



Technische Hochschule Georg Agricola

INDUSTRIEAUTOMATION

SPS PROGRAMMIERSPRACHEN

Bochum

Prof. Dr.-Ing. Lars N. Josler

INHALT DER VORLESUNG

- Überblick über Programmiersprachen
- Grundlegende Programmierfunktionen und Programmstrukturen der SPS
- Anweisungsliste (AWL)
- Strukturierter Text (ST)
- Kontaktplan (KOP)
- Funktionsbausteinsprache (FBS)
- Ablaufsprache (AS)





Übersicht über die **Programmiersprachen** nach IEC 61131:

Textsprachen

- | | |
|----------------------------|------------------------------|
| ■ Anweisungsliste (AWL) | <i>Instruction List (IL)</i> |
| ■ Strukturierter Text (ST) | <i>Structured Text (ST)</i> |

Grafische Sprachen

- | | |
|-------------------------------------|--|
| ■ Kontaktplan (KOP) | <i>Ladder Diagram (LD)</i> |
| ■ Funktionsbausteinsprache (FBS) | <i>Function Block Diagram (FBD)</i> |
| ■ Ablaufsprache (AS, auch textuell) | <i>Sequential function Chart (SFC)</i> |

BEZEICHNUNGEN: SIEMENS – DIN EN61131-3



Textsprachen nach DIN EN61131-3

Textsprachen nach Siemens

■ Anweisungsliste (AWL)	=
■ Strukturierter Text (ST)	Structured Control Language (SCL)
■ Kontaktplan (KOP)	=
■ Funktionsbausteinsprache (FBS)	Funktionsplan (FUP)
■ Ablaufsprache (AS)	Ablaufsteuerung (S7-Graph)

Anweisungsliste (AWL): BEISPIELE



 Instruction List (IL)

U "Taster_links"
O "Taster_rechts"
= "Leuchte"

Symbolisch

U %E0.0
O %E0.1
= %A0.0

Absolut

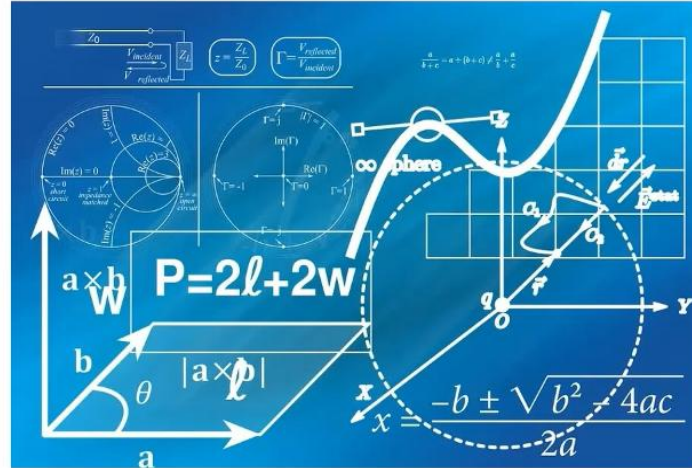
1	U	"Taster_links"	%E0.0	Taster Bedienpult links
2	O	"Taster_rechts"	%E0.1	Taster Bedienpult rechts
3	=	"Leuchte"	%A0.0	Leuchte Bedienpult

Strukturierter Text (ST): BEISPIELE



Structured Text (ST)

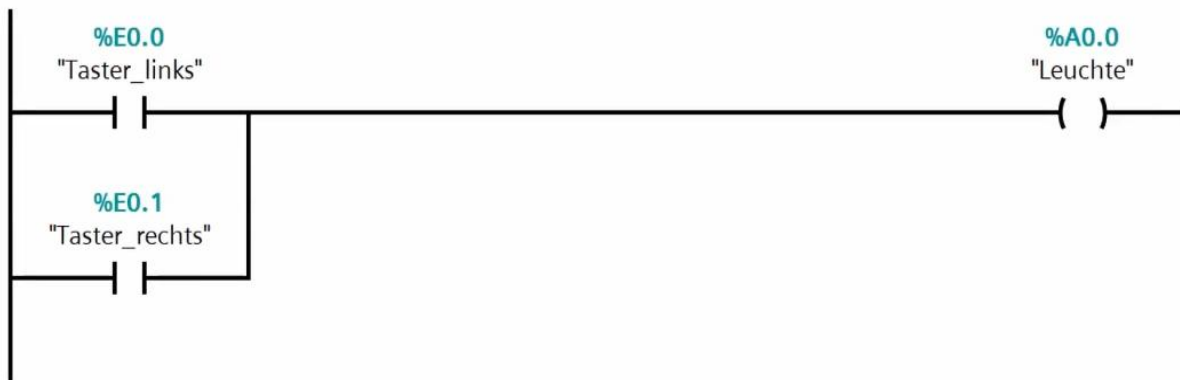
"Leuchte" := "Taster_links" OR "Taster_rechts";



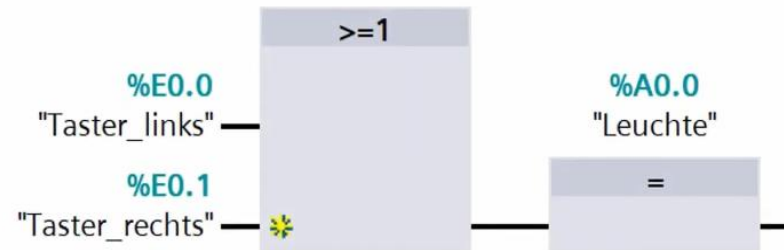
Kontaktplan (KOP): BEISPIELE



 Ladder Diagram (LD)



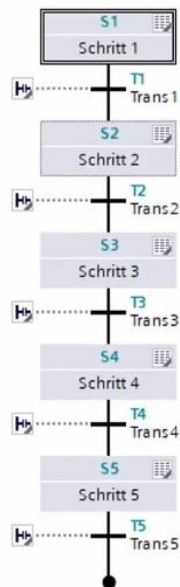
Funktionsbausteinsprache (FBS): BEISPIELE



Ablaufsprache (AS): BEISPIELE



Sequential Function Chart (SFC)





Für die Erstellung von Anwendersoftware können bei einer SPS unterschiedliche Programmiersprachen genutzt werden. Die Einigung auf diese Sprachen ist historisch begründet. So bildet der Kontaktplan die Relais-Schaltung früherer verbindungsprogrammierter Steuerungen nach. Er wird im amerikanischen Raum auch heute noch vielfach eingesetzt. In Europa wird als grafische Sprache hauptsächlich die Funktionsbausteinsprache verwendet, die an Blockschaltbilder und Logikpläne angelehnt ist.

- **Empfehlung zur Wahl einer Programmiersprache:**
 - Die unterschiedlichen Sprachen können miteinander kombiniert werden. Es können zum Beispiel in AWL geschriebene Unterprogramme in der Ablaufsprache aufgerufen werden. Die Sprache sollte daher nach der Aufgabenstellung mit dem Ziel einer bestmöglichen Übersichtlichkeit und Wartbarkeit ausgewählt werden:
 - Prozessabläufe lassen sich gut als Schrittketten beschreiben und werden deshalb in der Ablaufsprache AS programmiert,
 - Ansteuerprogramme für Aktoren und Sensoren bestehen zumeist aus Verknüpfungslogik und werden deshalb in der Funktionsbausteinsprache FBS programmiert,
 - Anwender-Funktionsbausteine und –Funktionen, die das Wartungspersonal in der Regel ohnehin als Black-Box ansieht, können effizient in Textsprachen wie AWL oder ST programmiert werden.



Grundfunktionen

- Zuweisung
- Negation (NOT, NICHT)
- UND (AND)
- ODER (OR)

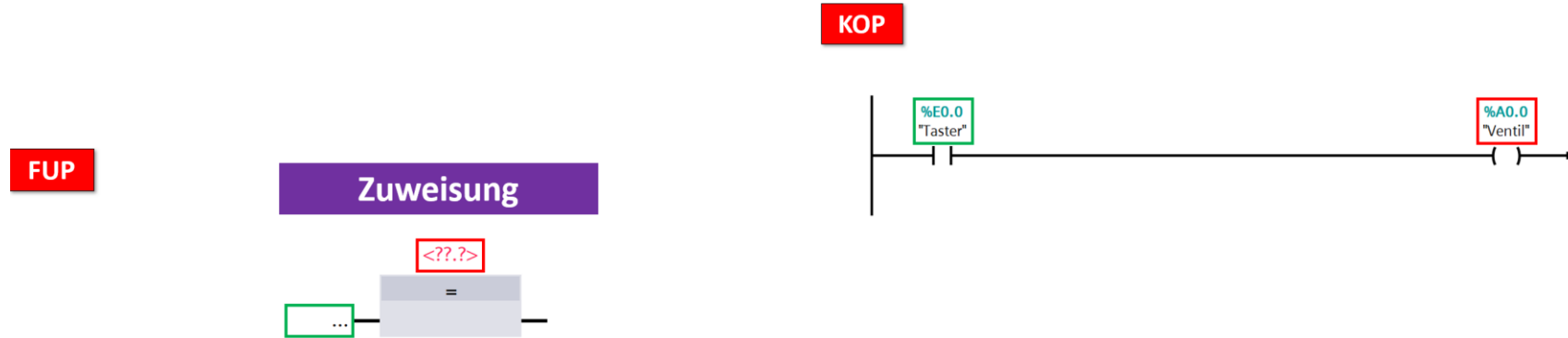
Erweiterte Funktion

- UND-NICHT (NAND)
- ODER-NICHT (NOR)
- EXKLUSIV ODER (XOR)

BITVERKNÜPFUNG



Name	Identität, Puffer	NICHT, Negation	UND, Konjunktion	NAND	ODER, Disjunktion	NOR	XOR, Antivalenz	Äquivalenz																																																																																																						
Schaltfunktion nach DIN 66000	$y = x$	$y = \bar{x},$ $y = \neg x$	$y = x_1 \wedge x_2$	$y = x_1 \bar{\wedge} x_2$	$y = x_1 \vee x_2$	$y = x_1 \bar{\vee} x_2$	$y = x_1 \leftrightarrow x_2$	$y = x_1 \leftrightarrow x_2$																																																																																																						
Wahrheitstabelle	<table><tr><td>x</td><td>y</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	y	0	0	1	1	<table><tr><td>x</td><td>y</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	y	0	1	1	0	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x ₁	x ₂	y	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x ₁	x ₂	y	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x ₁	x ₂	y	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x ₁	x ₂	y	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x ₁	x ₂	y	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><td>x₁</td><td>x₂</td><td>y</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x ₁	x ₂	y	0	0	1	0	1	0	1	0	0	1	1	1
x	y																																																																																																													
0	0																																																																																																													
1	1																																																																																																													
x	y																																																																																																													
0	1																																																																																																													
1	0																																																																																																													
x ₁	x ₂	y																																																																																																												
0	0	0																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												
x ₁	x ₂	y																																																																																																												
0	0	1																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x ₁	x ₂	y																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	1																																																																																																												
x ₁	x ₂	y																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	0																																																																																																												
x ₁	x ₂	y																																																																																																												
0	0	0																																																																																																												
0	1	1																																																																																																												
1	0	1																																																																																																												
1	1	0																																																																																																												
x ₁	x ₂	y																																																																																																												
0	0	1																																																																																																												
0	1	0																																																																																																												
1	0	0																																																																																																												
1	1	1																																																																																																												
Kurzbeschreibung			1 g. d., w. ¹ alle Eingänge 1	0 g. d., w. alle Eingänge 1	0 g. d., w. kein Eingang 1	1 g. d., w. kein Eingang 1	1 g. d., w. genau ein Eingang 1	1 g. d., w. alle Eingänge gleich																																																																																																						
Schaltsymbol nach DIN EN 60617-12																																																																																																														
Schaltsymbol nach alter DIN 40700-14																																																																																																														
Schaltsymbol nach IEEE Std 91-1984																																																																																																														
VHDL-Code	$y \leq x;$	$y \leq \text{not } x;$	$y \leq x_1 \text{ and } x_2;$	$y \leq x_1 \text{ nand } x_2;$	$y \leq x_1 \text{ or } x_2;$	$y \leq x_1 \text{ nor } x_2;$	$y \leq x_1 \text{ xor } x_2;$	$y \leq x_1 \text{ xnor } x_2;$																																																																																																						



Wenn der Operand im **Eingang** TRUE ist, dann wird der Operand im **Ausgang** angesteuert.

AWL

1	U	"Taster"	%E0.0	Taster Bedienpult
2	=	"Ventil"	%A0.0	Ventil Zulauf öffnen

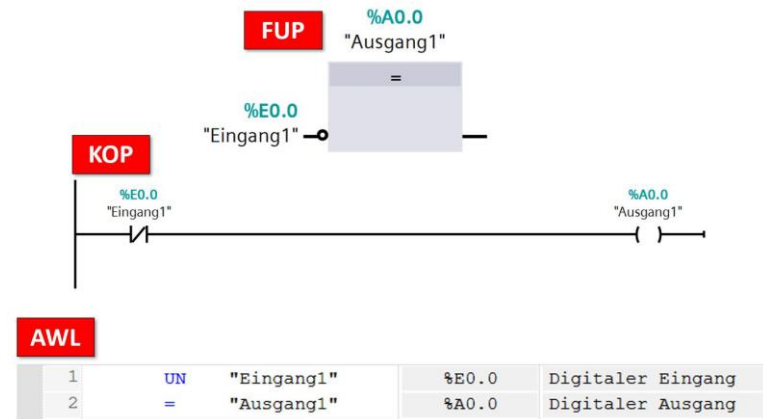
Was ist „Negieren“?

Diagram of a 2-bit adder circuit. The inputs are "Eingang1" (FALSE) and "Eingang2" (TRUE). The output is "Ausgang1" (2). The adder block is labeled "2-Bit Addierer".

E0.0	A0.0
0	0
1	1

Diagram illustrating a NOT gate. The input "Eingang1" is connected to the input of a NOT gate block. The output of the NOT gate is connected to the input of an AND gate block. The output of the AND gate is connected to the input of an OR gate block. The output of the OR gate is connected to the input of a final AND gate block. The output of the final AND gate is connected to the output of the circuit, labeled "Ausgang1".

E0.0	A0.0
0	1
1	0



UND-VERKNÜPFUNG

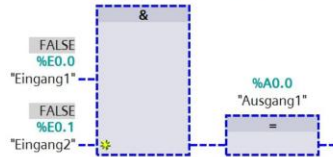


FUP

UND-Verknüpfung

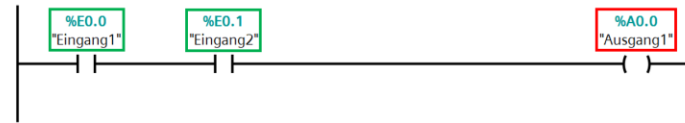


Wenn beide Operanden im **Eingang** TRUE sind, dann wird der Operand im **Ausgang** angesteuert.



E0.0	E0.1	A0.0
0	0	0
0	1	0
1	0	0
1	1	1

KOP



AWL

1	U	"Eingang1"	%E0.0	Digitaler Eingang
2	U	"Eingang2"	%E0.1	Digitaler Eingang
3	=	"Ausgang1"	%A0.0	Digitaler Ausgang

ODER-VERKNÜPFUNG

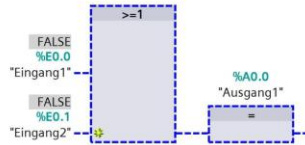


FUP

ODER-Verknüpfung



Wenn einer der Operanden im **Eingang** TRUE ist, dann wird der Operand im **Ausgang** angesteuert.



E0.0	E0.1	A0.0
0	0	0
0	1	1
1	0	1
1	1	1

KOP



AWL

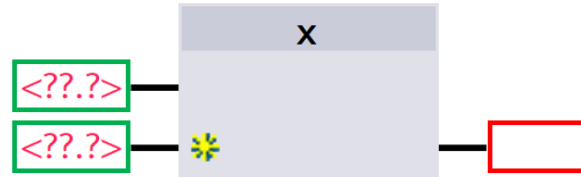
1	O	"Eingang1"	%E0.0	Digitaler Eingang
2	O	"Eingang2"	%E0.1	Digitaler Eingang
3	=	"Ausgang1"	%A0.0	Digitaler Ausgang

EXKLUSIV-ODER VERKNÜPFUNG

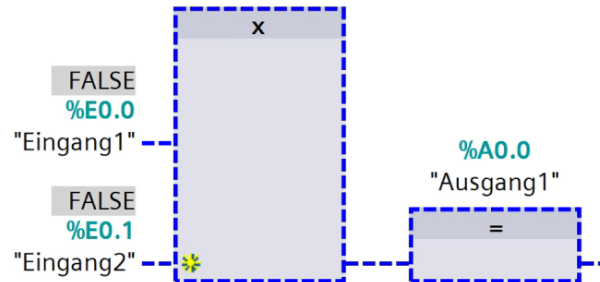


FUP

EXKLUSIV-ODER-Verknüpfung



Wenn nur einer der Operanden im **Eingang** TRUE ist, dann wird der Operand im **Ausgang** angesteuert.

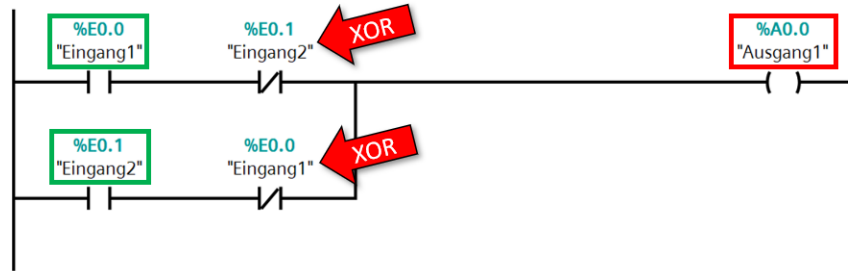


E0.0	E0.1	A0.0
0	0	0
0	1	1
1	0	1
1	1	0

EXKLUSIV-ODER VERKNÜPFUNG



KOP



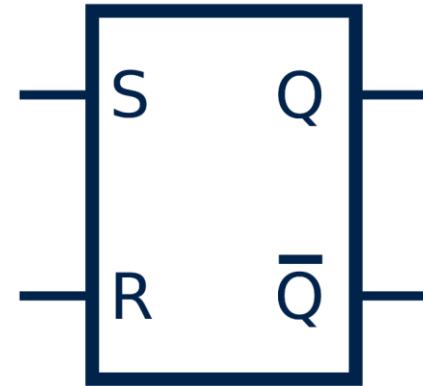
AWL

1	X	"Eingang1"	%E0.0	Digitaler Eingang
2	X	"Eingang2"	%E0.1	Digitaler Eingang
3	=	"Ausgang1"	%A0.0	Digitaler Ausgang



RS-Speicher/RS-Flipflop speichert (Setzen eines Signals) ein kurzzeitiges Signal, bis durch ein erneutes Signal dieser anfangs gespeicherte Signalzustand wieder gelöscht wird (Rücksetzen eines Signals).

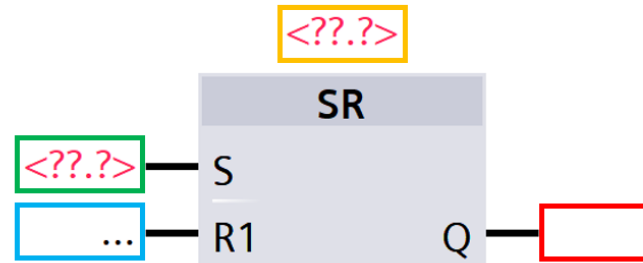
S	R	Q	$\bar{Q}$
0	0	kein Signalwechsel	
0	1	0	1
1	0	1	0
1	1	SR: Rücksetzen hat Vorrang $\rightarrow Q = 0$. RS: Setzen hat Vorrang $\rightarrow Q = 1$.	





FUP

SR-Flipflop



Wenn der Signalzustand am **Eingang „S“** TRUE ist, dann wird der **angegebenen Operand** gesetzt.

Wenn der Signalzustand am **Eingang „R1“** TRUE ist, dann wird der **angegebenen Operand** zurückgesetzt.

Der aktuelle Signalzustand des **angegebenen Operanden** wird auf den **Ausgang „Q“** übertragen.

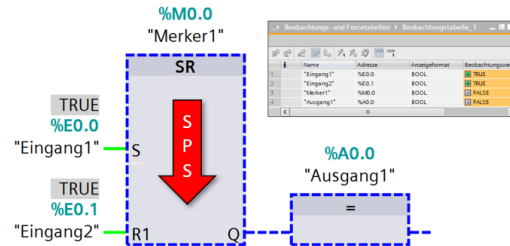
SETZEN/RÜCKSETZEN



Eine SPS arbeitet das Programm **von oben nach unten ab**.

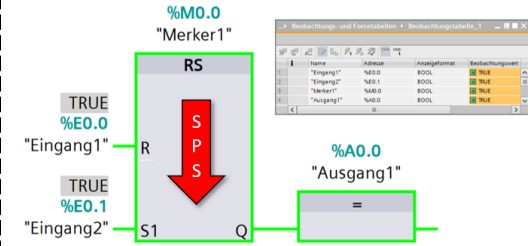
Sind die **Eingänge „S“** und **„R“** gleichzeitig aktiv, ist immer der untere Befehl dominant.

SR



Rücksetzdominant

RS



Setzdominant

SETZEN/RÜCKSETZEN



Siemens - C:\Users\sp4\Desktop\testprogramm\edit\verknuepfungen\@verknuepfungen

Totally Integrated Automation PORTAL

Projekt: Bearbeiten Ansicht Einfügen Online Editor Markierungen Fenster Hilfe

Projekt speichern Online verbunden Online-Verbindung trennen Projekt durchsuchen

Bitverknüpfungen SPS [CPU 1511-1 PN] Programmbausteine Main [OB1]

Geräte

- Bitverknüpfungen
- Neues Gerät hinzufügen
- Geräte & Netze
- SPS [CPU 1511-1 PN]
- Gerätekonfiguration
- Online & Diagnose
- Programmbausteine
- Neuen Baustein hinzufügen
- Main [OB1]
- Sonstiges
- Technologieobjekte
- Externe Quellen
- PLC-Variablen
- Alle Variablen anzeigen
- Neue Variablen-tabelle hinzufügen
- Standard-Variablen-tabelle [50]
- Unsere Variablen [4]
- PLC-Diagnostik
- Beobachtungs- und Forcistabellen
- Online-Sicherungen
- Traces
- Geräte-Prop-Daten
- Programminformationen
- PLC-Überwachungen & Meldung...
- PLC-Meldedaten
- Online-Kartendaten
- Lokale Module
- Nicht gruppierte Geräte
- Security-Einstellungen
- Geräteübergreifende Funktionen
- Gemeinsame Daten
- Dokumentationseinstellungen
- Sprachen & Ressourcen
- Version Control Interface
- Online-Zugänge
- Card Reader/USB-Speicher

Netzwerk 1: "Main Program Sweep (Cycle)"

Netzwerk 2: "Merker 'Band vorwärts' setzen"

Netzwerk 3: "Ausgang Fördermotor ansteuern"

Erreiter 1 (Erst)

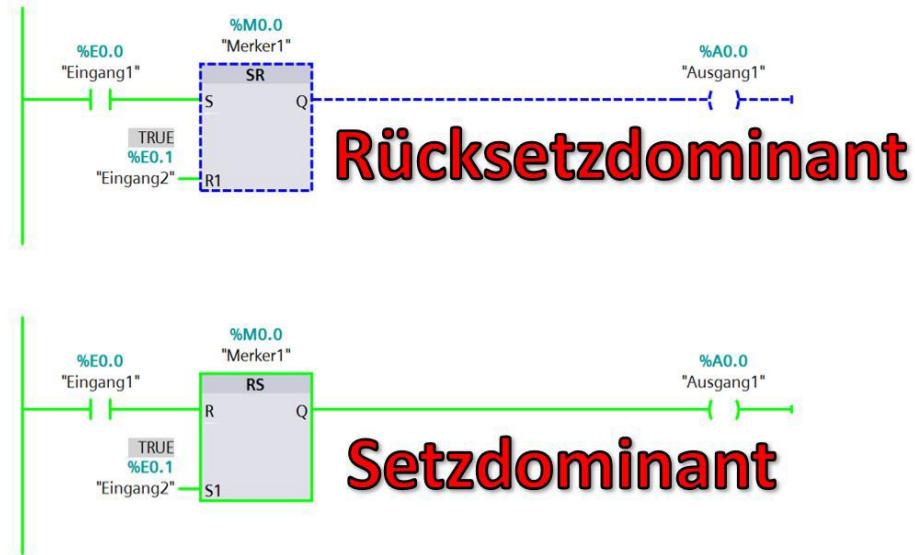
Erreiter 1 (Erst)	ERZWUNGEN	X
Sensor_Auslauf	X	X
Sensor_Einlauf	X	X

FACTORY I/O v2.4.3 - Ultimate Edition - SR

Siemens 57-PLCSM



KOP



SETZEN/RÜCKSETZEN



AWL

			Adresse	VKE	Wert	Extra	Variableninformation
1	U	"Eingang1"	%E0.0	0	0		Digitaler Eingang
2	S	"Merker1"	%M0.0	0	0		Bitmerker
3	U	"Eingang2"	%E0.1	0	0		Digitaler Eingang
4	R	"Merker1"	%M0.0	0	0		Bitmerker
5							
6	U	"Merker1"	%M0.0	0	0		Bitmerker
7	=	"Ausgang1"	%A0.0	0	0		Digitaler Ausgang

Rücksetzdominant

1	U	"Eingang1"	%E0.0
2	S	"Merker1"	%M0.0
3	U	"Eingang2"	%E0.1
4	R	"Merker1"	%M0.0
5			
6	U	"Merker1"	%M0.0
7	=	"Ausgang1"	%A0.0



Setzdominant

1	U	"Eingang1"	%E0.0
2	R	"Merker1"	%M0.0
3	U	"Eingang2"	%E0.1
4	S	"Merker1"	%M0.0
5			
6	U	"Merker1"	%M0.0
7	=	"Ausgang1"	%A0.0

IEC-ZEITEN: TON



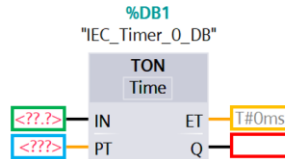
In TIA-Portal unter Anweisungen:

- Zeiten

TON:

- Einschaltverzögerung

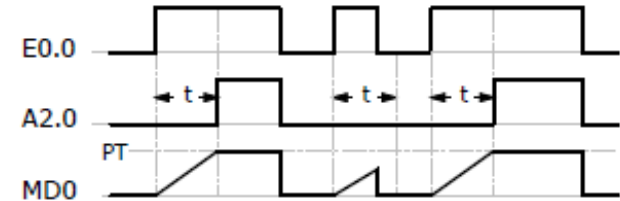
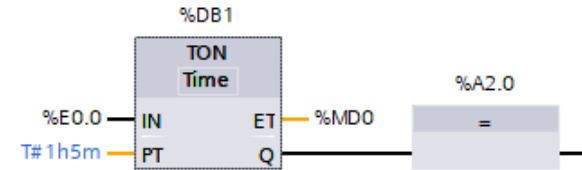
IEC-Zeit: TON



Mit der IEC-Zeit „TON“ wird das Einschalten des **Ausgang „Q“** um die Zeit am **Eingang „PT“** verzögert.

Die Zeit wird gestartet, wenn am **Eingang „IN“** ein 1-Signal anliegt.

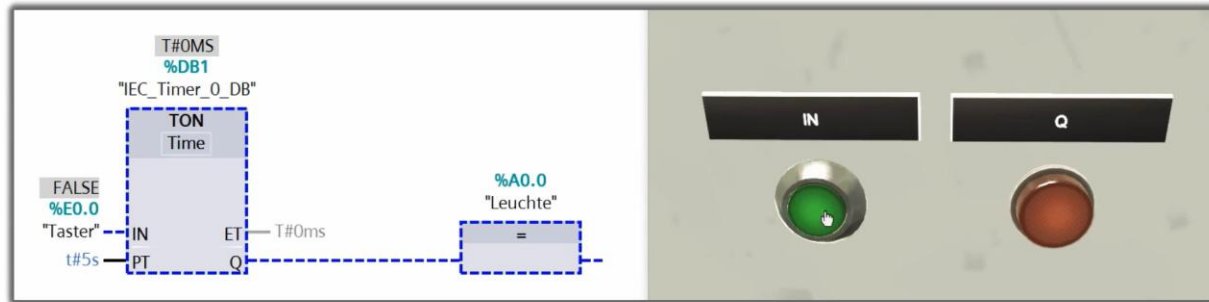
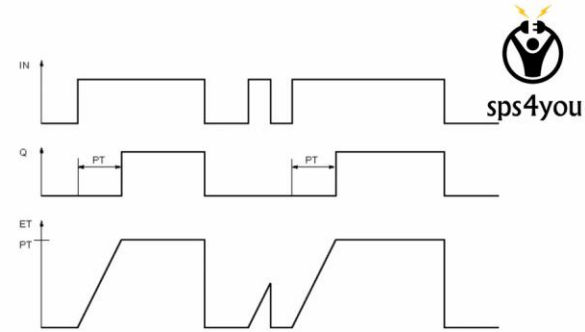
Über den **Ausgang „ET“** kann der aktuelle Zeitwert abgefragt werden. (optional)



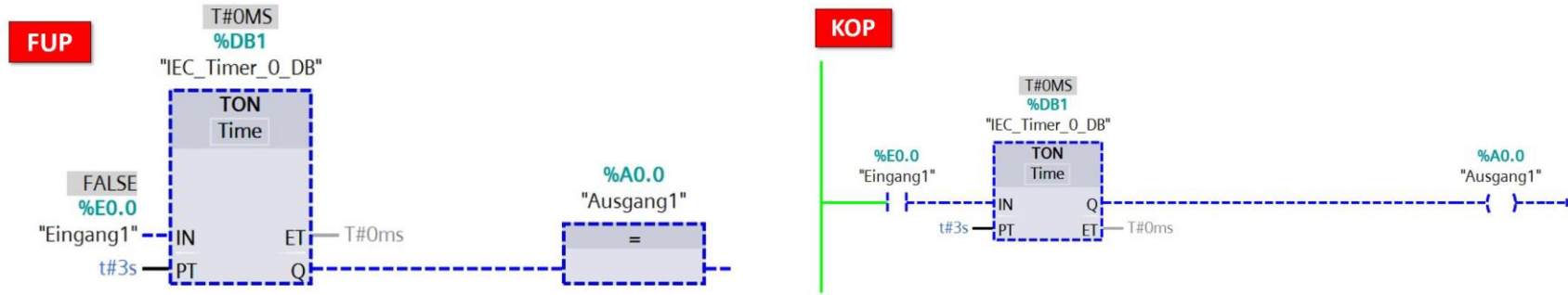
IE-ZEITEN: TON



Parameter	Deklaration	Datentyp		Speicherbereich		Beschreibung
		S7-1200	S7-1500	S7-1200	S7-1500	
IN	Input	BOOL	BOOL	E, A, M, D, L oder Konstante	E, A, M, D, L, T, Z, P oder Konstante	Starteingang
PT	Input	TIME	TIME, LTIME	E, A, M, D, L oder Konstante	E, A, M, D, L, P oder Konstante	Zeitdauer der Einschaltverzögerung. Der Wert am Parameter PT muss positiv sein.
Q	Output	BOOL	BOOL	E, A, M, D, L	E, A, M, D, L, P	Ausgang, der nach dem Ablauf der Zeit PT gesetzt wird.
ET	Output	TIME	TIME, LTIME	E, A, M, D, L	E, A, M, D, L, P	Aktueller Zeitwert



IEC-ZEITEN: TON



AWL

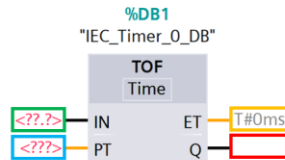
		Adresse	VKE	Wert	Ex..
1	CALL TON , "IEC_Timer_0_DB"	%DB1	In	Out	
2	Time				
3	IN := "Eingang1"	%E0.0	0		
4	PT := t#3s	t#3s	T#3S		
5	Q := "Ausgang1"	%A0.0		0	
6	ET :=				
7					



TIA-Portal: Anweisungen → Zeiten

TOF → Ausschaltverzögerung (zum Nachlaufen der Aktoren)

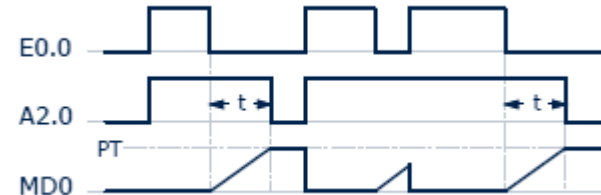
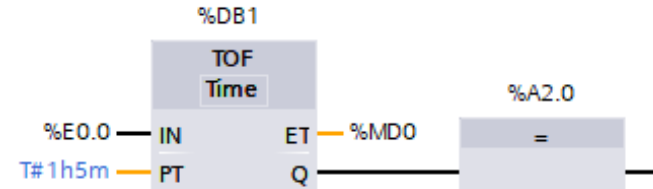
IEC-Zeit: TOF



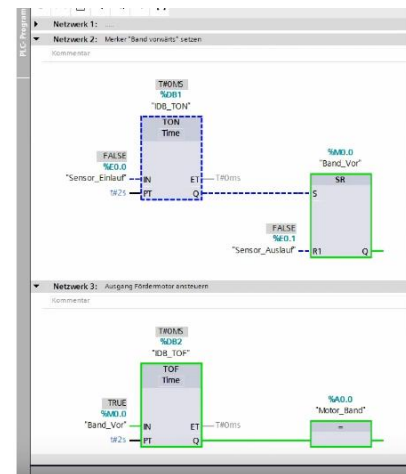
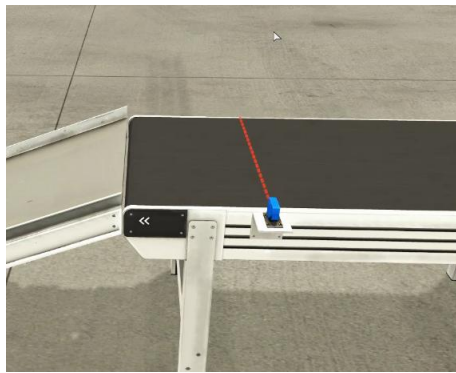
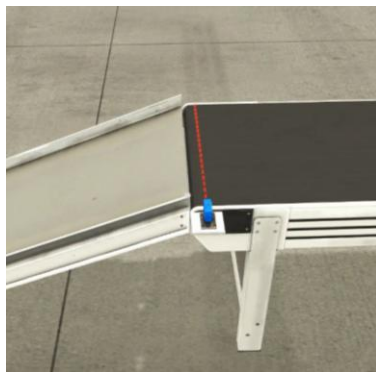
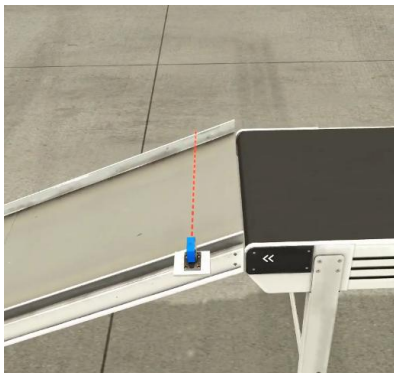
Mit der IEC-Zeit „TOF“ wird das Ausschalten des **Ausgang „Q“** um die Zeit am **Eingang „PT“** verzögert.

Die Zeit wird gestartet, wenn am **Eingang „IN“** eine negative Flanke erkannt wird.

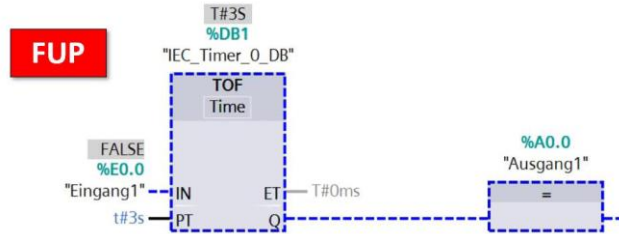
Über den **Ausgang „ET“** kann der aktuelle Zeitwert abgefragt werden. (optional)



IEC-ZEITEN:TOF

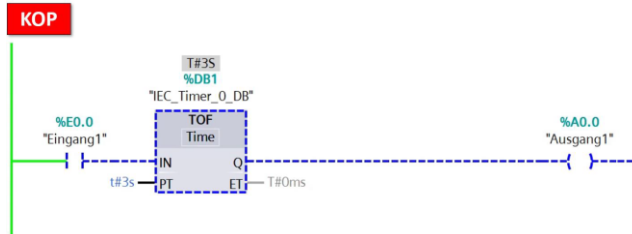


IEC-ZEITEN: TOF



AWL

		Adresse	VKE	Wert	Extra
1	CALL TOF , "IEC_Timer_0_DB"	%DB1	In	Out	
2	Time				
3	IN :="Eingang1"	%E0.0	0		
4	PT :=t#3s	t#3s	T#3S		
5	Q :="Ausgang1"	%A0.0		0	
6	ET :="Aktuelle_Zeitwert"	%MD10		T#3S	



TIA-Infosystem

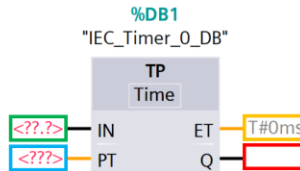
ET	Output	TIME
Länge (Bit)	Format	Wertebereich
32	Zeitdauer mit Vorzeichen	T#-24d_20h_31m_23s_648ms bis T#+24d_20h_31m_23s_647ms



TIA-Portal: Anweisungen → Zeiten

TP → Impulserzeugen

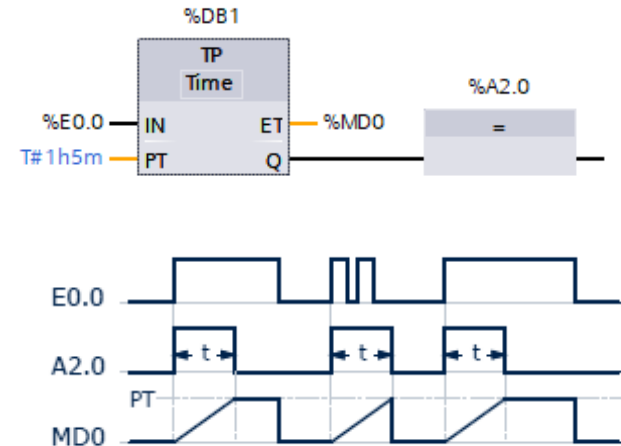
IEC-Zeit: TP



Mit der IEC-Zeit „TP“ wird der **Ausgang „Q“** für die Zeitdauer am **Eingang „PT“** angesteuert.

Die Zeit wird gestartet, wenn am **Eingang „IN“** eine positive Flanke erkannt wird.

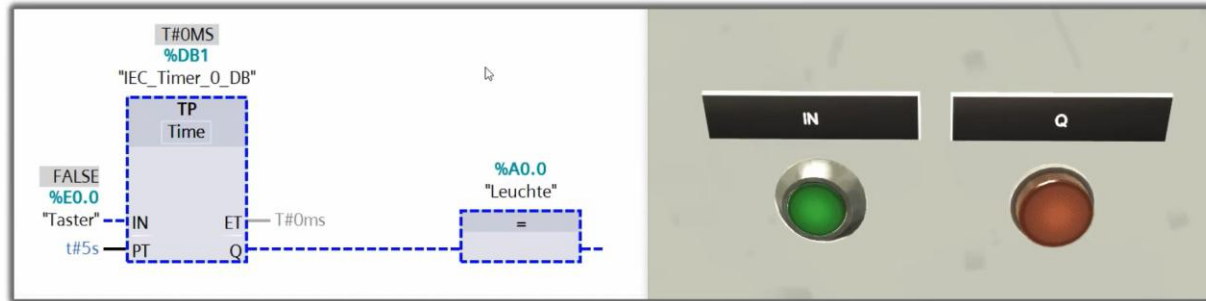
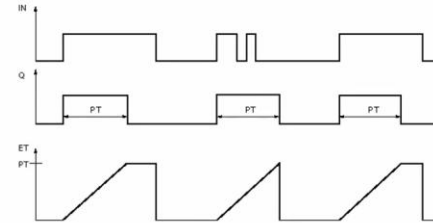
Über den **Ausgang „ET“** kann der aktuelle Zeitwert abgefragt werden. (optional)



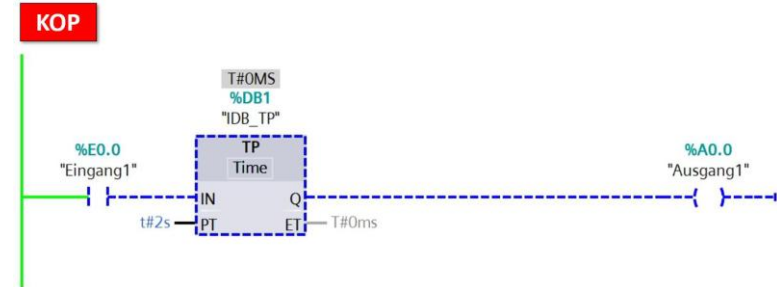
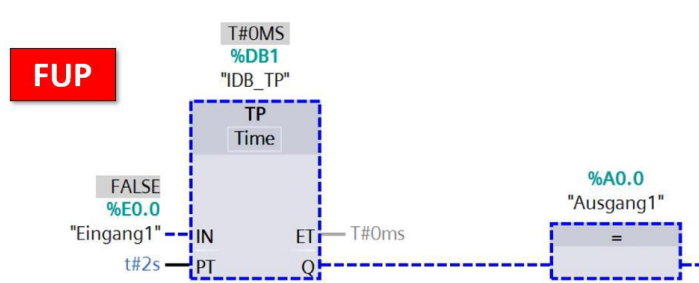
IEC-ZEITEN: TP



Parameter	Deklaration	Datentyp		Speicherbereich		Beschreibung
		S7-1200	S7-1500	S7-1200	S7-1500	
IN	Input	BOOL	BOOL	E, A, M, D, L oder Konstante	E, A, M, D, L, T, Z, P oder Konstante	Starteingang
PT	Input	TIME	TIME, LTIME	E, A, M, D, L oder Konstante	E, A, M, D, L, P oder Konstante	Zeitdauer des Impulses. Der Wert am Parameter PT muss positiv sein.
Q	Output	BOOL	BOOL	E, A, M, D, L	E, A, M, D, L, P	Impulsausgang
ET	Output	TIME	TIME, LTIME	E, A, M, D, L	E, A, M, D, L, P	Aktueller Zeitwert



IEC-ZEITEN: TP



AWL

		Adresse	VKE	Wert	Extra
1	CALL TP , "IDB_TP"	%DB1	In	Out	
2	Time				
3	IN :="Eingang1"	%E0.0	0		
4	PT :="T#2s"	T#2s	T#2S		
5	Q :="Ausgang1"	%A0.0		0	
6	ET :="Aktuelle_Zeitwert"	%MD10		T#0MS	



- **Sprache nach DIN EN 61131-3**
 - Optimierte für die SPS-Programmierung
 - Enthält Sprachelemente aus PASCAL
 - Ergänzt um typische SPS-Elemente, wie Ein-/Ausgänge, Zeitglieder und Zähler
- **Verwendung:**
 - Programmierung komplexer Algorithmen
 - Programmierung mathematischer Funktionen
 - Prozessoptimierung



- **AWL: Textsprache und der Assemblerprogrammierung ist sehr ähnlich**
- **Vorteile**
 - ähnlich Assembler, traditionelle (historische) SPS-Sprache (S7-1500 emuliert nur)
- **Nachteil**
 - unübersichtlich, fehleranfällig, umständlich, wird bald aussterben
- **Einsatz**
 - Anwender-Funktionsbausteine, Anwender-Funktionen

```
U      "Taster_links"  
O      "Taster_rechts"  
=      "Leuchte"
```

Symbolisch

```
U      %E0.0  
O      %E0.1  
=      %A0.0
```

Absolut

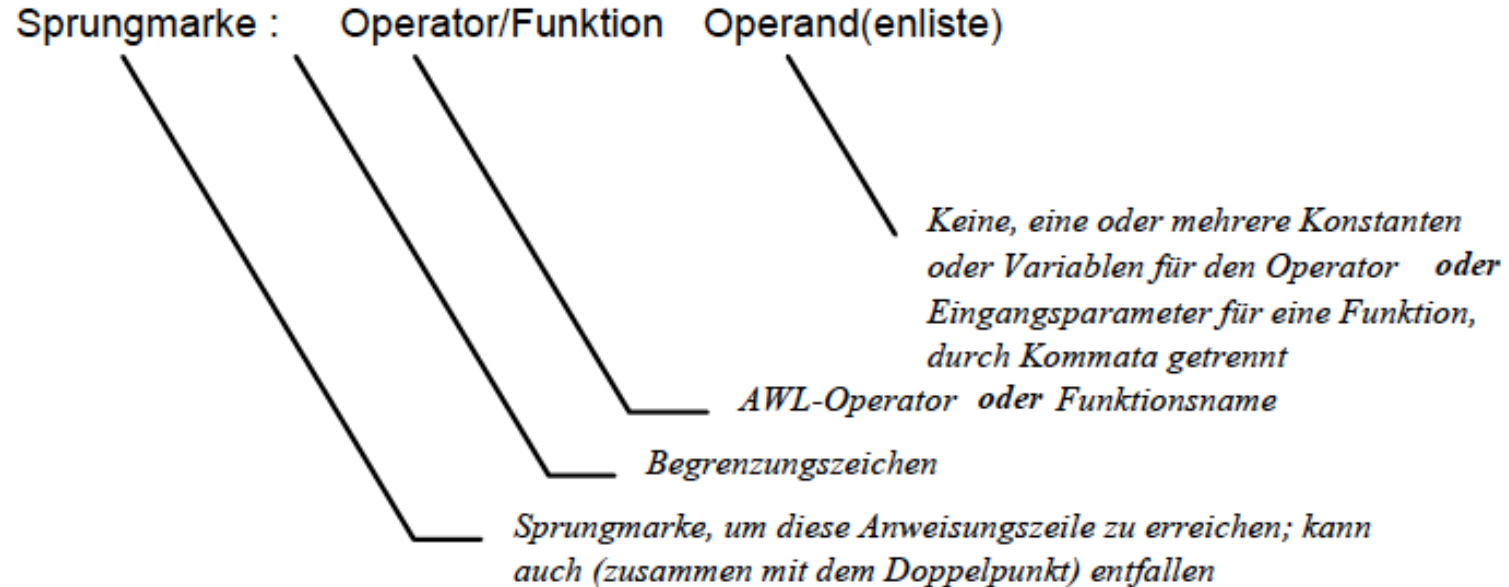


- Die Anweisungsliste, meist kurz AWL genannt, ist eine der in IEC 61131-3 standardisierten Methoden zur Programmierung von Speicherprogrammierbaren Steuerungen (SPS). Viele SPS-Hersteller bezeichnen die von ihnen verwendete Sprache auch dann als AWL, wenn sie sich nicht exakt an die IEC 61131-3 hält, sodass sich bestehende AWL-Programme kaum auf Steuerungen anderer Hersteller übertragen lassen.
- AWL dient hauptsächlich zur logischen Verknüpfung von Steuerungseingängen und -ausgängen. Typischerweise wird ein (digitaler) Eingang in das Arbeitsregister (auch Akkumulator genannt) geladen ("LD %IX0.1") mit anderen Eingängen, Konstanten oder Speicherwerten verknüpft („AND %IX0.2“) und auf einen Ausgang geschrieben ("ST %QX0.2").
- Die Hauptmerkmale von AWL sind, dass Operatoren nur einen Operanden besitzen
- Die Syntax der Sprache ist an die Assemblersprache angelehnt:
 - Umständliche und fehleranfällige Strukturierungsmöglichkeiten durch Sprungbefehle
 - Kleiner Programmcode, daher häufig auf älteren Steuerungen verbreitet
- AWL-Programme sind aber im Vergleich zu Programmen in höheren Sprachen - insbesondere bei größeren Projekten - sehr unübersichtlich und schlecht wartbar. Heute werden daher für die Programmierung von Steuerungen normalerweise die höheren Sprachen der IEC 61131-3, insbesondere strukturierter Text (ST) oder Ablaufsprache, Continuous Function Chart (Signalflussplan) oder vielfach auch "C" eingesetzt.

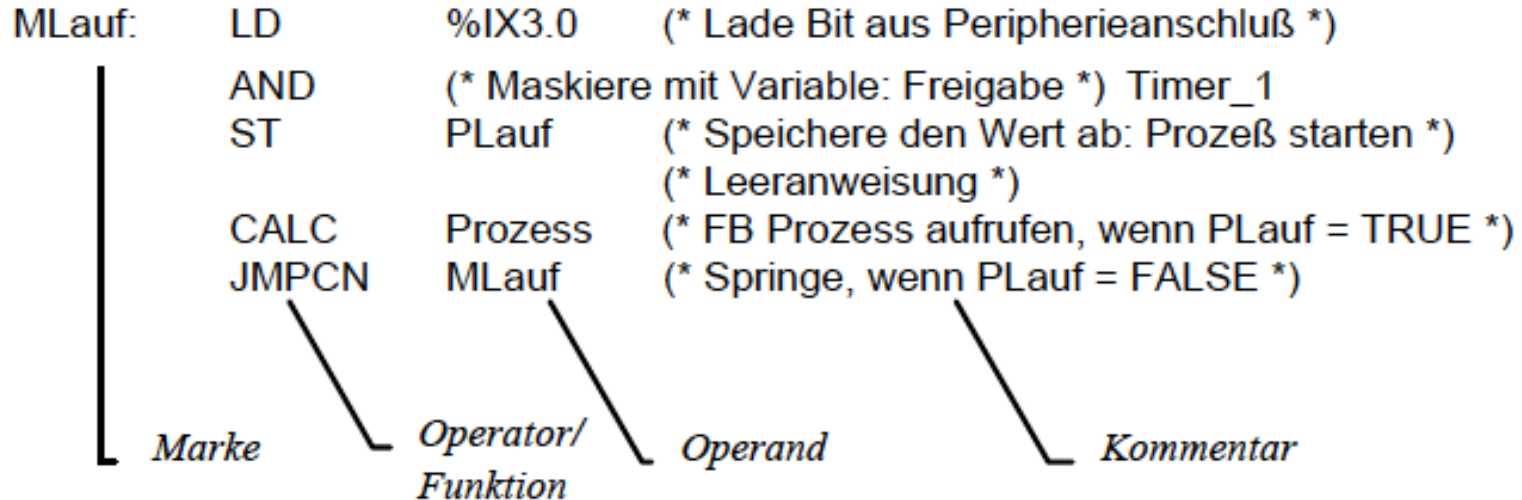
ANWEISUNGSLISTE (AWL)



AWL ist eine zeilenorientierte Sprache



ANWEISUNGSLISTE (AWL)



Anmerkung:
Übersicht

Der Kommentar kann an jeder Stelle platziert werden, jedoch empfiehlt es sich der
halber die Platzierung am Ende der Anweisungszeile

UNIVERSELLE AKKUMULATOR



- Enthält das aktuelle Ergebnis (AE) einer Operation
- Logischer Akkumulator ohne fest definierte Hardware-Registerlänge
- Abstraktes Akkumulatormodell mit Akkumulator-Keller (Stack)
- Keine speziellen Prozessor-Statusbits erforderlich
- Vergleichsergebnisse werden als boolescher Wert TRUE (1) oder FALSE (0) im AE abgelegt
- Unterstützt unterschiedliche Datentypen (z. B. BOOL, INT, REAL)
- Verarbeitung abgeleiteter Datentypen abhängig von Operator und Implementierung



```
VAR
  ErstOperand, ZweitOperand, Ergebnis: INT := 10;
  StringOp: String[30] := '12345678901234567890';
  StringErg: String[25];
END_VAR

...
B1: LD      ErstOperand    (* 10 {INT} *)
     ADD     ZweitOperand  (* 20 {INT} *)
     ST      Ergebnis      (* 20 {INT} *)
     GT      0             (* TRUE, da 20 > 0 {BOOL} *)
     JMPC    B2            (* springe, da AE=TRUE; alter Wert AE bleibt {BOOL} *)
     JMP     WeitWeg       (* AE undefiniert o. alter Wert AE bleibt; impl.abhängig *)
B2: LD      StringOp       (* 12345678901234567890 {STRING} *)
     ST      StringErg     (* 12345678901234567890 {STRING} *)
```



Modifizierer

- N Negierung des Operanden
- (Schachtelungsebenen durch Klammerung
- C Bedingte Ausführung des Operators (falls AE=TRUE)

Beispiel Negierung des Operanden

```
VAR    Var1: BOOL := FALSE;    END_VAR
LDN    FALSE (* TRUE {BOOL} ; entspricht LD 1 *)
ANDN   Var1    (* TRUE {BOOL} *)
```



Beispiel Schachtelung

[	LD	Var1	(* Lade Wert von Var1 *)
	AND(	Var2	(* Berechne den gesamten Klammerausdruck und *)
	OR	Var3	(* verknüpfe ihn mit dem Aktuellen Ergebnis (Var1) *)
	)		
	ST	Var4	(* Speichere Gesamtverknüpfung *)

$\text{Var4} := \text{Var1} \cdot (\text{Var2} + \text{Var3})$ $\text{Var4} := \text{Var1} \wedge (\text{Var2} \vee \text{Var3})$

LD	1	(* 1 *)		
ADD(	2	(* 2 *)		
	ADD(	3	(* 3 *)	
		ADD	4	(* 7 *)
	)			(* 9 *)
)				(* 10 *)
ST	Var1			(* 10 *)

Laden der Werte

Addieren von innen nach Außen



Bedingte Ausführungen

LD ErstOperand
GT 20
JMPC B2
JMP WeitWeg

(* Erzeuge AE = TRUE bei AE > 20, sonst FALSE *)

(* Springe nach B2, wenn AE TRUE, sonst nächster Befehl *)

(* Springe nach WeitWeg, unabhängig vom aktuellen AE *)

Operatoren	Operatorgruppe ^a	Erklärung
LD LDN	E	Lädt Operand (negiert) in AE
AND ANDN AND(ANDN(	W	UND-Verknüpfung des (negierten) Operanden mit AE; Ersatzdarstellung „&“ für AND ist statthaft.
OR ORN OR (ORN (	W	ODER-Verknüpfung des (negierten) Operanden mit AE
XOR XORN XOR(XORN(	W	Exklusiv-ODER-Verknüpfung des (negierten) Operanden mit AE
NOT	W	Bitweise boolesche Negierung (1er Komplement)
ST STN	U	Speichert AE in den Operand
S	U	Setzt Operand auf TRUE, wenn AE=1
R	U	Setzt Operand auf FALSE zurück, wenn AE=1
)	W	Schließen der Klammerenebene

^a Erklärung siehe Tab. 4.1.

OPERATIONEN FÜR SPRÜNGE UND AUFRUFE



Operatoren	Operatorgruppe ^a	Erklärung
JMP JMPC JMPCN	- oder U ^b U	(Un-) bedingter Sprung zu einer Sprungmarke
CAL CALC CALCN	- U ^c	(Un-) bedingter Aufruf eines Funktionsbausteins
RET RETC RETCN	- oder U ^d U	(Un-) bedingter Rücksprung aus Funktion oder Funktionsbaustein
Fun-Name	W ^c	Aufruf einer Funktion

a Erklärung siehe Tab. 4.1.

c gilt für folgenden Operator (beim Funktions- oder FB-Einsprung aber undefiniert!)

b Implementierungsabhängig

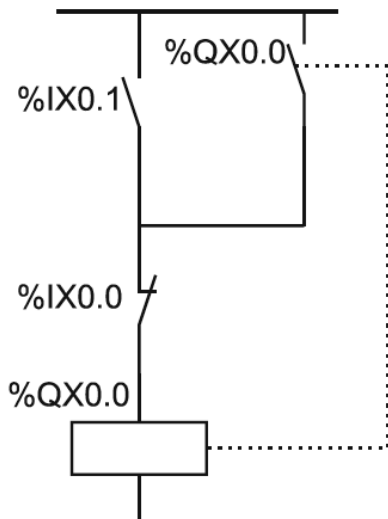
d bei Funktionen: Funktionswert ("U")

Tab. 4.5. Operatoren für Sprünge und Aufrufe

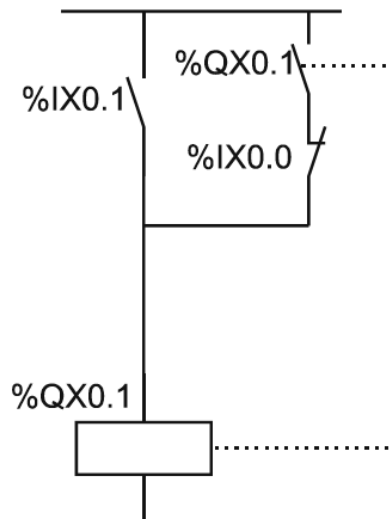
ANWEISUNGSLISTE (AWL) – BEISPIEL



Stromlaufplan A



Stromlaufplan B



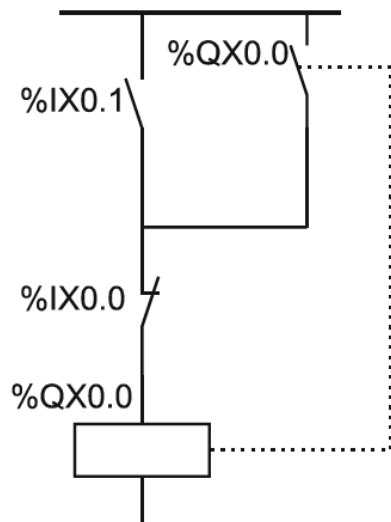
Program POWER61

ANWEISUNGSLISTE (AWL) – BEISPIEL



Start- und Stopp eines Motors

Stromlaufplan A



LD a

S x

LD b

R x

LD b

R x

LD a

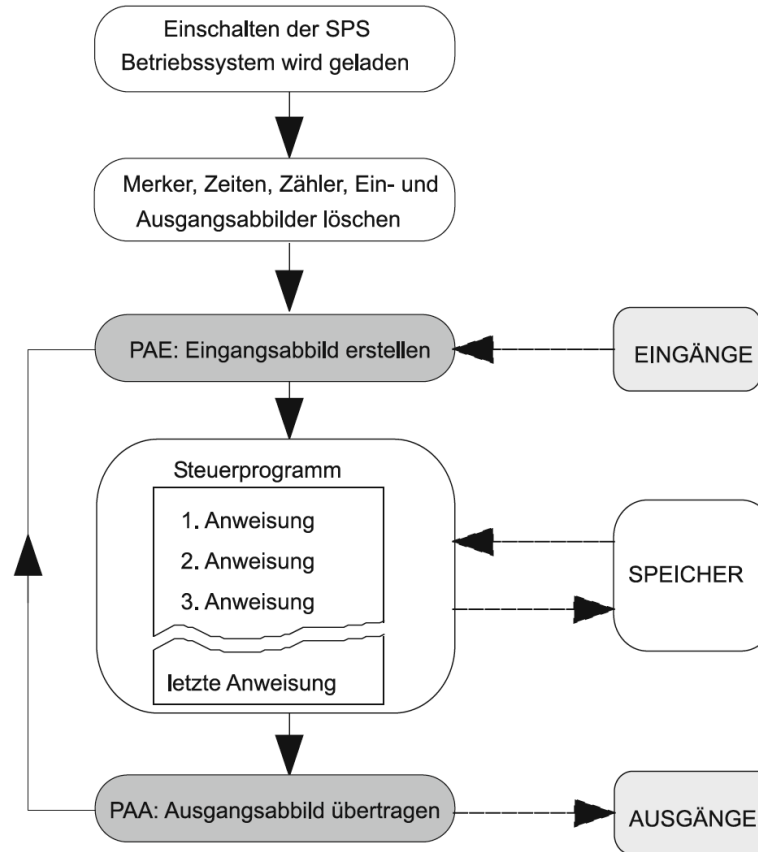
S x

```
var
  a AT %IX0.0: bool
  b AT %IX0.1: bool
  x AT %QX0.0: bool
  y AT %QX0.1: bool
end_var
```

Wo liegt der

Unterschied?

ANWEISUNGSLISTE (AWL) – BEISPIEL





Prozessabbild der Eingänge und Ausgänge

Vor jedem Zyklus werden die Werte der Eingänge in das Prozesseingangsregister gelesen. (Prozess-Abbild-Eingänge PAE). Während des Zyklus, wird immer nur auf das PAE zugegriffen.

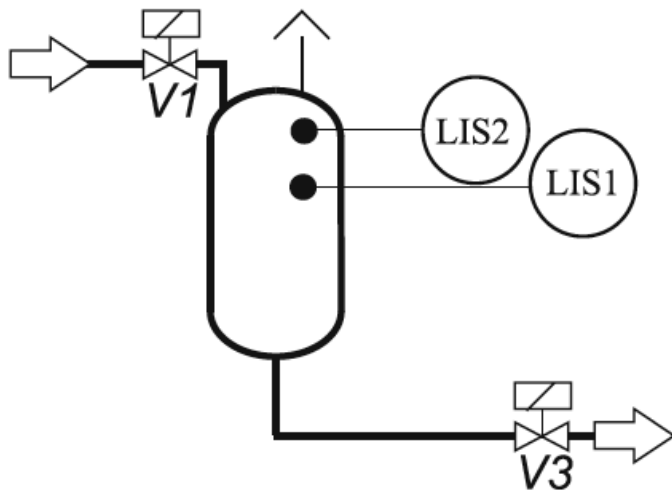
Am Ende des Zyklus werden die Ergebnisse in das Prozessausgangsregister (Prozess-Abbild-Ausgänge PAA) geschrieben.

Dies verhindert ein Flattern der Ausgänge

ANWEISUNGSLISTE (AWL) – BEISPIEL



Füllstandsüberwachung



Zuordnungsliste:

LIS1	Unterer Grenzwertgeber	%IX0.1
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
V3	Auslassventil	%QX0.3
Sw1	Entleerschalter	%IX0.7

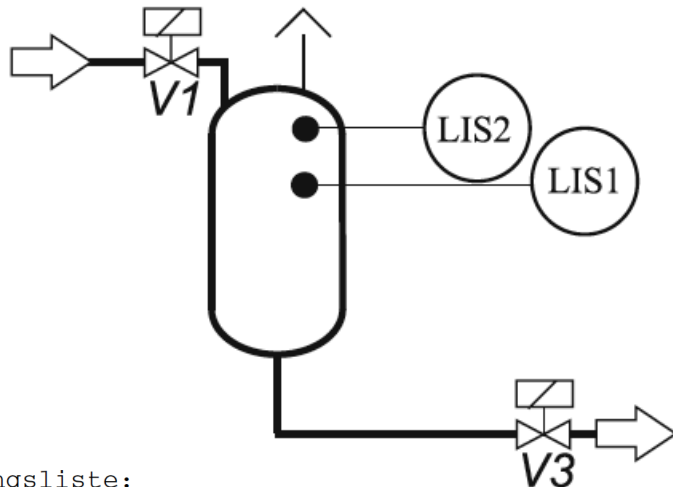
Programm schreiben:

- Erkennen wann Tank leer,
- dann füllen bis Tank voll

ANWEISUNGSLISTE (AWL) – BEISPIEL



Füllstandsüberwachung



Zuordnungsliste:

LIS1	Unterer Grenzwertgeber	%IX0.1
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
V3	Auslassventil	%QX0.3
Sw1	Entleerschalter	%IX0.7

Program Tank61

```
(*Processes: *)  
(* Levelling *)
```


Simulator: [ChatGPT Prompt Tank61](#)

FUNKTIONSBAUSTEIN (FB)



- Werden von Programmen und FBs aufgerufen oder können selbst Funktionen und FBs aufrufen
- Die Anwendung von Funktionsbausteinen geschieht in vier Schritten:

Baustein definieren

instanziieren

Baustein verfügbar machen

Parameter übergeben

parametrieren

Baustein mit Werten versorgen, Baustein liest ein

Baustein ausführen lassen

aufrufen

Baustein arbeitet

Ausgangswerte auslesen

Ergebnisse

Baustein gibt Werte aus, Werte können verwendet werden



- Funktionsbausteine müssen vor ihrer Verwendung instanziiert werden. Dazu wird im Deklarationsteil ein Instanzname mit dem Funktionsbausteintyp verknüpft.
 - FlipFlop: frei wählbarer Instanzname
 - SR: Funktionsbausteintyp (setzdominanter Speicher)
 - Bei der Instanziierung wird Speicher für interne Zustände und Variablen der Bausteininstanz bereitgestellt.
 - Ein Funktionsbaustein besitzt Ein- und Ausgänge sowie eine definierte Funktionalität. Er wird über seine Instanz aufgerufen, beispielsweise mit CAL FlipFlop in AWL.
 - Unterschied zwischen SR und RS

Funktionsbaustein	Priorität	Verhalten bei gleichzeitigem Setzen und Rücksetzen
SR (Set-Reset)	Setzdominant	Ausgang bleibt gesetzt (TRUE)
RS (Reset-Set)	Rücksetzdominant	Ausgang wird zurückgesetzt (FALSE)

```
PROGRAM Pulser
VAR
    FlipFlop: SR
END_VAR
```



- Vor der Ausführung eines Funktionsbausteins müssen dessen Eingängen Werte zugewiesen werden. Dies wird als Parametrierung bezeichnet.
- Dabei werden die Aktualparameter (konkrete Werte oder Variablen) den Formalparametern (definierte Eingänge des Funktionsbausteins) zugeordnet.
- In AWL gibt es zwei Möglichkeiten zur Parametrierung:
 - a) Parametrierung durch Laden und Speichern
 - Die Eingangsparameter werden vor dem Bausteinaufruf mit LD und ST zugewiesen.
 - Der Zugriff erfolgt über Instanzname.Formalparameter.
 - b) Parametrierung beim Bausteinaufruf
 - Die Aktualparameter werden direkt beim Aufruf den Formalparametern zugewiesen.
 - Die Zuweisung erfolgt durch := innerhalb der Parameterliste.

```
LD  a
ST  FlipFlop.R    (* Rücksetzen vorbereiten *)
LD  b
ST  FlipFlop.S1   (* Setzen vorbereiten *)
```

```
CAL FlipFlop (R:=a, S1:=b)
```



■ SR – Setzdominantes Flip-Flop

- S1: Setzeingang (hat Vorrang)
- R: Rücksetzeingang
- Sind beide TRUE, bleibt der Ausgang TRUE (1).

■ RS – Rücksetzdominantes Flip-Flop

- S: Setzeingang
- R1: Rücksetzeingang (hat Vorrang)
- Sind beide TRUE, wird der Ausgang FALSE (0).

■ Merke: Die Zahl 1 kennzeichnet den Eingang mit Vorrang:

- SR → Setzen hat Vorrang (S1)
- RS → Rücksetzen hat Vorrang (R1)

■ SR - setzdominant

- S1 = 0, R = 0: Zustand bleibt gespeichert.
- S1 = 1, R = 0: Ausgang wird gesetzt ($Q = 1$).
- S1 = 0, R = 1: Ausgang wird zurückgesetzt ($Q = 0$).
- S1 = 1, R = 1: Ausgang wird gesetzt ($Q = 1$), weil Set Vorrang hat.

■ RS – rücksetzdominant

- S = 0, R1 = 0: Zustand bleibt gespeichert.
- S = 1, R1 = 0: Ausgang wird gesetzt ($Q = 1$).
- S = 0, R1 = 1: Ausgang wird zurückgesetzt ($Q = 0$).
- S = 1, R1 = 1: Ausgang wird zurückgesetzt ($Q = 0$), weil Reset Vorrang hat.



- Nach der Instanziierung und Parametrierung muss der Funktionsbaustein im SPS-Programm aufgerufen werden.
- Der Aufruf erfolgt in AWL mit dem Operator CAL (Call).
- Durch den Aufruf wird der Funktionsbaustein ausgeführt:
 - Die aktuell zugewiesenen Eingangswerte werden verarbeitet.
 - Gespeicherte interne Zustände werden berücksichtigt.
 - Die Ausgangswerte werden berechnet und gegebenenfalls interne Zustände aktualisiert.
- Der Funktionsbaustein wird über seinen Instanznamen aufgerufen.

```
CAL FlipFlop      (* Funktionsbaustein ausführen *)
```



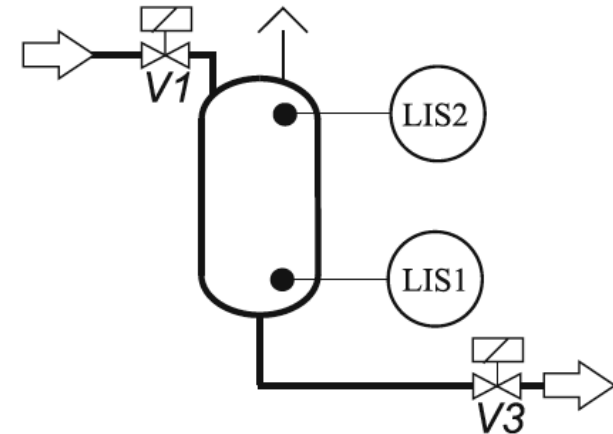
- Nach der Ausführung des Funktionsbausteins stehen die berechneten Ausgangswerte zur Verfügung.
- Auf die Ausgänge wird über den Instanznamen und den Ausgangsparameternamen zugegriffen.
- Die Schreibweise lautet Instanzname.Ausgangsparameter.
- Mit LD können die Ausgangswerte geladen und anschließend mit weiteren AWL-Anweisungen verarbeitet werden.
- Mit ST kann das Ergebnis beispielsweise auf einen physischen SPS-Ausgang geschrieben werden.

```
LD  FlipFlop.Q1
ST  x
```

ÜBUNG TANK 2

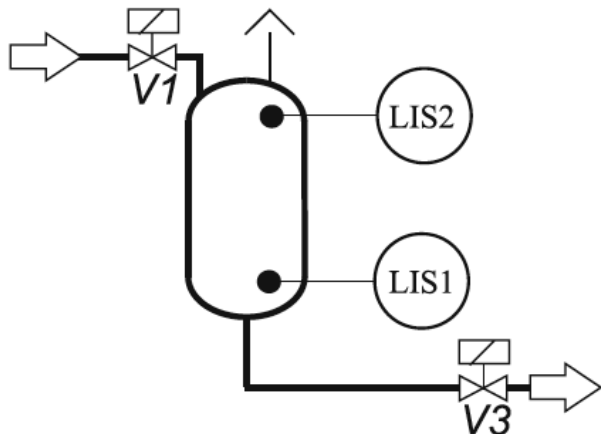


- Das Messgefäß soll so gesteuert werden, dass nach Impulsgabe über einen Taster „Füllen“ Wasser über das Magnetventil V1 einläuft, bis der Zustand „Voll“ vom Grenzwertgeber LIS2 gemeldet wird.
- Die LIS geben Signal „1“, wenn sie in Flüssigkeit eintauchen.
- Erstellen Sie die Anweisungsliste. Verwenden Sie für die Speicherung die Flip-Flop-Funktionsbausteine RS.



Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1

ANWEISUNGSLISTE (AWL) – BEISPIEL



Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1

Verwenden der Ausgangswerte

Initialisieren

Parametrierung

Aufruf im Programm

Program Tank62

```
(*Processes: *)  
(* Tanks (small) *)
```

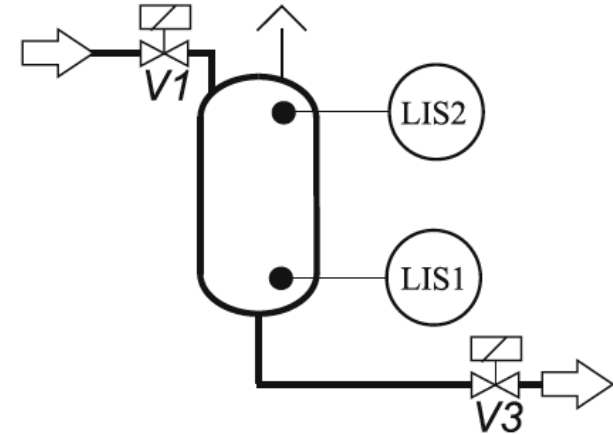
ANWEISUNGSLISTE (AWL) – BEISPIEL



Erweiterung um die Funktion: Tank entleeren

(Hinweis: Einseitige Verriegelung realisieren: V3 kann nur bei geschlossenem V1 geöffnet werden.
V1 kann unabhängig vom Zustand von V3 geöffnet werden.)

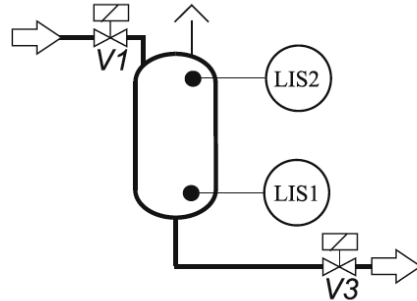
Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3



ANWEISUNGSLISTE (AWL) – BEISPIEL



Erweiterung um die Funktion: Tank entleeren

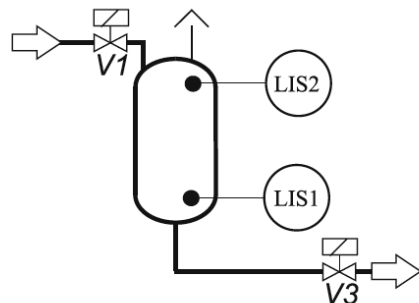


Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3

ANWEISUNGSLISTE (AWL) – BEISPIEL

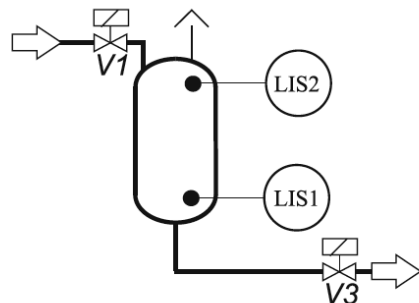


Nach dem Füllen soll das
Entleeren automatisch erfolgen



Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3

ANWEISUNGSLISTE (AWL) – BEISPIEL

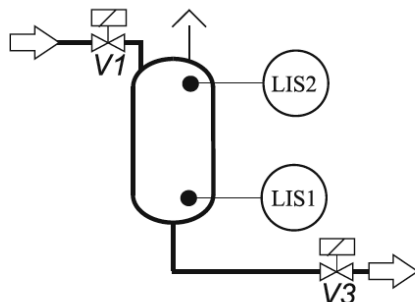


Nach Entleerung des Kessels, soll das Ventil noch weitere 3 sec. offen bleiben um zu gewährleisten, dass der Kessel komplett leer ist. Danach darf erst wieder der Befüllvorgang stattfinden.

Nach Befüllung soll für 1 Sekunde eine Hupe ertönen. Danach darf die Entleerung beginnen.

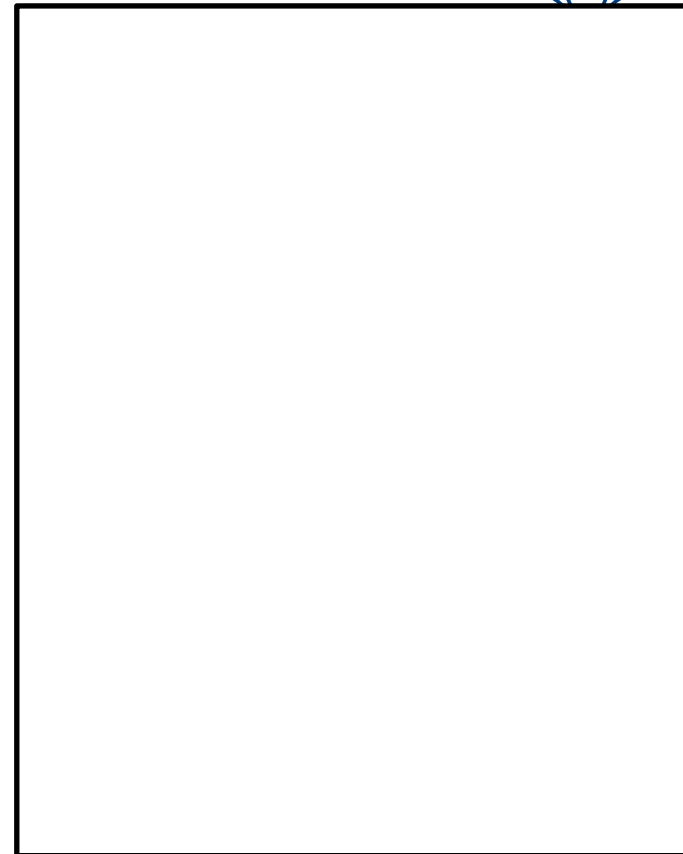
Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3

ANWEISUNGSLISTE (AWL) – BEISPIEL



Program Tank72

```
(*Processes: *  
(* Tanks (small) *)
```

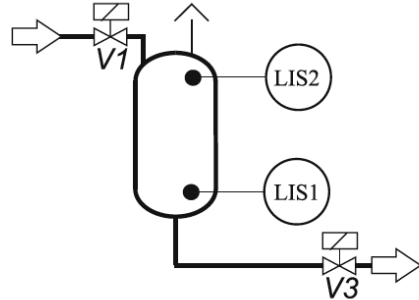


Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3

ANWEISUNGSLISTE (AWL) – BEISPIEL



Nach dem Füllen soll das
Entleeren automatisch erfolgen



```
Program Tank65
(*Processes:
(* Tanks (small)
```

```
*
*
```

Fill	Behälter füllen (Handtaster)	%IX0.7
LIS2	Oberer Grenzwertgeber	%IX0.2
V1	Einlassventil	%QX0.1
Empty	Behälter ablassen (Handtaster)	%IX0.6
LIS1	Unterer Grenzwertgeber	%IX0.1
V3	Auslassventil	%QX0.3

ANWEISUNGSLISTE (AWL) – ANHANG



AWL

Operator	Modifizierer	Operand Typ	Bedeutung
LD	N	<i>alle</i> ^a	Setzt das Aktuelle Ergebnis (AE) dem Operanden gleich
ST	N	<i>alle</i> ^a	Speichert das AE auf die Operanden-Adresse
S		BOOL	Setzt Operator auf 1 wenn aktuelles Ergebnis TRUE ist
R		BOOL	Setzt Operator auf 0 zurück wenn AE TRUE ist
AND	N (	ANY_BIT ^a	Bitweises UND
OR	N (	ANY_BIT ^a	Bitweises ODER
XOR	N (	ANY_BIT ^a	Bitweises Exklusiv-ODER
ADD	(	ANY_NUM, TIME	Addition

ANWEISUNGSLISTE (AWL) – ANHANG



Operator	Modifizierer	Operand Typ	Bedeutung
SUB	(	ANY_NUM, TIME	Subtraktion
MUL	(	ANY_NUM, TIME	Multiplikation
DIV	(	ANY_NUM, TIME	Division
GT	(	<i>alle</i>	Vergleich: >
GE	(	<i>alle</i>	Vergleich: >=
EQ	(	<i>alle</i>	Vergleich: =
NE	(	<i>alle</i>	Vergleich: <>
LE	(	<i>alle</i>	Vergleich: <

ANWEISUNGSLISTE (AWL) – ANHANG



Operator	Modifizierer	Operand Typ	Bedeutung
LT	(	<i>alle</i>	Vergleich: <=
JMP	C N	MARKE	Sprung zur MARKE
CAL	C N	NAME	Aufruf Funktionsbaustein NAME ^b
RET	C N		Rücksprung aus der Funktion oder FB
)			Bearbeitung der zurückgestellten Operation

STRUKTURIERTER TEXT (ST)



- Hochsprache mit Ähnlichkeit zu C / Pascal
- Bessere Strukturierung und Modularisierung möglich

```
VAR                                     (* Deklaration von Variablen *)
    SchalterLAT %IX0.1 : BOOL;
    SchalterRAT %IX0.2 : BOOL;
    Presse      AT %QX0.1 : BOOL;
END_VAR

IF (SchalterL = 1 AND SchalterR = 1) THEN (* Möglichkeit 1 *)
    Presse := 1;
ELSE
    Presse := 0;
END_IF

Presse := SchalterL & SchalterR;      (* Möglichkeit 2 *)
```

STRUKTURIERTER TEXT (ST)



- Der **strukturierte Text** (engl. Structured Text, Abkürzung: **ST**) wird für die Programmierung von Speicherprogrammierbaren Steuerungen (**SPS**) eingesetzt.
- Die Norm **EN 61131-3** legt unter anderem den Sprachumfang von ST fest. Dabei ist die Syntax der Sprachelemente ähnlich der Hochsprache Pascal.
- Bei allen Sprachen der EN 61131-3 gibt **keine Unterscheidung zwischen Groß- und Kleinschreibung** (Case Insensitive).
- Structured Text bietet dabei deutlich **mehr Strukturierungsmöglichkeiten** als AWL und wird häufig präferiert.



Vorteile im Vergleich zu AWL:

- Sehr kompakte Formulierung der Programmieraufgabe
- Übersichtlicher Aufbau der Programme durch Anweisungsblöcke
- Mächtige Konstrukte der Steuerung des Befehlsflusses

Nachteile:

- Übersetzung ist nicht direkt beeinflussbar (wird durch einen Compiler erfüllt)
- Effizienzverlust (übersetzte Programme sind in der Regel länger und langsamer)



Anweisung in ST

- Die Lösung einer Programmieraufgabe wird in einzelne Schritte (Anweisungen) zerlegt
- Anzahl von Anweisungen die durch Strichpunkt getrennt sind
- Zeilenende spielt keine Rolle
- Anweisung kann sich über mehrere Zeilen erstrecken
- Kommentare können in der Anweisung stehen → (*Kommentar*)

A:= B (*Dehnung*) + C (*Temperatur*);



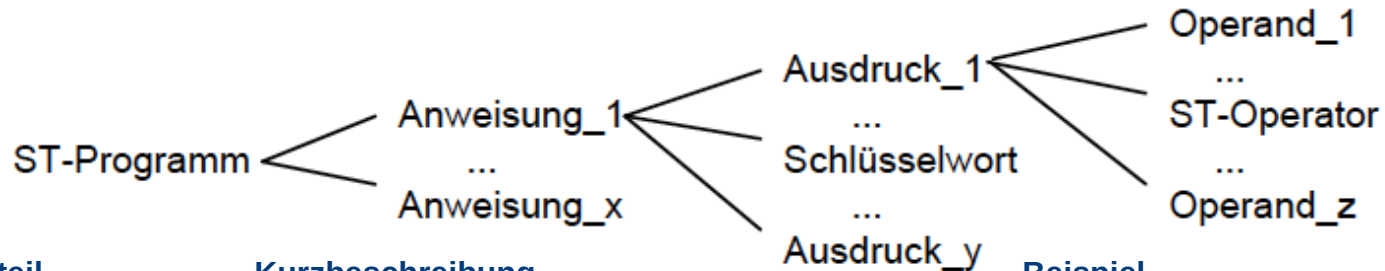
Schlüsselwort	Beschreibung	Beispiel	Erklärung
:=	Zuweisung	d := 10;	Zuweisung des rechts berechneten Wertes an den links stehenden Bezeichner
	Aufruf und Gebrauch von FBs ^a	FBName(Par1:=10, Par2:=20, Par3=>Erg);	Aufruf einer anderen POE vom Typ FB, sowie deren Parameterversorgung. „:=“ für Eingangs-, „=>“ für Ausgangsparameter
RETURN	Rücksprung	RETURN;	Rückkehr in die aufrufende POE
IF	Verzweigung	IF d < e THEN f:=1; ELSIF d=e THEN f:=2; ELSE f:= 3; END_IF;	Auswahl von Alternativen durch boolesche Ausdrücke
CASE	Multiauswahl	CASE f OF 1: g:=11; 2: g:=12; ELSE g:=FunName(); END_CASE;	Auswahl eines Anweisungsblocks, abhängig vom Wert des Ausdrucks f



Schlüsselwort	Beschreibung	Beispiel	Erklärung
FOR	Wiederholungsanweisung (1)	FOR h:=1 TO 10 BY 2 DO f[h/2] := h; END_FOR ;	Mehrfachdurchlauf eines Anweisungsblocks mit Start- und Endebedingung
WHILE	Wiederholungsanweisung (2)	WHILE m > 1 DO n := n / 2; END_WHILE ;	Mehrfachdurchlauf eines Anweisungsblocks mit Endebedingung
REPEAT	Wiederholungsanweisung (3)	REPEAT i := i*j; UNTIL i > 10000 END_REPEAT ;	Mehrfachdurchlauf eines Anweisungsblocks mit nachgeschalteter Endebedingung
EXIT	Schleifen-Abbruch	EXIT ;	Abbruch einer Wiederholungsanweisung
;	Leeranweisung	;;	



- Ausdrücke liefern die zur Bearbeitung von Anweisungen nötigen Werte
- Ein Ausdruck besteht aus Operanden und verknüpfenden ST-Operatoren



Bestandteil	Kurzbeschreibung	Beispiel
ST-Programm	Gesamtes Programm aus mehreren Anweisungen	PROGRAM Tank
Anweisung	Legt eine auszuführende Aktion fest	Ventil := TRUE;
Schlüsselwort	Steuert die Programmstruktur und den Ablauf	IF, THEN, ELSE
Ausdruck	Liefert einen Wert zur Verarbeitung	Temperatur > 50
Operand	Wert oder Variable, die verarbeitet wird	Temperatur, 50
ST-Operator	Verknüpft oder verarbeitet Operanden	+, -, >, AND



Literale

- numerische, Zeichen, Zeit
 - Beispiel: 12, 'abcdef', t#3d_7h, UINT#9

Variable

- Einzel-, Multielement-Variable
 - Beispiel: Var1, Var[1,2], Var1.Unterstruc
- Funktionsaufruf bzw. der errechnete Funktionswert FunName
 - Beispiel: Part1, Part2, Part3
- Weiterer Ausdruck
 - Beispiel: 10 + 20

OPERATOREN



Zur Verknüpfung der Operanden innerhalb von Ausdrücken gibt es in ST folgende Operatoren:

Operator	Beschreibung	Beispiel mit Wert des Ausdrucks	Rangfolge
()	Klammerung	$(2 * 3) + (4 * 5)$ Wert: 26	Hoch
	Funktionsaufruf	CONCAT ('AB', 'CD') Wert: 'ABCD'	
-	Negation	-10 Wert: -10	
NOT	Komplement	NOT TRUE Wert: FALSE	
**	Potenzierung	3 ** 4 Wert: 81	
*	Multiplikation	10 * 20 Wert: 200	
/	Division	20 / 10 Wert: 2	
MOD	Modulo	17 MOD 10 Wert: 7	

OPERATOREN



Operator	Beschreibung	Beispiel mit Wert des Ausdrucks	Rangfolge
+	Addition	.4 + 2.5 Wert: 3.9	
-	Subtraktion	3 - 2 - 1 Wert: 0	
<, >, <=, >=	Vergleich	10 > 20 Wert: FALSE	
=	Gleichheit	T#26h = T#1d2h Wert: TRUE	
◇	Ungleichheit	8#15 ◇> 13 Wert: FALSE	
&, AND	Boolesches UND	TRUE AND FALSE Wert: FALSE	
XOR	Exklusives Boolesches OR	TRUE XOR FALSE Wert: TRUE	
OR	Boolesches ODER	TRUE OR FALSE Wert: TRUE	niedrig

BEISPIEL FÜR DIE RANGFOLGE



Beispiel für die Rangfolge

$$(115 - (15 + 20 * 2)) \text{ MOD } 24 \text{ MOD } 10 + 10$$

- $20 * 2 = 40$
- $15 + 40 = 55$
- $115 - 55 = 60$
- $60 \text{ MOD } 24 = 12$
- $12 \text{ MOD } 10 = 2$
- $2 + 10 = 12$

FUNKTIONEN ALS OPERATOREN



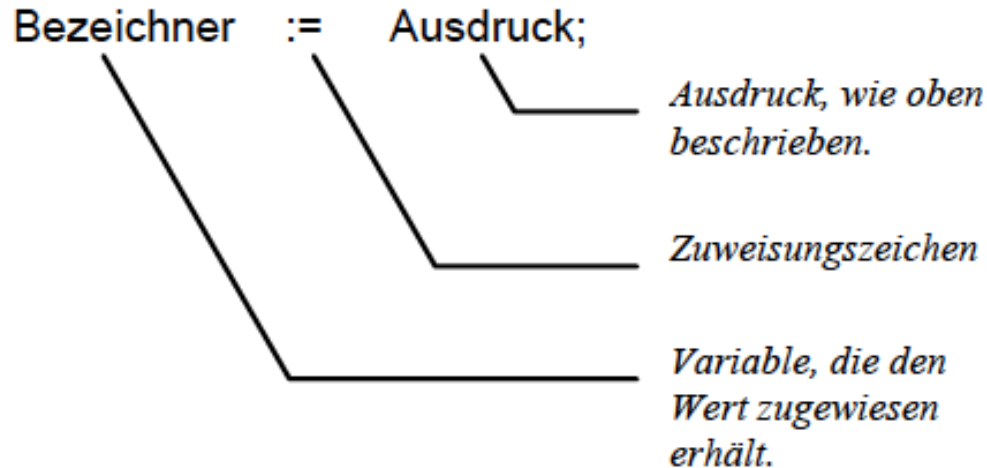
- Ein Ausdruck kann sowohl mit Hilfe eines Operators (z.B. $x+y$) oder einer Standard-Funktion (z.B. `ADD(x,y)`) beschrieben werden
- Funktionen werden über ihre Namen und in Klammern die Aktualparameter aufgerufen

<code>ADD (1, 2, 3);</code>	(* Vollständige Angabe der Aktualparameter *)
<code>LIMIT (MN := 0, MX := 10, IN := 4);</code>	(* Angabe mit Formalparameter *)
<code>AWFUN(EinP1:=10, AusPar=>Erg);</code>	(* Anwenderfunktion mit 1 Eingangsparameter, 1 Funktionswert (Rückgabewert) und 1 zusätzlichen Ausgangsparameter *)

Anweisung: Zuweisung



Eine Zuweisung **kopiert den rechts vom Gleichheitszeichen stehenden Wert in die links stehende Einzel- oder Multielement-Variable.**



Anweisung: Zuweisung – Beispiel



```
TYPE MulVar: STRUCT Var1: INT; Var2: REAL; END_STRUCT; END_TYPE
```

```
VAR  d:  INT;  
     e:  ARRAY [0..9] OF INT;  
     f:  REAL;  
     g:  MulVar (Var1:=10, Var2 := 2.3);  
     h:  MulVar;  
END_VAR  
d      := 10;                (* Zuweisung *)  
e[0]   := d ** 2;  h := g;    (* Zwei Zuweisungen in einer Zeile *)  
d      := REAL_TO_INT(f);    (* Zuweisung mit Funktionsaufruf *)
```

Wichtig:

Bei Zuweisungen ist auf die **Typverträglichkeit** zu achten d.h. die rechte und linke Seite müssen den gleichen Datentyp haben.

UMWANDELN VON VARIABLEN



Variable können auch gewandelt werden:

- *BYTE_TO_WORD*: Byte zu Word
- *WORD_TO_DWORD*: Word zu Doppelword
- *TIME_TO_DINT*: Zeit zu Doppelinteger
- *TIME_TO_STRING*: Zeit zu String
- *STRING_TO_WORD*: String zu Word
- *STRING_TO_DATE*: *String* zu Datum
- *LREAL_TO_STRING*: *LReal* zu String
- *WORD_TO_BYTE*: Word zu Byte

UMWANDELN VON VARIABLEN – BEISPIEL



Beispiel:

Messwert soll gewandelt und gespeichert werden

VAR

w_Messwert : WORD := 105 ;

b_MW : Byte := 0 ;

END_VAR

b_MW := WORD_TO_BYTE (w_Messwert) ; (* Fehler, wenn w_Messwert > 255 *)

AUFRUF VON FUNKTIONSBAUSTEINEN



In ST wird ein FB über seinen Namen und seine in Klammer stehenden Formalparameter aufgerufen.

```
FUNCTION_BLOCK FbTyp
  VAR_INPUT
    VarEin: INT, VarH: INT := 1;
  END_VAR
  VAR_OUTPUT VarAus: INT := 5;
  END_VAR

  IF (VarEin > VarH) THEN
    VarAus := 10;
  END_IF;
END_FUNCTION_BLOCK
```

Aufruf (Verwendung) des FBs FbName:

```
...
VAR FbName: FbTyp; ERG: INT;
END_VAR

FbName();
(* FbName.VarAus besitzt den Wert 5 *)
FbName(VarEin := 3, VarAus=>ERG);
(* Alternativaufruf: FbName(3, 1, Erg *);
(* FbName.VarAus besitzt den Wert 10,
   welcher nach ERG kopiert wird *)
...
```



- **Variablen:** Dienen zum Speichern, Initialisieren und Verarbeiten von Daten.
- **Deklaration:** Erfolgt zu Beginn jeder POE (Programmorganisationseinheit).
- **Datentyp:** Jeder Variablen wird ein Datentyp zugewiesen (z. B. BOOL, INT, REAL).
- **Eigenschaften:** Anfangswerte, Speicherverhalten (z. B. remanent) und physikalische Adressen können festgelegt werden.
- **Deklarationsblöcke:** Variablen werden nach ihrem Verwendungszweck gruppiert (VAR, VAR_INPUT, VAR_OUTPUT usw.).
- **Aufbau:** Jeder Block beginnt mit **VAR_*** und endet mit **END_VAR**.

```
(* Lokale Variable *)
VAR          VarLokal : BOOL;  END_VAR  (* lokale boolesche Var. *)
(* Aufrufschnittstelle: Eingangsparameter *)
VAR_INPUT    VarEin : REAL;     END_VAR  (* Eingangsvariable *)
VAR_IN_OUT   VarEinAus : UINT;  END_VAR  (* Ein- und Ausgangsvariable *)
(* Rückgabewerte: Ausgangsparameter *)
VAR_OUTPUT   VarAus : INT;      END_VAR  (* Ausgangsvariable *)
(* Globale Schnittstelle: Globale/Externe Variablen und Zugriffspfade *)
VAR_EXTERNAL VarGlob : WORD;    END_VAR  (* extern von anderer POE *)
VAR_GLOBAL   VarGlob : WORD;    END_VAR  (* global für andere POEs *)
VAR_ACCESS   VarPfad : WORD;    END_VAR  (* Zugriffspfade der Konfiguration *)
```

VARIABLENDEKLARATION NACH IEC 61131-3



Variablenart	Zugriffsrechte ^a		Erläuterung
	extern	intern	
VAR_TEMP <i>lokale Variable</i>	-	SL	Eine lokale Variable ist nur innerhalb der POE sichtbar und kann nur dort bearbeitet werden.
VAR_INPUT <i>Eingangsvariable</i>	S ^b	L	Eine Eingangsvariable ist in der aufrufenden POE sichtbar und dort beschreibbar. Sie darf innerhalb der POE nur gelesen werden.
VAR_OUTPUT <i>Ausgangsvariable</i>	L	SL	Eine Ausgangsvariable ist in der aufrufenden POE sichtbar und kann dort nur gelesen werden. Sie ist durch die POE les- und beschreibbar.
VAR_IN_OUT <i>Ein- und Ausgangsvariable</i>	SL	SL	Eine Ein-/Ausgangsvariable beinhaltet die kombinierten Eigenschaften von VAR_INPUT und VAR_OUTPUT: sie ist innerhalb und außerhalb der POE les- und beschreibbar.
VAR_EXTERNAL <i>Externe Variable</i>	SL	SL	Eine externe Variable wird benötigt, um innerhalb einer POE auf eine globale Variable einer anderen POE lesend und schreibend zuzugreifen. Sie ist nur für POEs sichtbar, die diese globale Variable unter VAR_EXTERNAL auflisten, alle anderen haben keinen Zugriff auf diese globale Variable. Bezeichner und Typ einer Variablen unter VAR_EXTERNAL muss mit der korrespondierenden VAR_GLOBAL Deklaration im PROGRAM übereinstimmen.
VAR_GLOBAL <i>Globale Variable</i>	SL	SL	Auf eine GLOBAL deklarierte Variable kann von mehreren POEs lesend und schreibend zugegriffen werden. Hierzu muss die Variable in den anderen POEs unter VAR_EXTERNAL mit identischem Namen und identischem Typ angegeben werden.
VAR_ACCESS <i>Zugriffspfade</i>	SL	SL	Globale Variable von Konfigurationen als Kommunikationskanal zwischen Komponenten (Ressourcen) von Konfigurationen (vgl. Kap. 6). Sie kann innerhalb der POE wie eine globale Variable benutzt werden.

a S = schreibend, L = lesend, SL = schreibend und lesend

b nur schreibbar als Formalparameter beim Aufruf



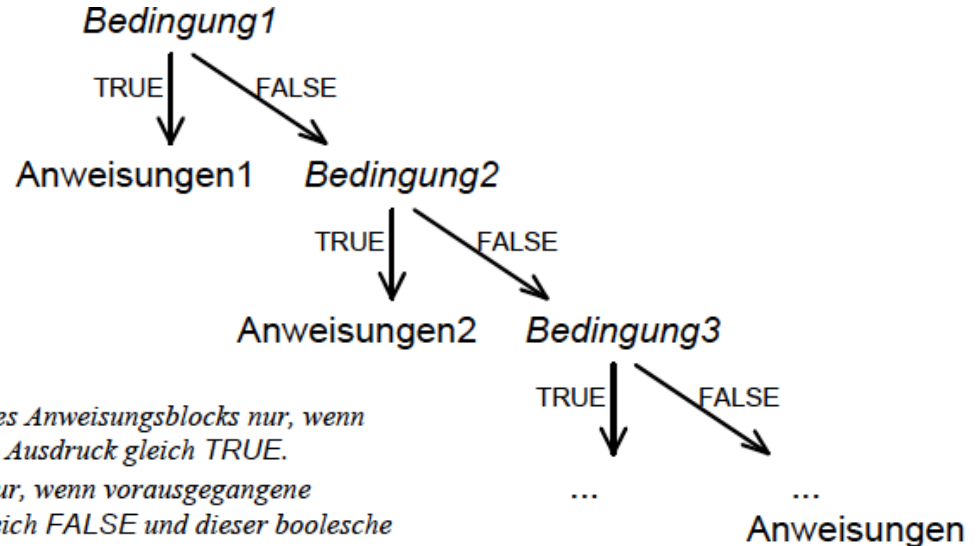
Soll eine Funktion, FB oder Programm verlassen werden, bevor das POE-Ende erreicht wird so kann dies durch die Anweisung RETURN erfolgen.

```
...  
(* Falls die Variable x kleiner als y, verlasse die POE *)  
IF x < y THEN RETURN;  
END_IF;  
...
```

Anweisung: Verzweigung, Multiauswahl

Sie dienen dazu, abhängig von einer Bedingung bestimmte Anweisungen auszuführen
Bedingungen sind die boolschen Ausdrücke **TRUE** oder **FALSE**.

Anweisung: IF



IF Ausdruck **THEN** Anweisungsblock;

Ausführung des Anweisungsblocks nur, wenn der boolesche Ausdruck gleich TRUE.

ELSIF Ausdruck **THEN** Anweisungsblock;

Ausführung nur, wenn vorausgegangene Ausdrücke gleich FALSE und dieser boolesche Ausdruck gleich TRUE. Diese Teilanweisung kann entfallen oder beliebig oft wiederholt werden.

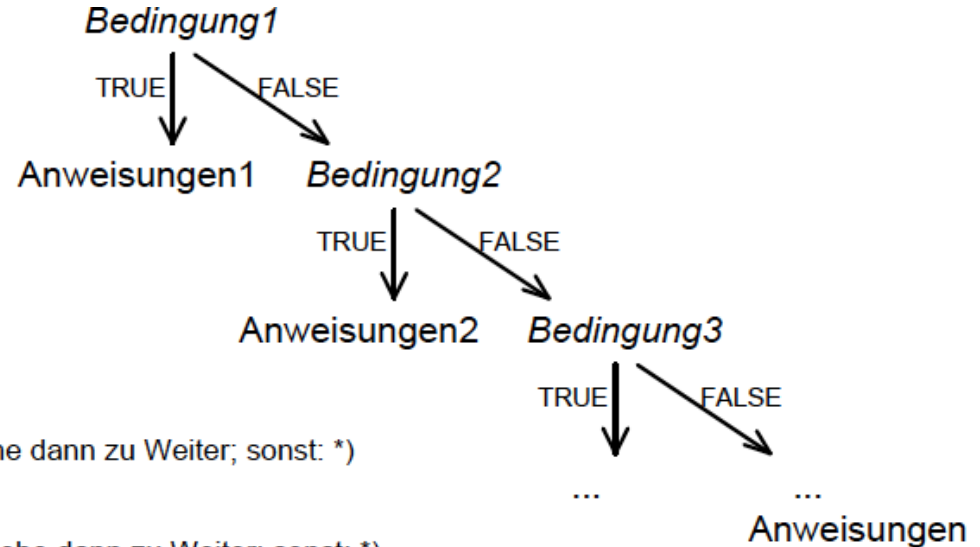
ELSE Anweisungsblock;

Ausführung nur, wenn vorausgegangene Ausdrücke gleich FALSE. Teilanweisung kann entfallen.

END_IF;

Schließende Klammer der IF-Anweisung

Anweisung: IF



```
IF Bedingung1 THEN
  Anweisungen1;
  (* Führe Anweisungen1 aus, falls Bedingung1 TRUE, gehe dann zu Weiter; sonst: *)
ELSIF Bedingung2 THEN
  Anweisungen2;
  (* Führe Anweisungen2 aus, falls Bedingung2 TRUE, gehe dann zu Weiter; sonst: *)
ELSIF Bedingung3 THEN
  ...
ELSE Anweisungen;
  (* Führe Anweisungen aus, falls keine der bisherigen Bedingungen zutrafen *)
END_IF;
(* Weiter *)
```

Anweisung: IF – Beispiel

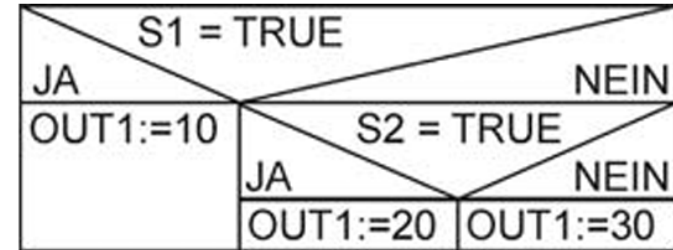


Beispiel:

Anweisungen

```
IF S1 THEN  
    OUT1 := 10 ;  
ELSIF S2 THEN  
    OUT1 := 20 ;  
ELSE  
    OUT1 := 30 ;  
END_IF ;
```

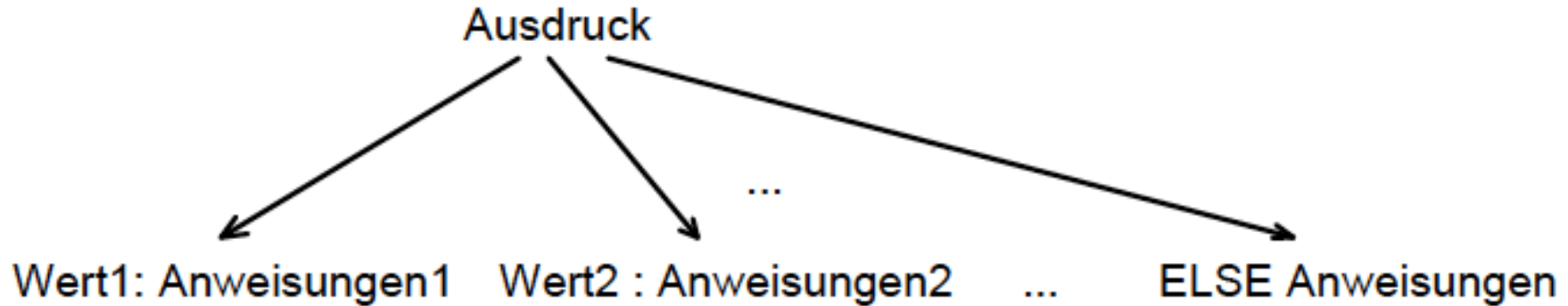
Zugehöriges Struktogramm



MULTIAUSWAHL UNTER VERWENDUNG VON CASE



Multiauswahl unter Verwendung von CASE:



MULTIAUSWAHL UNTER VERWENDUNG VON CASE



Die **CASE-Anweisung** dient der Auswahl unter 1-n alternativen Programmteilen und wird nach folgenden Regeln bearbeitet:

- Der Auswahl-Ausdruck muss einen Wert vom Typ INTEGER liefern
- Bei der Abarbeitung der CASE-Anweisung wird überprüft, ob der Wert des Auswahlausdrucks in einer angegebenen Werteliste enthalten ist. Bei Übereinstimmung wird der der Liste zugeordnete Anweisungsteil ausgeführt.
- Ergibt der Vergleichsvorgang keine Übereinstimmung, so wird der Anweisungsteil nach ELSE ausgeführt oder keine Anweisung, falls der ELSE-Zweig nicht vorhanden ist.

CASE Ausdruck **OF**

case_wert : Anweisungsblock;

ELSE Anweisungsblock;

END_CASE;

Ausführung nur, wenn der Ausdruck gleich dem angegebenen case-Wert. Diese Teilanweisung kann einmal oder mehrfach aufgeführt werden. case_wert ist ein- oder mehrfacher ganzzahliger Integerwert (SINT, INT, DINT, LINT), ein Aufzählungswert oder eine entsprechende Bereichsangabe: "Wert1..Wert2".

Ausführung nur, wenn der Ausdruck keinem case_wert entspricht. Teilanweisung kann entfallen.

Schließende Klammer der CASE-Anweisung

MULTIAUSWAHL UNTER VERWENDUNG VON CASE

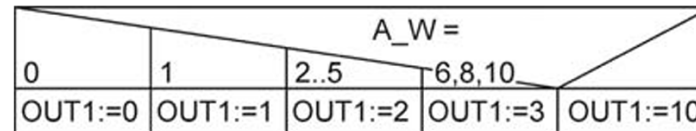


```
IF Bedingung1 THEN
  Anweisungen1;
  (* Führe Anweisungen1 aus, falls Bedingung1 TRUE, gehe dann zu Weiter; sonst: *)
ELSIF Bedingung2 THEN
  Anweisungen2;
  (* Führe Anweisungen2 aus, falls Bedingung2 TRUE, gehe dann zu Weiter; sonst: *)
ELSIF Bedingung3 THEN
  ...
ELSE Anweisungen;
  (* Führe Anweisungen aus, falls keine der bisherigen Bedingungen zutrafen *)
END_IF;
(* Weiter *)
```

Anweisungen

```
CASE A_W OF
  0: OUT1:=0;
  1: OUT1:=1;
  2..5: OUT1:=2;
  6, 8, 10: OUT1:=3;
ELSE:
  OUT1:=10;
END_CASE;
```

Zugehöriges Struktogramm





Das sind Ganzzahl-Datentypen (Integer) in der SPS-Programmierung nach IEC 61131-3. Sie unterscheiden sich in ihrer Speichergröße und ihrem Wertebereich.

Datentyp	Bedeutung	Speichergröße	Wertebereich
SINT	Short Integer	8 Bit (1 Byte)	–128 bis 127
INT	Integer	16 Bit (2 Byte)	–32.768 bis 32.767
DINT	Double Integer	32 Bit (4 Byte)	–2.147.483.648 bis 2.147.483.647
LINT	Long Integer	64 Bit (8 Byte)	-2^{63} bis $2^{63} - 1$



CASE Anweisung:

```
CASE VarInteger OF
  1:    Anweisungen1;
        (* Führe Anweisungen1 aus, falls VarInteger 1 liefert; gehe zu Weiter *)
  2,3:  Anweisungen2;
        (* Führe Anweisungen2 aus, falls VarInteger 2 oder 3 liefert; gehe zu Weiter *)
  10..20: Anweisungen3;
        (* Führe Anweisungen3 aus, falls VarInteger zwischen 10 und 20 liegt *)
        (* gehe zu Weiter *)
  ELSE  Anweisungen;
        (* Führe Anweisungen aus, falls keine der bisherigen Bedingungen zutraf *)
END_CASE;
...
(* Weiter *)
```

ANWEISUNG: WHILE

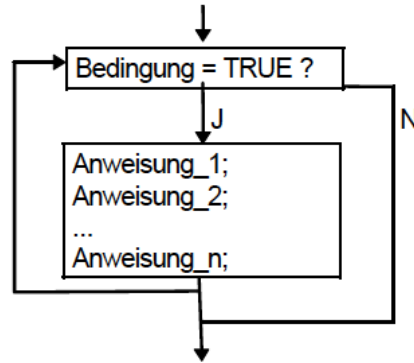


Die **WHILE Anweisung** erlaubt die **wiederholte Ausführung einer Anweisungsfolge unter der Kontrolle einer Durchführungsbedingung**, welche nach den Regeln eines logischen Ausdrucks gebildet wird.

Die **WHILE Anweisung** wird nach folgenden Regeln bearbeitet:

- Vor jeder Ausführung des Anweisungsteils wird die Durchführungsbedingung ausgewertet
- Der DO folgende Anweisungsteil wird solange wiederholt, wie die Durchführungsbedingung dem Wert TRUE liefert
- Tritt der Wert FALSE auf, wird die Schleife übersprungen und die der Schleife folgende Anweisung ausgeführt.

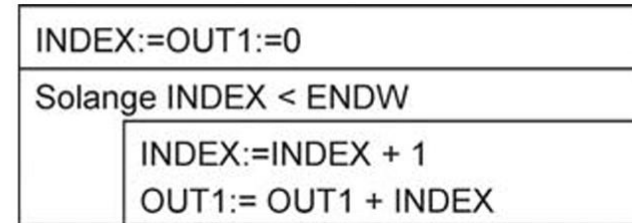
ANWEISUNG: WHILE



Anweisungen

```
INDEX:=0;  
OUT1:=0;  
WHILE INDEX < ENDW DO  
    INDEX:=INDEX + 1;  
    OUT1:=OUT1+INDEX;  
END_WHILE;
```

Zugehöriges Struktogramm



ANWEISUNG: REPEAT



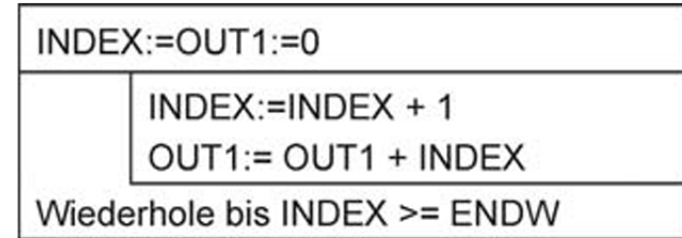
Eine **REPEAT-Anweisung** bewirkt die wiederholte Ausführung einer zwischen REPEAT und UNTIL stehenden Anweisungsfolge bis zum Eintreten einer Abbruchbedingung.

Die Bedingung wird nach der Ausführung des Rumpfes überprüft, d.h. der Rumpf wird mindestens einmal ausgeführt, auch wenn die Abbruchbedingung von Anfang an erfüllt ist.

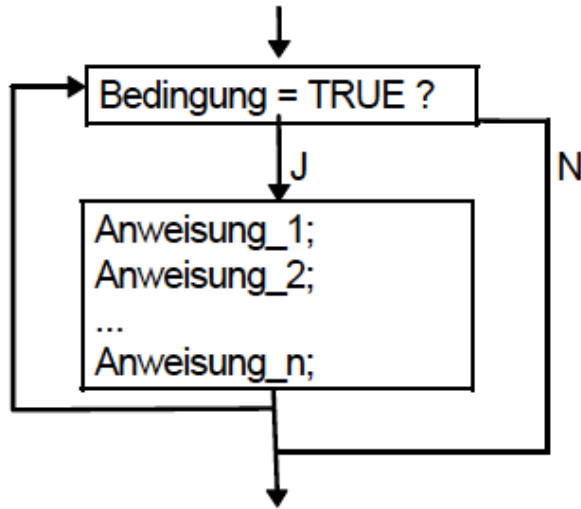
Anweisungen

```
OUT1:=0;  
INDEX:=0;  
REPEAT  
    INDEX:=INDEX + 1;  
    OUT1:=OUT1 + INDEX;  
UNTIL INDEX >= ENDW  
END_REPEAT;
```

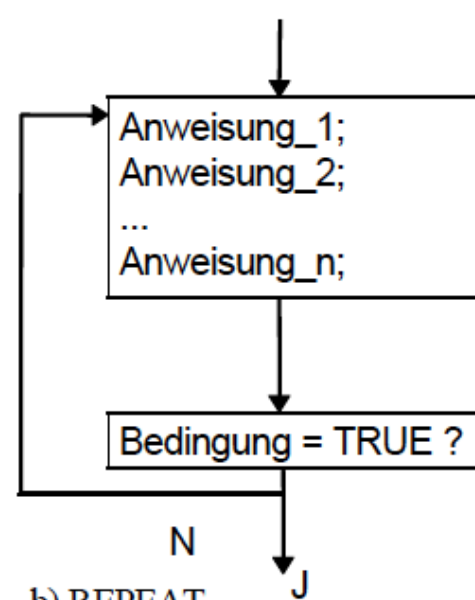
Zugehöriges Struktogramm



VERGLEICH WHILE UND REPEAT



a) WHILE



b) REPEAT

ANWEISUNG: FOR



Die **FOR Anweisung** benutzt eine ganzzahlige Steuervariable, die vor dem ersten Durchlauf mit einem frei bestimmbaran Anfangswert initialisiert wird und nach jedem Durchlauf um die Schrittweite erhöht oder verringert wird.

```
FOR Steuervariable := Ausdruck TO Ausdruck BY Ausdruck DO
    Anweisungsblock; \ Anfangswert \ Endwert \ Schrittweite
END_FOR;
\ Schließende Klammer der FOR - Anweisung
```

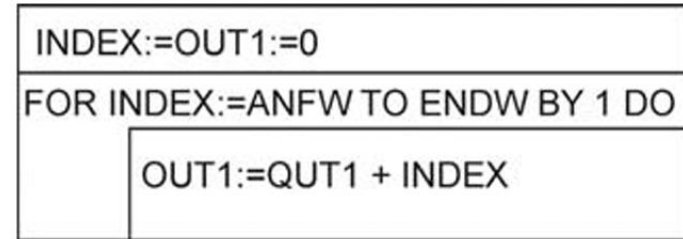


Beispiel:

Anweisungen

```
INDEX:=0;  
OUT1:=0;  
FOR INDEX:= ANFW TO ENDW BY 1 DO  
    OUT1:=OU1 + INDEX;  
END_FOR;
```

Zugehöriges Struktogramm



STANDARD-FUNKTIONSBAUSTEIN TON (EINSCHALTVERZÖGERUNG)



Der TON (Timer On Delay) ist ein Zeitbaustein, der das Einschalten eines Ausgangs um eine festgelegte Zeit verzögert.

- TON (Timer On Delay): Verzögert das Einschalten eines Ausgangs um eine vorgegebene Zeit.
- IN (BOOL): Startet die Zeitmessung bei `TRUE`.
- PT (TIME): Vorgegebene Verzögerungszeit.
- ET (TIME): Bereits abgelaufene Zeit.
- Q (BOOL): Wird `TRUE`, sobald die Verzögerungszeit erreicht ist.
- Rücksetzen: Bei `IN = FALSE` werden `Q = FALSE` und `ET = T#0s`.

```
VAR
    SchmorTest : TON;
    ZEITW      : TIME := T#2s;
END_VAR
```

```
SchmorTest(IN := Uebersteuer, PT := ZEITW);
LED := SchmorTest.Q;
```