYTE pointer must reference the external input and output memory area (P memory). Since P memory is accessed as bytes, words, or double words, the S_BYTE must reference an address that is byte-aligned, which means that the bit number of the pointer must be 0.

Note

The signal state of the corresponding bits in the process-image output table (Q memory) is also reset to 0.

Parameters

Table 1-2 describes the Reset Range of Immediate Outputs (RSETI) parameters.

Table 1-2 Reset Range of Immediate Outputs (FC100) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
S_BYTE	Input	Pointer*	P	Points to the first byte in the range
N	Input	INT	I, Q, M, D, L, P or constant	Size of the range of bytes to be reset to 0, specified by the number of bits in multiples of 8, (for example, 8, 16, etc.)

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the following conditions occur, the signal state of each bit in the range remains unchanged and the signal state of ENO is set to 0.

- The S_BYTE pointer references a memory area other than the I/O external input and output memory area (P memory).
- The S_BYTE pointer references an address that is not byte-aligned.
- The value of N is not a multiple of eight.

Example

Figure 1-2 shows how the RSETI instruction works. If the signal state of input I 0.0 is 1 (activated) and the MCR bit is 1, the RSETI function is executed. In this example, S_BYTE points to the first byte at location P2.0. The N parameter specifies 16 bits (2 bytes) to be reset. After the instruction is executed, the signal state of each bit in the range P2.0 through P3.7 is reset to 0.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

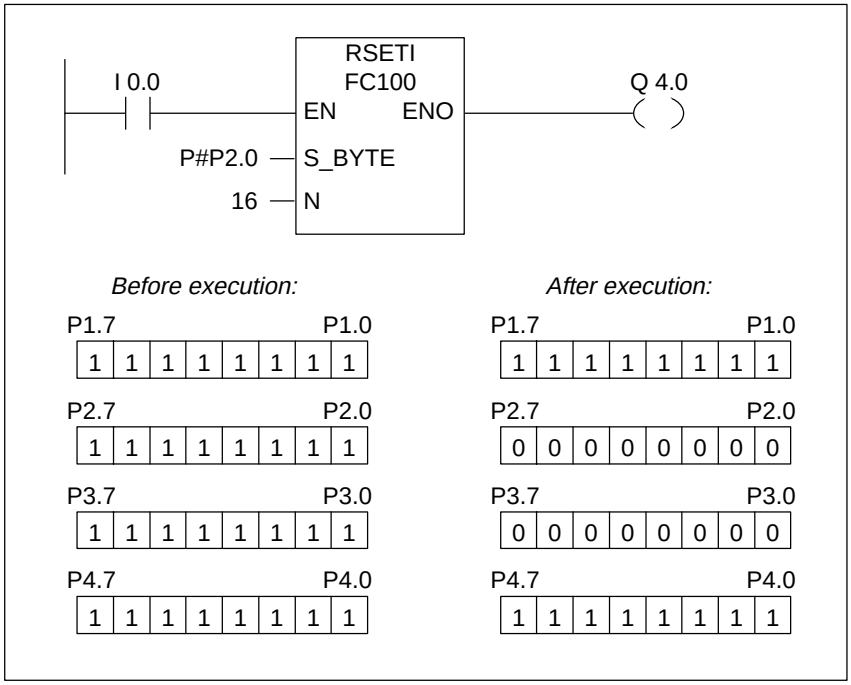


Figure 1-2 Reset Range of Immediate Outputs (RSETI)

1.3 Set Range of Outputs (SET): FC83

Description

The Set Range of Outputs (SET) function sets the signal state of each bit in a specified range to 1 if the MCR bit is 1. If the MCR bit is 0, the signal state of each of the bits in the range remains unchanged. The number of bits in the range to be set is specified by N, and the starting point of the range is pointed to by S_BIT.

Parameters

Table 1-3 describes the Set Range of Outputs (SET) parameters.

Table 1-3 Set Range of Outputs (FC83) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
S_BIT	Input	Pointer*	I, Q, M, D	Points to the first bit in the range
N	Input	INT	I, Q, M, D, L, P or constant	Number of bits in the range to be set

* Double word pointer format for area-crossing register indirect addressing

Error Information

If the S_BIT pointer references the I/O external input and output memory area (P memory), the signal state of each bit in the range remains unchanged and the signal state of ENO is set to 0.

Example

Figure 1-3 shows how the SET instruction works. If the signal state of input I 0.0 is 1 (activated) and the MCR bit is 1, the SET function is executed. In this example, S_BIT points to the first bit at location M0.0. The N parameter specifies 10 bits to be set. After the instruction is executed, the signal state of each of the 10 bits in the range M0.0 through M1.1 is set to 1.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

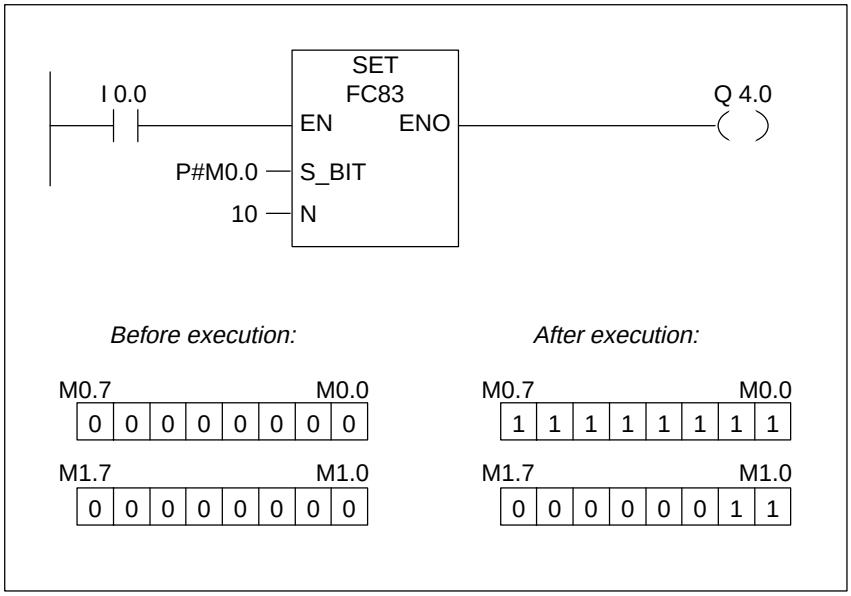


Figure 1-3 Set Range of Outputs (SET)

1.4 Set Range of Immediate Outputs (SETI): FC101

Description

The Set Range of Immediate Outputs (SETI) function sets the signal state of a range of bytes to 1 if the MCR bit is 1. If the MCR bit is 0, the signal state of each byte in the range remains unchanged. S_BYTE points to the first byte in the range, and N specifies the size of the range. The size of the range is expressed by specifying the number of bits in the range. For example, to specify a range of 2 bytes, you would enter 16 (16 bits) for the value of N.

Note

The value of N must be a multiple of eight (for example, 8, 16, 24, etc.).

The S_BYTE pointer must reference the external input and output memory area (P memory). Since P memory is accessed as bytes, words, or double words, the S_BYTE must reference an address that is byte-aligned, which means that the bit number of the pointer must be 0.

Note

The signal state of the corresponding bits in the process-image output table (Q memory) is also reset to 0.

Parameters

Table 1-4 describes the Set Range of Immediate Outputs (SETI) parameters.

Table 1-4 Set Range of Immediate Outputs (FC101) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
S_BYTE	Input	Pointer*	P	Points to the first byte in the range
N	Input	INT	I, Q, M, D, L, P or constant	Size of the range of bytes to be set to 1, specified by the number of bits in multiples of 8 (for example, 8, 16, etc.)

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the following conditions occur, the signal state of each bit in the range remains unchanged and the signal state of ENO is set to 0.

- The S_BYTE pointer references a memory area other than the I/O external inputs and outputs (P) memory area.
- The S_BYTE pointer references an address that is not byte-aligned.
- The value of N is not a multiple of eight.

Example

Figure 1-4 shows how the SETI instruction works. If the signal state of input I 0.0 is 1 (activated) and the MCR bit is 1, the SETI function is executed. In this example, S_BYTE points to the first byte at location P2.0. The N parameter specifies 16 bits (2 bytes) to be set. After the instruction is executed, the signal state of each bit in the range P2.0 through P3.7 is set to 1.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

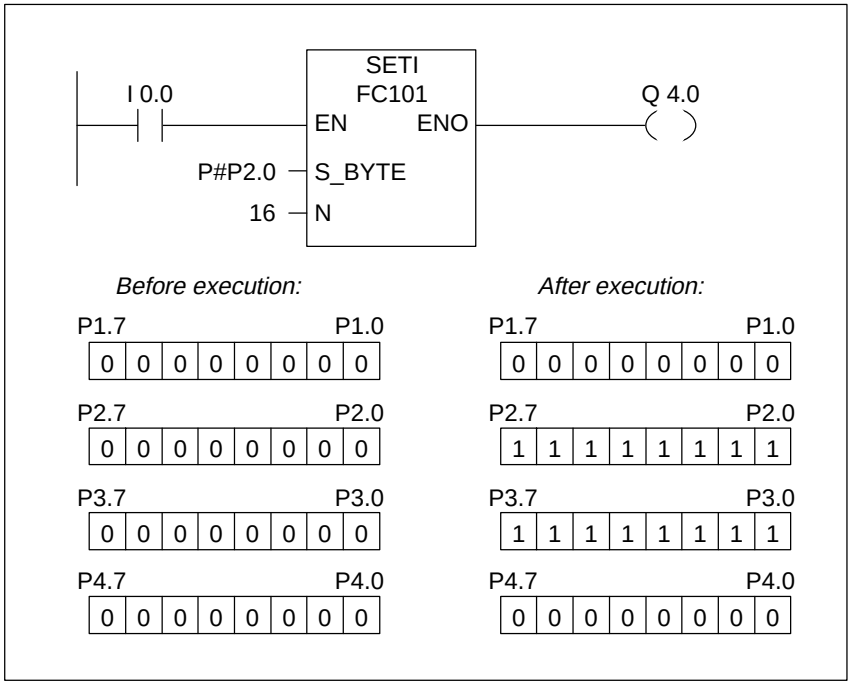


Figure 1-4 Set Range of Immediate Outputs (SETI)

Table Functions

2

This chapter describes the table functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility.

2.1 Add to Table (ATT): FC84

Description

The Add to Table (ATT) function adds DATA into the next entry of a table and increments the number of entries by one. The table consists of words. This function allows you to add entries into tables for use by the FIFO and LIFO functions.

- The first entry in the FIFO or LIFO table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the number of entries entered.
- The third entry in the table contains the first word of data.

Note

You must initialize the first two entries when you create the table.

Parameters

Table 2-1 describes the Add to Table (ATT) parameters.

Table 2-1 Add to Table (FC84) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
DATA	Input	WORD	I, Q, M, D, L, P or constant	Data to add to table
TABLE	Input	Pointer*	I, Q, M, D	Points to the start of the FIFO or LIFO table

* Double word pointer format for area-crossing register indirect addressing

Error Information

If the number of entries is equal to or greater than the table length, the data will not be added to the table and the signal state of ENO is set to 0.

Example

Figure 2-1 shows how the ATT instruction works. If the signal state of input I 0.0 is 1 (activated), the ATT function is executed. In this example, DATA is added as the fifth entry in the table and the number of entries increments from 4 to 5.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

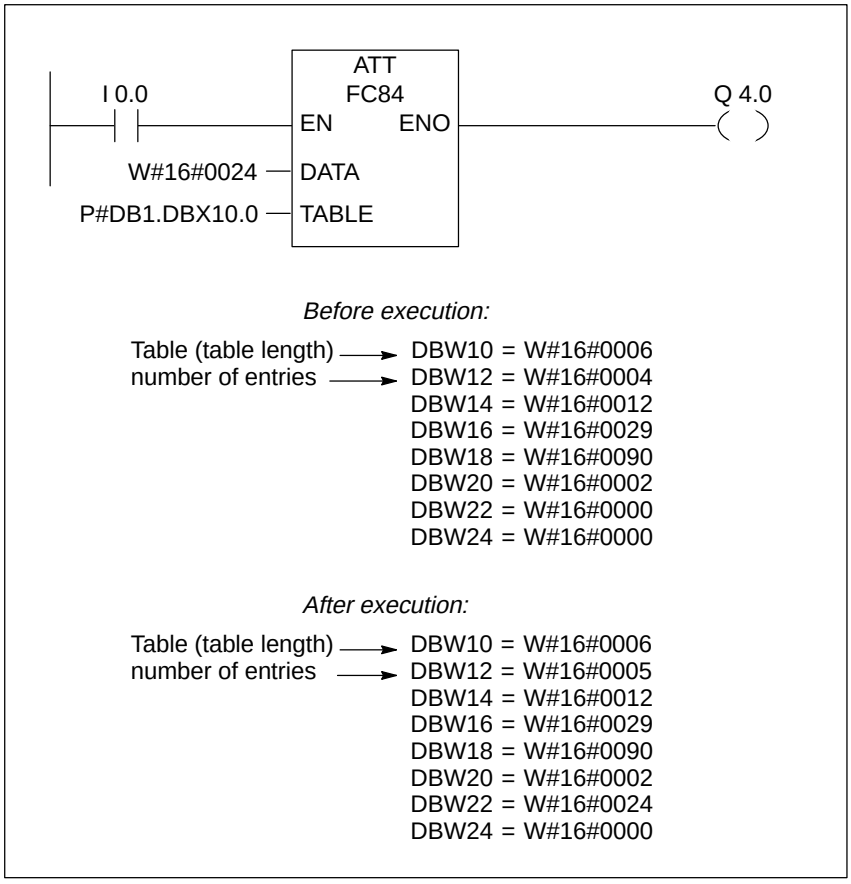


Figure 2-1 Add to Table (ATT)

2.2 First In/First Out Unload Table (FIFO): FC85

Description

The First In/First Out Unload Table (FIFO) function returns the oldest entry from the FIFO table as the function value. The number of entries decrements by one, and if more entries remain, they are shifted down in the table. The FIFO table consists of words. You use the ATT function to add values into the FIFO table.

- The first entry in the table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the number of entries entered.
- The third entry in the table contains the first word of data.

Parameters

Table 2-2 describes the First In/First Out Unload Table (FIFO) parameters.

Table 2-2 First In/First Out Unload Table (FC85) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
TABLE	Input	Pointer*	I, Q, M, D	Points to the start of the FIFO table
RET_VAL	Output	WORD	I, Q, M, D, L, P	The oldest entry from the FIFO table

* Double word pointer format for area-crossing register indirect addressing

Error Information

If the FIFO table is empty (number of entries = 0), the RET_VAL is not changed and the signal state of ENO is set to 0.

Example

Figure 2-2 shows how the FIFO instruction works. If the signal state of input I 0.0 is 1 (activated), the FIFO function is executed. In this example, the oldest entry in the table is returned as the function value (MW2). The number of entries is decremented from 5 to 4, and the remaining entries are shifted down in the table.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

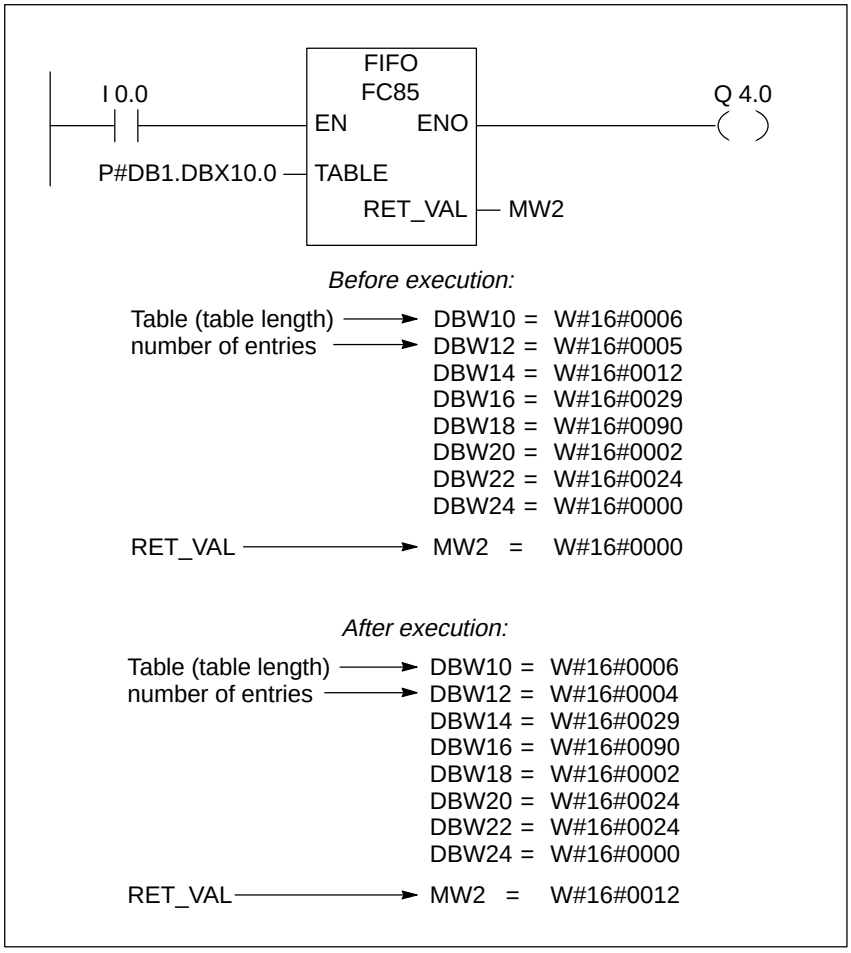


Figure 2-2 First In/First Out Unload Table (FIFO)

2.3 Table Find (TBL_FIND): FC86

Description

The Table Find (TBL_FIND) function is used to search for a distinct pattern or to search for a non-consistent pattern in a block of memory. This function performs the indicated compare command (CMD) between the source pattern (PATRN) and the source table (SRC) entries. It locates the next entry (after the entry indexed by INDX) in the table that satisfies the compare command and places its entry number in INDX. If no match is found, INDX points past the end of the table and the function's output is turned off.

- If CMD = 1, the function searches for the first value equal to the PATRN value.
- If CMD = 2, the function searches for the first value that is not equal to the PATRN value.
- The first entry in the table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the first table value.

Note

You must initialize the first entry (table length) of the table.

Parameters

Table 2-3 describes the Table Find (TBL_FIND) parameters.

Table 2-3 Table Find (FC86) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
SRC	Input	Pointer*	I, Q, M, D	Points to the start of the table
PATRN	Input	Pointer*	I, Q, M, D	Points to the pattern to be searched for
CMD	Input	BYTE	I, Q, M, D, L, P	Indicates command type: B#16#01 = equal B#16#02 = not equal
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the TBL_FIND function are as follows: B#16#02 = BYTE B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
INDX	Input_Output	WORD	I, Q, M, D, L	Index into table that provides: Input: starting entry number of the search Output: entry number of the matching value

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 2-4 occur, the table values are not changed. The signal state of ENO is set to 0 and the return value is set appropriately (see Table 2-4).

Table 2-4 Error Conditions for FC86

RET_VAL	Explanation
W#16#0008	No match was found
W#16#0009	Invalid E_TYPE and/or CMD

Example

Figure 2-3 shows how the TBL_FIND instruction works. If the signal state of input I 0.0 is 1 (activated), the TBL_FIND function is executed. In this example, since E_TYPE is 4, data in the table is stored in words starting at the entry pointed to by SRC. These words are compared against the pattern value 5555, which is stored in the location pointed to by PATRN. Because the CMD value is 1, the search locates the first table value in SRC that equals the pattern value. The INDX value points to the entry where the search is to begin. After the instruction is executed, the INDX value provides the entry number in the table where the compare command was satisfied.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

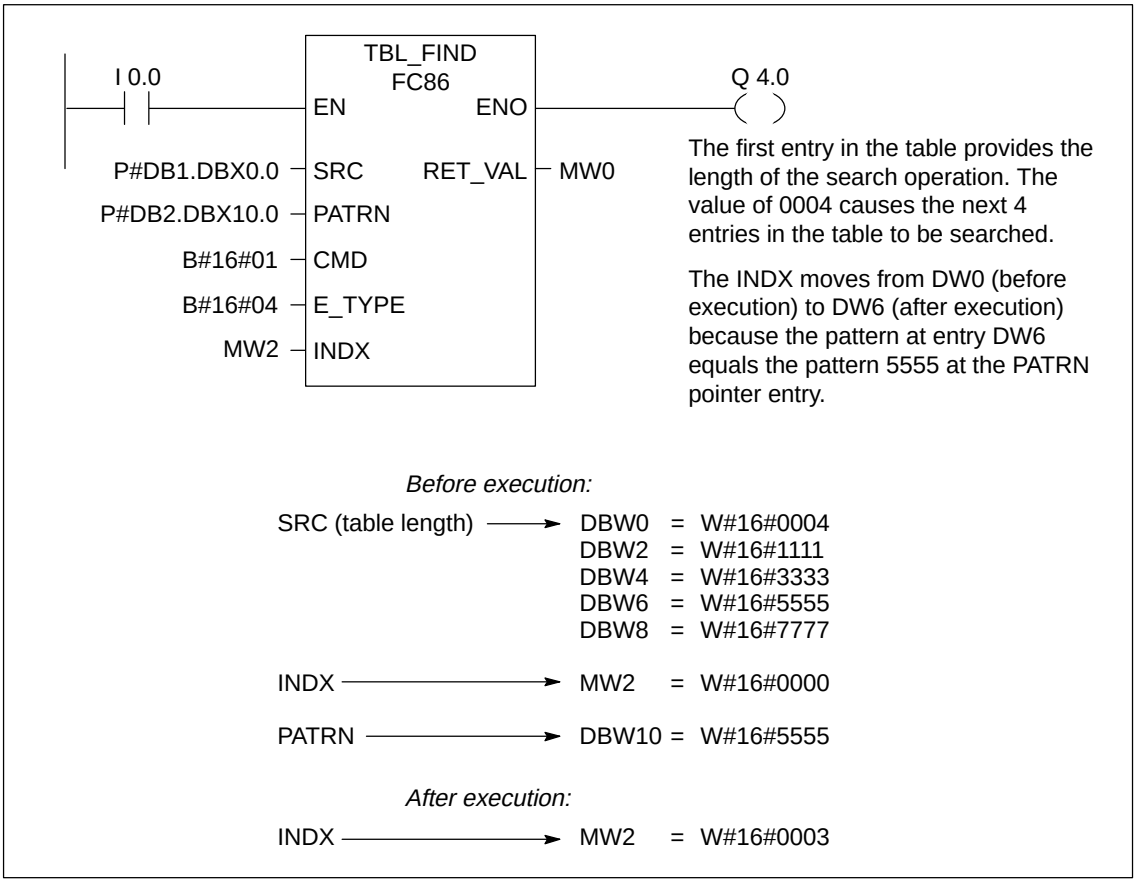


Figure 2-3 Table Find (TBL_FIND)

2.4 Last In/First Out Unload Table (LIFO): FC87

Description

The Last In/First Out Unload Table (LIFO) function returns the newest (most recent) entry from the LIFO table as the function value and decrements the number of entries by one. The LIFO table consists of words. You use the ATT function to add entries into the LIFO table.

- The first entry in the table contains the maximum number of entries in the table (table length).
- The second entry in the table contains the number of entries entered.
- The third entry in the table contains the first word of data.

Parameters

Table 2-5 describes the Last In/First Out Unload Table (LIFO) parameters.

Table 2-5 Last In/First Out Unload Table (FC87) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
TABLE	Input	Pointer*	I, Q, M, D	Points to the start of the LIFO table
RET_VAL	Output	WORD	I, Q, M, D, L, P	The newest entry from the LIFO table

* Double word pointer format for area-crossing register indirect addressing

Error Information

If the LIFO table is empty (number of entries = 0), the RET_VAL is not changed and the signal state of ENO is set to 0.

Example

Figure 2-4 shows how the LIFO instruction works. If the signal state of input I 0.0 is 1 (activated), the LIFO function is executed. In this example, the newest entry in the LIFO table is returned as the function value (MW2). The number of entries is decremented from 5 to 4.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

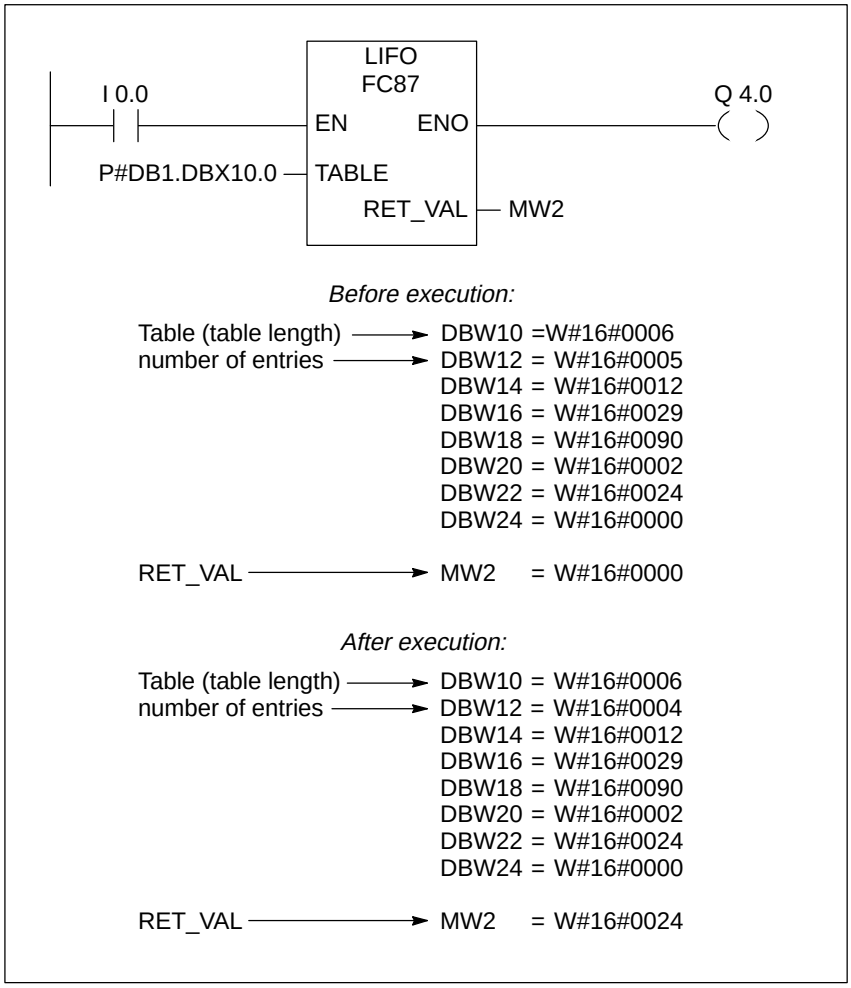


Figure 2-4 Last In/First Out Unload Table (LIFO)

2.5 Table (TBL): FC88

Description

The Table (TBL) function performs the indicated command (set by CMD) to the source table and writes the result in the same table entry.

- The first entry in the table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the first table value.
- If the E_TYPE is set to REAL, then the CMD value for ones complement is invalid.

Note

You must initialize the first entry when you create the table.

Parameters

Table 2-6 describes the Table (TBL) parameters.

Table 2-6 Table (FC88) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
SRC	Input	Pointer*	I, Q, M, D	Points to the start of the table
CMD	Input	BYTE	I, Q, M, D, L, P	Indicates type of command to be performed. Valid commands and their values are: B#16#03 = ones complement B#16#04 = clear B#16#05 = negate B#16#06 = square root
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the TBL function are as follows: B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

* Double word pointer format for area-crossing register indirect addressing

Error Information

If CMD or E_TYPE are invalid or CMD and E_TYPE are inconsistent with each other, the table values are not changed. The signal state of ENO is set to 0 and RET_VAL is set equal to W#16#0008.

Example

Figure 2-5 shows how the TBL instruction works. If the signal state of input I0.0 is 1 (activated), the TBL function is executed. In this example, SRC points to the data block locations that will be manipulated by the instruction. Since E_TYPE is 4, data in the table is stored in words starting at the entry pointed to by SRC. Because the CMD value is 4 (Clear), all words in the table are cleared (set to zero) when the TBL instruction is executed. The table length value of 5 in the first table entry causes the next five table entries to be cleared.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

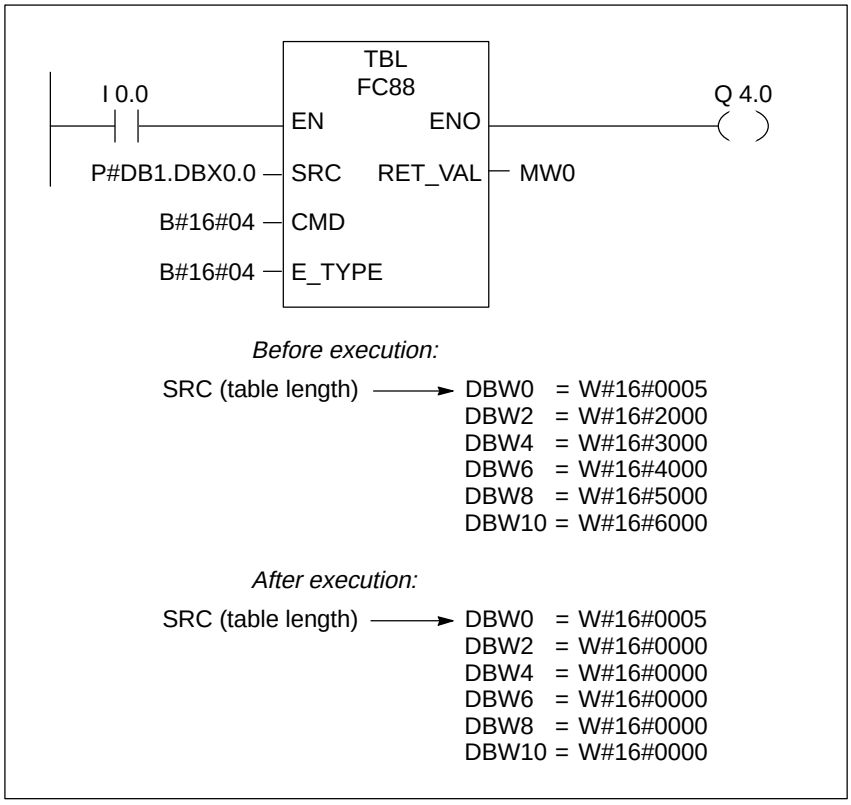


Figure 2-5 Table (TBL)

2.6 Move Table to Word (TBL_WRD): FC89

Description

The Move Table to Word (TBL_WRD) function copies the entry indicated by INDX from the SRC table to the entry pointed to by DEST, and then increments INDX as long as INDX is less than the table length that is given in the first word of the table, SRC[0]. If the INDX is set to the last table entry when this instruction is called, the Q output bit is set to 0 after execution.

- The first entry in the table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the first table value.

Note

You must initialize the first entry when you create the table.

Parameters

Table 2-7 describes the Move Table to Word (TBL_WRD) parameters.

Table 2-7 Move Table to Word (FC89) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
SRC	Input	Pointer*	I, Q, M, D	Points to the start of the table
DEST	Input	Pointer*	I, Q, M, D	Points to the destination
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid types for the TBL_WRD function are as follows: B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
Q	Output	BOOL	Q, M, D, L	Indicates 0 if the INDX variable contains the last entry of the table when the function is called
INDX	Input_Output	WORD	I, Q, M, L	Entry number of the entry to move

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 2-8 occur, the function is not executed. The signal state of ENO is set to 0 and the return value is set appropriately.

Table 2-8 Error Conditions for FC89

RET_VAL	Explanation
W#16#0007	Index is 0
W#16#0008	Invalid E_TYPE
W#16#0009	Index is beyond the end of the table

Example

Figure 2-6 shows how the TBL_WRD instruction works. If the signal state of input I 0.0 is 1 (activated) the TBL_WRD function is executed. Since the E_TYPE is 4, the word data that is stored in the table starting at the entry pointed to by SRC is copied to the entry pointed to by DEST. The INDX value points to the table entry to be moved. After the instruction is executed successfully, the INDX is automatically incremented to one entry past the entry that was moved. In this example, when the instruction is called, the INDX is not set to the last table entry, so Q is set to 1 after execution.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

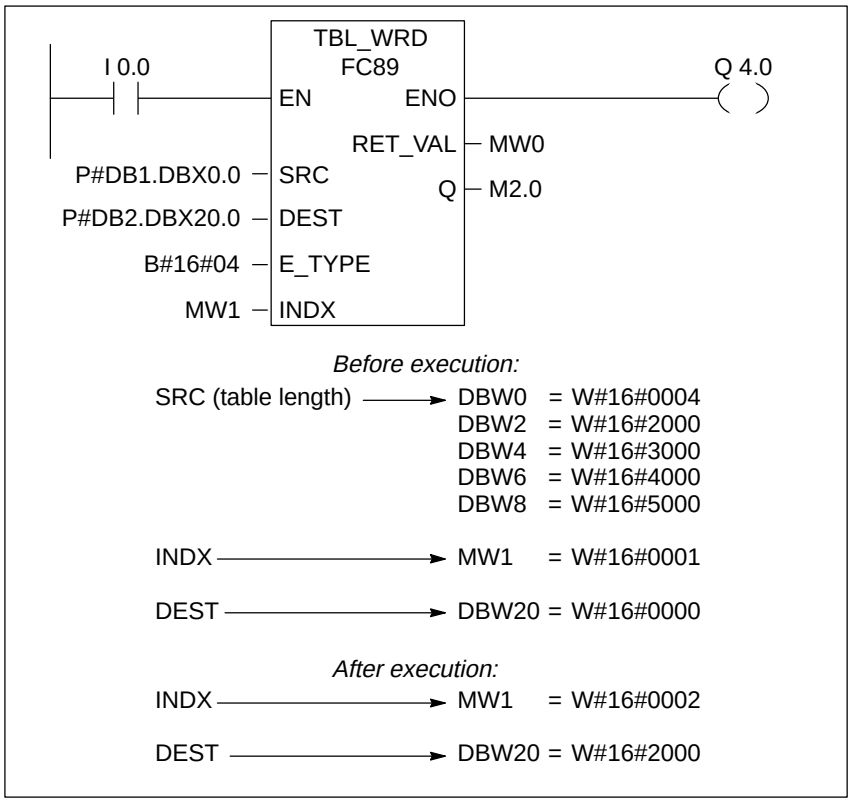


Figure 2-6 Move Table to Word (TBL_WRD)

2.7 Word to Table (WRD_TBL): FC91

Description

The Word to Table (WRD_TBL) function performs the indicated command (CMD) between the source data (pointed to by SRC) and the entry of the table at the offset indicated by INDX, then increments INDX as long as INDX is less than the length of the table.

- The first entry in the table contains the maximum number of entries of the table (table length).
- The second entry in the table contains the first table value.
- If E_TYPE is REAL, then CMD can only be “Move.”

Note

You must initialize the first entry when you create the table.

Parameters

Table 2-9 describes the Word to Table (WRD_TBL) parameters.

Table 2-9 Word to Table (FC91) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
SRC	Input	Pointer*	I, Q, M, D	Points to the source data
TABLE	Input	Pointer*	I, Q, M, D	Points to the start of the table
CMD	Input	BYTE	I, Q, M, D, L, P	Indicates the type of command to be performed. Valid commands and their values are: B#16#0E = move B#16#07 = and B#16#08 = or B#16#09 = exclusive or
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the WRD_TBL function are as follows: B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
Q	Output	BOOL	Q, M, D, L	Indicates 0 if the INDX contains the last entry number of the table
INDX	Input_Output	WORD	I, Q, M, D, L	Entry number of the entry to be operated on

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 2-10 occur, the function is not executed. The signal state of ENO is set to 0 and the return value is set appropriately.

Table 2-10 Error Conditions for FC91

RET_VAL	Explanation
W#16#0007	Index is 0
W#16#0008	Invalid CMD or E_TYPE, or CMD and E_TYPE are inconsistent with each other
W#16#0009	Index is beyond the end of the table

Example

Figure 2-7 shows how the WRD_TBL instruction works. If the signal state of input I 0.0 is 1 (activated), the WRD_TBL function is executed. Since the E_TYPE is 6, double word data is stored in the table starting at the memory location pointed to by TABLE. The table length in the first word indicates that the table contains three double words. The INDX value points to the table entry to be manipulated. Since the CMD value is 8, the instruction performs an OR command on the value that INDX points to. Since the INDX is 2, the second double word (66665544) is ORed with the value pointed to by SRC (11111111). After the instruction is executed, the result of the OR command (77775555) is written back into the table, and the INDX is automatically incremented one entry. If the INDX is set to the last table entry when this instruction is called, the Q output bit is set to 0 after execution. In this example, when the instruction is called, the INDX is not set to the last table entry, so Q is set to 1 after execution.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

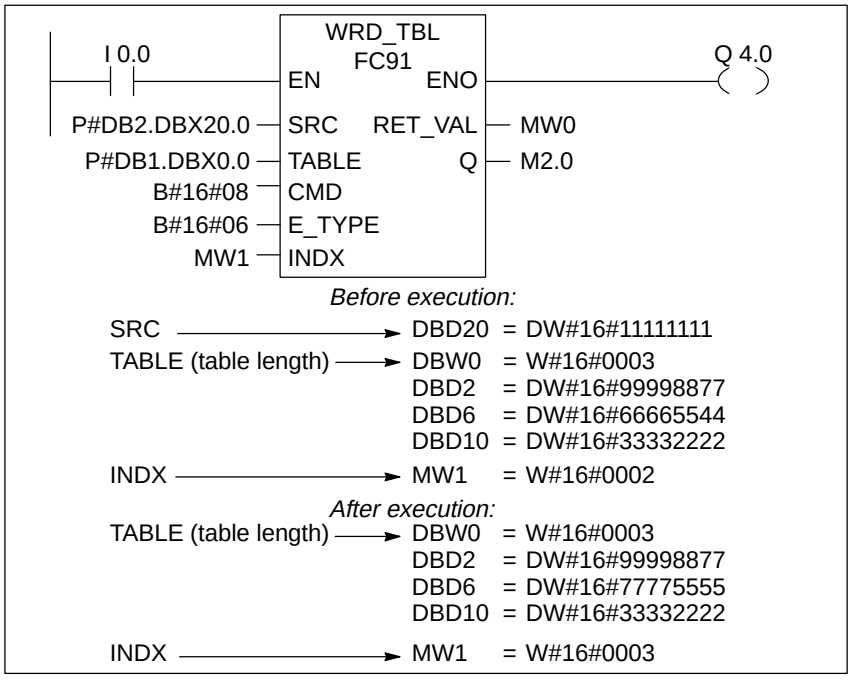


Figure 2-7 Word To Table (WRD_TBL)

2.8 Correlated Data Table (CDT): FC103

Description

The Correlated Data Table (CDT) function compares an input value (IN) to a pre-existing input table of values (IN_TBL) and locates the first value that is greater than or equal to the input value. If located, the index of the located value is used to copy the corresponding value in the output table (OUT_TBL) to the output value (OUT).

- The input table values should be in ascending order. That is, the smallest value is located in the first table entry and the largest value is located in the last table entry.
- The size of the input value, the table values, and the output value is determined from E_TYPE.
- The first entry in the table contains the number of entries of the table (table length).
- The second entry in the table contains the first table value.
- The number of entries in both tables must be equal and be greater than zero.

Note

You must initialize the first entry when you create each table.

Parameters

Table 2-11 describes the Correlated Data Table (CDT) parameters.

Table 2-11 Correlated Data Table (FC103) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN_TBL	Input	Pointer	I, Q, M, D	Points to the start of the input table
OUT_TBL	Input	Pointer	I, Q, M, D	Points to the start of the output table
IN	Input	Pointer	I, Q, M, D	Points to the input value
OUT	Input	Pointer	I, Q, M, D	Points to the output value
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the CDT function are as follows: B#16#05 = INT B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 2-12 occur, the function is not executed. The signal state of ENO is set to 0 and the return value is set appropriately.

Table 2-12 Error Conditions for FC103

RET_VAL	Explanation
W#16#0001	Invalid memory type specified for a function parameter
W#16#0002	Invalid E_TYPE
W#16#0003	The input and output table lengths do not match
W#16#0004	Table length is zero
W#16#0007	No value in IN_TBL was greater than or equal to the input value

Example

Figure 2-8 shows how the CDT instruction works. If the signal state of input I0.0 is 1 (activated), the CDT function is executed. In this example, both IN_TBL and OUT_TBL contain 5 table entries as indicated by the first word of each table. The data type of the table values is INTEGER as indicated by E_TYPE, and the value of IN is 22. The IN_TBL value that is greater than or equal to 22 is 64, which has an index of 5. The correlated value in OUT_TBL is 25. Therefore, the value 25 is written to OUT.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

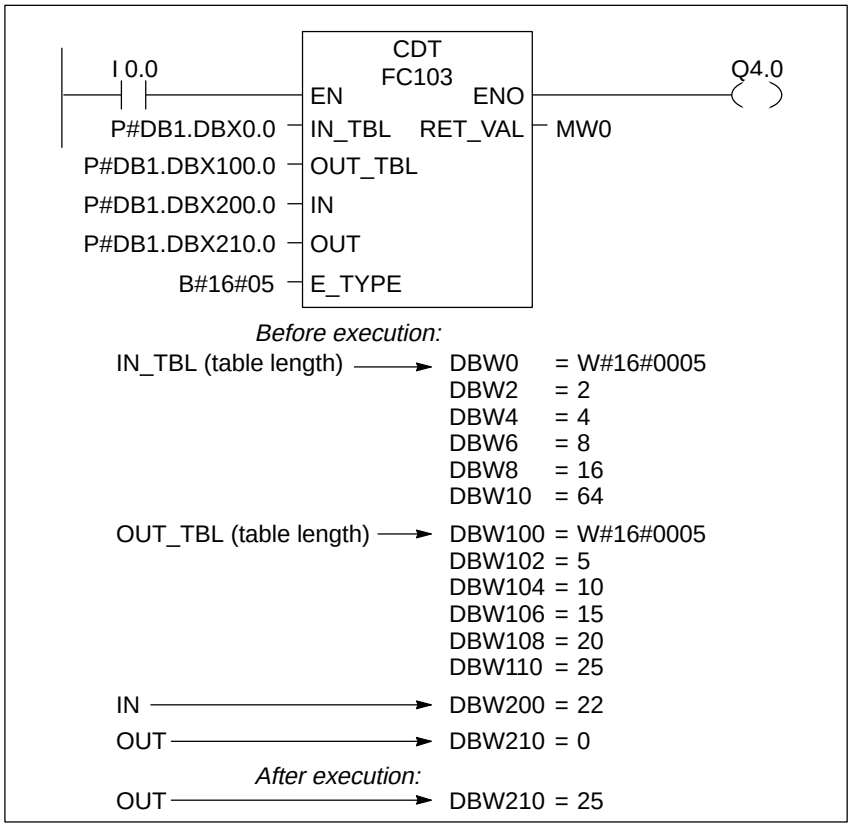


Figure 2-8 Correlated Data Table (CDT)

2.9 Table To Table (TBL_TBL): FC104

Description

The Table to Table (TBL_TBL) function performs the indicated command (CMD) between the corresponding entries of the two source tables (TBL1 and TBL2) and writes the result in the corresponding entries of the destination table (DEST_TBL).

- Data types of INT, DINT, or REAL are only valid for the math operations
- The first entry in the table contains the number of entries of the table (table length)
- The number of entries in all of the tables must be equal and be greater than zero

---

Note

You must initialize the first entry when you create each table.

Parameters

Table 2-13 describes the Table to Table (TBL_TBL) parameters.

Table 2-13 Table To Table (FC104) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
TBL1	Input	Pointer	I, Q, M, D	Points to start of the first source table
TBL2	Input	Pointer	I, Q, M, D	Points to start of the second source table
DEST_TBL	Input	Pointer	I, Q, M, D	Points to start of the destination table
CMD	Input	BYTE	I, Q, M, D, L, P	Indicates the type of command to be performed. Valid commands and their values are: B#16#07 = and B#16#08 = or B#16#09 = exclusive or B#16#0a = add B#16#0b = subtract B#16#0c = multiply B#16#0d = divide
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the TBL_TBL function are as follows: B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#160000

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 2-14 occur, the function is not executed. The signal state of ENO is set to 0 and the return value is set appropriately.

Table 2-14 Error Conditions for FC104

RET_VAL	Explanation
W#16#0001	Invalid memory type specified for a function parameter
W#16#0002	Invalid E_TYPE
W#16#0003	The input and output table lengths do not match
W#16#0004	Table length is zero
W#16#0005	E_TYPE and CMD are inconsistent with each other
W#16#0006	Invalid CMD

Example

Figure 2-9 shows how the TBL_TBL instruction works. If the signal state of input I0.0 is 1 (activated), the TBL_TBL function is executed. In this example, all tables contain 3 table entries as indicated by the first word of each table. The data type of the table values is WORD as indicated by E_TYPE. The command to be performed on TBL1 and TBL2 is AND as indicated by CMD.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

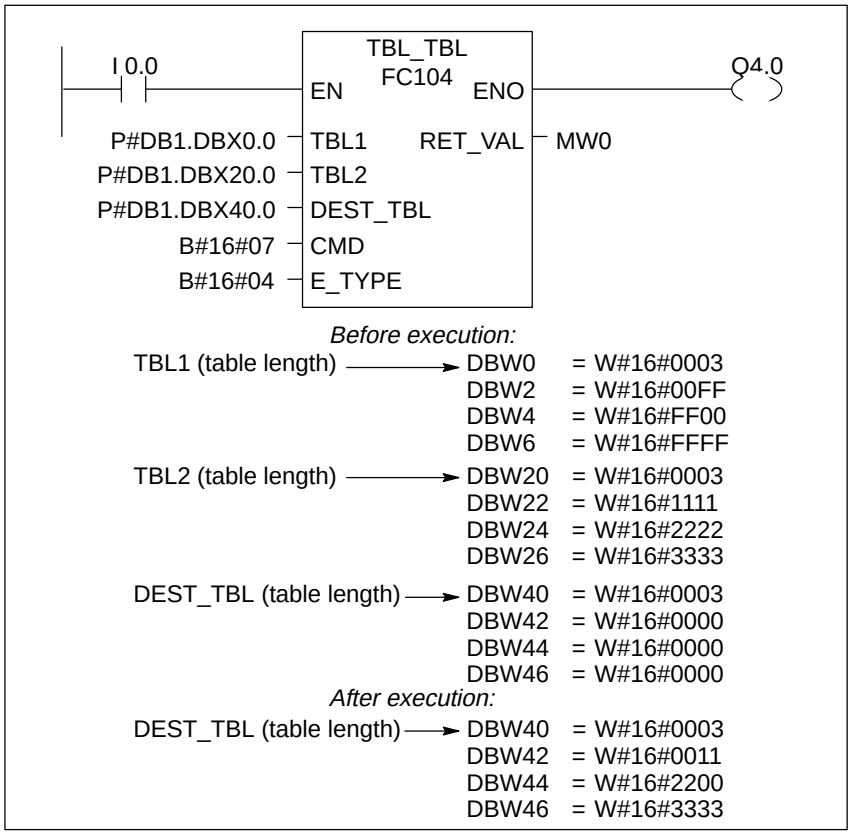


Figure 2-9 Table To Table (TBL_TBL)

Shift Functions

3

This section describes shift functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility.

3.1 Word Shift Register (WSR): FC90

Description

The Word Shift Register (WSR) function shifts data into a shift register from the indicated source. Each value is moved into the next location. LENGTH specifies the number of locations to be shifted. The data contained in the last location of the shift register is lost after the shift. New data is read from the source (S_DATA) each time the instruction is executed; this data is shifted into the shift register starting location (START) when the RESET input is set to 0. If the RESET input is set to 1, the register location is set to zeros when the instruction is executed. The output Q is turned on when the shift register is empty or cleared (that is, after a reset or after all zeros are shifted in).

Parameters

Table 3-1 describes the Word Shift Register (WSR) parameters.

Table 3-1 Word Shift Register (FC90) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
RESET	Input	BOOL	I, Q, M, D, L	Resets the shift register when set to 1
S_DATA	Input	Pointer*	I, Q, M, D	Points to the source data to be inserted into the table
START	Input	Pointer*	I, Q, M, D	Points to the start of the table
LENGTH	Input	WORD	I, Q, M, D, L, P	Number of items to be shifted
E_TYPE	Input	BYTE	I, Q, M, D, L, P	Indicates the data type of the table entries. Valid data types for the WSR function are as follows: B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL
Q	Output	BOOL	Q, M, D, L	Indicates 0 if the RESET is active (1) or all items to be shifted are a value of 0

* Double word pointer format for area-crossing register indirect addressing

Error Information

If an invalid E_TYPE is used, the function is not executed and the signal state of ENO is set to 0.

Example

Figure 3-1 shows how the WSR instruction works. If the signal state of input I 0.0 is 1 (activated) the WSR function is executed. Since the E_TYPE is 4, word data is stored in the table starting at the memory location pointed to by START. The LENGTH parameter is set to 4, indicating that 4 word locations will be shifted, starting with the first word at the START pointer. After the first value in a table is shifted to the next location, the first location is filled with the data pointed to by the S_DATA pointer. The last table value is lost. Whenever the RESET input is set to 1, the locations in the table are set to zero rather than being shifted.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

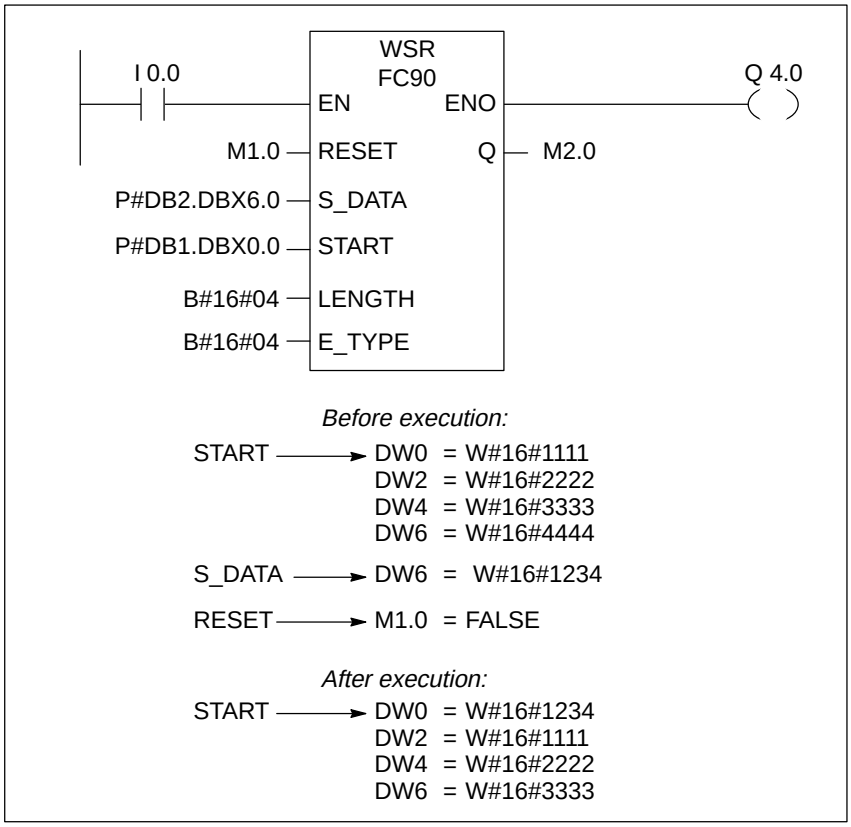


Figure 3-1 Word Shift Register (WSR)

3.2 Bit Shift Register (SHRB): FC92

Description

The Bit Shift Register (SHRB) function shifts a bit into a shift register from the indicated source in DATA. New data is read from the source each time the instruction is executed, and this data is shifted into the shift register starting location (S_BIT) while the RESET input has a signal state of 0. All successive bits are shifted by one. The bit contained in the last location (S_BIT+N) is lost after the shift. Whenever the RESET input is set to 1, the locations in the table are set to 0 rather than being shifted.

Parameters

Table 3-2 describes the Bit Shift Register (SHRB) parameters.

Table 3-2 Bit Shift Register (FC92) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
DATA	Input	BOOL	I, Q, M, D, L	Source data bit
RESET	Input	BOOL	I, Q, M, D, L	Resets the shift register when set to 1
S_BIT	Input	Pointer*	I, Q, M, D	Points to the starting bit in the shift register
N	Input	WORD	I, Q, M, D, L, P	Length of the shift register (number of bits to be shifted)

* Double word pointer format for area-crossing register indirect addressing

Error Information

This function does not detect any error conditions.

Example

Figure 3-2 shows how the SHRB instruction works. If the signal state of input I 0.0 is 1 (activated) the SHRB function is executed. The N parameter in this example is set to 14 (E in hexadecimal notation), indicating that 14 bit locations will be shifted, starting with the first bit at the S_BIT pointer location. After the bits are shifted, the first location is filled with the data indicated by the input DATA. The very last bit value is lost.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

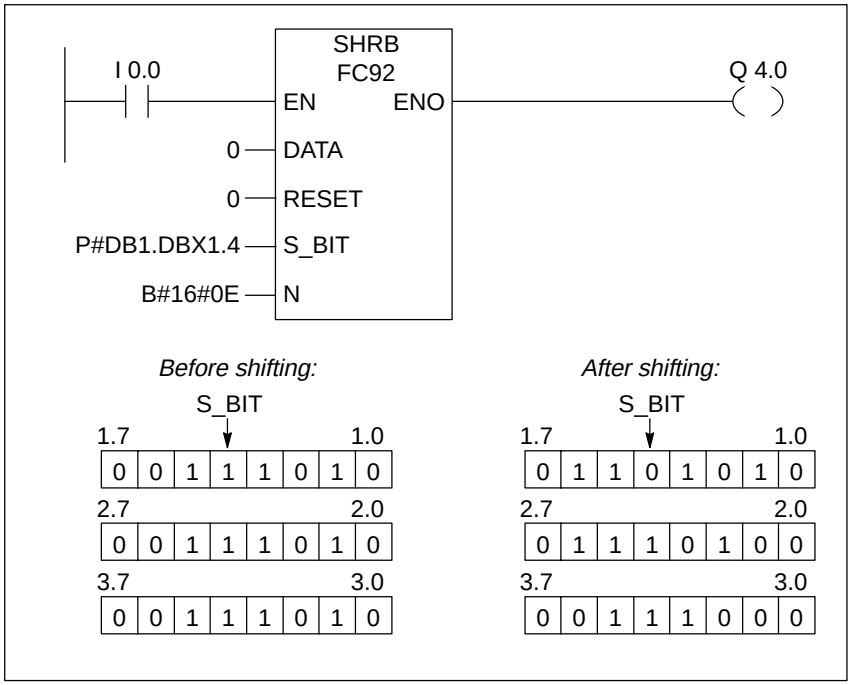


Figure 3-2 Bit Shift Register (SHRB)

Move Function and Function Block

4

This section describes move function (FC) and function block (FB) that you can add to your standard set of instructions to provide additional programming flexibility.

4.1 Indirect Block Move (IBLKMOV): FC81

Description

You can use the Indirect Block Move (IBLKMOV) instruction to move a block of data consisting of either bytes, words, integers, double words, or double integers from a source block to a destination block. The number of elements to be moved is determined by LENGTH and the element size is identified by E_TYPE. The S_DATA and D_DATA pointers identify the location of pointers that identify the starting locations of the source and destination data. Because of this indirect method of identifying the data to be moved, this function is called an indirect block move.

Parameters

Table 4-1 describes the Indirect Block Move (IBLKMOV) parameters.

Table 4-1 Indirect Block Move (FC81) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
S_DATA	Input	Pointer*	I, Q, M, D	Points to a pointer which identifies the starting location of the source data
LENGTH	Input	Pointer*	I, Q, M, D	Points to the length of the block to be moved
D_DATA	Input	Pointer*	I, Q, M, D	Points to a pointer which identifies the starting location of the destination data
E_TYPE	Input	BYTE	I, Q, M, D, L	Indicates the data type. Valid data types for the IBLKMOV function are as follows: B#16#02 = BYTE B#16#04 = WORD B#16#05 = INT B#16#06 = DWORD B#16#07 = DINT B#16#08 = REAL

* Double word pointer format for area-crossing register indirect addressing

Error Information

If an invalid E_TYPE is entered, the function is not executed and the signal state of ENO is set to 0.

Example

Figure 4-1 shows how the Indirect Block Move instruction works. If the signal state of input I 0.0 is 1 (activated), the function is executed. S_DATA points to DB1.DBX0.0 which contains the pointer DB1.DBX50.0 (the starting location of the source data). D_DATA points to DB1.DBX20.0, which contains the pointer DB2.DBX10.0 (the starting location of the destination data). After the function is executed, a block of two words is moved.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

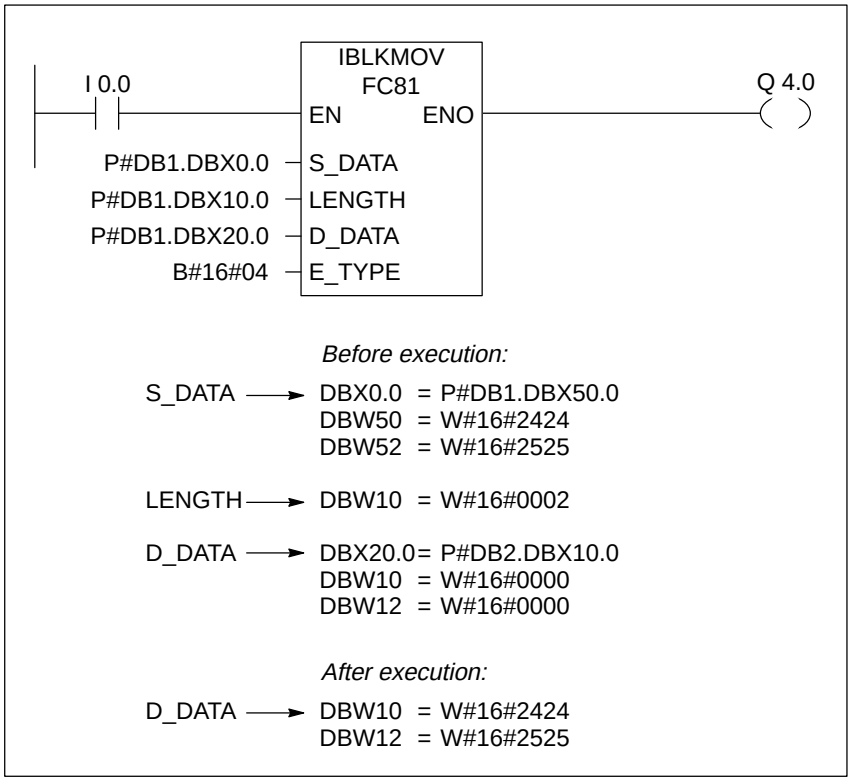


Figure 4-1 Indirect Block Move (IBLKMOV)

4.2 Pack Data (PACK): FB86

Description

The Pack Data (PACK) function block moves data interleaved between random locations and a table. The direction of transfer is specified by DIR. Each PACK instruction processes up to five blocks of data (P_DATA1 to P_DATA5). If DIR indicates ‘‘to,’’ the PACK moves data from these locations to the indicated table. Similarly, if the DIR indicates ‘‘from,’’ data is distributed from the table to the locations.

The rules for packing ‘‘to’’ a table are as follows:

- Single bits (BOOL) are packed into the next available bit in the table.
- 8-bit data types are packed into the next available byte in the table. When a byte is written to the table, unused bits in the previous byte are filled with zeros.
- 16- and 32-bit data types are packed into the next available word in the table. When a word is written to the table, unused bits in the previous word are filled with zeros.

The rules for packing data ‘‘from’’ a table are as follows:

- Sections of a table cannot be skipped.
- All BOOL points specified are packed from the table.
- 8-bit data types are packed from the first available byte in the table. That is, unused bits in the previous byte of the table are not included as part of a byte that is packed from the table.
- 16- and 32-bit data types are packed from the first available word in the table. That is, unused bits in the previous word of the table are not included as part of a word that is packed from the table.

Valid data types of the ANY pointer which are supported by the PACK function block are:

- BOOL
- WORD
- INT
- BYTE
- DINT
- REAL
- CHAR
- DWORD

Parameters

Table 4-2 describes the Pack Data (PACK) parameters.

Table 4-2 Pack Data (FB86) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function block is executed without error
TABLE	Input	Pointer	I, Q, M, D	Points to the start of the table
P_DATA1	Input	Any	I, Q, M, D	Points to the start of a block of data to pack
P_DATA2	Input	Any	I, Q, M, D	Points to the start of a block of data to pack
P_DATA3	Input	Any	I, Q, M, D	Points to the start of a block of data to pack
P_DATA4	Input	Any	I, Q, M,D	Points to the start of a block of data to pack
P_DATA5	Input	Any	I, Q, M, D	Points to the start of a block of data to pack
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
DIR	Static	BOOL	I, Q, M, D, L	Direction of pack. A signal state of 0 = to, and 1 = from

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any of the conditions shown in Table 4-3 occur, the function block is not executed. The signal state of ENO is set to 0 and ERR_CODE is set appropriately.

Table 4-3 Error Conditions for FB86

ERR_CODE	Explanation
W#16#0001	Invalid memory type specified for a function parameter
W#16#0002	Invalid E_TYPE

Example

Figure 4-2 shows how the Pack Data instruction works. If the signal state of input I0.0 is 1 (activated), the PACK function block is executed. In this example, there are four blocks of data being packed ‘‘to’’ the table.

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1 and ERR_CODE is set equal to W#16#0000.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

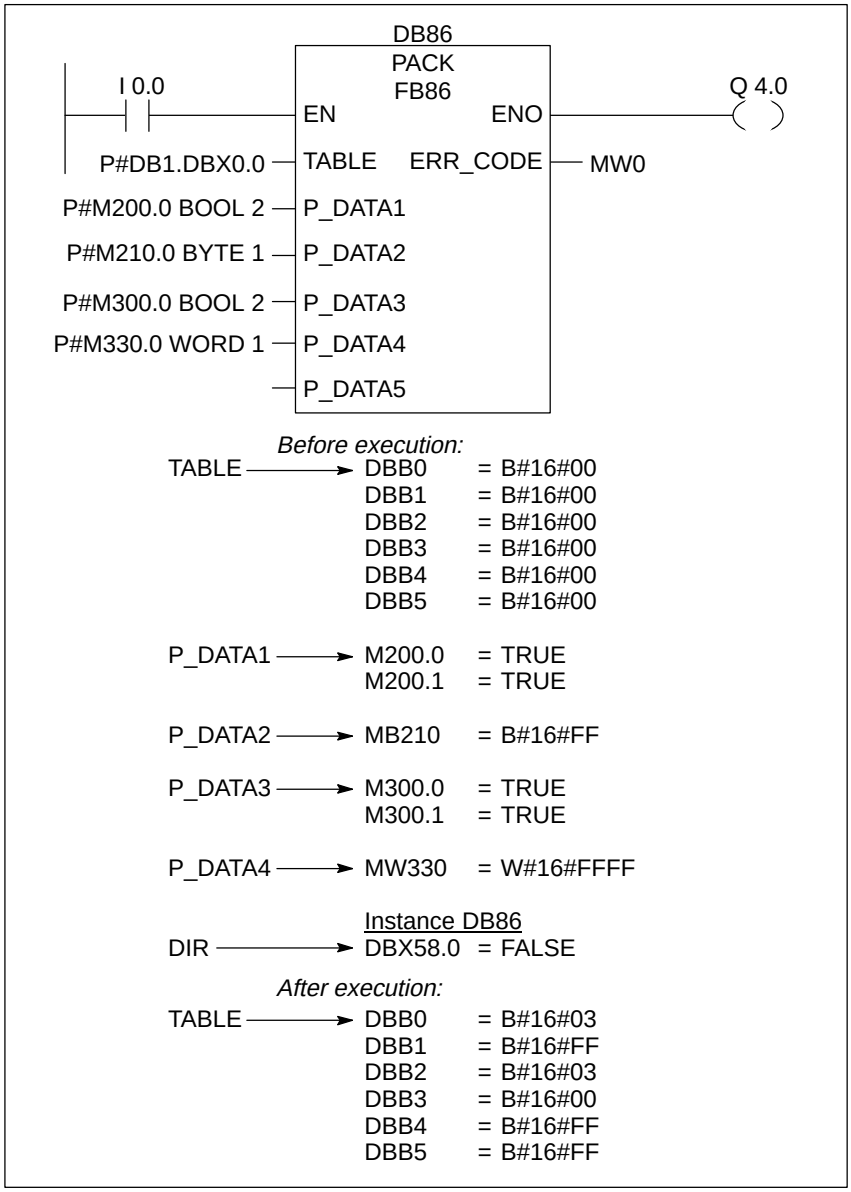


Figure 4-2 PACK DATA (PACK)

Timer Function and Function Blocks

5

This section describes the timer function (FC) and function blocks (FBs) that you can add to your standard set of instructions to provide additional programming flexibility.

5.1 Software Timer On Delay—Retentive (TONR): FC80

Description

The Software Timer On Delay—Retentive (TONR) function accumulates time until the current value of elapsed time (ET) equals or exceeds the preset value (PV). Since TONR uses the execution time of the last cycle of the OB (Organization Block) in which it runs to accumulate time, this function is intended to be used only in the OBs that have a repetitive nature, such as OB1 and the cyclic OBs.

Note

You must move the OB scan time from the start-up local variables in the variable declaration table of the OB to the global variable DELTA_T.

As long as the signal state of RESET is 0, the signal state of TMR_EN is 1, and ET is less than PV, the TONR function adds DELTA_T to ET. If the signal state of TMR_EN is not 1, then no time is added to ET. When ET reaches or exceeds PV, the signal state of output Q is set to 1. Once Q turns on, it remains on and ET is held at the last value until reset. The function resets ET to 0 and turns off output Q when the signal state of RESET is 1.

Parameters

Table 5-1 describes the Software Timer On Delay–Retentive (TONR) parameters.

Table 5-1 Software Timer On Delay—Retentive (FC80) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
TMR_EN	Input	BOOL	I, Q, M, D, L	Enables timer to accumulate time
RESET	Input	BOOL	I, Q, M, D, L	If RESET = 1, the timer is reset to 0
PV	Input	DINT	I, Q, M, D, L, P or constant	Preset value
DELTA_T	Input	INT	I, Q, M, D, L or constant	OB scan time for previous cycle
Q	Output	BOOL	Q, M, D, L	If ET equals or exceeds PV, the signal state of Q is set to 1
ET	Input_Output	DINT	I, Q, M, D, L	Current value of elapsed time

Error Information

This function does not detect any error conditions.

Example

Figure 5-1 shows how the TONR instruction works. If the signal state of input I 0.0 is 1 (activated), the TONR function is executed. If the signal state of I 0.1 is 1, the signal state of I 0.2 is 0, and ET is less than PV, then DELTA_T is added to ET. If ET is less than PV, the signal state of Q 1.1 will remain 0.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

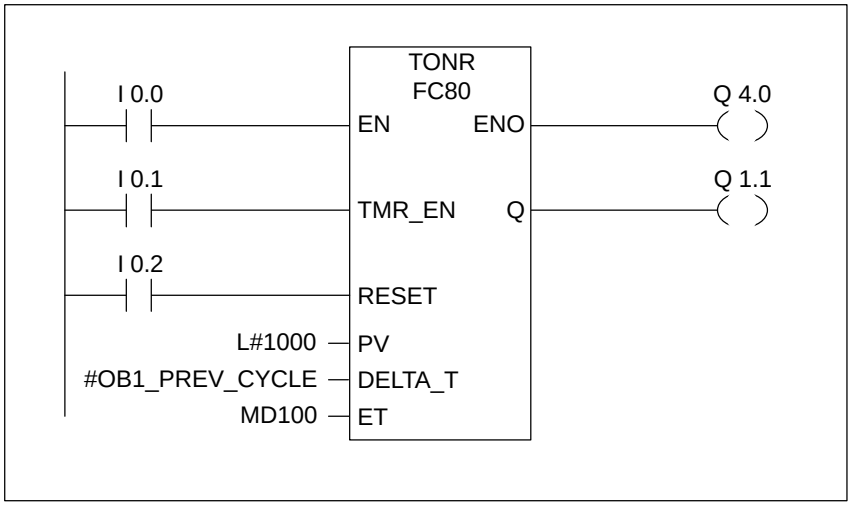


Figure 5-1 Timer On Delay—Retentive (TONR)

5.2 Discrete Control Alarm Timer (DCAT): FB81

Description

The Discrete Control Alarm Timer (DCAT) function block accumulates time from the command (CMD) input transition to open (or close) until the preset time (PT) is exceeded or the feedback input (O_FB or C_FB) indicates the device opened (or closed) within the prescribed time. If the preset time is exceeded before the feedback is received, the corresponding alarm is turned on. If the input command switches states before the preset time, the time is restarted.

- When the signal state of CMD input transitions from 0 to 1, the signal state of Q is set to 1, ET is set to zero, the signal states of both alarm outputs (OA, CA) are set to 0, and the signal state of CMD_HIS is set to 1.
- When the signal state of CMD input transitions from 1 to 0, the signal state of Q is set to 0, ET is set to zero, the signal states of both alarm outputs (OA, CA) are set to 0, and the signal state of CMD_HIS is set to 0.
- When the signal states of both CMD and CMD_HIS are 1, and the signal state of O_FB is 0, then the time (msec) delta since the last function block execution is added to ET. If ET exceeds PT, then the signal state of OA is set to 1, otherwise it is set to 0. The signal state of CMD_HIS is set equal to CMD.
- When the signal states of both CMD and CMD_HIS are 1, the signal state of O_FB is 1, and the signal state of C_FB is 0, then the signal state of the OA is set to 0. ET is set equal to PT so that if the signal state of O_FB is later set to 0, the alarm will be set on the next function block execution. The signal state of CMD_HIS is set equal to CMD.
- When the signal states of both CMD and CMD_HIS are 0, and the signal state of C_FB is 0, then the time (msec) delta since the last function block execution is added to ET. If ET exceeds PT, then the signal state of the CA is set to 1, otherwise it is set to 0. The signal state of CMD_HIS is set equal to CMD.
- When the signal states of both CMD and CMD_HIS are 0, the signal state of O_FB is 0, and the C_FB is 1, then the signal state of the CA is set to 0. ET is set equal to PT so that if the signal state of C_FB is later set to 0, the alarm will be set on the next function block execution. The signal state of CMD_HIS is set equal to CMD.
- If the signal states of both O_FB and C_FB are 1 simultaneously, then this is considered an error condition and the signal states of both alarm outputs are set to 1.

Parameters Table 5-2 describes the Discrete Control Alarm Timer (DCAT) parameters.

Table 5-2 Discrete Control Alarm Timer (FB81) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function block is executed without error
CMD	Input	BOOL	I, Q, M, D, L	A signal state of 0 indicates a close command and a signal state of 1 indicates an open command
O_FB	Input	BOOL	I, Q, M, D, L	Open feedback input
C_FB	Input	BOOL	I, Q, M, D, L	Closed feedback input
Q	Output	BOOL	I, Q, M, D, L	Follows CMD input
OA	Output	BOOL	I, Q, M, D, L	Open alarm output
CA	Output	BOOL	I, Q, M, D, L	Closed alarm output
ET	Static	DINT	I, Q, M, D, L	Current count of elapsed time, where 1 count = 1 msec
PT	Static	DINT	I, Q, M, D, L	Timer preset count, where 1 count = 1 msec
PREV_TIME	Static	DWORD	I, Q, M, D, L	Previous system time
CMD_HIS	Static	BOOL	I, Q, M, D, L	CMD history bit

Error Information

This function block does not detect any error conditions.

Example

Figure 5-2 shows how the DCAT instruction works. If the signal state of input I0.0 is 1 (activated), the DCAT function block is executed. In this example, the CMD input is transitioning from a signal state of 0 to 1 as indicated by CMD_HIS and CMD. Based upon these conditions, Q and CMD_HIS are given a signal state of 1, ET is set to 0, and both alarm outputs OA and CA are given a signal state of 0.

If the function block is executed without error, the signal state of ENO and Q4.0 are set to 1.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

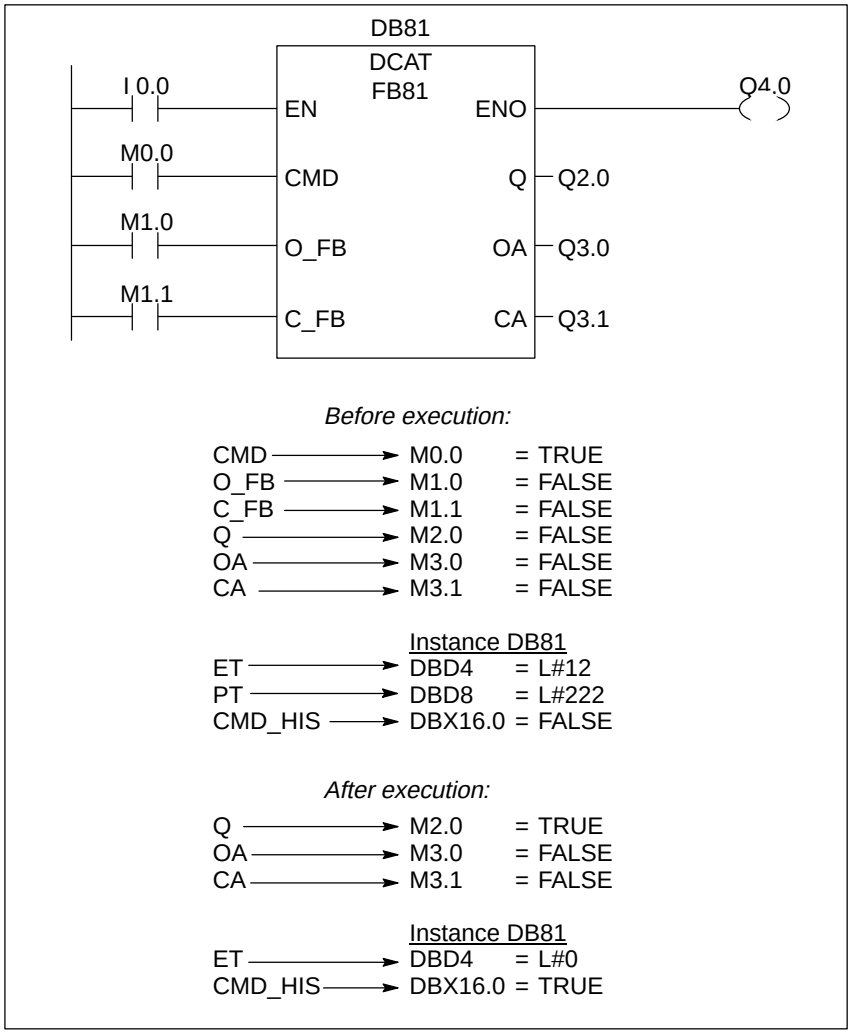


Figure 5-2 Discrete Control Alarm Timer (DCAT)

5.3 Motor Control Alarm Timer (MCAT): FB82

Description

The Motor Control Alarm Timer (MCAT) function block accumulates time from the on transition of one of the command inputs (open or close) until the preset time is exceeded or the corresponding feedback input indicates the device completed the commanded operation within the prescribed time. If the preset time is exceeded before the feedback is received, the corresponding alarm is turned on. Descriptions of the MCAT reaction to the various input conditions are summarized in the MCAT Truth Table (see Table 5-3).

Table 5-3 MCAT Truth Table

INPUTS								OUTPUTS								
ET	**O_HIS**	**C_HIS**	**O_CMD**	**C_CMD**	**S_CMD**	**O_FB**	**C_FB**	**OO**	**CO**	**OA**	**CA**	**ET**	**O_HIS**	**C_HIS**	**Q**	**STATE**
X	1	1	X	X	X	X	X	0	0	1	1	PT	0	0	0	Alarm
X	X	X	X	X	X	1	1	0	0	1	1	PT	0	0	0	Alarm
X	X	X	X	X	1	X	X	0	0	0	0	X	0	0	1	Stop
X	X	X	1	1	X	X	X	0	0	0	0	X	0	0	1	Stop
X	0	X	1	0	0	X	X	1	0	0	0	0	1	0	1	Begin Open
<PT	1	0	X	0	0	0	X	1	0	0	0	INC	1	0	1	Opening
X	1	0	X	0	0	1	0	0	0	0	0	PT	1	0	1	Opened
>=PT	1	0	X	0	0	0	X	0	0	1	0	PT	1	0	0	Open Alarm
X	X	0	0	1	0	X	X	0	1	0	0	0	0	1	1	Begin Close
<PT	0	1	0	X	0	X	0	0	1	0	0	INC	0	1	1	Closing
X	0	1	0	X	0	0	1	0	0	0	0	PT	0	1	1	Closed
>=PT	0	1	0	X	0	X	0	0	0	0	1	PT	0	1	0	Close Alarm
X	0	0	0	0	0	X	X	0	0	0	0	X	0	0	1	Stopped

Where:

INC = Add the time (msec) delta since the last function block execution to ET
PT = PT is set equal to ET
X = Not applicable
<PT = ET < PT
>= PT = ET >= PT

Parameters

Table 5-4 describes the Motor Control Alarm Timer (MCAT) parameters.

Table 5-4 Motor Control Alarm Timer (FB82) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
O_CMD	Input	BOOL	I, Q, M, D, L	Open command input
C_CMD	Input	BOOL	I, Q, M, D, L	Close command input
S_CMD	Input	BOOL	I, Q, M, D, L	Stop command input
O_FB	Input	BOOL	I, Q, M, D, L	Open feedback input
C_FB	Input	BOOL	I, Q, M, D, L	Closed feedback input
OO	Output	BOOL	I, Q, M, D, L	Open output
CO	Output	BOOL	I, Q, M, D, L	Close output
OA	Output	BOOL	I, Q, M, D, L	Open alarm output
CA	Output	BOOL	I, Q, M, D, L	Closed alarm output
Q	Output	BOOL	I, Q, M, D, L	A signal state of 1 indicates an alarm condition
ET	Static	DINT	I, Q, M, D, L	Current count of elapsed time, where 1 count = 1 msec
PT	Static	DINT	I, Q, M, D, L	Timer preset count, where 1 count = 1 msec
PREV_TIME	Static	DWORD	I, Q, M, D, L	Previous system time
O_HIS	Static	BOOL	I, Q, M, D, L	Open history bit
C_HIS	Static	BOOL	I, Q, M, D, L	Close history bit

Error Information

This function block does not detect any error conditions.

Example

Figure 5-3 shows how the MCAT instruction works. If the signal state of input I0.0 is 1 (activated), the MCAT function block is executed. In this example, based upon the status of the inputs, the MCAT is in the OPENING state and the outputs are set appropriately.

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

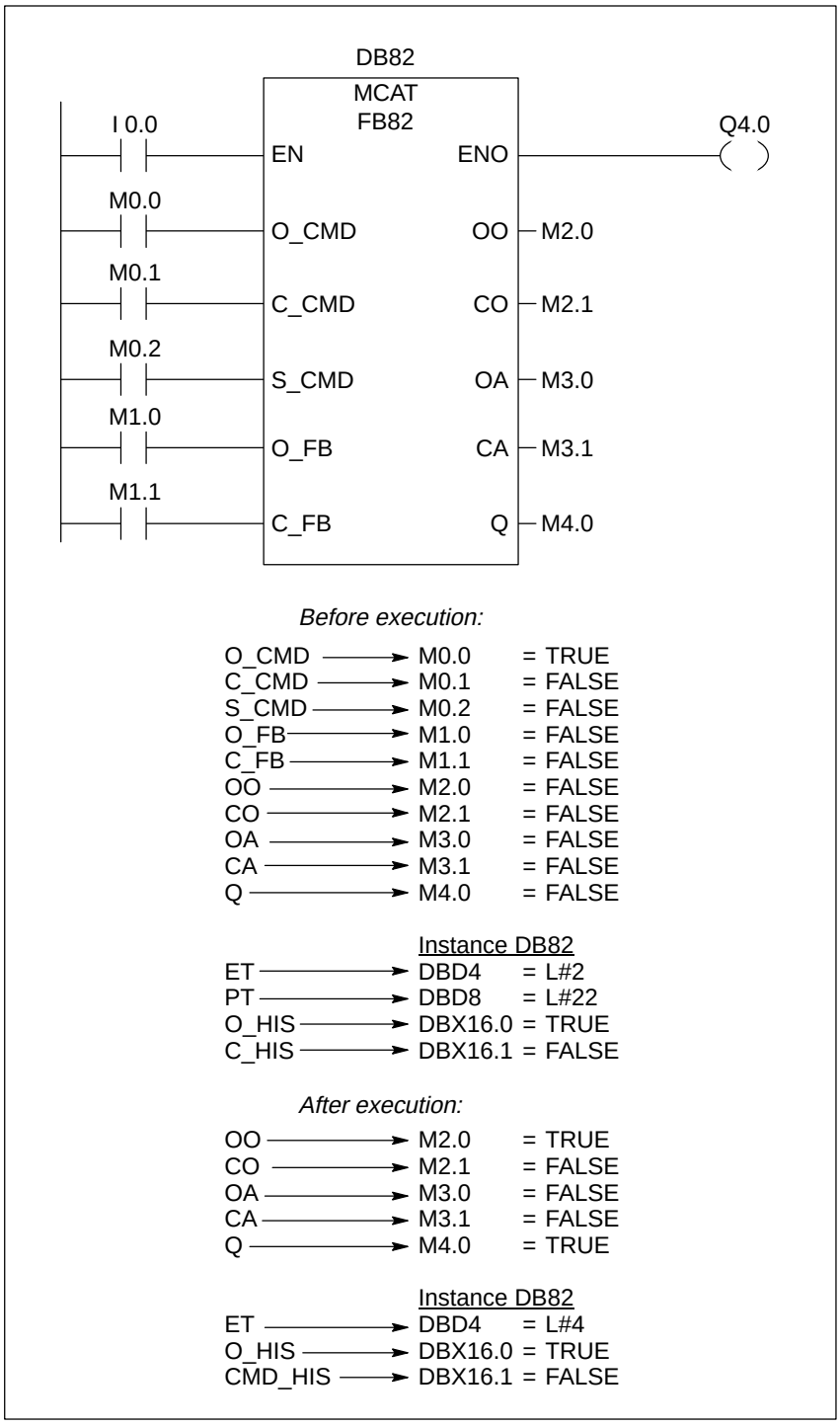


Figure 5-3 Motor Control Alarm Timer (MCAT)

5.4 Event Maskable Drum (DRUM): FB85

Description

The Event Maskable Drum (DRUM) function block drives the programmed output bits (OUT1 to OUT16) and the word output (OUT_WORD) with the programmed values (OUT_VAL) of the appropriate step conditioned by the enabling masks (S_MASK) for that step while the DRUM remains on that step. The DRUM will advance to the next step when either the event for the step is true and the programmed time for the current step has expired, or the jog input transitions from 0 to 1. When the signal state of RESET is 1, the DRUM is reset, which sets the current step equal to the preset step (DSP).

The amount of time spent on a step is determined by the product of the DRUM time base preset (DTBP) and the count preset/step values (S_PRESET) corresponding to each step. At the start of a new step, this calculated value is loaded into DCC which contains the time remaining for the current step. For example, if DTBP is equal to 2, and the preset value for step 1 is equal to 100 (100 msec), then DCC equals 200 (200 msec).

A step may be programmed with a time value, or an event, or both. Steps with an event bit and a time value of zero, advance to the next step as soon as the signal state of the event bit is 1. Steps with only a time start timing immediately. Steps with an event bit and a time value greater than 0 begin timing when the signal state of the event bit is 1. The event bits are initialized with a signal state of 1.

When the step pointer is on the last programmed step (LST_STEP) and the time for that step has expired, the signal state of output (Q) is set to 1, otherwise it is set to 0. Once Q is set, the DRUM remains on that step until reset.

The configurable mask (S_MASK) allows selection of the individual bits in the output word (OUT_WORD) and the output bits (OUT1 to OUT16) to be set/reset by the output values (OUT_VAL). When a bit of the configurable mask has a signal state of 1, the OUT_VAL value sets/resets the corresponding bit. When a bit of the configurable mask has a signal state of 0, the corresponding bit is left unchanged. Each of the bits of the configurable mask for all 16 steps is initialized with a signal state of 1.

The output bit OUT1 corresponds with the least-significant bit of the word output OUT_WORD. The output bit OUT16 corresponds with the most-significant bit of the word output OUT_WORD.

Parameters Table 5-5 describes the Event Maskable Drum (DRUM) parameters.

Table 5-5 Event Maskable Drum (FB85) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
RESET	Input	BOOL	I, Q, M, D, L	A signal state of 1 indicates a reset condition
JOG	Input	BOOL	I, Q, M, D, L	A transition of a signal state of 0 to 1 advances DRUM to the next step
DRUM_EN	Input	BOOL	I, Q, M, D, L	A signal state of 1 enables the drum to advance based on the event and timing criteria
LST_STEP	Input	BYTE	I, Q, M, D, L, or constant	Step number of the last step programmed
EVENT1	Input	BOOL	I, Q, M, D, L	Event bit 1; initial signal state is 1
EVENT2	Input	BOOL	I, Q, M, D, L	Event bit 2; initial signal state is 1
EVENT3	Input	BOOL	I, Q, M, D, L	Event bit 3; initial signal state is 1
EVENT4	Input	BOOL	I, Q, M, D, L	Event bit 4; initial signal state is 1
EVENT5	Input	BOOL	I, Q, M, D, L	Event bit 5; initial signal state is 1
EVENT6	Input	BOOL	I, Q, M, D, L	Event bit 6; initial signal state is 1
EVENT7	Input	BOOL	I, Q, M, D, L	Event bit 7; initial signal state is 1
EVENT8	Input	BOOL	I, Q, M, D, L	Event bit 8; initial signal state is 1
EVENT9	Input	BOOL	I, Q, M, D, L	Event bit 9; initial signal state is 1
EVENT10	Input	BOOL	I, Q, M, D, L	Event bit 10; initial signal state is 1
EVENT11	Input	BOOL	I, Q, M, D, L	Event bit 11; initial signal state is 1
EVENT12	Input	BOOL	I, Q, M, D, L	Event bit 12; initial signal state is 1
EVENT13	Input	BOOL	I, Q, M, D, L	Event bit 13; initial signal state is 1
EVENT14	Input	BOOL	I, Q, M, D, L	Event bit 14; initial signal state is 1
EVENT15	Input	BOOL	I, Q, M, D, L	Event bit 15; initial signal state is 1
EVENT16	Input	BOOL	I, Q, M, D, L	Event bit 16; initial signal state is 1
OUT1	Output	BOOL	I, Q, M, D, L	Output bit 1
OUT2	Output	BOOL	I, Q, M, D, L	Output bit 2
OUT3	Output	BOOL	I, Q, M, D, L	Output bit 3
OUT4	Output	BOOL	I, Q, M, D, L	Output bit 4
OUT5	Output	BOOL	I, Q, M, D, L	Output bit 5
OUT6	Output	BOOL	I, Q, M, D, L	Output bit 6
OUT7	Output	BOOL	I, Q, M, D, L	Output bit 7
OUT8	Output	BOOL	I, Q, M, D, L	Output bit 8
OUT9	Output	BOOL	I, Q, M, D, L	Output bit 9
OUT10	Output	BOOL	I, Q, M, D, L	Output bit 10

Table 5-5 Event Maskable Drum (FB85) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
OUT11	Output	BOOL	I, Q, M, D, L	Output bit 11
OUT12	Output	BOOL	I, Q, M, D, L	Output bit 12
OUT13	Output	BOOL	I, Q, M, D, L	Output bit 13
OUT14	Output	BOOL	I, Q, M, D, L	Output bit 14
OUT15	Output	BOOL	I, Q, M, D, L	Output bit 15
OUT16	Output	BOOL	I, Q, M, D, L	Output bit 16
Q	Output	BOOL	I, Q, M, D, L	A signal state of 1 indicates the last step timed out
OUT_WORD	Output	WORD	I, Q, M, D, L, P	Word location to which the drum writes the output values
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
JOG_HIS	Static	BOOL	I, Q, M, D, L	Jog history bit
EOD	Static	BOOL	I, Q, M, D, L	A signal state of 1 indicates the last step timed out
DSP	Static	BYTE	I, Q, M, D, L, P	Drum step preset
DSC	Static	BYTE	I, Q, M, D, L, P	Drum step current
DCC	Static	DWORD	I, Q, M, D, L, P	Drum count current
DTBP	Static	WORD	I, Q, M, D, L, P	Drum time base preset
PREV_TIME	Static	DWORD	I, Q, M, D, L	Previous system time
S_PRESET	Static	ARRAY of WORD	I, Q, M, D, L	Count preset for each step [1 to 16], where 1 count = 1 msec
OUT_VAL	Static	ARRAY of BOOL	I, Q, M, D, L	Output values for each step [1 to 16, 0 to 15]
S_MASK	Static	ARRAY of BOOL	I, Q, M, D, L	Configurable mask for each step [1 to 16, 0 to 15] Initial signal states are 1

Error Information

If any of the conditions shown in Table 5-6 occur, the DRUM will remain on the current step. The signal state of ENO is set to 0 and ERR_CODE is set appropriately.

Table 5-6 Error Conditions for FB85

ERR_CODE	Explanation
W#16#000B	LST_STEP value is less than 1 or greater than 16
W#16#000C	DSC value is less than 1 or greater than LST_STEP
W#16#000D	DSP value is less than 1 or greater than LST_STEP

Example

Figure 5-4 shows how the DRUM instruction works. If the signal state of input I0.0 is 1 (activated), the DRUM function block is executed. In this example, the DRUM advances from step 1 to step 2. The output bits (OUT1, etc.) and OUT_WORD are set based upon the configured mask for step 2 and the OUT_VAL bits for step 2.

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1 and ERR_CODE is set equal to W#16#0000.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

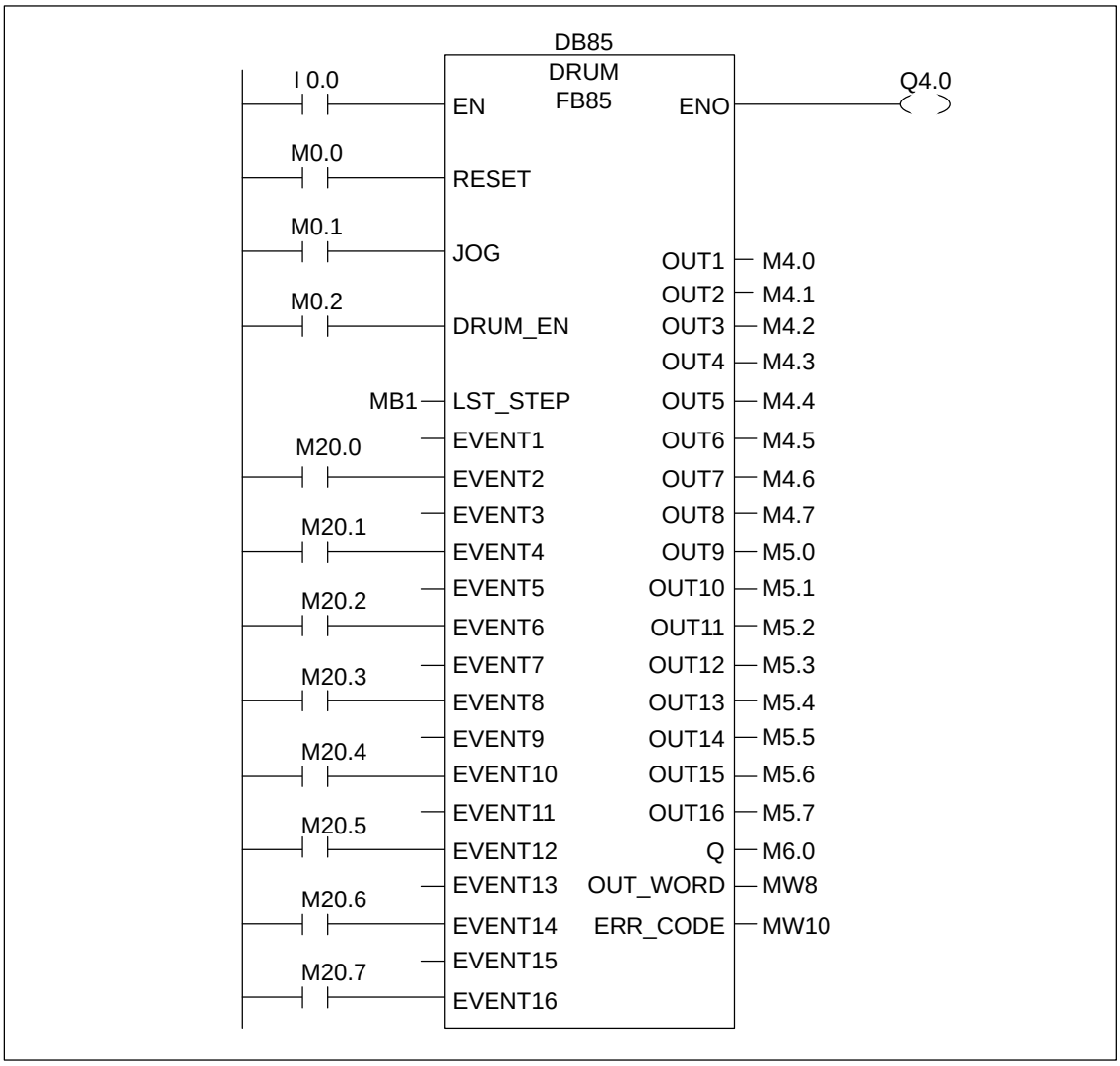


Figure 5-4 Event Maskable Drum (DRUM)

Before execution:

Inputs

RESET	M0.0	= FALSE
JOG	M0.1	= FALSE
DRUM_EN	M0.2	= TRUE
LST_STEP	MB1	= B#16#08
EVENT2	M20.0	= FALSE
EVENT4	M20.1	= FALSE
EVENT6	M20.2	= FALSE
EVENT8	M20.3	= FALSE
EVENT10	M20.4	= FALSE
EVENT12	M20.5	= FALSE
EVENT14	M20.6	= FALSE
EVENT16	M20.7	= FALSE

Instance DB85

JOG_HIS	DBX12.0	= FALSE
EOD	DBX12.1	= FALSE
DSP	DBB13	= W#16#0001
DSC	DBB14	= W#16#0001
DCC	DBD16	= DW#16#0000000A
DTBP	DBW20	= W#16#0001
S_PRESET [1]	DBW26	= W#16#0064
S_PRESET [2]	DBW28	= W#16#00C8
OUT_VAL [1,0]	DBX58.0	= TRUE
OUT_VAL [1,1]	DBX58.1	= TRUE
OUT_VAL [1,2]	DBX58.2	= TRUE
OUT_VAL [1,3]	DBX58.3	= TRUE
OUT_VAL [1,4]	DBX58.4	= TRUE
OUT_VAL [1,5]	DBX58.5	= TRUE
OUT_VAL [1,6]	DBX58.6	= TRUE
OUT_VAL [1,7]	DBX58.7	= TRUE
OUT_VAL [1,8]	DBX59.0	= TRUE
OUT_VAL [1,9]	DBX59.1	= TRUE
OUT_VAL [1,10]	DBX59.2	= TRUE
OUT_VAL [1,11]	DBX59.3	= TRUE
OUT_VAL [1,12]	DBX59.4	= TRUE
OUT_VAL [1,13]	DBX59.5	= TRUE
OUT_VAL [1,14]	DBX59.6	= TRUE
OUT_VAL [1,15]	DBX59.7	= TRUE
OUT_VAL [2,0]	DBX60.0	= FALSE
OUT_VAL [2,1]	DBX60.1	= FALSE
OUT_VAL [2,2]	DBX60.2	= FALSE
OUT_VAL [2,3]	DBX60.3	= FALSE
OUT_VAL [2,4]	DBX60.4	= FALSE
OUT_VAL [2,5]	DBX60.5	= FALSE
OUT_VAL [2,6]	DBX60.6	= FALSE
OUT_VAL [2,7]	DBX60.7	= FALSE
OUT_VAL [2,8]	DBX61.0	= FALSE
OUT_VAL [2,9]	DBX61.1	= FALSE
OUT_VAL [2,10]	DBX61.2	= FALSE
OUT_VAL [2,11]	DBX61.3	= FALSE
OUT_VAL [2,12]	DBX61.4	= FALSE
OUT_VAL [2,13]	DBX61.5	= FALSE
OUT_VAL [2,14]	DBX61.6	= FALSE
OUT_VAL [2,15]	DBX61.7	= FALSE

Before execution (continued):

S_MASK [2,0]	DBX92.0	= FALSE
S_MASK [2,1]	DBX92.1	= TRUE
S_MASK [2,2]	DBX92.2	= TRUE
S_MASK [2,3]	DBX92.3	= TRUE
S_MASK [2,4]	DBX92.4	= TRUE
S_MASK [2,5]	DBX92.5	= FALSE
S_MASK [2,6]	DBX92.6	= TRUE
S_MASK [2,7]	DBX92.7	= TRUE
S_MASK [2,8]	DBX93.0	= FALSE
S_MASK [2,9]	DBX93.1	= FALSE
S_MASK [2,10]	DBX93.2	= TRUE
S_MASK [2,11]	DBX93.3	= TRUE
S_MASK [2,12]	DBX93.4	= TRUE
S_MASK [2,13]	DBX93.5	= TRUE
S_MASK [2,14]	DBX93.6	= FALSE
S_MASK [2,15]	DBX93.7	= TRUE

Outputs

Q	M6.0	= FALSE
OUTWORD	MW8	= W#16#FFFF
OUT1	M4.0	= TRUE
OUT2	M4.1	= TRUE
OUT3	M4.2	= TRUE
OUT4	M4.3	= TRUE
OUT5	M4.4	= TRUE
OUT6	M4.5	= TRUE
OUT7	M4.6	= TRUE
OUT8	M4.7	= TRUE
OUT9	M5.0	= TRUE
OUT10	M5.1	= TRUE
OUT11	M5.2	= TRUE
OUT12	M5.3	= TRUE
OUT13	M5.4	= TRUE
OUT14	M5.5	= TRUE
OUT15	M5.6	= TRUE
OUT16	M5.7	= TRUE

After execution:

OUT1	M4.0	= TRUE
OUT2	M4.1	= FALSE
OUT3	M4.2	= FALSE
OUT4	M4.3	= FALSE
OUT5	M4.4	= FALSE
OUT6	M4.5	= TRUE
OUT7	M4.6	= FALSE
OUT8	M4.7	= FALSE
OUT9	M5.0	= TRUE
OUT10	M5.1	= TRUE
OUT11	M5.2	= FALSE
OUT12	M5.3	= FALSE
OUT13	M5.4	= FALSE
OUT14	M5.5	= FALSE
OUT15	M5.6	= TRUE
OUT16	M5.7	= FALSE
Q	M6.0	= FALSE
OUTWORD	MW8	= W#16#4321
ERR_CODE	MW10	= W#16#0000

Instance DB85

JOG_HIS	DBX12.0	= FALSE
EOD	DBX12.1	= FALSE
DSC	DBB14	= W#16#0002
DCC	DBD16	= DW#16#000000C8

Figure 5-4 Event Maskable Drum (DRUM) (continued)

Convert Functions and Function Block

6

This section describes conversion functions (FCs) and function block (FB) that you can add to your standard set of instructions to provide additional programming flexibility.

6.1 Seven Segment Decoder (SEG): FC93

Description

The Seven Segment Decoder (SEG) function converts each of the four hexadecimal digits in the designated source data word (IN) into four equivalent 7-segment display codes and writes it to the output destination double word (OUT).

Figure 6-1 shows the relationship between the input hex digits and the output bit patterns.

Digit	– g f e d c b a	Display
0 0 0 0	0 0 1 1 1 1 1 1	0
0 0 0 1	0 0 0 0 0 1 1 0	1
0 0 1 0	0 1 0 1 1 0 1 1	2
0 0 1 1	0 1 0 0 1 1 1 1	3
0 1 0 0	0 1 1 0 0 1 1 0	4
0 1 0 1	0 1 1 0 1 1 0 1	5
0 1 1 0	0 1 1 1 1 1 0 1	6
0 1 1 1	0 0 0 0 0 1 1 1	7
1 0 0 0	0 1 1 1 1 1 1 1	8
1 0 0 1	0 1 1 0 0 1 1 1	9
1 0 1 0	0 1 1 1 0 1 1 1	A
1 0 1 1	0 1 1 1 1 1 0 0	b
1 1 0 0	0 0 1 1 1 0 0 1	C
1 1 0 1	0 1 0 1 1 1 1 0	d
1 1 1 0	0 1 1 1 1 0 0 1	E
1 1 1 1	0 1 1 1 0 0 0 1	F

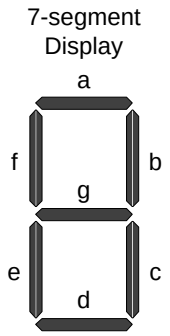


Figure 6-1 Seven Segment Output Bit Patterns

Parameters

Table 6-1 describes the Seven Segment Decoder (SEG) parameters.

Table 6-1 Seven Segment Decoder (FC93) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	WORD	I, M, D, P, or constant	Source data word in four hexadecimal digits
OUT	Output	DWORD	Q, M, D, L, P	Destination bit pattern in four bytes

Error Information

This function does not detect any error conditions.

Example

Figure 6-2 shows how the SEG instruction works. If the signal state of input I 0.0 is 1 (activated), the SEG function is executed.

If the function is executed without error, the signal states of ENO and Q 4.0 are set to 1.

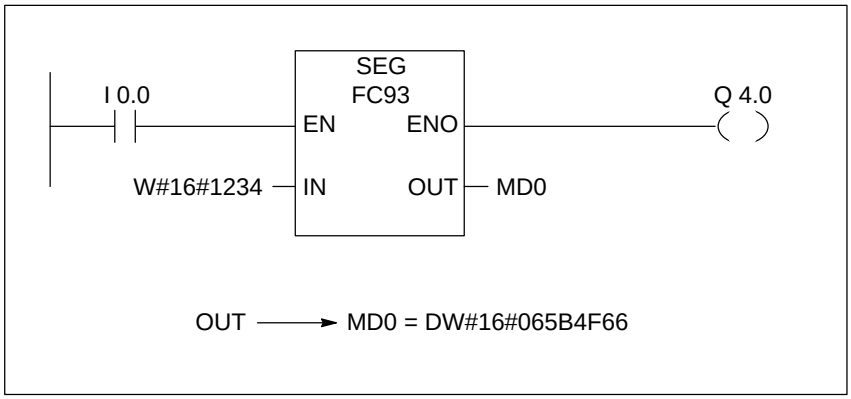


Figure 6-2 Seven Segment Decoder (SEG)

6.2 ASCII to Hex (ATH): FC94

Description

The ASCII to Hex (ATH) function converts the ASCII character string pointed to by IN into packed hexadecimal digits and stores these in the destination table pointed to by OUT. Since 8 bits are required for the ASCII character and only 4 bits for the hexadecimal digit, the output word length is only half of the input word length. The ASCII characters are converted and placed into the hexadecimal output in the same order as they are read in. If there is an odd number of ASCII characters, the hexadecimal digit is padded with zeros in the right-most nibble of the last converted hexadecimal digit.

Parameters

Table 6-2 describes the ASCII to Hex (ATH) parameters.

Table 6-2 ASCII to Hex (FC94) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	Pointer*	I, Q, M, D, L	Points to the starting location of an ASCII string
N	Input	INT	I, Q, M, L, P	Number of ASCII input characters to be converted
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
OUT	Output	Pointer*	Q, M, D, L	Points to the starting location of the table

* Double word pointer format for area-crossing register indirect addressing

Error Information

If any ASCII character is found to be invalid, it is converted as 0. The signal state of ENO is set to 0 and RET_VAL is set equal to W#16#0007.

Example

Figure 6-3 shows how the ATH instruction works. If the signal state of input I 0.0 is 1 (activated), the ATH function is executed. The N input parameter of 5 indicates that five ASCII characters are to be converted. The ASCII characters are stored in data block 1 beginning at the IN pointer location, DB1.DBX10.0. The output string will be located at the OUT pointer location beginning at DB2.DBX0.0 (data block 2). Since there is an odd number of ASCII input characters, the last hexadecimal digit contains all zeros in the right-most nibble, producing the hexadecimal value 0xC0. (Refer to Figure 6-4 for the equivalent hexadecimal value for each ASCII character.)

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

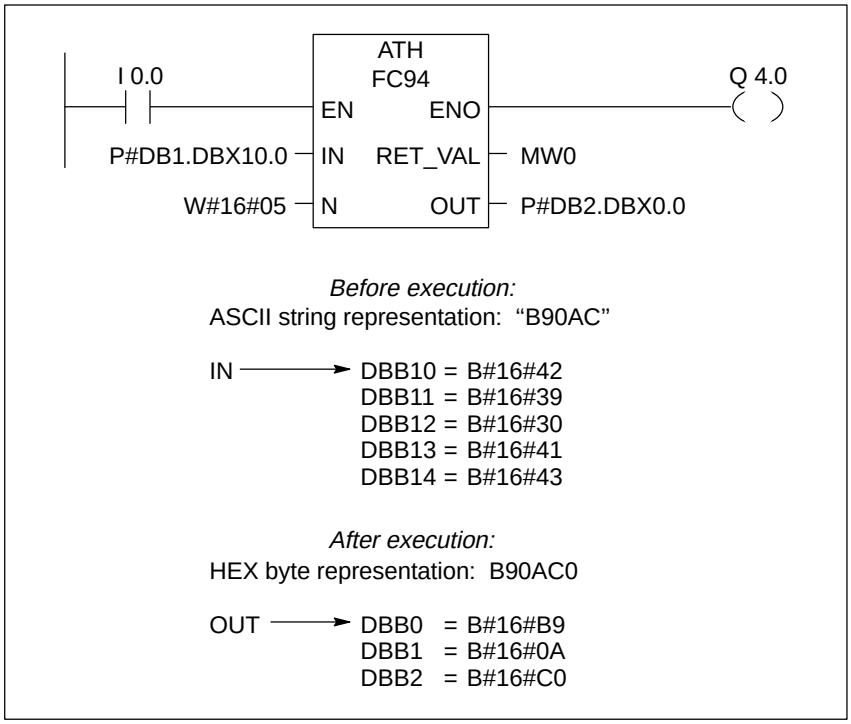


Figure 6-3 ASCII to Hex (ATH)

ASCII Character	ASCII Hex Value	Converted Hex Digit
0	30	0
1	31	1
2	32	2
3	33	3
4	34	4
5	35	5
6	36	6
7	37	7
8	38	8
9	39	9
A	41	A
B	42	B
C	43	C
D	44	D
E	45	E
F	46	F

Figure 6-4 ASCII Characters and Equivalent Hexadecimal Values

6.3 Hex to ASCII (HTA): FC95

Description

The Hex to ASCII (HTA) function converts packed hexadecimal digits, pointed to by IN, and stores them in the destination string pointed to by OUT. Since 8 bits are required for the character and only 4 bits for the hex digit, the output word length is two times that of the input word length. Each nibble of the hexadecimal digit is converted into a character in the same order as they are read in (left-most nibble of a hexadecimal digit is converted first, followed by the right-most nibble of that same digit).

Parameters

Table 6-3 describes the Hex to ASCII (HTA) parameters.

Table 6-3 Hex to ASCII (FC95) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	Pointer*	I, Q, M, D	Points to the starting location of the hexadecimal digit string
N	Input	WORD	I, Q, M, L, P	Number of hex input bytes to be converted
OUT	Output	Pointer*	Q, M, D, L	Points to the starting location of the destination table

* Double word pointer format for area-crossing register indirect addressing

Error Information

This function does not detect any error conditions.

Example

Figure 6-5 shows how the HTA instruction works. If the signal state of input I 0.0 is 1 (activated), the HTA function is executed. The N input parameter of 3 indicates that three hexadecimal characters are to be converted. The hex bytes are stored in data block 1 beginning at the IN pointer location, DB1.DBX10.0. The output string will be located at the OUT pointer location beginning at DB2.DBX0.0 (data block 2). (Refer to Figure 6-6 for the ASCII equivalent for each hex value.)

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1.

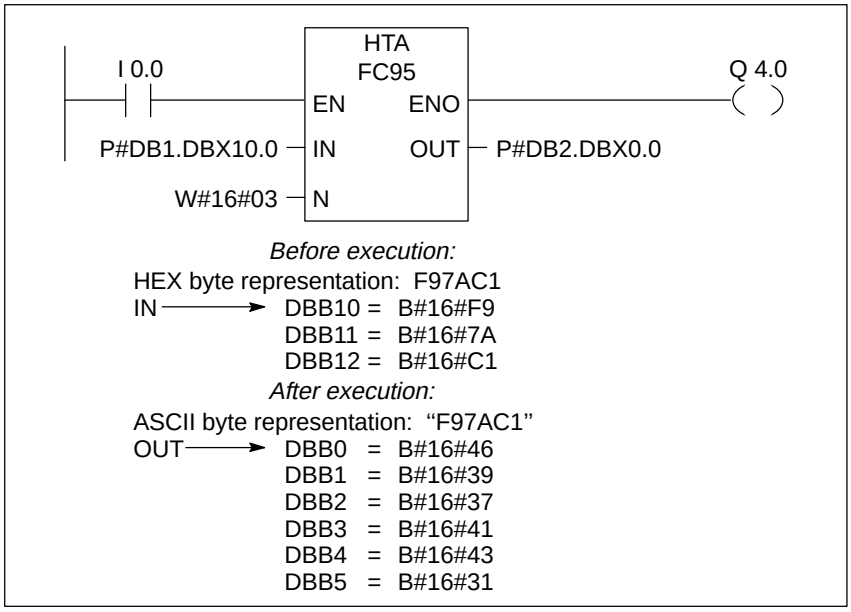


Figure 6-5 Hex to ASCII (HTA)

Hex Digit	ASCII Hex Value	Converted ASCII
0	30	0
1	31	1
2	32	2
3	33	3
4	34	4
5	35	5
6	36	6
7	37	7
8	38	8
9	39	9
A	41	A
B	42	B
C	43	C
D	44	D
E	45	E
F	46	F

Figure 6-6 Hexadecimal Digits and Equivalent ASCII Hex Values

6.4 Encode Binary Position (ENCO): FC96

Description

The Encode Binary Position (ENCO) function converts the contents of IN to the 5-bit binary number corresponding to the bit position of the right-most set bit in IN and returns the result as the function's value. If IN is either 0000 0001 or 0000 0000, a value of 0 is returned.

Parameters

Table 6-4 describes the Encode Binary Position (ENCO) parameters.

Table 6-4 Encode Binary Position (FC96) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, or constant	Value to be encoded
RET_VAL	Output	INT	Q, M, D, L, P	Value returned (contains 5-bit binary number)

Error Information

This function does not detect any error conditions.

Example

Figure 6-7 shows how the ENCO instruction works. If the signal state of input I 0.0 is 1 (activated), the ENCO function is executed.

If the function executes without error, the signal states of ENO and Q 4.0 are set to 1.

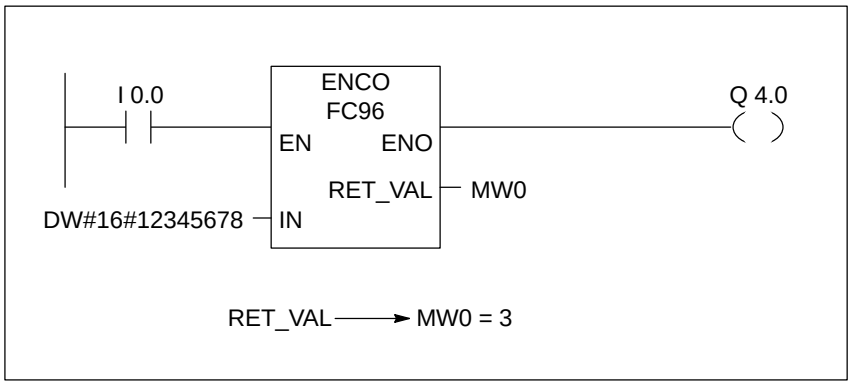


Figure 6-7 Encode Binary Position (ENCO)

6.5 Decode Binary Position (DECO): FC97

Description

The Decode Binary Position (DECO) function converts a 5-bit binary number (0 – 31) from input IN to a value by setting the corresponding bit position in the function's return value. If IN is greater than 31, a modulo 32 operation is performed to get a 5-bit binary number.

Parameters

Table 6-5 describes the Decode Binary Position (DECO) parameters.

Table 6-5 Decode Binary Position (FC97) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	WORD	I, M, D, L, P, or constant	Variable to decode
RET_VAL	Output	DWORD	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

Example

Figure 6-8 shows how the DECO instruction works. If the signal state of input I 0.0 is 1 (activated), the DECO function is executed.

If the function executes without error, the signal states of ENO and Q 4.0 are set to 1.

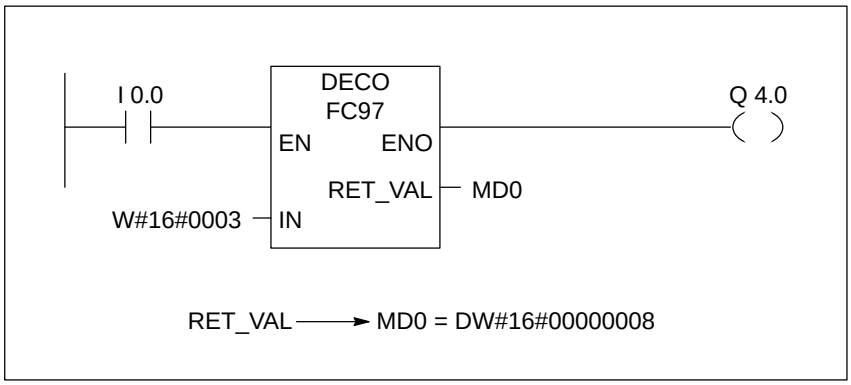


Figure 6-8 Decode Binary Position (DECO)

6.6 Tens Complement (BCDCPL): FC98

Description

The Tens Complement (BCDCPL) function returns the Tens complement of a 7-digit BCD number IN. The mathematical formula for this operation is the following:

10000000 (in BCD)
– 7-digit BCD value
= Tens complement value (in BCD)

Parameters

Table 6-6 describes the Tens Complement (BCDCPL) parameters.

Table 6-6 Tens Complement (FC98) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, or constant	7-digit BCD number
RET_VAL	Output	DWORD	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

Example

Figure 6-9 shows how the BCDCPL instruction works. If the signal state of input I 0.0 is 1 (activated), the BCDCPL function is executed.

If the function executes without error, the signal states of ENO and Q 4.0 are set to 1.

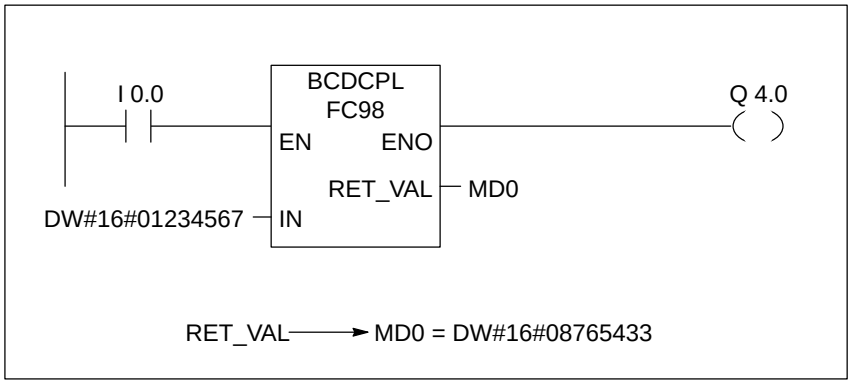


Figure 6-9 Tens Complement (BCDCPL)

6.7 Sum Number of Bits (BITSUM): FC99

Description

The Sum Number of Bits (BITSUM) function counts the number of bits that are set to a value of 1 in the input IN and returns this as the function's value.

Parameters

Table 6-7 describes the Sum Number of Bits (BITSUM) parameters.

Table 6-7 Sum Number of Bits (FC99) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, or constant	Variable to count bits in
RET_VAL	Output	INT	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

Example

Figure 6-10 shows how the BITSUM instruction works. If the signal state of input I 0.0 is 1 (activated), the BITSUM function is executed. In this example, the value returned in MW0 is 13 (D in hexadecimal notation), which is the sum of bits set to 1 in the double word input hex value DW#16#12345678.

If the function executes without error, the signal states of ENO and Q4.0 are set to 1.

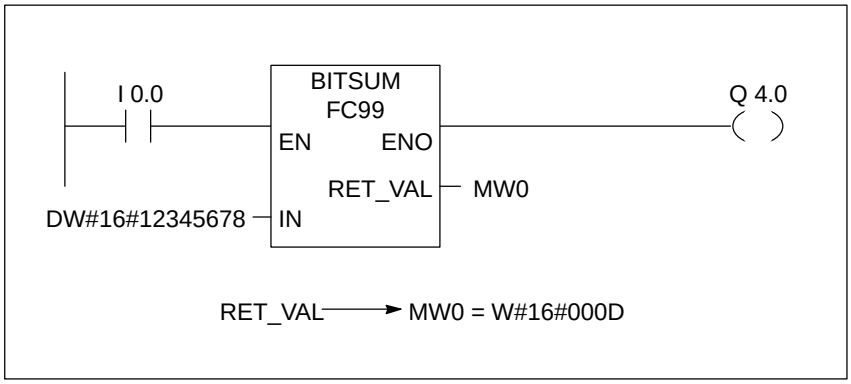


Figure 6-10 Sum Number of Bits (BITSUM)

6.8 Scaling Values (SCALE): FC105

Description

The Scaling Values (SCALE) function takes an integer value (IN) and converts it to a real value in engineering units scaled between a low and a high limit (LO_LIM and HI_LIM). The result is written to OUT. The SCALE function uses the equation:

OUT = [((FLOAT (IN) –K1)/(K2–K1)) * (HI_LIM – LO_LIM)] + LO_LIM

The constants K1 and K2 are set based upon whether the input value is BIPOLAR or UNIPOLAR.

- BIPOLAR: The input integer value is assumed to be between –27648 and 27648, therefore, K1 = –27648.0 and K2 = +27648.0
- UNIPOLAR: The input integer value is assumed to be between 0 and 27648, therefore, K1 = 0.0 and K2 = +27648.0

If the input integer value is greater than K2, the output (OUT) is clamped to HI_LIM, and an error is returned. If the input integer value is less than K1, the output (OUT) is clamped to LO_LIM, and an error is returned.

Reverse scaling can be obtained by programming LO_LIM > HI_LIM. With reverse scaling, the value of the output decreases as the value of the input increases.

Parameters

Table 6-8 describes the Scaling Values (SCALE) parameters.

Table 6-8 Scaling Values (FC105) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function executed without error
IN	Input	INT	I, Q, M, D, L, P, or constant	The input value to be scaled to a REAL value in engineering units
HI_LIM	Input	REAL	I, Q, M, D, L, P, or constant	Upper limit in engineering units
LO_LIM	Input	REAL	I, Q, M, D, L, P, or constant	Lower limit in engineering units
BIPOLAR	Input	BOOL	I, Q, M, D, L	A signal state of 1 indicates the input value is bipolar and a signal state of 0 indicates unipolar
OUT	Output	REAL	I, Q, M, D, L, P	The result of the scale conversion
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Error Information

If the input integer value is greater than K2, the output (OUT) is clamped to HI_LIM, and an error is returned. If the input integer value is less than K1, the output (OUT) is clamped to LO_LIM, and an error is returned. The signal state of ENO is set to 0 and RET_VAL is set equal to W#16#0008.

Example

Figure 6-11 shows how the SCALE instruction works. If the signal state of input I0.0 is 1 (activated), the SCALE function is executed. In this example, the integer value 22 will be converted to a REAL value scaled between 0.0 and 100.0 and written to OUT. The input value is BIPOLAR as indicated by the signal state of I2.0.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

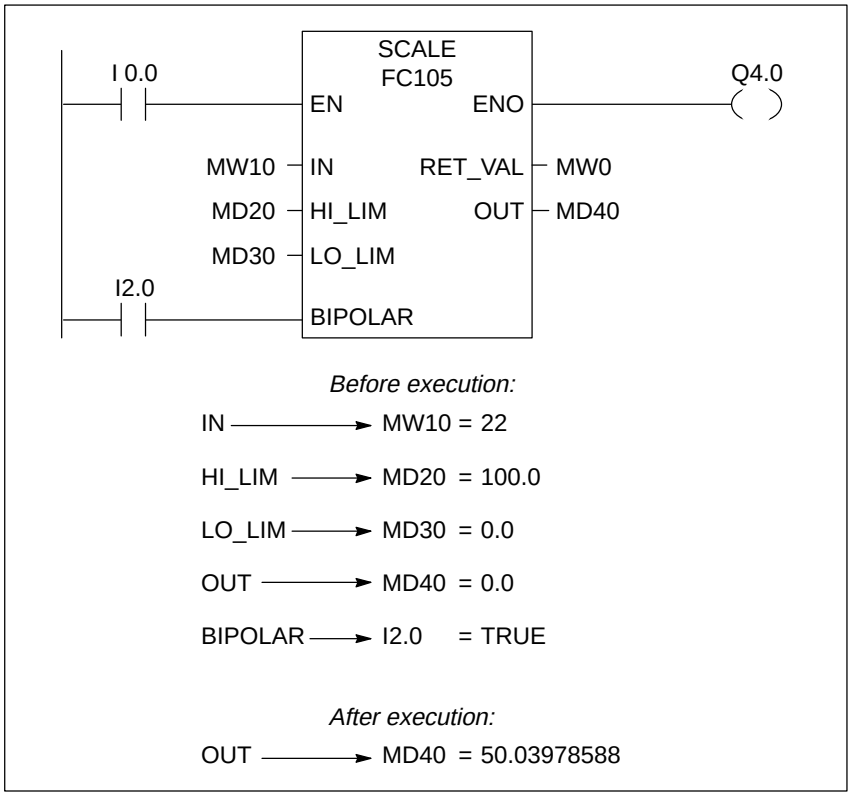


Figure 6-11 Scaling Values (SCALE)

6.9 Unscaling Values (UNSCALE): FC106

Description

The Unscaling Values (UNSCALE) function takes a real input value (IN) in engineering units scaled between a low and a high limit (LO_LIM and HI_LIM) and converts it to an integer value. The result is written to OUT. The UNSCALE function uses the equation:

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

and sets the constants K1 and K2 based upon whether the input value is BIPOLAR or UNIPOLAR.

- BIPOLAR: The output integer value is assumed to be between –27648 and 27648, therefore, K1 = –27648.0 and K2 = +27648.0
- UNIPOLAR: The output integer value is assumed to be between 0 and 27648, therefore K1 = 0.0 and K2 = +27648.0

If the input value is outside the LO_LIM and HI_LIM range, the output (OUT) is clamped to the nearer of either the low limit or the high limit of the specified range for its type (BIPOLAR or UNIPOLAR), and an error is returned.

Parameters

Table 6-9 describes the Unscaling Values (UNSCALE) parameters.

Table 6-9 Unscaling Values (FC106) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	REAL	I, Q, M, D, L, P, or constant	The input value to be unscaled to an integer value
HI_LIM	Input	REAL	I, Q, M, D, L, P, or constant	Upper limit in engineering units
LO_LIM	Input	REAL	I, Q, M, D, L, P, or constant	Lower limit in engineering units
BIPOLAR	Input	BOOL	I, Q, M, D, L	A signal state of 1 indicates the input value is bipolar and a signal state of 0 indicates unipolar
OUT	Output	INT	I, Q, M, D, L, P	The result of the unscale conversion
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Error Information

If the input real value is outside the LO_LIM and HI_LIM range the output (OUT) is clamped to the nearer of either the low limit or the high limit of the specified range for its type (BIPOLAR or UNIPOLAR), and an error is returned. The signal state of ENO is set to 0 and RET_VAL is set equal to W#16#0008.

Example

Figure 6-12 shows how the UNSCALE instruction works. If the signal state of input I0.0 is 1 (activated), the UNSCALE function is executed. In this example, the REAL value 50.03978588, scaled between 0.0 and 100.0 will be converted to an INTEGER value and written to OUT. The input value is BIPOLAR as indicated by the signal state of I2.0.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1 and RET_VAL is set equal to W#16#0000.

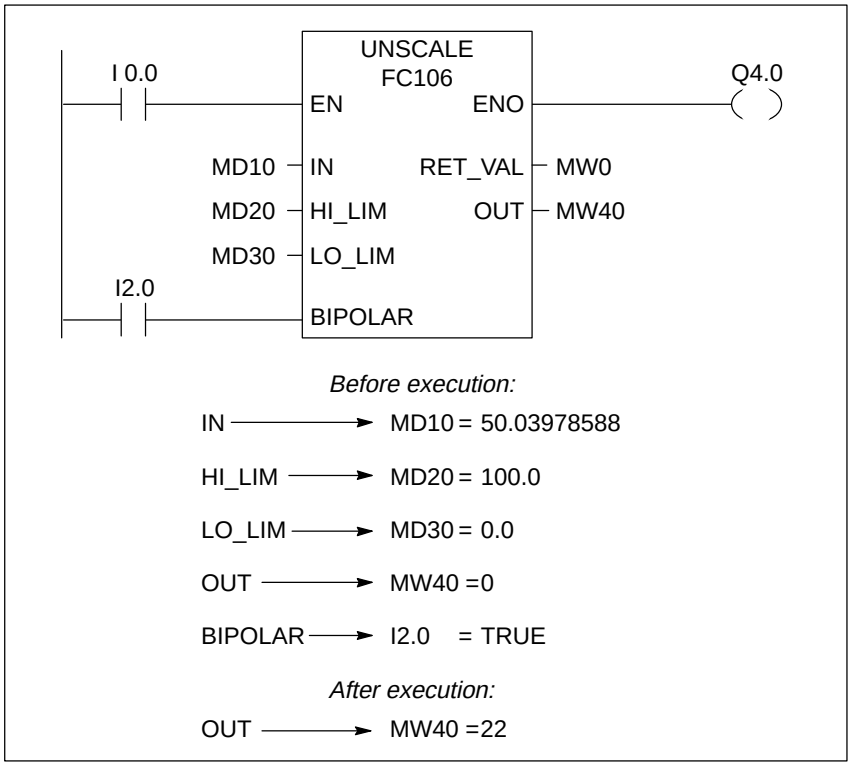


Figure 6-12 Unscaling Values (UNSCALE)

6.10 Lead/Lag Algorithm (LEAD_LAG): FB80

Description

The Lead/Lag Algorithm (LEAD_LAG) function block allows signal processing to be done on an analog variable. An output (OUT) is calculated based on an input (IN) and the specified gain (GAIN), lead (LD_TIME), and lag (LG_TIME) values. The gain value must be greater than zero. The LEAD_LAG algorithm uses the following equation:

where OUT =

$$\left[\frac{\text{LG\_TIME}}{\text{LG\_TIME} + \text{SAMPLE\_T}}\right] \text{PREV\_OUT} + \text{GAIN}\left[\frac{\text{LD\_TIME} + \text{SAMPLE\_T}}{\text{LG\_TIME} + \text{SAMPLE\_T}}\right] \text{IN} - \text{GAIN}\left[\frac{\text{LD\_TIME}}{\text{LG\_TIME} + \text{SAMPLE\_T}}\right] \text{PREV\_IN}$$

Typically, LEAD_LAG is used in conjunction with loops as a compensator in dynamic feed-forward control. LEAD_LAG consists of two parts. Phase lead shifts the phase of the function block's output so that it leads the input whereas phase lag shifts the output so that it lags the input. Because the lag operation is equivalent to an integration, it can be used as a noise suppressor or a low-pass filter. A lead operation is equivalent to a differentiation and is thus a high-pass filter. LEAD_LAG combined can cause the output phase to lag input at low frequency, and to lead input at high frequency, and can thus be used as a band-pass filter.

Parameters

Table 6-10 describes the Lead/Lag (LEAD_LAG) parameters.

Table 6-10 Lead/Lag (FB80) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function block is executed without error
IN	Input	REAL	I, Q, M, D, L, P, or constant	The input value of the current sample period to be processed
SAMPLE_T	Input	INT	I, Q, M, D, L, P, or constant	Sample time
OUT	Output	REAL	I, Q, M, D, L, P, or constant	The result of the LEAD_LAG operation
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
LD_TIME	Static	REAL	I, Q, M, D, L, P, or constant	Lead time in minutes
LG_TIME	Static	REAL	I, Q, M, D, L, P, or constant	Lag time in minutes
GAIN	Static	REAL	I, Q, M, D, L, P, or constant	Gain as % / % (the ratio of the change in output to the change in input as a steady state).
PREV_IN	Static	REAL	I, Q, M, D, L, P, or constant	Previous input
PREV_OUT	Static	REAL	I, Q, M, D, L, P, or constant	Previous output

Error Information

If GAIN is less than or equal to 0, the function block is not executed. The signal state of ENO is set to 0 and ERR_CODE is set equal to W#16#0009.

Example

Figure 6-13 shows how the LEAD_LAG instruction works. If the signal state of input I0.0 is 1 (activated), the LEAD_LAG function block is executed. In this example, the input value (IN) 2.0 is processed using the LEAD_LAG algorithm to produce the output (OUT).

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1 and ERR_CODE is set equal to W#16#0000.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

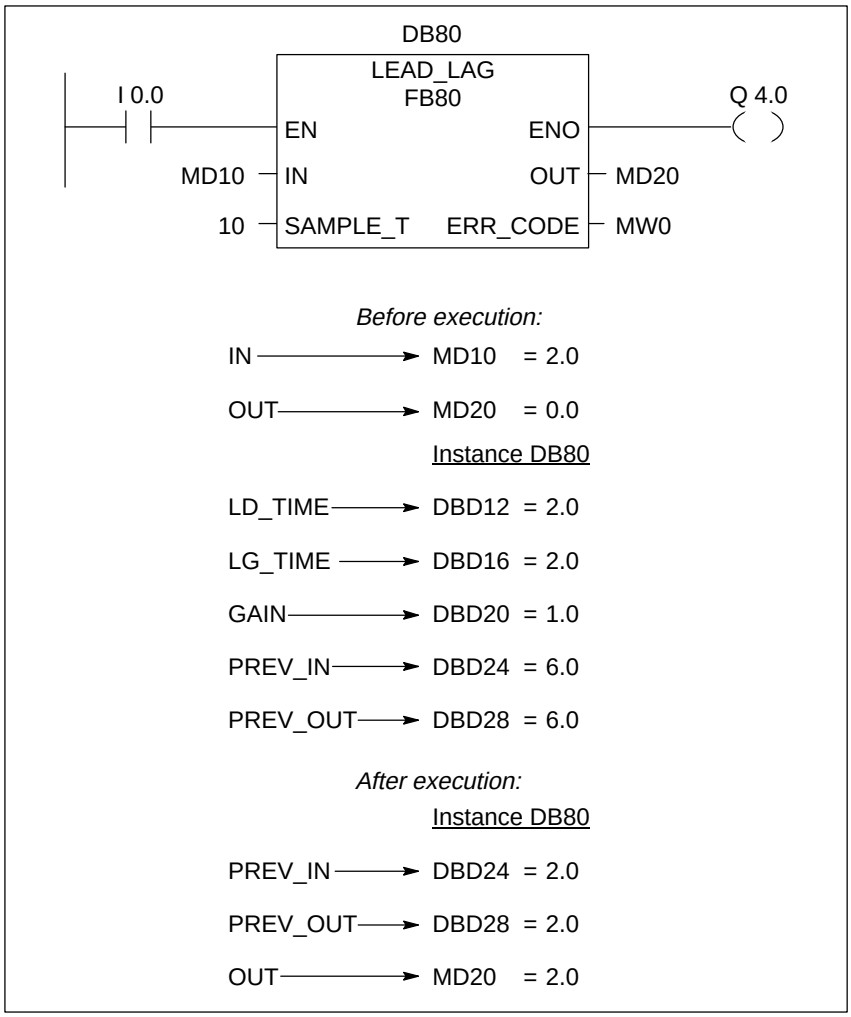


Figure 6-13 LEAD/LAG Algorithm (LEAD_LAG)

Floating Point Math Function

7

This section describes the floating point math function (FC) that you can add to your standard set of instructions to provide additional programming flexibility.

7.1 Standard Deviation (DEV): FC102

Description

The Standard Deviation (DEV) function computes the standard deviation of a set of values stored in a table (TBL) and stores the result in OUT. The standard deviation is computed according to the following formula:

$$\text{Standard Deviation} = \sqrt{\frac{(N * SqSum) - Sum^2}{N * (N - 1)}}$$

where: Sum = The sum of values in TBL
N = The number of values in TBL
SqSum = Sum of all the values in TBL squared

All calculations use IEEE floating point values with any necessary type conversions being done automatically by the function call.

- The first entry in the table contains the number of entries of the table (table length).
- The second entry in the table contains the first table value.
- The size of the table entries and the computed value (OUT) are determined by E_TYPE.

Parameters

Table 7-1 describes the Standard Deviation (DEV) parameters.

Table 7-1 Standard Deviation (FC102) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
TBL	Input	POINTER	I, Q, M, D	Points to the starting location of a table of values
OUT	Input	POINTER	I, Q, M, D	Points to location of the computed standard deviation value
E_TYPE	Input	BYTE	I, Q, M, D, L,P	Indicates the data type of the table entries. Valid data types for the DEV function are as follows: B#16#05 = INT B#16#07 = DINT B#16#08 = REAL
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Error Information

If any of the conditions shown in Table 7-2 occur, the function is not executed. The signal state of ENO is set to 0 and the return value is set appropriately.

Table 7-2 Error Conditions for FC102

RET_VAL	Explanation
W#16#0001	Invalid memory type specified for a function parameter
W#16#0002	Invalid E_TYPE
W#16#0004	The table length is zero

Example

Figure 7-1 shows how the Standard Deviation instruction works. If the signal state of input I0.0 is 1 (activated), the DEV function is executed. In this example, there are five table values as indicated by the first word in the table. The data type of the table values is REAL as indicated by E_TYPE.

If the function is executed without error, the signal states of ENO and Q4.0 are set to 1, and RET_VAL is set equal to W#16#0000.

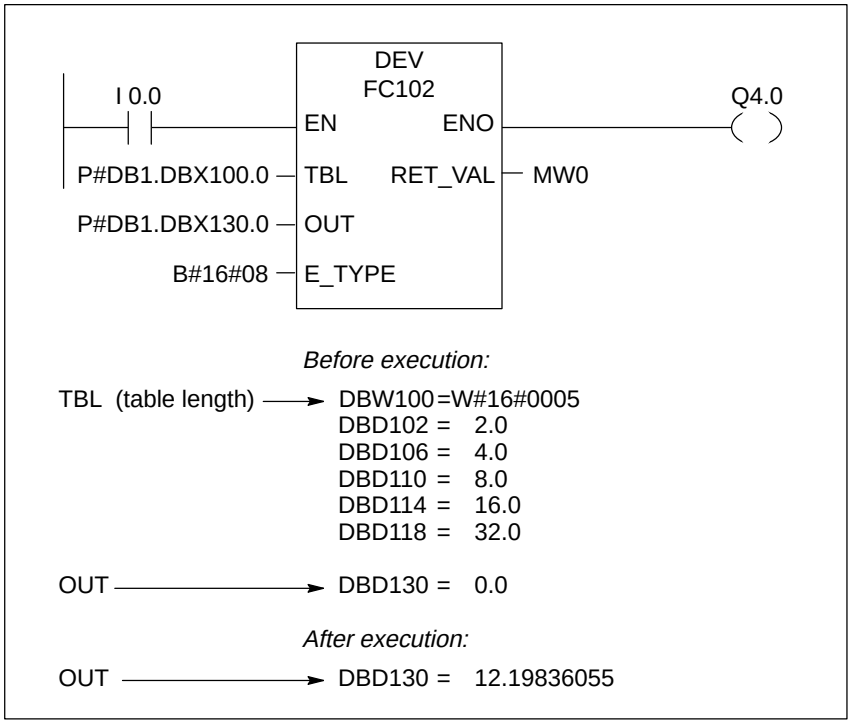


Figure 7-1 Standard Deviation (DEV)

Compare Function Blocks

8

This section describes compare function blocks (FBs) that you can add to your standard set of instructions to provide additional programming flexibility.

8.1 Index Matrix Compare (IMC): FB83

Description

The Index Matrix Compare (IMC) function block compares the signal state of up to 16 programmed input bits (IN_BIT0 to IN_BIT15) with the corresponding bits of a compare mask. Up to 16 steps with masks can be programmed. IN_BIT0 is compared to CMP_VAL [x, 0], where x is the step number, IN_BIT1 is compared to CMP_VAL [x,1], etc. The step number of the mask to compare against is indicated by the value of CMP_STEP. Unprogrammed input bits or unprogrammed bits of the masks have a default signal state of FALSE. If a match is found, the signal state of output (OUT) is set to 1, otherwise, it is set to 0.

Parameters Table 8-1 describes the Index Matrix Compare (IMC) parameters.

Table 8-1 Index Matrix Compare (FB83) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN_BIT0	Input	BOOL	I, Q, M, D, L	Input bit 0 to be compared to bit 0 of the mask
IN_BIT1	Input	BOOL	I, Q, M, D, L	Input bit 1 to be compared to bit 1 of the mask
IN_BIT2	Input	BOOL	I, Q, M, D, L	Input bit 2 to be compared to bit 2 of the mask
IN_BIT3	Input	BOOL	I, Q, M, D, L	Input bit 3 to be compared to bit 3 of the mask
IN_BIT4	Input	BOOL	I, Q, M, D, L	Input bit 4 to be compared to bit 4 of the mask
IN_BIT5	Input	BOOL	I, Q, M, D, L	Input bit 5 to be compared to bit 5 of the mask
IN_BIT6	Input	BOOL	I, Q, M, D, L	Input bit 6 to be compared to bit 6 of the mask
IN_BIT7	Input	BOOL	I, Q, M, D, L	Input bit 7 to be compared to bit 7 of the mask
IN_BIT8	Input	BOOL	I, Q, M, D, L	Input bit 8 to be compared to bit 8 of the mask
IN_BIT9	Input	BOOL	I, Q, M, D, L	Input bit 9 to be compared to bit 9 of the mask
IN_BIT10	Input	BOOL	I, Q, M, D, L	Input bit 10 to be compared to bit 10 of the mask
IN_BIT11	Input	BOOL	I, Q, M, D, L	Input bit 11 to be compared to bit 11 of the mask
IN_BIT12	Input	BOOL	I, Q, M, D, L	Input bit 12 to be compared to bit 12 of the mask
IN_BIT13	Input	BOOL	I, Q, M, D, L	Input bit 13 to be compared to bit 13 of the mask
IN_BIT14	Input	BOOL	I, Q, M, D, L	Input bit 14 to be compared to bit 14 of the mask
IN_BIT15	Input	BOOL	I, Q, M, D, L	Input bit 15 to be compared to bit 15 of the mask
CMP_STEP	Input	BYTE	I, Q, M, D, L, P	The step number of the mask to compare against
OUT	Output	BOOL	I, Q, M, D, L	A signal state of 1 indicates a match was found and a signal state of 0 indicates no match was found
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
CMP_VAL	Static	ARRAY OF BOOL	I, Q, M, D, L	The compare masks [0 to 15, 0 to 15], where the first number of the index is the step number and the second number is the bit number of the mask

Error Information

If the value of CMP_STEP is greater than 15, the function block is not executed. The signal state of ENO is set to 0 and ERR_CODE is set equal to W#16#000A.

Example

Figure 8-1 shows how the IMC instruction works. If the signal state of input I0.0 is 1 (activated), the IMC function block is executed. In this example, all 16 input bits are compared against the mask for step 2 (as indicated by CMP_STEP). The signal state of OUT is set TRUE since the input bits match the mask for step 2.

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1 and ERR_CODE is set equal to W#16#0000.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.

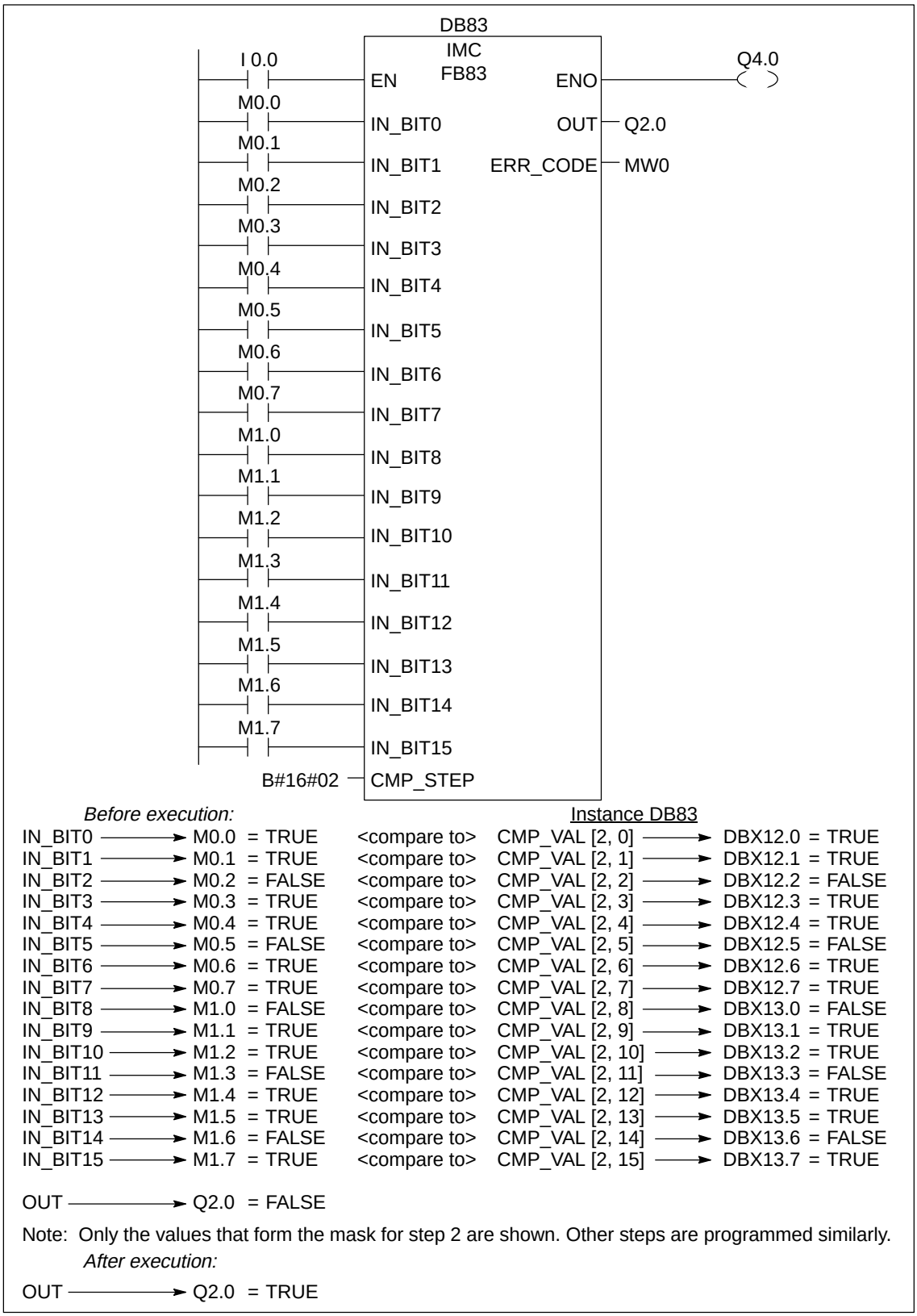


Figure 8-1 Index Matrix Compare (IMC)

8.2 Scan Matrix Compare (SMC): FB84

Description

The Scan Matrix Compare (SMC) function block compares the signal state of up to 16 programmed input bits (IN_BIT0 to IN_BIT15) with the corresponding bits of the compare mask of each step, beginning with step 1 and progressing through the last step programmed (LAST) or until a match is found. IN_BIT0 is compared to CMP_VAL [x,0], where x is the step number, IN_BIT1 is compared to CMP_VAL [x,1], etc. When a match is found, the signal state of output (OUT) is set to 1 and the step number with the matching mask is written to OUT_STEP. Unprogrammed input bits or unprogrammed bits of the masks have a default signal state of FALSE. If more than one step has a matching mask, only the first one encountered is indicated in OUT_STEP. If no match is found, the signal state of output (OUT) is set to 0 and OUT_STEP is one greater than LAST.

Parameters

Table 8-2 describes the Scan Matrix Compare (SMC) parameters.

Table 8-2 Scan Matrix Compare (FB84) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN_BIT0	Input	BOOL	I, Q, M, D, L	Input bit 0 to be compared to bit 0 of the mask
IN_BIT1	Input	BOOL	I, Q, M, D, L	Input bit 1 to be compared to bit 1 of the mask
IN_BIT2	Input	BOOL	I, Q, M, D, L	Input bit 2 to be compared to bit 2 of the mask
IN_BIT3	Input	BOOL	I, Q, M, D, L	Input bit 3 to be compared to bit 3 of the mask
IN_BIT4	Input	BOOL	I, Q, M, D, L	Input bit 4 to be compared to bit 4 of the mask
IN_BIT5	Input	BOOL	I, Q, M, D, L	Input bit 5 to be compared to bit 5 of the mask
IN_BIT6	Input	BOOL	I, Q, M, D, L	Input bit 6 to be compared to bit 6 of the mask
IN_BIT7	Input	BOOL	I, Q, M, D, L	Input bit 7 to be compared to bit 7 of the mask
IN_BIT8	Input	BOOL	I, Q, M, D, L	Input bit 8 to be compared to bit 8 of the mask
IN_BIT9	Input	BOOL	I, Q, M, D, L	Input bit 9 to be compared to bit 9 of the mask
IN_BIT10	Input	BOOL	I, Q, M, D, L	Input bit 10 to be compared to bit 10 of the mask
IN_BIT11	Input	BOOL	I, Q, M, D, L	Input bit 11 to be compared to bit 11 of the mask
IN_BIT12	Input	BOOL	I, Q, M, D, L	Input bit 12 to be compared to bit 12 of the mask
IN_BIT13	Input	BOOL	I, Q, M, D, L	Input bit 13 to be compared to bit 13 of the mask
IN_BIT14	Input	BOOL	I, Q, M, D, L	Input bit 14 to be compared to bit 14 of the mask
IN_BIT15	Input	BOOL	I, Q, M, D, L	Input bit 15 to be compared to bit 15 of the mask
OUT	Output	BOOL	I, Q, M, D, L	A signal state of 1 indicates a match was found and a signal state of 0 indicates no match was found
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Table 8-2 Scan Matrix Compare (FB84) Parameters

Parameter	Declaration	Data Type	Memory Area	Description
OUT_STEP	Output	BOOL	I, Q, M, D, L, P	Contains the step number with the matching mask, or contains the step number one greater than LAST if no match is found
LAST	Static	BYTE	I, Q, M, D, L, P	Specifies the step number of the last step to be scanned for a matching mask
CMP_VAL	Static	ARRAY OF BOOL	I, Q, M, D, L	The compare masks [0 to 15, 0 to 15], where the first number of the index is the step number and the second number is the bit number of the mask

Error Information

If the value of LAST is greater than 15, the function block is not executed. The signal state of EN0 is set to 0 and ERR_CODE is set equal to W#16#000E.

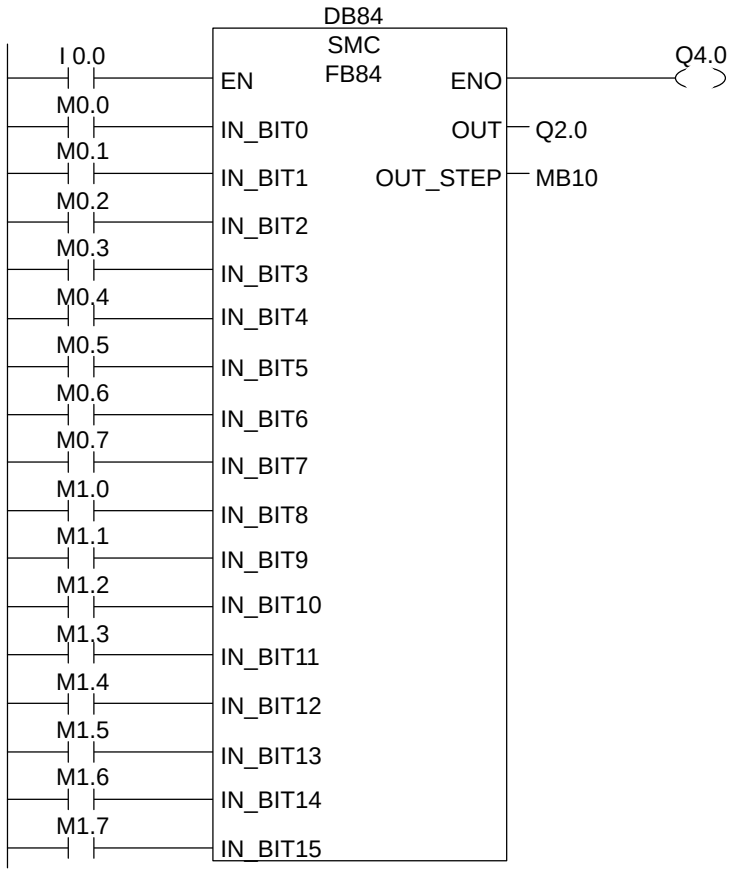
Example

Figure 8-2 shows how the SMC instruction works. If the signal state of input I0.0 is 1 (activated), the SMC function block is executed. In this example, all 16 input bits are compared against the masks for steps 0 through 5 (as indicated by LAST) or until a match is found. Only the masks for steps 0 through 2 are scanned since the mask for step 2 matches the input bits.

If the function block is executed without error, the signal states of ENO and Q4.0 are set to 1 and ERR_CODE is set equal to W#16#0000.

Note

Initialization of static parameters may be accomplished by using the Data Block Editor.



Before execution:

				Instance DB84	
IN_BIT0	M0.0 = TRUE	<compare to>	CMP_VAL [2, 0]	DBX12.0 = TRUE
IN_BIT1	M0.1 = TRUE	<compare to>	CMP_VAL [2, 1]	DBX12.1 = TRUE
IN_BIT2	M0.2 = FALSE	<compare to>	CMP_VAL [2, 2]	DBX12.2 = FALSE
IN_BIT3	M0.3 = TRUE	<compare to>	CMP_VAL [2, 3]	DBX12.3 = TRUE
IN_BIT4	M0.4 = TRUE	<compare to>	CMP_VAL [2, 4]	DBX12.4 = TRUE
IN_BIT5	M0.5 = FALSE	<compare to>	CMP_VAL [2, 5]	DBX12.5 = FALSE
IN_BIT6	M0.6 = TRUE	<compare to>	CMP_VAL [2, 6]	DBX12.6 = TRUE
IN_BIT7	M0.7 = TRUE	<compare to>	CMP_VAL [2, 7]	DBX12.7 = TRUE
IN_BIT8	M1.0 = FALSE	<compare to>	CMP_VAL [2, 8]	DBX13.0 = FALSE
IN_BIT9	M1.1 = TRUE	<compare to>	CMP_VAL [2, 9]	DBX13.1 = TRUE
IN_BIT10	M1.2 = TRUE	<compare to>	CMP_VAL [2, 10]	DBX13.2 = TRUE
IN_BIT11	M1.3 = FALSE	<compare to>	CMP_VAL [2, 11]	DBX13.3 = FALSE
IN_BIT12	M1.4 = TRUE	<compare to>	CMP_VAL [2, 12]	DBX13.4 = TRUE
IN_BIT13	M1.5 = TRUE	<compare to>	CMP_VAL [2, 13]	DBX13.5 = TRUE
IN_BIT14	M1.6 = FALSE	<compare to>	CMP_VAL [2, 14]	DBX13.6 = FALSE
IN_BIT15	M1.7 = TRUE	<compare to>	CMP_VAL [2, 15]	DBX13.7 = TRUE

OUT	Q2.0 = FALSE
OUT_STEP	MB10 = B#16#00
	Instance DB84
LAST	DB84 = B#16#05

Note: Only the values that form the mask for step 2 are shown. Other steps are programmed similarly.

After execution:

OUT	Q2.0 = TRUE
OUT_STEP	MB10 = B#16#02

Figure 8-2 Scan Matrix Compare (SMC)

.Glossary

A

Absolute Addressing

A type of addressing which indicates the actual location of a particular unit of data in storage in a CPU. Absolute addressing enables you to reference an I/O point, for example, using an address that includes the type of point (I = input, Q = output), the number of the I/O module, and the individual point. For example, output Q 4.0. The programmable logic controller interprets absolute addresses without the aid of a symbol table. *See* Symbolic Addressing.

Actual Parameter

An actual parameter is an address or value which is provided as an input or output during the call of a function block (FB) or function (FC). Actual parameters correspond to the formal parameters that are declared in the variable declaration table of the FB or FC.

Address

The address of a ladder logic instruction indicates a constant or the location where the instruction finds a value on which to perform an operation. The address can have a symbolic name, an absolute designation, or a combination of the two. The address can point to any of the following items:

- A constant, the value of a timer or counter, or an ASCII character string
- A location in the status word of the programmable logic controller
- A data block and a location within the data block area
- A function (FC), function block (FB), integrated system function (SFC), or integrated system function block (SFB) and the number of the function or block
- A label for a jump instruction
- An address identifier and a location within the memory area that is indicated by the address identifier (for example, I 1.0)
- The number of a timer or counter

The address of an instruction is sometimes referred to as the “operand” of the instruction.

Address Identifier

An address identifier is that part of the address of an instruction which provides such information as the memory area in which an instruction finds a value (data object) on which to perform an operation size of the value (data object) on which the instruction is to perform its operation. For example, in the address “IB10,” “IB” is the address identifier (where “I” indicates the input area of memory and “B” indicates a byte in that area).

B

Binary Result Bit

Bit 8 of the status word is called the binary result bit (BR bit). The BR bit forms a link between the processing of bits and words. This bit enables your program to interpret the result of a word operation as a binary result and to integrate this result in a binary logic chain.

For example, the BR bit makes it possible for you to write a function block (FB) or a function (FC) in statement list (STL, see the *STEP 7 Statement List Reference Manual*) and then call the FB or FC from ladder logic (LAD).

When writing a function block or function that you want to call from LAD, no matter whether you write the FB or FC in STL or LAD, you are responsible for managing the BR bit. The BR bit corresponds to the enable output (ENO) of a LAD box. You should use the SAVE instruction (in STL) or the —(SAVE) coil (in LAD) to store an RLO in the BR bit according to the following criteria:

- Store an RLO of 1 in the BR bit for a case where the FB or FC is executed without error.
- Store an RLO of 0 in the BR bit for a case where the FB or FC is executed with error.

You should program these instructions at the end of the FB or FC so that these are the last instructions that are executed in the block.

Warning

Possible unintentional resetting of the BR bit to 0.

When writing FBs and FCs in LAD, if you fail to manage the BR bit as described above, one FB or FC may overwrite the BR bit of another FB or FC.

To avoid this problem, store the RLO at the end of each FB or FC as described above.

C

CPU

The Central Processing Unit (CPU) module contains the user program and processes the data for the programmable logic controller.

D

Data Block (DB)

A data block (DB) stores data for the user program. You define the structure for the information that is stored in the DB. This information can be "shared" with (accessed by) all of the logic blocks in a program, or can be used as a specific instance of a particular FB (where the structure of the DB is associated with the variable declaration table of the FB).

Data Types

Data to be used in a program can be assigned a data type. When you define symbolic names in the Symbol Editor or when you define block-local variables in the variable declaration table, you must specify a data type. The data type defines the length and organization for the bits of memory that is being reserved by the CPU.

- Elementary data types: BOOL (boolean), BYTE, WORD, DWORD (double-word), CHAR (character), INT (integer), DINT (double-integer), REAL (floating point), TIME, DATE, TOD (time of day), and S5TIME. The operating system allocates a fixed length of memory for each of the elementary data types. For example, a boolean data type (BOOL) is one bit, a byte (BYTE) is 8 bits, a word (WORD) is 2 bytes (or 16 bits), and a double word (DWORD) is 4 bytes (or 32 bits).
- Complex data types: DT (data and time), STRING (up to 255 characters), STRUCT, UDT and ARRAY. Complex data types are typically larger than 32 bits (4 bytes). You can create combinations of data types either by defining a group of data types into a structure (STRUCT), or by defining a number of a specific data types into an array.
- Parameter types: TIMER (timer number), COUNTER (counter number), BLOCK_[DB, FB, FC, SDB, SFC, SFB] (number of the type of block identified), POINTER (pointer reference to an address), or ANY (allows an undefined, or "any," data type).

Direct Addressing

A type of addressing in which the address of an instruction points directly to the location of the value with which the instruction is to work. *Compare* "Immediate Addressing."

F

Formal Parameter

Formal parameters are declared in the variable declaration table of an FB or FC. When you call the FB or FC, you must supply an actual parameter (either an address or a value) to each formal parameter.

Function (FC)

A function (FC) is a logic block which contains a program segment but has no associated memory area. It functions like a sub-routine in a computer program. You create FCs and call them from your program. Since your program can call an FC many times (and pass different values for each call), an FC is defined as a reusable block. When the FC finishes processing, the local temporary data that was used by the FC is reallocated.

Function Block (FB)

A function block (FB) is a logic block which contains a program segment and has an associated memory area. Each time an FB is called, a data block (instance DB) must be provided. A single FB can be called several times, each time with a different instance DB. Parameters and static variables for the FB are stored in the instance DB.

I

Immediate Addressing

A type of addressing in which the actual value with which the instruction is to work is provided as an input parameter. This value is the address of the instruction. *Compare* “Direct Addressing.”

Instance Data Block (DB)

An instance DB provides memory for a specific call (or “instance”) of an FB. By creating multiple instances (instance DBs) of an FB, you can use one FB to control several devices.

The structure of an instance DB reflects the variable declaration table of an FB. The instance DB stores the actual parameters for the in, out, in_out, and var variables.

Instruction

A ladder logic instruction tells the CPU of your programmable controller what function the controller should perform. Ladder logic instructions can be in the form of elements or boxes.

L

Ladder Logic (LAD)

Ladder logic (LAD) is one of two languages of the STEP7 programming software that you can use to program your STEP 7-300 programmable logic controller (PLC). LAD is a graphic language whose components resemble elements of a hard-wired control relay panel.

Logic Block

Logic blocks are the blocks within STEP 7 that contain the program for the control logic. These are the organization blocks (OBs), functions and function blocks (FCs and FBs), and system functions and system function blocks (SFCs and SFBs). A data block (DB) is not considered a logic block.

M

Master Control Relay

The Master Control Relay (MCR) is an American relay ladder logic master switch for energizing and de-energizing power flow (current path). A de-energized current path corresponds to an instruction sequence that writes a zero value instead of the calculated value, or, to an instruction sequence that leaves the existing memory value unchanged.

Memory Area

A memory area represents the location in the CPU where an instruction finds a value (data object) on which to perform an operation. Your programmable logic controller has the following memory areas that you can designate as part of the address of an instruction:

- Process-image input
- Process-image output
- Bit memory
- I/O (peripheral I/O)
- Timer
- Counter
- Data block
- Temporary data (dynamic local data)

Mnemonic Representation

Mnemonic representation is an abbreviated form for displaying the names of addresses and programming instructions in the program (for example, “I” stands for “input” and “A” stands for the “And” instruction). STEP 7 supports the international representation (based on the English language), and the SIMATIC representation (based on the German representations of the instruction set and the SIMATIC addressing conventions).

N

Network

In STEP 7 ladder logic, a network is a rung of ladder logic instructions. *Compare* “Rung.”

O

Operand

See “Address”

P

Pointer

A pointer is a device that identifies the location of a variable. A pointer contains an address instead of a value. When assigning an actual parameter for the parameter type “pointer,” you provide the memory address. STEP 7 allows you to enter the pointer in either a pointer format or simply as an address (such as M 50.0). The following is an example of the pointer format for accessing data starting at M 50.0:

P#M50.0

R

Result of Logic Operation

Bit 1 of the status word is called the result of logic operation bit (RLO bit). This bit stores the result of a bit logic instruction or math comparison. The signal state of the RLO bit can provide information related to power flow. A signal state of 1 can indicate power flow (on); a signal state of 0 can indicate no power flow (off).

For example, the first instruction in a rung of ladder logic checks the signal state of a contact and produces a result of 1 or 0. The instruction stores the result of this signal state check in the RLO bit. The second instruction in a rung of bit logic instructions also checks the signal state of a contact and produces a result. Then the instruction combines this result with the value stored in the RLO bit of the status word according to the principles of Boolean logic (see First Check above). The result of this logic operation is stored in the RLO bit of the status word, replacing the former value in the RLO bit. Each subsequent instruction in the rung performs a logic operation on two values: the result produced when the instruction checks the contact, and the current RLO.

You can use a Boolean bit logic instruction on a first check to assign the state of the contents of a Boolean bit memory location to the RLO. You can also use the RLO to trigger jump instructions.

Rung

A rung is a row of ladder logic instructions, generally input contacts and box instructions, terminated by an output instruction at the end of the row. In STEP 7 ladder logic, a rung constitutes a network.

S

Statement List

Statement list (STL) is one of the languages of the STEP 7 programming software that you can use to communicate with your S7 300 programmable logic controller. Each statement in your program includes an instruction that uses a mnemonic abbreviation to represent a function of the programmable logic controller.

Symbolic Addressing

While every element in the CPU has an absolute address (such as “I 0.0”), you can also create a symbolic name which can be used for addressing. For example, you can associate input I 1.3 with “Pump_2_feedback”. The symbolic names are defined in a symbol table that you create with the Symbol Editor.

System Function (SFC)

A system function (SFC) is a pre-programmed, tested function that is integrated in the S7 operating system. You can call an SFC from your program. Because SFCs are a part of the operating system, they do not take up any space in the main memory. As with an FC, an SFC does not use an instance DB.

System Function Block (SFB)

A system function block (SFB) is a function block that is integrated in the S7 operating system. You can call an SFB from your program. Similar to the FB, the SFB has its own working memory in which data can be stored until the next time the SFB is called. This memory is implemented as an instance data block (instance DB). You must create this DB (which is opened as part of the Call instruction). Because SFBs are a part of the operating system, you do not have to load them.

U

User Program

The user program contains the control logic for an automation project. This control logic is stored in the form of instructions to the PLC which is controlling plant or a process.

V

Variable Declaration Table

All logic blocks have a variable declaration table. When you enter information in the variable declaration table, you declare (define) the parameters and variables that are used by the block.

Index

P

R

S

T

U

W

Siemens AG
A&D AS E 81

Oestliche Rheinbrueckenstr. 50
D-76181 Karlsruhe
Federal Republic of Germany

From:

Your Name: _

Your Title: _

Company Name: _

Street: _

City, Zip Code _

Country: _

Phone: _

Please check any industry that applies to you:

- ❒ Automotive
- ❒ Chemical
- ❒ Electrical Machinery
- ❒ Food
- ❒ Instrument and Control
- ❒ Nonelectrical Machinery
- ❒ Petrochemical
- ❒ Pharmaceutical
- ❒ Plastic
- ❒ Pulp and Paper
- ❒ Textiles
- ❒ Transportation
- ❒ Other _ _ _ _ _ _ _ _ _ _ _

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Please give each of the following questions your own personal mark within the range from 1 (very good) to 5 (poor).

1. Do the contents meet your requirements? ☐
2. Is the information you need easy to find? ☐
3. Is the text easy to understand? ☐
4. Does the level of technical detail meet your requirements? ☐
5. Please rate the quality of the graphics/tables: ☐

Additional comments:

_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _