

<b>1 Datentypen .....</b>	<b>3</b>
<b>2 Alphabetische Liste von FBS-Befehlen nach IEC-61131-3.....</b>	<b>5</b>
2.1 ABS: Absolutwertbildung.....	5
2.2 ADD: Addition .....	6
2.3 AND: UND-Funktion.....	6
2.4 BOOL_TO: Typumwandlung.....	7
2.5 BYTE_TO: Typumwandlung .....	8
2.6 CTD: Abwärts-Zähler .....	8
2.7 CTU: Aufwärts-Zähler .....	9
2.8 CTUD: Auf-/Abwärts-Zähler .....	10
2.9 DIV: Division.....	10
2.10 EQ: Gleich.....	11
2.11 F_TRIG: Erkennung fallender Flanken .....	12
2.12 GE: Größergleich .....	12
2.13 GT: Größer .....	13
2.14 INT_TO: Typumwandlung .....	14
2.15 LE: Kleinergleich .....	14
2.16 LIMIT: Begrenzung .....	15
2.17 LT: Kleiner .....	16
2.18 MAX: Maximalwert-Auswahl .....	16
2.19 MIN: Minimalwert-Auswahl.....	17
2.20 MOVE: Zuweisung .....	18
2.21 MUL: Multiplikation .....	18
2.22 MUX: Multiplexer.....	19
2.23 NE: Ungleich .....	20
2.24 NOT: Negation .....	20
2.25 OR: ODER-Funktion .....	21
2.26 R_TRIG: Erkennung steigender Flanken .....	22
2.27 REAL_TO: Typumwandlung .....	22
2.28 REAL_TRUNC: Typumwandlung.....	23
2.29 RS: Bistabiler Funktionsbaustein, Rücksetzen dominant .....	24
2.30 SR: Bistabiler Funktionsbaustein, Setzen dominant.....	24
2.31 SUB: Subtraktion.....	25
2.32 TIME_TO: Typumwandlung .....	26
2.33 TOF: Ausschaltverzögerung .....	26
2.34 TON: Einschaltverzögerung .....	28
2.35 TP: Puls .....	29
2.36 WORD_TO: Typumwandlung .....	30
2.37 XOR: Exklusiv-ODER-Funktion.....	30

<b>3 Programmiersprache „Strukturierter Text“ (ST) nach IEC-61131-3 .....</b>	<b>32</b>
3.1 Anweisungen .....	32
3.2 Zuweisungen.....	32
3.3 VAR...END_VAR.....	33
3.4 IF...THEN...END_IF .....	34
3.5 ELSE .....	34
3.6 ELSIF...THEN .....	35
3.7 CASE...OF...END_CASE .....	36
3.8 FOR...TO...BY...DO...END_FOR.....	36
3.9 WHILE...DO...END_WHILE .....	37
3.10 REPEAT...UNTIL...END_REPEAT .....	37
3.11 EXIT .....	38
3.12 Leeranweisung.....	38
3.13 Kommentare .....	38
3.14 Operatoren.....	39
<b>4 Programmiersprache „Anweisungsliste“ (AWL) nach IEC-61131-3 .....</b>	<b>40</b>
4.1 Anweisungen .....	40
4.2 Operatoren.....	41
4.3 Operanden .....	41
4.4 Modifizierer.....	41
4.4.1 N .....	41
4.4.2 C .....	42
4.4.3 ( Linke Klammer .....	42
4.4.4 VAR...END_VAR.....	42
4.5 Funktionsbaustein-Aufrufe .....	43
4.6 Funktions-Aufrufe.....	43
4.7 Kommentare .....	43
<b>5 Programmiersprache „Kontaktplan“ (KOP) nach IEC-61131-3.....</b>	<b>44</b>
5.1 Kontakte und Spulen.....	44
5.2 Linien.....	45
5.3 Stromschienen und Verbindungen .....	46
<b>6 Programmiersprache „Ablaufsprache“ (AS) nach IEC-61131-3 .....</b>	<b>47</b>
6.1 Bestimmungszeichen für Befehle .....	47
6.1.1 Bestimmungszeichen N (oder kein) .....	48
6.1.2 Bestimmungszeichen S und R .....	48
6.1.3 Bestimmungszeichen D .....	48
6.1.4 Bestimmungszeichen L .....	49
6.1.5 Bestimmungszeichen P.....	49

## 1 Datentypen

### **BOOL**

BOOL steht für den Datentyp "boolsch". Die Länge der Datenelemente ist 1 Bit (im Speicher abgelegt in 1 Byte). Der Wertebereich für Variablen dieses Datentyps ist 0 (FALSE) und 1 (TRUE).

### **BYTE**

BYTE steht für den Datentyp "Bit-Folge 8". Die Länge der Datenelemente ist 8 Bit. Ein numerischer Wertebereich kann diesem Datentyp nicht zugeordnet werden.

### **INT**

INT steht für den Datentyp "ganze Zahl (integer)". Die Länge der Datenelemente ist 16 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von  $-2^{15}$  bis  $2^{15} - 1$ .

### **DINT**

DINT steht für den Datentyp "doppelt lange ganze Zahl (double integer)". Die Länge der Datenelemente ist 32 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von  $-2^{31}$  bis  $2^{31} - 1$ .

### **UINT**

UINT steht für den Datentyp "vorzeichenlose ganze Zahl (unsigned integer)". Die Länge der Datenelemente ist 16 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von 0 bis  $2^{16} - 1$ .

### **UDINT**

UDINT steht für den Datentyp "vorzeichenlose doppelt lange ganze Zahl (unsigned double integer)". Die Länge der Datenelemente ist 32 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von 0 bis  $2^{32} - 1$ .

### **REAL**

REAL steht für den Datentyp "Gleitkomma-Zahl". Die Länge der Datenelemente ist 32 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von  $8.43\text{E}-37$  bis  $3.36\text{E}+38$ .

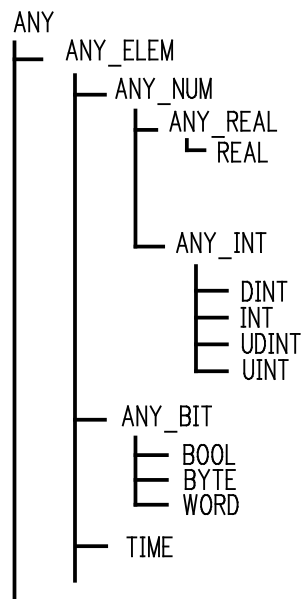
### **WORD**

WORD steht für den Datentyp "Bit-Folge 16". Die Länge der Datenelemente ist 16 Bit. Ein numerischer Wertebereich kann diesem Datentyp nicht zugeordnet werden.

### **TIME**

TIME steht für den Datentyp "Zeitdauer". Die Länge der Datenelemente ist 32 Bit. Der Wertebereich für Variablen dieses Datentyps reicht von 0 bis  $2^{32} - 1$ . Die Einheit für den Datentyp TIME ist 1 ms.

Die verwendeten Zusammenfassungen von Datentypen unterliegen der folgenden Hierarchie:



## 2 Alphabetische Liste von FBS-Befehlen nach IEC-61131-3

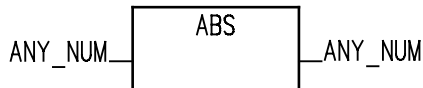
**Vorbemerkung:** Die nachfolgende Liste der Funktionen ist nicht vollständig. Die Darstellung und die Syntax der Befehle ist an die IEC-61131-3 Norm und an die FBS-Bibliothek der Software Concept angelehnt.

### 2.1 ABS: Absolutwertbildung

Die Funktion bildet den Absolutwert des Eingangswertes und gibt diesen am Ausgang aus.

Es können die Datentypen der Gruppe ANY\_NUM verarbeitet werden. Die Datentypen des Eingangs- und Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

#### Symbolische Darstellung



#### Zuweisung

OUT = | IN |

#### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	ANY_NUM	Eingangswert
OUT	ANY_NUM	Ausgangswert

**ABS**

**ADD**

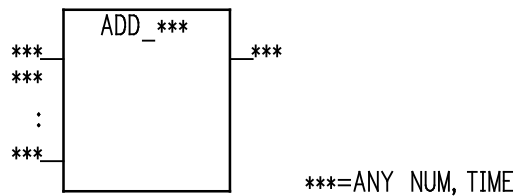
## 2.2 ADD: Addition

Die Funktion addiert Eingangswerte der Gruppe ANY\_NUM oder des Datentyps TIME und gibt das Ergebnis am Ausgang aus.

Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann bei allen Funktionen, mit Ausnahme von ADD\_TIME, erhöht werden.

### Symbolische Darstellung



### Zuweisung

ANY\_NUM:  $OUT = IN1 + IN2 + \dots + INn$

TIME:  $OUT = IN1 + IN2$

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN1	ANY_NUM, TIME	Summand
IN2	ANY_NUM, TIME	Summand
INn	ANY_NUM	Summand
OUT	ANY_NUM, TIME	Summe

**AND**

## 2.3 AND: UND-Funktion

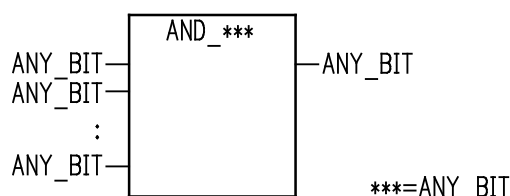
Die Funktion verknüpft (nach UND-Logik) die Bit-Folgen an den Eingängen und gibt das Ergebnis am Ausgang aus. Die Verknüpfung erfolgt bitweise.

Es können die Datentypen der Gruppe ANY\_BIT verarbeitet werden.

Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung



*Zuweisung*

OUT = IN1 & IN2 & ... & INn

*Parameterbeschreibung*

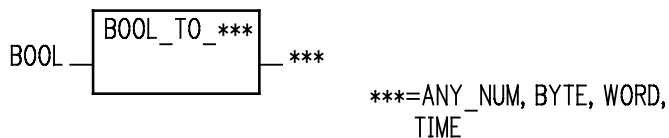
Parameter	Datentyp	Bedeutung
IN1	ANY_BIT	Eingangs-Bit-Folge
IN2	ANY_BIT	Eingangs-Bit-Folge
INn	ANY_BIT	Eingangs-Bit-Folge
OUT	ANY_BIT	Ausgangs-Bit-Folge

**2.4 BOOL\_TO: Typumwandlung**

Die Funktion konvertiert einen Eingangswert vom Datentyp BOOL zu einem Datentyp der Gruppe ANY\_NUM oder den Datentypen BYTE, WORD oder TIME.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Es wird das niederwertigste Bit des Ausgangswertes auf den Eingangswert gesetzt. Alle anderen Bits des Ausgangswertes werden auf null gesetzt.

*Symbolische Darstellung**Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN	BOOL	Eingangswert
OUT	ANY_NUM, BYTE, WORD, TIME	Ausgangswert

**BOOL\_TO**

**BYTE\_TO**

## 2.5 BYTE\_TO: Typumwandlung

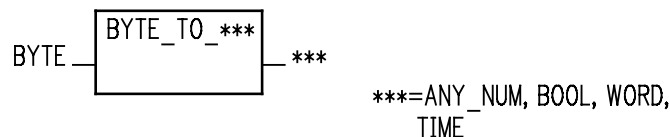
Die Funktion konvertiert einen Eingangswert vom Datentyp BYTE zu einem Datentyp der Gruppe ANY\_NUM oder den Datentypen BOOL, WORD oder TIME.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Bei der Konvertierung des Datentyps BYTE in einen Datentyp der Gruppe ANY\_NUM oder in den Datentyp TIME oder WORD wird das Bitmuster des Eingangs in die niederwertigsten Bits des Ausgangs übertragen. Die höherwertigen Bits des Ausgangs werden auf null gesetzt.

Bei der Konvertierung des Datentyps BYTE in den Datentyp BOOL wird das niederwertigste Bit des Eingangswertes zum Ausgang übertragen.

### Symbolische Darstellung



### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	BYTE	Eingangswert
OUT	ANY_NUM, BOOL, WORD, TIME	Ausgangswert

**CTD**

## 2.6 CTD: Abwärts-Zähler

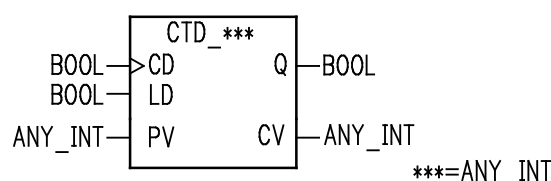
Der Funktionsbaustein wird zum Abwärtszählen verwendet.

Bei "1"-Signal am Eingang LD wird der Wert des Eingangs PV dem Ausgang CV zugewiesen. Bei jedem Übergang von "0" nach "1" am Eingang CD wird der Wert von CV um 1 vermindert.

Bei  $CV < 0$  wird der Ausgang Q "1".

Die Datentypen des Eingangs PV und des Ausgangs CV müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils ein spezieller Funktionsbaustein zur Verfügung.

### Symbolische Darstellung





*Parameterbeschreibung*

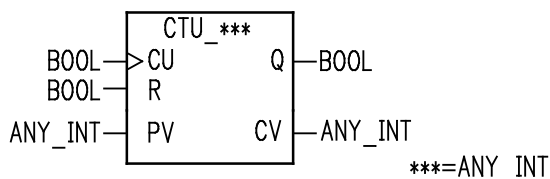
Parameter	Datentyp	Bedeutung
CD	BOOL	Trigger-Eingang
LD	BOOL	Lade Daten
PV	ANY_INT	Voreinstellungswert
Q	BOOL	Ausgang
CV	ANY_INT	Zählwert (Istwert)

**2.7 CTU: Aufwärts-Zähler**

Der Funktionsbaustein wird zum Aufwärtszählen verwendet.

Bei "1"-Signal am Eingang R wird der Wert "0" dem Ausgang CV zugewiesen. Bei jedem Übergang von "0" nach "1" am Eingang CU wird der Wert von CV um 1 erhöht. Bei  $CV \geq PV$  wird der Ausgang Q auf "1" gesetzt.

Die Datentypen des Eingangs PV und des Ausgangs CV müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils ein spezieller Funktionsbaustein zur Verfügung.

*Symbolische Darstellung**Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
CU	BOOL	Trigger-Eingang
R	BOOL	Reset
PV	ANY_INT	Voreinstellungswert
Q	BOOL	Ausgang
CV	ANY_INT	Zählwert (Istwert)

**CTU**

**CTUD**

## 2.8 CTUD: Auf-/Abwärts-Zähler

Der Funktionsbaustein wird zum Auf- und Abwärtszählen verwendet.

Bei "1"-Signal am Eingang R wird der Wert "0" dem Ausgang CV zugewiesen. Bei "1"-Signal am Eingang LD wird der Wert des Eingangs PV dem Ausgang CV zugewiesen. Bei jedem Übergang von "0" nach "1" am Eingang CU wird der Wert von CV um 1 erhöht. Bei jedem Übergang von "0" nach "1" am Eingang CD wird der Wert von CV um 1 vermindert.

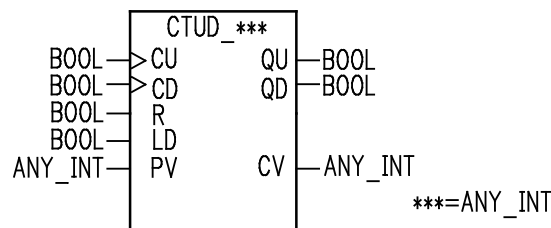
Bei gleichzeitigem "1"-Signal an Eingang R und LD ist der Eingang R dominant.

Bei  $CV \geq PV$  wird der Ausgang QU "1".

Bei  $CV \leq 0$  wird der Ausgang QD "1".

Die Datentypen des Eingangs PV und des Ausgangs CV müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils ein spezieller Funktionsbaustein zur Verfügung.

### Symbolische Darstellung



### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
CU	BOOL	Aufwärts-Zähler Trigger Eingang
CD	BOOL	Abwärts-Zähler Trigger Eingang
R	BOOL	Reset
LD	BOOL	Lade Daten
PV	ANY_INT	Voreinstellungswert
QU	BOOL	Aufwärts-Anzeige
QD	BOOL	Abwärts-Anzeige
CV	ANY_INT	Zählwert (Istwert)

**DIV**

## 2.9 DIV: Division

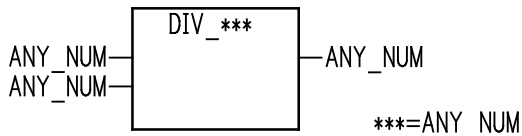
Die Funktion dividiert den Wert an Eingang IN1 durch den Wert an Eingang IN2 und gibt das Ergebnis am Ausgang aus.

Die Datentypen der Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Bei Division von Datentypen der Gruppe ANY\_INT wird beim Ergebnis eine eventuell vorhandene Nachkommastelle in Richtung Null abgeschnitten z.B.

$$7 : 3 = 2$$

$$-7 : 3 = -2$$

*Symbolische Darstellung**Zuweisung*

OUT = IN1 : IN2

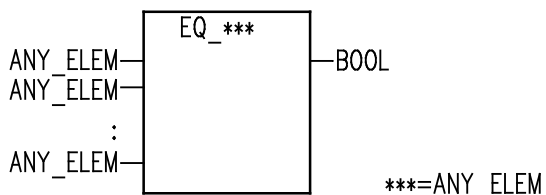
*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_NUM	Dividend
IN2	ANY_NUM	Divisor
OUT	ANY_NUM	Quotient

**2.10 EQ: Gleich**

Die Funktion überprüft die Eingänge auf Gleichheit, d.h. der Ausgang wird "1", wenn an allen Eingängen Gleichheit vorliegt; andernfalls bleibt der Ausgang auf "0".

Die Datentypen aller Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

*Symbolische Darstellung**Zuweisung*

OUT = 1, wenn (IN1 = IN2) & (IN2 = IN3) & ... & (IN(n-1) = INn)

*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingang
IN2	ANY_ELEM	2. Eingang
INn	ANY_ELEM	n. Eingang
OUT	BOOL	Ausgang

EQ

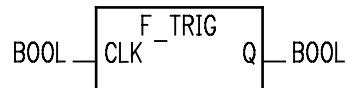
**F\_TRIG**

## 2.11 F\_TRIG: Erkennung fallender Flanken

Der Funktionsbaustein wird zur Erkennung fallender Flanken (1 → 0) verwendet.

Der Ausgang Q wird "1", wenn ein Übergang von "1" nach "0" am Eingang CLK erfolgt. Der Ausgang bleibt von einer Ausführung des Funktionsbausteins bis zur nächsten Ausführung auf "1"; danach kehrt der Ausgang zu "0" zurück.

*Symbolische Darstellung*



*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
CLK	BOOL	Takt-Eingang
Q	BOOL	Ausgang

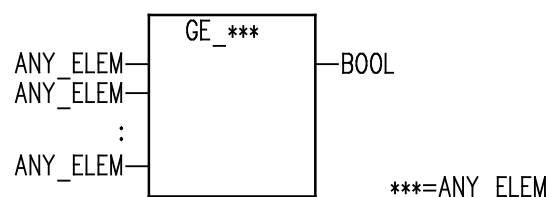
**GE**

## 2.12 GE: Größergleich

Die Funktion überprüft die Werte aufeinander folgender Eingänge auf eine fallende Folge oder Gleichheit.

Die Datentypen aller Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

*Symbolische Darstellung*



*Zuweisung*

OUT = 1, wenn  $(IN1 \geq IN2) \& (IN2 \geq IN3) \& \dots \& (IN(n-1) \geq INn)$

*Parameterbeschreibung*

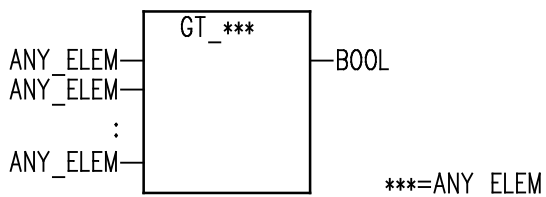
Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingang
IN2	ANY_ELEM	2. Eingang
INn	ANY_ELEM	n. Eingang
OUT	BOOL	Ausgang

## 2.13 GT: Größer

Die Funktion überprüft die Werte aufeinander folgender Eingänge auf eine fallende Folge.

Die Datentypen aller Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung



### Zuweisung

OUT = 1, wenn  $(IN1 > IN2) \& (IN2 > IN3) \& \dots \& (IN(n-1) > INn)$

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingang
IN2	ANY_ELEM	2. Eingang
INn	ANY_ELEM	n. Eingang
OUT	BOOL	Ausgang

GT

**INT\_TO**

## 2.14 INT\_TO: Typumwandlung

Die Funktion konvertiert einen Eingangswert vom Datentyp INT zu einem Ausgangswert vom Datentyp BOOL, BYTE, WORD, DINT, UDINT, UINT, REAL oder TIME.

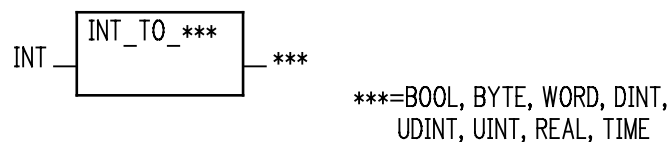
Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Negative Eingangswerte können nicht in die Datentypen UDINT oder UINT gewandelt werden.

Bei der Konvertierung eines Eingangswertes vom Datentyp INT in den Datentyp WORD wird das Bitmuster des Eingangs unverändert zum Ausgang übertragen.

Bei der Konvertierung eines Eingangswertes vom Datentyp INT in die Datentypen BOOL oder BYTE werden die niederwertigsten Bits des Eingangs zum Ausgang übertragen.

### Symbolische Darstellung



### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	INT	Eingangswert
OUT	BOOL, BYTE, WORD, DINT, UDINT, UINT, REAL, TIME	Ausgangswert

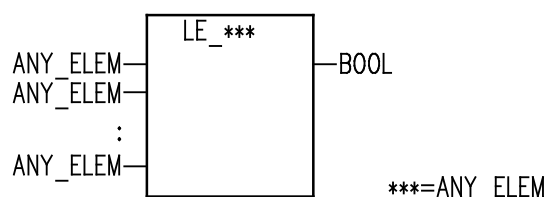
**LE**

## 2.15 LE: Kleingleich

Die Funktion überprüft die Werte aufeinander folgender Eingänge auf eine steigende Folge oder Gleichheit.

Die Datentypen aller Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung



*Zuweisung*

OUT = 1, wenn  $(IN1 \leq IN2) \& (IN2 \leq IN3) \& (IN(n-1) \leq INn)$

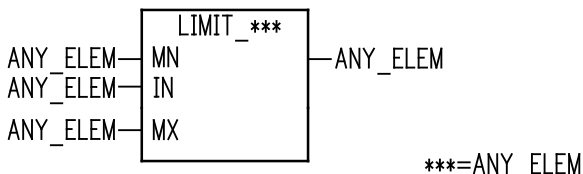
*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingang
IN2	ANY_ELEM	2. Eingang
INn	ANY_ELEM	n. Eingang
OUT	BOOL	Ausgang

**2.16 LIMIT: Begrenzung**

Die Funktion übergibt den unveränderten Eingangswert (IN) an den Ausgang, wenn der Eingangswert den Minimalwert (MN) nicht unterschreitet und den Maximalwert (MX) nicht überschreitet. Unterschreitet der Eingangswert (IN) den Minimalwert (MN), wird der Minimalwert an den Ausgang übergeben. Überschreitet der Eingangswert (IN) den Maximalwert (MX), wird der Maximalwert an den Ausgang übergeben.

Es können die Datentypen der Gruppe ANY\_ELEM verarbeitet werden. Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

*Symbolische Darstellung**Zuweisung*

OUT = IN, wenn  $(IN \geq MN) \& (IN \leq MX)$   
 OUT = MN, wenn  $IN < MN$   
 OUT = MX, wenn  $IN > MX$

*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
MN	ANY_ELEM	unterer Grenzwert
IN	ANY_ELEM	Eingang
MX	ANY_ELEM	oberer Grenzwert
OUT	ANY_ELEM	Ausgang

**LIMIT**

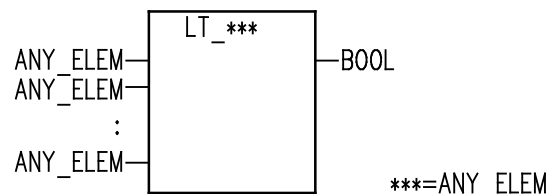
**LT**

## 2.17 LT: Kleiner

Die Funktion überprüft die Werte aufeinander folgender Eingänge auf eine steigende Folge.

Die Datentypen aller Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung



### Zuweisung

OUT = 1, wenn (IN1 < IN2) & (IN2 < IN3) & (IN(n-1) < INn)

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingangswert
IN2	ANY_ELEM	2. Eingangswert
INn	ANY_ELEM	n. Eingangswert
OUT	BOOL	Ausgangswert

**MAX**

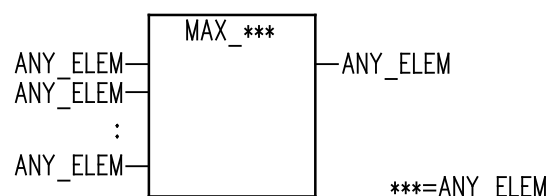
## 2.18 MAX: Maximalwert-Auswahl

Die Funktion gibt den größten Eingangswert am Ausgang aus.

Es können die Datentypen der Gruppe ANY\_ELEM verarbeitet werden. Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung





*Zuweisung*

$$\text{OUT} = \text{MAX} \{ \text{IN1}, \text{IN2}, \dots, \text{INn} \}$$
*Parameterbeschreibung*

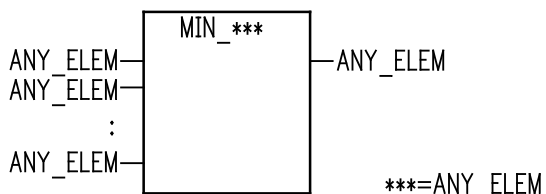
Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingangswert
IN2	ANY_ELEM	2. Eingangswert
INn	ANY_ELEM	n. Eingangswert
OUT	ANY_ELEM	Maximalwert

**2.19 MIN:** Minimalwert-Auswahl

Die Funktion gibt den kleinsten Eingangswert am Ausgang aus.

Es können die Datentypen der Gruppe ANY\_ELEM verarbeitet werden. Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

*Symbolische Darstellung**Zuweisung*

$$\text{OUT} = \text{MIN} \{ \text{IN1}, \text{IN2}, \dots, \text{INn} \}$$
*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingangswert
IN2	ANY_ELEM	2. Eingangswert
INn	ANY_ELEM	n. Eingangswert
OUT	ANY_ELEM	Minimalwert

**MIN**

**MOVE**

## 2.20 MOVE: Zuweisung

Die Funktion weist den Eingangswert dem Ausgang zu.

Die Funktion ist generisch, d.h. der zu verarbeitende Datentyp wird durch die zuerst an die Funktion angelegte Variable bestimmt.

Soll eine direkte Adresse einer Variablen zugewiesen werden oder umgekehrt, muss immer zuerst die Variable an die Funktion angelegt werden. Eine direkte Adresse an Ein- und Ausgang der Funktion ist nicht zulässig, da hierbei keine eindeutige Definition des Datentyps möglich ist.

Die Datentypen des Eingangs- und Ausgangswertes müssen gleich sein.

### Symbolische Darstellung



### Zuweisung

OUT = IN

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	ANY	Eingangswert
OUT	ANY	Ausgangswert

**MUL**

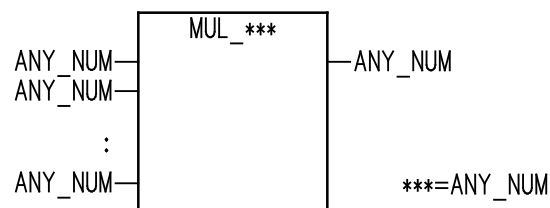
## 2.21 MUL: Multiplikation

Die Funktion multipliziert die Eingangswerte und gibt das Ergebnis am Ausgang aus.

Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

### Symbolische Darstellung



*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_NUM	Multiplikand (Faktor)
IN2	ANY_NUM	Multiplikator (Faktor)
INn	ANY_NUM	Multiplikator (Faktor)
OUT	ANY_NUM	Produkt

**2.22 MUX:** Multiplexer

Die Funktion übergibt in Abhängigkeit vom Wert am Eingang K den entsprechende Eingang an den Ausgang.

z.B.

K = 0 → Eingang IN0 wird an den Ausgang übergeben

K = 1 → Eingang IN1 wird an den Ausgang übergeben

K = 5 → Eingang IN5 wird an den Ausgang übergeben

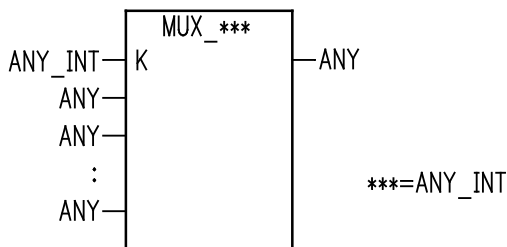
K = n → Eingang INn wird an den Ausgang übergeben

Es können die Datentypen der Gruppe ANY verarbeitet werden.

Die Datentypen an den Eingängen IN0 bis INn und am Ausgang müssen gleich sein.

Für die Verarbeitung der verschiedenen Datentypen (ANY\_INT) an Eingang K steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

**MUX***Symbolische Darstellung**Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
K	ANY_INT	Selektions-Eingang
IN0	ANY	0. Eingang
IN1	ANY	1. Eingang
IN2	ANY	2. Eingang
INn	ANY	n. Eingang
OUT	ANY	Ausgang

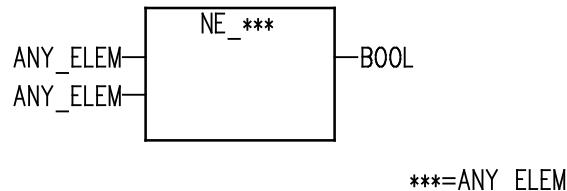
**NE**

## 2.23 NE: Ungleich

Die Funktion überprüft die Eingangswerte auf Ungleichheit.

Die Datentypen der Eingangswerte müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

### Symbolische Darstellung



### Zuweisung

OUT = 1, wenn IN1 <> IN2

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN1	ANY_ELEM	1. Eingang
IN2	ANY_ELEM	2. Eingang
OUT	BOOL	Ausgang

**NOT**

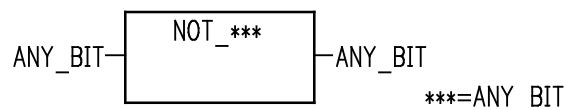
## 2.24 NOT: Negation

Die Funktion negiert bitweise die Eingangs-Bit-Folge und gibt das Ergebnis am Ausgang aus.

Es können die Datentypen der Gruppe ANY\_BIT verarbeitet werden.

Die Datentypen des Eingangs- und Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

### Symbolische Darstellung



### Zuweisung

OUT = NOT IN

*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN	ANY_BIT	Eingangs-Bit-Folge
OUT	ANY_BIT	Ausgangs-Bit-Folge

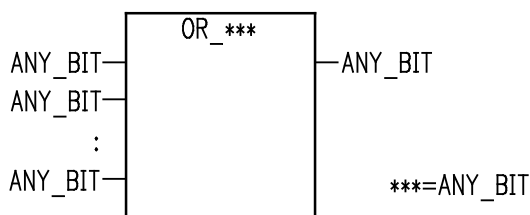
**2.25 OR: ODER-Funktion**

Die Funktion verknüpft (nach ODER-Logik) die Bit-Folgen an den Eingängen und gibt das Ergebnis am Ausgang aus. Die Verknüpfung erfolgt bitweise.

Es können die Datentypen der Gruppe ANY\_BIT verarbeitet werden.

Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Die Anzahl der Eingänge kann erhöht werden.

*Symbolische Darstellung**Zuweisung*

OUT = IN1 OR IN2 OR ... OR INn

*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_BIT	Eingangs-Bit-Folge
IN2	ANY_BIT	Eingangs-Bit-Folge
INn	ANY_BIT	Eingangs-Bit-Folge
OUT	ANY_BIT	Ausgangs-Bit-Folge

**OR**

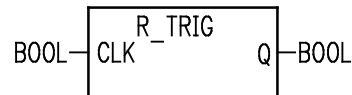
**R\_TRIG**

## 2.26 R\_TRIG: Erkennung steigender Flanken

Der Funktionsbaustein wird zur Erkennung steigender Flanken (0 → 1) verwendet.

Der Ausgang Q wird "1", wenn ein Übergang von "0" nach "1" am Eingang CLK erfolgt. Der Ausgang bleibt von einer Ausführung des Funktionsbausteins bis zur nächsten Ausführung auf "1"; danach kehrt der Ausgang zu "0" zurück.

*Symbolische Darstellung*



*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
CLK	BOOL	Takt-Eingang
Q	BOOL	Ausgang

**REAL\_TO**

## 2.27 REAL\_TO: Typumwandlung

Die Funktion konvertiert einen Eingangswert vom Datentyp REAL zu einem Datentyp der Gruppe ANY\_BIT oder ANY\_INT oder in den Datentyp TIME.

Bei der Konvertierung nach ANY\_BIT werden die niederwertigsten Bits des Eingangswertes zum Ausgang übertragen.

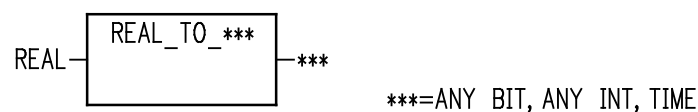
Bei der Konvertierung nach ANY\_INT und TIME wird nach den Vereinbarungen der IEC 559 gerundet, z.B.

1.4 → 1  
 1.5 → 2  
 2.4 → 2  
 2.5 → 2  
 2.6 → 3

Negative Eingangswerte können nicht in die Datentypen UDINT, UINT oder TIME gewandelt werden.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

*Symbolische Darstellung*



*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN	REAL	Eingangswert
OUT	ANY_BIT, ANY_INT, TIME	Ausgangswert

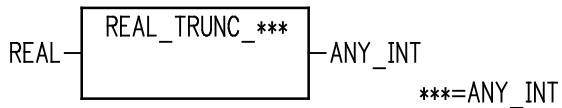
**2.28 REAL\_TRUNC: Typumwandlung**

Die Funktion konvertiert (mit Abschneiden in Richtung Null) einen Eingangswert vom Datentyp REAL zu einem Datentyp der Gruppe ANY\_INT z.B.

1.6 → 1  
 -1.6 → -1  
 1.4 → 1  
 -1.4 → -1

Negative Eingangswerte können nicht in die Datentypen UDINT oder UINT gewandelt werden.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

*Symbolische Darstellung**Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN	REAL	Eingangswert
OUT	ANY_INT	Ausgangswert

**REAL\_TRUNC**

**RS**

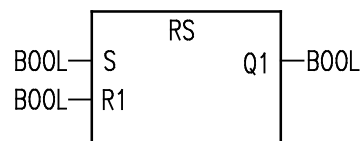
## 2.29 RS: Bistabiler Funktionsbaustein, Rücksetzen dominant

Der Funktionsbaustein wird als RS-Speicher mit der Eigenschaft "Rücksetzen dominant" verwendet.

Der Ausgang Q1 wird "1", wenn der Eingang S "1" wird. Dieser Zustand bleibt auch erhalten, wenn der Eingang S wieder "0" wird. Der Ausgang Q1 wird erst wieder "0", wenn der Eingang R1 "1" wird. Sind die Eingänge S und R1 gleichzeitig "1", setzt der dominierende Eingang R1 den Ausgang Q1 auf "0".

Der Anfangszustand von Q1 beim ersten Aufruf des Funktionsbausteins ist "0".

### Symbolische Darstellung



### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
S	BOOL	Setzen
R1	BOOL	Rücksetzen (dominant)
Q1	BOOL	Ausgang

**SR**

## 2.30 SR: Bistabiler Funktionsbaustein, Setzen dominant

Der Funktionsbaustein wird als SR-Speicher mit der Eigenschaft "Setzen dominant" verwendet.

Der Ausgang Q1 wird "1", wenn der Eingang S1 "1" wird. Dieser Zustand bleibt auch erhalten, wenn der Eingang S1 wieder "0" wird. Der Ausgang Q1 wird erst wieder "0", wenn der Eingang R "1" wird. Sind die Eingänge S1 und R gleichzeitig "1", setzt der dominierende Eingang S1 den Ausgang Q1 auf "1".

Der Anfangszustand von Q1 beim ersten Aufruf des Funktionsbausteins ist "0".

### Symbolische Darstellung



### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
S1	BOOL	Setzen (dominant)
R	BOOL	Rücksetzen
Q1	BOOL	Ausgang



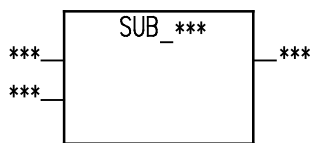
## 2.31 SUB: Subtraktion

Die Funktion subtrahiert den Wert am Eingang IN2 vom Wert am Eingang IN1 und gibt das Ergebnis am Ausgang aus.

Es können die Datentypen der Gruppe ANY\_NUM und der Datentyp TIME verarbeitet werden.

Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

### Symbolische Darstellung



\*\*\*=ANY NUM, TIME

### Zuweisung

OUT = IN1 - IN2

### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN1	ANY_NUM, TIME	Minuend
IN2	ANY_NUM, TIME	Subtrahend
OUT	ANY_NUM, TIME	Differenz

**SUB**

**TIME\_TO**

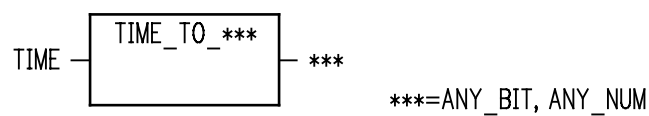
### 2.32 TIME\_TO: Typumwandlung

Die Funktion konvertiert einen Eingangswert vom Datentyp TIME zu einem Datentyp der Gruppe ANY\_BIT oder ANY\_NUM.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Bei der Konvertierung eines Eingangswertes vom Datentyp TIME zu einem Ausgangswert vom Datentyp BOOL, BYTE, WORD, INT oder UINT werden jeweils die niederwertigsten Bits vom Eingang zum Ausgang übertragen.

#### Symbolische Darstellung



#### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	TIME	Eingangswert
OUT	ANY_BIT, ANY_NUM	Ausgangswert

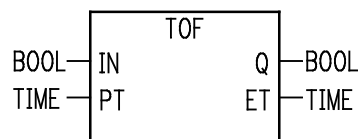
**TOF**

### 2.33 TOF: Ausschaltverzögerung

Der Funktionsbaustein wird als Ausschaltverzögerung verwendet.

Der Anfangszustand von ET beim ersten Aufruf des Funktionsbausteins ist "0".

#### Symbolische Darstellung

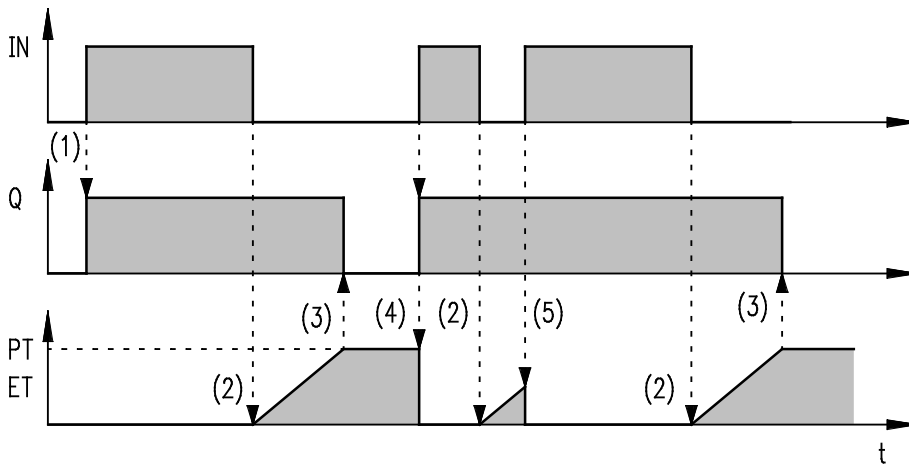


#### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	BOOL	Verzögerung starten
PT	TIME	Voreinstellung der Verzögerungszeit
Q	BOOL	Ausgang
ET	TIME	interne Zeit

*Detailbeschreibung*

## Taktdiagramm der Ausschaltverzögerung TOF



- 1 Wenn IN "1" wird, wird Q "1".
- 2 Wenn IN "0" wird, wird die interne Zeit (ET) gestartet.
- 3 Erreicht die interne Zeit den Wert von PT, wird Q "0".
- 4 Wenn IN "1" wird, wird Q "1" und die interne Zeit gestoppt/rückgesetzt.
- 5 Wird IN "1" bevor die interne Zeit den Wert von PT erreicht hat, wird die interne Zeit gestoppt/rückgesetzt, ohne dass Q "0" wurde.

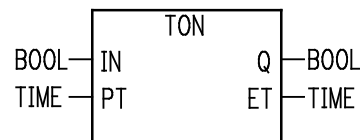
**TON**

## 2.34 TON: Einschaltverzögerung

Der Funktionsbaustein wird als Einschaltverzögerung verwendet.

Der Anfangszustand von ET beim ersten Aufruf des Funktionsbausteins ist "0".

### Symbolische Darstellung

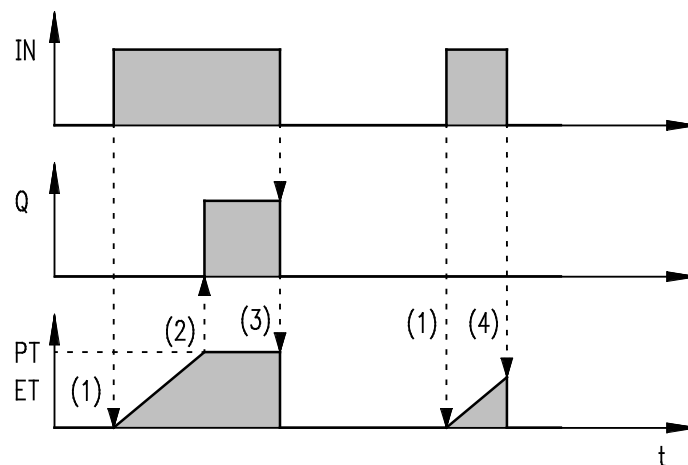


### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	BOOL	Verzögerung starten
PT	TIME	Voreinstellung der Verzögerungszeit
Q	BOOL	Ausgang
ET	TIME	interne Zeit

### Detailbeschreibung

Taktdiagramm der Einschaltverzögerung TON



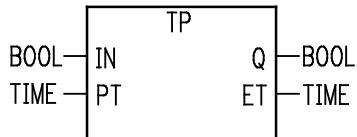
- 1 Wenn IN "1" wird, wird die interne Zeit (ET) gestartet.
- 2 Erreicht die interne Zeit den Wert von PT, wird Q "1".
- 3 Wird IN "0", wird Q "0" und die interne Zeit gestoppt/rückgesetzt.
- 4 Wird IN "0" bevor die interne Zeit den Wert von PT erreicht hat, wird die interne Zeit gestoppt/rückgesetzt, ohne dass Q "1" wurde.

### 2.35 TP: Puls

Der Funktionsbaustein wird zur Erzeugung eines Pulses mit definierter Zeitdauer verwendet.

Der Anfangszustand von ET beim ersten Aufruf des Funktionsbausteins ist "0".

#### Symbolische Darstellung

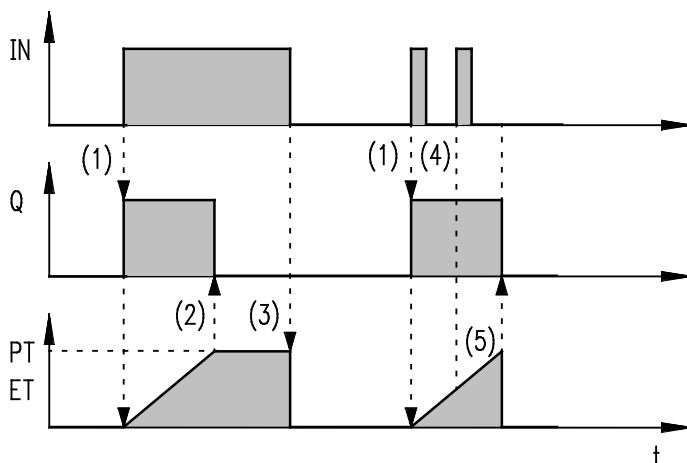


#### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	BOOL	Puls auslösen
PT	TIME	Voreinstellung der Pulsdauer
Q	BOOL	Ausgang
ET	TIME	interne Zeit

#### Detailbeschreibung

Taktdiagramm vom Puls TP



- 1 Wenn IN "1" wird, wird Q "1" und die interne Zeit (ET) gestartet.
- 2 Erreicht die interne Zeit den Wert von PT, wird Q "0" (unabhängig von IN).
- 3 Die interne Zeit wird gestoppt/rückgesetzt, wenn IN "0" wird.
- 4 Hat die interne Zeit den Wert von PT noch nicht erreicht, hat ein Takt an IN keinen Einfluss auf die interne Zeit.
- 5 Hat die interne Zeit den Wert von PT erreicht und ist IN "0", wird die interne Zeit gestoppt/rückgesetzt und Q wird "0".

TP

**WORD\_TO**

### 2.36 WORD\_TO: Typumwandlung

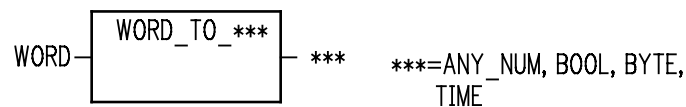
Die Funktion konvertiert einen Eingangswert vom Datentyp WORD zu einem Datentyp der Gruppe ANY\_NUM oder den Datentypen BOOL, BYTE oder TIME.

Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung.

Bei der Konvertierung des Datentyps WORD in den Datentyp DINT, UDINT, REAL oder TIME wird das Bitmuster des Eingangs in die niederwertigsten Bits des Ausgangs übertragen. Die höherwertigen Bits des Ausgangs werden auf null gesetzt.

Bei der Konvertierung des Datentyps WORD in den Datentyp BOOL oder BYTE, werden die niederwertigsten Bits des Eingangswertes zum Ausgang übertragen.

#### Symbolische Darstellung



#### Parameterbeschreibung

Parameter	Datentyp	Bedeutung
IN	WORD	Eingangswert
OUT	ANY_NUM, BOOL, BYTE, TIME	Ausgangswert

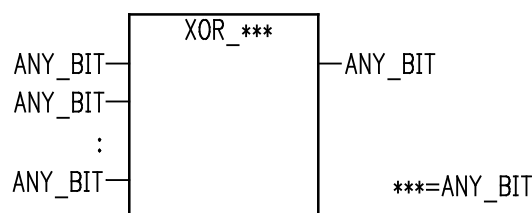
**XOR**

### 2.37 XOR: Exklusiv-ODER-Funktion

Die Funktion verknüpft (nach Exklusiv-ODER-Logik) die Bit-Folgen an den Eingängen und gibt das Ergebnis am Ausgang aus. Die Verknüpfung erfolgt bitweise.

Es können die Datentypen der Gruppe ANY\_BIT verarbeitet werden. Die Datentypen aller Eingangswerte und der des Ausgangswertes müssen gleich sein. Für die Verarbeitung der verschiedenen Datentypen steht jeweils eine spezielle Funktion zur Verfügung. Die Anzahl der Eingänge kann erhöht werden.

#### Symbolische Darstellung



#### Zuweisung

OUT = IN1 XOR IN2 XOR ... XOR INn

*Parameterbeschreibung*

Parameter	Datentyp	Bedeutung
IN1	ANY_BIT	Eingangs-Bit-Folge
IN2	ANY_BIT	Eingangs-Bit-Folge
INn	ANY_BIT	Eingangs-Bit-Folge
OUT	ANY_BIT	Ausgangs-Bit-Folge

### 3 Programmiersprache „Strukturierter Text“ (ST) nach IEC-61131-3

**Vorbemerkung:** Die nachfolgende Liste der Befehle und Schlüsselwörter ist nicht vollständig. Die Darstellung und die Syntax der Befehle ist an die IEC-61131-3 Norm und an die Software Concept angelehnt.

#### 3.1 Anweisungen

Anweisungen sind die "Befehle" der Programmiersprache ST. Sie müssen durch Semikolon abgeschlossen werden. Es können mehrere Anweisungen (durch Semikolon getrennt) in einer Zeile stehen.

Die folgenden Anweisungen stehen zur Verfügung:

Zuweisungen  
 VAR...END\_VAR  
 Funktionsbaustein-Aufrufe  
 IF...THEN...END\_IF  
 ELSE  
 ELSIF...THEN  
 CASE...OF...END\_CASE  
 FOR...TO...BY...DO...END\_FOR  
 WHILE...DO...END\_WHILE  
 REPEAT  
 EXIT  
 Leeranweisung

#### 3.2 Zuweisungen

Die Zuweisung ersetzt den aktuellen Wert einer Einzel- oder Multi-Elementvariablen durch das Ergebnis der Auswertung eines Ausdrucks. Eine Zuweisung besteht aus einer Variablenangabe auf der linken Seite, gefolgt vom Zuweisungsoperator ":=", gefolgt vom Ausdruck, der auszuwerten ist. Beide Variablen müssen den gleichen Datentyp haben.

*Beispiele:*

A := B ;

Der Variablen A wird der aktuelle Wert der Variablen B zugewiesen.

Die Zuweisung wird auch benutzt um Variablen ein Literal zuzuweisen.

C := 15 ;

Um einer Variablen einen Wert zuzuweisen, der von einer Funktion oder einem Funktionsbaustein zurückgegeben wird, ist folgende Form zu verwenden:

X := MOD\_INT(C,A) ;  
 Y := TON1.Q ;

Und um einer Variablen einen Wert zuzuweisen, der das Ergebnis einer Operation ist, kann folgende Schreibweise eingesetzt werden:

X := (A+B-C)\*D ;



### 3.3 VAR...END\_VAR

Die VAR-Anweisung dient zur Deklaration der verwendeten Funktionsbausteine und der Deklaration von direkten Adressen, falls diese nicht mit dem Default-Datentyp verwendet werden sollen.

Die END\_VAR-Anweisung kennzeichnet das Ende der Deklaration.  
VAR...END\_VAR wird nur einmal am Anfang der Section eingegeben. Hier müssen alle verwendeten FBs/DFBs und direkten Adressen (die vom Default-Datentyp abweichen) deklariert werden.

*Beispiel:*

```
VAR
  TIMER : TON;
  TAKT : SYSCLOCK;
END_VAR
```

#### Funktionsbaustein-Aufrufe

Es können alle Funktionsbausteine aufgerufen werden, die durch eine FBS-Bibliothek zur Verfügung gestellt werden. Die Parameter sind entsprechend des Funktionsbausteins zu übergeben.

Funktionsbausteine werden durch eine Anweisung aufgerufen, die aus dem Exemplar-Namen des FBs besteht, dem eine eingeklammerte Liste von Wertzuweisungen (Aktualparameter) an die Formalparameter folgt. Die Reihenfolge, in der die Formalparameter in einem Funktionsbaustein-Aufruf aufgezählt sind, ist nicht signifikant. Es ist nicht erforderlich allen Formalparametern einen Wert zuzuweisen. Falls einem Formalparameter kein Wert zugewiesen wird, wird bei der Ausführung des Funktionsbausteins der im Variablen-Editor festgelegte Initialwert verwendet. Wurde kein Initialwert definiert, wird der Defaultwert (0) verwendet.

Auch wenn der Funktionsbaustein keine Eingänge hat oder die Eingänge nicht parametrisiert werden sollen, muss der Funktionsbaustein aufgerufen werden, bevor dessen Ausgänge verwendet werden können.

*Beispiel:*

```
TAKT ();
ZAEHL (CU:=TAKT.CLK3, R:=reset; PV:=100);
PULSE (IN:=ZAEHL.Q, PT:=t#1s);
```

### VAR...END\_VAR

**IF...THEN...END\_IF**

### 3.4 IF...THEN...END\_IF

Die IF-Anweisung legt fest, dass eine Anweisung oder eine Gruppe von Anweisungen nur ausgeführt wird, wenn der zugehörige boolsche Ausdruck den Wert 1 (wahr) hat. Falls die Bedingung 0 (falsch) ist, wird die Anweisung bzw. die Anweisungsgruppe nicht ausgeführt.

Die THEN-Anweisung kennzeichnet das Ende der Bedingung und den Anfang der Anweisung(en).

Die END\_IF-Anweisung kennzeichnet das Ende der Anweisung(en).

*Beispiele:*

```
IF A>B THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=C - A ;
END_IF ;
```

Mit NOT kann die Bedingung invertiert werden (Ausführung der Anweisungen bei 0).

```
IF NOT FLAG THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=C - A ;
END_IF ;
```

**ELSE**

### 3.5 ELSE

Die ELSE-Anweisung folgt immer einer IF...THEN-, ELSIF...THEN- oder CASE-Anweisung.

Folgt die ELSE-Anweisung auf IF oder ELSIF, wird die Anweisung oder die Gruppe von Anweisungen nur ausgeführt, wenn die zugehörigen boolschen Ausdrücke der IF- oder ELSIF- Anweisung den Wert 0 (falsch) haben. Falls die Bedingung der IF- oder ELSIF-Anweisung 1 (wahr) ist, wird die Anweisung bzw. die Anweisungsgruppe nicht ausgeführt.

Folgt die ELSE-Anweisung auf CASE, wird die Anweisung oder die Gruppe von Anweisungen nur ausgeführt, wenn keine Marke den Wert des Selektors enthält. Falls eine Marke den Wert des Selektors enthält, wird die Anweisung bzw. die Anweisungsgruppe nicht ausgeführt.

```
IF A>B THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=C - A ;
ELSE
  C:=A + B ;
  B:=C * A ;
END_IF ;
```

Es können beliebig viele IF...THEN...ELSE...END\_IF-Anweisungen verschachtelt werden, um komplexe Auswahlanweisungen zu erzeugen.

### 3.6 ELSIF...THEN

Die ELSIF-Anweisung folgt immer einer IF...THEN-Anweisung. Die ELSIF-Anweisung legt fest, dass eine Anweisung oder eine Gruppe von Anweisungen nur ausgeführt wird, wenn der zugehörige boolsche Ausdruck der IF-Anweisung den Wert 0 (falsch) hat und der zugehörige boolsche Ausdruck der ELSIF-Anweisung den Wert 1 (wahr) hat. Falls die Bedingung der IF-Anweisung 1 (wahr) ist oder die Bedingung der ELSIF-Anweisung 0 (falsch) ist, wird die Anweisung bzw. die Anweisungsgruppe nicht ausgeführt.

Die THEN-Anweisung kennzeichnet das Ende der ELSIF-Bedingung(en) und den Anfang der Anweisung(en).

*Beispiele:*

```
IF A>B THEN
  C:=SIN_REAL(A) * COS_REAL(B) ;
  B:=SUB_REAL(C,A) ;
ELSIF A=B THEN
  C:=ADD_REAL(A,B) ;
  B:=MUL_REAL(C,A) ;
END_IF ;
```

Es können beliebig viele IF...THEN...ELSIF...THEN...END\_IF-Anweisungen verschachtelt werden, um komplexe Auswahlanweisungen zu erzeugen.

```
IF A>B THEN
  IF B=C THEN
    C:=SIN_REAL(A) * COS_REAL(B) ;
  ELSE
    B:=SUB_REAL(C,A) ;
  END_IF ;
ELSIF A=B THEN
  C:=ADD_REAL(A,B) ;
  B:=MUL_REAL(C,A) ;
ELSE
  C:= DIV_REAL (A,B) ;
END_IF ;
```

#### ELSIF...THEN

### CASE...OF... END\_CASE

## 3.7 CASE...OF...END\_CASE

Die CASE-Anweisung besteht aus einem Ausdruck vom Datentyp INT (dem "Selektor") und einer Liste von Anweisungsgruppen. Jede Gruppe wird mit einer Marke versehen, die aus einer oder mehreren Ganzzahlen (ANY\_INT) oder Bereichen von ganzzahligen Werten besteht. Es wird die erste Gruppe, von Anweisungen ausgeführt, deren Marke den berechneten Wert des Selektors enthält. Andernfalls wird keine der Anweisungen ausgeführt.

Die OF-Anweisung kennzeichnet den Anfang der Marken.  
Innerhalb der CASE-Anweisung kann eine ELSE-Anweisung erfolgen, deren Anweisungen ausgeführt werden, falls keine Marke den Wert des Selektors enthält.  
Die END\_CASE-Anweisung kennzeichnet das Ende der Anweisung(en).

*Beispiel:*

```
CASE SELECT OF      1..4: X:=A-C/2;
                   5: X:=A-C/3;
                   6..10: X:=A-C/5;

ELSE X:=A;
END_CASE;
```

### FOR...TO...BY... DO...END\_FOR

## 3.8 FOR...TO...BY...DO...END\_FOR

Die FOR-Anweisung wird benutzt, falls die Anzahl der Wiederholungen im Voraus bestimmt werden kann. Andernfalls werden WHILE oder REPEAT benutzt.

Die FOR-Anweisung wiederholt eine Anweisungsfolge bis zur END\_FOR-Anweisung. Die Anzahl der Wiederholungen wird durch den Anfangswert, den Endwert und die Steuervariable bestimmt. Anfangswert, Endwert und Steuervariable müssen vom gleichen Datentyp sein (DINT oder INT) und dürfen nicht durch eine der wiederholten Anweisungen verändert werden. Die FOR-Anweisung inkrementiert den Wert der Steuervariablen von einem Anfangswert zu einem Endwert. Der Wert des Inkrements ist auf 1 voreingestellt. Die Prüfung der Endbedingung wird am Anfang jeder Wiederholung durchgeführt, sodass die Anweisungsfolge nicht ausgeführt wird, falls der Anfangswert den Endwert überschritten oder unterschritten hat.

Für den Endwert gilt ein leicht eingeschränkter Wertebereich  
INT: - 32 767 bis 32 766  
DINT: - 2 147 483 646 bis 2 147 483 645

Die DO-Anweisung kennzeichnet das Ende der Wiederholungsdefinition und den Anfang der Anweisung(en).

Die Wiederholung kann mit der EXIT-Anweisung vorzeitig beendet werden.

Die END\_FOR-Anweisung kennzeichnet das Ende der Anweisung(en).

Soll ein anderes Inkrement als 1 verwendet werden, kann dies durch BY definiert werden. Das Inkrement, der Anfangswert, der Endwert und die Steuervariable müssen vom gleichen Datentyp sein (DINT oder INT). Das Kriterium der Bearbeitungsrichtung (vorwärts, rückwärts) ist das Vorzeichen des BY - Ausdrucks. Ist dieser Ausdruck positiv, so läuft die Schleife aufwärts, ist er negativ, läuft die Schleife abwärts.

*Beispiele:*

```
FOR I := 1 TO 100 DO  
    A := COS_REAL(I);  
END_FOR;
```

```
FOR I:= 100 TO 1 BY -5 DO  
    A := COS_REAL(I);  
END_FOR;
```

### 3.9 WHILE...DO...END\_WHILE

Die WHILE-Anweisung bewirkt, dass eine Folge von Anweisungen wiederholt ausgeführt wird, bis der zugehörige boolsche Ausdruck 0 (falsch) ist. Falls der Ausdruck von Anfang an falsch ist, wird die Gruppe der Anweisungen überhaupt nicht ausgeführt.

Die DO-Anweisung kennzeichnet das Ende der Wiederholungsdefinition und den Anfang der Anweisung(en).

Die Wiederholung kann mit der EXIT-Anweisung vorzeitig beendet werden.

Die END\_WHILE-Anweisung kennzeichnet das Ende der Anweisung(en).

*Beispiel:*

```
WHILE X <= 1000 DO  
    X := X+10;  
END_WHILE;
```

### 3.10 REPEAT...UNTIL...END\_REPEAT

Die REPEAT-Anweisung bewirkt, dass eine Folge von Anweisungen wiederholt (mindestens einmal) ausgeführt wird, bis die zugehörige boolsche Bedingung 1 (wahr) ist. Die UNTIL-Anweisung kennzeichnet die Endebedingung.

Die Wiederholung kann mit der EXIT-Anweisung vorzeitig beendet werden.

Die END\_REPEAT-Anweisung kennzeichnet das Ende der Anweisung(en).

*Beispiel:*

```
REPEAT  
    X := X+1;  
UNTIL X = 100  
END_REPEAT;
```

**WHILE...DO...  
END\_WHILE**

**REPEAT...UNTIL...  
END\_REPEAT**

**EXIT**

### 3.11 EXIT

Die EXIT-Anweisung wird benutzt, um Wiederholungsanweisungen (FOR, WHILE, REPEAT) zu beenden, bevor die Ende-Bedingung erfüllt ist.

Wenn die EXIT-Anweisung innerhalb einer geschachtelten Wiederholung liegt, wird die innerste Schleife (in der EXIT liegt) verlassen. Als Nächstes wird die erste Anweisung nach dem Schleifenende (END\_FOR, END\_WHILE oder END\_REPEAT) ausgeführt.

*Beispiel:*

```
SUM := 0 ;
FOR I := 1 TO 3 DO
  FOR J := 1 TO 2 DO
    IF FLAG=1 THEN EXIT;
  END_IF ;
  SUM := SUM + J ;
END_FOR ;
SUM := SUM + I ;
END_FOR
```

### 3.12 Leeraanweisung

Leeraanweisungen werden durch ein Semikolon (;) erzeugt.

### 3.13 Kommentare

Kommentare der Programme beginnen mit der Zeichenfolge (\* und enden mit der Zeichenfolge \*). Zwischen diesen beiden Zeichenfolgen kann ein beliebiger Kommentar eingegeben werden. Kommentare können an einer beliebigen Position eingegeben werden.

Geschachtelte Kommentare sind nicht zulässig.

*Beispiel:*

(\* Hier beginnt das Hauptprogramm \*)

### 3.14 Operatoren

Ein Operator ist ein Symbol für eine auszuführende arithmetische oder logische Operation.

Operatoren sind generisch, d.h. sie passen sich automatisch dem Datentyp des Operanden an. Die Auswertung eines Ausdrucks besteht aus Anwenden der Operatoren auf die Operanden, in der Reihenfolge, die durch die Rangfolge der Operatoren definiert ist. Der Operator mit der höchsten Rangfolge in einem Ausdruck wird zuerst ausgeführt, gefolgt vom Operator der nächst-niedrigeren Rangfolge usw. bis die Auswertung vollendet ist. Operatoren mit gleichem Rang werden von links nach rechts ausgeführt, wie im Ausdruck beschrieben. Diese Reihenfolge kann durch Klammerung geändert werden.

**	Potenzierung
-	Negation
NOT	Komplement
*	Multiplikation
/	Division
MOD	Modulo
+	Addition
-	Subtraktion
<	Kleiner-Vergleich
>	Größer-Vergleich
<=	Kleiner-gleich-Vergleich
>=	Größer-gleich-Vergleich
=	Gleichheit
<>	Ungleichheit
&, AND	Logisches UND
XOR	Logisches Exklusiv-ODER
OR	Logisches ODER

## 4 Programmiersprache „Anweisungsliste“ (AWL) nach IEC-61131-3

**Vorbemerkung:** Die nachfolgende Liste der Befehle und Schlüsselwörter ist nicht vollständig. Die Darstellung und die Syntax der Befehle ist an die IEC-61131-3 Norm und an die Software Concept angelehnt.

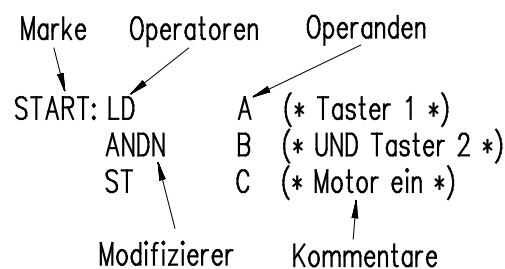
Mit der Programmiersprache „Anweisungsliste“ können Funktionsbausteine und Funktionen bedingt oder unbedingt aufgerufen, Zuweisungen ausgeführt und Sprünge innerhalb der Section bedingt oder unbedingt ausgeführt werden.

### 4.1 Anweisungen

Eine Anweisungsliste setzt sich aus einer Folge von Anweisungen zusammen.

Jede Anweisung beginnt in einer neuen Zeile und wird von einem Operator ggf. mit Modifizierer und, falls erforderlich für die jeweilige Operation, von einem oder mehreren Operanden gefolgt. Falls mehrere Operanden verwendet werden, werden diese durch Kommata getrennt. Vor der Anweisung kann eine Marke stehen, die von einem Doppelpunkt gefolgt wird.

*Beispiele:*



```

LD      run_pulse
STN     RUN_TIMER.IN
CAL     RUN_TIMER(PT := t#1s)
LD      RUN_TIMER.ET
ST      animatetime
LD      RUN_TIMER.Q
ST      run_pulse
JMPCN   ende
ende:   LD      run_light
        ST      run_light1
  
```



## 4.2 Operatoren

Ein Operator ist ein Symbol für eine auszuführende arithmetische Operation, eine auszuführende logische Operation oder den Aufruf einer Funktion. Operatoren sind generisch, d.h. sie passen sich automatisch dem Datentyp des Operanden an.

LD	Lädt den Wert des Operanden in den Akku
ST	Speichert den Wert des Akkus im Operanden
S	Setzt den Operanden auf 1, wenn der Akku-Inhalt 1 ist
R	Setzt den Operanden auf 0, wenn der Akku-Inhalt 1 ist
AND	Logisches UND
OR	Logisches ODER
XOR	Logisches Exklusiv-ODER
ADD	Addition
SUB	Subtraktion
MUL	Multiplikation
DIV	Division
GT	Vergleich: >
GE	Vergleich: >=
EQ	Vergleich: =
NE	Vergleich: <>
LE	Vergleich: <=
LT	Vergleich: <
JMP	Sprung zur Marke
CAL	Aufruf eines Funktionsbausteins
FUNCNAME	Ausführen einer Funktion
)	Bearbeitung zurückgestellter Operationen

## 4.3 Operanden

Ein Operand ist ein Literal, eine Variable, eine Multielement-Variable, ein Element einer Multielement-Variablen, FB-/DFB-Ausgang oder eine direkte Adresse.

## 4.4 Modifizierer

Modifizierer beeinflussen die Ausführung des vorangehenden Operators.

### 4.4.1 N

Der Modifizierer N wird verwendet, um den Wert eines Operanden bitweise zu invertieren.

Der Modifizierer kann nur auf Operanden vom Datentyp ANY\_BIT angewendet werden.

*Beispiel:*

```
LD A
ANDN B
ST C
```

N

**C**

#### 4.4.2 C

Der Modifizierer C wird verwendet, um die zugehörige Anweisung auszuführen, falls der Wert des Akkus "1" (TRUE) ist.

Der Modifizierer kann nur auf Operanden vom Datentyp BOOL angewendet werden.

*Beispiele:*

```
LD A
AND B
JMPC START
```

CN, Kombination von C und N

```
LD A
AND B
JMPCN START
```

**(**

#### 4.4.3 ( Linke Klammer

Der Modifizierer linke Klammer "(" wird verwendet, um die Auswertung des Operanden zurückzustellen, bis der Operator rechte Klammer ")" erscheint. Die Anzahl der rechte Klammer-Operationen muss gleich der Anzahl der linke Klammer-Modifizierer sein. Klammern können verschachtelt werden.

*Beispiel:*

```
LD A
AND B
AND( C
OR D
)
ST E
```

**VAR...END\_VAR**

#### 4.4.4 VAR...END\_VAR

Die VAR-Anweisung dient zur Deklaration der verwendeten Funktionsbausteine und der Deklaration von direkten Adressen, falls diese nicht mit dem Default-Datentyp verwendet werden sollen.

Die END\_VAR-Anweisung kennzeichnet das Ende der Deklaration.

VAR...END\_VAR wird nur einmal am Anfang der Section eingegeben. Hier müssen alle verwendeten FBs/DFBs und direkten Adressen (die vom Default-Datentyp abweichen) deklariert werden.

*Beispiel:*

```
VAR
  RUN_TIMER : TON; (* Blink timer *)
  TAKT : SYSCLOCK;
  ZAEHL : CTU_DINT;
END_VAR
```

## 4.5 Funktionsbaustein-Aufrufe

Der Aufruf kann in 3 Formen ausgeführt werden:

mit CAL und einer Liste der Eingangsparameter  
mit CAL und Laden/Speichern der Eingangsparameter  
durch Gebrauch der Eingangsoperatoren

Auch wenn der Funktionsbaustein keine Eingänge hat oder die Eingänge nicht parametrisiert werden sollen, muss der Funktionsbaustein aufgerufen werden, bevor dessen Ausgänge verwendet werden können.

*Beispiel:*

```
VAR
  TAKT : SYSCLOCK;
  ZAEHL : CTU_DINT;
END_VAR
CAL TAKT ()
CAL ZAEHL (CU := takt, R := %IX10, PV := 100)
LD ZAEHL.Q
ST %QX1
```

## 4.6 Funktions-Aufrufe

Funktionen werden durch eine Anweisungsliste aufgerufen, die aus dem Laden des ersten Aktualparameters in den Akku und dem Namen der Funktion besteht. Diese wird, falls erforderlich, von einer Liste der weiteren Aktualparameter gefolgt. Die Reihenfolge, in der die Formalparameter in einem Funktions-Aufruf aufgezählt sind, ist signifikant. Die Liste der Aktualparameter kann direkt nach einem Komma umgebrochen werden. Das Ergebnis der Funktion wird nach dem Ausführen der Funktion zum Akku-Inhalt und kann durch ST in einem Operanden gespeichert oder direkt weiterverarbeitet werden.

*Beispiel:*

```
LD A
SIN_REAL
ST ergebnis
```

## 4.7 Kommentare

Kommentare beginnen mit der Zeichenfolge (\* und enden mit der Zeichenfolge \*). Zwischen diesen beiden Zeichenfolgen kann ein beliebiger Kommentar eingegeben werden. Kommentare können an einer beliebigen Position eingegeben werden. Geschachtelte Kommentare sind nicht zulässig.

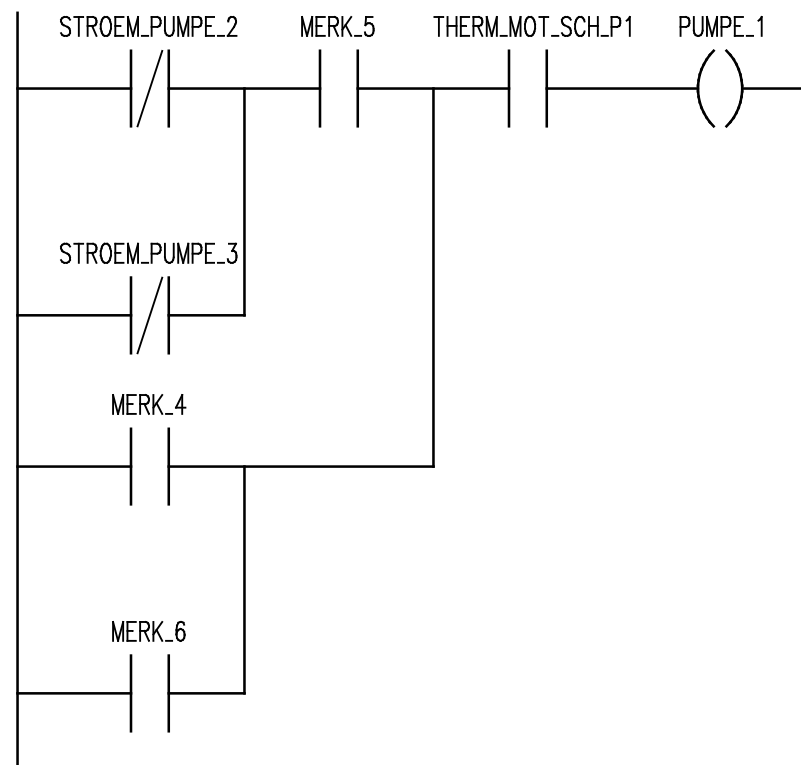
*Beispiel:*

(\* Hier beginnt das Hauptprogramm \*)

## 5 Programmiersprache „Kontaktplan“ (KOP) nach IEC-61131-3

**Vorbemerkung:** Die nachfolgende Liste der Symbole und Darstellungen ist nicht vollständig. Die Darstellung ist an die IEC-61131-3 Norm angelehnt.

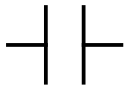
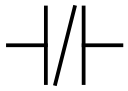
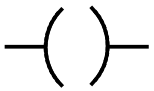
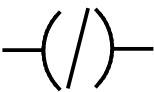
*Beispiel:*





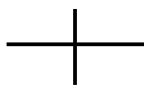
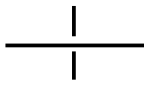
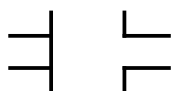
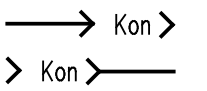
### 5.1 Kontakte und Spulen

Ein Kontakt ist ein KOP-Element, das einen Zustand an die horizontale Verbindung auf seiner rechten Seite übergibt. Dieser Zustand ergibt sich aus der booleschen UND-Verknüpfung des Zustands der horizontalen Verbindung auf seiner linken Seite mit dem Zustand der zugehörigen Variablen/direkten Adresse. Ein Kontakt verändert nicht den Wert der zugehörigen Variablen/direkten Adresse.

Eine Spule ist ein LD-Element, das den Zustand der horizontalen Verbindung auf seiner linken Seite unverändert an die horizontale Verbindung auf seiner rechten Seite übergibt. Dabei wird der Zustand in der zugehörigen Variablen/direkten Adresse gespeichert.

Element	Darstellung
Schließer (Abfrage auf den booleschen Wert "1")	
Öffner (Abfrage auf den booleschen Wert "0")	
Spule (Ergebniszuweisung)	
Negative Spule (Negierte Ergebniszuweisung)	

## 5.2 Linien

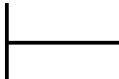
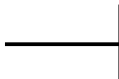

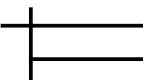
Element	Darstellung
Horizontale Linie	
Vertikale Linie	
Linienverbindung (horizontal u. vertikal)	
Linienkreuzung (ohne Verbindung)	
Ecken (verbunden und nicht verbunden)	
Konnektor	

### 5.3 Stromschienen und Verbindungen

Verbindungen sind Verknüpfungen zwischen Kontakten, Spulen und FFBs.

Es können mehrere Verbindungen mit einem Kontakt, einer Spule oder einem FFB-Ausgang verbunden werden. Die Verbindungsstellen werden dabei durch einen gefüllten Kreis gekennzeichnet.

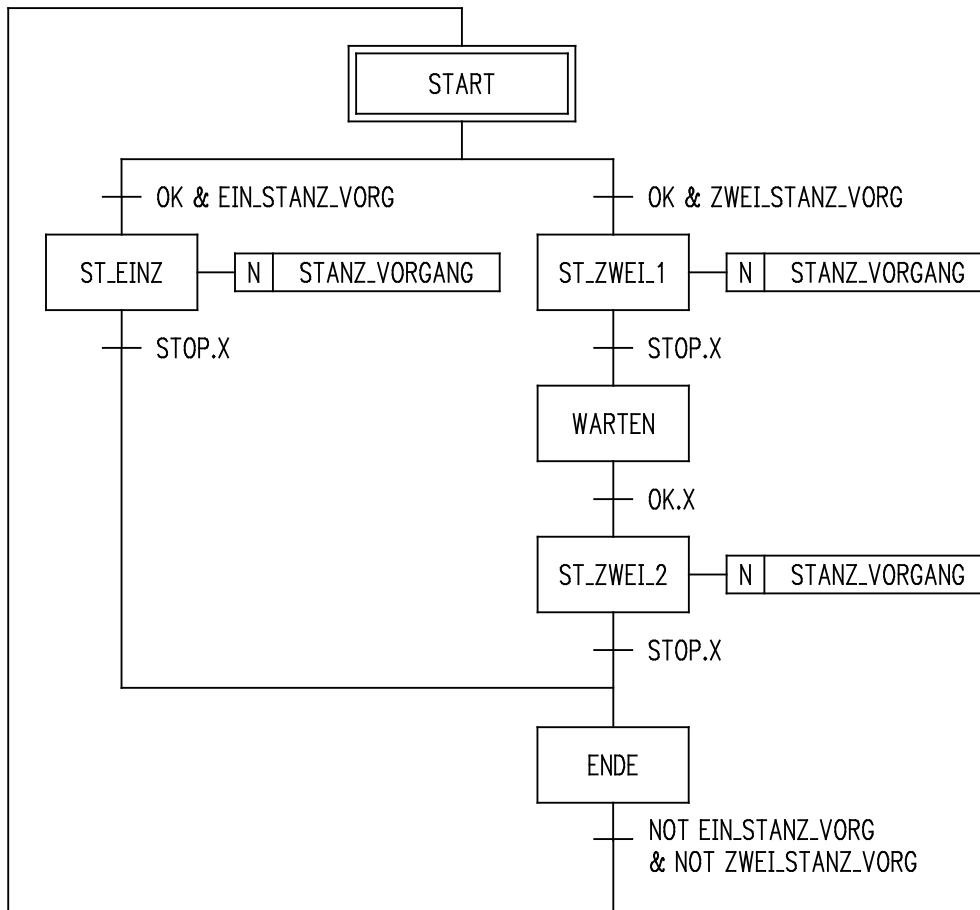
Die Datentypen der zu verbindenden Ein-/Ausgänge müssen übereinstimmen.

Element	Darstellung
Linke Stromschiene mit angebundener horizontaler Verbindung	
Rechte Stromschiene mit angebundener horizontaler Verbindung	
Horizontale Verbindung	
Vertikale Verbindung mit angebundenen horizontalen Verbindungen	

## 6 Programmiersprache „Ablaufsprache“ (AS) nach IEC-61131-3

**Vorbemerkung:** Die nachfolgende Liste der Symbole und Darstellungen ist nicht vollständig. Die Darstellung ist an die IEC-61131-3 Norm angelehnt.

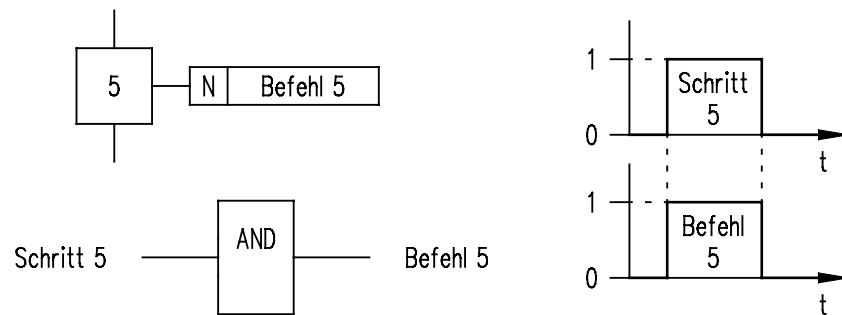
*Beispiel:*



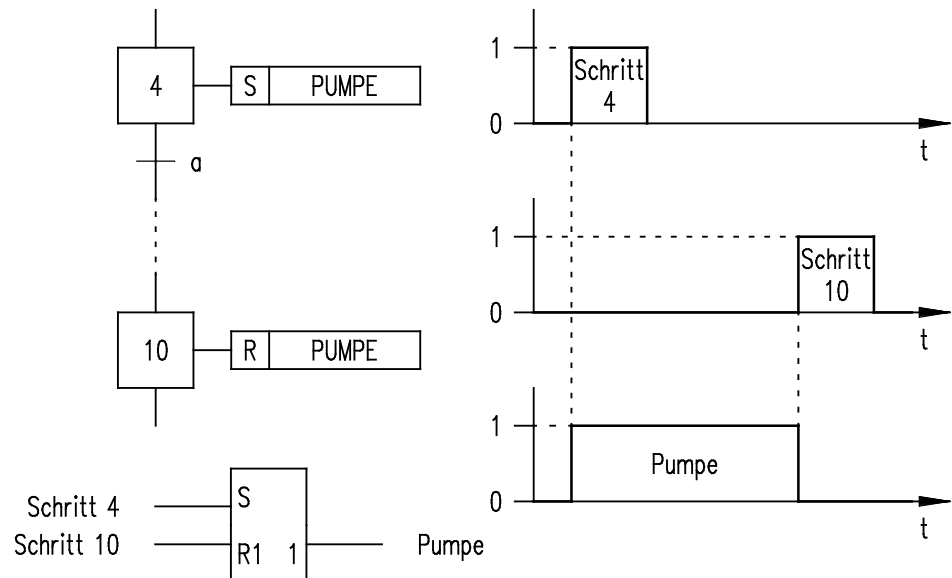
### 6.1 Bestimmungszeichen für Befehle

Bestimmungs- zeichen	Bedeutung
kein oder N	nicht gespeichert (Not stored)
S	Setzen, gespeichert (Set)
R	Rücksetzen, gespeichert (Reset)
D	Zeitverzögert (Delayed)
L	Zeitbegrenzt (Limited)
P	Puls (Pulse)

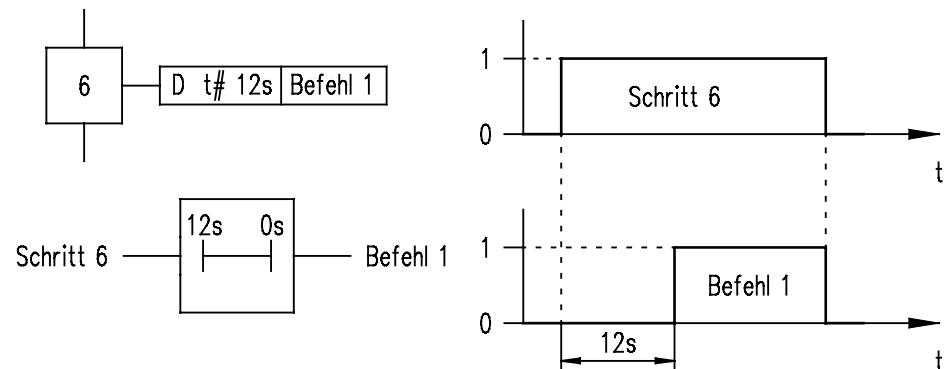
### 6.1.1 Bestimmungszeichen N (oder kein)



### 6.1.2 Bestimmungszeichen S und R

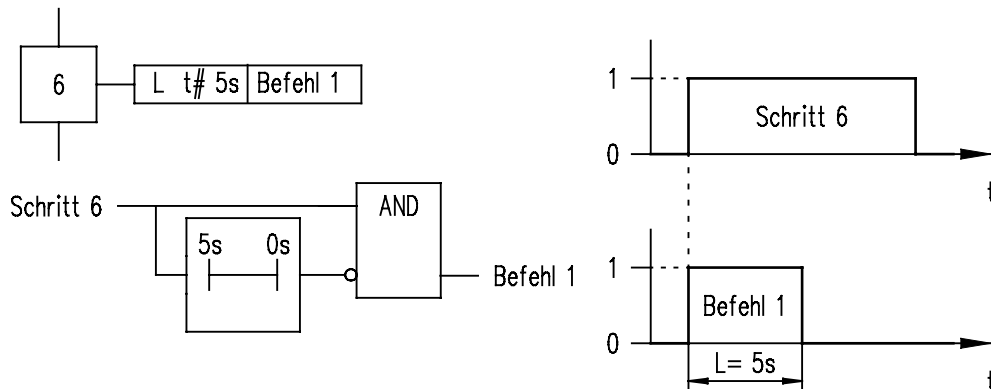


### 6.1.3 Bestimmungszeichen D





### 6.1.4 Bestimmungszeichen L



### 6.1.5 Bestimmungszeichen P

