

# Automatisierungstechnik

## 1 Einführung

**Automatisierung:** Reale Aufgaben der Automatisierungstechnik sind im Allgemeinen sehr komplex. Als umfassender Ausdruck für Steuerungs-, Regelungs- und Visualisierungs-Vorgänge hat sich der Begriff der Automatisierung durchgesetzt. Er beinhaltet, dass Automatisierungsgeräte selbsttätig Programme befolgen und dabei Entscheidungen auf Grund vorgegebener Führungsgrößen und rückgeführter Prozessgrößen aus der Anlage sowie erforderlicher Daten aus internen Speichern des Systems treffen, um daraus notwendige Ausgangsgrößen für den Betriebsprozess zu bilden.

**Steuerung:** Steuern oder Steuerung wird als Ablauf in einem System definiert, bei dem eine oder mehrere Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der *offene Wirkungsablauf über der Steuerstrecke*. Eine Steuerung liegt also vor, wenn Eingangsgrößen nach einer festgelegten Gesetzmäßigkeit Ausgangsgrößen beeinflussen. Die Auswirkung einer nicht vorhersehbaren Störgröße wird nicht ausgeglichen.

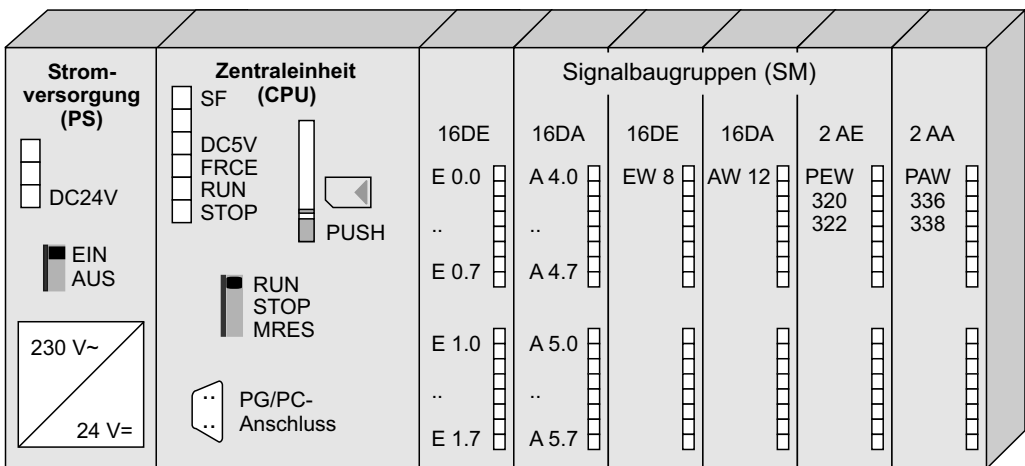
**Regelung:** Immer dann, wenn Störgrößenänderungen das System nicht hinnehmbar beeinflussen können, werden Regelungen erforderlich. Die Regelgröße (Aufgabengröße) muss sich messtechnisch erfassen lassen, denn eine Regelung ist ein Vorgang, bei dem die Regelgröße fortlaufend erfasst, mit der Führungsgröße verglichen und abhängig vom Ergebnis dieses Vergleichs im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Der sich dabei ergeben-

de Wirkungsablauf findet in einem geschlossenen Kreis, dem *Regelkreis*, statt. Programmierbare Automatisierungsgeräte können die Aufgaben des Steuerns und Regelns ausführen, denn beide Funktionen beruhen auf Programmen unter Verwendung desselben Operationsvorrates.

## 2 Automatisierungsgeräte

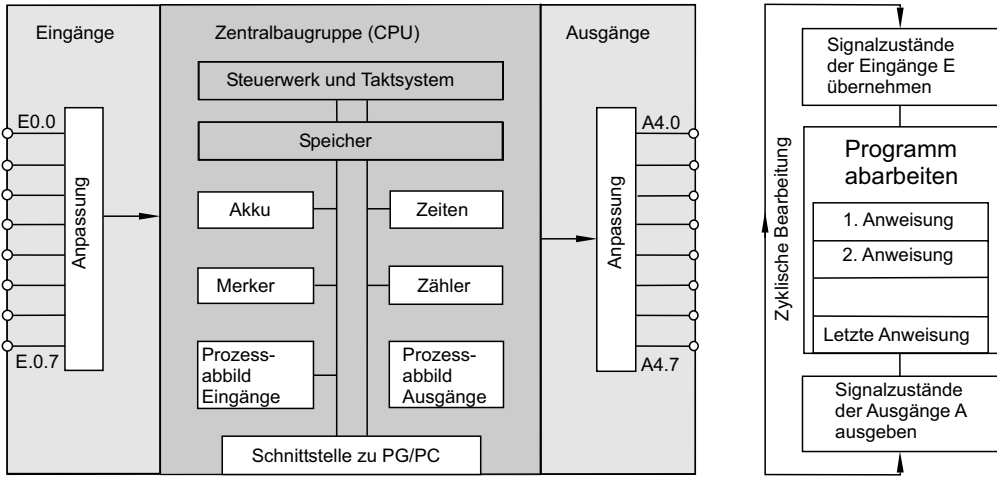
Die derzeit am weitesten verbreitete Hardware-Plattform der Steuerungstechnik ist die *Speicherprogrammierbare Steuerung SPS*, wie sie in Bild 1 abgebildet ist, dort jedoch ohne den heute schon üblichen Anschluss an ein Feldbusystem zur Vernetzung mit anderen Steuerungskomponenten.

Eine Speicherprogrammierbare Steuerung hat die Struktur eines Rechners, deren Funktion als Programm gespeichert ist. Sie besteht im einfachsten Fall aus einer *Stromversorgung PS*, einem *Steuerungsprozessor CPU*, einigen zentralen *digitalen Eingabe- und Ausgabebaugruppen* sowie einem internen Bussystem. Bei Bedarf können auch Baugruppen zur *Analogwertverarbeitung* oder für besondere Funktionen wie *Regler*, schnelle *Zähler* und *Positionierungen* hinzukommen. Die Peripheriebaugruppen und die Programmiersprachen sind auf die Belange der Steuerungstechnik ausgerichtet. Speicherprogrammierbare Steuerungen gibt es als modulare und kompakte Systeme für unterschiedliche Anforderungsniveaus. Ein typisches Merkmal von SPS-Steuerungen ist die zyklische Programmbearbeitung, wie in Bild 2 angedeutet.



**Bild 1** Aufbau einer Speicherprogrammierbaren Steuerung ohne Feldbusanschluss

DE = Digitale Eingänge, DA = Digitale Ausgänge, AE = Analoge Eingänge, AA = Analoge Ausgänge



**Bild 2** SPS-Struktur und zyklische Programmabarbeitung

In der Praxis werden nicht mehr ausschließlich SPS-basierte Automatisierungssysteme eingesetzt, sondern vernetzte Steuerungssysteme bestehend aus SPSen und PCs mit unterschiedlichen Aufgaben, wobei die SPS mehr prozessnah und der PC mehr übergeordnet und datenverarbeitend genutzt wird. Soll die Steuerungsfunktionalität nicht mehr von einer modernen SPS, sondern von einem PC ausgeführt werden, so gibt es diese Lösung als Industrie-PC mit eingebauter SPS-Karte und zusätzlicher Feldbus-Schnittstelle. In Sonderfällen kommt auch eine sog. Soft-SPS-Lösung in Frage, bei der in einem PC die Funktion einer SPS softwaremäßig nachgebildet und Steuerungsaufgaben „nebenher“ miterledigt werden. Für die Programmerstellung macht es keinen Unterschied, ob die Hardware-Plattform ein PC oder eine SPS ist.

### 3 Grundzüge der SPS-Norm IEC 61131-3

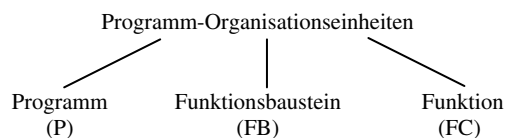
Auch wenn das Programmieren von Steuerungen und Regelungen noch immer ein sehr individueller Prozess ist, muss seit Jahren der Standard der SPS-Programmier-Norm IEC 61131-3 zu Grunde gelegt werden, der eine rationellere Programmerstellung und größere Herstellerunabhängigkeit zum Ziel hat. Den Anwendungsprogrammierern müssen wirksame Mittel an die Hand gegeben werden, um wiederverwendbare und damit kostensparende Programme entwickeln zu können. Die Norm IEC 61131-3 bzw. DIN EN 61131-3 richtet sich daher in erster Linie an SPS-Hersteller. Für Anwendungsprogrammierer ist die SPS-Norm eher ein Dokument im Hintergrund, denn sie sind für ihre Arbeit auf ein reales IEC 61131-3 kompatibles Programmier- und SPS-System angewiesen.

Die Grundzüge dieser SPS-Norm werden in den folgenden Abschnitten dargelegt und bilden die Voraussetzungen dafür, mit einem der SPS-Norm entsprechenden Programmiersystem richtig umgehen zu können.

#### 3.1 Programmorganisationskonzept

Es ist ein hierarchisch gegliedertes System von so genannten Programmorganisationseinheiten (POE) eingeführt worden, bestehend aus einem Hauptprogrammtyp (Schlüsselwort: PROGRAM) mit Zugriffsmöglichkeit auf SPS-Eingänge/-Ausgänge und zwei Unterprogrammtypen davon einen Typ mit Gedächtnisfunktion (Schlüsselwort: FUNCTION\_BLOCK) und einen Typ ohne Gedächtnisfunktion (Schlüsselwort: FUNCTION). Durch dieses Programmorganisationskonzept in Verbindung mit dem Datentyp- und Variablenkonzept zur Entwicklung strukturierter Programme wird die Wiederverwendbarkeit von Bausteinen bei späteren Anwendungen ermöglicht. Dadurch ist die neue Anwendungsprogrammierung sehr viel anspruchsvoller als die herkömmliche SPS-Programmierung geworden.

Ein Steuerungsprogramm (Anwenderprogramm) ist eine in Programm-Organisations-Einheiten (POE) gegliederte Einheit. Anstelle der unhandlichen Bezeichnung Programmorganisationseinheit wird vereinfacht auch nur von Bausteinen gesprochen.



Jeder Baustein besteht aus einem

- Deklarationsteil für die Definition aller lokal verwendeten Variablen und einem
- Rumpf für die Anweisungen (Befehle) des ausführbaren Programms.

**Funktion:** Dieser POE-Typ ist geeignet, wenn ein Funktionsergebnis ausschließlich aus den Eingangsvariablen des Bausteins zu ermitteln ist und unter dem Funktionsnamen des Bausteins zur Verfügung gestellt werden soll. Der Aufruf einer Funktion mit denselben Werten der Eingangsvariablen liefert deshalb immer denselben Ausgangswert zurück. Die SPS-Norm enthält einen Katalog von Standardfunktionen, die in SPS-Systemen zur Verfügung stehen sollten. Falls eine spezielle Funktion benötigt wird, kann diese vom Anwender selbst erzeugt werden, dabei muss jedoch beachtet werden, dass keine internen Zustandsvariablen deklarierbar sind, da der Baustein Typ Funktion dafür keine Speicherfähigkeit (Gedächtnis) besitzt. Eine Funktion stellt das Funktionsergebnis unter dem deklarierten Funktionsnamen zur Verfügung, sodass keine Ausgangsvariable deklariert werden muss. Es ist jedoch zulässig, Funktionen mit mehreren Ausgangsvariablen zu bilden. Funktionen können innerhalb eines Programmzyklus mehrfach aufgerufen werden, um mit unterschiedlichen Werten der Eingangsvariablen entsprechende Funktionsergebnisse zu ermitteln.

**Funktionsbaustein:** Dieser POE-Typ ist geeignet, wenn aus den Werten von Eingangs- und Ausgangsvariablen sowie bausteininterner Zustandsvariablen neue Ergebnisse für eine oder mehrere Ausgangsvariablen ermittelt werden sollen. Alle Werte der Ausgangs- und Zustandsvariablen bleiben von einer Bearbeitung des Funktionsbausteins bis zur folgenden erhalten. Das bedeutet, dass es bei einer erneuten Bearbeitung des Funktionsbausteins mit den gleichen Werten der Eingangsvariablen zu anderen Ausgangsergebnissen kommen kann. Anschaulich spricht man hier von einem Baustein Typ mit Gedächtnis. Um die Fähigkeiten eines Funktionsbausteins in einem Programm auch mehrfach nutzen zu können, ist die sog. *Instanziierung* der Funktionsbausteine erforderlich, worunter man das Erzeugen einer Kopie (Instanz) des Bausteines versteht. Jede Instanz muss mit einem eigenen Namen versehen werden. Unter dem Instanznamen werden die jeweils letztgültigen Variablenwerte auf entsprechenden Speicherplätzen verwaltet, während das Bausteinprogramm im Original verbleibt und dort den Instanzen bei deren Ausführung zur Verfügung steht. Die SPS-Norm schlägt viele Standardfunktionsbausteine vor, die in SPS-Systemen verfügbar sein sollten. Spezielle Funktionsbausteine können vom Anwender selbst erzeugt werden. Auch sie unterliegen bei ihrer Anwendung der Instanzenbildung.

**Programm:** Dieser POE-Typ bildet die oberste Hierarchieebene der Programmorganisationseinheiten. Die SPS-Norm sieht vor, dass Programme (P) in Ressourcen (SPS-Systeme) auch instanziiert werden können. Einige SPS-Systeme verwenden den Baustein Typ Programm (P) als alleiniges Hauptprogramm zur Organisation des Anwenderprogramms, das aus Funktionen (FC) und Funktionsbausteinen (FB) besteht. Der Programminhalt eines solchen (Haupt-)Programms besteht dann nur aus Aufrufen der Funktionen und Funktionsbausteine und um deren Eingangs-/Ausgangs-Variablen mit realen SPS-Ein-/Ausgängen zu verbinden.

### 3.2 Deklaration von FB- und FC-Bausteinen

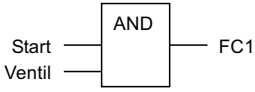

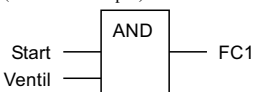
Anwenderprogramme bestehen immer auch aus sog. abgeleiteten Funktionen (FC) und/oder Funktionsbausteinen (FB), die erst durch Deklaration und Programmierung erzeugt werden müssen. Dabei sind Vorschriften zu beachten. Die Deklaration bezieht sich auf die Festlegung des Bausteintyps und auf die Bildung der Außenschnittstelle mit ihren Eingangs- und Ausgangsvariablen (-parametern) sowie der außen nicht erkennbaren bausteininternen Zustandsvariablen bei Funktionsbausteinen. Die Programmierung bezieht sich auf den Bausteinrumpf, der die Steuerungslogik enthalten muss. Deklaration und Programmierung kann in Textform oder in Grafik erfolgen.

#### • Deklaration einer Funktion mit dem Funktionsnamen FC 1

Bei der Deklaration in Textform sind folgende Elemente zu verwenden:

- das einleitende Schlüsselwort **FUNCTION** gefolgt vom Funktionsnamen, einem Doppelpunkt und dem Datentyp des Funktionswertes,
- das Konstrukt **VAR\_INPUT...END VAR**, mit dem die Namen und Datentypen der Eingangsvariablen der Funktion festgelegt werden,
- das Konstrukt **VAR\_OUT...END VAR** für die Namen und Datentypen von Ausgangsvariablen,
- das Konstrukt **VAR\_IN\_OUT...END VAR**, das die Namen und Datentypen von Durchgangsvariablen der Funktion festlegt,
- falls erforderlich das Konstrukt **VAR...END VAR**, mit dem die Namen und Datentypen von internen temporären Hilfsvariablen festgelegt werden können, deren Daten jedoch bei Beendigung der Funktion verloren gehen,
- einem Funktionsrumpf mit dem auszuführenden Programm,
- das abschließende Schlüsselwort **END\_FUNCTION**.

■ **Beispiel 1:** Deklaration einer Funktion FC

Allgemein	Ausführung in Textform	Ausführung in Grafik
<b>FUNCTION FC1 : BOOL ;</b> (*Außenschnittstelle*) <b>VAR_INPUT</b> Bezeichner1 : Datentyp ; Bezeichner2 : Datentyp ; <b>END_VAR</b> (*Funktionsrumpf*) Programm <b>END_FUNCTION</b>	<b>FUNCTION FC1 : BOOL ;</b> (*Außenschnittstelle*) <b>VAR_INPUT</b> Start : BOOL ; Ventil : BOOL ; <b>END_VAR</b> (*Funktionsrumpf*)  <b>END_FUNCTION</b>	<b>FUNCTION FC1 : BOOL ;</b> (*Außenschnittstelle*)  (*Funktionsrumpf*)  <b>END_FUNCTION</b>

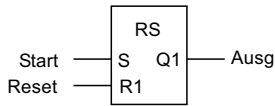

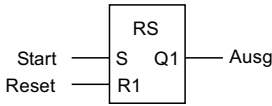
• **Deklaration eines Funktionsbausteins mit dem Namen FB 1:**

Die Deklaration eines Anwender-Funktionsbausteins in Textform erfolgt in ähnlicher Weise wie bei der Anwender-Funktion, dabei sind folgende Elemente zu verwenden:

- das einleitende Schlüsselwort **FUNCTION\_BLOCK** gefolgt vom Funktionsbausteinnamen ohne einen Datentyp,
- das Konstrukt **VAR\_INPUT...END VAR**, mit dem die Namen und Datentypen der Eingangsvariablen des Funktionsbausteins festgelegt werden,
- das Konstrukt **VAR\_OUTPUT...END VAR**, mit dem die Namen und Datentypen der Ausgangsvariablen des Funktionsbausteins deklariert werden (mehrere Ausgangsvariablen sind zulässig),

- das Konstrukt **VAR\_IN\_OUT...END VAR**, das die Namen und Datentypen von Durchgangsvariablen des Funktionsbausteins festlegt, deren Werte innerhalb des Bausteins durch das Programm verändert werden dürfen,
- das Konstrukt **VAR...END VAR**, das die Namen und Datentypen der bausteininternen Zustandsvariablen des Funktionsbausteins festlegt. Dieses Konstrukt wird auch verwendet, um im Funktionsbaustein eine Instanz eines Standardfunktionsbausteins zu erzeugen,
- einen Funktionsbausteinrumpf mit dem auszuführenden Programm,
- das abschließende Schlüsselwort **END\_FUNCTION\_BLOCK**.

■ **Beispiel 2:** Deklaration eines Funktionsbausteins FB

Allgemein	Ausführung in Textform	Ausführung in Grafik
<b>FUNCTION_BLOCK FB 1</b> (*Außenschnittstelle*) <b>VAR_INPUT</b> Bezeichner_1 : Datentyp ; Bezeichner_2 : Datentyp ; <b>END_VAR</b> <b>VAR_OUTPUT</b> Bezeichner_3 : Datentyp ; <b>END_VAR</b> <b>VAR</b> Bezeichner_4 : Datentyp ; <b>END_VAR</b> (*Funktionsbausteinrumpf*)  Programm <b>END_FUNCTION_BLOCK</b>	<b>FUNCTION_BLOCK FB 1</b> (*Außenschnittstelle*) <b>VAR</b> Start : BOOL ; Reset : BOOL ; <b>END_VAR</b> <b>VAR_OUTPUT</b> Ausg : BOOL ; <b>END_VAR</b> <b>VAR</b> SRO_1 : RS ; <b>END_VAR</b> (*Funktionsbausteinrumpf*)  <b>END_FUNCTION_BLOCK</b>	<b>FUNCTION_BLOCK FB 1</b> (*Außenschnittstelle*)  (*Funktionsbausteinrumpf*)  <b>END_FUNCTION_BLOCK</b>

**3.3 Variablen**

In Funktionen (FC) und Funktionsbausteinen (FB) sollte nur mit symbolischen Variablen programmiert werden, um damit bibliotheksfähige Programme zu erhalten, die keine Festlegungen bezüglich der Ver-

wendung realer SPS-Eingänge/Ausgänge oder Merker enthalten. Erst auf der Ebene der Programme (P) sollte die Zuordnung der SPS-Eingänge/Ausgänge/Zähler/Zeitglieder zu den direkten (realen) Eingangs- und Ausgangsvariablen erfolgen.

Eine **Variable** ist ein mit einem Namen (Bezeichner) versehener Speicherplatz, der im Anwenderprogramm als Platzhalter für Daten fungiert, die sich zur Laufzeit des Programms ändern können. Diese Variablen können symbolisch oder direkt adressiert sein.

Unter **Daten** sollen hier Informationen aus technischen Anlagen verstanden werden, wie z.B. Messdaten über Temperaturen, Füllstände, Durchflussmengen, die verarbeitet und gespeichert werden müssen. Die Variablen sind die Mittel, um die Daten zu erfassen. Dabei wird für die Variablen ein bestimmter Datentyp festgelegt. Dieser Datentyp hängt direkt zusammen mit den auf ihn zulässigen Operationen.

**Einzelement-Variablen** enthalten einzelne Datenelemente mit einfachem Datentyp, die in der Tabelle 1 auszugsweise dargestellt und bei der Deklaration der Variablen zu berücksichtigen sind.

Die Deklaration symbolischer Variablen erfolgt im dafür vorgesehenen Deklarationsteil der Bausteine unter einem Namen (Bezeichner) durch Verwendung von Schlüsselwörtern wie VAR, VAR\_INPUT, VAR\_OUTPUT, VAR\_IN\_OUT und Angabe eines Datentyps wie bei der Deklaration der Bausteine FC und FB bereits gezeigt. Der Speicherort wird durch das Betriebssystem automatisch festgelegt. Der Gültigkeitsbereich einer symbolischen Variablen ist lokal auf den Baustein beschränkt, in dem sie deklariert wurde, d.h. die Variable ist nur in diesem Baustein

bekannt. Eine Ausnahme von dieser Regel bilden symbolisch adressierte Globalvariablen, die in allen Bausteinen der SPS bekannt sind aber nur sparsam verwendet werden sollten. Bei der Deklaration von Variablen in Textform sind die in Tabelle 2 angegebenen Schlüsselwörter zu verwenden. Bei grafischer Deklaration führt das Programmiersystem die Anwendung dieser Regeln im Hintergrund aus.

**Multielement-Variablen** enthalten mehrere Datenelemente die in Feldern (Arrays) oder Strukturen zusammengefasst sind

- Feld (Array):

Ein Feld ist eine Sammlung von Datenelementen des gleichen Datentyps, die durch in eckigen Klammern [ ] angegebene Feldindizes angesprochen werden. Als Datentyp für Indizes sind z.B. INT, WORD, BYTE, BOOL zulässig. Es gibt eindimensionale und mehrdimensionale Felder.

■ **Beispiel 3:** Eindimensionales Feld

```
(*Deklaration*)
VAR
  Tabelle : ARRAY [0...3] OF BYTE :=
    16#00, 16#0F, 16#80, 16#FF
END_VAR
VAR_OUTPUT
  Wert : BYTE ;
END_VAR
(*Abfrage*)
  Wert := Tabelle [2]
```

**Tabelle 1** Elementare Datentypen

Schlüsselwort	Datentyp	Größe	Schreibweise und Wertebereiche
<b>Bit-Datentypen</b>			
BOOL	Boolesche Variablen	1 Bit	FALSE, TRUE
BYTE	Bit-Folge und 8 Bit-Hex-Zahlen	8 Bit	16# 0...FF
WORD	Bit-Folge und 16 Bit-Hex-Zahlen	16 Bit	16# 0000...FFFF
DWORD	Bit-Folge und 32 Bit-Hex-Zahlen	32 Bit	16# 0000_0000...FFFF_FFFF
CHAR	ASCII-Zeichen	8Bit	'A'
<b>Arithmetiktypen</b>			
INT	Ganze Zahlen (Festpunktzahlen)	16 Bit	-32768 bis +32767
DINT	Ganze Zahlen (Festpunktzahlen)	32 Bit	L# -2147483648 bis +2147483647
REAL	Reelle Zahlen (Gleitpunktzahlen)	32 Bit	Dezimalzahl mit Punkt: 341.7 oder Exponentialdarstellung: 3.417 E+02
<b>Zeittypen</b>			
TIME	Zeitdauer (IEC-Format)	32 Bit	TIME# -24d20h31m bis +24d20h31m
TIME OF DAY	Uhrzeit (Tageszeit)	32 Bit	TIME_OF_DAY#23:59:59,9
DATE	Datum	16 Bit	DATE#1990-01-01
Hinweis: Es existiert kein besonderer Datentyp für BCD-Zahlen (Binär Codierte Dezimalzahlen), diese sind eine Teilmenge der Hexadezimalzahlen, für die es die Datentypen BYTE, WORD und DWORD gibt.			

**Tabelle 2** Schlüsselwörter zur Deklaration von Variablen

Schlüsselwort	Bezeichnung und Gebrauch der Variablen
<b>VAR_INPUT ... END_VAR</b>	Eingangsvariable: Von außerhalb kommend, nicht innerhalb des Bausteins änderbar.
<b>VAR_OUTPUT ... END_VAR</b>	Ausgangsvariable: Nach außen lieferbar.
<b>VAR_IN_OUT ... END_VAR</b>	Durchgangsvariable: Von außen kommend, innerhalb des Bausteins änderbar, nach außen lieferbar.
<b>VAR ... END_VAR</b>	Zustandsvariable: Gebrauch nur innerhalb des Bausteines
<b>VAR_GLOBAL ... END_VAR</b>	Globalvariable: Globaler Geltungsbereich in SPS
<b>VAR RETAIN</b>	Zustandsvariable gepuffert (Remanenzverhalten).
<b>CONSTANT</b>	„Konstante Variable“, nicht veränderbar.
<b>AT</b>	Zuweisung eines direkten Speicherortes.

• **Struktur:**

Eine Struktur ist eine Sammlung von Datenelementen unterschiedlicher Datentypen. Die Datenelemente sind in einer Hierarchie geordnet, z.B. Produkt, Version, Seriennummer, Datum.

**Direkte Variablen**

In der obersten Hierarchieebene der Programmorganisationseinheiten steht der Bausteintyp Programm (P). In diesem Baustein müssen nicht nur die zum Anwenderprogramm gehörenden Funktionsbausteine (FB) und Funktionen (FC) aufgerufen, sondern auch die Verbindungen zu den SPS-Eingängen/Ausgängen hergestellt werden. Nach den Vorschriften der Norm IEC 61131-3 sind diese Adressen dem Programm (P) jedoch nicht automatisch bekannt, d.h. sie müssen erst durch Deklaration bekannt gemacht werden, dazu dienen die sog. direkten Variablen.

Bei der direkten Adressierung von Variablen wird als Variablenname der physikalische Speicherort des

Datenelements verwendet, also ein SPS-Eingang/Ausgang oder auch Merker. Zur Unterscheidung der direkten Variablen von ihrem Speicherort wird ein vorgesetztes Prozentzeichen (%) gefolgt von einem Präfix zur Kennzeichnung des Speicherortes und ein Präfix für die Speichergröße verwendet, wie in der Tabelle 3 angegeben.

■ **Beispiel 4:** Deklaration einer direkten Variablen

Allgemein	Ausführung in Textform
<b>VAR</b> AT %Operand : Datentyp ; <b>END_VAR</b>	<b>VAR</b> AT %IX4.7 : BOOL ; <b>END_VAR</b>

In einer zweiten Variante können zur Erzielung einer besseren Programmlesbarkeit Variablennamen eingeführt werden, die jedoch im Unterschied zu den richtigen symbolischen Variablen direkt mit dem physikalischen Speicherort (SPS-Eingang/Ausgang, Merker) verbunden sind (Beispiel 5).

**Tabelle 3** Präfix für Speicherort und Größe der Operanden

Präfix	Bedeutung	Beispiele für direkte Variablen
I	Speicherort Eingang	Einzel-Eingänge %IX0.7... %IX0.0
Q	Speicherort Ausgang	Einzel-Ausgänge %QX0.7...%QX0.0
M	Speicherort Merker	Eingangsbyte %IB0 = %IX0.7...%IX0.0 Ausgangsbyte %QB0 = %QX0.7...%QX0.0
X	(Einzel)-Bit-Größe	Eingangswort %IW0 = %IB0+%IB1
B	Byte-(8 Bit) Größe	Ausgangswort %QW0 = %QB0+%QB1
W	Wort-(16 Bit) Größe	Eingangsdoppelwort %ID0 = %IW0+%IW1
D	Doppelwort-(32 Bit)	

■ **Beispiel 5:** Deklaration einer direkten Variablen mit symbolischen Namen

Allgemein	Beispiel
<b>VAR</b> Bezeichner AT %Operand : Datentyp ; <b>END_VAR</b>	<b>VAR</b> Endschalter AT %IX4.7 : BOOL ; <b>END_VAR</b>

Anm.: Direkte Variablen sollten nicht in den Bausteintypen FB und FC verwendet werden, weil diese Programmteile dadurch hardware-spezifische Festlegungen enthalten würden wie beispielsweise, an welchen SPS-Eingang ein bestimmter Sensor angeschlossen wird. Das aber widerspricht der Forderung nach einer bibliotheksfähigen Gestaltung von FB- und FC-Bausteinen.

**Globale Variablen**

Globale Variablen sind solche, die durch eine entsprechende Deklaration in allen Bausteintypen innerhalb einer SPS bekannt gemacht wurden, also einen globalen Geltungsbereich haben. Der Deklarationsort ist der Baustein Programm (P).

Globale Variablen werden verwendet, um auf einfache Art einen bausteinübergreifenden Datenaustausch zu erreichen. Das kann jedoch unerwünschte Nebenwirkungen haben und die Verwendbarkeit eines Bausteins einschränken. Eine bessere Lösung besteht in der Deklaration von Übergabevariablen im Baustein Programm (P), um die entsprechenden Eingänge und Ausgänge der dort aufgerufenen Bausteine miteinander zu verbinden.

■ *Beispiel 6:* Deklaration einer Globalvariablen

```
PROGRAM PRG
(*Deklaration*)
VAR_GLOBAL
  Bezeichner : Datentyp ;
END_VAR
(*Programmrumpf*)
Programm
END_PROGRAM
```

**3.4 Programmiersprachen**

Zur Erstellung der Steuerungsprogramme mit Hilfe einer Programmiersoftware stehen gemäß DIN EN 61131-3 fünf Programmiersprachen zur Verfügung stehen: Zwei textuelle Fachsprachen (AWL, ST) und zwei grafische Fachsprachen (KOP, FBS) sowie die übergeordnete Ablaufsprache (AS) mit grafischen und textuellen Elementen.

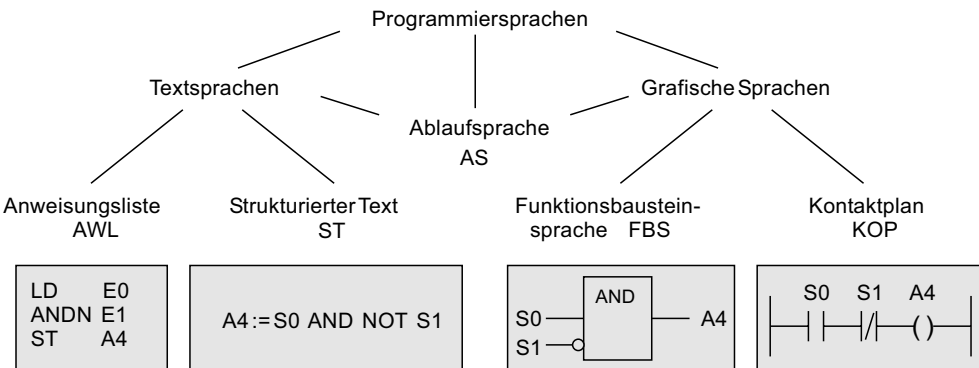
Der Programmierer kann auswählen, ob er das Bausteinprogramm in einer grafischen Sprache oder in einer Textsprache programmieren will.

- Für die grafischen Sprachen stehen in Standardbibliotheken grafische Symbole für Standardfunktionen (FC) und Standardfunktionsbausteine (FB) zur Verfügung. Diese Bausteinsymbole müssen in das Anwenderprogramm je nach Bedarf eingefügt werden, um die bezeichneten Aktionen dort ausführen zu lassen.
- Die Textsprachen benutzen anstelle von Standardbausteinen sog. Standardoperatoren, die der Programmiersprache implizit bekannt sind. Ein Operator ist ein textliches Symbol, das eine Aktion darstellt, die an einem Operanden, d.h. einer symbolischen oder direkten Variable, ausgeführt wird.

Bei der Textsprache AWL stehen die einzelnen Anweisungen in Listenform untereinander. Jede Anweisung besteht aus einem Operator, der ggf. mit einem Modifizierer ergänzt wird {N für eine Negation, C für eine Bedingung, ( für Klammer auf, ) für Klammer zu}, gefolgt von einem oder mehreren durch Kommas getrennten Operanden. Eine Anweisung kann auch durch eine Sprungmarke gekennzeichnet sein. Die Verarbeitung der Anweisungen erfolgt in einem Ergebnisregister, dieses enthält das sog. Aktuelle Ergebnis (AE), das auf folgende Weise für das Beispiel in Bild 3 ermittelt wird:

```
LD    E0  Lade den Wert des 1. Operanden (E0) in
        das Ergebnisregister
ANDN  E1  Verknüpfe das Aktuelle Ergebnis mit
        dem negierten Wert des 2. Operanden
        (E1)
ST    A4  Speichere das veränderte Aktuelle
        Ergebnis im Zieloperanden (A4)
```

Die Tabelle 4 zeigt die Standardoperatoren der Anweisungsliste (AWL) mit kurzen Erläuterungen, auf die nicht mehr weiter eingegangen wird. Die den Standardoperatoren entsprechenden Standardbausteine werden weiter unten jedoch noch näher dargestellt.



**Bild 3** Übersicht zu genormten SPS-Programmiersprachen

**Tabelle 4** Operatoren der Anweisungsliste (AWL)

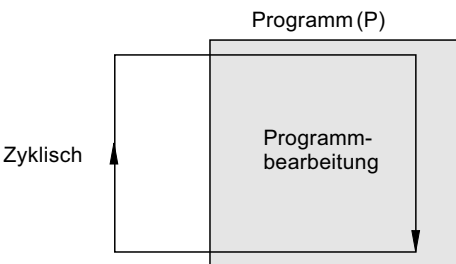
Operator	Modifizierer	Operand/Typ	Bedeutung
LD (Load)	N	Variable/Bool	Setzt das aktuelle Ergebnis (AE) dem Operanden gleich. Speichert das aktuelle Ergebnis (boolesche 1 oder 0) auf die Operandenadresse.
ST (Store)	N		
S (Set) R (Reset)			Setzt booleschen Operator auf 1 Setzt booleschen Operator auf 0 zurück
AND OR XOR )	N, ( N, ( N, ( )		Boolesches UND Boolesches ODER Boolesches Exklusiv-ODER Bearbeitet die eingeklammerte Operation
CAL (Call) JMP	C, N C, N	Instanzname Marke	Aufruf eines Funktionsbausteins Sprung zur Marke
GT GE EQ NE LE LT	( ( ( ( ( (		Vergleich auf größer als, > Vergleich auf größer gleich, >= Vergleich auf ist gleich, = Vergleich auf ungleich, <> Vergleich auf kleiner gleich, <= Vergleich auf kleiner als, <
ADD SUB MUL DIV	( ( ( (		Addition Subtraktion Multiplikation Division

### 4 Programmstrukturen

Unter einer Programmstruktur versteht man den Aufbau eines Anwenderprogramms aus Codebausteinen. Es können drei Strukturen unterschieden werden.

#### 4.1 Lineares Programm

Das gesamte Programm befindet sich in dem zyklisch bearbeiteten Programm (P). Die CPU arbeitet die Anweisungen der Reihe nach ab und beginnt dann wieder von vorne, wie in Bild 4 gezeigt.

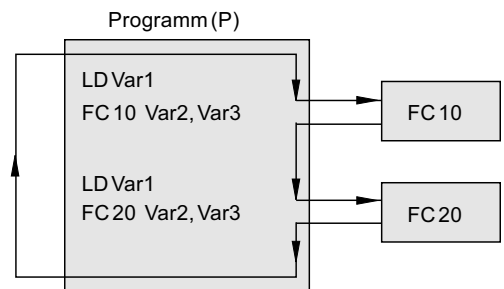


**Bild 4** Lineares Programm

#### 4.2 Gegliedertes Programm

Das Programm ist in mehrere Bausteine aufgeteilt, wobei jeder Baustein nur das Programm einer Teil-

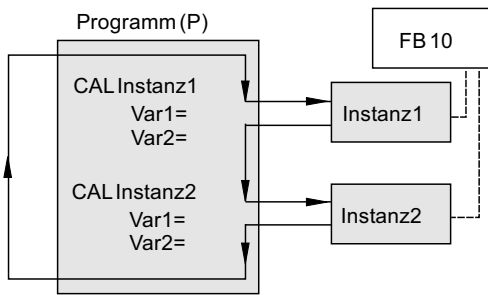
aufgabe enthält. Die Programmorganisationseinheit Programm (P) enthält die Aufruf-Anweisungen, nach deren Reihenfolge die einzelnen Bausteine bearbeitet werden. Gegenüber dem linearen Programm besteht der Vorteil in den besser überschaubaren kleinen Einheiten. Bild 5 lässt die Gliederungsstruktur und den Aufruf der beiden Funktionen mit der erforderlichen Parameterübergabe erkennen.



**Bild 5** Das Steuerungsprogramm besteht aus zwei unterschiedlichen Programmteilen

#### 4.3 Parametrierbares Programm

Ein Unterprogramm ist parametrierbar, wenn es seine Eingangs- und Ausgangsvariablen dem Hauptprogramm bei jedem Aufruf erneut zur Verfügung stellt und sein Programm für jeden Aufruf erneut ausführt,



**Bild 6** Das Steuerungsprogramm besteht aus zwei Instanzen eines Programmteils

um so aufrufspezifische Ergebnisse zu erzielen. Im Bild 6 ist die Parametrierbarkeit am Beispiel eines Funktionsbausteins (FB) dargestellt. Grundsätzlich ist dies auch bei Funktion (FC) möglich, jedoch mit der Einschränkung der nicht gegebenen Gedächtnisfähigkeit dieses Bausteintyps.

### 5 Eingabe- und Ausgabesignale

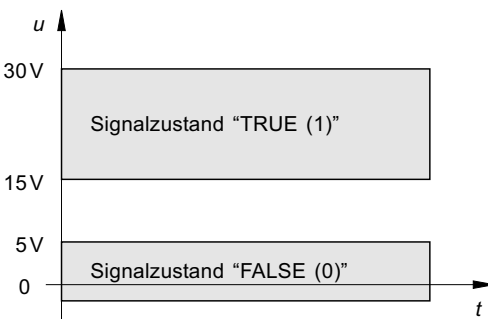
In den technischen Prozessen treten physikalische Größen wie Temperaturen, Drucke, Durchflüsse etc. auf. Automatisierungsgeräte können in der Regel nur elektrische Signale erkennen und ausgeben. Wo erforderlich, muss also eine Signalumwandlung erfolgen. Man unterscheidet verschiedene Signalarten.

#### 5.1 Binäre Signale

Ein binäres Signal ist ein 1 Bit-Signal, das nur einen von zwei möglichen Signalzuständen annehmen kann. Ein typischer Binärsignal-Geber ist ein Schalter.

Ein Signal heißt binär, wenn es nur zweier Werte fähig ist: (TRUE = 1, FALSE = 0). Die SPS-Hersteller haben für ihre Steuerungskomponenten ein Toleranzschema festgelegt, das den Wertebereich konkreter Spannungen den binären Signalzuständen zuordnet, die von den Geräten verarbeitet werden.

Die Automatisierungsgeräte können nicht den Schaltzustand von angeschlossenen Schaltern, sondern nur



**Bild 7** Signalzustände und Spannungspegel

anliegende Signale erkennen, d.h. die unterschiedliche Wirkung von Öffner- und Schließerkontakten in Anlagen muss bei der Programmerstellung bedacht werden.

Offene (unbeschaltete) Steuerungseingänge erzeugen Signalzustand „0“.

#### 5.2 Digitale Signale

Ein digitales Signal ist eine mehrstellige Bitkette, die durch Codierung eine festgelegte Bedeutung erhält, z.B. als Zahlenwert. Ein typischer Digitalisignal-Geber ist ein Zifferneinsteller. Um z.B. die Zahlen 0 bis 9 darstellen zu können, sind vier Binärstellen erforderlich.

Dezimalzahl	Dualzahl				Wert
	8	4	2	1	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	

1 Binärstelle = 1 Bit

1 Byte = 8 Bit

1 Wort = 2 Byte = 16 Bit

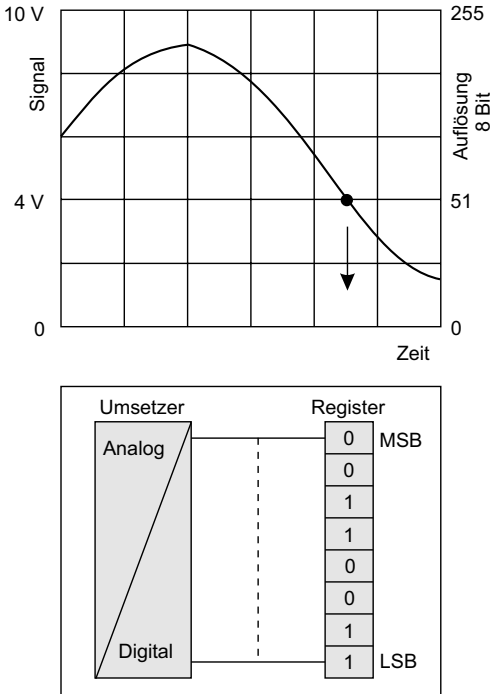
1 Doppelwort = 32 Bit

#### 5.3 Analoge Signale

Für ein analoges Signal ist charakteristisch, dass der Signalparameter (z.B. die Spannung) innerhalb bestimmter Grenzen jeden beliebigen Wert annehmen kann. Automatisierungsgeräte können intern keine analogen Signale verarbeiten. So genannte Analogbaugruppen nehmen eine Signalumsetzung vor und wandeln ein analoges Signal in ein digitales Signal um bzw. auch umgekehrt.

### 6 Eingabe-/Ausgabebaugruppen

Die im Bild 1 angedeuteten Eingabe- und Ausgabebaugruppen der SPS werden üblicherweise als zentrale digitale Eingabe- und Ausgabebaugruppen bezeichnet im Gegensatz zu den dezentralen Baugruppen (Slaves), die über ein Feldbussystem angeschlossen sind. Die digitalen Eingabe- und Ausgabebaugruppen umfassen meistens 1 Byte = 8 Bit; 2 Byte = 16 Bit; 4 Byte = 32 Bit Eingänge bzw. Ausgänge. Im Steuerungsprogramm können Bits, Bytes, Worte oder Doppelworte abgefragt oder angesteuert werden. Bei Analogbaugruppen sind entsprechend die Anzahl der Eingänge bzw. Ausgänge angegeben.



**Bild 8** Ein Spannungswert wird in eine Zahl umgesetzt (vereinfacht)

*Digitaleingabebaugruppen* gibt es für DC 24 V und AC 120/230 V mit Potenzialtrennung über Optokoppler sowie Anzeige des aktuellen Signalzustandes durch Leuchtdioden. Aufgrund von Filtermaßnahmen gegen Störsignale liegt die Frequenzobergrenze für Eingangssignale bei etwa 50 Hz. Die Digitaleingabebaugruppen formen die Pegel der externen digitalen Signale aus dem Prozess in den internen Signalpegel des SPS-Systems um. Die Baugruppen sind z.B. geeignet für den Anschluss von Schaltern und 2-Draht-Näherungsschaltern (BERO).

*Digitalausgabebaugruppen* gibt es für Lastspannungen DC 24 V oder AC 120/230 V bei spezifizierter Strombelastbarkeit und Potenzialtrennung mittels Optokoppler. Die Schaltfrequenz der Ausgänge wird nach ohmscher Last, induktiver Last und Lampenlast unterschieden und liegt im Bereich bis 100 Hz. Die Digitalausgabebaugruppen formen den internen Signalpegel des SPS-Systems in die externen, für den Prozess benötigten Signalpegel um. Die Baugruppen sind z.B. geeignet für den Anschluss von Magnetventilen, Schützen, Kleinmotoren, Lampen und Motorstartern.

*Analogeingabebaugruppen* wandeln analoge Signale aus dem Prozess in digitale Signale für die interne Verarbeitung innerhalb der SPS um. Es können Spannungs- und Stromgeber, Thermolemente, Widerstände und Widerstandsthermometer angeschlossen werden:

Spannung	z.B. $\pm 10$ V
Strom	z.B. 4 bis 20 mA
Widerstand	z.B. 0 ... 300 Ohm
Thermolement	z.B. Typ E, N, K mit Kennlinien – Linearisierung
Widerstandsthermometer	z.B. Pt 100-Standard mit Kennlinien-Linearisierung

Die Baugruppen verfügen über eine parametrierbare Auflösung von z.B. 9 bis 15 Bit + Vorzeichen, unterschiedliche Messbereiche (einstellbar durch Messbereichsmodule und Software) sowie Alarmfähigkeit (Diagnose und Grenzwertalarne an die CPU).

*Analogausgabebaugruppen* wandeln digitale Signale aus der SPS in analoge Signale für den Prozess um und sind für den Anschluss analoger Aktoren geeignet. Als Ausgangsbereiche werden angeboten:

Spannungsausgang	z.B. $\pm 10$ V
Stromausgang	z.B. 0 bis 20 mA

Die Baugruppen haben eine Auflösung von 12 bis 15 Bit. Es sind unterschiedliche Messbereiche je Kanal einstellbar.

## 7 Verknüpfungssteuerungen

SPS-Programme werden mit einem IEC 61131-3 orientierten Projektierungssystem (z.B. CoDeSys) als Projekte angelegt, in Bausteine programmiert und unter einem Dateinamen abgespeichert.

Als Verknüpfungssteuerungen bezeichnet man solche Programme, die Ausgangssignale überwiegend unter Verwendung einfacher Logikbeziehungen, Speicherfunktionen, Zeitglieder, Zähler u.a. in zyklischer Bearbeitungsweise erzeugen. Damit stellen Verknüpfungssteuerungen den größten Teil der SPS-Programme dar. Neben den Verknüpfungssteuerungen gibt es noch die Programmart der Ablaufsteuerungen.

### 7.1 Logische Grundverknüpfungen in verschiedenen Darstellungen

Nachfolgend wird eine Auswahl wichtiger Programmierfunktionen zur Realisierung von Verknüpfungssteuerungen gezeigt. In Bild 9 werden die logischen Grundverknüpfungen UND, ODER, NEGATION gezeigt. Die in diesen Beispielen vorkommenden Eingangs-/Ausgangsbezeichnungen wie E1, E2, A4 oder auch S1 usw. sind als kurze Namen von deklarerter Variablen mit zutreffenden Datentyp in entsprechenden Bausteinen zu betrachten und nicht zu verwechseln mit bekannten SPS-Operanden.

### 7.2 Zusammengesetzte logische Grundverknüpfungen

Zu den logischen Grundverknüpfungen zählen auch die beiden häufig vorkommenden Strukturen UND-

Funktion	Zeitdiagramm	FBS	KOP	AWL
<b>UND</b> $A4 = E1 \wedge E2$ $A4 = E1 \& E2$ $A4 = E1 E2$				LD E1 AND E2 ST A4
<b>ODER</b> $A4 = E1 \vee E2$				LD E1 OR E2 ST A4
<b>NICHT</b> $A4 = \bar{E1}$				LDN E1 ST A4
<b>Ausgangs-NEGATION</b> $A4 = \bar{E1} \wedge E2$				LD E1 AND E2 STN A4

**Bild 9** Logischen Grundverknüpfungen nach IEC 61131-3

vor-ODER sowie ODER-vor-UND, die in Bild 10 in grafischer Form (FBS) und Textform (AWL) dargestellt sind. Die zugehörigen Schaltfunktionen heißen

Disjunktive Normalform für die UND-vor-ODER-Struktur und Konjunktive Normalform für die ODER-vor-UND-Struktur.

<b>UND-vor-ODER-Verknüpfung</b> <b>1) Allgemeiner Fall</b> siehe nachfolgend bei DNF $A = E1 E2 \bar{E3} \vee E1 \bar{E2} \vee E3$		LD E1 AND E2 ANDN E3 OR( AND E1 ANDN E2 ) OR E3 ST A4
<b>2) Spezieller Fall:</b> <b>Antivalenz (Exklusiv-ODER)</b> $A = E1 \leftrightarrow E2$		LD E1 XOR E2 ST A5
<b>ODER-vor-UND-Verknüpfung</b> $A = (E1 \vee E2) \wedge (\bar{E1} \vee \bar{E2}) \wedge E3$		LD E1 OR E2 AND( ORN E1 ORN E2 ) AND E3 ST A4

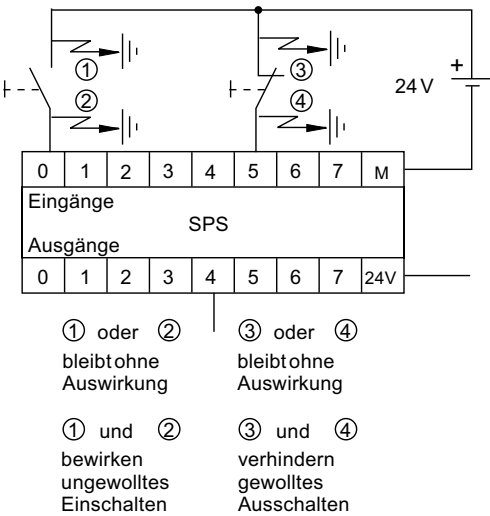
**Bild 10** Zusammengesetzten logischen Grundverknüpfungen nach IEC 61131-3

### 7.3 Schließer- und Öffnerkontakte, Drahtbruchsicherheit, Erdschlussgefahr

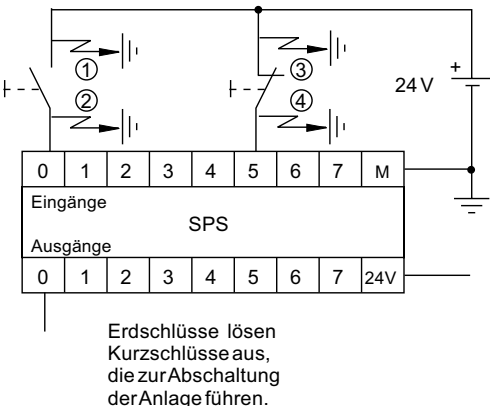
Befehlsgeber können auf einer Schließer- oder Öffnerfunktion beruhen, d.h. bei Betätigung ein 1-Signal oder ein 0-Signal an den Steuerungseingang liefern.

Ein Startbefehl für eine Steuerung ist mit einem 1-Signal, ein Haltbefehl mit einem 0-Signal auszuführen. Bei Gleichzeitigkeit muss der Haltbefehl Vorrang haben.

Das Einschalten einer Steuerung durch einen Schließerkontakt (Arbeitsstromprinzip) und das Ausschalten mit einem Öffnerkontakt (Ruhestromprinzip) macht die Steuerung drahtbruchsicher. Bei Auftreten eines Drahtbruchs erfolgt kein unbeabsichtigtes Einschalten der Steuerung, jedoch wird eine eingeschaltete Steuerung abgeschaltet. Die Erdschlussgefahr erfordert geerdete Steuerkreise oder Isolationsüberwachung.



**Bild 11** Auswirkungen von Erdschlüssen im nicht geerdeten Steuerkreis

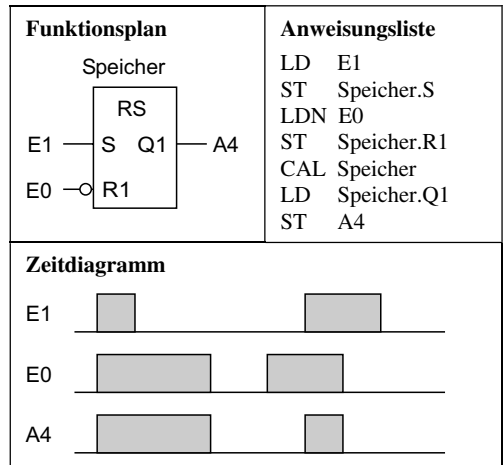


**Bild 12** Auswirkung von Erdschlüssen im geerdeten Steuerkreis

### 7.4 Speicherfunktionen

Viele Steuerungsaufgabe erfordern die Verwendung von Speicherfunktionen. Eine Speicherfunktion liegt dann vor, wenn ein kurzzeitig auftretender Signalzustand über den Programmzyklus hinaus festgehalten, d.h. gespeichert und erst zu einem späteren Zeitpunkt wieder gelöscht werden muss. Die Ausführung einer Speicherfunktion umfasst das Setzen und Rücksetzen des Speichers. Für die Speicherfunktionen stehen bistabile Standardfunktionsbausteine zur Verfügung, aus denen sich Instanzen zur Verwendung im Anwenderprogramm bilden lassen. Zu unterscheiden sind zwei Speicherbausteintypen, die unterschiedlich auf gleichzeitiges Eintreffen von Setz- und Rücksetzsignalen reagieren.

#### Speichern mit vorrangigem Rücksetzen



#### Speichern mit vorrangigem Setzen

