

# SIEMENS

## SIMATIC

### Function Block Diagram (FBD) for S7-300 and S7-400 Programming

#### Reference Manual

This manual is part of the documentation  
package with the order number:  
**6ES7810-4CA08-8BW1**

**Edition 03/2006**  
A5E00706955-01

Preface, Contents	
Bit Logic Instructions	<b>1</b>
Comparison Instructions	<b>2</b>
Conversion Instructions	<b>3</b>
Counter Instructions	<b>4</b>
Data Block Instructions	<b>5</b>
Jump Instructions	<b>6</b>
Integer Math Instructions	<b>7</b>
Floating-point Math Instructions	<b>8</b>
Move Instructions	<b>9</b>
Program Control Instructions	<b>10</b>
Shift and Rotate Instructions	<b>11</b>
Status Bit Instructions	<b>12</b>
Timer Instructions	<b>13</b>
Word Logic Instructions	<b>14</b>
<b>Appendix</b>	
Overview of All FBD Instructions	<b>A</b>
Programming Examples	<b>B</b>
Working with Function Block Diagram	<b>C</b>
Index	

## Safety Guidelines

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring to property damage only have no safety alert symbol. The notices shown below are graded according to the degree of danger.



---

### Danger

indicates that death or severe personal injury **will** result if proper precautions are not taken.

---



---

### Warning

indicates that death or severe personal injury **may** result if proper precautions are not taken.

---



---

### Caution

with a safety alert symbol indicates that minor personal injury can result if proper precautions are not taken.

---

---

### Caution

without a safety alert symbol indicates that property damage can result if proper precautions are not taken.

---

---

### Notice

indicates that an unintended result or situation can occur if the corresponding notice is not taken into account.

---

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

## Qualified Personnel

The device/system may only be set up and used in conjunction with this documentation. Commissioning and operation of a device/system may only be performed by **qualified personnel**. Within the context of the safety notices in this documentation qualified persons are defined as persons who are authorized to commission, ground and label devices, systems and circuits in accordance with established safety practices and standards.

## Prescribed 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.

Correct, reliable operation of the product requires proper transport, storage, positioning and assembly as well as careful operation and maintenance.

---

## Trademarks

All names identified by ® are registered trademarks of the Siemens AG.

The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

## Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

# Preface

## Purpose

This manual is your guide to creating user programs in the Function Block Diagram (FBD) programming language.

The manual also includes a reference section that describes the syntax and functions of the language elements of Function Block Diagram.

## Basic Knowledge Required

The manual is intended for S7 programmers, operators, and maintenance/service personnel.

In order to understand this manual, general knowledge of automation technology is required.

In addition to, computer literacy and the knowledge of other working equipment similar to the PC (e.g. programming devices) under the operating systems MS Windows 2000 Professional, MS Windows XP Professional or MS Windows Server 2003 are required.

## Scope of the Manual

This manual is valid for release 5.4 of the STEP 7 programming software package.

## Compliance with Standards

FBD corresponds to the "Function Block Diagram" language defined in the International Electrotechnical Commission's standard IEC 1131-3. For further details, refer to the table of standards in the STEP 7 file NORM\_TBL.WRI.

## Requirements

To use the Function Block Diagram manual effectively, you should already be familiar with the theory behind S7 programs which is documented in the online help for STEP 7. The language packages also use the STEP 7 standard software, so you should be familiar with handling this software and have read the accompanying documentation.

This manual is part of the documentation package "STEP 7 Reference".

The following table displays an overview of the STEP 7 documentation:

Documentation	Purpose	Order Number
STEP 7 Basic Information with <ul style="list-style-type: none"> <li>• Working with STEP 7, Getting Started Manual</li> <li>• Programming with STEP 7</li> <li>• Configuring Hardware and Communication Connections, STEP 7</li> <li>• From S5 to S7, Converter Manual</li> </ul>	Basic information for technical personnel describing the methods of implementing control tasks with STEP 7 and the S7-300/400 programmable controllers.	6ES7810-4CA08-8BW0
STEP 7 Reference with <ul style="list-style-type: none"> <li>• Ladder Logic (LAD)/Function Block Diagram (FBD)/Statement List (STL) for S7-300/400 manuals</li> <li>• Standard and System Functions for S7-300/400 Volume 1 and Volume 2</li> </ul>	Provides reference information and describes the programming languages LAD, FBD, and STL, and standard and system functions extending the scope of the STEP 7 basic information.	6ES7810-4CA08-8BW1

Online Helps	Purpose	Order Number
Help on STEP 7	Basic information on programming and configuring hardware with STEP 7 in the form of an online help.	Part of the STEP 7 Standard software.
Reference helps on STL/LAD/FBD Reference help on SFBs/SFCs Reference help on Organization Blocks	Context-sensitive reference information.	Part of the STEP 7 Standard software.

## Online Help

The manual is complemented by an online help which is integrated in the software. This online help is intended to provide you with detailed support when using the software.

The help system is integrated in the software via a number of interfaces:

- The context-sensitive help offers information on the current context, for example, an open dialog box or an active window. You can open the context-sensitive help via the menu command **Help > Context-Sensitive Help**, by pressing F1 or by using the question mark symbol in the toolbar.
- You can call the general Help on STEP 7 using the menu command **Help > Contents** or the "Help on STEP 7" button in the context-sensitive help window.
- You can call the glossary for all STEP 7 applications via the "Glossary" button.

This manual is an extract from the "Help on Function Block Diagram". As the manual and the online help share an identical structure, it is easy to switch between the manual and the online help.

## Further Support

If you have any technical questions, please get in touch with your Siemens representative or responsible agent.

You will find your contact person at:

<http://www.siemens.com/automation/partner>

You will find a guide to the technical documentation offered for the individual SIMATIC Products and Systems here at:

<http://www.siemens.com/simatic-tech-doku-portal>

The online catalog and order system is found under:

<http://mall.automation.siemens.com/>

## Training Centers

Siemens offers a number of training courses to familiarize you with the SIMATIC S7 automation system. Please contact your regional training center or our central training center in D 90327 Nuremberg, Germany for details:

Telephone: +49 (911) 895-3200.

Internet: <http://www.sitrain.com>

## Technical Support

You can reach the Technical Support for all A&D products

- Via the Web formula for the Support Request  
<http://www.siemens.com/automation/support-request>
- Phone: + 49 180 5050 222
- Fax: + 49 180 5050 223

Additional information about our Technical Support can be found on the Internet pages <http://www.siemens.com/automation/service>

## Service & Support on the Internet

In addition to our documentation, we offer our Know-how online on the internet at:

<http://www.siemens.com/automation/service&support>

where you will find the following:

- The newsletter, which constantly provides you with up-to-date information on your products.
- The right documents via our Search function in Service & Support.
- A forum, where users and experts from all over the world exchange their experiences.
- Your local representative for Automation & Drives.
- Information on field service, repairs, spare parts and more under "Services".

# Contents

<b>1</b>	<b>Bit Logic Instructions</b>	<b>1-1</b>
1.1	Overview of Bit Logic Instructions .....	1-1
1.2	>=1 : OR Logic Operation.....	1-2
1.3	& : AND Logic Operation .....	1-3
1.4	AND-before-OR Logic Operation and OR-before-AND Logic Operation .....	1-4
1.5	XOR : Exclusive OR Logic Operation.....	1-6
1.6	Insert Binary Input.....	1-7
1.7	Negate Binary Input.....	1-8
1.8	= : Assign .....	1-9
1.9	# : Midline Output.....	1-11
1.10	R : Reset Output .....	1-13
1.11	S : Set Output .....	1-14
1.12	RS : Reset_Set Flip Flop .....	1-15
1.13	SR : Set_Reset Flip Flop .....	1-17
1.14	N : Negative RLO Edge Detection.....	1-19
1.15	P : Positive RLO Edge Detection.....	1-20
1.16	SAVE : Save RLO to BR Memory .....	1-21
1.17	NEG : Address Negative Edge Detection.....	1-22
1.18	POS : Address Positive Edge Detection.....	1-23
<b>2</b>	<b>Comparison Instructions</b>	<b>2-1</b>
2.1	Overview of Comparison Instructions.....	2-1
2.2	CMP ? I : Compare Integer.....	2-2
2.3	CMP ? D : Compare Double Integer.....	2-3
2.4	CMP ? R : Compare Real.....	2-4
<b>3</b>	<b>Conversion Instructions</b>	<b>3-1</b>
3.1	Overview of Conversion Instructions .....	3-1
3.2	BCD_I : BCD to Integer .....	3-2
3.3	I_BCD : Integer to BCD .....	3-4
3.4	BCD_DI : BCD to Double Integer .....	3-5
3.5	I_DI : Integer to Double Integer .....	3-7
3.6	DI_BCD : Double Integer to BCD .....	3-8
3.7	DI_R : Double Integer to Real .....	3-9
3.8	INV_I : Ones Complement Integer .....	3-10
3.9	INV_DI : Ones Complement Double Integer .....	3-11
3.10	NEG_I : Twos Complement Integer.....	3-12
3.11	NEG_DI : Twos Complement Double Integer.....	3-13
3.12	NEG_R : Negate Real Number .....	3-14
3.13	ROUND : Round to Double Integer .....	3-15
3.14	TRUNC : Truncate Double Integer Part.....	3-16
3.15	CEIL : Ceiling.....	3-17
3.16	FLOOR : Floor .....	3-18

<b>4</b>	<b>Counter Instructions</b>	<b>4-1</b>
4.1	Overview of Counter Instructions .....	4-1
4.2	S_CUD : Assign Parameters and Count Up/Down .....	4-3
4.3	S_CU : Assign Parameters and Count Up .....	4-5
4.4	S_CD : Assign Parameters and Count Down.....	4-7
4.5	SC : Set Counter Value .....	4-9
4.6	CU : Up Counter .....	4-11
4.7	CD : Down Counter.....	4-12
<b>5</b>	<b>Data Block Instructions</b>	<b>5-1</b>
5.1	OPN : Open Data Block.....	5-1
<b>6</b>	<b>Jump Instructions</b>	<b>6-1</b>
6.1	Overview of Jump Instructions .....	6-1
6.2	JMP : Unconditional Jump in a Block .....	6-2
6.3	JMP : Conditional Jump in a Block .....	6-3
6.4	JMPN : Jump-If-Not .....	6-5
6.5	LABEL : Jump Label.....	6-7
<b>7</b>	<b>Integer Math Instructions</b>	<b>7-1</b>
7.1	Overview of Integer Math Instructions .....	7-1
7.2	Evaluating the Bits of the Status Word with Integer Math Instructions.....	7-2
7.3	ADD_I : Add Integer.....	7-3
7.4	SUB_I : Subtract Integer.....	7-5
7.5	MUL_I : Multiply Integer.....	7-6
7.6	DIV_I : Divide Integer.....	7-7
7.7	ADD_DI : Add Double Integer .....	7-8
7.8	SUB_DI : Subtract Double Integer.....	7-9
7.9	MUL_DI : Multiply Double Integer.....	7-10
7.10	DIV_DI : Divide Double Integer .....	7-11
7.11	MOD_DI : Return Fraction Double Integer .....	7-12
<b>8</b>	<b>Floating-Point Math Instructions</b>	<b>8-1</b>
8.1	Overview of Floating-Point Math .....	8-1
8.2	Evaluating the Bits of the Status Word with Floating-Point Instructions.....	8-2
8.3	Basic Instructions.....	8-3
8.3.1	ADD_R : Add Real.....	8-3
8.3.2	SUB_R : Subtract Real.....	8-5
8.3.3	MUL_R : Multiply Real.....	8-6
8.3.4	DIV_R : Divide Real.....	8-7
8.3.5	ABS : Forming the Absolute Value of a Floating-Point Number.....	8-8
8.4	Extended Instructions .....	8-9
8.4.1	SQR : Forming the Square of a Floating-Point Number.....	8-9
8.4.2	SQRT : Forming the Square Root of a Floating-Point Number .....	8-10
8.4.3	EXP : Forming the Exponential Value of a Floating-Point Number.....	8-11
8.4.4	LN : Forming the Natural Logarithm of a Floating-Point Number .....	8-12
8.4.5	Forming Trigonometric Functions of Angles as Floating-Point Numbers.....	8-13
<b>9</b>	<b>Move Instructions</b>	<b>9-1</b>
9.1	MOVE : Assign Value .....	9-1



<b>10</b>	<b>Program Control Instructions</b>	<b>10-1</b>
10.1	Overview of Program Control Instructions.....	10-1
10.2	CALL : Calling an FC/SFC without Parameters .....	10-2
10.3	CALL_FB : Call FB as Box .....	10-4
10.4	CALL_FC (Call FC as Box).....	10-6
10.5	CALL_SFB : Call System FB as Box.....	10-8
10.6	CALL_SFC (Call System FC as Box) .....	10-10
10.7	Calling Multiple Instances.....	10-12
10.8	Calling a Block from a Library.....	10-12
10.9	Master Control Relay Instructions .....	10-13
10.10	Important Notes on Using MCR Functions.....	10-14
10.11	MCR</MCR> : Master Control Relay On/Off.....	10-15
10.12	MCRA/MCRD : Master Control Relay Activate/Deactivate .....	10-19
10.13	RET : Return.....	10-22
<b>11</b>	<b>Shift and Rotate Instructions</b>	<b>11-1</b>
11.1	Shift Instructions .....	11-1
11.1.1	Overview of Shift Instructions .....	11-1
11.1.2	SHR_I : Shift Right Integer .....	11-2
11.1.3	SHR_DI : Shift Right Double Integer .....	11-4
11.1.4	SHL_W : Shift Left Word.....	11-6
11.1.5	SHR_W : Shift Right Word.....	11-8
11.1.6	SHL_DW : Shift Left Double Word .....	11-9
11.1.7	SHR_DW : Shift Right Double Word .....	11-10
11.2	Rotate Instructions.....	11-12
11.2.1	Overview of Rotate Instructions.....	11-12
11.2.2	ROL_DW : Rotate Left Double Word.....	11-12
11.2.3	ROR_DW : Rotate Right Double Word.....	11-14
<b>12</b>	<b>Status Bit Instructions</b>	<b>12-1</b>
12.1	Overview of Status Bit Instructions.....	12-1
12.2	OV : Exception Bit Overflow .....	12-2
12.3	OS : Exception Bit Overflow Stored.....	12-4
12.4	UO : Exception Bit Unordered .....	12-6
12.5	BR : Exception Bit BR Memory.....	12-7
12.6	<> 0 : Result Bits .....	12-8
<b>13</b>	<b>Timer Instructions</b>	<b>13-1</b>
13.1	Overview of Timer Instructions.....	13-1
13.2	Memory Areas and Components of a Timer.....	13-2
13.3	S_PULSE : Assign Pulse Timer Parameters and Start.....	13-5
13.4	S_PEXT : Assign Extended Pulse Timer Parameters and Start .....	13-7
13.5	S_ODT : Assign On-Delay Timer Parameters and Start .....	13-9
13.6	S_ODTS : Assign Retentive On-Delay Timer Parameters and Start .....	13-11
13.7	S_OFFDT : Assign Off-Delay Timer Parameters and Start.....	13-13
13.8	SP : Start Pulse Timer .....	13-15
13.9	SE : Start Extended Pulse Timer .....	13-17
13.10	SD : Start On-Delay Timer.....	13-19
13.11	SS : Start Retentive On-Delay Timer.....	13-21
13.12	SF Start Off-Delay Timer .....	13-23

<b>14</b>	<b>Word Logic Instructions</b>	<b>14-1</b>
14.1	Overview of Word Logic Instructions .....	14-1
14.2	WAND_W : AND Word (Word) .....	14-2
14.3	WOR_W : OR Word (Word).....	14-3
14.4	WXOR_W : Exclusive OR Word (Word).....	14-4
14.5	WAND_DW : AND Double Word (Word) .....	14-5
14.6	WOR_DW : OR Double Word (Word) .....	14-6
14.7	WXOR_DW : Exclusive OR Double Word (Word).....	14-7
<b>A</b>	<b>Overview of All FBD Instructions</b>	<b>A-1</b>
A.1	FBD Instructions Sorted According to German Mnemonics (SIMATIC).....	A-1
A.2	FBD Instructions Sorted According to English Mnemonics (International).....	A-5
<b>B</b>	<b>Programming Examples</b>	<b>B-1</b>
B.1	Overview of Programming Examples .....	B-1
B.2	Example: Bit Logic Instructions .....	B-2
B.3	Example: Timer Instructions .....	B-5
B.4	Example: Counter and Comparison Instructions.....	B-9
B.5	Example: Integer Math Instructions .....	B-12
B.6	Example: Word Logic Instructions .....	B-13
<b>C</b>	<b>Working with Function Block Diagram</b>	<b>C-1</b>
C.1	EN/ENO Mechanism.....	C-1
C.1.1	Adder with EN and with ENO Connected.....	C-2
C.1.2	Adder with EN and without ENO Connected .....	C-3
C.1.3	Adder without EN and with ENO Connected .....	C-4
C.1.4	Adder without EN and without ENO Connected .....	C-5
C.2	Parameter Transfer.....	C-6
<b>Index</b>		<b>Index-1</b>

# 1 Bit Logic Instructions

## 1.1 Overview of Bit Logic Instructions

### Description

Bit logic instructions work with two digits, 1 and 0. These two digits form the base of a number system called the binary system. The two digits 1 and 0 are called binary digits or bits. In conjunction with AND, OR, XOR and outputs, a 1 stands for logical YES and a 0 for logical NO.

The bit logic instructions interpret signal states of 1 and 0 and combine them according to Boolean logic. These combinations produce a result of 1 or 0 that is called the "result of logic operation" (RLO).

There are bit logic instructions to perform the following functions:

- AND, OR and Exclusive OR: these instructions check the signal state and produce a result that is either copied to the RLO bit or combined with it.
- AND-before-OR Logic Operation and OR-before-AND Logic Operation
- Assign and Midline Output. these instructions assign the RLO or store it temporarily.

The following instructions react to an RLO of 1:

- S : Set Output
- R : Reset Output
- SR : Set\_Reset Flip Flop
- RS : Reset\_Set Flip Flop

Other instructions react to a positive or negative edge transition to perform the following functions:

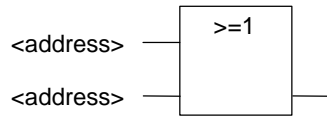
- N : Negative RLO Edge Detection
- P : Positive RLO Edge Detection
- NEG : Address Negative Edge Detection
- POS : Address Positive Edge Detection

The remaining instructions affect the RLO directly in the following ways:

- Insert Binary Input
- Negate Binary Input
- SAVE : Save RLO to BR Memory

## 1.2 >=1 : OR Logic Operation

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, T, C, D, L	The address indicates the bit whose signal state will be checked.

### Description

With the **OR** instruction, you can check the signal states of two or more specified addresses at the inputs of an OR box.

If the signal state of one of the addresses is 1, the condition is satisfied and the instruction produces the result 1. If the signal state of all addresses is 0, the condition is not satisfied and the instruction produces the result 0.

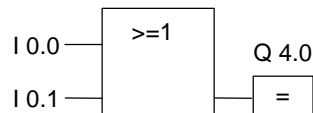
If the **OR** instruction is the first instruction in a string of logic operations, it saves the result of its signal state check in the RLO bit.

Each **OR** instruction that is not the first instruction in the string of logic operations combines the result of its signal state check with the value stored in the RLO bit. These values are combined according to the OR truth table.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

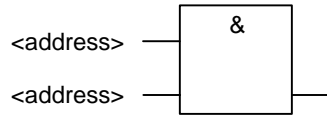
### Example



Output Q4.0 is set when the signal state is 1 at input I0.0 OR at input I0.1.

### 1.3 & : AND Logic Operation

#### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, T, C, D, L	The address indicates the bit whose signal state will be checked.

#### Description

With the **AND** instruction, you can check the signal states of two or more specified addresses at the inputs of an AND box.

If the signal state of all operands is 1, the condition is satisfied and the instruction provides the result 1. If the signal state of an address is 0, the condition is not satisfied and the instruction produces the result 0.

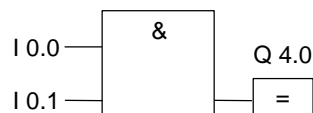
If the **AND** instruction is the first instruction in a string of logic operations, it saves the result of its signal state check in the RLO bit.

Every **AND** instruction that is not the first instruction in the string of logic operations, combines the result of its signal state check with the value stored in the RLO bit. These values are combined according to the AND truth table.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

#### Example



Output Q4.0 is set when the signal state is 1 at input I0.0 AND I0.1.

## 1.4 AND-before-OR Logic Operation and OR-before-AND Logic Operation

### Description

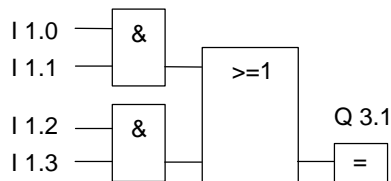
With the **AND-before-OR** instruction, you can check the result of a signal state according to the OR truth table.

With an AND-before-OR logic operation the signal state is 1 when at least one AND logic operation is satisfied.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example



The signal state is 1 at output Q3.1 when at least one AND logic operation is satisfied

The signal state is 0 at output Q3.1 when no AND logic operation is satisfied.

### Description

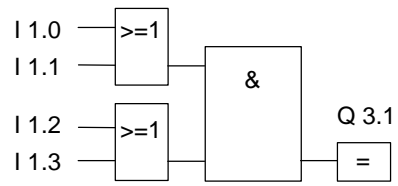
With the **OR-before-AND** instruction, you can check the result of a signal state check according to the AND truth table.

With an **OR-before-AND** logic operation the signal state is 1 when all OR logic operations are satisfied.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

**Example**

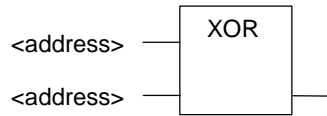


The signal state is 1 at output Q3.1 when both OR logic operations are satisfied.

The signal state is 0 at output Q3.1 when at least one OR logic operation is not satisfied.

## 1.5 XOR : Exclusive OR Logic Operation

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, T, C, D, L	The address indicates the bit whose signal state will be checked.

### Description

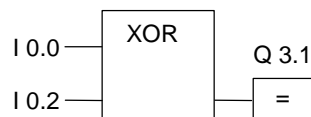
With the **Exclusive OR** instruction, you can check the result of a signal state check according to the Exclusive OR truth table.

With an **Exclusive OR** logic operation, the signal state is 1 when the signal state of one of the two specified addresses is 1. You can also use the Exclusive OR function several times. The mutual result of logic operation is then "1" if an impair number of checked addresses is "1".

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example



The signal state is 1 at output Q3.1 when the signal state is 1 at either input I0.0 OR at input I0.2 (exclusively, in other words not at both).



## 1.6 Insert Binary Input

### Symbol

<address>



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, T, C, D, L	The address indicates the bit whose signal state will be checked.

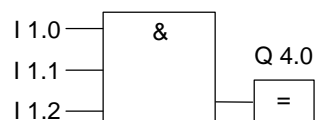
### Description

The **Insert Binary Input** instruction inserts a further binary input to an AND, OR, or XOR box.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	1	X	-

### Example



Output Q4.0 is 1 when the signal state at I1.0 AND I1.1 AND I1.2 is 1.

## 1.7 Negate Binary Input

### Symbol



### Description

The **Negate Binary Input** instruction negates the RLO.

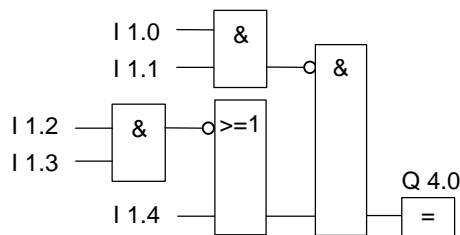
When you negate the result of logic operation, you must remember certain rules:

- If the result of logic operation at the first input of an AND or OR box is negated, there is no nesting.
- If the result of logic operation is negated but not at the first input of an OR box, the entire binary logic operation before the input is included in the OR logic operation.
- If the result of logic operation is negated but not at the first input of a AND box, the entire binary logic operation before the input is included in the AND logic operation.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	1	X	-

### Example

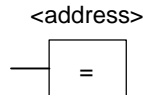


Output Q4.0 is 1 when:

- the signal state at I1.0 AND I1.1 is NOT 1
- AND the signal state at I1.2 AND I1.3 is NOT 1
- OR the signal state at I1.4 is NOT 1.

## 1.8 = : Assign

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies the bit to which the signal state of the string of logic operations is assigned.

### Description

The **Assign** instruction produces the result of logic operation. The box at the end of a logic operation has the signal 1 or 0 according to the following criteria:

- The output has the signal 1 when the conditions of the logic operation before the output box are satisfied.
- The output has the signal 0 when the conditions of the logic operation before the output box are not satisfied.

The FBD logic operation assigns the signal state to the output that is addressed by the instruction (to achieve the same effect, the signal state of the RLO bit could also be assigned to the address). If the conditions of the FBD logic operations are satisfied, the signal state at the output box is 1. Otherwise the signal state is 0. The **Assign** instruction is influenced by the Master Control Relay (MCR).

For more detailed information about the functions of the MCR, refer to MCR on/off.

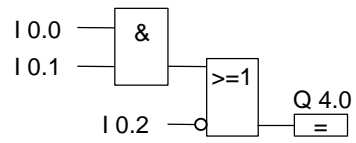
You can only place the **Assign** box at the right-hand end of the string of logic operations. You can, however, use several **Assign** boxes.

You can create a negated assignment with the **Negate Input** instruction.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	-	0

### Example

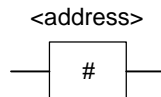


The signal state at output Q4.0 is 1 when:

- the signal state is 1 at inputs I0.0 AND I0.1
- OR I0.2 is 0

## 1.9 # : Midline Output

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, *L	The address specifies the bit to which the RLO will be assigned.

\* You can only use an address in the local data stack if it is declared in the variable declaration table in the TEMP area of a code block (FC, FB, OB).

### Description

The **Midline Output** instruction is an intermediate element that buffers the RLO. More precisely, this element buffers the bit logic operation of the last branch to be opened before the Midline Output.

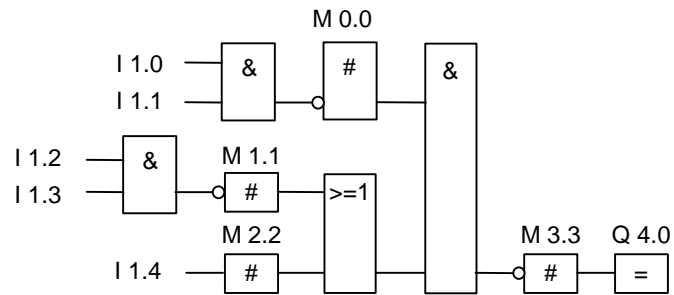
The **Midline Output** instruction is affected by the Master Control Relay (MCR). For more detailed information about how the MCR functions, refer to MCR on/off.

You can create a negated Midline Output by negating the input of the Midline Output.

### Status Word

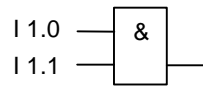
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	-	1

**Example**

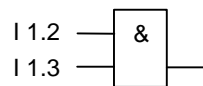


The Midline Outputs buffer the following results of the logic operations:

M0.0 buffers the negated RLO of



M1.1 saves the negated RLO of

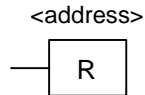


M2.2 saves the RLO of I1.4

M3.3 saves the negated RLO of the entire bit logic operation

## 1.10 R : Reset Output

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL TIMER COUNTER	I, Q, M, T, C, D, L	The address specifies which bit will be reset.

### Description

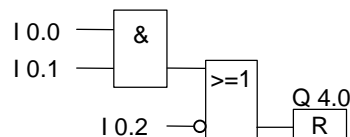
The **Reset Output** instruction is only executed when the RLO is 1. If the RLO is 1, this instruction resets the specified address to 0. If the RLO is 0, the instruction does not affect the specified address which remains unchanged.

The **Reset Output** instruction is affected by the Master Control Relay (MCR). For more detailed information about how the MCR functions, refer to MCR on/off.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	-	0

### Example



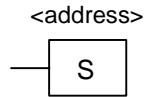
The signal state at output Q4.0 is reset to 0 only when:

- The signal state is 1 at inputs I0.0 AND I0.1
- OR the signal state at input I0.2 is 0.

If the RLO of the branch is 0, the signal state at output Q4.0 is unchanged.

## 1.11 S : Set Output

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies which bit will be set.

### Description

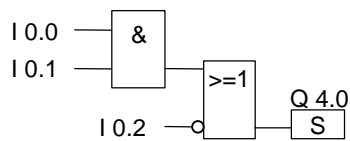
The **Set Output** instruction is only executed when the RLO is 1. If the RLO is 1, this instruction sets the specified address to 1. If the RLO is 0, the instruction does not affect the specified address which remains unchanged.

The **Set Output** instruction is affected by the Master Control Relay (MCR). For more detailed information about how the MCR functions, refer to MCR on/off.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	-	0

### Example



The signal state at output Q4.0 is set to 1 only when:

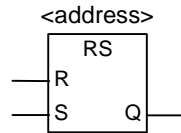
- The signal state is 1 at inputs I0.0 AND I0.1
- OR the signal state at input I0.2 is 1.

If the RLO of the branch is 0, the signal state of Q4.0 is not changed.



## 1.12 RS : Reset\_Set Flip Flop

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies which bit will be set or reset.
S	BOOL	I, Q, M, D, L, T, C	Reset instruction enabled
R	BOOL	I, Q, M, D, L, T, C	Set instruction enabled
Q	BOOL	I, Q, M, D, L	Signal state of <address>

### Description

The **Reset\_Set Flip Flop** instruction executes instructions such as Set (S) or Reset (R) only when the RLO is 1. An RLO of 0 does not affect these instructions, the address specified in the instruction is not changed.

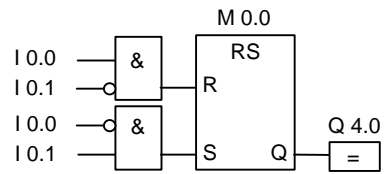
**Reset\_Set Flip Flop** is reset when the signal state at input R is 1 and the signal state at input S is 0. If input R is 0 and input S is 1, the flip flop is set. If the RLO at both inputs is 1, the flip flop is set.

The **Reset\_Set Flip Flop** instruction is affected by the Master Control Relay (MCR). For more detailed information about how the MCR functions, refer to MCR on/off.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example

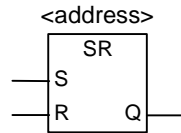


If I0.0 is 1 and I0.1 is 0, the memory bit M0.0 is reset and output Q4.0 is 0. If I0.0 is 0 and I0.1 is 1, the memory bit M0.0 is set and output Q4.0 is 1.

If both signal states are 0, there is no change. If both signal states are 1, the Set instruction dominates due to the order of the instructions. M 0.0 is set and Q4.0 is 1.

## 1.13 SR : Set\_Reset Flip Flop

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies which bit will be set or reset.
S	BOOL	I, Q, M, D, L, T, C	Set instruction enabled
R	BOOL	I, Q, M, D, L, T, C	Reset instruction enabled
Q	BOOL	I, Q, M, D, L	Signal state of <address>

### Description

The **Set\_Reset Flip Flop** instruction executes Set (S) or Reset (R) instructions only when the RLO is 1. An RLO of 0 has no effect on these instructions, the address specified in the instruction remains unchanged.

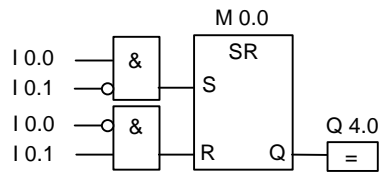
**Set\_Reset Flip Flop** is set when the signal state at input S is 1 and the signal state at input R is 0. If input S is 0 and input R is 1, the flip flop is reset. If the RLO at both inputs is 1 the flip flop is reset.

The **Set\_Reset Flip Flop** instruction is affected by the Master Control Relay (MCR). For more detailed information about how the MCR functions, refer to MCR on/off.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example

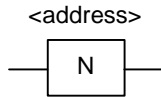


If I0.0 is 1 and I0.1 is 0, memory bit M0.0 is set and Q4.0 is 1. If I0.0 is 0 and I0.1 is 1, the memory bit M0.0 is reset and Q4.0 is 0.

If both signal states are 0, there is no change. If both signal states are 1, the reset instruction dominates due to the order of the instructions. M0.0 is reset and Q 4.0 is 0.

## 1.14 N : Negative RLO Edge Detection

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies which edge memory bit will store the previous RLO.

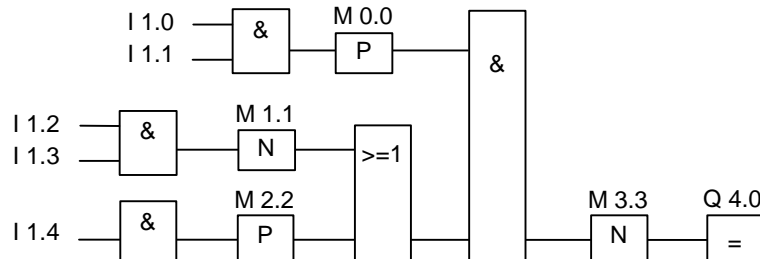
### Description

The **Negative RLO Edge Detection** instruction detects a change from 1 to 0 (falling edge) at the specified address and indicates this by setting the RLO to 1 after the instruction. The current signal state of the RLO is compared with the signal state of the address (the edge memory bit). If the signal state of the address is 1 and the RLO prior to the instruction was 0, the RLO is 1 (pulse) after the instruction, in all other cases it is 0. The RLO prior to the instruction is stored in the address.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	X	1

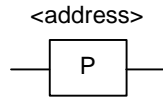
### Example



The edge memory bit M3.3 stores the signal state of the previous RLO.

## 1.15 P : Positive RLO Edge Detection

### Symbol



Parameter	Data Type	Memory Area	Description
<address>	BOOL	I, Q, M, D, L	The address specifies which edge memory bit will store the previous RLO.

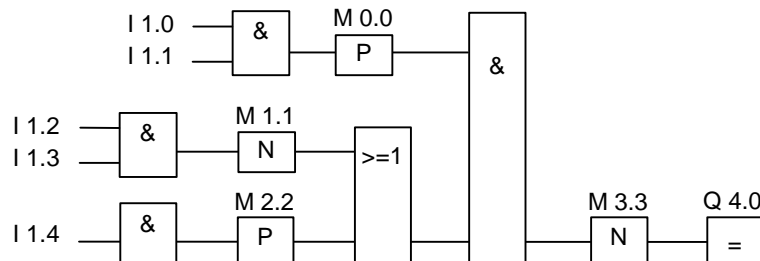
### Description

The **Positive RLO Edge Detection** instruction detects a change from 0 to 1 (rising edge) at the specified address and indicates this with an RLO of 1 after the instruction. The current signal state at the RLO is compared with the signal state of the address (the edge memory bit). If the signal state of the address is 0 and the RLO is 1 before the instruction, the RLO will be 1 (pulse) after the instruction, in all other cases the RLO is 0. The RLO prior to the instruction is stored in the address.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	X	X	1

### Example



The edge memory bit M3.3 stores the signal state of the previous RLO.

## 1.16 SAVE : Save RLO to BR Memory

### Symbol



### Description

The **Save RLO to BR Memory** instruction saves the RLO in the BR bit of the status word. The first check bit FC is not reset.

For this reason, if there is an AND logic operation in the next network, the state of the BR bit is included in the logic operation.

For the instruction **SAVE** (LAD, FBD, STL), the following applies and not the recommended use specified in the manual and online help:

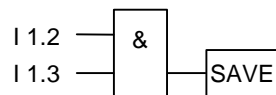
We do not recommend that you use SAVE and then check the BR bit in the same block or in subordinate blocks, because the BR bit can be modified by many instructions occurring inbetween. It is advisable to use the SAVE instruction before exiting a block, since the ENO output (= BR bit) is then set to the value of the RLO bit and you can then check for errors in the block.

With the **Save RLO to BR Memory** instruction, the RLO of a network can form part of a logic operation in a subordinate block. The CALL instruction in the calling block resets the first check bit.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	-	-	-

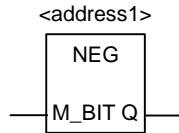
### Example



The result of logic operation (RLO) is written to the BR bit.

## 1.17 NEG : Address Negative Edge Detection

### Symbol



Parameter	Data Type	Memory Area	Description
<address1>	BOOL	I, Q, M, D, L	Signal to be checked for a negative (falling) edge change.
M_BIT	BOOL	Q, M, D	The M_BIT address specifies the edge memory bit in which the previous signal state of NEG is stored. Only use the process input image I memory area for the M_BIT when no input module is already using this address.
Q	BOOL	I, Q, M, D, L	One-shot output.

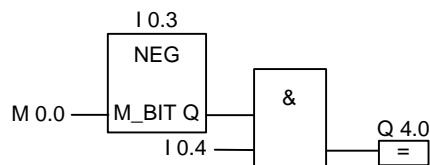
### Description

The **Address Negative Edge Detection** instruction compares the signal state of <address1> with the signal state of the previous check that is stored in the M\_BIT parameter. If a change from 1 to 0 occurred, output Q has the value 1, in all other situations it has the value 0.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	1	X	1

### Example



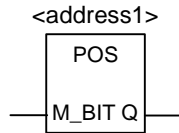
Output Q4.0 is 1 when:

- there is a falling edge at input I0.3
- AND the signal state at input I0.4 is 1.



## 1.18 POS : Address Positive Edge Detection

### Symbol



Parameter	Data Type	Memory Area	Description
<address1>	BOOL	I, Q, M, D, L	Signal to be checked for a positive (rising) edge.
M_BIT	BOOL	Q, M, D	The M_BIT address specifies the edge memory bit used to store the previous signal state of POS. You should only use the process image input area I for the M_BIT when no input module is already using this address.
Q	BOOL	I, Q, M, D, L	One-shot output.

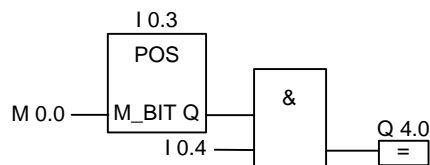
### Description

The **Address Positive Edge Detection** instruction compares the signal state of <address1> with the signal state of the previous signal check that is stored in the parameter M\_BIT. If there has been a change from 0 to 1, output Q has the value 1, in all other cases it has the value 0.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	1	X	1

### Example



Output Q4.0 is 1 when:

there is a rising edge at input I0.3

AND the signal state is 1 at input I0.4.



## 2 Comparison Instructions

### 2.1 Overview of Comparison Instructions

#### Description

IN1 and IN2 are compared according to the type of comparison you choose:

- == IN1 is equal to IN2
- <> IN1 is not equal to IN2
- > IN1 is greater than IN2
- < IN1 is less than IN2
- >= IN1 is greater than or equal to IN2
- <= IN1 is less than or equal to IN2

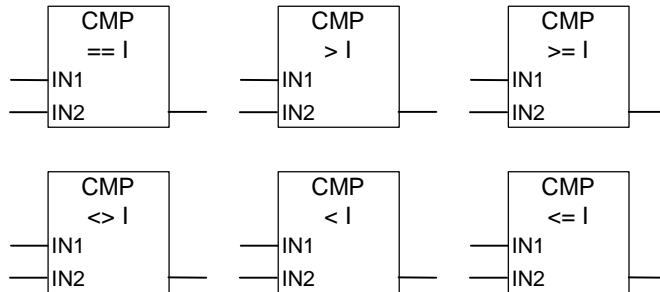
If the comparison is true, the RLO of the function is "1". Otherwise, it is 0. You cannot negate the comparison result itself, but you can achieve the same effect as negation by using the opposite compare function.

The following comparison instructions are available:

- CMP ? I : Compare Integer
- CMP ? D : Compare Double Integer
- CMP ? R : Compare Real

## 2.2 CMP ? I : Compare Integer

### Symbol



Parameter	Data Type	Memory Area	Description
IN1	INT	I, Q, M, D, L or constant	First value to compare
IN2	INT	I, Q, M, D, L or constant	Second value to compare
Box output	BOOL	I, Q, M, D, L	Result of the comparison

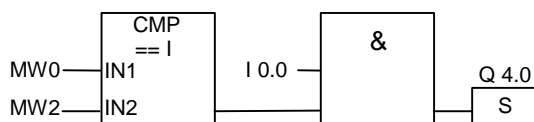
### Description

The **Compare Integer** instruction compares two values on the basis of 16-bit floating-point numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison you select from the list box.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	0	-	0	X	X	1

### Example

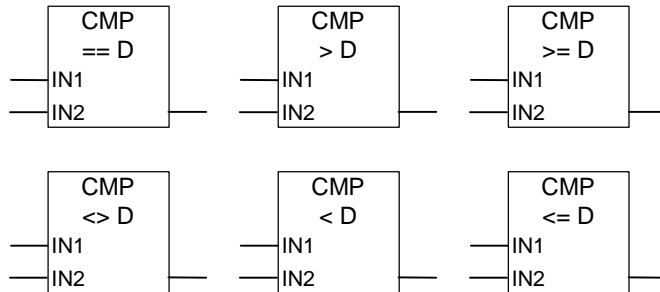


Q 4.0 is set when:

- MW0 is equal to MW2
- AND the signal state is 1 at input I0.0.

## 2.3 CMP ? D : Compare Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
IN1	DINT	I, Q, M, D, L or constant	First value to compare
IN2	DINT	I, Q, M, D, or constant L	Second value to compare
Box output	BOOL	I, Q, M, D, L	Result of the comparison

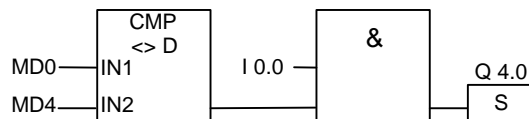
### Description

The **Compare Double Integer** instruction compares two values on the basis of 32-bit floating-point numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison you select from the list box.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	X	X	0	-	0	X	X	1

### Example

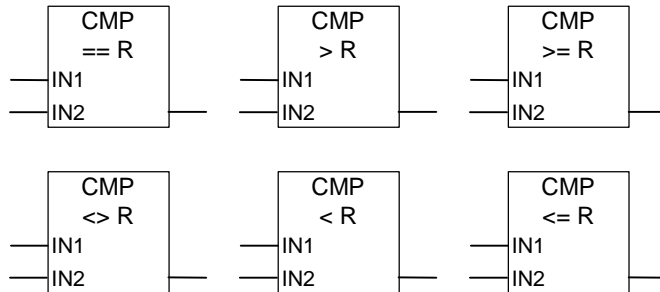


Q 4.0 is set when:

- MD0 is not equal to MD4
- AND the signal state at input I 0.0 is 1.

## 2.4 CMP ? R : Compare Real

### Symbol



Parameter	Data Type	Memory Area	Description
IN1	REAL	I, Q, M, D, L or constant	First value to compare
IN2	REAL	I, Q, M, D, L or constant	Second value to compare
Box output	BOOL	I, Q, M, D, L	Result of the comparison

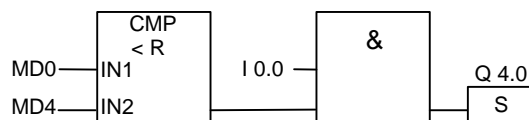
### Description

The **Compare Real** instruction compares two values on the basis of real numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison you select from the list box.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	X	X	X	X	0	X	X	1

### Example



Q 4.0 is set when:

- MD0 is less than MD4
- AND the signal state at input I 0.0 is 1.

## 3 Conversion Instructions

### 3.1 Overview of Conversion Instructions

#### Description

You can use the following instructions to convert binary coded decimal numbers and integers to other types of numbers:

- BCD\_I : BCD to Integer
- I\_BCD : Integer to BCD
- BCD\_DI : BCD to Double Integer
- I\_DI : Integer to Double Integer
- DI\_BCD : Double Integer to BCD
- DI\_R : Double Integer to Real

You can use one of the following instructions to form the complement of an integer or to invert the sign of a floating-point number:

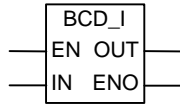
- INV\_I : Ones Complement Integer
- INV\_DI : Ones Complement Double Integer
- NEG\_I : Twos Complement Integer
- NEG\_DI : Twos Complement Double Integer
- NEG\_R : Negate Real Number

You can use any of the following instructions to convert a 32-bit IEEE floating-point number in accumulator 1 to a 32-bit integer (double integer). The individual instructions differ in their method of rounding:

- ROUND : Round to Double Integer
- TRUNC : Truncate Double Integer Part
- CEIL : Ceiling
- FLOOR : Floor

### 3.2 BCD\_I : BCD to Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	WORD	I, Q, M, D, L or constant	Number in BCD format
OUT	INT	I, Q, M, D, L	Integer value of the BCD number
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

The **BCD to Integer** instruction reads the content of the input parameter IN as a three-digit number in binary coded decimal format (BCD, " 999) and converts this number to an integer value. The output parameter OUT contains the result.

ENO always has the same signal state as EN.

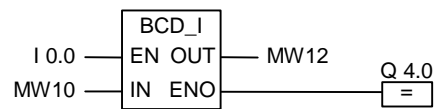
If any of the individual decimal numbers in the BCD number is in the invalid range between 10 and 15, a BCD error occurs when the conversion is attempted, causing the following reaction:

- The CPU changes to the STOP mode. "BCD Conversion Error" is entered in the diagnostic buffer with event ID number 2521.
- If OB121 is programmed, it is called.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

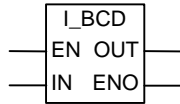


**Example**

The conversion is executed if the signal state of I0.0 is 1. The content of memory word MW10 is read as a three-digit number in BCD format and converted to an integer. The result is stored in memory word MW12. If the conversion is executed, the signal state of output Q4.0 is 1 (ENO = EN).

### 3.3 I\_BCD : Integer to BCD

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	INT	I, Q, M, D, L or constant	Integer
OUT	WORD	I, Q, M, D, L	BCD value of the integer
ENO	BOOL	I, Q, M, D, L	Enable output

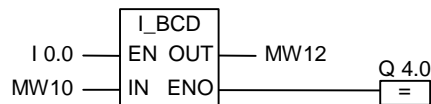
#### Description

The **Integer to BCD** instruction reads the content of the input parameter IN as an integer value and converts this value to a three-digit number in binary coded decimal format (BCD, " 999). The output parameter OUT contains the result. If an overflow occurs, ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

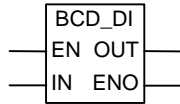
#### Example



The conversion is executed if the signal state of I0.0 is 1. The content of memory word MW10 is read as an integer and converted to a three-digit number in BCD format. The result is stored in memory word MW12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.

### 3.4 BCD\_DI : BCD to Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	DWORD	I, Q, M, D, L or constant	Number in BCD format
OUT	DINT	I, Q, M, D, L	Double integer value of the BCD number
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

The **BCD to Double Integer** instruction reads the content of the input parameter IN as a seven-digit number in binary coded decimal format (BCD, " 9,999,999) and converts this number to a double integer value. The output parameter OUT contains the result.

ENO always has the same signal state as EN.

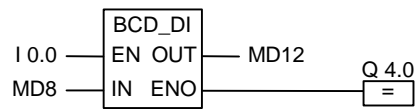
If any of the individual decimal numbers in the BCD number is in the invalid range between 10 and 15, a BCD error occurs when the conversion is attempted, causing the following reaction:

- The CPU changes to the STOP mode. "BCD Conversion Error" is entered in the diagnostic buffer with event ID number 2521.
- If OB121 is programmed, it is called.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

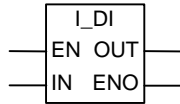
### Example



The conversion is executed if the signal state of I0.0 is 1. The content of memory double word MD8 is read as a seven-digit number in BCD format and converted to a double integer. The result is stored in MD12. If the conversion is executed, the signal state of output Q4.0 is 1 (ENO = EN).

### 3.5 I\_DI : Integer to Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	INT	I, Q, M, D, L or constant	Value to be converted
OUT	DINT	I, Q, M, D, L	Result
ENO	BOOL	I, Q, M, D, L	Enable output

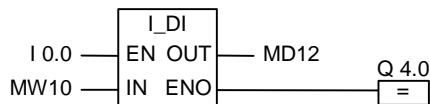
#### Description

The **Integer to Double Integer** instruction reads the content of the input parameter IN as an integer and converts the integer to a double integer. The output parameter OUT contains the result. ENO always has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

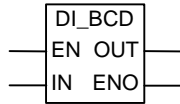
#### Example



The conversion is executed if the signal state of I0.0 is 1. The content of memory word MW10 is read as an integer and converted to a double integer. The result is stored in memory double word MD12. If the conversion is executed, the signal state of output Q4.0 is 1 (ENO = EN).

### 3.6 DI\_BCD : Double Integer to BCD

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	DINT	I, Q, M, D, L or constant	Double Integer
OUT	DWORD	I, Q, M, D, L	BCD value of the double integer
ENO	BOOL	I, Q, M, D, L	Enable output

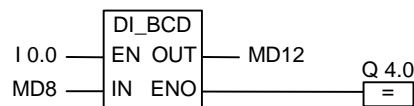
#### Description

The **Double Integer to BCD** instruction reads the content of the input parameter IN as a double integer value and converts this value to a seven-digit number in BCD format (" 9 999 999). The output parameter OUT contains the result. If an overflow occurs, ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

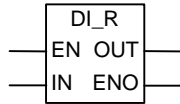
#### Example



The conversion is executed if the signal state of I0.0 is 1. The content of memory double word MD8 is read as a double integer and converted to a seven-digit number in BCD format. The result is stored in MD12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.

### 3.7 DI\_R : Double Integer to Real

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	DINT	I, Q, M, D, L or constant	Value to be converted
OUT	REAL	I, Q, M, D, L	Result
ENO	BOOL	I, Q, M, D, L	Enable output

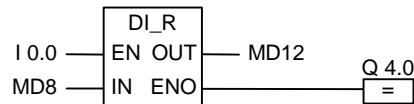
#### Description

The **Double Integer to Real** instruction reads the content of the input parameter IN as a double integer value and converts this value to a real number. The output parameter OUT contains the result. ENO always has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

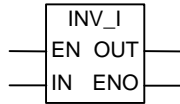
#### Example



The conversion is executed if the signal state of I0.0 is 1. The contents of memory double word MD8 is read as an integer and converted to a real number. The result is stored in memory double word MD12. If the conversion is not executed, the signal state of output Q4.0 is 0 (ENO=EN).

### 3.8 INV\_I : Ones Complement Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	INT	I, Q, M, D, L or constant	Input value
OUT	INT	I, Q, M, D, L	Ones complement of the integer
ENO	BOOL	I, Q, M, D, L	Enable output

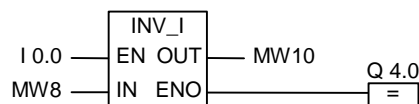
#### Description

The **Ones Complement Integer** instruction reads the content of the input parameter IN and performs the Boolean word logic instruction Exclusive Or Word masked by FFFFH, so that the value of every bit is inverted. The output parameter OUT contains the result. ENO always has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

#### Example



The conversion is executed if the signal state of I0.0 is 1. The value of every bit in MW8 is inverted:

MW8 = 01000001 10000001 →

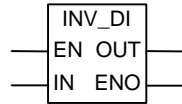
MW10 = 10111110 01111110

The conversion is not executed when the signal state of I0.0 is 0 and Q4.0 is 0 (ENO = EN).



### 3.9 INV\_DI : Ones Complement Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	DINT	I, Q, M, D, L or constant	Input value
OUT	DINT	I, Q, M, D, L	Ones complement of the double integer
ENO	BOOL	I, Q, M, D, L	Enable output

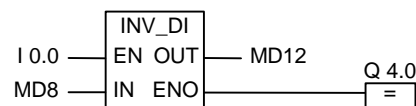
#### Description

The **Ones Complement Double Integer** instruction reads the content of the input parameter IN and performs the Boolean word logic operation Exclusive Or Word masked by FFFF FFFFH, so that the value of every bit is inverted. The output parameter OUT contains the result. ENO always has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	-	-	-	-	0	1	1	1

#### Example



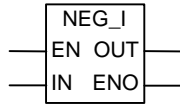
The conversion is executed if the signal state of I0.0 is 1. The value of every bit of memory double word MD8 is inverted:

MD8 = F0FF FFF0 → MD12 = 0F00 000F

If the conversion is not executed, Q4.0 is 0 (ENO = EN).

### 3.10 NEG\_I : Twos Complement Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	INT	I, Q, M, D, L or constant	Input value
OUT	INT	I, Q, M, D, L	Twos complement of the integer
ENO	BOOL	I, Q, M, D, L	Enable output

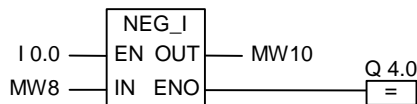
#### Description

The **Twos Complement Integer** instruction reads the content of the input parameter IN and changes the sign (for example, from a positive value to a negative value). The output parameter OUT contains the result. The signal state of EN is and ENO is always the same except when the signal state of EN is 1 and an overflow occurs. In this case, the signal state of ENO is 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

#### Example



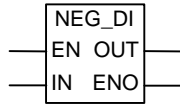
The conversion is executed if the signal state of I0.0 is 1. The value of memory word MW8 is output at O to memory word MW10 with the opposite sign:

$$MW8 = +10 \rightarrow MW10 = -10$$

If the signal state of EN is 1 and an overflow occurs, ENO is 0 and the signal state of Q4.0 is 0. If the conversion is not executed, Q4.0 is 0 (ENO = EN).

### 3.11 NEG\_DI : Twos Complement Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	DINT	I, Q, M, D, L or constant	Input value
OUT	DINT	I, Q, M, D, L	Twos complement of the double integer
ENO	BOOL	I, Q, M, D, L	Enable output

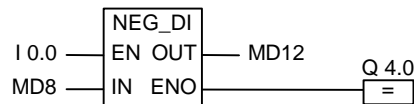
#### Description

The **Twos Complement Double Integer** instruction reads the content of the input parameter IN and changes the sign (for example, from a positive value to a negative value). The output parameter OUT contains the result. The signal state of EN is and ENO is always the same except when the signal state of EN is 1 and an overflow occurs. In this case, the signal state of ENO is 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

#### Example



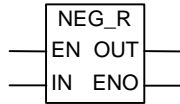
The conversion is executed if the signal state of I0.0 is 1. The value of memory double word MD8 is output at O to memory double word MD10 with the opposite sign.

$$MW8 = +10 \rightarrow MW10 = -10$$

If the signal state of EN is 1 and an overflow occurs, ENO is 0 and the signal state of Q4.0 is 0. If the conversion is not executed, Q4.0 is 0 (ENO = EN).

### 3.12 NEG\_R : Negate Real Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Input value
OUT	REAL	I, Q, M, D, L	The result is the negated input value.
ENO	BOOL	I, Q, M, D, L	Enable output

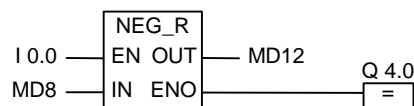
#### Description

The **Negate Real Number** instruction reads the content of the input parameter IN and inverts the sign bit (the instruction changes the sign of the number. for example, from 0 for plus to 1 for minus). The bits of the exponent and mantissa remain the same. The output parameter OUT provides the result. ENO always has the same signal state as EN except when the signal state of EN is 1 and an overflow occurs. In this case, the signal state of ENO is 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	-	-	0	X	X	1

#### Example



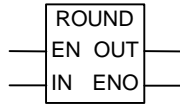
The conversion is executed if the signal state of I0.0 is 1. The value of memory double word MD8 is output at O to memory double word MD12 with the opposite sign as shown in the following example:

MD8 = + 6.234 → MD12 = - 6.234

If the conversion is not executed, the signal state of output Q4.0 is 0 (ENO = EN).

### 3.13 ROUND : Round to Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Value to be rounded
OUT	DINT	I, Q, M, D, L	IN rounded to the next double integer
ENO	BOOL	I, Q, M, D, L	Enable output

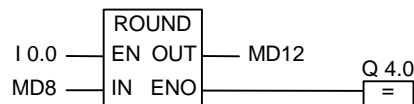
#### Description

The **Round to Double Integer** instruction reads the content of the input parameter IN as a real number and converts this number to a double integer. The result is the nearest integer and is contained in output parameter OUT. If the fraction is x.5, the number is rounded to the even number (for example: 2.5 -> 2, 1.5 -> 2). If an overflow occurs, ENO is set to 0. If the input value is not a real number, the OV bit and the OS bit have the value 1 and ENO has the value 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

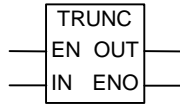
#### Example



The conversion is executed if I0.0 is 1. The content of memory double word MD8 is read as a real number and converted to a double integer. The result of this round-to-nearest function is stored in memory double word MD12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.

### 3.14 TRUNC : Truncate Double Integer Part

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Value to be truncated
OUT	DINT	I, Q, M, D, L	Integer component of IN
ENO	BOOL	I, Q, M, D, L	Enable output

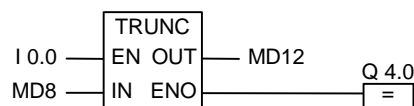
#### Description

The **Truncate Double Integer Part** instruction reads the content of the input parameter IN as a real number and converts this number to a double integer (for example 1.5 becomes 1). The result is the integer component of the real number). The output parameter OUT contains the result. If an overflow occurs, ENO is set to 0. If the input value is not a real number, the OV bit and the OS bit have the value 1 and ENO has the value 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

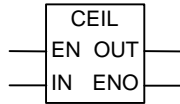
#### Example



The conversion is executed if the signal state of I0.0 is 1. The content of memory double word MD8 is read as a real number and converted to a double integer according to the "round to zero principle". The integer component is the result and is stored in memory double word MD12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.

### 3.15 CEIL : Ceiling

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Value to be converted
OUT	DINT	I, Q, M, D, L	Result
ENO	BOOL	I, Q, M, D, L	Enable output

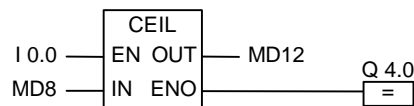
#### Description

The **Ceiling** instruction reads the content of the input parameter IN as a real number and converts this number to a double integer (for example: +1.2 -> +2; -1.5 -> -1). The result is the lowest integer which is greater than or equal to the specified real number. The output parameter OUT contains the result. If an overflow occurs, ENO is 0. If the input value is not a real number, the OV bit and the OS bit have the value 1 and ENO has the value 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

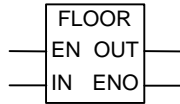
#### Example



The conversion is executed if I0.0 is 1. The content of memory double word MD8 is read as a real number and converted to a double integer by rounding to the next higher (or equal) whole number. The result is stored in memory double word MD12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.

### 3.16 FLOOR : Floor

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Value to be converted
OUT	DINT	I, Q, M, D, L	Result
ENO	BOOL	I, Q, M, D, L	Enable output

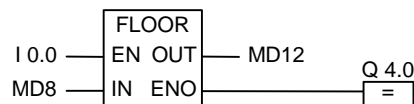
#### Description

The **Floor** instruction reads the content of the input parameter IN as a real number and converts this number to a double integer. The result is the highest integer which is lower than or equal to the specified real number. The output parameter OUT contains the result. If an overflow occurs, ENO is set to 0. If the input value is not a real number, the OV bit and the OS bit have the value 1 and ENO has the value 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	X	X	0	X	X	1

#### Example



The conversion is executed if I0.0 is 1. The content of memory double word MD8 is read as a real number and converted to a double integer by rounding to the next lower (or equal) whole number. The result is stored in memory double word MD12. If an overflow occurs, the signal state of output Q4.0 is 0. If the signal state at input EN is 0 (meaning that the conversion is not executed), the signal state of output Q4.0 is also 0.



## 4 Counter Instructions

### 4.1 Overview of Counter Instructions

#### Area in Memory

Counters have an area reserved for them in the memory of your CPU. This memory area reserves one 16-bit word for each counter address. When you program in FBD, 256 counters are supported. The counter instructions are the only functions that have access to the counter memory area.

#### Count Value

Bits 0 through 9 of the counter word contain the count value in binary code. The count value is moved to the counter word when a counter is set. The range of the count value is 0 to 999.

You can vary the count value within this range by using the following counter instructions:

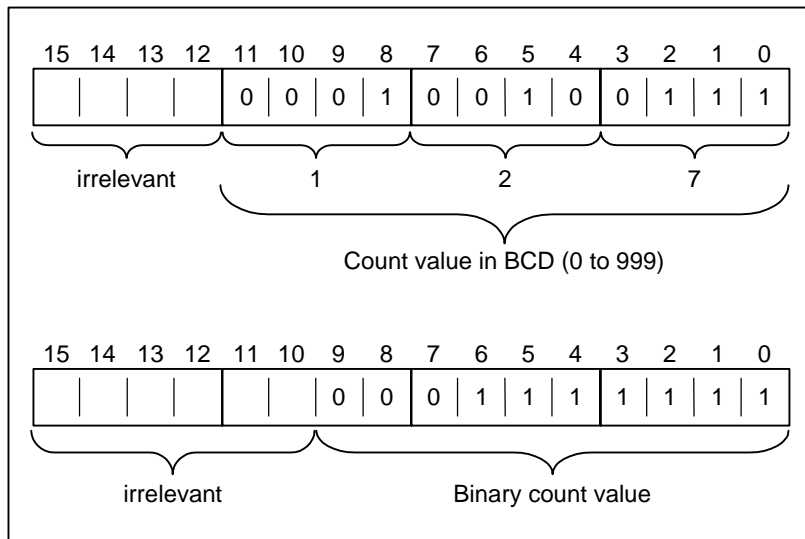
- S\_CUD : Assign Parameters and Count Up/Down
- S\_CU : Assign Parameters and Count Up
- S\_CD : Assign Parameters and Count Down
- SC : Set Counter Value
- CU : Up Counter
- CD : Down Counter

### Bit Configuration in the Counter

You provide a counter with a preset value by entering a number from 0 to 999, for example 127, in the following format: C#127. The C# stands for binary coded decimal format (BCD format: each set of four bits contains the binary code for one decimal value).

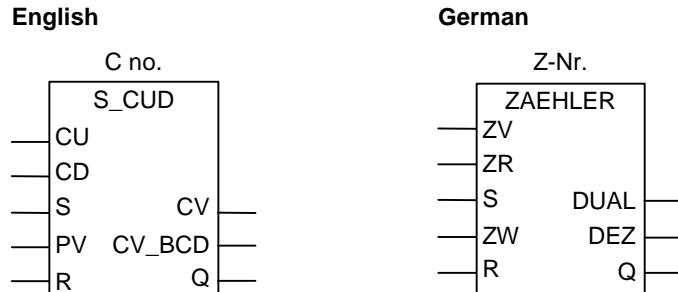
Bits 0 through 11 of the counter contain the count value in binary coded decimal format.

The following figure shows the contents of the counter after you have loaded the count value 127, and the contents of the counter cell after the counter has been set.



## 4.2 S\_CUD : Assign Parameters and Count Up/Down

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	COUNTER	C	Counter identification number. The range depends on the CPU.
CU	ZV	BOOL	I, Q, M, D, L	ZV input: Up Counter
CD	ZR	BOOL	I, Q, M, D, L	ZR input: Down Counter
S	S	BOOL	I, Q, M, D, L, T, C	Input for presetting the counter
PV	ZW	WORD	I, Q, M, D, L or constant	Count value in the range between 0 and 999 or Count value entered as C#<value> in BCD format
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
CV	DUAL	WORD	I, Q, M, D, L	Current count value (hexadecimal number)
CV_BCD	DEZ	WORD	I, Q, M, D, L	Current count value (BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the counter

### Description

A rising edge (change in signal state from 0 to 1) at input S of the **Assign Parameters and Count Up/Down** instruction sets the counter with the value at the Preset Value (PV) input. The counter is incremented by 1 if the signal state at input CU changes from 0 to 1 (rising edge) and the value of the counter is less than 999. The counter is decremented by 1 if the signal state at input CD changes from 0 to 1 (rising edge) and the value of the counter is higher than 0. If there is a rising edge at both count inputs, both operations are executed and the count remains the same.

If the counter is set and if RLO = 1 at the inputs CU/CD, the counter will count accordingly in the next scan cycle, even if there was no change from a positive to a negative edge or vice versa.

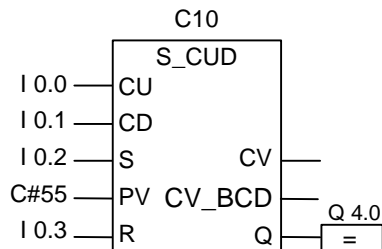
The counter is reset if the signal on input R is "1". Resetting the counter sets the count value to 0.

A signal state check for 1 at output Q produces a result of 1 when the count is greater than 0; the check produces a result of 0 when the count is equal to 0.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example



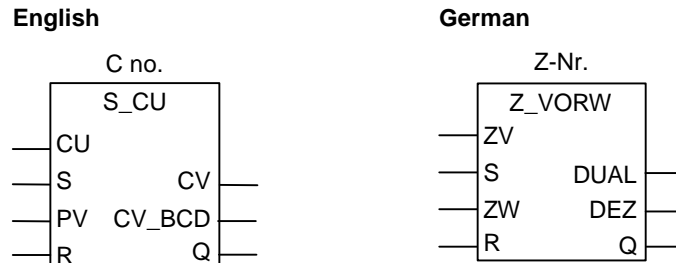
A change in signal state from 0 to 1 at input I0.2 sets counter C10 with the value 55. If the signal state of input I0.0 changes from 0 to 1, the value of counter C10 is incremented by 1, except when the value of counter C10 is already 999. If input I0.1 changes from 0 to 1, counter C10 is decremented by 1, except when the value of counter C10 is already 0. If I0.3 changes from 0 to 1, the value of C10 is set to 0. Q4.0 is 1, when C 10 is not equal to 0.

### Note

Avoid to use a counter at several program points (risk of counting errors).

## 4.3 S\_CU : Assign Parameters and Count Up

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	COUNTER	C	Counter identification number. The range depends on the CPU.
CU	ZV	BOOL	I, Q, M, D, L	ZV input: Up Counter
S	S	BOOL	I, Q, M, D, L, T, C	Input for presetting the counter
PV	ZW	WORD	I, Q, M, D, L or constant	Count value in the range between 0 and 999 or Count value entered as C#<value> in BCD format
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
CV	DUAL	WORD	I, Q, M, D, L	Current count value (hexadecimal number)
CV_BCD	DEZ	WORD	I, Q, M, D, L	Current count value (BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the counter

### Description

A rising edge (change in signal state from 0 to 1) at input S of the **Assign Parameters and Count Up** instruction sets the counter with the value at the Preset Value (PV) input. With a rising edge at input CU, the count value is incremented by 1 when the count value is less than 999.

If the counter is set and if RLO = 1 at the inputs CU, the counter will count accordingly in the next scan cycle, even if there was no change from a positive to a negative edge or vice versa.

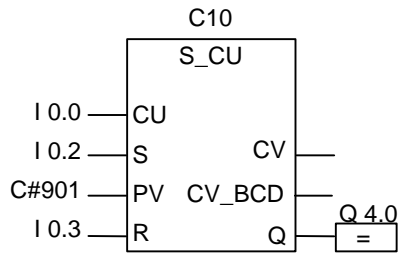
The counter is reset if the signal on input R is "1". Resetting the counter sets the count value to 0.

A signal state check for 1 at output Q produces a result of 1 when the count is greater than 0. The check produces a result of 0 when the count is equal to 0.

**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

**Example**



A change in signal state from 0 to 1 at input I0.2 sets counter C10 with the value 901. If the signal state of I0.0 changes from 0 to 1, the value of counter C10 is incremented by 1, unless the value of C10 is equal to 999. If I0.3 changes from 0 to 1, the value of C10 is set to 0. The signal state of output Q4.0 is 1 if C10 is not equal to 0.

---

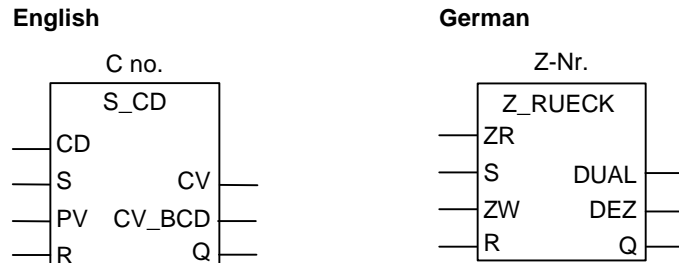
**Note**

Avoid to use a counter at several program points (risk of counting errors).

---

## 4.4 S\_CD : Assign Parameters and Count Down

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	COUNTER	C	Counter identification number. The range depends on the CPU.
CD	ZR	BOOL	I, Q, M, D, L	CD input: Down Counter
S	S	BOOL	I, Q, M, D, L, T, C	Input for presetting the counter
PV	ZW	WORD	I, Q, M, D, L or constant	Count value in the range between 0 and 999 or Count value entered as C#<value> in BCD format
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
CV	DUAL	WORD	I, Q, M, D, L	Current count value (hexadecimal number)
CV_BCD	DEZ	WORD	I, Q, M, D, L	Current count value (BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the counter

### Description

A rising edge (change in signal state from 0 to 1) at input S of the **Assign Parameters and Count Down** instruction sets the counter with the value at the Preset Value input (PV). With a rising edge at input CD, the counter is decremented by 1 when the count value is greater than 0.

If the counter is set and if RLO = 1 at the inputs CD, the counter will count accordingly in the next scan cycle, even if there was no change from a positive to a negative edge or vice versa.

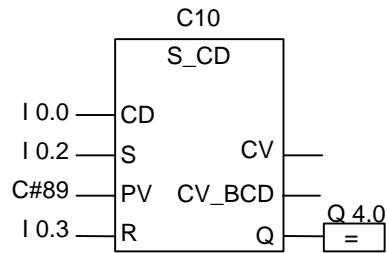
The counter is reset if the signal on input R is "1". Resetting the counter sets the count value to 0.

A signal state check for 1 at output Q produces a result of 1 when the count is greater than 0; the check produces a result of 0 when the count is equal to 0.

**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

**Example**



A change in signal state from 0 to 1 at input I0.2 sets counter C10 with the value 89. If the signal state of input I0.0 changes from 0 to 1, the value of counter C10 is decreased by 1, unless the value of counter C10 is equal to 0. The signal state of output Q4.0 is 1 if counter C10 is not equal to 0. If I0.3 changes from 0 to 1, the value of C10 is set to 0.

---

**Note**

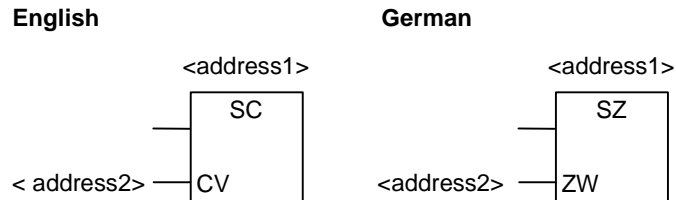
Avoid to use a counter at several program points (risk of counting errors).

---



## 4.5 SC : Set Counter Value

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
Counter no.	Counter no	COUNTER	C	The address1 specifies the number of the counter which is to be preset with a value.
CV	ZW	WORD	I, Q, M, D, L or constant	The value to be preset (address2) can lie between 0 and 999. When you enter a constant, C# must stand in front of the value specified by the BCD format, for example, C#100.

### Description

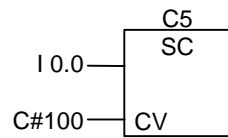
The **Set Counter Value** instruction assigns a preset value to the counter you specified. This instruction is only executed if there is a rising edge (change in signal state from 0 to 1) in the RLO.

You can only place the box **Set Counter Value** at the right-hand end of a logic string. You can use a number of **Set Counter Value** boxes.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example



The counter C5 is preset with the value 100 if the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO). C# indicates that you are entering a value in BCD format.

If there is no rising edge, the counter value of C5 is not changed.

## 4.6 CU : Up Counter

### Symbol



Parameter	Data Type	Memory Area	Description
Counter no.	COUNTER	C	The address specifies the number of the counter which is to be incremented.

### Description

The **Up Counter** instruction increments the value of a specified counter by 1 if there is a rising edge (change in signal state from 0 to 1) in the RLO and the value of the counter is less than 999. If there is no rising edge in the RLO or if the counter already has the value 999, it is not incremented.

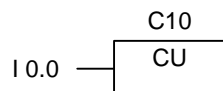
The **Set Counter Value** instruction sets the value of the counter.

You can only place the box **Up Counter** at the right-hand end of a logic string. You can use a number of **Up Counter** boxes.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example



If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the value of the counter C10 is incremented by 1 (unless the value of C10 is already 999).

If there is no rising edge, the value of C10 remains unchanged.

## 4.7 CD : Down Counter

### Symbol



Parameter	Data Type	Memory Area	Description
Counter no.	COUNTER	C	The address specifies the number of the counter which is to be decremented.

### Description

The **Down Counter** instruction decrements the value of a specified counter by 1 if there is a rising edge (change in signal state from 0 to 1) in the RLO and the value of the counter is greater than 0. If there is no rising edge in the RLO or if the counter already has the value 0, it is not decremented.

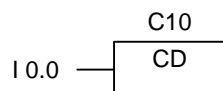
The **Set Counter Value** instruction sets the value of the counter.

You can only place the box **Down Counter** at the right-hand end of a logic string. You can use a number of **Down Counter** boxes.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example



If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the value of the counter C10 is decremented by 1 (unless the value of C10 is already 0).

If there is no rising edge, the value of C10 remains unchanged.

# 5 Data Block Instructions

## 5.1 OPN : Open Data Block

### Symbol

<DB-Number> or  
<DI-Number>



Parameter	Data Type	Memory Area	Description
Number of the DB or DI	BLOCK_DB	-	Number of the DB or DI; Range depends on the CPU.

### Description

You can use the **Open Data Block** instruction to open an existing data block as a shared data block (DB) or instance data block (DI). The number of the data block is transferred to the DB or DI register. The subsequent DB and DI commands access the corresponding blocks depending on the register contents.

### Status Word

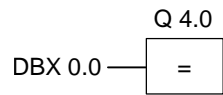
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	-	-	-

## Example

Network 1



Network 2



DB10 is the currently opened data block. The scan at DBX0.0 therefore refers to bit 0 of data byte 0 of data block DB10. The signal state of this bit is assigned to output Q 4.0.

# 6 Jump Instructions

## 6.1 Overview of Jump Instructions

### Description

You can use this instruction in all logic blocks, for example in organization blocks (OBs), function blocks (FBs) and functions (FCs).

The following Jump instructions are available:

- JMP Unconditional Jump in a Block
- JMP Conditional Jump in a Block
- JMPN Jump-If-Not

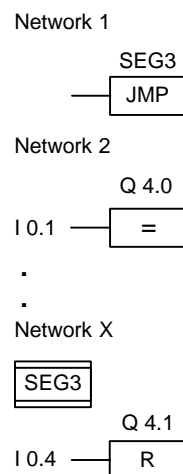
### Jump Label as Address

The address of a Jump instruction is a label. The jump label indicates the destination to which you want the program to jump.

You enter the label above the JMP box. A label consists of a maximum of four characters. The first character must be a letter; the other characters can be letters or numbers (for example, SEG3).

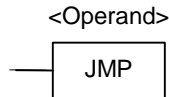
### Jump Label as Destination

The destination label must be at the beginning of a network. You enter the destination label at the beginning of the network by selecting LABEL from the FBD list box. An empty box appears. In the box, you type the name of the label.



## 6.2 JMP : Unconditional Jump in a Block

### Symbol



Parameter	Data Type	Memory Area	Description
Name of a jump label	-	-	The address specifies the label to which the program will jump unconditionally.

### Description

The **Unconditional Jump in a Block** instruction corresponds to a "go to label" instruction. None of the instructions between the jump operation and the label is executed.

You can use this instruction in all logic blocks, for example in organization blocks (OBs), function blocks (FBs) and functions (FCs).

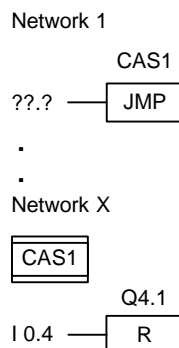
There must not be any logic operation before the **Unconditional Jump in a Block** box.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	-	-	-

The instruction does not change the bits in the status word.

### Example

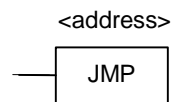


The jump is always executed. None of the instructions between the jump instruction and the label is executed.



## 6.3 JMP : Conditional Jump in a Block

### Symbol



Parameter	Data Type	Memory Area	Description
Name of a jump label	-	-	The address specifies the label to which the program will jump if the RLO is 1.

### Description

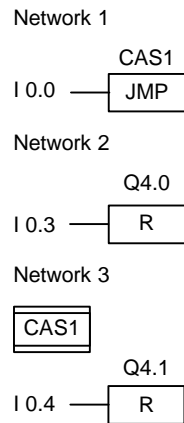
The **Conditional Jump in a Block** instruction corresponds to a "go to label" instruction if the RLO is 1. The FBD element "Unconditional Jump" is also used for this operation, however it is made conditional by the preceding logic operation. The conditional jump is only executed when the result of this logic operation is 1. None of the instructions between the jump operation and the label is executed.

You can use this instruction in all logic blocks, for example in organization blocks (OBs), function blocks (FBs) and functions (FCs).

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	1	1	0

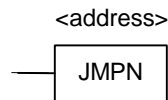
### Example



If the signal state of input I0.0 is 1, the jump to label CAS1 is executed. The instruction to reset output Q4.0 is not executed, even if the signal state of input I0.3 is 1.

## 6.4 JMPN : Jump-If-Not

### Symbol



Parameter	Data Type	Memory Area	Description
Name of a jump label	-	-	The address specifies the label to which the program will jump if the RLO is 0.

### Description

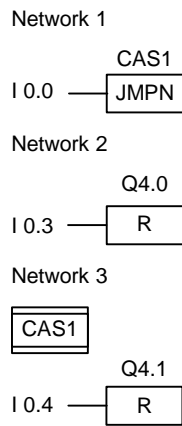
The **Jump-If-Not** instruction corresponds to a "go to label" instruction that is executed if the RLO is 0.

You can use this instruction in all logic blocks, for example in organization blocks (OBs), function blocks (FBs) and functions (FCs).

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	1	1	0

### Example

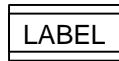


If the signal state of input I0.0 is 0, the jump to label CAS1 is executed. The instruction to reset output Q4.0 is not executed, even if the signal state of input I0.3 is 1.

None of the instructions between the jump operation and the label is executed.

## 6.5 LABEL : Jump Label

### Symbol

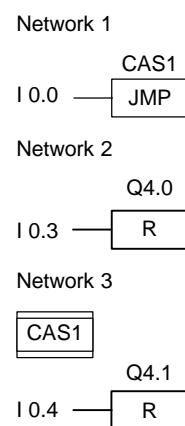


### Description

The jump label is the identifier for the destination of a jump instruction. A label consists of a maximum of four characters. The first character must be a letter; the other characters can be letters or numbers, for example CAS1.

A jump label must exist for every jump or jump-if-not instruction (**JMP** or **JMPN**).

### Example



If I0.0 = 1, the jump to label CAS1 is executed.

Due to the jump, the operation "Reset output" at Q 4.0 is not executed even if I0.3 = 1.



# 7 Integer Math Instructions

## 7.1 Overview of Integer Math Instructions

### Description

Using integer math, you can carry out the following operations with **two integer numbers** (16 and 32 bits):

- ADD\_I : Add Integer
- SUB\_I : Subtract Integer
- MUL\_I : Multiply Integer
- DIV\_I : Divide Integer
- ADD\_DI : Add Double Integer
- SUB\_DI : Subtract Double Integer
- MUL\_DI : Multiply Double Integer
- DIV\_DI : Divide Double Integer
- MOD\_DI : Return Fraction Double Integer

## 7.2 Evaluating the Bits of the Status Word with Integer Math Instructions

### Description

The integer math instructions affect the following bits in the status word: CC1 and CC0, OV and OS.

The following tables show the signal state of the bits in the status word for the results of instructions with Integers (16 and 32 bits):

Valid Range for the Result	CC 1	CC 0	OV	OS
0 (zero)	0	0	0	*
16 bits: $-32\,768 \leq \text{result} < 0$ (negative number) 32 bits: $-2\,147\,483\,648 \leq \text{result} < 0$ (negative number)	0	1	0	*
16 bits: $32\,767 \geq \text{result} > 0$ (positive number) 32 bits: $2\,147\,483\,647 \geq \text{result} > 0$ (positive number)	1	0	0	*

\* The OS bit is not affected by the result of the instruction.

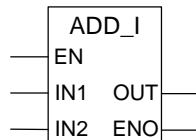
Invalid Range for the Result	A1	A0	OV	OS
Underflow (addition) 16 bits: result = -65536 32 bits: result = -4 294 967 296	0	0	1	1
Underflow (multiplication) 16 bits: result < -32 768 (negative number) 32 bits: result < -2 147 483 648 (negative number)	0	1	1	1
Overflow (addition, subtraction) 16 bits: result > 32 767 (positive number) 32 bits: result > 2 147 483 647 (positive number)	0	1	1	1
Overflow (multiplication, division) 16 bits: result > 32 767 (positive number) 32 bits: result > 2 147 483 647 (positive number)	1	0	1	1
Underflow (addition, subtraction) 16 bits: result < -32. 768 (negative number) 32 bits: result < -2 147 483 648 (negative number)	1	0	1	1
Division by 0	1	1	1	1

Operation	A1	A0	OV	OS
+D: result = -4 294 967 296	0	0	1	1
/D or MOD: division by 0	1	1	1	1



## 7.3 ADD\_I : Add Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	INT	I, Q, M, D, L or constant	First value for addition
IN2	INT	I, Q, M, D, L or constant	Second value for addition
OUT	INT	I, Q, M, D, L	Result of addition
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

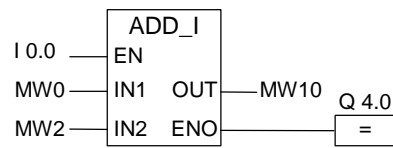
A signal state of 1 at the Enable (EN) input activates the **Add Integer** instruction. This instruction adds inputs IN1 and IN2. The result can be scanned at OUT. If the result is outside the permissible range for an integer, the OV and OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

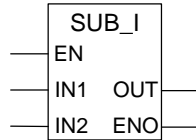
### Example



A signal state of 1 at input I0.0 activates the ADD\_I box. The result of the addition MW0 + MW2 is entered in memory word MW10. If the result is outside the permitted range for an integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.4 SUB\_I : Subtract Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	INT	I, Q, M, D, L or constant	Minuend (value from which second value is subtracted)
IN2	INT	I, Q, M, D, L or constant	Subtrahend (value subtracted from the first value)
OUT	INT	I, Q, M, D, L	Result of subtraction
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

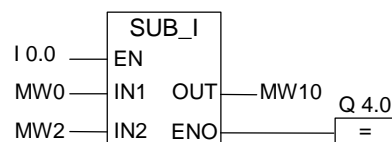
A signal state of 1 at the Enable (EN) input activates the **Subtract Integer** instruction. This instruction subtracts input IN2 from IN1. The result can be scanned at OUT. If the result is outside the permitted range for an integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

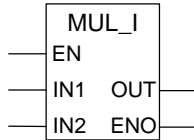
### Example



A signal state of 1 at input I0.0 activates the SUB\_I box. The result of the subtraction MW0 - MW2 is entered in memory word MW10. If the result is outside the permitted range for an integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.5 MUL\_I : Multiply Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	INT	I, Q, M, D, L or constant	Multiplicand (value that is multiplied by the second value)
IN2	INT	I, Q, M, D, L or constant	Multiplier (value by which the first value is multiplied)
OUT	INT	I, Q, M, D, L	Result of the multiplication
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

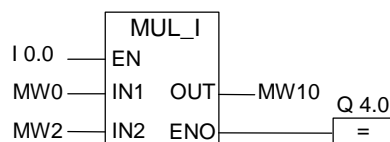
A signal state of 1 at the Enable (EN) input activates the **Multiply Integer** instruction. This instruction multiplies input IN1 by IN2. The result is a 32-bit integer that can be scanned at OUT. If the result is outside the permitted range for a 16-bit integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

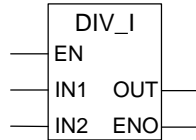
### Example



A signal state of 1 at input I0.0 activates the MUL\_I box. The result of the multiplication MW0 x MW2 is entered in memory word MW10. If the result is outside the permitted range for a 16-bit integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.6 DIV\_I : Divide Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	INT	I, Q, M, D, L or constant	Dividend
IN2	INT	I, Q, M, D, L or constant	Divisor
OUT	INT	I, Q, M, D, L	Result of division
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

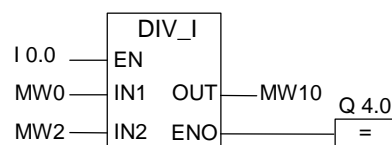
A signal state of 1 at the Enable (EN) input activates the **Divide Integer** instruction. This instruction divides input IN1 by IN2. The integer quotient (truncated result) can be scanned at OUT. The remainder cannot be scanned. If the quotient is outside the permitted range for an integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

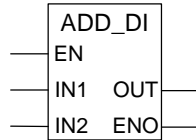
### Example



A signal state of 1 at input I0.0 activates the DIV\_I box. The quotient of dividing MW0 by MW2 is entered in memory word MW10. If the quotient is outside the permitted range for an integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.7 ADD\_DI : Add Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DINT	I, Q, M, D, L or constant	First value for addition
IN2	DINT	I, Q, M, D, L or constant	Second value for addition
OUT	DINT	I, Q, M, D, L	Result of addition
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

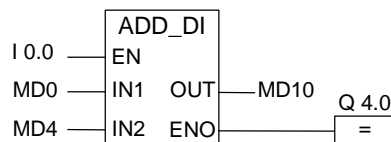
A signal state of 1 at the Enable (EN) input activates the **Add Double Integer** instruction. This instruction adds inputs IN1 and IN2. The result can be scanned at OUT. If the result is outside the permissible range for a double integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

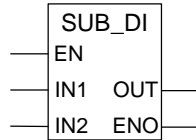
### Example



A signal state of 1 at input I0.0 activates the ADD\_DI box. The result of the addition MD0 + MD4 is entered in memory double word MD10. If the result is outside the permitted range for a double integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.8 SUB\_DI : Subtract Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DINT	I, Q, M, D, L or constant	Minuend (value from which second value is subtracted)
IN2	DINT	I, Q, M, D, L or constant	Subtrahend (value subtracted from the first value)
OUT	DINT	I, Q, M, D, L	Result of subtraction
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

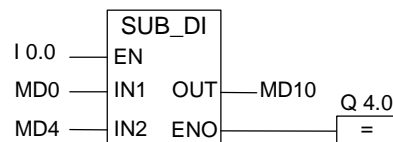
A signal state of 1 at the Enable (EN) input activates the **Subtract Double Integer** instruction. This instruction subtracts input IN2 from IN1. The result can be scanned at OUT. If the result is outside the permitted range for a double integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

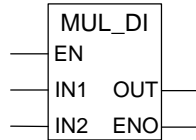
### Example



A signal state of 1 at input I0.0 activates the SUB\_DI box. The result of the subtraction MD0 - MD4 is entered in memory double word MD10. If the result is outside the permitted range for a double integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.9 MUL\_DI : Multiply Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DINT	I, Q, M, D, L or constant	Multiplicand (value that is multiplied by the second value)
IN2	DINT	I, Q, M, D, L or constant	Multiplier (value by which the first value is multiplied)
OUT	DINT	I, Q, M, D, L	Result of the multiplication
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

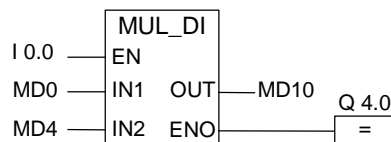
A signal state of 1 at the Enable (EN) input activates the **Multiply Double Integer** instruction. This instruction multiplies inputs IN1 and IN2. The result can be scanned at OUT. If the result is outside the permitted range for a double integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

### Example

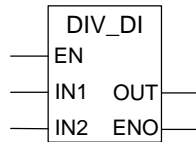


A signal state of 1 at input I0.0 activates the MUL\_DI box. The result of the multiplication MD0 x MD4 is entered in memory double word MD10. If the result is outside the permitted range for a double integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.



## 7.10 DIV\_DI : Divide Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DINT	I, Q, M, D, L or constant	Dividend
IN2	DINT	I, Q, M, D, L or constant	Divisor
OUT	DINT	I, Q, M, D, L	Result of division
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

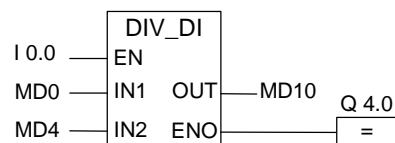
A signal state of 1 at the Enable (EN) input activates the **Divide Double Integer** instruction. This instruction divides input IN1 by IN2. The quotient (truncated result) can be scanned at OUT. The Divide Double Integer instruction stores the quotient as a single 32-bit value in DINT format. This instruction does not produce a remainder. If the quotient is outside the permitted range for a double integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

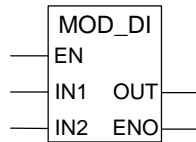
### Example



A signal state of 1 at input I0.0 activates the DIV\_DI box. The quotient of dividing MD0 by MD4 is entered in memory double word MD10. If the quotient is outside the permitted range for a double integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

## 7.11 MOD\_DI : Return Fraction Double Integer

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DINT	I, Q, M, D, L or constant	Dividend
IN2	DINT	I, Q, M, D, L or constant	Divisor
OUT	DINT	I, Q, M, D, L	Remainder
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

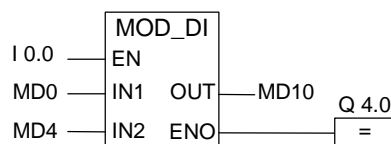
A signal state of 1 at the Enable (EN) input activates the **Return Fraction Double Integer** instruction. This instruction divides input IN1 by IN2. The remainder (fraction) can be scanned at OUT. If the result is outside the permitted range for a double integer, the OV and the OS bit of the status word are 1 and the ENO is 0.

See also Evaluating the Bits of the Status Word with Integer Math Instructions.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

### Example



A signal state of 1 at input I0.0 activates the MOD\_DI box. The remainder (fraction) of dividing MD0 by MD4 is stored in memory double word MD10. If the result is outside the permitted range for a double integer or the signal state of input I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

# 8 Floating-Point Math Instructions

## 8.1 Overview of Floating-Point Math

### Description

The IEEE 32-bit floating-point numbers belong to the data type called REAL. You can use the floating-point math instructions to perform the following math instructions using **two** 32-bit IEEE floating-point numbers:

- ADD\_R : Add Real
- SUB\_R : Subtract Real
- MUL\_R : Multiply Real
- DIV\_R : Divide Real

You can carry out the following operations with **one** 32-bit IEEE floating-point number:

- Form the absolute value (ABS) of a floating-point number
- Form the square (SQR) or square root (SQRT) of a floating-point number
- Form the natural logarithm (LN) of a floating-point number
- Form the exponential value of a floating-point number (EXP) to base e (= 2.71828...)
- Form the following trigonometric functions of an angle, represented as a 32-bit floating-point number:
  - sine (SIN) and arc sine (ASIN)
  - cosine (COS) and arc cosine (ACOS)
  - tangent (TAN) and arc tangent (ATAN)

## 8.2 Evaluating the Bits of the Status Word with Floating-Point Instructions

### Description

Floating-point instructions affect the following bits in the status word: CC 1 and CC 0, OV and OS.

The following tables show the signal state of the bits in the status word for the results of instructions with floating-point numbers (32 bits):

Valid Range for the Result	CC 1	CC 0	OV	OS
+0, -0 (zero)	0	0	0	*
$-3.402823E+38 < \text{result} < -1.175494E-38$ (negative number)	0	1	0	*
$+1.175494E-38 < \text{result} < 3.402824E+38$ (positive number)	1	0	0	*

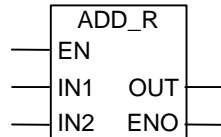
\* The OS bit is not affected by the result of the instruction.

Invalid Range for the Result	CC 1	CC 0	OV	OS
Underflow $-1.175494E-38 < \text{result} < -1.401298E-45$ (negative number)	0	0	1	1
Underflow $+1.401298E-45 < \text{result} < +1.175494E-38$ (positive number)	0	0	1	1
Overflow Result $< -3.402823E+38$ (negative number)	0	1	1	1
Overflow Result $> 3.402823E+38$ (positive number)	1	0	1	1
Not a valid floating-point number or illegal instruction (input value outside the valid range)	1	1	1	1

## 8.3 Basic Instructions

### 8.3.1 ADD\_R : Add Real

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	REAL	I, Q, M, D, L or constant	First number to be added
IN2	REAL	I, Q, M, D, L or constant	Second number to be added
OUT	REAL	I, Q, M, D, L	Result of addition
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

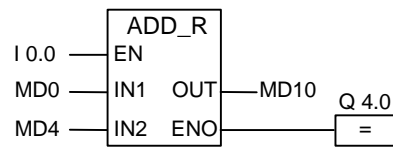
A signal state of 1 at the Enable input (EN) activates the **Add Real** instruction. This instruction adds inputs IN1 and IN2. The result can be scanned at output OUT. If either of the inputs or the result is not a floating-point number, the OV bit and OS bit are set to 1 and ENO is set to 0.

See also Evaluating the Bits of the Status Word with Floating-Point Instructions.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

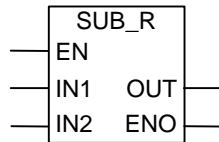
### Example



A signal state of 1 at input I0.0 activates the ADD\_R box. The result of the addition MD0 + MD4 is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

### 8.3.2 SUB\_R : Subtract Real

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	REAL	I, Q, M, D, L or constant	Minuend (from which the second value is subtracted)
IN2	REAL	I, Q, M, D, L or constant	Subtrahend (that is subtracted from the first value)
OUT	REAL	I, Q, M, D, L	Result of subtraction
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

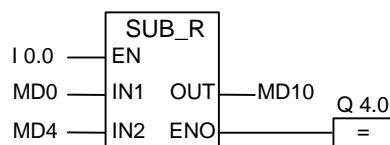
A signal state of 1 at the Enable input (EN) activates the **Subtract Real** instruction. This instruction subtracts input IN2 from IN1. The result can be scanned at output OUT. If either of the inputs or the result is not a floating-point number, the OV bit and the OS bit are set to 1 and ENO is set to 0.

See also Evaluating the Bits of the Status Word with Floating-Point Instructions.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

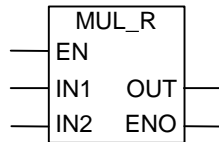
#### Example



A signal state of 1 at input I0.0 activates the SUB\_R box. The result of the subtraction MD0 - MD4 is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

### 8.3.3 MUL\_R : Multiply Real

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	REAL	I, Q, M, D, L or constant	Multiplicand (value to be multiplied)
IN2	REAL	I, Q, M, D, L or constant	Multiplier (value by which first value is multiplied)
OUT	REAL	I, Q, M, D, L	Result of multiplication
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

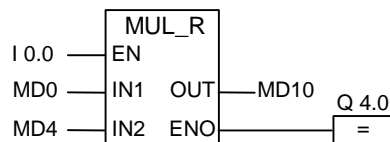
A signal state of 1 at the Enable input (EN) activates the **Multiply Real** instruction. This instruction multiplies input IN1 by IN2. The result can be scanned at output OUT. If either of the inputs or the result is not a floating-point number, the OV bit and the OS bit are set to 1 and ENO is set to 0.

See also Evaluating the Bits of the Status Word with Floating-Point Instructions.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

#### Example

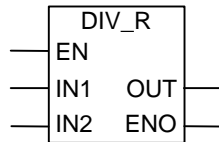


A signal state of 1 at input I0.0 activates the MUL\_R box. The result of the multiplication MD0 x MD4 is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.



### 8.3.4 DIV\_R : Divide Real

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	REAL	I, Q, M, D, L or constant	Dividend (value to be divided by second value)
IN2	REAL	I, Q, M, D, L or constant	Divisor (value by which first value is divided)
OUT	REAL	I, Q, M, D, L	Result of division
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

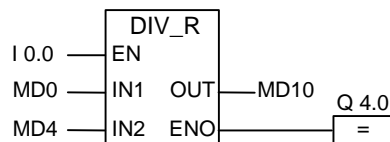
A signal state of 1 at the Enable input (EN) activates the **Divide Real** instruction. This instruction divides input IN1 by IN2. The result can be scanned at output OUT. If either of the inputs or the result is not a floating-point number, the OV bit and the OS bit are set to 1 and ENO is set to 0.

See also Evaluating the Bits of the Status Word with Floating-Point Instructions.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

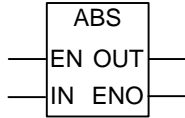
#### Example



A signal state of 1 at input I0.0 activates the DIV\_R box. The result of dividing MD0 by MD4 is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0 and the instruction is not executed.

### 8.3.5 ABS : Forming the Absolute Value of a Floating-Point Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Input value: floating-point number
OUT	REAL	I, Q, M, D, L	Output value: absolute value of the floating-point number
ENO	BOOL	I, Q, M, D, L	Enable output

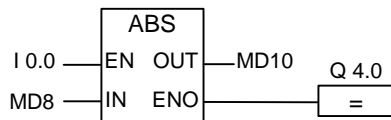
#### Description

With the **Form the Absolute Value of a Floating-Point Number** instruction, you can form the absolute value of floating-point number.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	-	-	-	-	0	X	X	1

#### Example



If I0.0 = 1, the absolute value of MD8 is output at MD12.

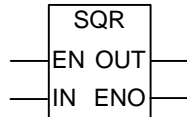
MD8 = +6.234 results in MD12 = 6.234 x 1.

Output Q4.0 is 0 if the conversion is not executed (ENO = EN = 0).

## 8.4 Extended Instructions

### 8.4.1 SQR : Forming the Square of a Floating-Point Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Square of the number
ENO	BOOL	I, Q, M, D, L	Enable output

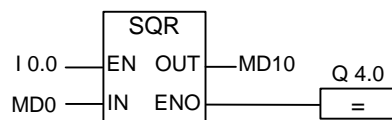
#### Description

With the **Form the Square of a Floating-Point Number** instruction, you can square a floating-point number. If either of the inputs or the result is not a floating-point number, the OV bit and OS bit are set to 1 and ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

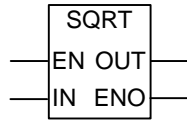
#### Example



A signal state of 1 at input I0.0 activates the SQR box. The result of SQR (MD0) is entered in the memory double word MD10. If MD0 is less than 0 or if either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0.

### 8.4.2 SQRT : Forming the Square Root of a Floating-Point Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Square root of the number
ENO	BOOL	I, Q, M, D, L	Enable output

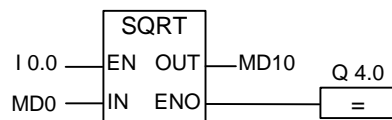
#### Description

With the **Form the Square Root of a Floating-Point Number** instruction, you can extract the square root of a floating-point number. This instruction returns a positive result, if the value at the address is greater than "0". If either of the inputs or the result is not a floating-point number, the OV bit and OS bit are set to 1 and ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

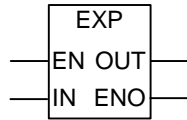
#### Example



A signal state of 1 at input I0.0 activates the SQRT box. The result of SQRT (MD0) is entered in memory double word MD10. If MD0 is less than 0 or if either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0.

### 8.4.3 EXP : Forming the Exponential Value of a Floating-Point Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Exponent of the number
ENO	BOOL	I, Q, M, D, L	Enable output

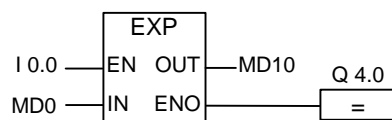
#### Description

With the **Form the Exponential Value of a Floating-Point Number** instruction, you can form the exponential value of a floating-point number to base e (= 2.71828...). If either of the inputs or the result is not a floating-point number, the OV bit and OS bit are set to 1 and ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

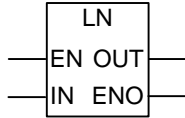
#### Example



A signal state of 1 at input I0.0 activates the EXP box. The result of EXP (MD0) is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I 0.0 is 0, output Q4.0 is set to 0.

### 8.4.4 LN : Forming the Natural Logarithm of a Floating-Point Number

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Natural logarithm of the number
ENO	BOOL	I, Q, M, D, L	Enable output

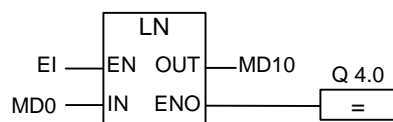
#### Description

With the **Form the Natural Logarithm of a Floating-Point Number** instruction, you can form the natural logarithm of a floating-point number. If either of the inputs or the result is not a floating-point number, the OV bit and OS bit are set to 1 and ENO is set to 0.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

#### Example



A signal state of 1 at input I0.0 activates the LN box. The result of LN (MD0) is entered in memory double word MD10. If MD0 is less than 0 or if either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0.

### 8.4.5 Forming Trigonometric Functions of Angles as Floating-Point Numbers

#### Description

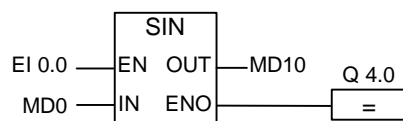
With the following instructions, you can form trigonometric functions of angles represented as 32-bit IEEE floating-point numbers.

Instruction	Meaning
SIN	Form the sine of a floating-point number of an angle specified in radians.
COS	Form the cosine of a floating-point number of an angle specified in radians.
TAN	Form the tangent of a floating-point number of an angle specified in radians.
ASIN	Form the arc sine of a floating-point number . The result is an angle specified in radians. The value is in the following range: -p / 2 <= arc sine <= + p / 2, where p = 3.14...
ACOS	Form the arc cosine of a floating-point number . The result is an angle specified in radians. The value is in the following range: 0 v arc cosine <= + p, where p = 3.14...
ATAN	Form the arc tangent of a floating-point number . The result is an angle specified in radians. The value is in the following range: -p / 2 <= arc tangent <= + p / 2, where p = 3.14...

#### Status Word

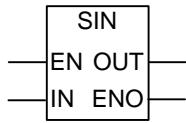
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	X	0	X	X	1

#### Example



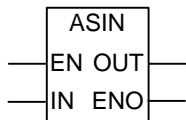
A signal state of 1 at input I0.0 activates the SIN box. The result of SIN (MD0) is entered in memory double word MD10. If either of the inputs or the result is not a floating-point number and if the signal state of I0.0 is 0, output Q4.0 is set to 0.

**Symbol**



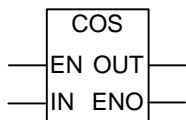
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Sine of the number
ENO	BOOL	I, Q, M, D, L	Enable output

**Symbol**



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Arc sine of the number
ENO	BOOL	I, Q, M, D, L	Enable output

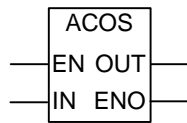
**Symbol**



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Cosine of the number
ENO	BOOL	I, Q, M, D, L	Enable output

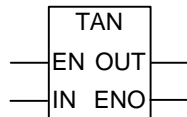


**Symbol**



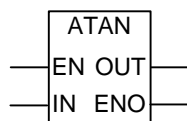
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Arc cosine of the number
ENO	BOOL	I, Q, M, D, L	Enable output

**Symbol**



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Tangent of the number
ENO	BOOL	I, Q, M, D, L	Enable output

**Symbol**



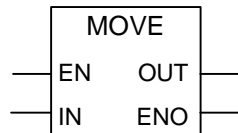
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	REAL	I, Q, M, D, L or constant	Number
OUT	REAL	I, Q, M, D, L	Arc tangent of the number
ENO	BOOL	I, Q, M, D, L	Enable output



## 9 Move Instructions

### 9.1 MOVE : Assign Value

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN	All elementary data types with a length of 8, 16 or 32 bits	I, Q, M, D, L or constant	Source value
OUT	All elementary data types with a length of 8, 16 or 32 bits	I, Q, M, D, L	Destination address
ENO	BOOL	I, Q, M, D, L	Enable output

#### Description

With the **Assign Value** instruction, you can assign specific values to variables.

The value specified at the IN input is copied to the address specified at the OUT output. ENO has the same signal state as EN.

With the MOVE box, the Assign Value instruction can copy all elementary data types with lengths of 8, 16, or 32 bits. User-defined data types such as arrays or structures must be copied with the system function SFC 20 "BLKMOV".

The **Assign Value** instruction is affected by the Master Control Relay (MCR). For more information on how the MCR functions, see MCR</MCR> : Master Control Relay On/Off.

**Status Word**

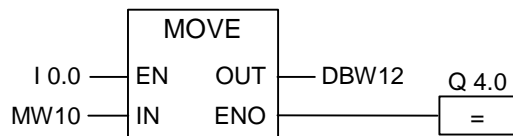
	<b>BR</b>	<b>CC1</b>	<b>CC0</b>	<b>OV</b>	<b>OS</b>	<b>OR</b>	<b>STA</b>	<b>RLO</b>	<b>FC</b>
writes	1	-	-	-	-	0	1	1	1

**Note**

When moving a value to a data type of a different length, higher-value bytes are truncated as necessary or filled up with zeros:

<b>Example: Double Word</b>	<b>1111 1111</b>	<b>0000 1111</b>	<b>1111 0000</b>	<b>0101 0101</b>
<b>Move</b>	<b>Result</b>			
to a double word:	1111 1111	0000 1111	1111 0000	0101 0101
to a byte:				0101 0101
to a word:			1111 0000	0101 0101
<b>Example: Byte:</b>				<b>1111 0000</b>
<b>Move</b>	<b>Result</b>			
to a byte:				1111 0000
to a word:			0000 0000	1111 0000
to a double word:	0000 0000	0000 0000	0000 0000	1111 0000

**Example**



The instruction is executed, when input I0.0 is 1. The content of MW10 is copied to data word 12 of the open DB.

If the instruction is executed, Q4.0 is set to 1.

# 10 Program Control Instructions

## 10.1 Overview of Program Control Instructions

### Description

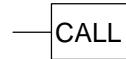
The following instructions are available for performing program control operations:

- CALL: Calling an FC/SFC without Parameters
- CALL\_FB : Call FB as Box
- CALL\_FC : Call FC as Box
- CALL\_SFB : Call System FB as Box
- CALL\_SFC : Call System FC as Box
- Calling Multiple Instances
- Calling a Block from a Library
  
- Master Control Relay Instructions
- Important Notes on Using MCR Functions
- MCR< Master Control Relay On
- MCR> Master Control Relay Off
- MCRA Master Control Relay Activate
- MCRD Master Control Relay Deactivate
  
- RET Return

## 10.2 CALL : Calling an FC/SFC without Parameters

### Symbol

<FC-/SFC number>



Parameter	Data Type	Memory Area	Description
Number	BLOCK_FC	-	Number of the FC or SFC (for example FC10 or SFC59). The SFCs that are available depend on your CPU.  A conditional call with a parameter of the data type BLOCK_FC as the address is only possible in an FB and not in an FC.

### Description

With the **Call FC/SFC without Parameters** instruction, you can call a function (FC) or a system function (SFC) that has no parameters. The call is conditional or unconditional depending on the preceding logic operation (see the example).

In the code section of a function (FC), you cannot specify any parameter of the type BLOCK\_FC as the address for a conditional call. You can, however, specify a parameter of the type BLOCK\_FC as the address in a function block (FB).

A conditional call is executed only if the RLO is 1. If a conditional call is not executed, the RLO after the call instruction is 0. If the instruction is executed, the following functions are performed:

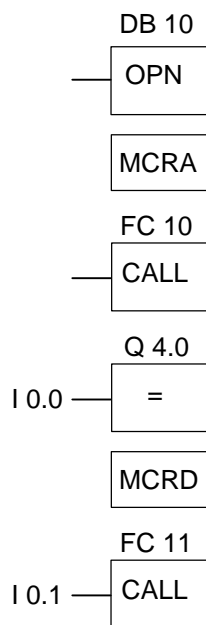
- The address required to return to the calling block is saved.
- The data block registers are saved (data block and instance data block).
- The MA bit (active MCR bit) is written to the block stack (BSTACK).
- The new local data area is created for the called FC or SFC.

Program execution is then continued in the called block.

### Status Word

		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
Conditional call	writes	-	-	-	-	0	0	1	1	0
Unconditional call	writes	-	-	-	-	0	0	1	-	0

### Example



If the unconditional call for FC10 is executed, the CALL instruction performs the following functions:

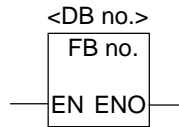
- Saves the address required to return to the current FB.
- Saves the selectors for DB10 and for the instance data block of the FB.
- Pushes the MA bit, set to 1 in the MCRA instruction, to the block stack (BSTACK) and resets this bit to 0 for the called FC10

Program execution continues in FC10. If you want to use the MCR function in FC10, you must reactivate it there. When FC10 is completed, program execution returns to the calling FB. The MA bit is restored. DB10 and the instance data block of the user-defined FB are the current DBs again, regardless of which DBs were used by FC10.

After the return jump from FC10, the signal state of input I0.0 is assigned to output Q4.0. The call for FC11 is a conditional call. It is executed only if the signal state of input I0.1 is 1. If the call is executed, the function is the same as for calling FC10.

## 10.3 CALL\_FB : Call FB as Box

### Symbol



The symbol depends on the function block (whether and how many parameters exist). EN, ENO and the name or number of the FB must exist.

Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
FB no.	BLOCK_FB	-	Number of the FB/DB, range depends on the CPU
DB no.	BLOCK_DB	-	

### Description

**CALL\_FB** (call FB as box) is executed when the signal state of EN is 1. If the CALL\_FB instruction call is executed:

- The return address of the calling block is saved,
- The selection data for the two currently open data blocks (DB and instance DB) are saved,
- A new local data area is created for the called function block.
- The MA bit (active MCR bit) is pushed to the BSTACK,

### Status Word

		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
Conditional call	writes	X	-	-	-	0	0	X	X	X
Unconditional call	writes	-	-	-	-	0	0	X	X	X



## Example

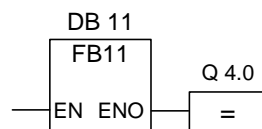
Network 1



Network 2



Network 3



Network 4



The networks shown above are program sections of a function block created by the user. DB10 is opened in this block and the MCR activated. If the unconditional FB11 call is executed, the following happens:

The return address of the calling function block and the selection data for DB10 and the instance data block of the calling function block are saved. The MA bit that was set to 1 by the MCRA instruction is pushed to the BSTACK and then set to 0 for the called function block FB11. The program is continued in FB11. If FB11 requires the MCR, the MCR must be reactivated in the function block. The signal state of the RLO must be saved in the BR bit by the [SAVE] instruction to allow error evaluation in the calling FB. When FB11 has been executed, the program returns to the calling function block. The MA bit is restored and the instance data block of the function block written by the user once again becomes the open data block. If FB11 is executed correctly, the signal state of ENO is 1 and the state of Q4.0 is therefore also 1.

---

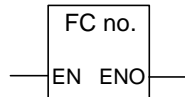
### Note

With FB/SFB calls, the number of the previously open data block is lost. The required DB must be opened again.

---

## 10.4 CALL\_FC (Call FC as Box)

### Symbol



The symbol depends on the function (whether and how many parameters exist). EN, ENO and the name or number of the FC must exist.

Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
FC no.	BLOCK_FC	-	Number of the FC, range depends on the CPU

### Description

**CALL\_FC** (call FC as box) calls a function (FC). The call is executed when EN has a signal state of 1. If the CALL\_FC instruction is executed:

- The return address of the calling block is saved,
- A new local data area is created for the called function.
- The MA bit (active MCR bit) is pushed to the BSTACK,

Finally, program execution is resumed in the called function.

The BR bit is checked to determine the ENO. The user must assign the required signal state (error evaluation) to the BR bit in the called block [SAVE].

If you call a function and the variable declaration table of the called block has IN, OUT, and IN\_OUT declarations, these variables are added in the program for the calling block as a list of formal parameters.

When calling the function, you **must** assign actual parameters to the formal parameters at the call location. Any initial values in the function declaration have no significance.

### Status Word

		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
Conditional call	writes	X	-	-	-	0	0	X	X	X
Unconditional call	writes	-	-	-	-	0	0	X	X	X

## Example

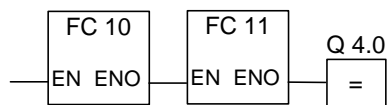
Network 1



Network 2



Network 3



The networks shown above are program sections of a function block created by the user. DB10 is opened in this block and the MCR activated. If the unconditional FC10 call is executed, the following happens:

The return address of the calling function block and the selection data for DB10 and the instance data block of the calling function block are saved. The MA bit that was set to 1 by the MCRA instruction is pushed to the BSTACK and then set to 0 for the called block FC10. The program is continued in FC10. If FC10 requires the MCR, the MCR must be reactivated in FC10. The signal state of the RLO must be saved in the BR bit by the [SAVE] instruction to allow error evaluation in the calling FB. When FC10 has been executed, the program returns to the calling function block. The MA bit is restored. After FC10 has been executed, the program is resumed in the calling FB depending on the signal state of ENO, as follows:

ENO = 1 FC11 is executed

ENO = 0 The program starts in the next network

If FC11 was correctly executed, ENO is set to 1 and Q4.0 is therefore also set to 1.

---

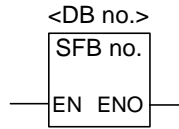
### Note

After the program returns to the calling block, it is not always guaranteed that the previously open DB is opened again. Please refer to the note in the readme file.

---

## 10.5 CALL\_SFB : Call System FB as Box

### Symbol



The symbol depends on the system function block (whether and how many parameters exist). EN, ENO and the name or number of the SFB must exist.

Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
SFB no.	BLOCK_SFB	-	Number of the SFB/DB, range depends on the CPU
DB no.	BLOCK_DB	-	

### Description

**CALL\_SFB** (call SFB as box) is executed when the signal state of EN is 1. If the CALL\_SFB instruction is executed:

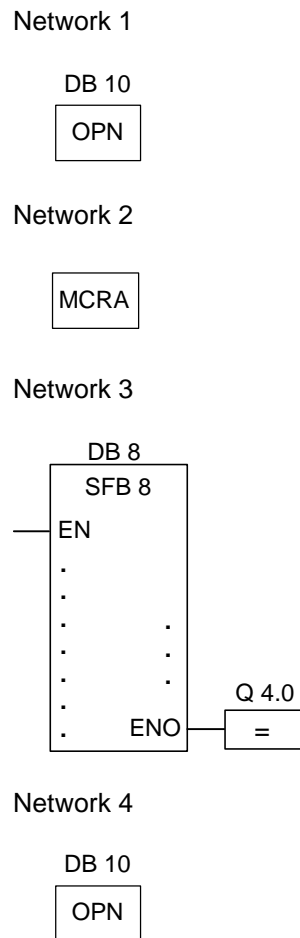
- The return address of the calling block is saved,
- The selection data for the two currently open data blocks (DB and instance DB) are saved,
- A new local data area is created for the called system function block.
- The MA bit (active MCR bit) is pushed to the BSTACK,

Finally, program execution is resumed in the called system function block. ENO is 1 if the function was called and no errors occurred.

### Status Word

		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
Conditional call	writes	X	-	-	-	0	0	X	X	X
Unconditional call	writes	-	-	-	-	0	0	X	X	X

## Example



The networks shown above are program sections of a function block created by the user. DB10 is opened in this block and the MCR activated. If the unconditional SFB8 call is executed, the following happens:

The return address of the calling function block and the selection data for DB10 and the instance data block of the calling function block are saved. The MA bit that was set to 1 by the MCRA instruction is pushed to the BSTACK and then set to 0 for the called system function block SFB8. The program is continued in SFB8. When SFB8 has been executed, the program returns to the calling function block. The MA bit is restored and the instance data block of the function block written by the user once again becomes the current data block. If SFB8 is executed correctly, the signal state of ENO is 1 and the state of Q4.0 is therefore also 1.

---

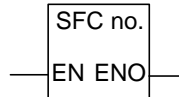
### Note

With FB/SFB calls, the number of the previously open data block is lost. The required DB must be opened again.

---

## 10.6 CALL\_SFC (Call System FC as Box)

### Symbol



The symbol depends on whether and how many parameters exist). EN, ENO and the name or number of the SFC must exist.

Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D	Enable input
ENO	BOOL	I, Q, M, L, D	Enable output
SFC no.	BLOCK_SFC	-	Number of the SFC, range depends on the CPU

### Description

**CALL\_SFC** (call system FC as box) calls a system function. The call is executed when the signal state of EN is 1. If the CALL\_SFC instruction is executed:

- The return address of the calling block is saved,
- A new local data area is created for the called system function.
- The MA bit (active MCR bit) is pushed to the BSTACK,

Finally, program execution is resumed in the called system function. ENO is 1 if the function was called and no errors occurred.

### Status Word

		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
Conditional call	writes	X	-	-	-	0	0	X	X	X
Unconditional call	writes	-	-	-	-	0	0	X	X	X

### Example

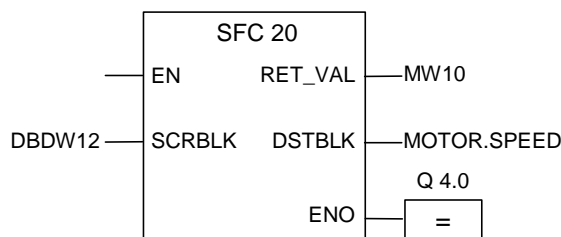
Network 1



Network 2



Network 3



The networks shown above are program sections of a function block created by the user. DB10 is opened in this block and the MCR activated. If the unconditional SFC20 call is executed, the following happens:

The return address of the calling function block and the selection data for DB10 and the instance data block of the calling function block are saved. The MA bit that was set to 1 by the MCRA instruction is pushed to the BSTACK and then set to 0 for the called block SFC20. The program is continued in SFC20. When SFC20 has been executed, the program returns to the calling function block. The MA bit is restored.

After SFC20 has been executed, the program is resumed in the calling FB depending on the signal state of ENO, as follows:

ENO = 1    Q4.0 = 1

ENO = 0    Q4.0 = 0

---

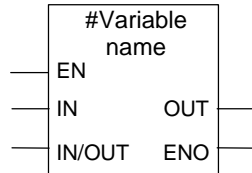
#### Note

After the program returns to the calling block, it is not always guaranteed that the previously open DB is opened again. Please refer to the note in the readme file.

---

## 10.7 Calling Multiple Instances

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
ENO	BOOL	I, Q, M, D, L	Enable output
# Variable name	FB/SFB	-	Name of the multiple instance

### Description

You create a multiple instance by declaring a static variable of the type function block. Only multiple instances that have already been declared are listed in the program element catalog. The symbol of a multiple instance changes depending on whether and how many parameters exist. EN, ENO and the variable name are always present.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	0	0	X	X	X

## 10.8 Calling a Block from a Library

The libraries available in the SIMATIC Manager can be used here to select a block that:

- Is integrated in your CPU operating system ("Standard Library" library for STEP 7 projects in version 3 and "stdlibs (V2)" for STEP 7 projects in version 2)
- You saved yourself in a library because you wanted to use it a number of times.



## 10.9 Master Control Relay Instructions

Important Notes on Using MCR Functions

### Definition of the Master Control Relay

The Master Control Relay (MCR, see also MCR</MCR> : Master Control Relay On/Off) is used to activate and deactivate signal flow. A deactivated signal flow 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. Operations triggered by the instructions shown below are dependent on the MCR.

The **Assign** and **Midline Output** instructions write a 0 to the memory if the MCR is 0. The **Set Output** and **Reset Output** instructions leave the existing value unchanged.

Instructions affected by MCR zones are as follows:

- # : Midline Output
- = : Assign
- R : Reset Output
- S : Set Output
- SR : Set\_Reset Flip Flop
- RS : Reset\_Set Flip Flop
- MOVE : Assign Value

### Instructions dependent on MCR and their reactions to the signal state of the MCR

Signal State of MCR	Assign, Midline Output	Set or Reset Output	Move
0 ("OFF")	Writes 0. (Imitates a relay that falls to its quiet state when voltage is removed.)	Does not write. (Imitates a relay that remains in its current state when voltage is removed.)	Writes 0. (Imitates a component that produces a value of 0 when voltage is removed.)
1 ("ON")	Normal processing	Normal processing	Normal processing

## 10.10 Important Notes on Using MCR Functions



---

### Take care with blocks in which the Master Control Relay was activated with MCRA

- If the MCR is deactivated, the value 0 is written by all assignments in program segments between Master Control Relay On and Master Control Relay Off. This is valid for **all** boxes which contain an assignment, including the parameter transfer to blocks.
  - The MCR is deactivated if the RLO was = 0 before a Master Control Relay On instruction.
- 



---

### Danger: PLC in STOP or undefined runtime characteristics!

The compiler also uses write access to local data behind the temporary variables defined in VAR\_TEMP for calculating addresses. This means the following command sequences will set the PLC to STOP or lead to undefined runtime characteristics:

#### Formal parameter access

- Access to components of complex FC parameters of the type STRUCT, UDT, ARRAY, STRING
- Access to components of complex FB parameters of the type STRUCT, UDT, ARRAY, STRING from the IN\_OUT area in a block with multiple instance capability (version 2 block).
- Access to parameters of a function block with multiple instance capability (version 2 block) if its address is greater than 8180.0.
- Access in a function block with multiple instance capability (version 2 block) to a parameter of the type BLOCK\_DB opens DB0. Any subsequent data access sets the CPU to STOP. T 0, C 0, FC0, or FB0 are also always used for TIMER, COUNTER, BLOCK\_FC, and BLOCK\_FB.

#### Parameter passing

- Calls in which parameters are transferred.

#### LAD/FBD

- T branches and midline outputs in Ladder or FBD starting with RLO = 0.

#### Remedy

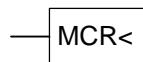
Free the above commands from their dependence on the MCR:

- **Deactivate** the Master Control Relay using the Master Control Relay Deactivate instruction **before** the statement or network in question.
  - **Activate** the Master Control Relay **again** using the Master Control Relay Activate instruction after the statement or network in question.
-

## 10.11 MCR</MCR> : Master Control Relay On/Off

Important Notes on Using MCR Functions

### Symbol

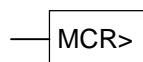


### MCR On

The **Master Control Relay On** (MCR<) instruction triggers an operation that saves the RLO in the MCR stack and opens an MCR zone. The instructions shown in MCR Instructions are influenced by this RLO that is saved in the MCR stack when the MCR zone is opened.

The MCR stack works like a LIFO (Last In, First Out) buffer. Only eight entries are possible. If the stack is already full, the **Master Control Relay On** instruction produces an MCR stack error (MCRF).

### Symbol



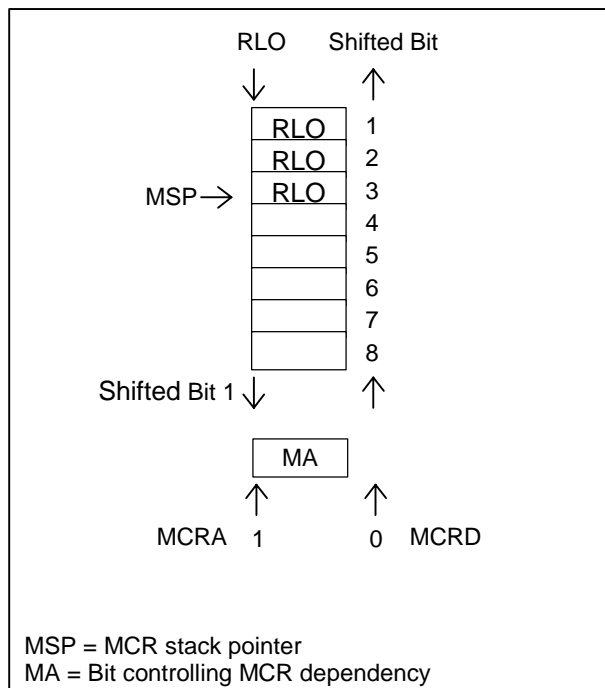
### MCR Off

The **Master Control Relay Off** (MCR>) instruction closes the MCR zone that was opened last. The instruction does this by removing the RLO entry from the MCR stack. The RLO was saved there by the **Master Control Relay On** instruction. The entry released at the other end of the LIFO (Last In, First Out) MCR stack is set to 1.

If the stack is already empty, the **Master Control Relay Off** instruction produces an MCR stack error (MCRF).

### MCR-Stack

The MCR is controlled by a stack which is one bit wide and eight entries deep. The MCR is activated as long as all eight entries in the stack are equal to 1. The MCR< instruction copies the RLO to the MCR stack. The MCR> instruction removes the last entry from the stack and sets the released stack address to 1. If an error occurs, for example, if there are more than eight MCR> instructions in succession, or you attempt to execute the instruction MCR> when the stack is empty, the MCRF error message is activated. The monitoring of the MCR stack is based on the stack pointer (MSP: 0 = empty, 1 = one entry, 2 = two entries, ..., 8 = eight entries).



The **MCR<** instruction adopts the signal state of the RLO and copies it to the MCR bit.

The **MCR>** instruction sets the MCR bit to 1 unconditionally. Because of this characteristic, every other instruction between the instructions MCRA and MCRD operates independent of the MCR bit.

### Nesting the Instructions MCR< and MCR>

You can nest the instructions **MCR<** and **MCR>**. The maximum nesting depth is eight, in other words, you can write a maximum of eight MCR< instructions in succession before inserting an MCR> instruction. You must program an equal number of MCR< and MCR> instructions.

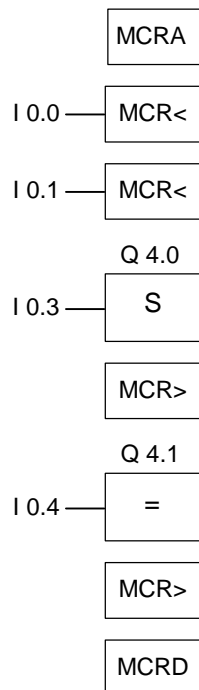
If the MCR< instructions are nested, the MCR bit of the lower nesting level is formed. The MCR< instruction then combines the current RLO with the current MCR bit according to the AND truth table.

When an MCR> instruction completes a nesting level, it fetches the MCR bit from a higher level.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	1	-	0

### Example



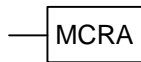
When the **MCRA** instruction activates the MCR function, you can create up to eight nested MCR zones. In the example, there are two MCR zones. The first MCR> instruction works together with the second MCR< instruction. All instructions between the second set of MCR brackets (MCR<MCR>) belong to the second MCR zone. The operations are executed as follows

- If I0.0 = 1: the signal state of input I0.4 is assigned to output Q4.1.
- If I0.0 = 0: the signal state of output Q4.1 is 0 regardless of the signal state of input I0.4. Output Q4.0 remains unchanged regardless of the signal state of input I0.3.
- If I0.0 and I0.1 = 1: output Q4.0 is set to 1 if I0.3 = 1 and Q4.1 = I0.4.
- If I0.1 = 0: output Q4.0 remains unchanged regardless of the signal state of input I0.3 and input I0.0.

## 10.12 MCRA/MCRD : Master Control Relay Activate/Deactivate

Important Notes on Using MCR Functions

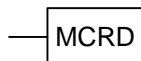
### Symbol



### MCR Activate

With the **Activate Master Control Relay**, instruction, you make subsequent commands dependent on the MCR. After entering this command, you can program the MCR zones with these instructions (see MCR</MCR> : Master Control Relay On/Off). When your program activates an MCR area, all MCR actions depend on the content of the MCR stack.

### Symbol



### MCR Deactivate

With the **Deactivate Master Control Relay** instruction, subsequent commands are no longer dependent on the MCR. After this instruction, you cannot program any more MCR zones. When your program deactivates an MCR area, the MCR is always energized irrespective of the entries in the MCR stack.

The MCR stack and the bit that controls its dependency (the MA bit) relate to individual levels and must be saved and fetched every time you change the sequence level. They are preset at the beginning of every sequence level (MCR input bits 1 to 8 are set to 1, the MCR stack pointer is set to 0 and the MA bit is set to 0).

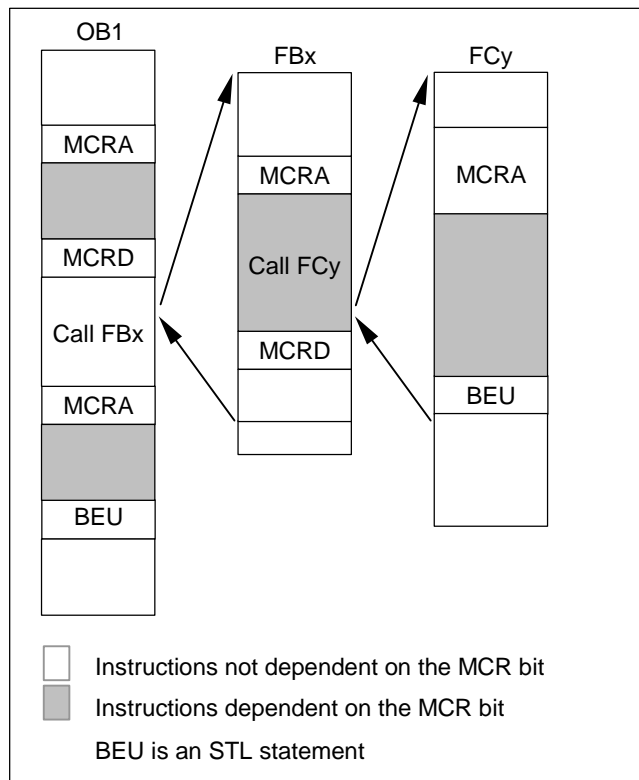
The MCR stack is transferred from block to block and the MA bit is saved and set to 0 every time a block is called. It is fetched back at the end of the block.

The MCR can be implemented in such a way that it optimizes the run time of code-generating CPUs. The reason for this is that the dependency of the MCR is not passed on by the block; it must be explicitly activated by an MCR instruction. A code-generating CPU recognizes this instruction and generates the additional code necessary for the evaluation of the MCR stack until it recognizes an MCR instruction or reaches the end of the block. With instructions outside the MCRA/MCRD range, there is no increase of the run time.

The instructions **MCRA** and **MCRD** must always be used in pairs within your program.

### Activate and Deactivate a MCR zone

The operations programmed between MCRA and MCRD depend on the signal state of the MCR bit. Operations programmed outside an MCRA-MCRD sequence do not depend on the signal state of the MCR bit. If an MCRD instruction is missing, the operations programmed between the instructions MCRA and BEU depend on the MCR bit.

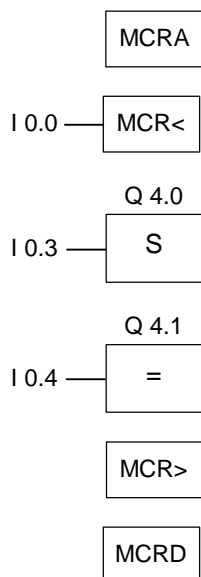


### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	-	-	-	-



## Example



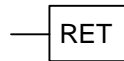
The MCRA instruction activates the MCR function until the next MCRD. The instructions between MCR< and MCR> are processed dependent on the MA bit (here I0.0):

- If the signal state of input I0.0 is 1:
  - Output Q4.0 is set to 1 if the signal state of input I0.3 is 1.
  - Output Q4.0 remains unchanged if the signal state of input I0.3 is 0.
  - The signal state of input I0.4 is assigned to output Q4.1.
- If the signal state of input I0.0 is 0:
  - Output Q4.0 remains unchanged regardless of the signal state of input I0.3.
  - Output Q4.1 is 0 regardless of the signal state of input I0.4.

You must program the dependency of the functions (FCs) and function blocks (FBs) in the blocks yourself. If this function or function block is called from an MCRA/MCRD sequence, not all instructions within this sequence are automatically dependent on the MCR bit. To achieve this, use the instruction MCRA of the block called.

## 10.13 RET : Return

### Symbol



### Description

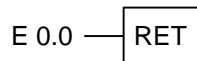
You can use the **Return** instruction to exit blocks. You can exit a block conditionally.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	*	-	-	-	0	0	1	1	0

\* The operation **RET** is shown internally in the sequence "SAVE; BEC, ". This also affects the BR bit

### Example



If the signal state of input I0.0 is 1, the block is exited.

# 11 Shift and Rotate Instructions

## 11.1 Shift Instructions

### 11.1.1 Overview of Shift Instructions

#### Description

You can use the Shift instructions to move the contents of input IN bit by bit to the left or the right (see also CPU Registers). Shifting  $n$  bits to the left multiplies the contents of input IN by 2 to the power  $n$  ( $2^n$ ); shifting  $n$  bits to the right divides the contents of input IN by 2 to the power  $n$  ( $2^n$ ). For example, if you shift the binary equivalent of the decimal value 3 to the left by 3 bits, you obtain the binary equivalent of the decimal value 24. If you shift the binary equivalent of the decimal value 16 to the right by 2 bits, you obtain the binary equivalent of the decimal value 4.

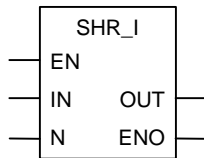
The number that you supply for input parameter N indicates the number of bits by which the value is shifted. The bit places that are vacated by the Shift instruction are either padded with zeros or with the signal state of the sign bit (0 stands for positive and 1 stands for negative). The signal state of the bit that is shifted last is loaded into the CC1 bit of the status word, see also CPU Registers. The CC0 and OV bits of the status word are reset to 0. You can use jump instructions to evaluate the CC1 bit.

The following Shift instructions are available:

- SHR\_I : Shift Right Integer
- SHR\_DI : Shift Right Double Integer
- SHL\_W : Shift Left Word
- SHR\_W : Shift Right Word
- SHL\_DW : Shift Left Double Word
- SHR\_DW : Shift Right Double Word

### 11.1.2 SHR\_I : Shift Right Integer

#### Symbol



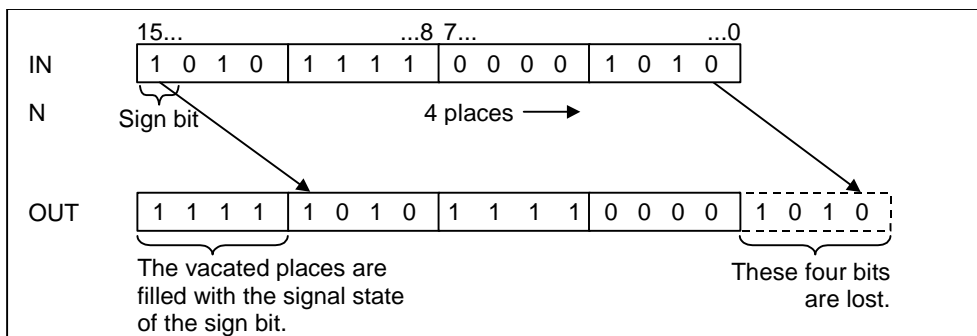
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	INT	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	INT	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Right Integer** instruction. This instruction shifts bits 0 to 15 of input IN bit by bit to the right. Input N specifies the number of bits by which the value will be shifted. If N is higher than 16, the command behaves as if N were 16. The bit positions at the left are padded according to the signal state of bit 15 (the sign of an integer number). They are filled with zeros if the number is positive, and with ones if it is negative. The result of the shift operation can be scanned at output OUT.

The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if N is not equal to zero.

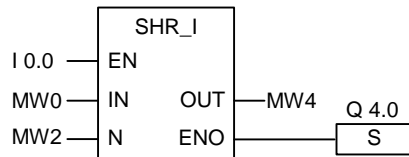
ENO has the same signal state as EN.



**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

**Example**

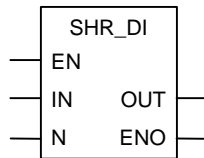


The instruction is activated if the signal state of I0.0 is 1. Memory word MW0 is shifted to the right by the number of bits specified in memory word MW2.

The result is entered in memory word MW4. Output Q4.0 is set to 1.

### 11.1.3 SHR\_DI : Shift Right Double Integer

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	DINT	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	DINT	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Right Double Integer** instruction. This instruction shifts the entire contents of input IN bit by bit to the right. Input N specifies the number of bits by which the value will be shifted. If N is higher than 32, the command behaves as if N were 32. The bit positions at the left are padded according to the signal state of bit 31 (the sign of a double integer number). They are filled with zeros if the number is positive, and with ones if it is negative. The result of the shift operation can be scanned at output OUT.

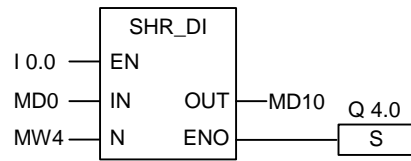
The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if N is not equal to zero.

ENO has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

**Example**

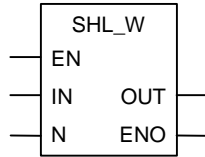


The instruction is activated if the signal state of I0.0 is 1. Memory double word MD0 is shifted to the right by the number of bits specified in memory word MW4.

The result is entered in memory double word MD10. Output Q4.0 is set to 1.

### 11.1.4 SHL\_W : Shift Left Word

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	WORD	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	WORD	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

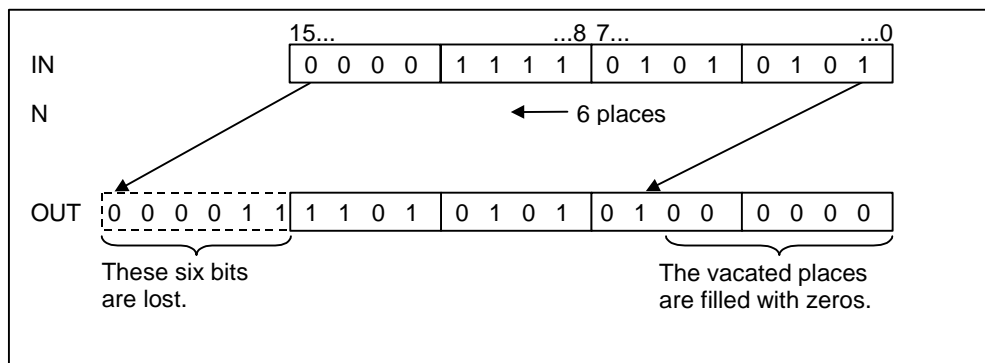
#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Left Word** instruction. This instruction shifts bits 0 to 15 of input IN bit by bit to the left.

Input N specifies the number of bits by which to shift the value. If N is higher than 16, the command writes 0 to output OUT and sets the CC0 and OV bits of the status word to 0. The bit positions at the right are padded with zeros. The result of the shift operation can be scanned at output OUT.

The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if the value of N is not equal to 0.

ENO has the same signal state as EN.

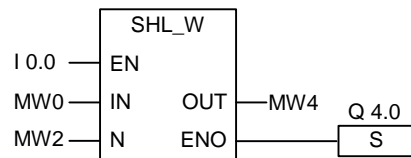




**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

**Example**



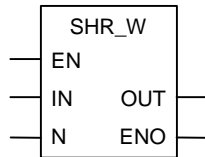
The instruction is activated if the signal state of I0.0 is 1.

Memory word MW0 is shifted to the left by the number of bits specified in memory word MW2

The result is entered in memory word MW4. Output Q4.0 is set to 1.

### 11.1.5 SHR\_W : Shift Right Word

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	WORD	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	WORD	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Right Word** instruction. This instruction shifts bits 0 to 15 of input IN bit by bit to the right. Bits 16 to 31 are not affected. Input N specifies the number of bits by which the value will be shifted. If N is greater than 16, the command writes 0 to output OUT and resets the CC0 and OV bits of the status word to 0. The vacated bit positions at the left are padded with zeros. The result of the shift operation can be scanned at output OUT.

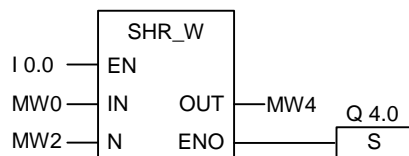
The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if N is not equal to zero.

ENO has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

#### Example

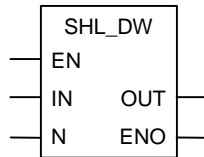


The instruction is activated if the signal state of I0.0 is 1. Memory word MW0 is shifted to the right by the number of bits specified in memory word MW2.

The result is entered in memory word MW4. Output Q4.0 is set to 1.

### 11.1.6 SHL\_DW : Shift Left Double Word

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	DWORD	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	DWORD	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Left Double Word** instruction. This instruction shifts bits 0 to 31 of input IN bit by bit to the left. Input N specifies the number of bits by which the value will be shifted. If N is greater than 32, the command writes 0 to output OUT and sets the CC0 and OV bits of the status word to 0. The vacated bit positions at the right are padded with zeros. The result of the shift operation can be scanned at output OUT.

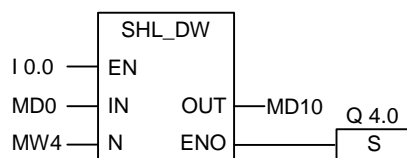
The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if the value of N is not equal to 0.

ENO has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

#### Example



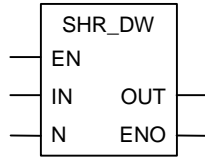
The instruction is activated if the signal state of I0.0 is 1.

Memory double word MD0 is shifted to the left by the number of bits specified in memory word MW4.

The result is entered in memory double word MD10. Output Q4.0 is set to 1.

### 11.1.7 SHR\_DW : Shift Right Double Word

#### Symbol



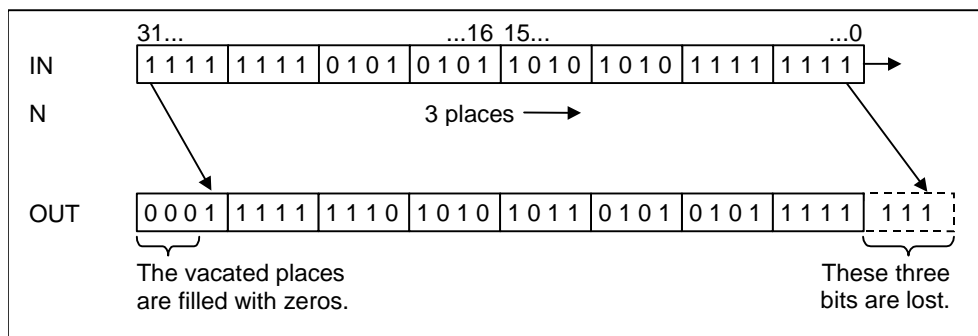
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	DWORD	I, Q, M, L, D	Value to be shifted
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be shifted
OUT	DWORD	I, Q, M, L, D	Result of the shift instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Shift Right Double Word** instruction. This instruction shifts bits 0 to 31 of input IN bit by bit to the right. Input N specifies the number of bits by which the value will be shifted. If N is higher than 32, the command writes 0 to output OUT and resets the CC0 and OV bits of the status word to 0. The vacated bit positions at the left are padded with zeros. The result of the shift operation can be scanned at output OUT.

The operation triggered by this instruction always resets the CC1 and OV bits of the status word to 0 if N is not equal to zero.

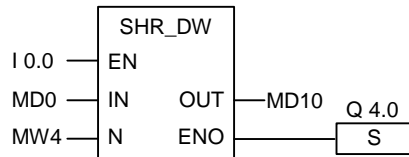
ENO has the same signal state as EN.



**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

**Example**



The instruction is activated if the signal state of I0.0 is 1. Memory double word MD0 is shifted to the right by the number of bits specified in memory word MW4.

The result is entered in MD10. Output Q4.0 is set to 1.

## 11.2 Rotate Instructions

### 11.2.1 Overview of Rotate Instructions

#### Description

You can use the Rotate instructions to rotate the entire contents of input IN bit by bit to the left or to the right (see also CPU Registers). The vacated bit positions are filled with the signal states of the bits that are shifted out of input IN.

The number that you specify for input parameter N is the number of bits by which the value will be rotated.

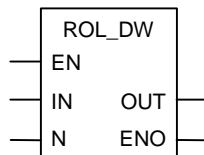
Depending on the instruction, rotation uses the CC1 bit of the status word. The CC0 bit of the status word is reset to 0.

The following Rotate instructions are available:

- ROL\_DW : Rotate Left Double Word
- ROR\_DW : Rotate Right Double Word

### 11.2.2 ROL\_DW : Rotate Left Double Word

#### Symbol



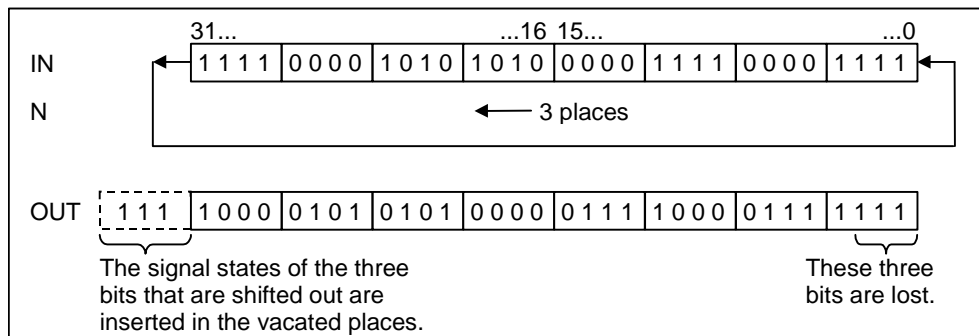
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	DWORD	I, Q, M, L, D	Value to be rotated
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be rotated
OUT	DWORD	I, Q, M, L, D	Result of the rotate instruction
ENO	BOOL	I, Q, M, L, D	Enable output

**Description**

A signal state of 1 at the Enable input (EN) activates the **Rotate Left Double Word** instruction. This instruction rotates the entire contents of input IN bit by bit to the left. Input N specifies the number of bits by which to rotate. If N is higher than 32, the double word is rotated ((N-1) modulo 32) + 1) places. The bit positions at the right are filled with the signal states of the bits rotated. The result of the rotate operation can be scanned at output OUT.

The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if N is not equal to zero.

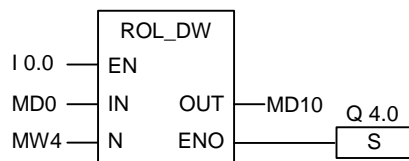
ENO has the same signal state as EN.



**Status Word**

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

**Example**

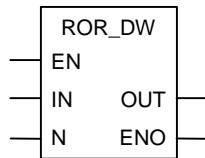


The instruction is activated if the signal state of I0.0 is 1. Memory double word MD0 is rotated to the left by the number of bits specified in memory word MW4.

The result is entered in memory double word MD10. Output Q4.0 is set to 1.

### 11.2.3 ROR\_DW : Rotate Right Double Word

#### Symbol



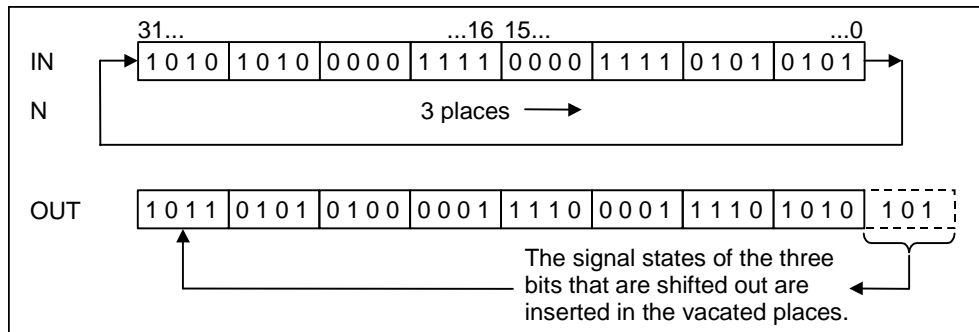
Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, L, D, T, C	Enable input
IN	DWORD	I, Q, M, L, D	Value to be rotated
N	WORD	I, Q, M, L, D	Number of bit positions by which the value will be rotated
OUT	DWORD	I, Q, M, L, D	Result of the rotate instruction
ENO	BOOL	I, Q, M, L, D	Enable output

#### Description

A signal state of 1 at the Enable input (EN) activates the **Rotate Right Double Word** instruction. This instruction rotates the entire contents of input IN bit by bit to the right. Input N specifies the number of bits by which the value will be rotated. If N is higher than 32, the double word is rotated  $((N-1) \text{ modulo } 32) + 1$  places. The value of N can be between 0 and 31. The bit positions at the left are filled with the signal states of the bits rotated. The result of the rotate operation can be scanned at output OUT.

The operation triggered by this instruction always resets the CC0 and OV bits of the status word to 0 if N is not equal to zero.

ENO has the same signal state as EN.

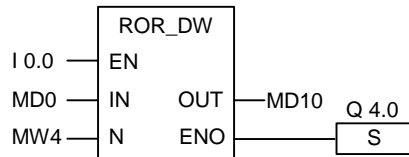




### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	X	X	X	X	-	X	X	X	1

### Example



The instruction is activated if the signal state of I0.0 is 1. Memory double word MD0 is rotated to the right by the number of bits specified in memory word MW4.

The result is entered in memory double word MD10. Output Q4.0 is set to 1.



# 12 Status Bit Instructions

## 12.1 Overview of Status Bit Instructions

### Description

The status bit instructions are bit logic instructions that work with the bits of the status word (see CPU Registers). Each of these instructions reacts to one of the following conditions that is indicated by one or more bits of the status word:

- The binary result bit (BR) is set (has a signal state of 1).
- The result of a math function is relative to 0 in one of the following ways: == 0, <> 0, > 0, < 0, >= 0, <= 0.
- The result of a math function is unordered (UO).
- A math function produced an overflow (OV) or a or a stored overflow (OS).

When a status bit instruction is connected in series, it combines the result of its signal state check with the previous result of logic operation according to the And truth table. When a status bit instruction is connected in parallel, it combines its result with the previous RLO according to the Or truth table.

### Status word

The status word is a Register in the memory of your CPU that contains bits that you can reference in the address of bit and word logic instructions. Structure of the status word:

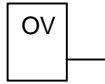
$2^{15}$ ...	...	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
			BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC

You can evaluate the bits in the status word

- by Integer Math Functions
- by Floating-Point Functions

## 12.2 OV : Exception Bit Overflow

### Symbol



### Description

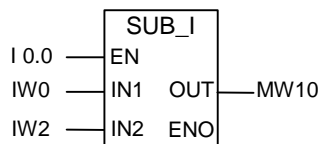
You can use the **Exception Bit Overflow** instruction to detect an overflow (OV) in the last math function. If, after the system executes a math function, the result is outside the permitted negative range or outside the permitted positive range, the OV bit in the status word (see also CPU Registers) is set. The instruction checks the signal state of this bit. This bit is reset if the math functions were free of errors

### Status Word

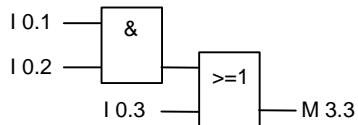
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example

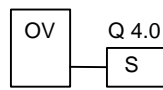
Network 1



Network 2



Network 3

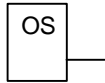


If the signal state at input I0.0 is 1, the SUB\_I box is activated. If the result of the math function input word IW0 minus input word IW2 is outside the permitted range for an integer, the OV bit in the status word is set. The result of a signal state check at OV is 1. Output Q4.0 is set if the check at OV is 1 and the RLO of network 2 is 1 (if the RLO prior to output Q4.0 is 1).

If the signal state of input I0.0 is 0 (not activated), the signal state of both EN and ENO is 0. If the signal state of EN is 1 (activated) and the result of the math function is out of range, the signal state of ENO is 0.

## 12.3 OS : Exception Bit Overflow Stored

### Symbol



### Description

In an AND operation, this instruction combines the result of its check with the previous result of logic operation according to the AND truth table. In an OR operation, the OR truth table is used.

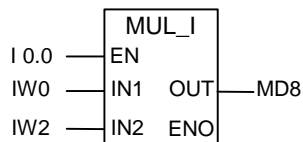
You can use the **Exception Bit Overflow Stored** instruction to recognize a previous overflow (overflow stored, OS) in a math function. If, after the system executes a math function, the result is outside the permitted negative range or outside the permitted positive range, the OS bit in the status word (see also CPU Registers) is set. The instruction checks the signal state of this bit. Unlike the OV (overflow) bit, the OS bit remains set even if later math functions were executed free of errors (see also OV : Exception Bit Overflow ).

### Status Word

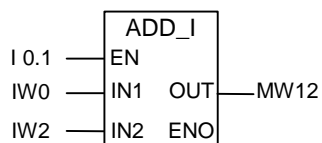
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example

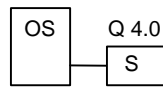
Network 1



Network 2



### Network 3



If the signal state at input I0.0 is 1, the MUL\_I box is activated. If the signal state at input I0.1 is 1, the ADD\_I box is activated. If the result of one of the math functions is outside the permissible range for an integer, the OS bit in the status word is set.

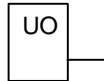
The result of a signal state check at OS is 1 and output Q4.0 is set.

Network 1: if the signal state of input I0.0 is 0 (not activated), the signal state of both EN and ENO is 0. If the signal state of EN is 1 (activated) and the result of the math function is out of range, the signal state of ENO is 0.

Network 2: if the signal state of input I0.1 is 0 (not activated), the signal state of both EN and ENO is 0. If the signal state of EN is 1 (activated) and the result of the math function is out of range, the signal state of ENO is 0.

## 12.4 UO : Exception Bit Unordered

### Symbol



### Description

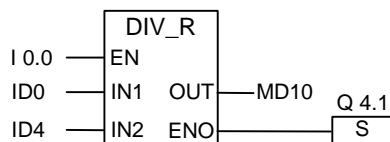
You can use the **Exception Bit Unordered** instruction to check whether or not the result of a floating-point math function is unordered (in other words, whether one of the values in the math function is not a valid floating-point number). The condition code bits of the status word (CC 1 and CC 0, see CPU Registers) are evaluated. If the result of the math function is unordered (UO) the signal state check produces a result of 1. If the combination in CC 1 and CC 0 does not indicate unordered, the result of the signal state check is 0.

### Status Word

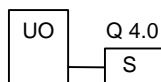
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

### Example

Network 1



Network 2



If the signal state at input I0.0 is 1, the DIV\_R box is activated. If the value of either input double word ID0 or ID4 is not a valid floating-point number, the floating-point math function is unordered.

If the signal state of EN is 1 (activated) and an error occurs while the instruction is being executed, the signal state of ENO is 0.

Output Q4.0 is set if the function DIV\_R is executed, but one of the values in the math function is not a valid floating-point number. If the signal state of input I0.0 is 0 (not activated), the signal state of both EN and ENO is 0.



## 12.5 BR : Exception Bit BR Memory

### Symbol

English



German



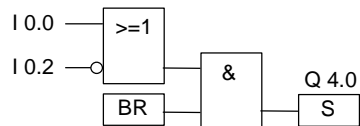
### Description

You can use the **Exception Bit BR Memory** instruction to check the signal state of the BR bit (Binary Result) of the status word (see CPU Registers).

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1







### Example



Output Q4.0 is set if the signal state at input I0.0 is 1 OR the signal state at input I0.2 is 0, and, in addition to this RLO, the signal state of the BR bit is 1.

## 12.6 <> 0 : Result Bits

### Symbols

	The Result Bit instruction <b>for equal to 0</b> determines whether or not the result of a math instruction is equal to 0.
	The Result Bit instruction <b>for not equal to 0</b> determines whether or not the result of a math instruction is not equal to 0.
	The Result Bit instruction <b>for greater than 0</b> determines whether or not the result of a math instruction is greater than 0.
	The Result Bit instruction <b>for less than 0</b> determines whether or not the result of a math instruction is less than 0.
	The Result Bit instruction <b>for greater than or equal to 0</b> determines whether or not the result of a math instruction greater than or equal to 0.
	The Result Bit instruction <b>for less than or equal to 0</b> determines whether or not the result of a math instruction is less than or equal to 0.

### Description

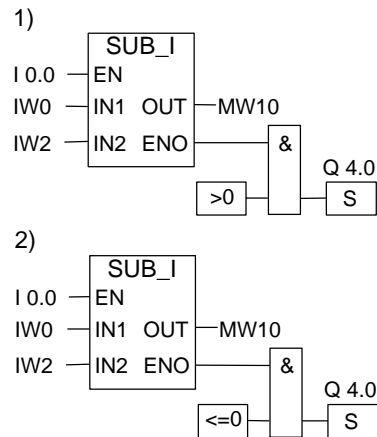
You can use the **Result Bit** instructions to determine the relationship of the result of a math function to zero, in other words, whether the result is ==0, <>0; >0, <0, >=0 OR <=0. The condition code bits of the status word (CC 1 and CC 0, see CPU Registers) are evaluated. If the comparison condition indicated in the address is fulfilled, the result of this signal state check is 1.

In an AND operation, this instruction combines the result of its check with the previous result of logic operation (RLO) according to the AND truth table. In an OR operation, this instruction combines the result of its check with the previous RLO according to the OR truth table.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

## Example



If the signal state at input I0.0 is 1, the SUB\_I box is activated. If the value of input word IW0 is greater than the value of input word IW2, the result of the math function  $IW0 - IW2$  is greater than 0. If the signal state of EN is 1 (activated) and an error occurs while the instruction is being executed, the signal state of ENO is 0.

1. Output Q4.0 is set if the function is executed correctly and the result is less than or equal to 0. If the signal state of input I0.0 is 0 (not activated), the signal state of both EN and ENO is 0.
2. Output Q4.0 is set if the function is executed correctly and the result is less than or equal to 0. If the signal state of input I0.0 is 0 (not activated), the signal state of both EN and ENO is 0.



# 13 Timer Instructions

## 13.1 Overview of Timer Instructions

### Description

You can find information for setting and selecting the correct time under "Location of a Timer in Memory and Components of a Timer".

The following timer instructions are available:

- S\_PULSE : Assign Pulse Timer Parameters and Start
- S\_PEXT : Assign Extended Pulse Timer Parameters and Start
- S\_ODT : Assign On-Delay Timer Parameters and Start
- S\_ODTS : Assign Retentive On-Delay Timer Parameters and Start
- S\_OFFDT : Assign Off-Delay Timer Parameters and Start
- SP : Start Pulse Timer
- SE : Start Extended Pulse Timer
- SD : Start On-Delay Timer
- SS : Start Retentive On-Delay Timer
- SF : Start Off-Delay Timer

## 13.2 Memory Areas and Components of a Timer

### Memory Area

Timers have an area reserved for them in the memory of your CPU. This memory area reserves one 16-bit word for each timer address. When you program in FBD, 256 timers are supported. Please refer to your CPU's technical information to check the number of timer words available.

The following functions access the timer memory area:

- Timer instructions
- Updating of timer words by the clock timing. In the RUN mode, this CPU function decrements a given time value by one unit at the interval specified by the time base until the time value is zero.

### Time Value

Bits 0 through 9 of the timer word contain the time value in binary code. The time value specifies a number of units. When the timer is updated, the time value is decremented by one unit at intervals specified by the time base. The time value is decremented until it is equal to zero.

You can pre-load a time value using either of the following formats:

- **S5T#aH\_bM\_cS\_dMS**
  - Where H = hours, M = minutes, S = seconds, and MS = milliseconds; a, b, c, d are defined by the user.
  - The time base is selected automatically, and the value is rounded to the next lower number with that time base.

The maximum time value that you can enter is 9,990 seconds, or 2H\_46M\_30S.

S5TIME#4S = 4 seconds

s5t#2h\_15m = 2 hours and 15 minutes

S5T#1H\_12M\_18S = 1 hour, 12 minutes, and 18 seconds

### Time Base

Bits 12 and 13 of the timer word contain the time base in binary code. The time base defines the interval at which the time value is decremented by one unit. The smallest time base is 10 ms; the largest is 10 s.

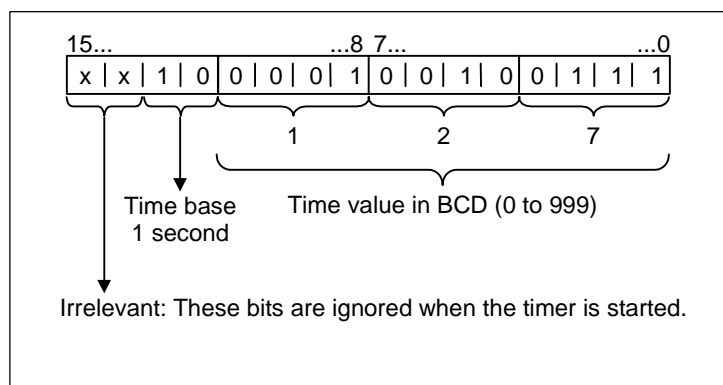
Time Base	Binary Code for the Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

Because time values are saved at only one time interval, values that are not exact multiples of a time interval are truncated. Values with a resolution too high for the required range are rounded down to within the required range but not to the desired resolution. The following table shows the possible resolutions and the corresponding ranges.

Resolution	Time Base
0.01 seconds	10MS to 9S_990MS
0.1 seconds	100MS to 1M_39S_900MS
1 second	1S to 16M_39S
10 seconds	10S to 2HR_46M_30S

### Bit Configuration in the Timer Cell

When a timer is started, the contents of the timer cell are used as the time value. Bits 0 through 11 of the timer cell contain the time value in binary coded decimal format (BCD format: each group of four bits contains the binary code for one decimal value). Bits 12 and 13 contain the time base in binary code. The following Figure shows the contents of the timer cell loaded with timer value 127 with a time base of 1 second.

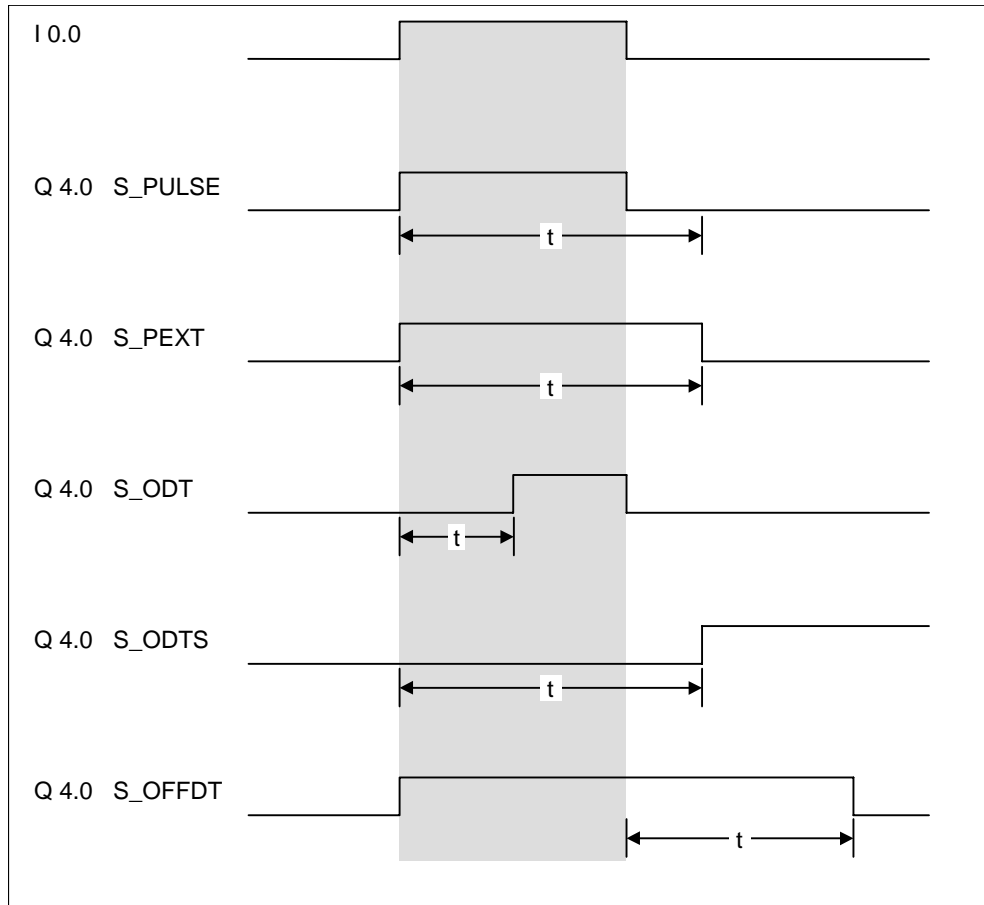


### Reading the Time and the Time Base

Each timer box provides two outputs, BI and BCD, for which you can specify a word location. The BI output provides the time value in binary format, the time base is not displayed. The BCD output provides the time base and the time value in binary coded decimal (BCD) format.

### Choosing the right Timer

This overview is intended to help you choose the right timer for your timing job.

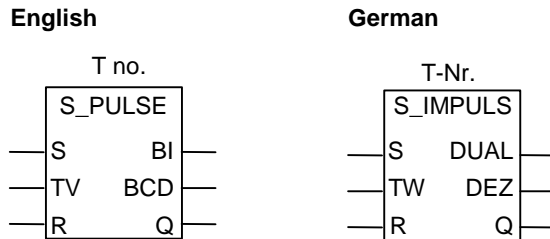


Timer	Description
<b>S_PULSE</b> Pulse timer	The maximum time that the output signal remains at 1 is the same as the programmed time value $t$ . The output signal stays at 1 for a shorter period if the input signal changes to 0.
<b>S_PEXT</b> Extended pulse timer	The output signal remains at 1 for the programmed length of time, regardless of how long the input signal stays at 1.
<b>S_ODT</b> On-delay timer	The output signal changes to 1 only when the programmed time has elapsed and the input signal is still 1.
<b>S_ODTS</b> Retentive on-delay timer	The output signal changes from 0 to 1 only when the programmed time has elapsed, regardless of how long the input signal stays at 1.
<b>S_OFFDT</b> Off-delay timer	The output signal changes to 1 when the input signal changes to 1 or while the timer is running. The time is started when the input signal changes from 1 to 0.



### 13.3 S\_PULSE : Assign Pulse Timer Parameters and Start

#### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	TIMER	T	Timer identification number. The range depends on the CPU.
S	S	BOOL	I, Q, M, D, L, T, C	Start input
TV	TW	S5TIME	I, Q, M, D, L or constant	Preset time value (range 0-9999)
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
BI	DUAL	WORD	I, Q, M, D, L	Time remaining (value in integer format)
BCD	DEZ	WORD	I, Q, M, D, L	Time remaining (value in BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the timer

#### Description

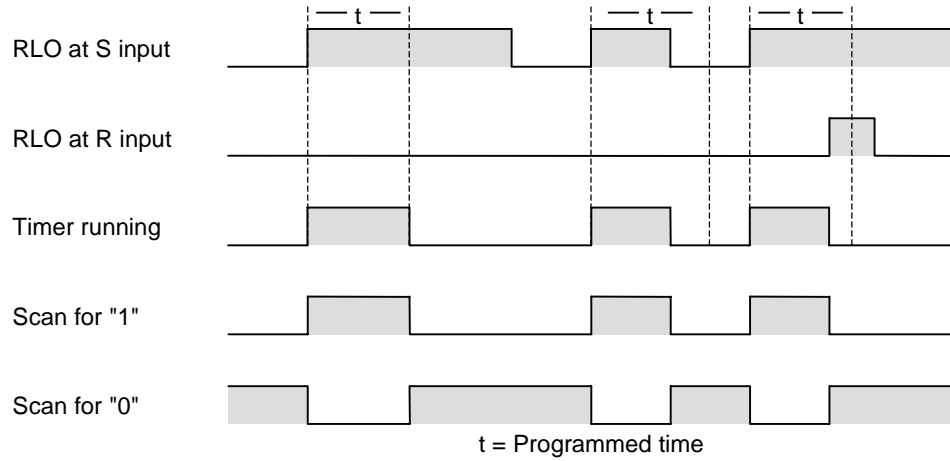
The **Assign Pulse Timer Parameters and Start** instruction starts a specified timer if there is a rising edge (a change in signal state from 0 to 1) at the Start (S) input. A signal change is always necessary to start a timer. The timer continues to run for the time specified at the Time Value (TV) input until the programmed time elapses, as long as the signal state at input TV is 1. While the timer is running, a signal state check for 1 at output Q produces a result of 1. If there is a change from 1 to 0 at the S input before the time has elapsed, the timer is stopped. Then a signal state check for 1 at output Q produces a result of 0.

While the timer is running, a change from 0 to 1 at the Reset (R) input of the timer resets the timer. This change also resets the time and the time base to zero. A signal state of 1 at the R input of the timer has no effect if the timer is not running.

The current time value can be scanned at outputs BI and BCD. The time value at BI is in binary format; at BCD it is in binary coded decimal format.

### Timing Diagram

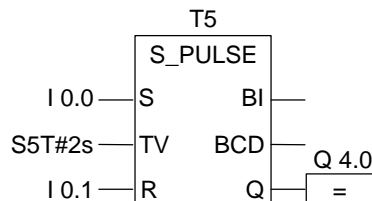
Pulse timer characteristics:



### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

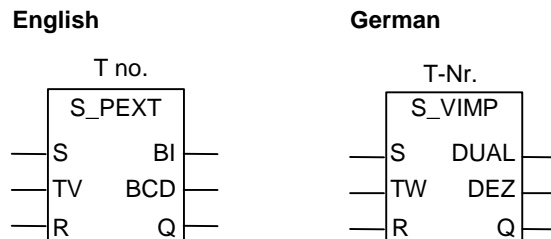
### Example



If the signal state of input I0.0 changes from 0 to 1 (if there is a rising edge in the RLO), timer T5 is started. The timer continues to run with the specified time of two seconds (2s) as long as input I0.0 is 1. If the signal state of input I0.0 changes from 1 to 0 before the time elapses, the timer is stopped. If the signal state of input I0.1 changes from 0 to 1 while the timer is running, the timer is reset. The signal state of output Q4.0 is 1 as long as the timer is running.

## 13.4 S\_PEXT : Assign Extended Pulse Timer Parameters and Start

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	TIMER	T	Timer identification number. The range depends on the CPU.
S	S	BOOL	I, Q, M, D, L, T, C	Start input
TV	TW	S5TIME	I, Q, M, D, L or constant	Preset time value (range 0-9999)
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
BI	DUAL	WORD	I, Q, M, D, L	Time remaining (value in integer format)
BCD	DEZ	WORD	I, Q, M, D, L	Time remaining (value in BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the timer

### Description

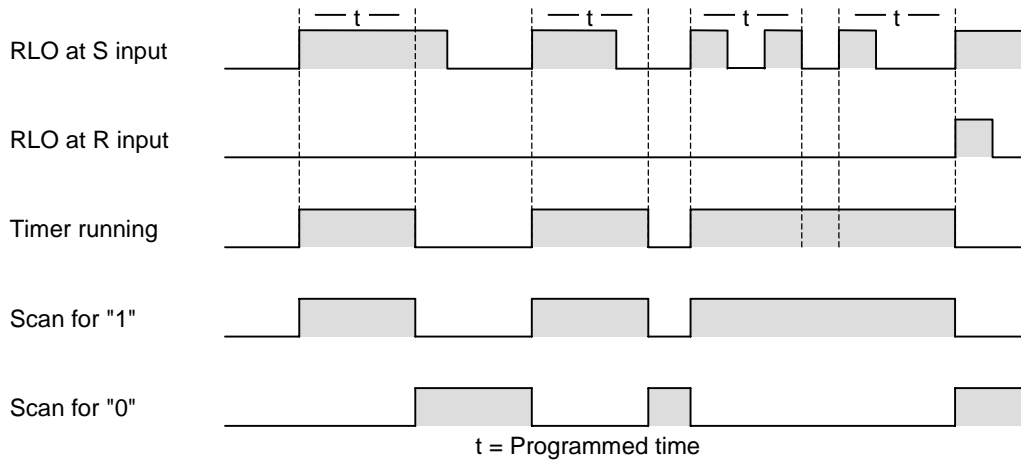
The **Assign Extended Pulse Timer Parameters and Start** instruction starts a specified timer if there is a rising edge (change in signal state from 0 to 1) at the Start (S) input. A signal change is always necessary to start a timer. The timer continues to run for the time specified at the Time Value (TV) input, even if the signal state at the S input changes to 0 before the time has elapsed. A signal state check for 1 at output Q produces a result of 1 as long as the timer is running. The timer is restarted with the specified time if the signal state at input S changes from 0 to 1 while the timer is running.

A change from 0 to 1 at the Reset (R) input of the timer while the timer is running resets the timer. This change also resets the time and the time base to zero.

The current time value can be scanned at the outputs BI and BCD. The time value at BI is in binary format; at BCD it is in binary coded decimal format.

### Timing Diagram

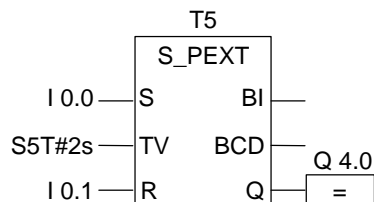
Extended pulse timer characteristics:



### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

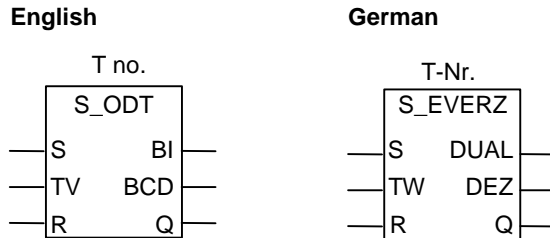
### Example



If the signal state of input I0.0 changes from 0 to 1 (rising edge in the RLO), timer T5 is started. The timer continues to run for the specified time of two seconds (2s) regardless of a falling edge at input S. If the signal state of input I0.0 changes from 0 to 1 before the time has elapsed, the timer is restarted. If the signal state of input I0.1 changes from 0 to 1 while the timer is running, the timer is reset. The signal state of output Q4.0 is 1 as long as the timer is running.

## 13.5 S\_ODT : Assign On-Delay Timer Parameters and Start

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	TIMER	T	Timer identification number. The range depends on the CPU.
S	S	BOOL	I, Q, M, D, L, T, C	Start input
TV	TW	S5TIME	I, Q, M, D, L or constant	Preset time value (range 0-9999)
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
BI	DUAL	WORD	I, Q, M, D, L	Time remaining (value in integer format)
BCD	DEZ	WORD	I, Q, M, D, L	Time remaining (value in BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the timer

### Description

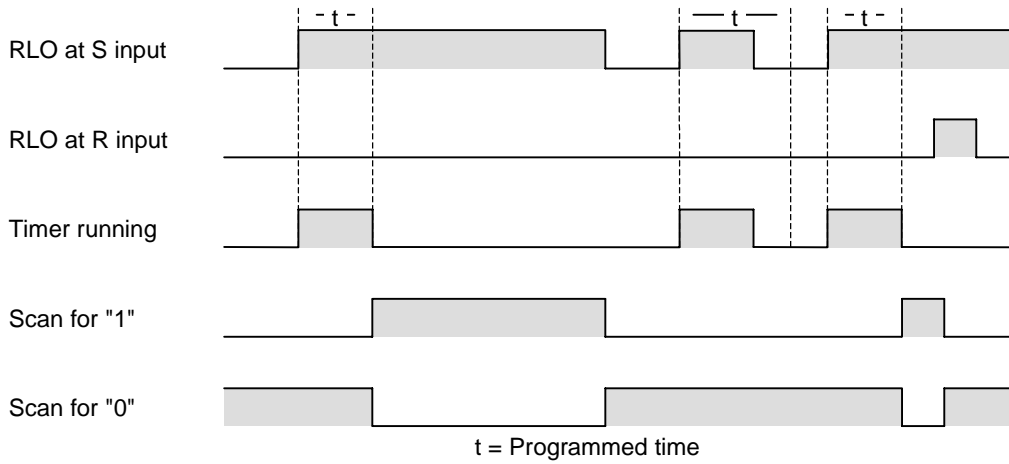
The **Assign On-Delay Timer Parameters and Start** instruction starts a specified timer if there is a rising edge (change in signal state from 0 to 1) at the Start (S) input. A signal change is always necessary to start a timer. The timer continues to run for the time specified at the Time Value (TV) input as long as the signal state at input S is 1. A signal state check for 1 at output Q produces a result of 1 when the time has elapsed without error and when the signal state at input S is still 1. When the signal state at input S changes from 1 to 0 while the timer is running, the timer is stopped. In this case, a signal state check for 1 at output Q always produces the result 0.

A change from 0 to 1 at the Reset (R) input of the timer while the timer is running resets the timer. This change also resets the time and the time base to zero. The timer is also reset if the signal state is 1 at the R input while the timer is not running.

The current time value can be scanned at the outputs BI and BCD. The time value at BI is in binary format; at BCD it is in binary coded decimal format.

### Timing Diagram

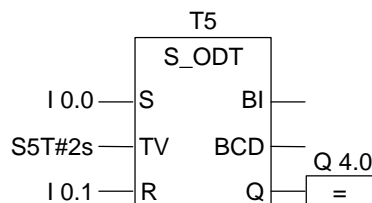
On-Delay timer characteristics:



### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	X	X	X	1

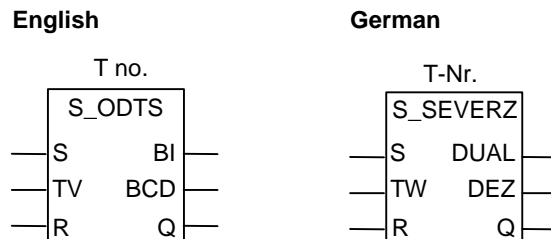
### Example



If the signal state of input I0.0 changes from 0 to 1 (rising edge in the RLO), timer T5 is started. If the specified time of two seconds (2s) elapses and the signal state of input I0.0 is still 1, the signal state of output Q4.0 is 1. If the signal state of input I0.0 changes from 1 to 0, the timer is stopped and output Q4.0 is 0. If the signal state of input I0.1 changes from 0 to 1 while the timer is running, the timer is restarted.

## 13.6 S\_ODTS : Assign Retentive On-Delay Timer Parameters and Start

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	TIMER	T	Timer identification number. The range depends on the CPU.
S	S	BOOL	I, Q, M, D, L, T, C	Start input
TV	TW	S5TIME	I, Q, M, D, L or constant	Preset time value (range 0-9999)
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
BI	DUAL	WORD	I, Q, M, D, L	Time remaining (value in integer format)
BCD	DEZ	WORD	I, Q, M, D, L	Time remaining (value in BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the timer

### Description

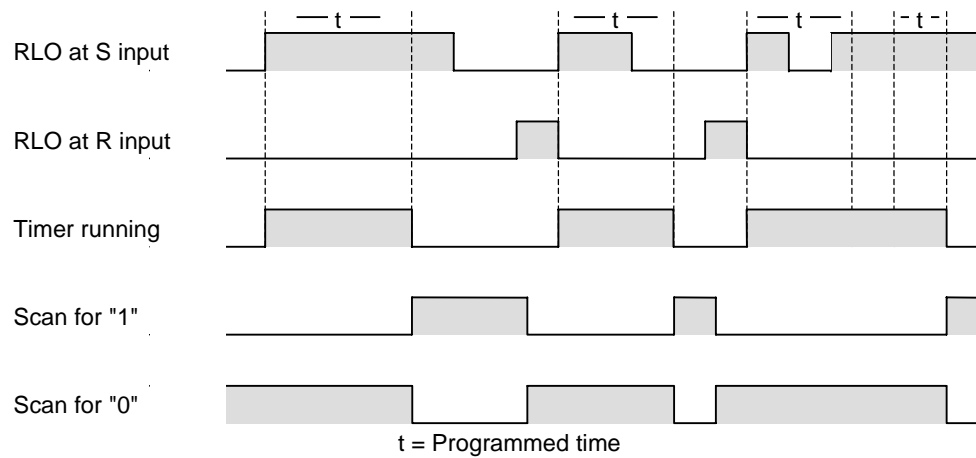
The **Assign Retentive On-Delay Timer Parameters and Start** instruction starts a specified timer if there is a rising edge (change in signal state from 0 to 1) at the Start (S) input. A signal change is always necessary to start a timer. The timer continues to run for the time specified at the Time Value (TV) input, even if the signal state at input S changes to 0 before the timer has expired. A signal state check for 1 at output Q produces a result of 1 when the time has elapsed, regardless of the signal state at input S when the reset input (R) remains at 0. The timer is restarted with the specified time if the signal state at input S changes from 0 to 1 while the timer is running.

A change from 0 to 1 at the Reset (R) input of the timer resets the timer regardless of the RLO at the S input.

The current time value can be scanned at the outputs BI and BCD. The time value at BI is in binary format; at BCD it is in binary coded decimal format.

### Timing Diagram

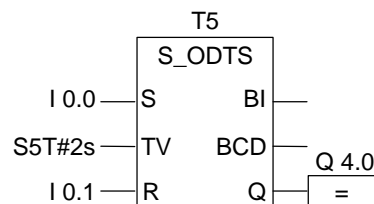
Retentive On-Delay timer characteristics:



### Status word

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
writes:	-	-	-	-	-	x	x	x	1

### Example

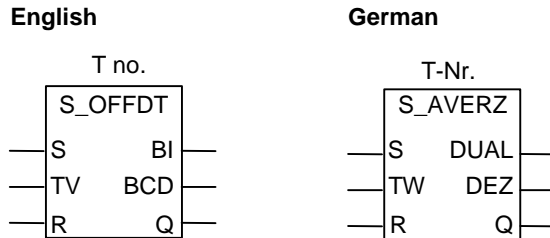


If the signal state of input I0.0 changes from 0 to 1 (rising edge in the RLO), timer T5 is started. The timer continues to run regardless of a signal change at input I0.0 from 1 to 0. If the signal state of input I0.0 changes from 0 to 1 before the time has elapsed, the timer is restarted. If the signal state of input I0.1 changes from 0 to 1 while the timer is running, the timer is restarted. The signal state of output Q4.0 is 1 if the time has elapsed and the signal state at I0.1 remains 0.



## 13.7 S\_OFFDT : Assign Off-Delay Timer Parameters and Start

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
no.	Nr.	TIMER	T	Timer identification number. The range depends on the CPU.
S	S	BOOL	I, Q, M, D, L, T, C	Start input
TV	TW	S5TIME	I, Q, M, D, L or constant	Preset time value (range 0-9999)
R	R	BOOL	I, Q, M, D, L, T, C	Reset input
BI	DUAL	WORD	I, Q, M, D, L	Time remaining (value in integer format)
BCD	DEZ	WORD	I, Q, M, D, L	Time remaining (value in BCD format)
Q	Q	BOOL	I, Q, M, D, L	Status of the timer

### Description

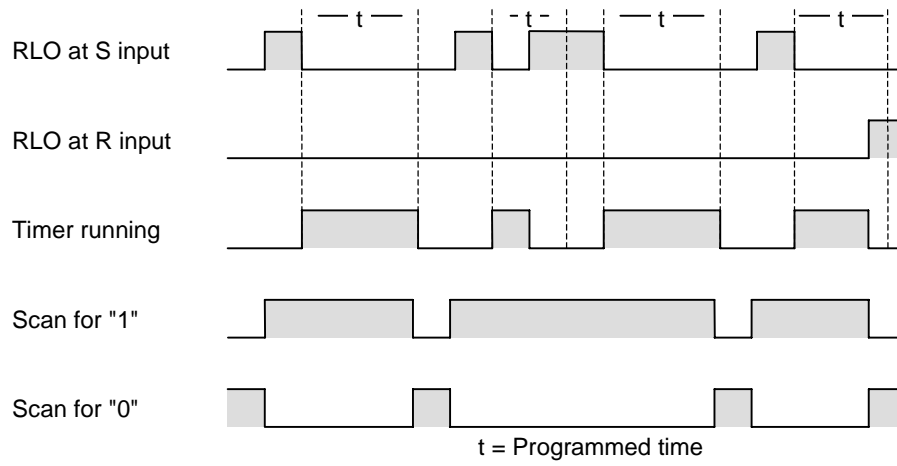
The **Assign Off-Delay Timer Parameters and Start** instruction starts a specified timer if there is a falling edge (change in signal state from 1 to 0) at the Start (S) input. A signal change is always necessary to start a timer. The result of a signal state check for 1 at output Q is 1 when the signal state at the S input is 1 or when the timer is running. The timer is reset when the signal state at input S changes from 0 to 1 while the timer is running. The timer is not restarted until the signal state at input S changes again from 1 to 0.

A change from 0 to 1 at the Reset (R) input of the timer while the timer is running resets the timer.

The actual time value can be scanned at the outputs BI and BCD. The time value at BI is in binary format; at BCD it is in binary coded decimal format.

### Timing Diagram

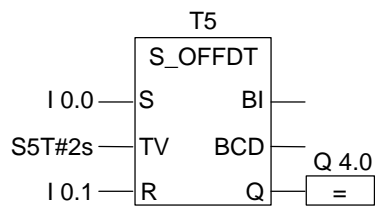
Off-Delay timer characteristics:



### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	x	x	x	1

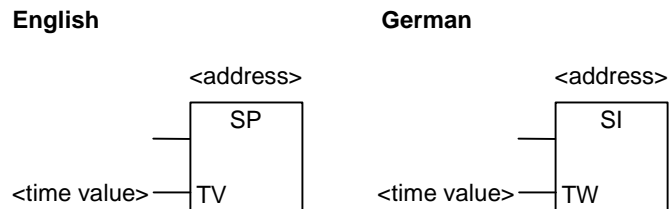
### Example



If the signal state at input I0.0 changes from 1 to 0, the timer is started. Output Q4.0 is 1 when I0.0 is 1 or the timer is running. If the signal state at I0.1 changes from 0 to 1 while the timer is running, the timer is reset.

## 13.8 SP : Start Pulse Timer

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
Timer no.	Timer no.	TIMER	T	The address specifies the number of the timer which is to be started.
TV	TW	S5TIME	E, A, M, D, L or constant	Timer value (S5TIME format)

### Description

The **Start Pulse Timer** instruction starts the specified timer if there is a rising edge (change in signal state from 0 to 1) in the RLO. As long as the RLO is positive, the timer continues running with the specified value. The result of a signal state check for 1 is 1 when the timer is running. If the RLO changes from 1 to 0 before the timer has elapsed, the timer is stopped. In this case the result of the signal state check for 1 is 0.

You will find more information about memory areas and components of a timer in Memory Areas.

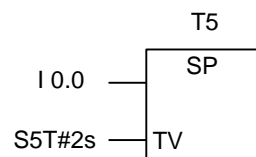
You can only place the box **Start Pulse Timer** at the right-hand end of a logic string. You can use a number of **Start Pulse Timer** boxes.

### Status Word

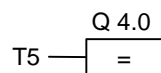
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example

Network 1



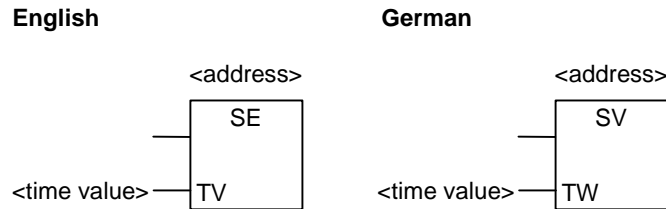
Network 2



If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the timer T5 is started. As long as the signal state is 1, the timer continues to run with the specified value of 2 seconds. If the signal state at I0.0 changes from 1 to 0 while the timer is running, the timer is stopped. Output Q4.0 is 1 when the timer is running.

## 13.9 SE : Start Extended Pulse Timer

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
Timer no.	Timer no.	TIMER	T	The address specifies the number of the timer which is to be started.
TV	TW	S5TIME	E, A, M, D, L or constant	Timer value (S5TIME format)

### Description

The **Start Extended Pulse Timer** instruction starts the specified timer if there is a rising edge (change in signal state from 0 to 1) in the RLO. The timer continues running with the specified value if the RLO changes to 0 before the timer elapses. The result of a signal state check for 1 is 1 when the timer is running. If the RLO changes from 0 to 1 while the timer is running, the timer is restarted with the specified time.

You will find more information about memory areas and components of a timer in Memory Areas.

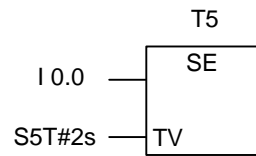
You can only place the box **Start Extended Pulse Timer** at the right-hand end of a logic string. You can use a number of **Start Extended Pulse Timer** boxes.

### Status Word

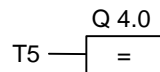
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example

Network 1



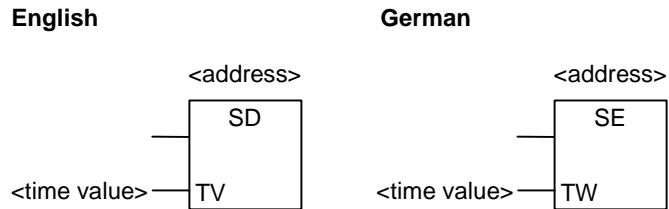
Network 2



If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the timer T5 is started. The timer continues to run without being influenced by a falling edge in the RLO. If the signal state at I0.0 changes from 0 to 1 before the specified time has elapsed, the timer is restarted. Output Q4.0 is 1 when the timer is running.

## 13.10 SD : Start On-Delay Timer

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
Timer no.	Timer no.	TIMER	T	The address specifies the number of the timer which is to be started.
TV	TW	S5TIME	E, A, M, D, L or constant	Timer value (S5TIME format)

### Description

The **Start On-Delay Timer** instruction starts the specified timer if there is a rising edge (change in signal state from 0 to 1) in the RLO. The result of a signal state check for 1 is 1 if the specified time has elapsed without error and the RLO still has the value 1. If the RLO changes from 1 to 0 while the timer is running, the timer is stopped. In this case the result of the signal state check for 1 is always 0.

You will find more information about memory areas and components of a timer in Memory Areas.

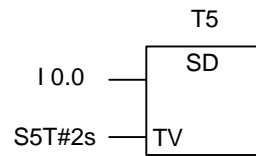
You can only place the box **Start On-Delay Timer** at the right-hand end of a logic string. You can use a number of **Start On-Delay Timer** boxes.

### Status Word

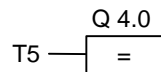
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example

Network 1



Network 2



If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the timer T5 is started. If the signal state at I0.0 is still 1 when the timer elapses, the output Q4.0 is 1. If the signal state changes from 1 to 0, the timer is stopped.



## 13.11 SS : Start Retentive On-Delay Timer

### Symbol



Parameter English	Parameter German	Data Type	Memory Area	Description
Timer no.	Timer no.	TIMER	T	The address specifies the number of the timer which is to be started.
TV	TW	S5TIME	E, A, M, D, L or constant	Timer value (S5TIME format)

### Description

The **Start Retentive On-Delay Timer** instruction starts the specified timer if there is a rising edge (change in signal state from 0 to 1) in the RLO. The timer continues running with the specified value if the RLO changes to 0 before the timer elapses. The result of a signal state check for 1 is 1 if the timer has elapsed independent of the RLO. If the RLO changes from 0 to 1 while the timer is running, the timer is restarted with the specified time.

You will find more information about memory areas and components of a timer in Memory Areas.

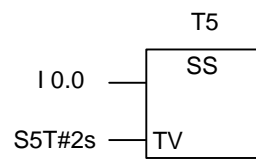
You can only place the box **Start Retentive On-Delay Timer** at the right-hand end of a logic string. You can use a number of **Start Retentive On-Delay Timer** boxes.

### Status Word

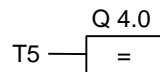
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example

Network 1



Network 2

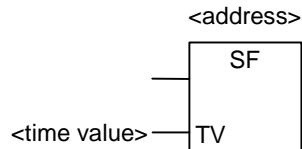


If the signal state at input I0.0 changes from 0 to 1 (rising edge in the RLO), the timer T5 is started. The timer continues to run independent of whether the signal state at input I0.0 changes from 1 to 0. If the signal state changes from 0 to 1 before the specified time has elapsed, the timer is restarted. Output Q4.0 is 1 when the timer has elapsed.

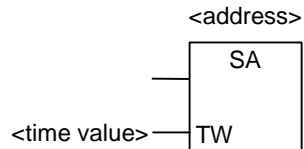
## 13.12 SF Start Off-Delay Timer

### Symbol

English



German



Parameter English	Parameter German	Data Type	Memory Area	Description
Timer no.	Timer no.	TIMER	T	The address specifies the number of the timer which is to be started.
TV	TW	S5TIME	E, A, M, D, L or constant	Timer value (S5TIME format)

### Description

The **Start Off-Delay Timer** instruction starts the specified timer if there is a falling edge (change in signal state from 1 to 0) in the RLO. The result of a signal state check for 1 is 1 if the RLO is 1 or if the timer is running. If the RLO changes from 0 to 1 while the timer is running, the timer is reset. The timer is then restarted when the RLO changes from 1 to 0.

You will find more information about memory areas and components of a timer in Memory Areas.

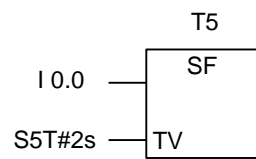
You can only place the box **Start Off-Delay Timer** at the right-hand end of a logic string. You can use a number of **Start Off-Delay Timer** boxes.

### Status Word

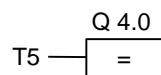
	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	-	-	-	-	-	0	-	-	0

### Example

Network 1



Network 2



If the signal state at input I0.0 changes from 1 to 0, the timer is started.

If the signal state changes from 0 to 1, the timer is reset.

The signal state at output Q4.0 is 1 if the signal state at input I0.0 is 1 or the timer is running.

# 14 Word Logic Instructions

## 14.1 Overview of Word Logic Instructions

### Description

Word logic instructions compare pairs of words (16 bits) and double words (32 bits) bit by bit, according to Boolean logic.

The value of the result at output OUT relative to 0 affects the CC1 bit in the status word as follows:

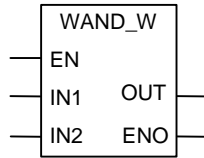
- If the result at output OUT is not equal to 0, the CC1 bit in the status word is set to 1.
- If the result at output OUT is 0, the CC1 bit in the status word is set to 0.

The following instructions are available for performing word logic operations:

- WAND\_W : AND Word (Word)
- WOR\_W : OR Word (Word)
- WXOR\_W : Exclusive OR Word (Word)
- WAND\_DW : AND Double Word (Word)
- WOR\_DW : OR Double Word (Word)
- WXOR\_DW : Exclusive OR Double Word (Word)

## 14.2 WAND\_W : AND Word (Word)

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	WORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	WORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	WORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

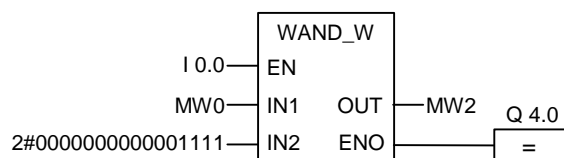
### Description

The **(Word) AND Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the AND truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT. ENO has the same signal state as EN.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

### Example



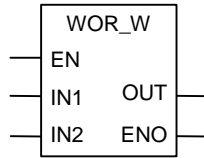
The instruction is activated when the signal state of I0.0 is 1. Only bits 0 to 3 are relevant, all other bits of MW0 are masked.

IN1 = 0101010101010101  
 IN2 = 0000000000001111  
 OUT = 0000000000000101

Q4.0 is 1 if the instruction is executed.

### 14.3 WOR\_W : OR Word (Word)

#### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	WORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	WORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	WORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

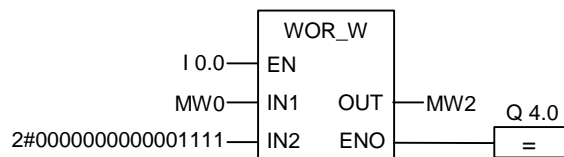
#### Description

The **(Word) OR Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the OR truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT. ENO has the same signal state as EN.

#### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

#### Example



The instruction is activated when I0.0 is 1. The bits in MW0 and in the constant are Ored and bits 0 to 3 set to 1, all other bits of MW0 are entered unchanged in MW2

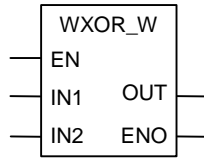
```

IN1      =      0101010101010101
IN2      =      0000000000001111
OUT      =      0101010101011111
    
```

Q4.0 is 1 if the instruction is executed.

## 14.4 WXOR\_W : Exclusive OR Word (Word)

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L	Enable input
IN1	WORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	WORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	WORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

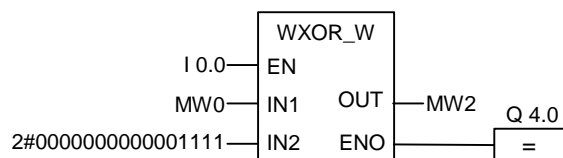
### Description

The **(Word) OR Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the OR truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT. ENO has the same signal state as EN.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

### Example



The instruction is activated when input I0.0 is 1.

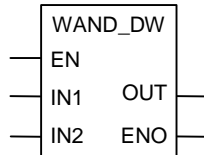
IN1 = 0101010101010101  
 IN2 = 0000000000001111  
 OUT = 0101010101011010

Q4.0 is 1 if the instruction is executed.



## 14.5 WAND\_DW : AND Double Word (Word)

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DWORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	DWORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	DWORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

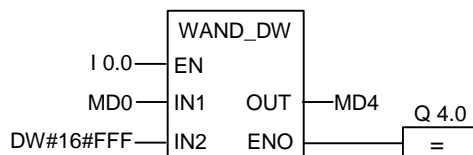
### Description

The **(Word) AND Double Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the AND truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT. ENO has the same signal state as EN.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

### Example



The instruction is activated when I0.0 is 1. Only bits 0 to 11 are relevant, all other bits of MD4 are masked.

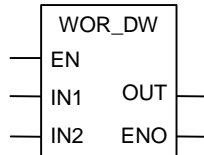
```

IN1      =      0101010101010101      0101010101010101
IN2      =      0000000000000000      0000111111111111
OUT      =      0000000000000000      0000101010101010
    
```

Q4.0 is 1 if the instruction is executed.

## 14.6 WOR\_DW : OR Double Word (Word)

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DWORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	DWORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	DWORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

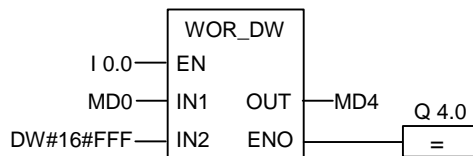
### Description

The **(Word) OR Double Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the OR truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT. ENO has the same signal state as EN.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

### Example



The instruction is activated when I0.0 is 1. The bits in MD0 and in the constant are ORed and bits 0 to 11 set to 1, all other bits of MD0 are entered unchanged in MD4.

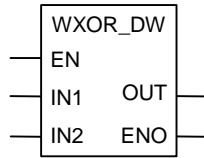
```

IN1      =      0101010101010101    0101010101010101
IN2      =      0000000000000000    0000111111111111
OUT      =      0101010101010101    0101111111111111
    
```

Q4.0 is 1 if the instruction is executed.

## 14.7 WXOR\_DW : Exclusive OR Double Word (Word)

### Symbol



Parameter	Data Type	Memory Area	Description
EN	BOOL	I, Q, M, D, L, T, C	Enable input
IN1	DWORD	I, Q, M, D, L or constant	First value of the logic operation
IN2	DWORD	I, Q, M, D, L or constant	Second value of the logic operation
OUT	DWORD	I, Q, M, D, L	Result of the logic operation
ENO	BOOL	I, Q, M, D, L	Enable output

### Description

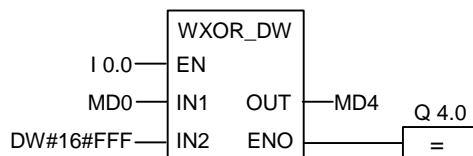
The **(Word) Exclusive OR Double Word** instruction is activated by signal state 1 at the Enable input (EN) and combines the two digital values at inputs IN1 and IN2 bit by bit according to the EXCLUSIVE OR truth table. The values are interpreted as pure bit patterns. The result can be scanned at output OUT.

ENO has the same signal state as EN.

### Status Word

	BR	CC1	CC0	OV	OS	OR	STA	RLO	FC
writes	1	X	0	0	-	X	1	1	1

### Example



The instruction is activated when input I0.0 is 1.

IN1	=	0101010101010101	0101010101010101
IN2	=	0000000000000000	0000111111111111
OUT	=	0101010101010101	0101101010101010

Q4.0 is 1 if the instruction is executed.



# A Overview of All FBD Instructions

## A.1 FBD Instructions Sorted According to German Mnemonics (SIMATIC)

German Mnemonics	English Mnemonics	Program Elements Catalog	Description
&	&	Bit logic instruction	AND Logic Operation
>=1	>=1	Bit logic instruction	OR Logic Operation
=	=	Bit logic instruction	Assign
#	#	Bit logic instruction	Midline Output
---	---	Bit logic instruction	Insert Binary Input
---o	---o	Bit logic instruction	Negate Binary Input
==0	==0	Status bits	Result Bits
>0	>0	Status bits	Result Bits
>=0	>=0	Status bits	Result Bits
<0	<0	Status bits	Result Bits
<=0	<=0	Status bits	Result Bits
<>0	<>0	Status bits	Result Bits
ABS	ABS	Floating point instruction	Forming the Absolute Value of a Floating-Point Number
ACOS	ACOS	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
ADD_DI	ADD_DI	Integer math instruction	Add Double Integer
ADD_I	ADD_I	Integer math instruction	Add Integer
ADD_R	ADD_R	Floating point instruction	Add Real
ASIN	ASIN	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
ATAN	ATAN	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
BCD_DI	BCD_DI	Convert	BCD to Double Integer
BCD_I	BCD_I	Convert	BCD to Integer
BIE	BR	Status bits	Exception Bit BR Memory
CALL	CALL	Program control	Calling an FC/SFC without Parameters
CALL_FB	CALL_FB	Program control	CALL_FB (Call FB as Box)
CALL_FC	CALL_FC	Program control	CALL_FC (Call FC as Box)
CALL_SFB	CALL_SFB	Program control	CALL_SFB (Call System FB as Box)
CALL_SFC	CALL_SFC	Program control	CALL_SFC (Call System FC as Box)
CEIL	CEIL	Convert	Ceiling
CMP ? D	CMP ? D	Compare	Compare Double Integer

German Mnemonics	English Mnemonics	Program Elements Catalog	Description
CMP ? I	CMP ? I	Compare	Compare Integer
CMP ? R	CMP ? R	Compare	Compare Real
COS	COS	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
DI_BCD	DI_BCD	Convert	Double Integer to BCD
DI_R	DI_R	Convert	Double Integer to Real
DIV_DI	DIV_DI	Integer math instruction	Divide Double Integer
DIV_I	DIV_I	Integer math instruction	Divide Integer
DIV_R	DIV_R	Floating point instruction	Divide Real
EXP	EXP	Floating point instruction	Forming the Exponential Value of a Floating-Point Number
FLOOR	FLOOR	Convert	Floor
I_BCD	I_BCD	Convert	Integer to BCD
I_DI	I_DI	Convert	Integer to Double Integer
INV_I	INV_I	Convert	Ones Complement Integer
INV_DI	INV_DI	Convert	Ones Complement Double Integer
JMP	JMP	Jumps	Unconditional Jump in a Block
JMP	JMP	Jumps	Conditional Jump in a Block
JMPN	JMPN	Jumps	Jump-If-Not
LABEL	LABEL	Jumps	Jump Label
LN	LN	Floating point instruction	Forming the Natural Logarithm of a Floating-Point Number
MCR>	MCR>	Program control	Master Control Relay On/Off
MCR<	MCR<	Program control	Master Control Relay On/Off
MCRA	MCRA	Program control	Master Control Relay Activate/Deactivate
MCRD	MCRD	Program control	Master Control Relay Activate/Deactivate
MOD_DI	MOD_DI	Integer math instruction	Return Fraction Double Integer
MOVE	MOVE	Move	Assign Value
MUL_DI	MUL_DI	Integer math instruction	Multiply Double Integer
MUL_I	MUL_I	Integer math instruction	Multiply Integer
MUL_R	MUL_R	Floating point instruction	Multiply Real
N	N	Bit logic instruction	Negative RLO Edge Detection
NEG	NEG	Bit logic instruction	Address Negative Edge Detection
NEG_DI	NEG_DI	Convert	Twos Complement Double Integer
NEG_I	NEG_I	Convert	Twos Complement Integer
NEG_R	NEG_R	Convert	Negate Real Number
OPN	OPN	DB call	Open Data Block
OS	OS	Status bits	Exception Bit Overflow Stored
OV	OV	Status bits	Exception Bit Overflow
P	P	Bit logic instruction	Positive RLO Edge Detection
POS	POS	Bit logic instruction	Address Positive Edge Detection
R	R	Bit logic instruction	Reset Output
RET	RET	Program control	Return
ROL_DW	ROL_DW	Shift/Rotate	Rotate Left Double Word

German Mnemonics	English Mnemonics	Program Elements Catalog	Description
ROUND	ROUND	Convert	Round to Double Integer
ROR_DW	ROR_DW	Shift/Rotate	Rotate Right Double Word
RS	RS	Bit logic instruction	Reset_Set Flip Flop
S	S	Bit logic instruction	Set Output
SA	SF	Timers	Start Off-Delay Timer
SAVE	SAVE	Bit logic instruction	Save RLO to BR Memory
S_AVERZ	S_OFFDT	Timers	Assign Off-Delay Timer Parameters and Start
SE	SD	Timers	Assign On-Delay Timer Parameters and Start
S_EVERZ	S_ODT	Timers	Assign On-Delay Timer Parameters and Start
SHL_DW	SHL_DW	Shift/Rotate	Shift Left Double Word
SHL_W	SHL_W	Shift/Rotate	Shift Left Word
SHR_DI	SHR_DI	Shift/Rotate	Shift Right Double Integer
SHR_DW	SHR_DW	Shift/Rotate	Shift Right Double Word
SHR_I	SHR_I	Shift/Rotate	Shift Right Integer
SHR_W	SHR_W	Shift/Rotate	Shift Right Word
SI	SP	Timers	Start Pulse Timer
S_IMPULS	S_PULSE	Timers	Assign Pulse Timer Parameters and Start
SIN	SIN	Floating point-instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
SQR	SQR	Floating point-instruction	Forming the Square (SQR) of a Floating-Point Number
SQRT	SQRT	Floating point-instruction	Forming the Square Root (SQRT) of a Floating-Point Number
SR	SR	Bit logic instruction	Set_Reset Flip Flop
SS	SS	Timers	Start Retentive On-Delay Timer
S_SEVERZ	S_ODTS	Timers	Assign Retentive On-Delay Timer Parameters and Start
SUB_DI	SUB_DI	Integer math-instruction	Subtract Double Integer
SUB_I	SUB_I	Integer math-instruction	Subtract Integer
SUB_R	SUB_R	Floating point-instruction	Subtract Real
SV	SE	Timers	Start Extended Pulse Timer
S_VIMP	S_PEXT	Timers	Assign Extended Pulse Timer Parameters and Start
SZ	SC	Counters	Set Counter Value
TAN	TAN	Floating point-instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
TRUNC	TRUNC	Convert	Truncate Double Integer Part
UO	UO	Status bits	Exception Bit Unordered
WAND_DW	WAND_DW	Word logic instruction	AND Double Word (Word)
WAND_W	WAND_W	Word logic instruction	AND Word (Word)
WOR_DW	WOR_DW	Word logic instruction	OR Double Word (Word)
WOR_W	WOR_W	Word logic instruction	OR Word (Word)
WXOR_DW	WXOR_DW	Word logic instruction	Exclusive OR Double Word (Word)
WXOR_W	WXOR_W	Word logic instruction	Exclusive OR Word (Word)
XOR	XOR	Bit logic instruction	Exclusive OR Logic Operation

<b>German Mnemonics</b>	<b>English Mnemonics</b>	<b>Program Elements Catalog</b>	<b>Description</b>
ZAEHLER	S_CUD	Counters	Assign Parameters and Count Up/Down
ZR	CD	Counters	Down Counter
Z_RUECK	S_CD	Counters	Assign Parameters and Count Down
ZV	CU	Counters	Up Counter
Z_VORW	S_CU	Counters	Assign Parameters and Count Up



## A.2 FBD Instructions Sorted According to English Mnemonics (International)

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
&	&	Bit logic instruction	AND Logic Operation
>=1	>=1	Bit logic instruction	OR Logic Operation
=	=	Bit logic instruction	Assign
#	#	Bit logic instruction	Midline Output
---	---	Bit logic instruction	Insert Binary Input
---o	---o	Bit logic instruction	Negate Binary Input
==0	==0	Status bits	Result Bits
>0	>0	Status bits	Result Bits
>=0	>=0	Status bits	Result Bits
<0	<0	Status bits	Result Bits
<=0	<=0	Status bits	Result Bits
<>0	<>0	Status bits	Result Bits
ABS	ABS	Floating point instruction	Forming the Absolute Value of a Floating-Point Number
ACOS	ACOS	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
ADD_DI	ADD_DI	Integer math instruction	Add Double Integer
ADD_I	ADD_I	Integer math instruction	Add Double Integer
ADD_R	ADD_R	Floating point instruction	Add Real
ASIN	ASIN	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
ATAN	ATAN	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
BCD_DI	BCD_DI	Convert	BCD to Double Integer
BCD_I	BCD_I	Convert	BCD to Integer
BR	BIE	Status bits	Exception Bit BR Memory
CALL	CALL	Program control	Calling an FC/SFC without Parameters
CALL_FB	CALL_FB	Program control	CALL_FB (Call FB as Box)
CALL_FC	CALL_FC	Program control	CALL_FC (Call FC as Box)
CALL_SFB	CALL_SFB	Program control	CALL_SFB (Call System FB as Box)
CALL_SFC	CALL_SFC	Program control	CALL_SFC (Call System FC as Box)
CD	ZR	Counters	Down Counter
CEIL	CEIL	Convert	Ceiling
CMP ? D	CMP ? D	Compare	Compare Double Integer
CMP ? I	CMP ? I	Compare	Compare Integer
CMP ? R	CMP ? R	Compare	Compare Real
COS	COS	Floating point instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
CU	ZV	Counters	Up Counter
DI_BCD	DI_BCD	Convert	Double Integer to BCD
DI_R	DI_R	Convert	Double Integer to Real

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
DIV_DI	DIV_DI	Integer math instruction	Divide Double Integer
DIV_I	DIV_I	Integer math instruction	Divide Integer
DIV_R	DIV_R	Floating point instruction	Divide Real
EXP	EXP	Floating point instruction	Forming the Exponential Value of a Floating-Point Number
FLOOR	FLOOR	Convert	Floor
I_BCD	I_BCD	Convert	Integer to BCD
I_DI	I_DI	Convert	Integer to Double Integer
INV_I	INV_I	Convert	Ones Complement Integer
INV_DI	INV_DI	Convert	Ones Complement Double Integer
JMP	JMP	Jumps	Unconditional Jump in a Block
JMP	JMP	Jumps	Conditional Jump in a Block
JMPN	JMPN	Jumps	Jump-If-Not
LABEL	LABEL	Jumps	Jump Label
LN	LN	Floating point instruction	Forming the Natural Logarithm of a Floating-Point Number
MCR>	MCR>	Program control	Master Control Relay On/Off
MCR<	MCR<	Program control	Master Control Relay On/Off
MCRA	MCRA	Program control	Master Control Relay Activate/Deactivate
MCRD	MCRD	Program control	Master Control Relay Activate/Deactivate
MOD_DI	MOD_DI	Integer math instruction	Return Fraction Double Integer
MOVE	MOVE	Move	Assign Value
MUL_DI	MUL_DI	Integer math instruction	Multiply Double Integer
MUL_I	MUL_I	Integer math instruction	Multiply Integer
MUL_R	MUL_R	Floating point instruction	Multiply Real
N	N	Bit logic instruction	Negative RLO Edge Detection
NEG	NEG	Bit logic instruction	Address Negative Edge Detection
NEG_DI	NEG_DI	Convert	Twos Complement Double Integer
NEG_I	NEG_I	Convert	Twos Complement Integer
NEG_R	NEG_R	Convert	Negate Real Number
OPN	OPN	DB call	Open Data Block
OS	OS	Status bits	Exception Bit Overflow Stored
OV	OV	Status bits	Exception Bit Overflow
P	P	Bit logic instruction	Positive RLO Edge Detection
POS	POS	Bit logic instruction	Address Positive Edge Detection
R	R	Bit logic instruction	Reset Output
RET	RET	Program control	Return
ROL_DW	ROL_DW	Shift/Rotate	Rotate Left Double Word
ROUND	ROUND	Convert	Round to Double Integer
ROR_DW	ROR_DW	Shift/Rotate	Rotate Right Double Word
RS	RS	Bit logic instruction	Reset_Set Flip Flop
S	S	Bit logic instruction	Set Output
SAVE	SAVE	Bit logic instruction	Save RLO to BR Memory

English Mnemonics	German Mnemonics	Program Elements Catalog	Description
SC	SZ	Counters	Set Counter Value
S_CD	Z_RUECK	Counters	Assign Parameters and Count Down
S_CU	Z_VORW	Counters	Assign Parameters and Count Up
S_CUD	ZAEHLER	Counters	Assign Parameters and Count Up/Down
SD	SE	Timers	Start On-Delay Timer
SE	SV	Timers	Start Extended Pulse Timer
SF	SA	Timers	Start Off-Delay Timer
SHL_DW	SHL_DW	Shift/Rotate	Shift Left Double Word
SHL_W	SHL_W	Shift/Rotate	Shift Left Word
SHR_DI	SHR_DI	Shift/Rotate	Shift Right Double Integer
SHR_DW	SHR_DW	Shift/Rotate	Shift Right Double Word
SHR_I	SHR_I	Shift/Rotate	Shift Right Integer
SHR_W	SHR_W	Shift/Rotate	Shift Right Word
SIN	SIN	Floating point-instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
S_ODT	S_EVERZ	Timers	Assign On-Delay Timer Parameters and Start
S_ODTS	S_SEVERZ	Timers	Assign Retentive On-Delay Timer Parameters and Start
S_OFFDT	S_AVERZ	Timers	Assign Off-Delay Timer Parameters and Start
SP	SI	Timers	Start Pulse Timer
S_PEXT	S_VIMP	Timers	Assign Extended Pulse Timer Parameters and Start
S_PULSE	S_IMPULS	Timers	Assign Pulse Timer Parameters and Start
SQR	SQR	Floating point-instruction	Forming the Square (SQR) of a Floating-Point Number
SQRT	SQRT	Floating point-instruction	Forming the Square Root (SQRT) of a Floating-Point Number
SR	SR	Bit logic instruction	Set_Reset Flip Flop
SS	SS	Timers	Start Retentive On-Delay Timer
SUB_DI	SUB_DI	Integer math-instruction	Subtract Double Integer
SUB_I	SUB_I	Integer math-instruction	Subtract Integer
SUB_R	SUB_R	Floating point-instruction	Subtract Real
TAN	TAN	Floating point-instruction	Forming Trigonometric Functions of Angles as Floating-Point Numbers
TRUNC	TRUNC	Convert	Truncate Double Integer Part
UO	UO	Status bits	Exception Bit Unordered
WAND_DW	WAND_DW	Word logic instruction	AND Double Word (Word)
WAND_W	WAND_W	Word logic instruction	AND Word (Word)
WOR_DW	WOR_DW	Word logic instruction	OR Double Word (Word)
WOR_W	WOR_W	Word logic instruction	OR Word (Word)
WXOR_DW	WXOR_DW	Word logic instruction	Exclusive OR Double Word (Word)
WXOR_W	WXOR_W	Word logic instruction	Exclusive OR Word (Word)
XOR	XOR	Bit logic instruction	Exclusive OR Logic Operation



## B Programming Examples

### B.1 Overview of Programming Examples

#### Practical Applications

Each FBD instruction triggers a specific operation. When you combine these instructions into a program, you can accomplish a wide variety of automation tasks. This chapter provides the following examples of practical applications of the FBD instructions:

- Controlling a conveyor belt using bit logic instructions
- Detecting direction of movement on a conveyor belt using bit logic instructions
- Generating a clock pulse using timer instructions
- Keeping track of storage space using counter and comparison instructions
- Solving a problem using integer math instructions
- Setting the length of time for heating an oven

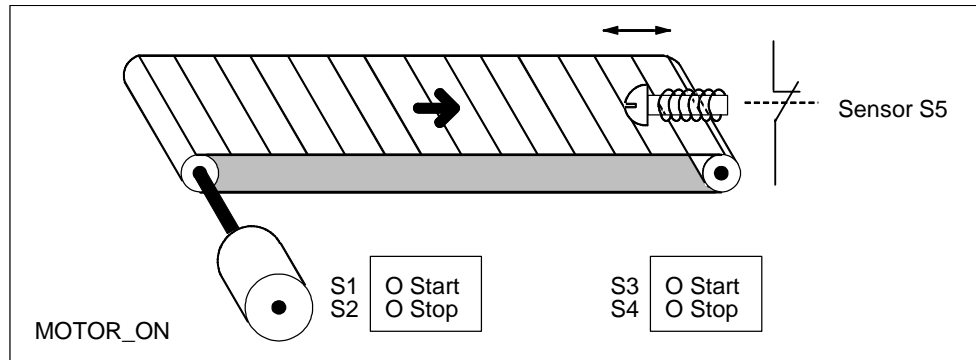
#### Instructions Used

Mnemonic	Program Elements Catalog	Description
WAND_W	Word logic instruction	(Word) And Word
WOR_W	Word logic instruction	(Word) Or Word
S_CD	Counters	Down Counter
S_CU	Counters	Up Counter
R	Bit logic instruction	Reset Output
S	Bit logic instruction	Set Output
P	Bit logic instruction	Positive RLO Edge Detection
ADD_I	Floating-Point instruction	Add Integer
DIV_I	Floating-Point instruction	Divide Integer
MUL_I	Floating-Point instruction	Multiply Integer
CMP >=I, CMP <=I	Compare	Compare Integer
&	Bit logic instruction	AND
>=1	Bit logic instruction	OR
=	Bit logic instruction	Assign
JMPN	Jumps	Jump-If-Not
RET	Program control	Return
MOVE	Move	Assign a Value
SE	Timers	Extended Pulse Timer

## B.2 Example: Bit Logic Instructions

### Example 1: Controlling a Conveyor Belt

The following figure shows a conveyor belt that can be activated electrically. There are two push button switches at the beginning of the belt: S1 for START and S2 for STOP. There are also two push button switches at the end of the belt: S3 for START and S4 for STOP. It is possible to start or stop the belt from either end. Also, sensor S5 stops the belt when an item on the belt reaches the end.



### Absolute and symbolic Programming

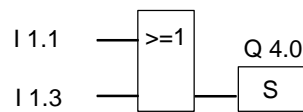
You can write a program to control the conveyor belt using **absolute values** or **symbols** that represent the various components of the conveyor system.

You need to make a symbol table to correlate the symbols you choose with absolute values (see the STEP 7 Online Help).

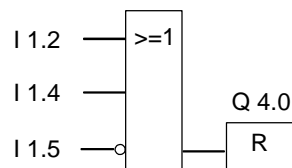
System Component	Absolute Address	Symbol	Symbol Table
Push Button Start Switch	I 1.1	S1	I 1.1 S1
Push Button Stop Switch	I 1.2	S2	I 1.2 S2
Push Button Start Switch	I 1.3	S3	I 1.3 S3
Push Button Stop Switch	I 1.4	S4	I 1.4 S4
Sensor	I 1.5	S5	I 1.5 S5
Motor	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

### Function Block Diagram to control the Conveyor belt

Network 1: Pressing either start switch turns the motor on.

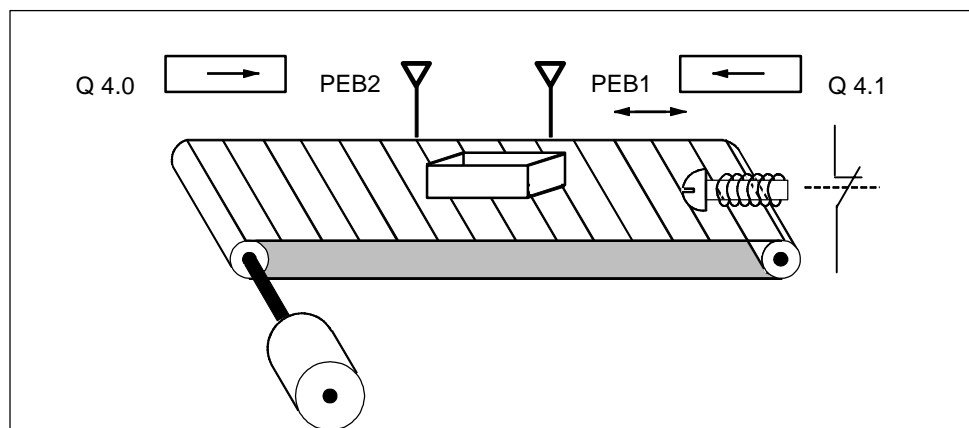


Network 2: Pressing either stop switch or the sensor at the end of the belt turns the motor off.



### Example 2: Detecting the Direction of a Conveyor Belt

The following figure shows a conveyor belt that is equipped with two photoelectric barriers (PEB1 and PEB2) that are designed to detect the direction in which a package is moving on the belt. Each photoelectric light barrier functions like a normally open contact.



### Absolute and symbolic Programming

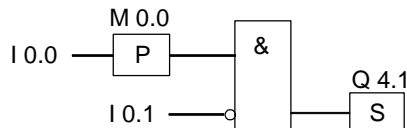
You can write a program to activate a direction display for the conveyor belt system using **absolute values** or **symbols** that represent the various components of the conveyor system.

You need to make a symbol table to correlate the symbols you choose with absolute values (see the STEP 7 Online Help).

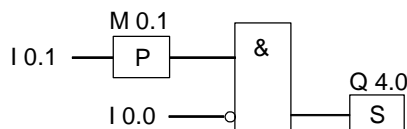
System Component	Absolute Address	Symbol	Symbol Table
Photoelectric barrier 1	I 0.0	PEB1	I 0.0 PEB1
Photoelectric barrier 2	I 0.1	PEB2	I 0.1 PEB2
Display for movement to right	Q 4.0	RIGHT	Q 4.0 RIGHT
Display for movement to left	Q 4.1	LEFT	Q 4.1 LEFT
Pulse memory bit 1	M 0.0	PMB1	M 0.0 PMB1
Pulse memory bit 2	M 0.1	PMB2	M 0.1 PMB2

### Function Block Diagram for Detecting the Direction of a Conveyor Belt

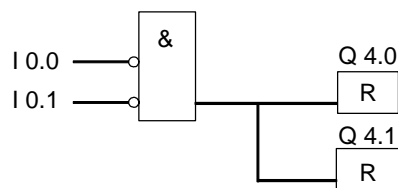
Network 1: If there is a transition in signal state from 0 to 1 (positive edge) at input I 0.0 and, at the same time, the signal state at input I 0.1 is 0, then the package on the belt is moving to the left.



Network 2: If there is a transition in signal state from 0 to 1 (positive edge) at input I 0.1 and, at the same time, the signal state at input I 0.0 is 0, then the package on the belt is moving to the right.



Network 3: If one of the photoelectric barriers is interrupted, this means that there is a package between the barriers. The direction pointer shuts off.





## B.3 Example: Timer Instructions

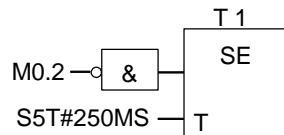
### Clock Pulse Generator

You can use a clock pulse generator or flasher relay when you need to produce a signal that repeats periodically. A clock pulse generator is common in a signalling system that controls the flashing of indicator lamps.

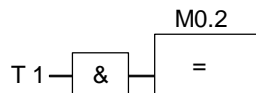
When you use the S7-300, you can implement the clock pulse generator function by using time-driven processing in special organization blocks. The example shown in the following FBD program, however, illustrates the use of timer functions to generate a clock pulse. The sample program shows how to implement a freewheeling clock pulse generator by using a timer.

### Function Block Diagram to Generate a Clock Pulse (pulse duty factor 1:1)

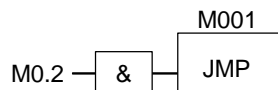
Network 1: If the signal state of timer T1 is 0, load the time value 250 ms into T1 and start T1 as an extended-pulse timer.



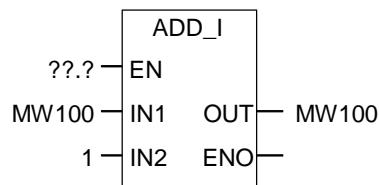
Network 2: The state of the timer is saved temporarily in an auxiliary memory marker.



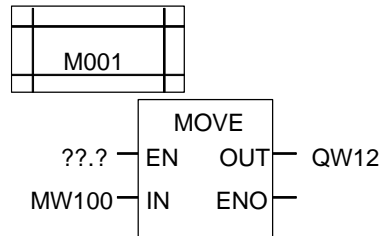
Network 3: If the signal state of timer T1 is 1, jump to jump label M001.



Network 4: When the timer T1 expires, the memory word 100 is incremented by 1.

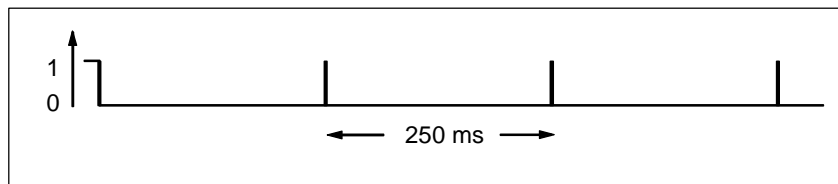


Network 5: The **MOVE** instruction allows you to output the different clock frequencies at outputs Q12.0 through Q13.7.



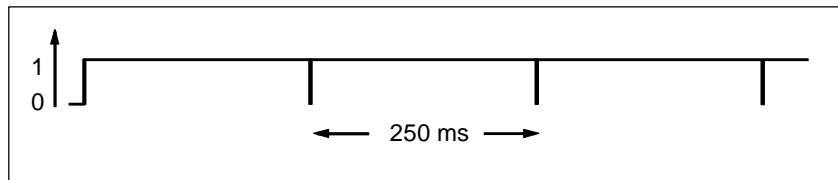
### Signal Check

A signal check of timer T1 produces the following result of logic operation (RLO) for the negated input parameter of the AND logic operation (M0.2) in the clock pulse example:



As soon as the time runs out, the timer is restarted. Because of this, the signal produces a signal state of 1 only briefly.

The negated (inverted) RLO:



Every 250 ms the RLO bit is 0. The jump is ignored and the contents of memory word MW100 is incremented by 1.

### Achieving a Specific Frequency

From the individual bits of memory bytes MB101 and MB100 you can achieve the following frequencies:

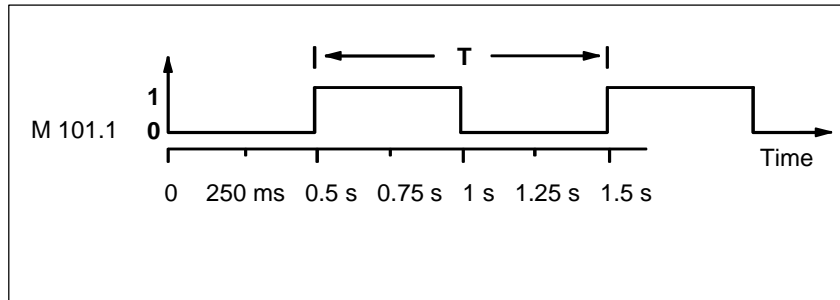
Bits of MB101/MB100	Frequency in Hz	Duration
M 101.0	2.0	0.5 s (250 ms on / 250 ms off)
M 101.1	1.0	1 s (0.5 s on / 0.5 s off)
M 101.2	0.5	2 s (1 s on / 1 s off)
M 101.3	0.25	4 s (2 s on / 2 s off)
M 101.4	0.125	8 s (4 s on / 4 s off)
M 101.5	0.0625	16 s (8 s on / 8 s off)
M 101.6	0.03125	32 s (16 s on / 16 s off)
M 101.7	0.015625	64 s (32 s on / 32 s off)
M 100.0	0.0078125	128 s (64 s on / 64 s off)
M 100.1	0.0039062	256 s (128 s on / 128 s off)
M 100.2	0.0019531	512 s (256 s on / 256 s off)
M 100.3	0.0009765	1024 s (512 s on / 512 s off)
M 100.4	0.0004882	2048 s (1024 s on / 1024 s off)
M 100.5	0.0002441	4096 s (2048 s on / 2048 s off)
M 100.6	0.000122	8192 s (4096 s on / 4096 s off)
M 100.7	0.000061	16384 s (8192 s on / 8192 s off)

### Signal states of the Bits of Memory MB 101

Scan Cycle	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Time Value in ms
0	0	0	0	0	0	0	<b>0</b>	0	250
1	0	0	0	0	0	0	<b>0</b>	1	250
2	0	0	0	0	0	0	1	0	250
3	0	0	0	0	0	0	1	1	250
4	0	0	0	0	0	1	<b>0</b>	0	250
5	0	0	0	0	0	1	<b>0</b>	1	250
6	0	0	0	0	0	1	1	0	250
7	0	0	0	0	0	1	1	1	250
8	0	0	0	0	1	0	<b>0</b>	0	250
9	0	0	0	0	1	0	<b>0</b>	1	250
10	0	0	0	0	1	0	1	0	250
11	0	0	0	0	1	0	1	1	250
12	0	0	0	0	1	1	<b>0</b>	0	250

**Signal state of Bit 1 of MB 101 (M 101.1)**

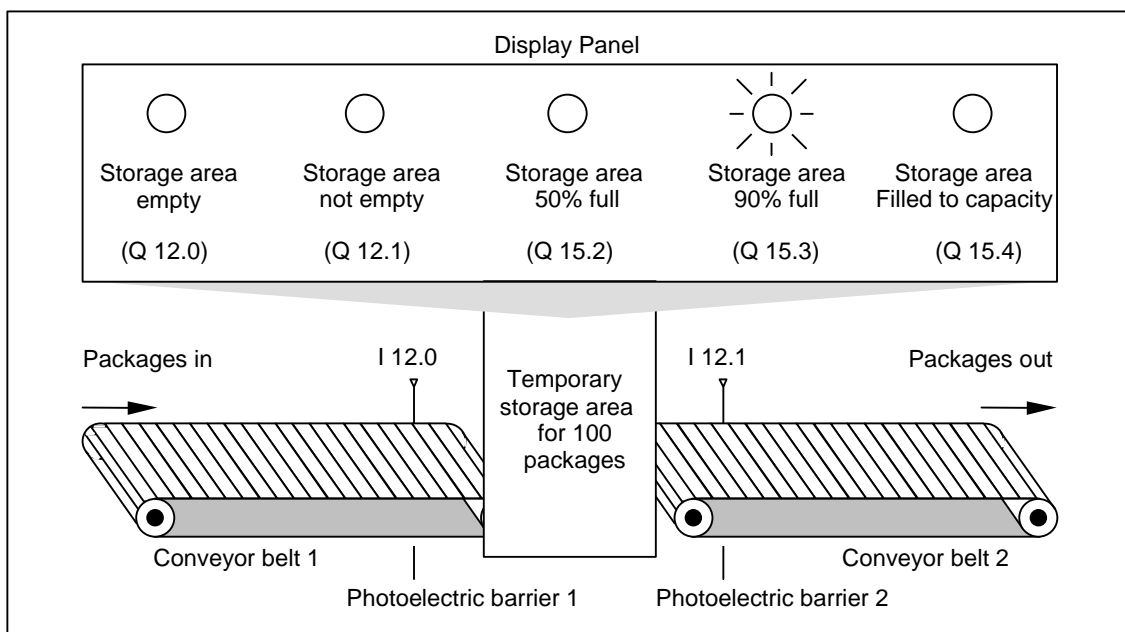
Frequency =  $1/T = 1/1 \text{ s} = 1 \text{ Hz}$



## B.4 Example: Counter and Comparison Instructions

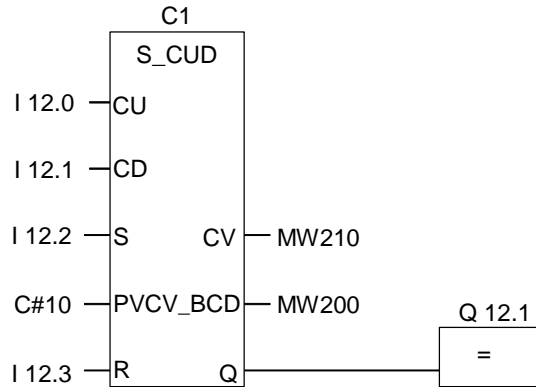
### Storage Area with Counter and Comparator

The following figure shows a system with two conveyor belts and a temporary storage area in between them. Conveyor belt 1 delivers packages to the storage area. A photoelectric barrier at the end of conveyor belt 1 near the storage area determines how many packages are delivered to the storage area. Conveyor belt 2 transports packages from the temporary storage area to a loading dock where trucks take the packages away for delivery to customers. A photoelectric barrier at the end of conveyor belt 2 near the storage area determines how many packages leave the storage area to go to the loading dock. A display panel with five lamps indicates the fill level of the temporary storage area.

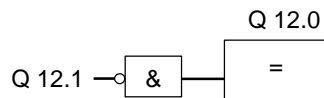


### Function Block Diagram that Activates the Indicator Lamps on the Display Panel

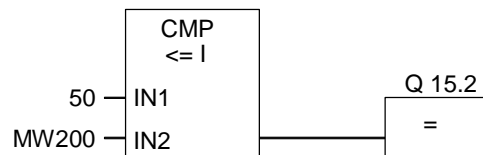
Network 1: Counter C1 counts up at each signal change from "0" to "1" at input CU and counts down at each signal change from "0" to "1" at input CD. With a signal change from "0" to "1" at input S, the counter value is set to the value PV. A signal change from "0" to "1" at input R resets the counter value to "0". MW200 contains the current counter value of C1. Q12.1 indicates "storage area not empty".



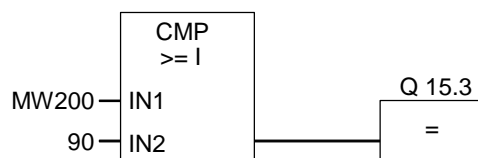
Network 2: Q12.0 indicates "storage area empty".



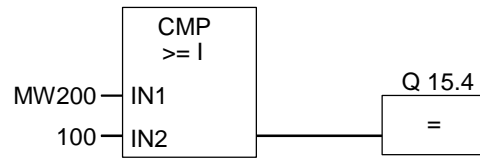
Network 3: If 50 is less than or equal to the counter value (in other words if the current counter value is greater than or equal to 50), the indicator lamp for "storage area 50% full" is lit.



Network 4: Network 4: If the counter value is greater than or equal to 90, the indicator lamp for "storage area 90% full" is lit.



Network 5: If the counter value is greater than or equal to 100, the indicator lamp for "storage area full" is lit.



## B.5 Example: Integer Math Instructions

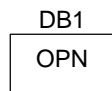
### Solving a Math Problem

The sample program shows you how to use three integer math instructions to produce the same result as the following equation:

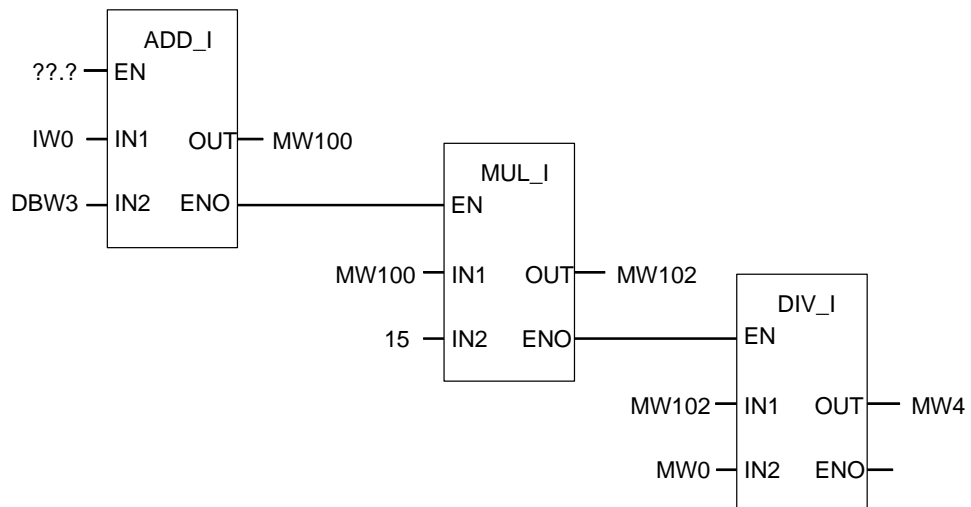
$$MW4 = ((IW0 + DBW3) \times 15) / MW0$$

### Function Block Diagram

Network 1: Open Data Block DB1.



Network 2: Input word IW0 is added to shared data word DBW3 (data block must be defined and opened) and the sum is loaded into memory word MW100. MW100 is then multiplied by 15 and the answer stored in memory word MW102. MW102 is divided by MW0 with the result stored in MW4.

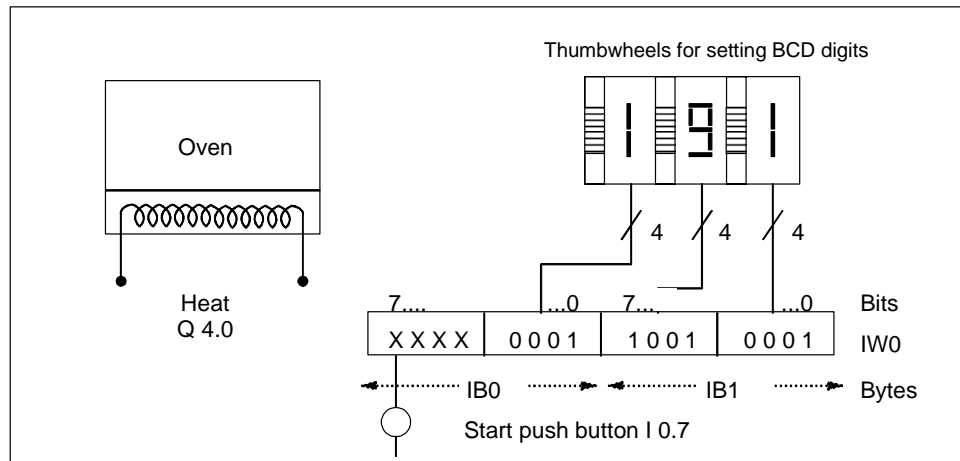




## B.6 Example: Word Logic Instructions

### Heating an Oven

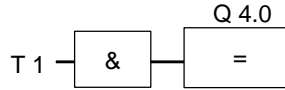
The operator of the oven starts the oven heating by pushing the start push button. The operator can set the length of time for heating by using the thumbwheel switches shown in the figure. The value that the operator sets indicates seconds in binary coded decimal (BCD) format.



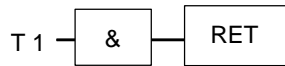
System Component	Absolute Address
Start Push Button	I 0.7
Thumbwheel for ones	I 1.0 to I 1.3
Thumbwheel for tens	I 1.4 to I 1.7
Thumbwheel for hundreds	I 0.0 to I 0.3
Heating starts	Q 4.0

### Function Block Diagram

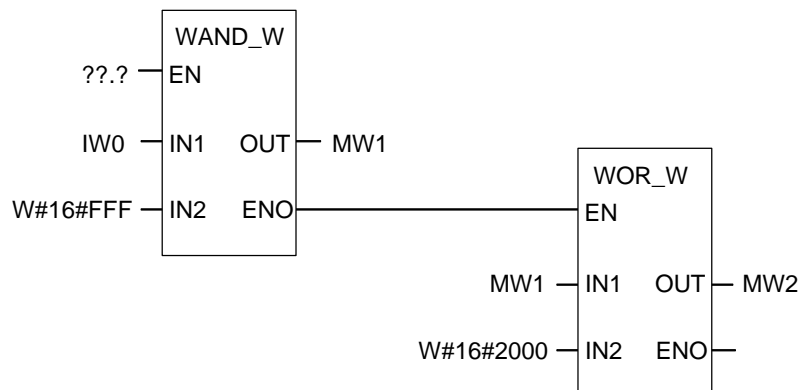
Network 1: If the timer is running, then turn on the heater.



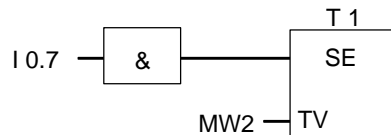
Network 2: If the timer is running, the **Return** instruction ends the processing here.



Network 3: Mask input bits I 0.4 through I 0.7 (that is, reset them to 0). These bits of the thumbwheel inputs are not used. The 16 bits of the thumbwheel inputs are combined with W#16#0FFF according to the **(Word) And Word** instruction. The result is loaded into memory word MW1. In order to set the time base of seconds, the preset value is combined with W#16#2000 according to the **(Word) Or Word** instruction, setting bit 13 to 1 and resetting bit 12 to 0.



Network 4: Start timer T1 as an extended pulse timer if the start push button is pressed, loading as a preset value memory word MW2 (derived from the logic above).



## C Working with Function Block Diagram

### C.1 EN/ENO Mechanism

The enable (EN) and enable output (ENO) of FBD/LAD boxes is achieved by means of the BR bit.

If EN and ENO are connected, the following applies:

#### **ENO = EN AND NOT (box error)**

If no error occurs (box error = 0), ENO = EN.

The EN/ENO mechanism is used for:

- Math instructions,
- Transfer and conversion instructions,
- Shift and rotate instructions,
- Block calls.

This mechanism is **not** used for:

- Comparisons,
- Counters,
- Timers.

Around the actual instructions in the box, additional STL instructions are generated for the EN/ENO mechanism with dependency on the existing preceding and subsequent logic operations. The four possible cases are shown using the example of an adder:

- Adder with EN and with ENO Connected
- Adder with EN and without ENO Connected
- Adder without EN and with ENO Connected
- Adder without EN and without ENO Connected

## Note on Creating Your Own Blocks

If you want to program blocks which you want to call in FBD or LAD, you must ensure that the BR bit is set when the block is exited. The fourth example shows that this is not automatically the case. You cannot use the BR as a memory bit because it is constantly overwritten by the EN/ENO mechanism. Instead, use a temporary variable in which you save any errors which occur. Initialize this variable with 0. At each point in the block at which you think an unsuccessful instruction represents an error for the whole block, set this variable using the assistance of the EN/ENO mechanism. A NOT and a SET coil will be sufficient for this. At the end of the block program the following network:

```
end:  AN error
      SAVE
```

Ensure that this network is processed in every case, which means you must not use BEC within the block and skip this network.

### C.1.1 Adder with EN and with ENO Connected

If the adder has an EN and an ENO connected, the following STL instructions are triggered:

```
1  A   I   0.0           // EN connection
2  JNB _001             // Shift RLO into BR and jump if RLO = 0
3  L   in1             // Box parameter
4  L   in2             // Box parameter
5  +I                  // Actual addition
6  T   out             // Box parameter
7  AN   OV             // Error recognition
8  SAVE                // Save error in BR
9  CLR                // First check
10 _001: A   BR         // Shift BR into RLO
11 =   Q   4.0
```

Following line 1 the RLO contains the result of the preceding logic operation. The JNB instruction copies the RLO into the BR bit and sets the first check bit.

- If the RLO = 0, the program jumps to line 10 and resumes with A BR. The addition is not executed. In line 10 the BR is copied into the RLO again and 0 is thus assigned to the output.
- If the RLO = 1, the program does not jump, meaning the addition is executed. In line 7 the program evaluates whether an error occurred during addition, this is then stored in BR in line 8. Line 9 sets the first check bit. Now the BR bit is copied back into the RLO in line 10 and thus the output shows whether the addition was successful or not.  
The BR bit is not changed by lines 10 and 11, so it also shows whether the addition was successful.

## C.1.2 Adder with EN and without ENO Connected

If the adder has an EN but no ENO connected, the following STL instructions are triggered:

```
1 A    I    0.0 // EN connection
2 JNB _001      // Shift RLO into BR and jump if RLO = 0
3 L    in1      // Box parameter
4 L    in2      // Box parameter
5 +I          // Actual addition
6 T    out      // Box parameter
7 _001: NOP    0
```

Following line 1 the RLO contains the result of the preceding logic operation. The JNB instruction copies the RLO into the BR bit and sets the first check bit.

- If the RLO = 0, the program jumps to line 7 and the addition is not executed. The RLO and BR are 0.
- If RLO was 1, the program does not jump, meaning the addition is executed. The program does not evaluate whether an error occurred during addition. The RLO and BR are 1.

### C.1.3 Adder without EN and with ENO Connected

If the adder has no EN but an ENO connected, the following STL instructions are triggered:

```
1 L    in1    // Box parameter
2 L    in2    // Box parameter
3 +I                    // Actual addition
4 T    out    // Box parameter
5 AN   OV     // Error recognition
6 SAVE                    // Save error in BR
7 CLR                    // First check
8 A    BR     // Shift BR into RLO
9 = Q 4.0
```

The addition is executed in every case. In line 5 the program evaluates whether an error occurred during addition, this is then stored in BR in line 6. Line 7 sets the first check bit. Now the BR bit is copied back into the RLO in line 8 and thus the output shows whether the addition was successful or not.

The BR bit is not changed by lines 8 and 9, so it also shows whether the addition was successful.

#### C.1.4 Adder without EN and without ENO Connected

If the adder has no EN and no ENO connected, the following STL instructions are triggered:

```
1 L    in1    // Box parameter
2 L    in2    // Box parameter
3 +I                    // Actual addition
4 T    out    // Box parameter
5 NOP 0
```

The addition is executed. The RLO and the BR bit remain unchanged.

## C.2 Parameter Transfer

The parameters of a block are transferred as a value. With function blocks a copy of the actual parameter value in the instance data block is used in the called block. With functions a copy of the actual value lies in the local data stack. Pointers are not copied. Prior to the call the INPUT values are copied into the instance DB or to the L stack. After the call the OUTPUT values are copied back into the variables. Within the called block you can only work on a copy. The STL instructions required for this are in the calling block and remain hidden from the user.

---

### Note

If memory bits, inputs, outputs or peripheral I/Os are used as actual address of a function they are treated in a different way than the other addresses. Here, updates are carried out directly, not via L-Stack.

### Exception:

If the corresponding formal parameter is an input parameter of the data type **BOOL**, the current parameters are updated via the L stack.

---



### CAUTION

When programming the called block, ensure that the parameters declared as OUTPUT are also written. Otherwise the values output are random! With function blocks the value will be the value from the instance DB noted by the last call, with functions the value will be the value which happens to be in the L stack.

Note the following points:

- Initialize all OUTPUT parameters if possible.
- Try not to use any Set and Reset instructions. These instructions are dependent on the RLO. If the RLO has the value 0, the random value will be retained.
- If you jump within the block, ensure that you do not skip any locations where OUTPUT parameters are written. Do not forget BEC and the effect of the MCR instructions.



# Index

<b>#</b>	
# .....	1-11
<b>&amp;</b>	
& : .....	1-3
<b>&lt;</b>	
<> 0 .....	12-8
<b>=</b>	
= .....	1-9
<b>&gt;</b>	
>=1 .....	1-2
<b>A</b>	
ABS .....	8-8
Add Double Integer .....	7-8
Add Integer .....	7-3
Add Real .....	8-3
ADD_DI .....	7-8
ADD_I .....	7-3
ADD_R .....	8-3
Adder with EN and with ENO Connected .....	C-2
Adder with EN and without ENO Connected .....	C-3
Adder without EN and with ENO Connected .....	C-4
Adder without EN and without ENO Connected .....	C-5
Address Negative Edge Detection .....	1-22
Address Positive Edge Detection .....	1-23
AND Double Word (Word) .....	14-5
AND Logic Operation .....	1-3
AND Word (Word) .....	14-2
AND-before-OR Logic Operation and OR-before-AND Logic Operation .....	1-4
Assign .....	1-9
Assign Extended Pulse Timer Parameters and Start .....	13-7
Assign Off-Delay Timer Parameters and Start .....	13-13

Assign On-Delay Timer Parameters and Start .....	13-9
Assign Parameters and Count Down .....	4-7
Assign Parameters and Count Up .....	4-5
Assign Parameters and Count Up/Down .....	4-3
Assign Pulse Timer Parameters and Start .....	13-5
Assign Retentive On-Delay Timer Parameters and Start .....	13-11
Assign Value .....	9-1

## B

BCD to Double Integer .....	3-5
BCD to Integer .....	3-2
BCD_DI .....	3-5
BCD_I .....	3-2
BR .....	12-7

## C

CALL .....	10-2
CALL_FB .....	10-4
CALL_FB (Call FB as Box) .....	10-4
CALL_FC .....	10-6
CALL_FC (Call FC as Box) .....	10-6
CALL_SFB .....	10-8
CALL_SFB (Call System FB as Box) .....	10-8
CALL_SFC .....	10-10
CALL_SFC (Call System FC as Box) .....	10-10
Calling a Block from a Library .....	10-12
Calling an FC/SFC without Parameters .....	10-2
Calling Multiple Instances .....	10-12
CD .....	4-12
CEIL .....	3-17
Ceiling .....	3-17
CMP ? D .....	2-3
CMP ? I .....	2-2
CMP ? R .....	2-4
Compare Double Integer .....	2-3
Compare Integer .....	2-2
Compare Real .....	2-4
Conditional Jump in a Block .....	6-3
CU .....	4-11

<b>D</b>		<b>I</b>	
DI_BCD .....	3-8	I_BCD .....	3-4
DI_R .....	3-9	I_DI .....	3-7
DIV_DI .....	7-11	Important Notes on	
DIV_I .....	7-7	Using MCR Functions .....	10-14
DIV_R .....	8-7	Insert Binary Input .....	1-7
Divide Double Integer .....	7-11	Integer to BCD .....	3-4
Divide Integer .....	7-7	Integer to Double Integer .....	3-7
Divide Real .....	8-7	INV_DI .....	3-11
Double Integer to BCD .....	3-8	INV_I .....	3-10
Double Integer to Real .....	3-9		
Down Counter .....	4-12	<b>J</b>	
		JMP .....	6-2, 6-3
<b>E</b>		JMPN .....	6-5
EN/ENO Mechanism .....	C-1, C-2	Jump Label .....	6-7
Evaluating the Bits of the Status Word		Jump-If-Not .....	6-5
with Floating-Point Instructions .....	8-2		
Evaluating the Bits of the Status Word		<b>L</b>	
with Integer Math Instructions .....	7-2	LABEL .....	6-7
Example		LN .....	8-12
Bit Logic Instructions .....	B-2		
Counter and Comparison Instructions	B-9	<b>M</b>	
Integer Math Instructions .....	B-12	Master Control Relay Activate/Deactivate	
Timer Instructions .....	B-5	.....	10-19
Word Logic Instructions .....	B-13	Master Control Relay Instructions .....	10-13
Examples .....	B-1	Master Control Relay On/Off .....	10-15
Exception Bit BR Memory .....	12-7	MCR< .....	10-15
Exception Bit Overflow .....	12-2	MCR> .....	10-15
Exception Bit Overflow Stored .....	12-4	MCRA .....	10-19
Exception Bit Unordered .....	12-6	MCRD .....	10-19
Exclusive OR Double Word (Word) .....	14-7	Memory Areas and Components	
Exclusive OR Logic Operation .....	1-6	of a Timer .....	13-2
Exclusive OR Word (Word) .....	14-4	Midline Output .....	1-11
EXP .....	8-11	Mnemonics	
		English (International) .....	A-5
<b>F</b>		German (SIMATIC) .....	A-1
FBD Instructions Sorted According to		MOD_DI .....	7-12
English Mnemonics (International) ....	A-5	MOVE .....	9-1
FBD Instructions Sorted According to		MUL_DI .....	7-10
German Mnemonics (SIMATIC) .....	A-1	MUL_I .....	7-6
Floor .....	3-18	MUL_R .....	8-6
FLOOR .....	3-18	Multiply Double Integer .....	7-10
Forming the Absolute Value of a		Multiply Integer .....	7-6
Floating-Point Number .....	8-8	Multiply Real .....	8-6
Forming the Exponential Value of a			
Floating-Point Number .....	8-11		
Forming the Natural Logarithm of a			
Floating-Point Number .....	8-12		
Forming the Square (SQR) of a			
Floating-Point Number .....	8-9		
Forming the Square Root (SQRT) of a			
Floating-Point Number .....	8-10		
Forming Trigonometric Functions of			
Angles as Floating-Point Numbers ...	8-13		

<b>N</b>	
N .....	1-19
NEG .....	1-22
NEG_DI .....	3-13
NEG_I .....	3-12
NEG_R .....	3-14
Negate Binary Input.....	1-8
Negate Real Number.....	3-14
Negative RLO Edge Detection .....	1-19
<b>O</b>	
Ones Complement Double Integer.....	3-11
Ones Complement Integer .....	3-10
Open Data Block .....	5-1
OPN.....	5-1
OR Double Word (Word).....	14-6
OR Logic Operation.....	1-2
OR Word (Word).....	14-3
OS .....	12-4
OV .....	12-2
Overview of Bit Logic Instructions .....	1-1
Overview of Comparison Instructions .....	2-1
Overview of Conversion Instructions.....	3-1
Overview of Counter Instructions .....	4-1
Overview of Floating-Point Math .....	8-1
Overview of Integer Math Instructions....	7-1
Overview of Jump Instructions .....	6-1
Overview of Program Control Instructions .....	10-1
Overview of Programming Examples.....	B-1
Overview of Status Bit Instructions.....	12-1
Overview of Timer Instructions.....	13-1
Overview of Word Logic Instructions....	14-1
<b>P</b>	
P .....	1-20
Parameter Transfer .....	C-6
POS.....	1-23
Positive RLO Edge Detection.....	1-20
Practical Applications .....	B-1
<b>R</b>	
R .....	1-13
Reset Output .....	1-13
Reset_Set Flip Flop.....	1-15
Result Bits .....	12-8
RET .....	10-22
Return.....	10-22
Return Fraction Double Integer.....	7-12
ROL_DW .....	11-12
ROR_DW.....	11-14
Rotate Instructions - Overview .....	11-12
Rotate Left Double Word.....	11-12
Rotate Right Double Word .....	11-14
ROUND.....	3-15
Round to Double Integer.....	3-15
RS .....	1-15
<b>S</b>	
S .....	1-14
S_CD .....	4-7
S_CU .....	4-5
S_CUD.....	4-3
S_ODT .....	13-9
S_ODTS.....	13-11
S_OFFDT .....	13-13
S_PEXT .....	13-7
S_PULSE.....	13-5
SAVE .....	1-21
Save RLO to BR Memory .....	1-21
SC.....	4-9
SD .....	13-19
SE .....	13-17
Set Counter Value.....	4-9
Set Output.....	1-14
Set_Reset Flip Flop .....	1-17
SF.....	13-23
Shift Instructions - Overview .....	11-1
Shift Left Double Word.....	11-9
Shift Left Word .....	11-6
Shift Right Double Integer.....	11-4
Shift Right Double Word .....	11-10
Shift Right Integer .....	11-2
Shift Right Word.....	11-8
SHL_DW .....	11-9
SHL_W.....	11-6
SHR_DI.....	11-4
SHR_DW .....	11-10
SHR_I .....	11-2
SHR_W .....	11-8
SP .....	13-15
SQR .....	8-9
SQRT .....	8-10
SR .....	1-17
SS .....	13-21
Start Extended Pulse Timer.....	13-17
Start Off-Delay Timer.....	13-23
Start On-Delay Timer.....	13-19
Start Pulse Timer .....	13-15
Start Retentive On-Delay Timer.....	13-21
SUB_DI .....	7-9
SUB_I.....	7-5
SUB_R.....	8-5
Subtract Double Integer.....	7-9
Subtract Integer .....	7-5
Subtract Real .....	8-5

**T**

TRUNC ..... 3-16  
Truncate Double Integer Part ..... 3-16  
Twos Complement Double Integer ..... 3-13  
Twos Complement Integer ..... 3-12

**U**

Unconditional Jump in a Block ..... 6-2  
UO ..... 12-6  
Up Counter ..... 4-11

**W**

WAND\_DW ..... 14-5  
WAND\_W ..... 14-2  
WOR\_DW ..... 14-6  
WOR\_W ..... 14-3  
WXOR\_DW ..... 14-7  
WXOR\_W ..... 14-4

**X**

XOR ..... 1-6