

SIMATIC

STEP 7
Standard Functions
Part 2

Reference Manual

C79000-G7076-C113-01

C79000-H7076-C113-01

<hr/>	
Preface Contents	
<hr/>	
Bit Logic Functions	1
<hr/>	
Table Functions	2
<hr/>	
Shift Functions	3
<hr/>	
Move Functions	4
<hr/>	
Timer Functions	5
<hr/>	
Conversion Functions	6
<hr/>	
Glossary, Index	
<hr/>	
Remarks Form	
<hr/>	

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 danger:



Danger

indicates that death, severe personal injury or substantial property damage will result if proper precautions are not taken.



Warning

indicates that death, severe personal injury or substantial property damage can result if proper precautions are not taken.



Caution

indicates that minor personal injury or property damage can result if proper precautions are not taken.

Note

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

Qualified Personnel

The device/system may 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 circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage

Note the following:



Warning

This device and its components may only be used for the applications described in the catalog or the 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, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

SIMATIC® and SINEC® are registered trademarks of SIEMENS AG.

Copyright © Siemens AG 1995 All Rights Reserved

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Siemens AG
Automation Group
Industrial Automation Systems
Postfach 4848, D-90327 Nürnberg
Siemens Aktiengesellschaft

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.

Technical data subject to change.

© Siemens AG 1995

Order No. C79000-G7076-C113

Preface

Purpose	<p>This manual provides descriptions and examples of S7 functions (FCs) in ladder logic (LAD) representation. These FCs are available to program your S7-300 programmable logic controller (PLC). This manual is intended as a reference to provide the necessary information for each function.</p>
Where to Find the S7 Functions	<p>The S7 functions described in this manual are stored in the SIMATIC directory, in a subdirectory called FBLIB2. Using your STEP 7 file manager, you can copy the functions you need to your destination program directory. First make certain that your program does not contain any FCs with the same number as the ones you want to copy from FBLIB2. If you have FCs with matching numbers, you must renumber either your program FC(s) or the one(s) you want to copy to your program.</p>
Audience	<p>This manual is intended for engineers, programmers, and maintenance personnel who have a general knowledge of programmable logic controllers.</p>
How to Use This Manual	<p>This manual groups the FCs into the following functional areas:</p> <ul style="list-style-type: none">• Bit logic functions (Chapter 1)• Table functions (Chapter 2)• Shift functions (Chapter 3)• Move functions (Chapter 4)• Timer functions (Chapter 5)• Conversion functions (Chapter 6)• The glossary provides an alphabetical listing of definitions of key terms and expressions that are applicable to ladder logic programming. <p>This manual provides information on the function of specific instructions. To write a program, you may need to consult the following other manuals:</p> <ul style="list-style-type: none">• The <i>STEP 7 Program Design Programming Manual</i> describes how to design an entire program.• The <i>STEP 7 User Manual</i> provides information on the user interface.

**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.

Manual	Contents
<i>STEP 7 Program Design Programming Manual</i>	Provides basic information on designing STEP 7 programs: <ul style="list-style-type: none">• Guidelines and examples for the various methods of designing a program• Overview of the CPU functionality relating to the design of a program (such as memory management, addressing, data types, etc.)• Specific information concerning the design and structure of STEP 7 programs (such as the different types of blocks and how to call them, using local variables in the program, passing parameters between blocks, etc.)
<i>STEP 7 User Manual</i>	Information on working with the STEP 7 applications: <ul style="list-style-type: none">• Installing the STEP 7 software• Planning a programming session• Creating logic blocks and data blocks• Creating user programs as a text file• Using symbolic names for addresses• Configuring and assigning parameters to modules• Loading and testing user programs• Configuring the communication between CPUs• Guidelines for networking programmable controllers
<i>STEP 7 Statement List Reference Manual</i>	Reference manual for programming with the statement list (STL) representation form of the STEP 7 programming language: <ul style="list-style-type: none">• Basic information (for example, the structure of STL, number formats, syntax)• Description of all STEP 7 instructions (with programming examples)• Description of the various addressing capabilities of STEP 7 (with examples)• Description of all functions that are integrated in the CPUs• Description of the internal registers of the CPU

Manual	Contents
<i>STEP 7 Ladder Logic Reference Manual</i>	Reference manual for programming with the ladder logic (LAD) representation form of the STEP 7 programming language: <ul style="list-style-type: none"> • Basic information (for example, the structure of LAD, number formats, syntax) • Description of all STEP 7 instructions (with programming examples) • Description of the various addressing capabilities of STEP 7 (with examples) • Description of all functions that are integrated in the CPUs • Description of the internal registers of the CPU
<i>STEP 7 Standard and System Functions Reference Manual</i>	Provides detailed descriptions of standard loadable functions and of complex system functions that are integrated in the firmware of the CPUs
<i>STEP 7 Converting STEP 5 Programs Manual</i>	Provides information on converting STEP 5 programs to STEP 7: <ul style="list-style-type: none"> • Working with the S5/S7 converter • Rules for conversion • Using converted STEP 5 standard function blocks in STEP 7
<i>STEP 7 Master Index of Manuals</i>	Provides an index to the entire STEP 7 documentation set.

You can find additional information in the on-line help and in the STEP 7 tutorial.

Additional Manuals

The manuals listed in the following table describe the S7-300 hardware.

Manual	Contents
<i>STEP 7 Hardware and Installation Manual</i>	Describes the hardware of the S7-300: <ul style="list-style-type: none"> • Configuring the S7-300 hardware • Installing the S7-300 • Wiring and preparing the S7-300 for system operation • Features and technical specifications of the S7-300 modules
<i>S7-300 Instruction List, CPU 312, CPU 314</i>	Lists the statement list instructions and includes the typical execution time in microseconds for each instruction for CPU 312 and CPU 314
<i>PG 7xx Programming Device</i>	Describes the hardware of the programming device: <ul style="list-style-type: none"> • Set up and operation of the programming device • Expansion capabilities • Configuration • Diagnostics

Scope of Manual

This manual applies to the STEP 7 programming software package for the S7-300 series of programmable controllers. The S7 functions described in this manual are the following:

Software Timer On Delay—Retentive (TONR): . .	FC80
Indirect Block Move (IBLKMOV):	FC81
Reset Range of Outputs (RSET):	FC82
Reset Range of Immediate Outputs (RSETI):	FC100
Set Range of Outputs (SET):	FC83
Set Range of Immediate Outputs (SETI):	FC101
Add to Table (ATT):	FC84
First In/First Out Unload Table (FIFO):	FC85
Table Find (TBL_FIND):	FC86
Last In/First Out Unload Table (LIFO):	FC87
Table (TBL):	FC88
Move Table to Word (TBL_WRD):	FC89
Word Shift Register (WSR):	FC90
Word to Table (WRD_TBL):	FC91
Bit Shift Register (SHRB):	FC92
Seven Segment Decoder (SEG):	FC93
ASCII to Hex (ATH):	FC94
Hex to ASCII (HTA):	FC95
Encode Binary Position (ENCO):	FC96
Decode Binary Position (DECO):	FC97
Ten's Complement (BCDCPL):	FC98
Sum Number of Bits (BITSUM):	FC99

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.

Contents

1 Bit Logic Functions

1.1	Overview	1-3
	What Is Described in This Chapter?	1-3
	Where Do You Look For More Information?	1-3
1.2	Reset Range of Outputs (RSET): FC82	1-4
	Description	1-4
	Parameters	1-4
	Error Information	1-5
	Example	1-5
1.3	Reset Range of Immediate Outputs (RSETI): FC100	1-6
	Description	1-6
	Parameters	1-6
	Error Information	1-7
	Example	1-7
1.4	Set Range of Outputs (SET): FC83	1-8
	Description	1-8
	Parameters	1-8
	Error Information	1-9
	Example	1-9
1.5	Set Range of Immediate Outputs (SETI): FC101	1-10
	Description	1-10
	Parameters	1-10
	Error Information	1-11
	Example	1-11

2 Table Functions

2.1	Overview	2-3
	What Is Described in This Chapter?	2-3
	Where Do You Look For More Information?	2-3
2.2	Add to Table (ATT): FC84	2-4
	Description	2-4
	Parameters	2-4
	Error Information	2-5
	Example	2-5
2.3	First In/First Out Unload Table (FIFO): FC85	2-6
	Description	2-6
	Parameters	2-6
	Error Information	2-7
	Example	2-7

2.4	Table Find (TBL_FIND): FC86	2-8
	Description	2-8
	Parameters	2-8
	Error Information	2-9
	Example	2-9
2.5	Last In/First Out Unload Table (LIFO): FC87	2-10
	Description	2-10
	Parameters	2-10
	Error Information	2-11
	Example	2-11
2.6	Table (TBL): FC88	2-12
	Description	2-12
	Parameters	2-12
	Error Information	2-13
	Example	2-13
2.7	Move Table to Word (TBL_WRD): FC89	2-14
	Description	2-14
	Parameters	2-14
	Error Information	2-15
	Example	2-15
2.8	Word to Table (WRD_TBL): FC91	2-16
	Description	2-16
	Parameters	2-16
	Error Information	2-17
	Example	2-17
3	Shift Functions	
3.1	Overview	3-3
	What Is Described in This Chapter?	3-3
	Where Do You Look For More Information?	3-3
3.2	Word Shift Register (WSR): FC90	3-4
	Description	3-4
	Parameters	3-4
	Error Information	3-5
	Example	3-5
3.3	Bit Shift Register (SHRB): FC92	3-6
	Description	3-6
	Parameters	3-6
	Error Information	3-7
	Example	3-7

4	Move Functions	
4.1	Overview	4-3
	What Is Described in This Chapter?	4-3
	Where Do You Look For More Information?	4-3
4.2	Indirect Block Move (IBLKMOV): FC81	4-4
	Description	4-4
	Parameters	4-4
	Error Information	4-5
	Example	4-5
5	Timer Functions	
5.1	Overview	5-3
	What Is Described in This Chapter?	5-3
	Where Do You Look For More Information?	5-3
5.2	Software Timer On Delay—Retentive (TONR): FC80	5-4
	Description	5-4
	Parameters	5-4
	Error Information	5-5
	Example	5-5
6	Conversion Functions	
6.1	Overview	6-3
	What Is Described in This Chapter?	6-3
	Where Do You Look For More Information?	6-3
6.2	Seven Segment Decoder (SEG): FC93	6-4
	Description	6-4
	Parameters	6-4
	Error Information	6-5
	Example	6-5
6.3	ASCII to Hex (ATH): FC94	6-6
	Description	6-6
	Parameters	6-6
	Error Information	6-6
	Example	6-7
6.4	Hex to ASCII (HTA): FC95	6-8
	Description	6-8
	Parameters	6-8
	Error Information	6-8
	Example	6-9
6.5	Encode Binary Position (ENCO): FC96	6-10
	Description	6-10
	Parameters	6-10
	Error Information	6-11
	Example	6-11

6.6	Decode Binary Position (DECO): FC97	6-12
	Description	6-12
	Parameters	6-12
	Error Information	6-13
	Example	6-13
6.7	Tens Complement (BCDCPL): FC98	6-14
	Description	6-14
	Parameters	6-14
	Error Information	6-15
	Example	6-15
6.8	Sum Number of Bits (BITSUM): FC99	6-16
	Description	6-16
	Parameters	6-16
	Error Information	6-17
	Example	6-17

Figures

1-1	Reset Range of Outputs (RSET)	1-5
1-2	Reset Range of Immediate Outputs (RSETI)	1-7
1-3	Set Range of Outputs (SET)	1-9
1-4	Set Range of Immediate Outputs (SETI)	1-11
2-1	Add to Table (ATT)	2-5
2-2	First In/First Out Unload Table (FIFO)	2-7
2-3	Table Find (TBL_FIND)	2-9
2-4	Last In/First Out Unload Table (LIFO)	2-11
2-5	Table (TBL)	2-13
2-6	Move Table to Word (TBL_WRD)	2-15
2-7	Word to Table (WRD_TBL)	2-17
3-1	Word Shift Register (WSR)	3-5
3-2	Bit Shift Register (SHRB)	3-7
4-1	Indirect Block Move (IBLKMOV)	4-5
5-1	Timer On Delay—Retentive (TONR)	5-5
6-1	Seven Segment Output Bit Patterns	6-4
6-2	Seven Segment Decoder (SEG)	6-5
6-3	ASCII to Hex (ATH)	6-7
6-4	ASCII Characters and Equivalent Hexadecimal Values	6-7
6-5	Hex to ASCII (HTA)	6-9
6-6	Hexadecimal Digits and Equivalent ASCII Hex Values	6-9
6-7	Encode Binary Position (ENCO)	6-11
6-8	Decode Binary Position (DECO)	6-13
6-9	Tens Complement (BCDCPL)	6-15
6-10	Sum Number of Bits (BITSUM)	6-17

Tables

1-1	Reset Range of Outputs (FC82) and Parameters	1-4
1-2	Reset Range of Immediate Outputs (FC100) and Parameters ..	1-6
1-3	Set Range of Outputs (FC83) and Parameters	1-8
1-4	Set Range of Immediate Outputs (FC101) and Parameters	1-10
2-1	Add to Table (FC84) and Parameters	2-4
2-2	First In/First Out Unload Table (FC85) and Parameters	2-6
2-3	Table Find (FC86) and Parameters	2-8
2-4	Last In/First Out Unload Table (FC87) and Parameters	2-10
2-5	Table (FC88) and Parameters	2-12
2-6	Move Table to Word (FC89) and Parameters	2-14
2-7	Word to Table (FC91) and Parameters	2-16
3-1	Word Shift Register (FC90) and Parameters	3-4
3-2	Bit Shift Register (FC92) and Parameters	3-6
4-1	Indirect Block Move (FC81) and Parameters	4-4
5-1	Software Timer On Delay—Retentive (FC80) and Parameters ..	5-4
6-1	Seven Segment Decoder (FC93) and Parameters	6-4
6-2	ASCII to Hex (FC94) and Parameters	6-6
6-3	Hex to ASCII (FC95) and Parameters	6-8
6-4	Encode Binary Position (FC96) and Parameters	6-10
6-5	Decode Binary Position (FC97) and Parameters	6-12
6-6	Tens Complement (FC98) and Parameters	6-14
6-7	Sum Number of Bits (FC99) and Parameters	6-16

Bit Logic Functions

1.1	Overview	1-3
	What Is Described in This Chapter?	1-3
	Where Do You Look For More Information?	1-3
1.2	Reset Range of Outputs (RSET): FC82	1-4
	Description	1-4
	Parameters	1-4
	Error Information	1-5
	Example	1-5
1.3	Reset Range of Immediate Outputs (RSETI): FC100	1-6
	Description	1-6
	Parameters	1-6
	Error Information	1-7
	Example	1-7
1.4	Set Range of Outputs (SET): FC83	1-8
	Description	1-8
	Parameters	1-8
	Error Information	1-9
	Example	1-9
1.5	Set Range of Immediate Outputs (SETI): FC101	1-10
	Description	1-10
	Parameters	1-10
	Error Information	1-11
	Example	1-11

Figures

1-1	Reset Range of Outputs (RSET)	1-5
1-2	Reset Range of Immediate Outputs (RSETI)	1-7
1-3	Set Range of Outputs (SET)	1-9
1-4	Set Range of Immediate Outputs (SETI)	1-11

Tables

1-1	Reset Range of Outputs (FC82) and Parameters	1-4
1-2	Reset Range of Immediate Outputs (FC100) and Parameters ..	1-6
1-3	Set Range of Outputs (FC83) and Parameters	1-8
1-4	Set Range of Immediate Outputs (FC101) and Parameters	1-10

1.1 Overview

What Is Described in This Chapter?

This section describes bit logic functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

1.2 Reset Range of Outputs (RSET): FC82

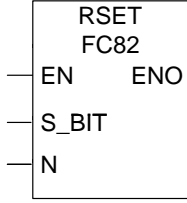
Description

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 shows the RSET box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	S_BIT	Pointer*	I, Q, M, D	Points to the first bit in the range
	N	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 (P) memory area, the signal state of each bit in the range remains unchanged and the ENO bit 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 the 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 errors, the signal states of ENO and Q 4.0 are set to 1.

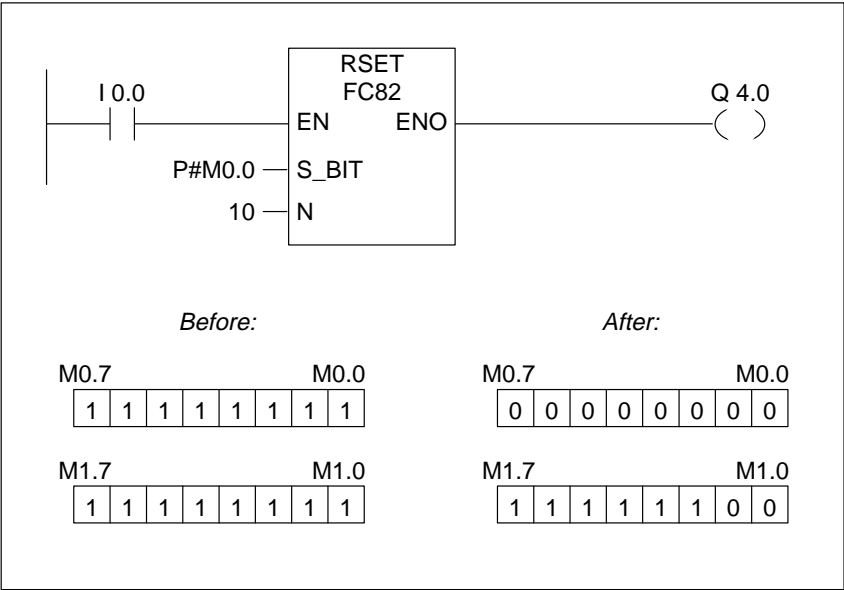


Figure 1-1 Reset Range of Outputs (RSET)

1.3 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

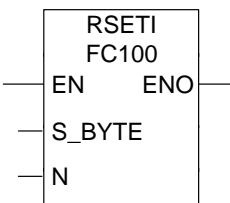
The value of N must be a multiple of eight (for example, 8, 16, 24, etc.).

The S_BYTE pointer must reference the external input and output (P) memory area. 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.

Parameters

Table 1-2 shows the RSETI box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	S_BYTE	Pointer*	P	Points to the first byte in the range
	N	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

The enable output (ENO) is not activated (that is, its signal state is 0) and the signal state of each byte in the range remains unchanged if any one of the following conditions occurs.

- 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.

If none of the above conditions occurs, the ENO is set to 1.

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 the 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 byte in the range P2.0 through P3.7 is reset to 0.

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

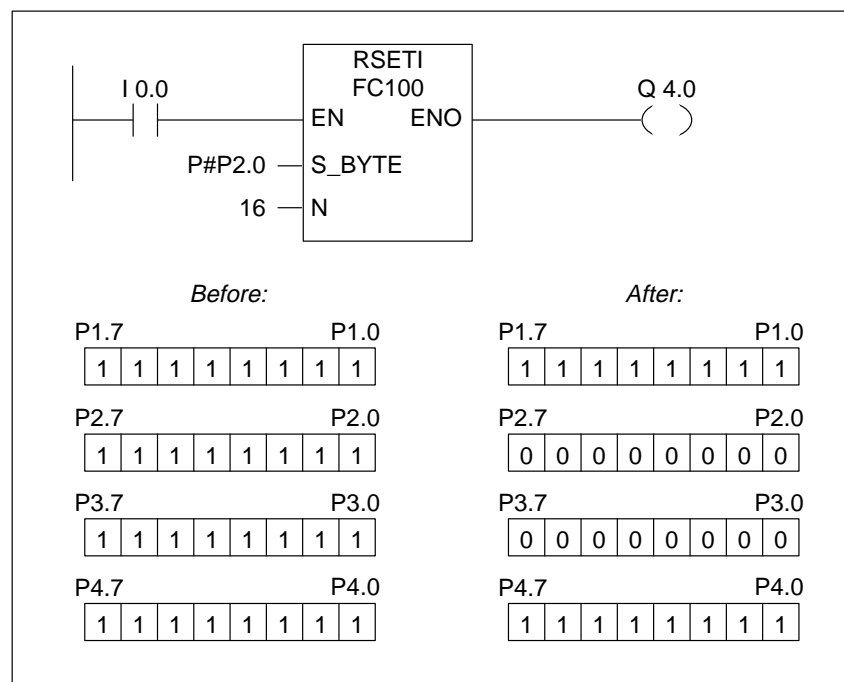
1

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

1.4 Set Range of Outputs (SET): FC83

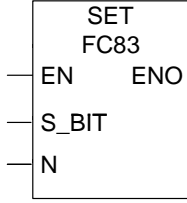
Description

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 shows the SET box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	S_BIT	Pointer*	I, Q, M, D	Points to the first bit in the range
	N	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 (P) memory area, the signal state of each bit in the range remains unchanged and the ENO bit 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 the 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 errors, the signal states of ENO and Q 4.0 are set to 1.

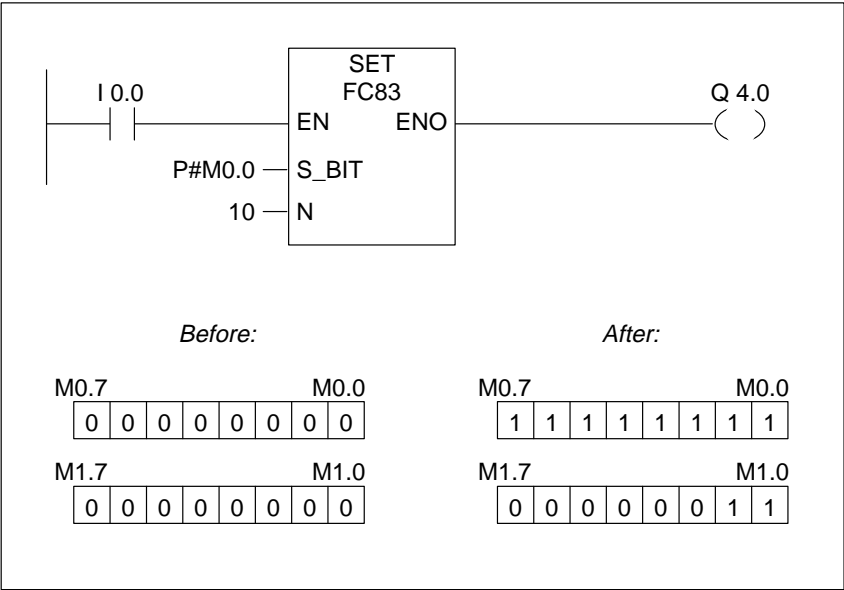


Figure 1-3 Set Range of Outputs (SET)

1.5 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 (P) memory area. 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.

Parameters

Table 1-4 shows the SETI box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	S_BYTE	Pointer*	P	Points to the first byte in the range
	N	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

The enable output (ENO) is not activated (that is, its signal state is 0) and the signal state of each byte in the range remains unchanged if any one of the following conditions occurs.

- 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.

If none of the above conditions occurs, the ENO is set to 1.

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 the 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 byte in the range P2.0 through P3.7 is set to 1.

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

1

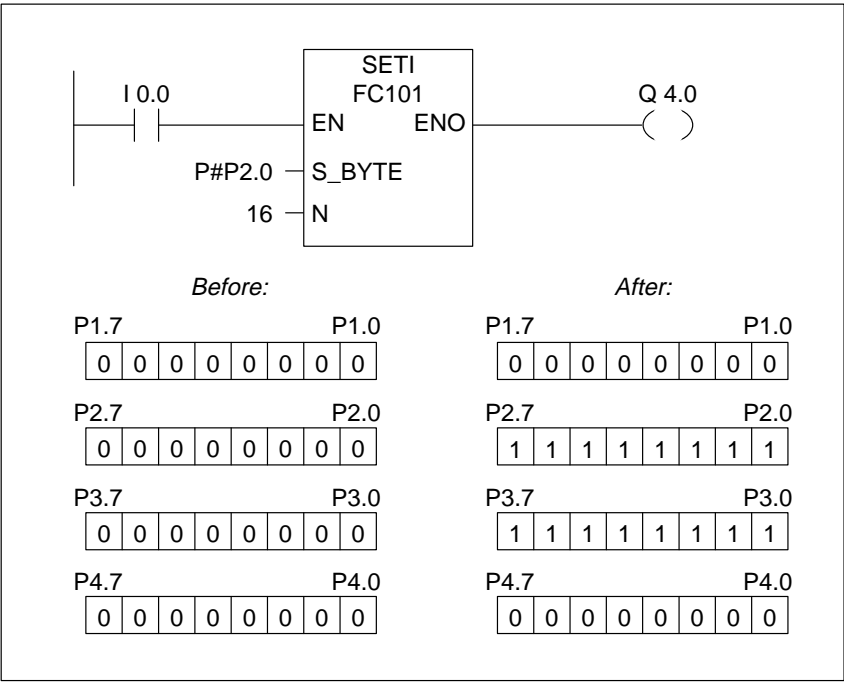


Figure 1-4 Set Range of Immediate Outputs (SETI)

2

Table Functions

2.1	Overview	2-3
	What Is Described in This Chapter?	2-3
	Where Do You Look For More Information?	2-3
2.2	Add to Table (ATT): FC84	2-4
	Description	2-4
	Parameters	2-4
	Error Information	2-5
	Example	2-5
2.3	First In/First Out Unload Table (FIFO): FC85	2-6
	Description	2-6
	Parameters	2-6
	Error Information	2-7
	Example	2-7
2.4	Table Find (TBL_FIND): FC86	2-8
	Description	2-8
	Parameters	2-8
	Error Information	2-9
	Example	2-9
2.5	Last In/First Out Unload Table (LIFO): FC87	2-10
	Description	2-10
	Parameters	2-10
	Error Information	2-11
	Example	2-11
2.6	Table (TBL): FC88	2-12
	Description	2-12
	Parameters	2-12
	Error Information	2-13
	Example	2-13
2.7	Move Table to Word (TBL_WRD): FC89	2-14
	Description	2-14
	Parameters	2-14
	Error Information	2-15
	Example	2-15
2.8	Word to Table (WRD_TBL): FC91	2-16
	Description	2-16
	Parameters	2-16
	Error Information	2-17
	Example	2-17

Figures

2-1	Add to Table (ATT)	2-5
2-2	First In/First Out Unload Table (FIFO)	2-7
2-3	Table Find (TBL_FIND)	2-9
2-4	Last In/First Out Unload Table (LIFO)	2-11
2-5	Table (TBL)	2-13
2-6	Move Table to Word (TBL_WRD)	2-15
2-7	Word to Table (WRD_TBL)	2-17

Tables

2-1	Add to Table (FC84) and Parameters	2-4
2-2	First In/First Out Unload Table (FC85) and Parameters	2-6
2-3	Table Find (FC86) and Parameters	2-8
2-4	Last In/First Out Unload Table (FC87) and Parameters	2-10
2-5	Table (FC88) and Parameters	2-12
2-6	Move Table to Word (FC89) and Parameters	2-14
2-7	Word to Table (FC91) and Parameters	2-16

2.1 Overview

What Is Described in This Chapter?

This section describes table functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

2.2 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 make entries into tables for use by the FIFO and LIFO functions.

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

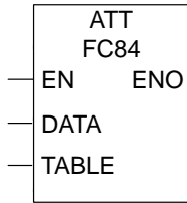
Note

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

Parameters

Table 2-1 shows the ATT box and describes the parameters.

Table 2-1 Add to Table (FC84) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	DATA	WORD	I, Q, M, D, L, P or constant	Data to add to table
	TABLE	Pointer*	I, Q, M, D	Points to the starting location 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 maximum number, the data will not be added to the table and the ENO bit 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 the example, DATA is added as the fifth entry in the table and the number of entries is incremented from 4 to 5.

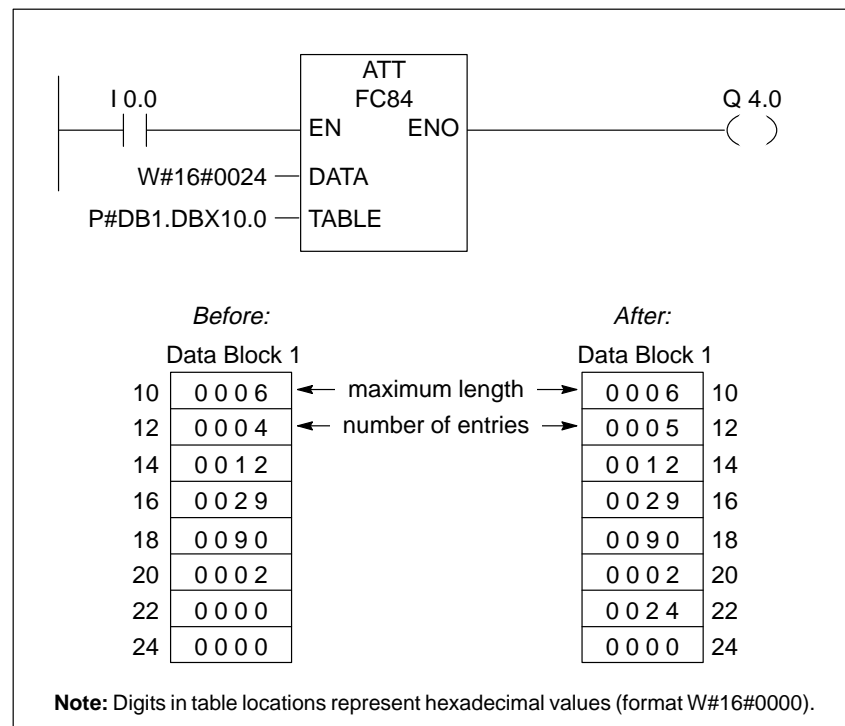


Figure 2-1 Add to Table (ATT)

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

Description

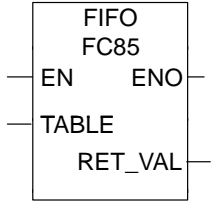
The First In/First Out Unload Table (FIFO) function returns the oldest entry from the FIFO table as the function value. The number of entries is decremented by one, and if more entries remain, they are shifted down in the table. The FIFO table consists of words. You use the ATT function to add values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

- The first location in the table contains the maximum length of the table.
- The second location contains the number of entries.
- The third location of the table contains the first word of data.

Parameters

Table 2-2 shows the FIFO box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	TABLE	Pointer*	I, Q, M, D	Points to the starting location of the FIFO table
	RET_VAL	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 remains unchanged and the ENO bit 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 the example, the oldest entry in the table is returned as the function value (MW2.0). 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 successfully, the signal states of ENO and Q 4.0 are set to 1.

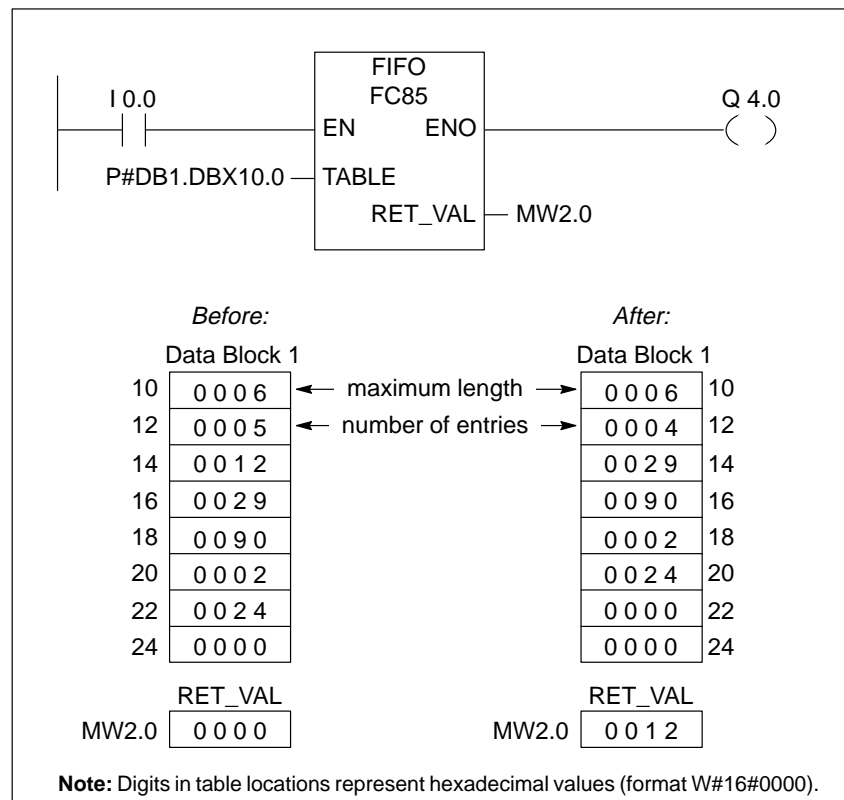


Figure 2-2 First In/First Out Unload Table (FIFO)

2.4 Table Find (TBL_FIND): FC86

Description

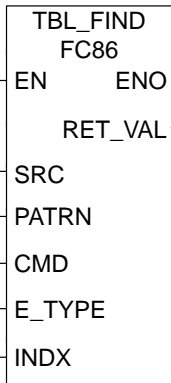
Use the Table Find (TBL_FIND) function to search for a distinct pattern or to search for a non-consistent pattern in a block of memory. The function call performs the indicated compare (CMD) between the source pattern (PATRN) and the source table (SRC) location. It locates the next element (after the element indexed by INDX) in the table that satisfies the compare condition and places its element 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 a 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 length of the table (number of elements) is stored in the first word of the table.
- The first element of the table starts at the second word.

Parameters

Table 2-3 shows the TBL_FIND box and describes the parameters.

Table 2-3 Table Find (FC86) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	SRC	Pointer*	I, Q, M, D	Points to the starting location of the table
	PATRN	Pointer*	I, Q, M, D	Points to the pattern to be searched for
	CMD	BYTE	I, Q, M, D, L, P	Indicates to search for a value that is either equal or not equal to the pattern: 1 = search for distinct pattern 2 = search for first pattern mismatch
	E_TYPE	BYTE	I, Q, M, D, L, P	Indicates the type of table elements 2 = BYTE 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = REAL
	INDX	WORD	I, Q, M, D, L	Index into table that provides: Input: starting element of the search Output: element number of the matching value
	RET_VAL	WORD	I, Q, M, D, L, P	Returns a value of 0 if the instruction is executed successfully; see Error Information for values other than 0.

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

Error Information

The enable output (ENO) is not activated (its signal state is 0), and the table locations remain unchanged if any one of the following conditions occurs:

- No match is found.
- An invalid E_TYPE is used.
- An invalid CMD code is used.

In addition, the RET_VAL is set to one of the following values:

- RET_VAL = 8 indicates that no match was found.
- RET_VAL = 9 indicates that an invalid E_TYPE and/or invalid CMD parameter was used.

Example

Figure 2-3 shows how the TBL_FIND instruction works. In this example, since E_TYPE is 4, data in the table is stored in words starting at the location 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 matches the pattern value. The INDX value points to the location where the search is to begin. After the instruction is executed, the INDX value provides the location in the table where the comparison command was satisfied.

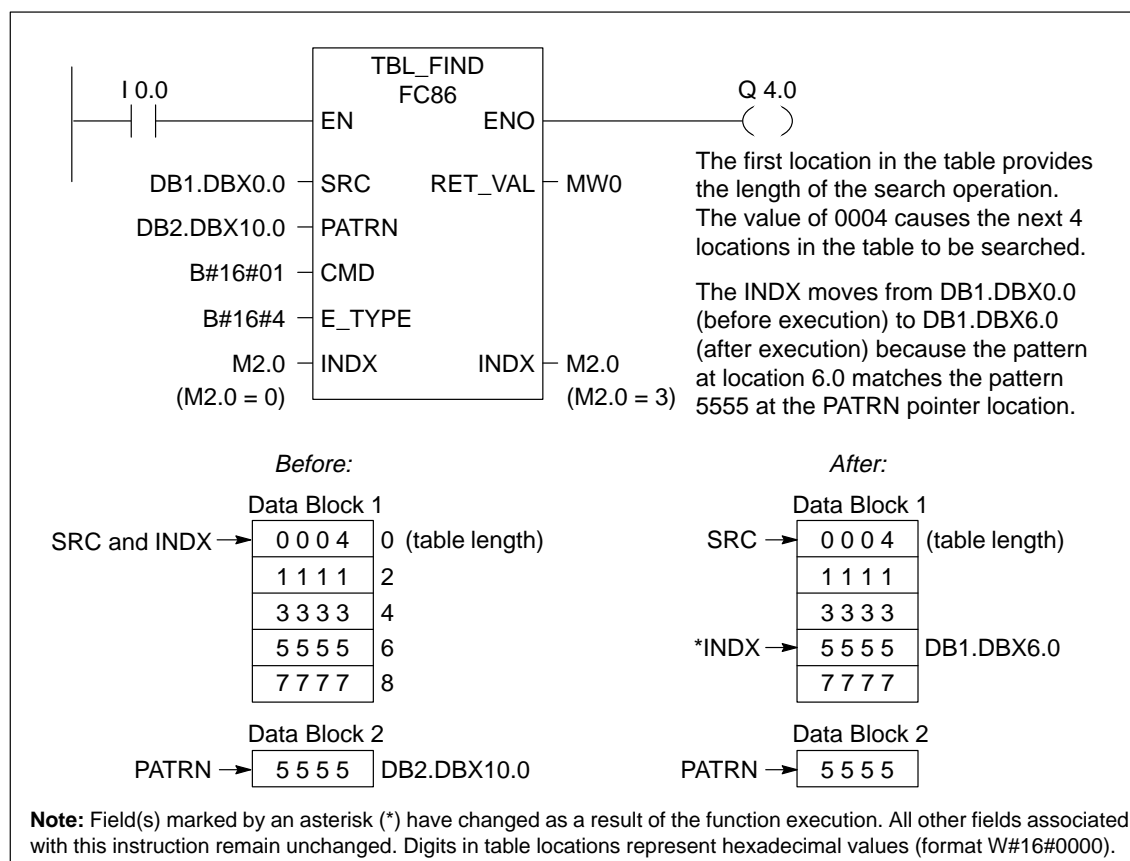


Figure 2-3 Table Find (TBL_FIND)

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

Description

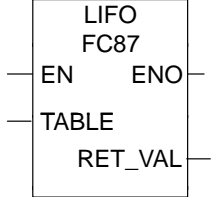
The Last In/First Out Unload Table (LIFO) function returns the newest (most recent) entry from the LIFO table as the function value and decrements the number of entries by one. The LIFO table consists of words. You use the ATT function to enter values into the LIFO table.

- The first location in the table contains the maximum length of the table.
- The second location contains the number of entries.
- The third location in the table contains the first word of data.

Parameters

Table 2-4 shows the LIFO box and describes the parameters.

Table 2-4 Last In/First Out Unload Table (FC87) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	TABLE	Pointer*	I, Q, M, D	Points to the starting location of the LIFO table
	RET_VAL	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 remains unchanged and the ENO bit 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 the example, the newest entry in the LIFO table is returned as the function value (MW2.0). The number of entries is decremented from 5 to 4.

If the function is executed successfully, the signal states of ENO and Q 4.0 are set to 1.

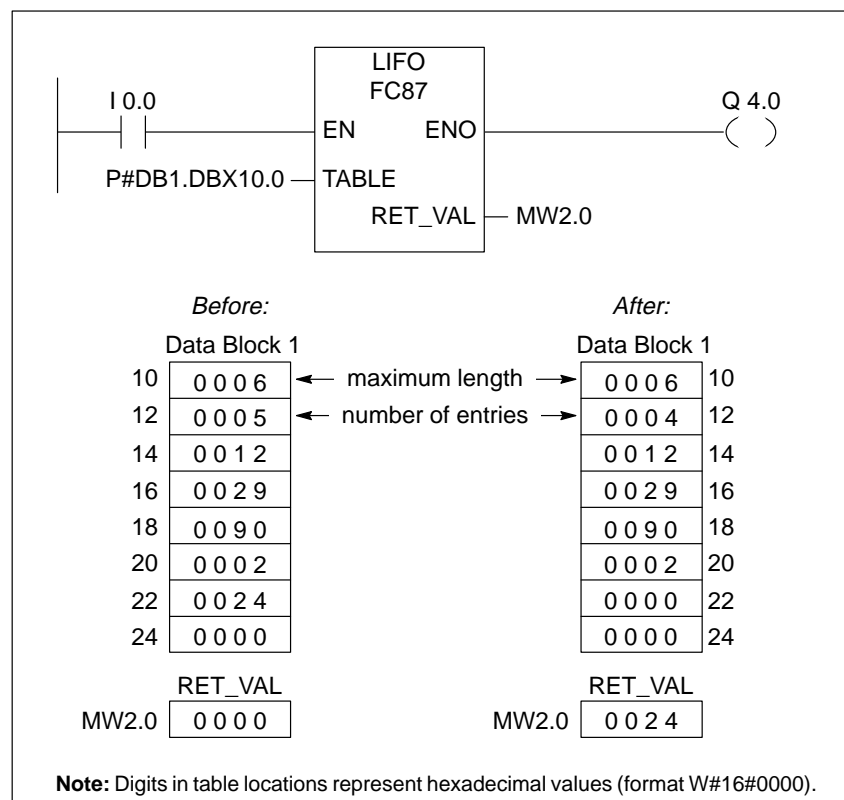


Figure 2-4 Last In/First Out Unload Table (LIFO)

2.6 Table (TBL): FC88

Description

The Table (TBL) function performs the indicated operation (set by CMD) to the source table and stores the result in the same table location.

- The length of the table (number of elements) is stored in the first word of the table.
- The first element of the table starts at the second word.
- The E_TYPE parameter is a coded byte that determines the element size.

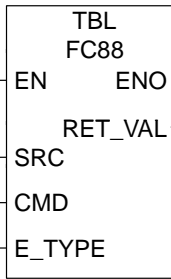
Note

If the E_TYPE is set to REAL (8), then the CMD value for Ones complement (3) is invalid.

Parameters

Table 2-5 shows the TBL box and describes the parameters.

Table 2-5 Table (FC88) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	SRC	Pointer*	I, Q, M, D	Points to the starting location of the table
	CMD	BYTE	I, Q, M, D, L, P	Indicates type of operation to be done: 3 = Ones complement 4 = Clear 5 = Negate 6 = Square root
	E_TYPE	BYTE	I, Q, M, D, L, P	Indicates the type of table elements: 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = REAL
	RET_VAL	WORD	I, Q, M, D, L, P	Returns a value of 0 if the instruction is executed successfully; see Error Information for values other than 0.

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

Error Information

If an invalid E_TYPE or an invalid CMD is used, the ENO bit is set to 0.

The following values are returned by the function:

- RET_VAL = 0 indicates no failure.
- RET_VAL = 8 indicates CMD and E_TYPE are inconsistent or invalid.

Example

Figure 2-5 shows how the TBL instruction works. 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 location 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 location causes the next five table locations to be cleared.

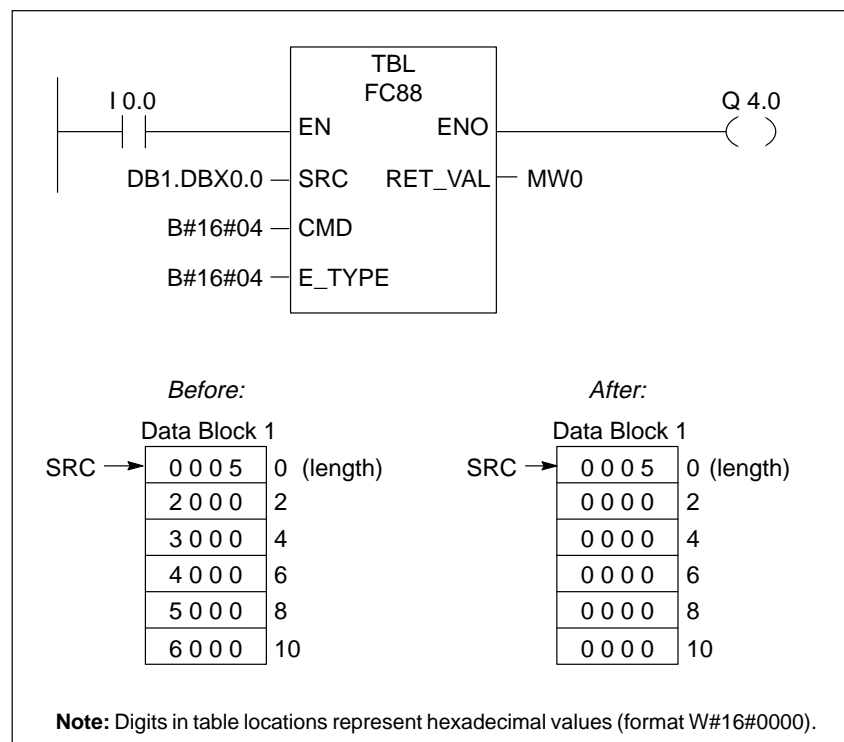


Figure 2-5 Table (TBL)

2.7 Move Table to Word (TBL_WRD): FC89

Description

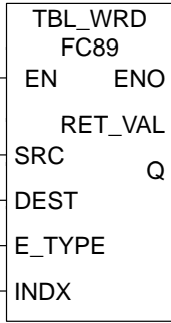
The Move Table to Word (TBL_WRD) function copies the element indicated by INDX from the SRC table to the location pointed to by DEST, and then increments INDX as long as INDX is less than the maximum length that is given in the first word of the table, SRC[0]. If the INDX is set to the last table entry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

- The length of the table (number of elements) is stored in the first word of the table.
- The first element of the table starts at the second word.

Parameters

Table 2-6 shows the TBL_WRD box and describes the parameters.

Table 2-6 Move Table to Word (FC89) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	SRC	Pointer*	I, Q, M, D	Points to the starting location of the table
	DEST	Pointer*	I, Q, M, D	Points to the word (destination) location
	E_TYPE	BYTE	I, Q, M, D, L, P	Indicates the type of table elements 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = REAL
	Q	BOOL	Q, M, D, L	Indicates 0 if the INDX variable contains the last entry of the table when the function is called
	INDX	WORD	I, Q, M, L	Element number of the entry to move
	RET_VAL	WORD	I, Q, M, D, L, P	Returns a value of 0 if the instruction is executed successfully; see Error Information for values other than 0.

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

Error Information

If an invalid E_TYPE or an invalid index is used, the ENO bit is set to 0.

The following values are returned by the function:

- RET_VAL = 0 indicates no failure
- RET_VAL = 7 indicates index is at table location 0
- RET_VAL = 8 indicates invalid E_TYPE
- RET_VAL = 9 indicates 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 6, the double word data that is stored in the table starting at the memory location pointed to by SRC is copied to the location pointed to by DEST. The INDX value points to the table location to be moved. After the instruction is executed successfully, the INDX is automatically incremented to one location past the entry that was moved. In the example, one location is a double word, or 4 bytes. In this example, when the instruction is called, the INDX is not set to the last table location, so Q is set to 1 after execution.

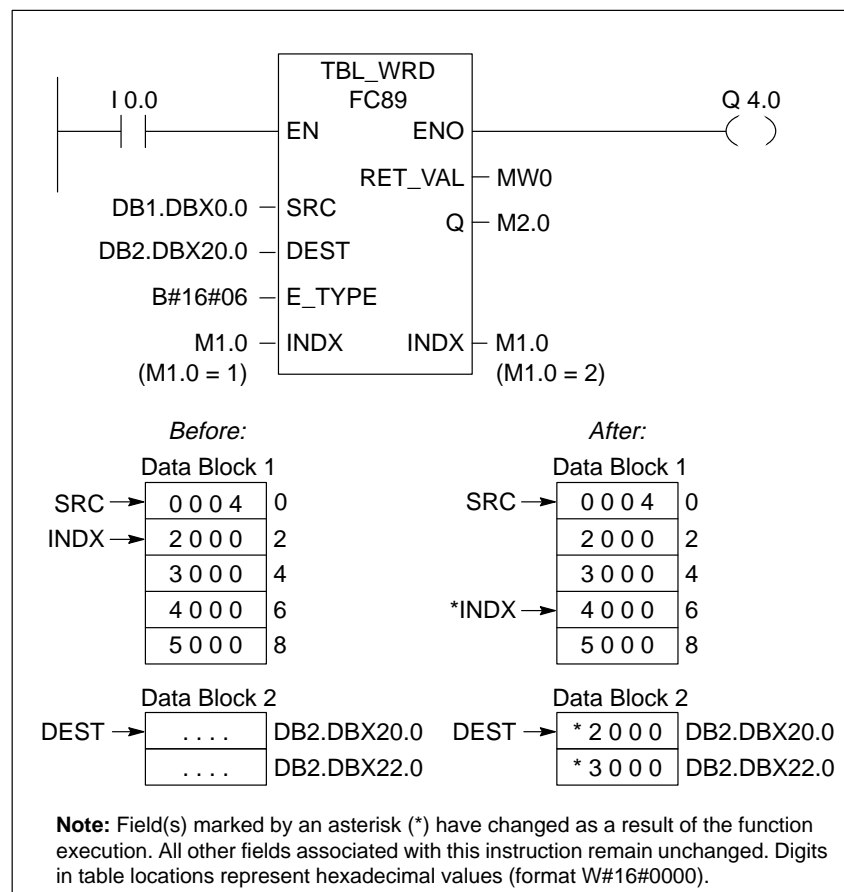


Figure 2-6 Move Table to Word (TBL_WRD)

2.8 Word to Table (WRD_TBL): FC91

Description

The Word to Table (WRD_TBL) function performs the indicated operation (CMD) between the source element (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 length of the table (number of elements) is stored in the first word of the table.
- The first element of the table starts at the second word.
- The E_TYPE parameter is a coded byte that determines the element size. If E_TYPE is real, then CMD can only be “Move.”

Parameters

Table 2-7 shows the WRD_TBL box and describes the parameters.

Table 2-7 Word to Table (FC91) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	SRC	Pointer*	I, Q, M, D	Points to the source data element
	TABLE	Pointer*	I, Q, M, D	Points to the starting location of the table
	CMD	BYTE	I, Q, M, D, L, P	Indicates the type of operation to be performed by the instruction: 14 = Move 7 = AND 8 = OR 9 = XOR
	E_TYPE	BYTE	I, Q, M, D, L, P	Indicates the type of table elements 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = REAL
	Q	BOOL	Q, M, D, L	Indicates 0 if the INDX variable contains the last entry of the table
	INDX	WORD	I, Q, M, D, L	Element number of the entry to be operated on
	RET_VAL	WORD	I, Q, M, D, L, P	Returns a value of 0 if the instruction is executed successfully; see Error Information for values other than 0.

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

Error Information

If an invalid E_TYPE is used, or if an invalid index of 0 is used, the ENO bit is set to 0.

The following values are returned by the function:

RET_VAL = 0 indicates no failure.

RET_VAL = 7 indicates index is at table location 0.

RET_VAL = 8 indicates CMD and E_TYPE are inconsistent or invalid.

RET_VAL = 9 indicates 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 length field in the first word indicates that the table contains three double words. The INDX value points to the table location to be manipulated. Since the CMD value is 8, the instruction performs an OR operation 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 operation (77775555) is written back into the table, and the INDX is automatically incremented one location. 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 location, so Q is set to 1 after execution.

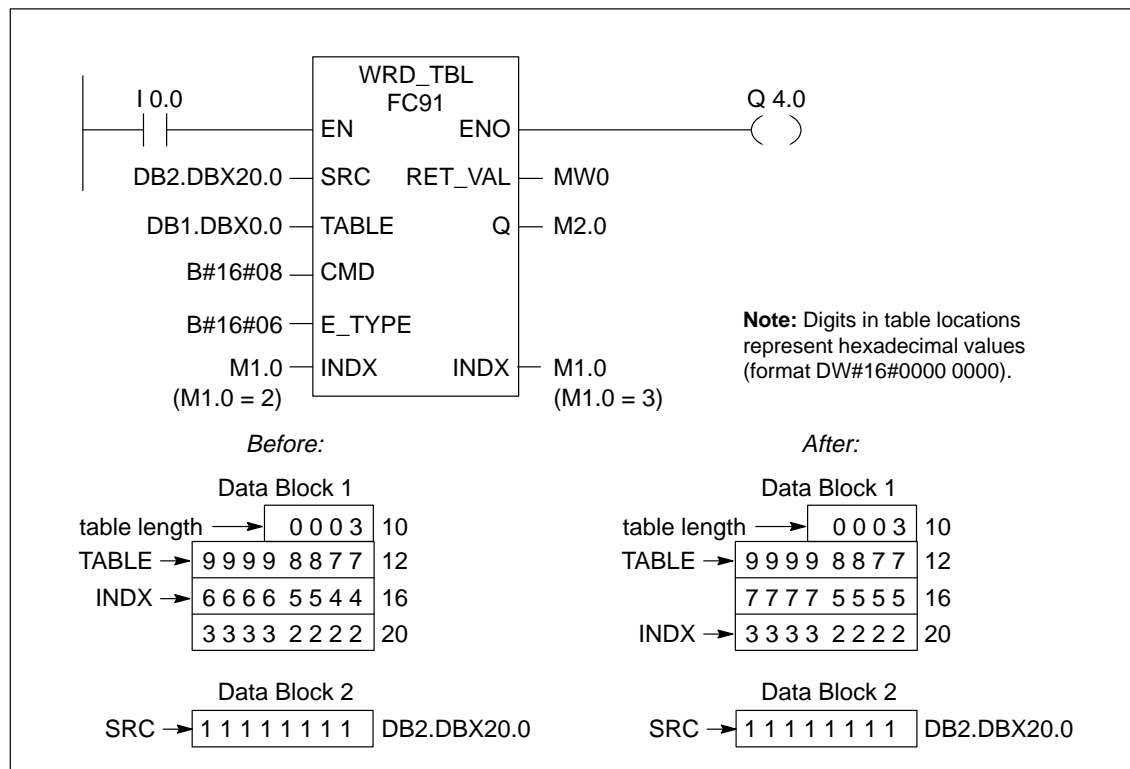
2

Figure 2-7 Word to Table (WRD_TBL)

3

Shift Functions

3.1	Overview	3-3
	What Is Described in This Chapter?	3-3
	Where Do You Look For More Information?	3-3
3.2	Word Shift Register (WSR): FC90	3-4
	Description	3-4
	Parameters	3-4
	Error Information	3-5
	Example	3-5
3.3	Bit Shift Register (SHRB): FC92	3-6
	Description	3-6
	Parameters	3-6
	Error Information	3-7
	Example	3-7

Figures

3-1	Word Shift Register (WSR)	3-5
3-2	Bit Shift Register (SHRB)	3-7

Tables

3-1	Word Shift Register (FC90) and Parameters	3-4
3-2	Bit Shift Register (FC92) and Parameters	3-6

3.1 Overview

What Is Described in This Chapter?

This section describes shift functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

3.2 Word Shift Register (WSR): FC90

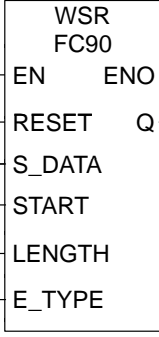
Description

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 shows the WSR box and describes the parameters.

Table 3-1 Word Shift Register (FC90) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	RESET	BOOL	I, Q, M, D, L	Resets the shift register when set to 1
	S_DATA	Pointer*	I, Q, M, D	Points to the source data element to be inserted into the table
	START	Pointer*	I, Q, M, D	Points to the starting location of the table
	LENGTH	WORD	I, Q, M, D, L, P	Number of items to be shifted
	E_TYPE	BYTE	I, Q, M, D, L, P	Indicates the type of table elements 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = REAL
	Q	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 ENO bit 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.

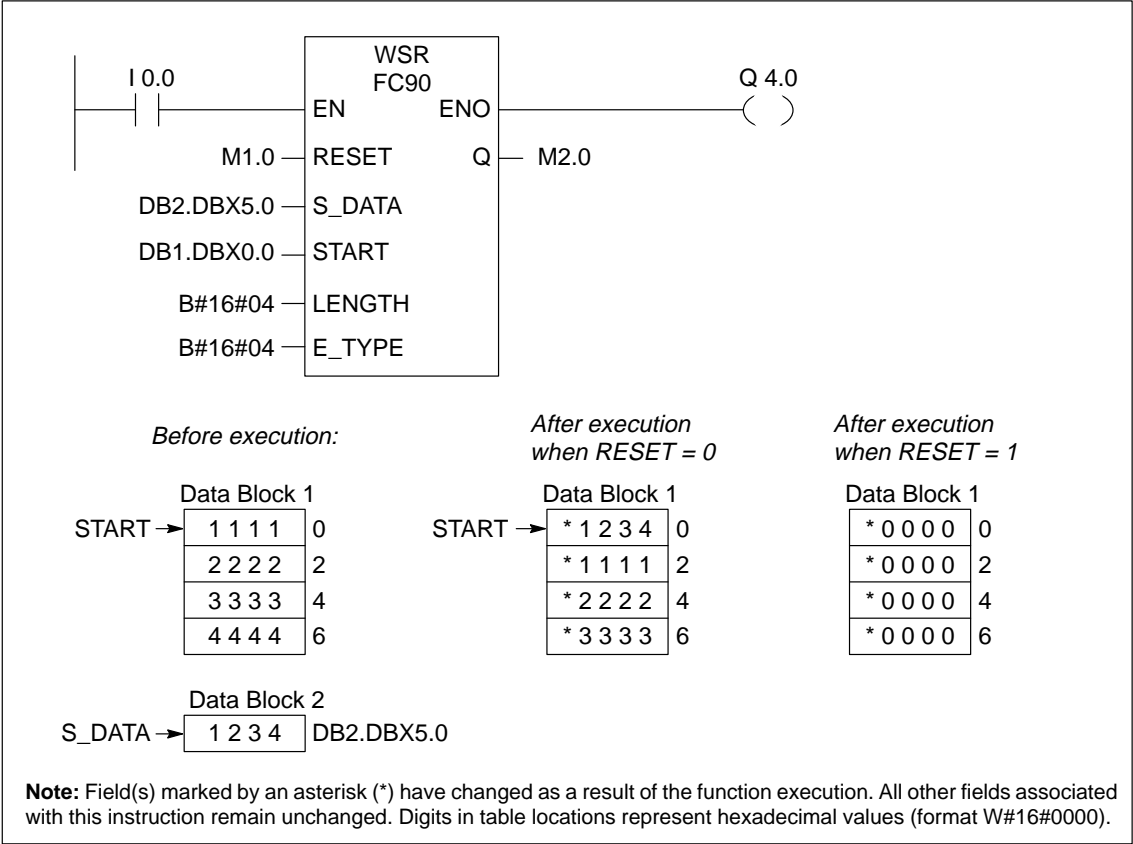


Figure 3-1 Word Shift Register (WSR)

3.3 Bit Shift Register (SHRB): FC92

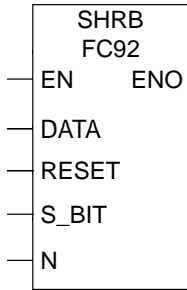
Description

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 shows the SHRB box and describes the parameters.

Table 3-2 Bit Shift Register (FC92) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	DATA	BOOL	I, Q, M, D, L	Source data bit
	RESET	BOOL	I, Q, M, D, L	Resets the shift register when set to 1
	S_BIT	Pointer*	I, Q, M, D	Points to the starting bit in the shift register
	N	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.

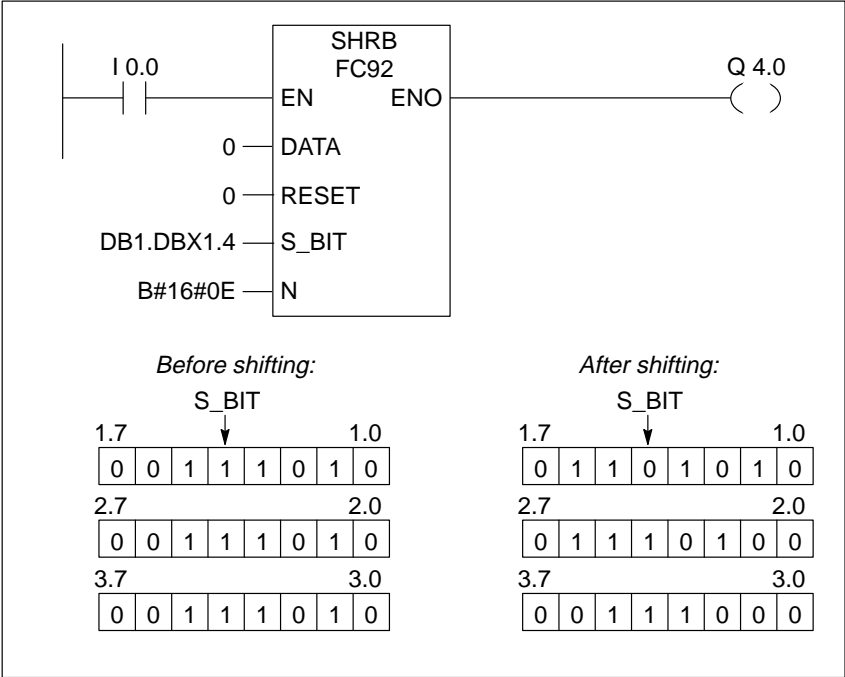


Figure 3-2 Bit Shift Register (SHRB)

4

Move Functions

4.1	Overview	4-3
	What Is Described in This Chapter?	4-3
	Where Do You Look For More Information?	4-3
4.2	Indirect Block Move (IBLKMOV): FC81	4-4
	Description	4-4
	Parameters	4-4
	Error Information	4-5
	Example	4-5

Figures

4-1	Indirect Block Move (IBLKMOV)	4-5
-----	-------------------------------------	-----

Tables

4-1	Indirect Block Move (FC81) and Parameters	4-4
-----	---	-----

4.1 Overview

What Is Described in This Chapter?

This section describes move functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

4.2 Indirect Block Move (IBLKMOV): FC81

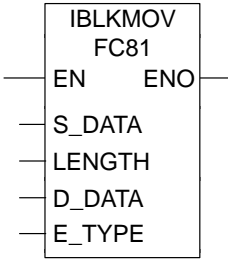
Description

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 shows the IBLKMOV box and describes the parameters.

Table 4-1 Indirect Block Move (FC81) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	S_DATA	Pointer*	I, Q, M, D	Points to a pointer which identifies the starting location of the source data
	LENGTH	Pointer*	I, Q, M, D	Points to the length of the block to be moved
	D_DATA	Pointer*	I, Q, M, D	Points to a pointer which identifies the starting location of the destination data
	E_TYPE	BYTE (coded as described at right)	I, Q, M, D, L	The E_TYPE parameter determines the element size, coded as follows: 2 = BYTE 4 = WORD 5 = INT 6 = DWORD 7 = DINT 8 = 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 (no data is moved) and the ENO bit 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, 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 no errors occur, the signal states of ENO and Q 4.0 are set to 1.

If the signal state of input I 0.0 is 0 (not activated), the signal state of both EN and ENO is 0.

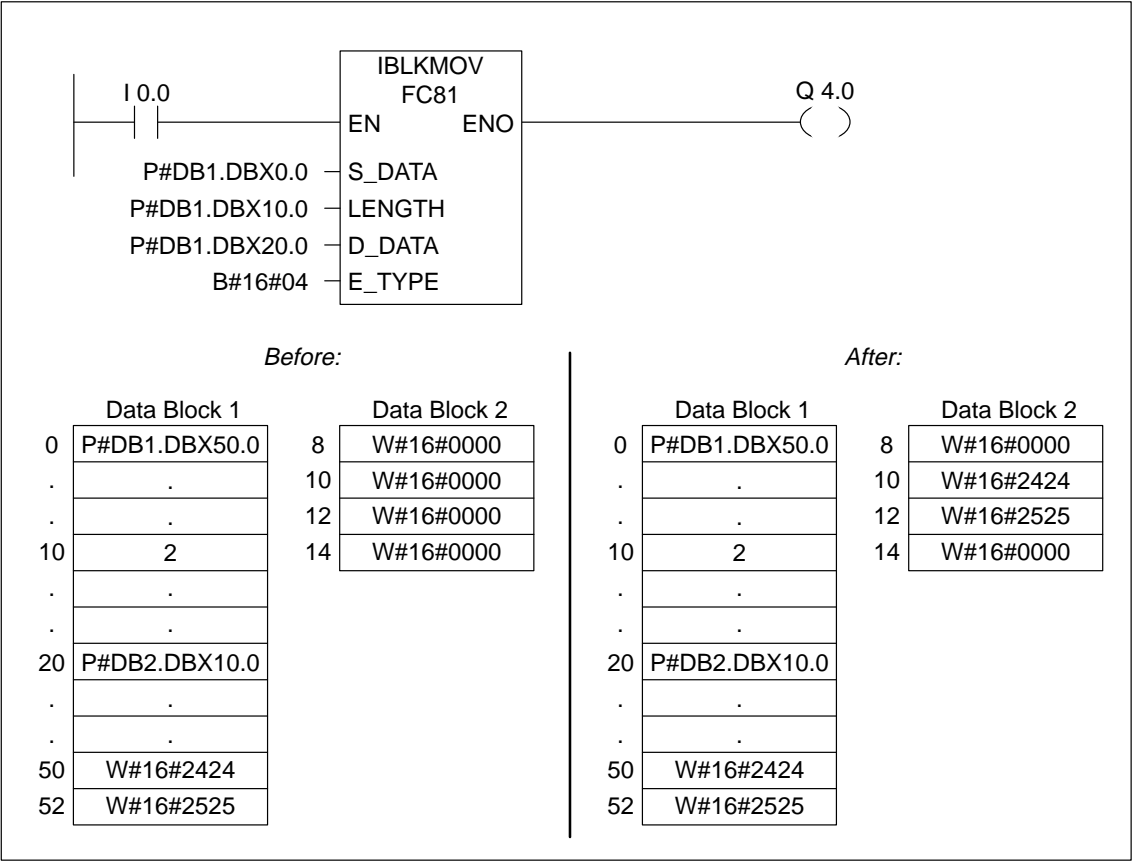


Figure 4-1 Indirect Block Move (IBLKMOV)

5

Timer Functions

5.1	Overview	5-3
	What Is Described in This Chapter?	5-3
	Where Do You Look For More Information?	5-3
5.2	Software Timer On Delay—Retentive (TONR): FC80	5-4
	Description	5-4
	Parameters	5-4
	Error Information	5-5
	Example	5-5

Figures

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

Tables

5-1	Software Timer On Delay—Retentive (FC80) and Parameters .	5-4
-----	---	-----

5.1 Overview

What Is Described in This Chapter?

This section describes timer functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

5.2 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

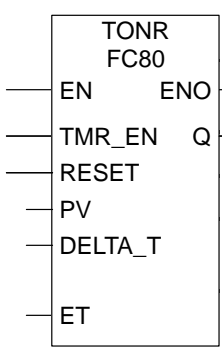
You must move the OB scan time from the start-up local variables in the variable declaration table of the OB to the global variable DELTA_T.

As long as the signal state of RESET is 0, the signal state of TMR_EN is 1, and the signal state of Q is 0, 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 shows the TONR box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	TMR_EN	BOOL	I, Q, M, D, L	Enables timer to accumulate time
	RESET	BOOL	I, Q, M, D, L	If RESET = 1, the timer is reset to 0.
	PV	DINT	I, Q, M, D, L, P or constant	Preset value
	DELTA_T	INT	I, Q, M, D, L or constant	OB scan time for previous cycle
	Q	BOOL	Q, M, D, L	If ET equals or exceeds PV, the signal state of Q is set to 1.
	ET	DINT	I, Q, M, D, L or constant	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 and the signal states of I 0.2 and Q 1.1 are both 0, then DELTA_T is added to ET ($100 + 50 = 150$). Since ET is less than PV, the signal state of Q 1.1 remains 0. If no errors occurred, the signal states of ENO and Q 4.0 are set to 1.

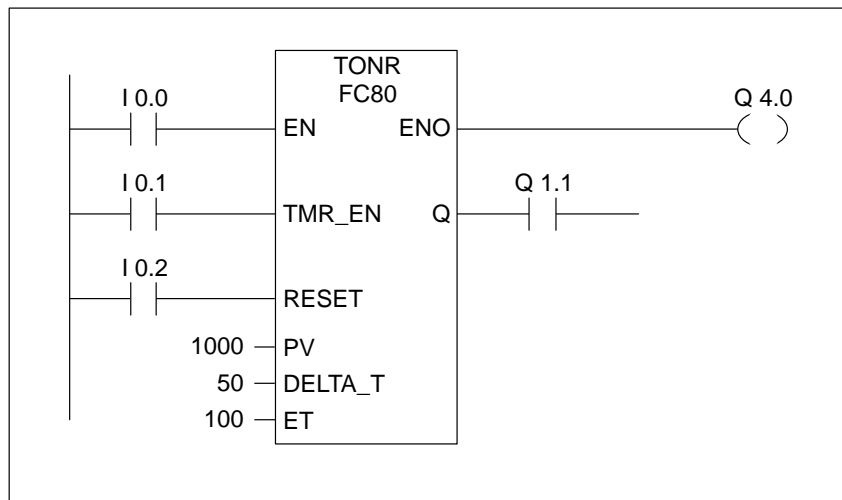


Figure 5-1 Timer On Delay—Retentive (TONR)

Conversion Functions

6.1	Overview	6-3
	What Is Described in This Chapter?	6-3
	Where Do You Look For More Information?	6-3
6.2	Seven Segment Decoder (SEG): FC93	6-4
	Description	6-4
	Parameters	6-4
	Error Information	6-5
	Example	6-5
6.3	ASCII to Hex (ATH): FC94	6-6
	Description	6-6
	Parameters	6-6
	Error Information	6-6
	Example	6-7
6.4	Hex to ASCII (HTA): FC95	6-8
	Description	6-8
	Parameters	6-8
	Error Information	6-8
	Example	6-9
6.5	Encode Binary Position (ENCO): FC96	6-10
	Description	6-10
	Parameters	6-10
	Error Information	6-11
	Example	6-11
6.6	Decode Binary Position (DECO): FC97	6-12
	Description	6-12
	Parameters	6-12
	Error Information	6-13
	Example	6-13
6.7	Tens Complement (BCDCPL): FC98	6-14
	Description	6-14
	Parameters	6-14
	Error Information	6-15
	Example	6-15
6.8	Sum Number of Bits (BITSUM): FC99	6-16
	Description	6-16
	Parameters	6-16
	Error Information	6-17
	Example	6-17

Figures

6-1	Seven Segment Output Bit Patterns	6-4
6-2	Seven Segment Decoder (SEG)	6-5
6-3	ASCII to Hex (ATH)	6-7
6-4	ASCII Characters and Equivalent Hexadecimal Values	6-7
6-5	Hex to ASCII (HTA)	6-9
6-6	Hexadecimal Digits and Equivalent ASCII Hex Values	6-9
6-7	Encode Binary Position (ENCO)	6-11
6-8	Decode Binary Position (DECO)	6-13
6-9	Tens Complement (BCDCPL)	6-15
6-10	Sum Number of Bits (BITSUM)	6-17

Tables

6-1	Seven Segment Decoder (FC93) and Parameters	6-4
6-2	ASCII to Hex (FC94) and Parameters	6-6
6-3	Hex to ASCII (FC95) and Parameters	6-8
6-4	Encode Binary Position (FC96) and Parameters	6-10
6-5	Decode Binary Position (FC97) and Parameters	6-12
6-6	Tens Complement (FC98) and Parameters	6-14
6-7	Sum Number of Bits (FC99) and Parameters	6-16

6.1 Overview

What Is Described in This Chapter?

This section describes conversion functions (FCs) that you can add to your standard set of instructions to provide additional programming flexibility. Each function is listed with the full name, the abbreviation, and the FC number. Each function is described according to the following topics:

- Description — a basic functional description.
- Parameters — a table showing the box instruction, a description of each parameter, and the memory areas that are valid for each parameter.
- Error Information — errors that would prevent the function from being executed.
- Example — a figure consisting of a representation of the function with example parameters and a graphic representation of the results of the function execution.

Where Do You Look For More Information?

For more information on programming and addressing, refer to the following sources:

- *STEP 7 Program Design Programming Manual*
- *STEP 7 Ladder Logic Reference Manual*
- *STEP 7 Statement List Reference Manual*
- On-line Help

6.2 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 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

7-segment Display

Figure 6-1 Seven Segment Output Bit Patterns

Parameters

Table 6-1 shows the SEG box and describes the parameters.

Table 6-1 Seven Segment Decoder (FC93) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	WORD	I, M, D, P, or constant	Source data word in four hexadecimal digits
	OUT	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. The signal states of ENO and Q 4.0 are set to 1.

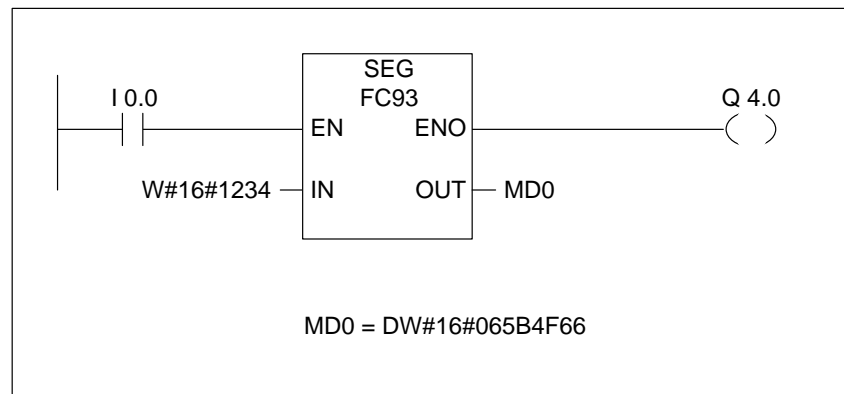


Figure 6-2 Seven Segment Decoder (SEG)

6.3 ASCII to Hex (ATH): FC94

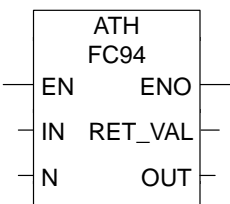
Description

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 shows the ATH box and describes the parameters.

Table 6-2 ASCII to Hex (FC94) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	Pointer*	I, Q, M, D, L	Points to the starting location of an ASCII string
	N	INT	I, Q, M, L, P	Number of ASCII input characters to be converted
	OUT	Pointer*	Q, M, D, L	Points to the starting location of the table
	RET_VAL	WORD	I, Q, M, D, L, P	Returns a value of 0 if the instruction is executed successfully; see Error Information for values other than 0.

* 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 and the ENO bit is set to 0.

The following values are returned by the function:

- RET_VAL = 0 indicates successful conversion (no invalid ASCII characters are found).
- RET_VAL = 7 indicates an invalid ASCII character input, and the character becomes a 0 during the conversion.

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.)

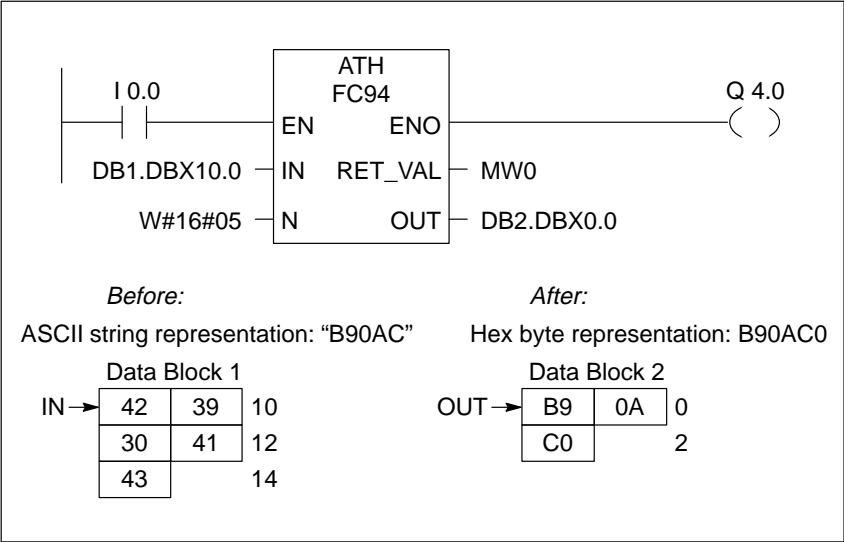


Figure 6-3 ASCII to Hex (ATH)

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

6.4 Hex to ASCII (HTA): FC95

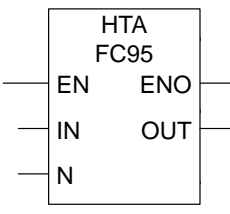
Description

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 shows the HTA box and describes the parameters.

Table 6-3 Hex to ASCII (FC95) and Parameters

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

After the function is executed, the signal states of ENO and Q 4.0 are set to 1.

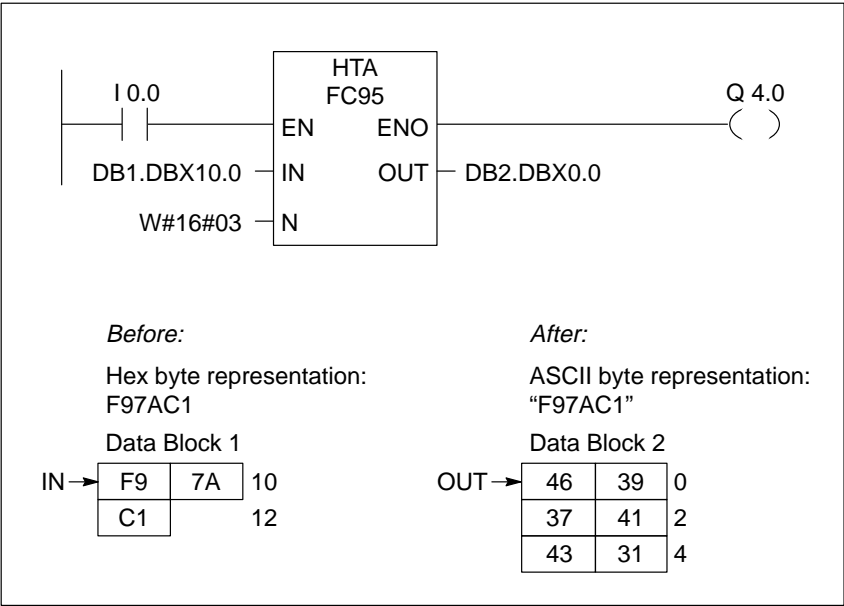


Figure 6-5 Hex to ASCII (HTA)

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

6.5 Encode Binary Position (ENCO): FC96

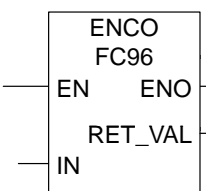
Description

The Encode Binary Position (ENCO) function converts the contents of IN to the 5-bit binary number corresponding to the bit position of the rightmost 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 shows the ENCO box and describes the parameters.

Table 6-4 Encode Binary Position (FC96) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	DWORD	I, M, D, L, P, or constant	Value to be encoded
	RET_VAL	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. The signal states of ENO and Q 4.0 are set to 1.

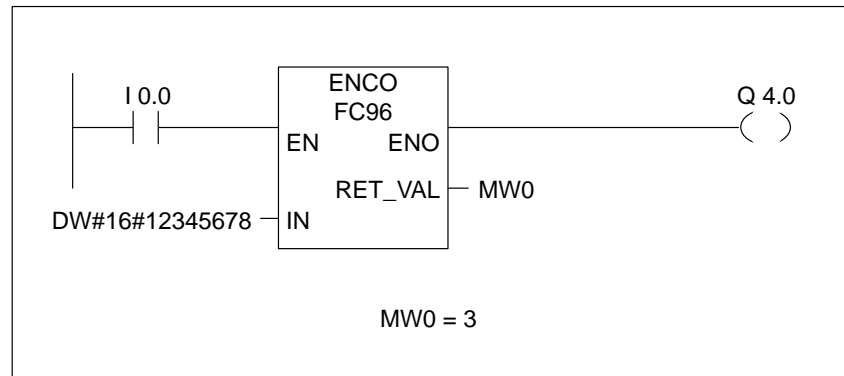


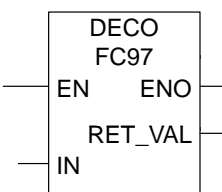
Figure 6-7 Encode Binary Position (ENCO)

6.6 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 shows the DECO box and describes the parameters.

Table 6-5 Decode Binary Position (FC97) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	WORD	I, M, D, L, P, or constant	Variable to decode
	RET_VAL	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. The signal states of ENO and Q 4.0 are set to 1.

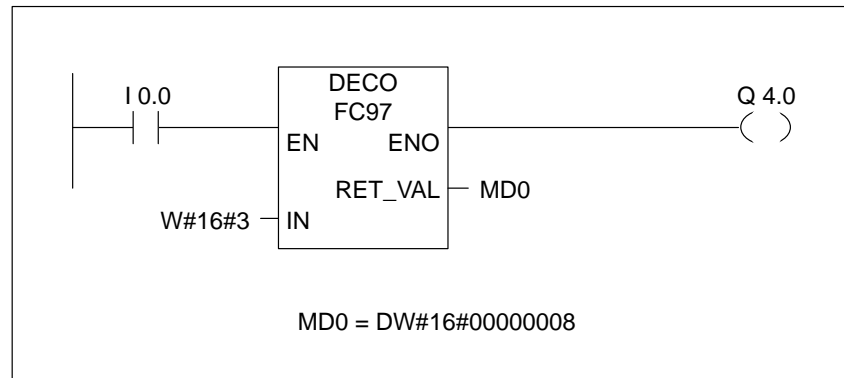


Figure 6-8 Decode Binary Position (DECO)

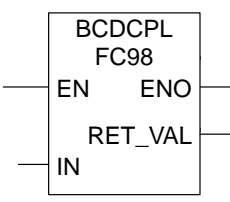
6.7 Tens Complement (BCDCPL): FC98

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:

$$\begin{aligned} &10000000 \text{ (in BCD)} \\ &- \text{7-digit BCD value} \\ &= \text{Tens complement value (in BCD)} \end{aligned}$$

Parameters Table 6-6 shows the BCDCPL box and describes the parameters.

Table 6-6 Tens Complement (FC98) and Parameters

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	DWORD	I, M, D, L, P, or constant	7-digit BCD number
	RET_VAL	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 BCD CPL instruction works. If the signal state of input I 0.0 is 1 (activated), the BCD CPL function is executed. The signal states of ENO and Q 4.0 are set to 1.

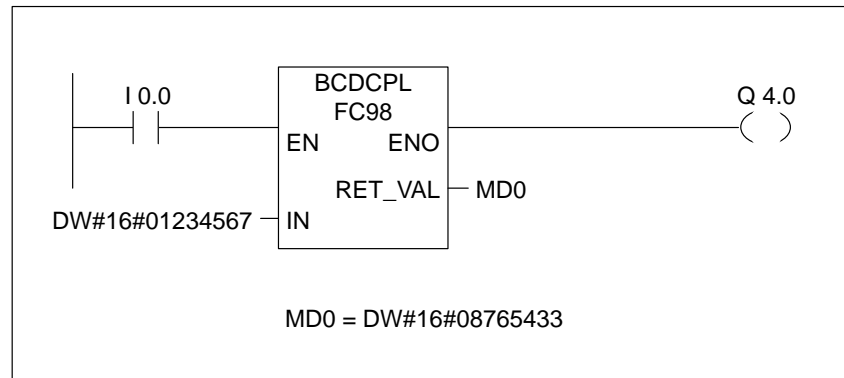


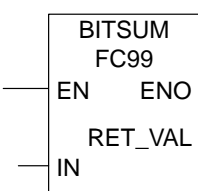
Figure 6-9 Tens Complement (BCD CPL)

6.8 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 shows the BITSUM box and describes the parameters.

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

LAD Box	Parameter	Data Type	Memory Area	Description
	EN	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box.
	ENO	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error.
	IN	DWORD	I, M, D, L, P, or constant	Variable to count bits in
	RET_VAL	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 the 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. After the function is executed, the signal states of ENO and Q4.0 are set to 1.

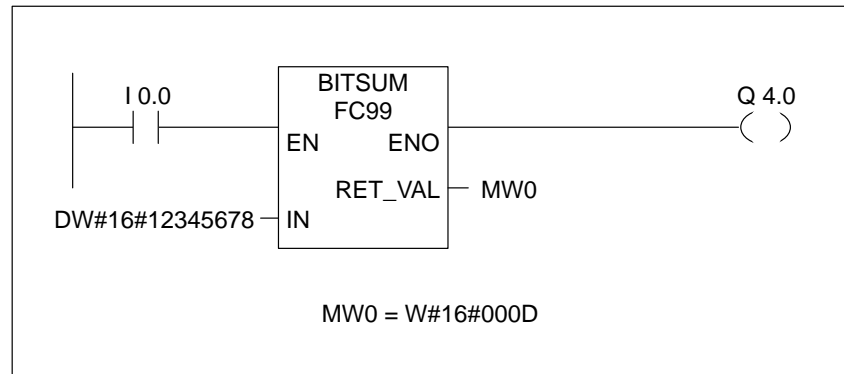


Figure 6-10 Sum Number of Bits (BITSUM)

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

A

Add to table (ATT), 2-4-2-5
ASCII to Hex (ATH), 6-6-6-7
Assistance, technical, iv

B

Bit logic functions
 reset range of immediate outputs (RSETI),
 1-6-1-7
 reset range of outputs (RSET), 1-4-1-5
 set range of immediate outputs (SETI),
 1-10-1-12
 set range of outputs (SET), 1-8-1-9
Bit shift register (SHRB), 3-6-3-8

C

Conversion functions
 ASCII to hex (ATH), 6-6-6-7
 decode binary position (DECO), 6-12-6-13
 encode binary position (ENCO), 6-10-6-11
 hex to ASCII (HTA), 6-8-6-9
 seven segment decoder (SEG), 6-4-6-5
 sum number of bits (BITSUM), 6-16-6-18
 tens complement (BCDCPL), 6-14-6-15

D

Decode binary position (DECO), 6-12-6-13

E

Encode binary position (ENCO), 6-10-6-11

F

First in/first out unload table (FIFO), 2-6-2-7
Functions (FCs)
 list, iv
 location, iii

H

Hex to ASCII (HTA), 6-8-6-9

I

Indirect block move (IBLKMOV), 4-4-4-6

L

Last in/first out unload table (LIFO), 2-10-2-11
Location of functions (FCs), iii

M

Manuals, related, iii
Move functions, indirect block move (IBLK-
 MOV), 4-4-4-6
Move table to word (TBL_WRD), 2-14-2-15

R

Related manuals, iii
Reset range of immediate outputs (RSETI),
 1-6-1-7
Reset range of outputs (RSET), 1-4-1-5

S

Set range of immediate outputs (SETI),
1-10-1-12
Set range of outputs (SET), 1-8-1-9
Seven segment decoder (SEG), 6-4-6-5
Shift functions
 bit shift register (SHRB), 3-6-3-8
 word shift register (WSR), 3-4-3-5
Software timer on delay—retentive (TONR),
5-4-5-6
Sum number of bits (BITSUM), 6-16-6-18

T

Table (TBL), 2-12-2-13
Table find (TBL_FIND), 2-8-2-9

Table functions

add to table (ATT), 2-4-2-5
first in/first out unload table (FIFO), 2-6-2-7
last in/first out unload table (LIFO),
2-10-2-11
move table to word (TBL_WRD), 2-14-2-15
table (TBL), 2-12-2-13
table find (TBL_FIND), 2-8-2-9
word to table (WRD_TBL), 2-16-2-18
Technical assistance, iv
Tens complement (BCDCPL), 6-14-6-15
Timer functions, software timer on delay—re-
tentive (TONR), 5-4-5-6

W

Word shift register (WSR), 3-4-3-5
Word to table (WRD_TBL), 2-16-2-18

Siemens AG
AUT 1282
Östliche Rheinbrückenstr. 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:

- | | |
|--|---|
| <input type="checkbox"/> Automotive | <input type="checkbox"/> Pharmaceutical |
| <input type="checkbox"/> Chemical | <input type="checkbox"/> Plastic |
| <input type="checkbox"/> Electrical Machinery | <input type="checkbox"/> Pulp and Paper |
| <input type="checkbox"/> Food | <input type="checkbox"/> Textiles |
| <input type="checkbox"/> Instrument and Control | <input type="checkbox"/> Transportation |
| <input type="checkbox"/> Nonelectrical Machinery | <input type="checkbox"/> Other |
| <input type="checkbox"/> Petrochemical | |



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 do not forget to state the title, order number, and release of your manual.

Title of Your Manual:

Order No. of Your Manual: Release:

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:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....