

Steuerungen analysieren und projektieren

Der heute stetig wachsende Technisierungsgrad erfordert in immer mehr Bereichen effiziente und kostengünstige Steuerungs- bzw. Regelungseinrichtungen.

Unterschiedliche Schwierigkeitsgrade der Problemstellungen und die daraus resultierenden Anforderungen an die Steuerungen erfordern jeweils passende Lösungsstrategien aus Sicht der Komplexität und der Wirtschaftlichkeit.

Die Palette reicht von einfachen Schützsteuerungen bis zu aufwändigen SPS-Systemen.

Innerhalb der SPS-Technik bieten sich heute unterschiedliche Varianten an. Für die Informatikerinnen und Informatiker im Betrieb stellt sich die Frage, mit welcher Lösung das geforderte Ziel optimal erreicht wird.

Anhand der Vorstellung von drei signifikanten Steuerungsproblemen und deren beste Lösung mit Grundlagenübermittlung zur jeweiligen Lösungsmethode lernen Informatikerinnen und Informatiker in den Lernbereichen eins bis vier die spezifischen Vorteile jeder Methode kennen.

Alle notwendigen Informationen und Arbeitsunterlagen sind in diesem Lernmodul enthalten.

Eine Zusammenfassung steuerungstechnischer Grundlagen ist im Modul „Formeln Steuerungstechnik“ enthalten.

Steht Ihnen entsprechende Hard- und Software zur Verfügung, können Sie Teile dieses Lernmoduls am PC nachvollziehen.

Dieses Lernmodul ist im häuslichen Studium zu erarbeiten.

Der benötigte Zeitaufwand liegt bei ca. 36 Stunden.

Zusätzlich finden im Begleitunterricht 19 Stunden Festigung und Vertiefung fachspezifischer und fächerübergreifender Zusammenhänge sowie die Beschreibung von Lösungsverfahren zur Bearbeitung typischer Aufgaben und Problemstellungen statt.

In diesen 19 Stunden ist das Laborprojekt „Steuerungstechnik“ im Umfang von ca. 8 Stunden integriert.

LERNMODUL 1

Ziele

Ausgangssituation

Planung

**Projekt 1 des
Lernmoduls****Säge mit Absaugvorrichtung**

Ziel des Projektes ist die Realisierung einer Verknüpfungssteuerung für eine Säge mit Absaugvorrichtung, die folgende Funktionen erfüllen soll:

- beim Einschalten soll der Sägemotor und der Abluftventilator gleichzeitig starten
- beim Ausschalten der Anlage soll für eine einstellbare Zeit der Abluftventilator nachlaufen.

Im Lernbereich zwei werden die grundlegenden Funktionen zur Verwirklichung einer Verknüpfungssteuerung vermittelt.

**Projekt 2 des
Lernmoduls****Verladeanlage**

Ziel des Projektes ist die Realisierung einer Ablaufsteuerung für eine Verladeanlage für Schüttgüter, die folgende Funktion erfüllen soll:

- beim Einschalten soll der Wagen zuerst in die Leerstation fahren und danach den Pendelbetrieb zwischen Füll- und Leerstation aufnehmen.

Im Lernbereich drei werden die grundlegenden Strukturen, Befehle und Aktionen für eine Ablaufsteuerung dargestellt und erläutert.

**Projekt 3 des
Lernmoduls****Verkaufsautomat**

Ziel des Projektes ist die Realisierung einer Steuerung mit der Programmiersprache „Strukturierter Text“ für einen Verkaufsautomaten für Blumengebinde unterschiedlicher Preisklassen, die folgende Funktionen erfüllen soll:

- nach Einwurf eines Geldbetrages und Auswahl des gewünschten Blumenstraußes erfolgt von der Steuerung die Prüfung von
 1. Geldbetrag
 2. Wechselgeld
 3. Ware
- bei Erfüllung aller Voraussetzungen wird der gewünschte Blumenstrauß weg-optimiert ausgegeben.

Im Lernbereich vier werden die grundlegenden Befehle und Programmstrukturen für die Programmierung dargestellt und erläutert.

Inhaltsverzeichnis

1 Vergleich Schützsteuerungen-SPS/EVA-Prinzip	4
1.1 Einführung in die Steuerungstechnik	4
1.2 Schützsteuerung	9
1.3 EVA-Prinzip	23
1.4 SPS-Steuerung	25
2 Realisierung einer Steuerung mit Verknüpfungselementen (KOP, AWL, FBS) 33	
2.1 Boolesche Grundfunktionen	34
2.2 Speicher	51
2.3 Zeitverzögerungen	61
2.4 Zähler	69
3 Realisierung einer Steuerung durch eine Ablaufsprache (AS)	78
3.1 Normgerechte Darstellung von Ablaufsteuerungen	80
3.2 Programmierung von Schritten und Übergängen	86
3.3 Strukturen von Schrittketten	92
3.4 Befehle, Aktionen	100
4 Realisierung einer Steuerung mittels strukturiertem Text (ST)	125
4.1 Programmaufbau	128
4.2 Syntax	128
4.3 Programmierung von Verzweigungen	133
4.4 Programmierung von Schleifen	135
Lösungsanhang	142

Lernbereich**1 Vergleich Schützsteuerungen - SPS/EVA-Prinzip****1.1 Einführung in die Steuerungstechnik**

Produktivitätsfortschritte beruhen in ganz wesentlichem Maße auf Automatisierung, dem selbsttätigen Ablauf von Prozessen. Wesentliche Elemente der Automatisierung (der Automatisierungstechnik) sind:

- Messtechnik (Sensortechnik)
- Steuerungstechnik
- Regelungstechnik
- Antriebstechnik
- Stelltechnik (Aktorik)

Messtechnik

Physikalische Zustände, Positionen, Stückzahlen usw. werden durch Messfühler, Abtaster, Zähler usw. erfasst.

Steuerungstechnik

Der Ablauf des Steuerungsprozesses ist abhängig von einer Willensäußerung oder einem äußeren Anlass (Führungsgröße) oder einer Vorschrift (Programm). Zum Beispiel: Anlauf einer Werkzeugmaschine durch Betätigung des Starttasters oder Stoppen einer Verfahrbewegung bei Erreichen eines Grenztasters.

Regelungstechnik

Aufgabe der Regelungstechnik ist es, physikalische Größen während des Prozessgeschehens auf einen Vorgabewert (Sollwert) zu halten.

Antriebstechnik

Für den Antrieb von Arbeitsmaschinen werden Kraftmaschinen (Elektromotor, Verbrennungsmotor, Turbine) benötigt. In der Regel liefern diese Kraftmaschinen eine drehende Ausgangsbewegung, die oftmals durch geeignete mechanische Maßnahmen in eine geradlinige (lineare) Bewegungsform umzuwandeln ist. Dies ist eine wesentliche Aufgabe der Antriebstechnik. In der elektrischen Antriebstechnik sind die Kraftmaschinen ausschließlich Elektromotoren (rotierende Drehstrom-, Gleichstrom-, Einphasen-Wechselstrommotoren sowie in Sonderfällen Linearmotoren).

Raumsparende Antriebssysteme lassen sich mithilfe der Pneumatik und Hydraulik realisieren.

Stelltechnik

Die Aufgabe der Stelltechnik besteht darin, physikalische Größen durch Einwirkung auf Massenströme und/oder Energieströme zu beeinflussen (Schalter, Ventile, Widerstände).

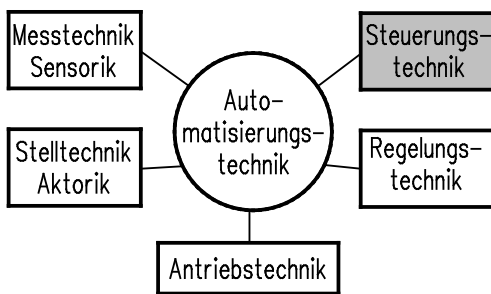


Abbildung 1 Elemente der Automatisierungstechnik

Innerhalb der Automatisierungstechnik kommt der **Steuerungstechnik** eine hohe Bedeutung zu, zumal sie Bestandteil jeder noch so kleinen Automatisierungsaufgabe ist.

Definition der Steuerung (DIN 19226)

Die Steuerung (das Steuern) ist der Vorgang in einem System, bei dem mehrere Größen als **Eingangsgrößen** andere Größen als **Ausgangsgrößen** auf Grund der dem **System eigentümlichen Gesetzmäßigkeit** beeinflussen.

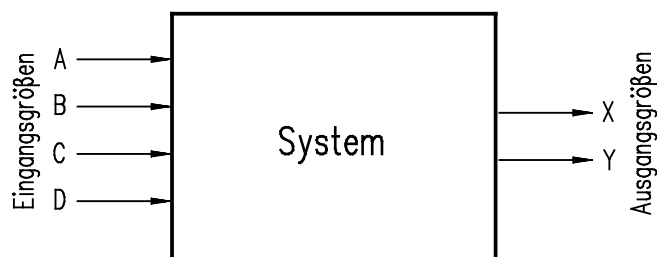


Abbildung 2 Prinzip einer Steuerung

Zu beachten ist, dass die Bezeichnung Steuerung nicht nur für den Vorgang des Steuerns, sondern ebenfalls für die Anlage verwendet wird, in der die **Steuerung** stattfindet.

Steuerungsaufgaben

1. Aufgabe: Informationssammlung mithilfe von Eingabegeräten (z.B. Schalter, Taster, Grenztaster, Messfühler).
2. Aufgabe: Registrierung und Speicherung von Daten vor Auswertung und Weiterverarbeitung. Die Registrierung kann z.B. mithilfe von elektromechanischen Speichern, Halbleiterspeichern usw. erfolgen.
3. Aufgabe: Logische Verknüpfung der Eingangssignale zwecks Bildung von Ausgangssignalen.
4. Aufgabe: Nach Bildung eines Ausgangssignals ist die entsprechende Aktion zum richtigen Zeitpunkt in der richtigen Reihenfolge auszulösen.

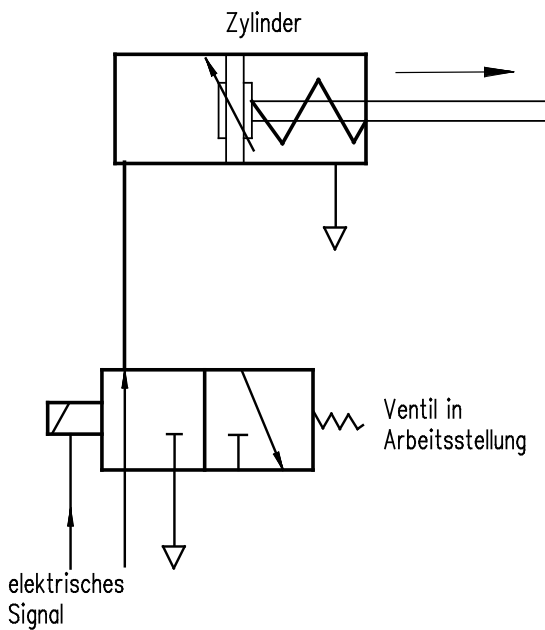


Abbildung 5 Steuerventil und Zylinder

Der Zylinder wird mit Druckluft beaufschlagt; der Kolben fährt aus. Solange ein elektrisches Signal anliegt, bleibt der Kolben (gegen die Federkraft) ausgefahren. Entfällt das **elektrische Signal**, wird das Ventil durch Federkraft in Ruhestellung zurückgebracht. Der Zylinder fährt dann wieder ein.

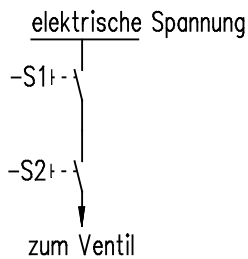


Abbildung 6 Ansteuerung des Steuerventils

Das elektrische Signal liegt an, wenn die Taster S1 und S2 (also beide Taster) betätigt werden. Deshalb wird von **Zweihandbetrieb** gesprochen.

Diese Überlegungen werden nun dazu genutzt, den **Signalflussplan** der Steuerung zu entwickeln.

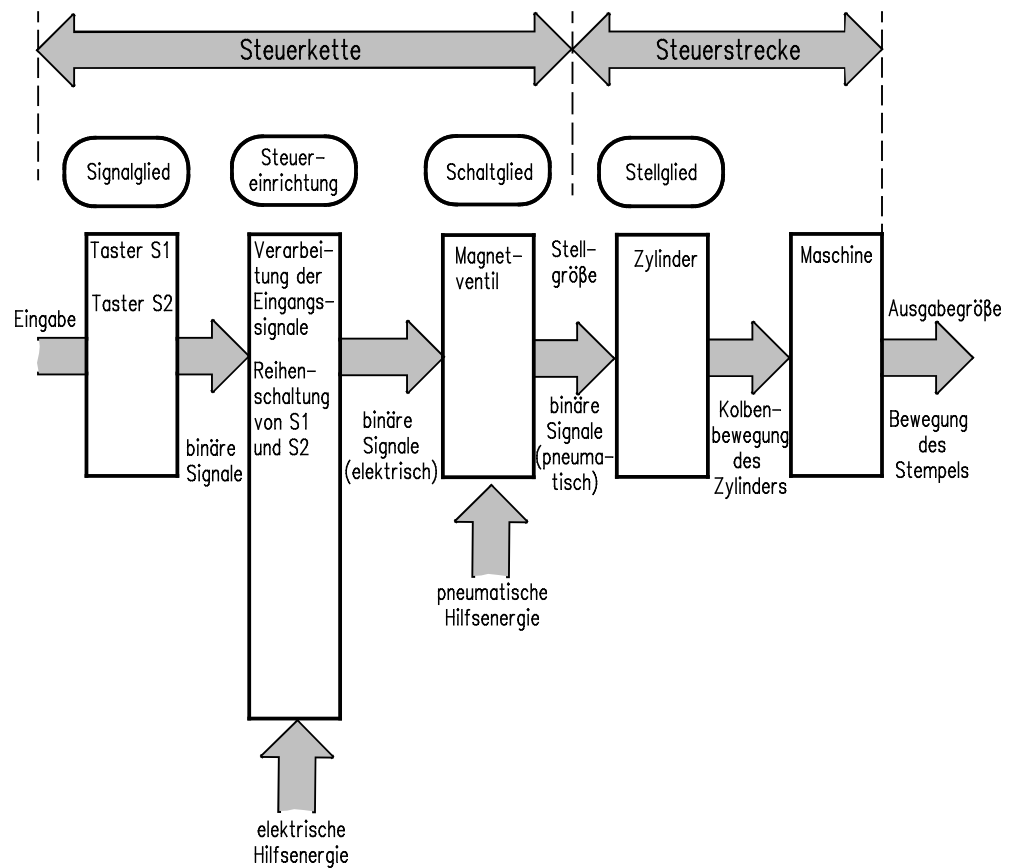


Abbildung 7 Signalflussplan der Stempelsteuerung

Kennzeichnend für eine **Steuerung** ist der **offene Wirkungsablauf** innerhalb der Steuerkette. Die Steuerkette ist **offen**, d.h. die innerhalb der Steuerkette wirksamen Signale **wirken nur in einer Richtung**. Keinesfalls auf den Anfang oder einer anderen Stelle der Kette zurück.

Steuerstrecke

Teil des Wirkungsweges einer Steuerung, der den zu beeinflussenden Teil der Anlage darstellt. Dieser Anlagenbereich führt den zu steuernden Prozess aus (Stempelvorgang).

Steuereinrichtung

Teil des Wirkungsweges einer Steuerung, durch den die Steuerstrecke mithilfe eines **Stellgliedes** beeinflusst wird.

Stellglied

Das Stellglied gibt den Energiefluss zur Steuerstrecke frei und beeinflusst die Ausgabegröße (auch Aufgabengröße genannt) der Steuerung im Sinne der Aufgabenstellung. Beeinflusst wird das Stellglied von der **Stellgröße**.

1.2 Schützsteuerung

Schütze sind elektromagnetisch betätigte Schaltgeräte. Sie haben ein **elektrisches Betätigungsglied**, werden somit mit **Hilfsenergie** betätigt. Wird die Schützspule von Strom durchflossen (Steuerstrom), zieht das Schütz an. Die Schließer schließen, die Öffner öffnen dann. Solange der Steuerstrom fließt, bleibt das Schütz angezogen.

Wird der Spulenstrom unterbrochen, fällt das Schütz wieder ab. Eine Rückstellfeder bringt die Kontakte in Ruhelage zurück: Schließer geöffnet, Öffner geschlossen.

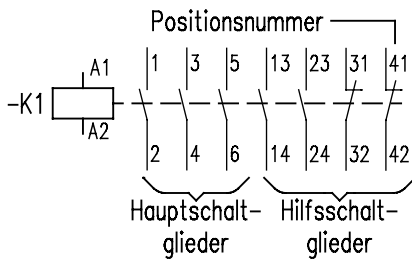


Abbildung 8 Schaltsymbol eines Schützes

Schütz

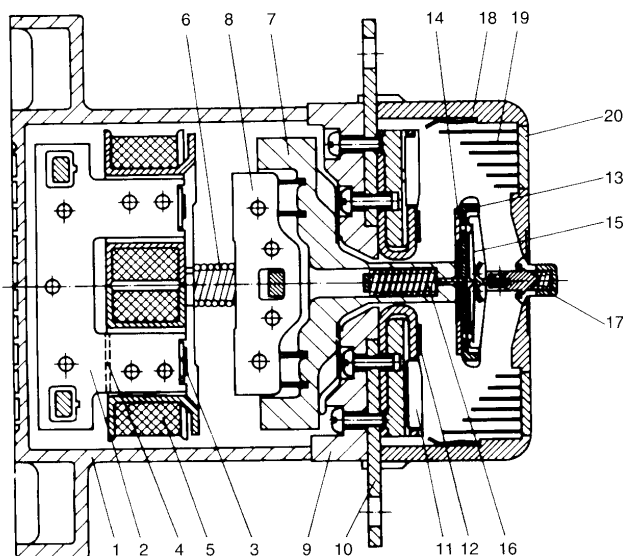
Betriebsmittelkennzeichen: K
 Kontaktbezeichnung Spule: A1, A2
 Hauptschaltglieder: 1...6 (einziffrig)
 Hilfsschaltglieder:

Öffner 1, 2
 Schließer 3, 4 (zweiffrig einschließlich Positionsnnummer)

Hauptschaltglieder können auch große Lastströme sicher schalten. Sie haben eine **Doppelunterbrechung** und eine Lichtbogenlöschkammer.

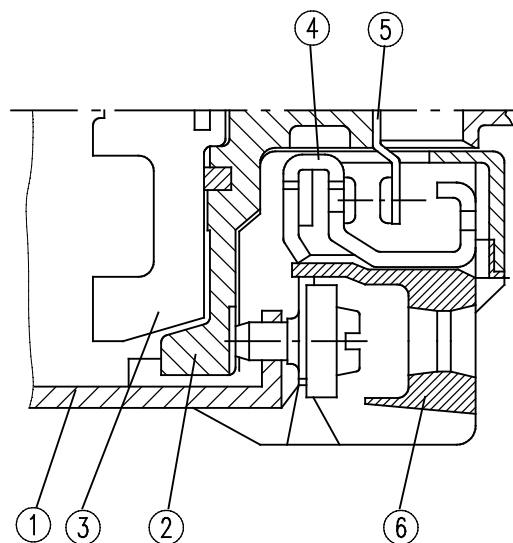
Hauptschütze (auch **Lastschütze** genannt), besitzen drei **Hauptschaltglieder** (entsprechend den drei Außenleitern des Drehstromsystems).

Hilfsschaltglieder werden für Nennströme von 2...20 A ausgelegt. Sie haben Doppelunterbrechung, jedoch keine Lichtbogenlöschkammer.



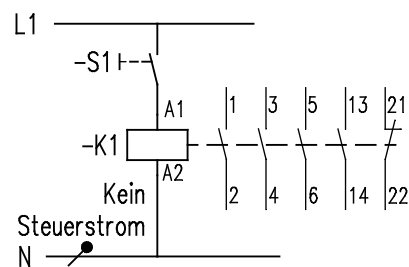
- 1 = Sockel
- 2 = Festmagnet
- 3 = Kurzschlußring
- 4 = Luftspalt
- 5 = Spule
- 6 = Rückstellfeder
- 7 = Schaltkopf
- 8 = beweglicher Magnet (Anker)
- 9 = Schaltstückträgerplatte
- 10 = Hauptanschluß
- 11 = Festschaltstück mit Laufhorn
- 12 = aktives Kontaktstück
- 13 = Schaltbrücke
- 14 = aktives Kontaktstück
- 15 = Kontaktblattfeder
- 16 = Kontaktdruckfeder
- 17 = Schaltstellungsanzeige
- 18 = Lichtbogenkammer
- 19 = Löschbleche
- 20 = Abdeckung

Abbildung 9 Schnittbild eines Drehstromschützes



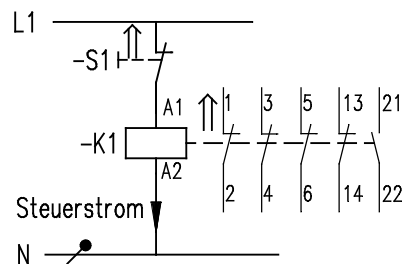
- ① untere Gehäusehalbschale
- ② Schaltkopf
- ③ beweglicher Magnet (Anker)
- ④ Festschaltstück
- ⑤ Schaltbrücke
- ⑥ Abdeckung, abnehmbar bei Schaltstückwechsel

Abbildung 10 Hauptkontakte eines Drehstromschützes



kein Steuerstrom,
Spule wird nicht erregt,
kein Magnetfeld,
Schütz zieht nicht an

Abbildung 11 Schütz abgefallen



Steuerstrom,
Spule wird erregt,
Magnetfeld,
Schütz zieht an

Abbildung 12 Schütz angezogen

Hilfsschütze haben ausschließlich Hilfsschaltglieder. **Hilfsschütze** dienen überwiegend zur Bildung logischer Verknüpfungen.

Schützspulen können entweder mit **Wechselstrom** oder mit **Gleichstrom** erregt werden. Außerdem sind unterschiedliche **Steuerspannungen** möglich.

Die **Leistungsaufnahme** hängt von der Baugröße des Schützes ab. Bei Wechselstromschützen ist zwischen **Anzugs-** und **Halteleistung** zu unterscheiden. Der große Luftspalt beim Anziehen eines Schützes bewirkt, dass die Anzugsleistung größer als die Halteleistung ist.

Auswahl von Schützen

Die Auswahl von Schützen erfolgt vorrangig nach der Art des anzuschließenden Verbrauchers. Durch Festlegung von **Gebrauchskategorien** für die häufigsten An-

wendungsfälle wird den Herstellern und Anwendern eine vereinfachte Entscheidungsgrundlage zur Verfügung gestellt.

Vorrangige **Auswahlkriterien** sind:

- Nennbetriebsspannung
- Nennbetätigungsspannung
- Gebrauchskategorie
- Verbraucherart
- Schaltstücklebensdauer
- Kurzschlusschutz

Die **Nennbetriebsspannung** beträgt in den meisten Fällen 400 V/50 Hz. Weitere wichtige Spannungen sind: 230 V, 500 V, 660 V jeweils bei 50 Hz. Drehstromschütze können auch bei Gleichstrom bis 220 V verwendet werden, wobei wegen der schwierigen Lösbedingungen i.allg. mehrere Schaltstrecken in Reihe geschaltet werden.

Bei der **Nennbetätigungsspannung** wird grundsätzlich zwischen der Wechselstrombetätigung und der Gleichstrombetätigung unterschieden.

Wechselspannung: 24, 48, 110, 127, 230 V

Gleichspannung: 24, 48, 125, 220, 250 V

Die **Gebrauchskategorie** dient zur Klassifizierung der gebräuchlichsten Anwendungsfälle. AC (alternating current, Wechselstrom) und DC (direct current, Gleichstrom). Der häufigste Fall ist AC-3, das direkte Einschalten eines Kurzschlussläufermotors (Käfigläufermotors).

Wechselstrom

AC-1 Nicht induktive oder schwach induktive Last, Widerstandsöfen

AC-2 Schleifringläufermotoren: Anlassen, Gegenstrombremsen

AC-3 Käfigläufermotoren: Anlassen, Ausschalten während des Laufes

AC-4 Käfigläufermotoren: Anlassen, Gegenstrombremsen, Tippen

Unter **Tippen** wird das einmalige oder wiederholte kurzzeitige Einschalten eines Motors verstanden, um kleine Bewegungen von Maschinen zu bewirken.

Gleichstrom

DC-1 Nicht induktive oder schwach induktive Last, Widerstandsöfen

DC-2 Nebenschlussmotoren: Anlassen, Ausschalten während des Laufes

DC-3 Nebenschlussmotoren: Anlassen, Gegenstrombremsen, Tippen

DC-4 Reihenschlussmotoren: Anlassen, Ausschalten während des Laufes

DC-5 Reihenschlussmotoren: Anlassen, Gegenstrombremsen

Verbraucherart

Bestimmte Verbrauchergruppen sind nicht in Gebrauchskategorien klassifiziert und stellen besondere Ansprüche an die Schütze; z.B. durch hohe Einschaltspitzen oder Schalten von Kondensatoren oder Beleuchtungseinrichtungen.

Schaltstücklebensdauer

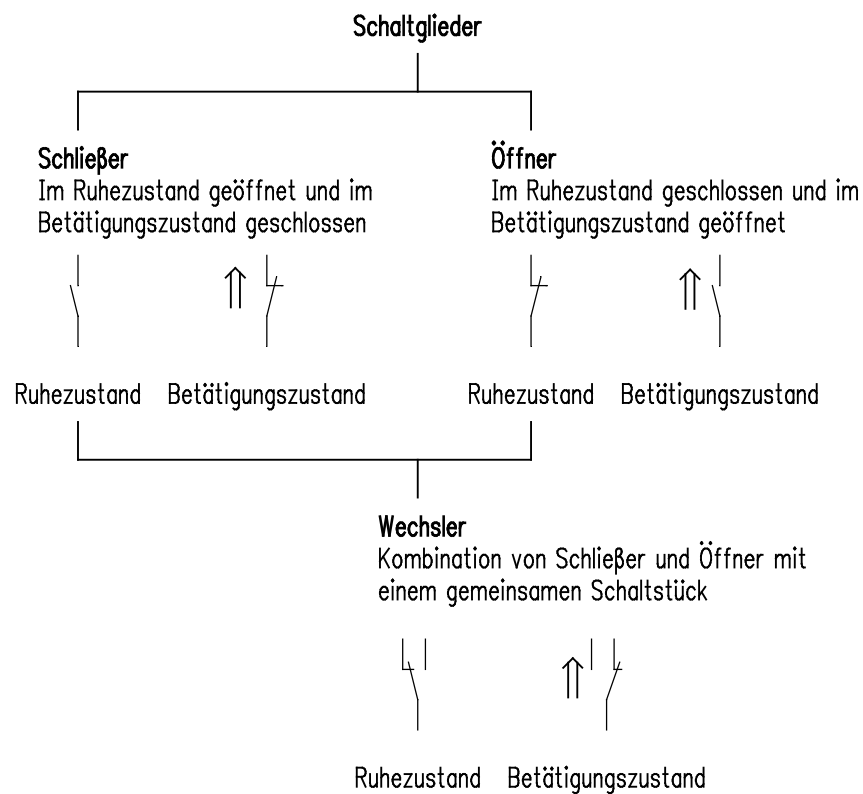
Ein Drehstromschütz nach AC-3 hat eine elektrische Lebensdauer der Schaltstücke von mindestens 1/20 der mechanischen Lebensdauer; i.allg. werden 1 Millionen **Schaltspiele** (Ein- und Ausschaltvorgang) überschritten.

Kurzschlusschutz

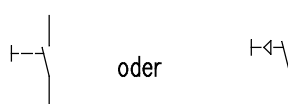
Schütze erfordern einen Kurzschlusschutz, damit keine Schäden auftreten: mit Ausnahme von leichtem Kontaktbrand und Verschweißen der Schaltstücke. Wenn letzteres auch verhindert werden soll, muss eine „Versicherung verschleißfrei“ gewählt werden. Im Übrigen leistet hier auch der Steuertransformator gute Dienste.

Schützsaltungen

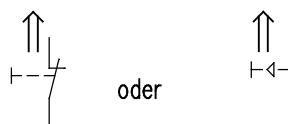
Schützsaltungen bestehen im Wesentlichen aus Schützen und Schaltgeräten. **Schaltgeräte** verfügen über Schaltglieder oder Kontakte zum Schließen oder Öffnen, sowie **Betätigungsglieder** zur Betätigung der Schaltglieder. Schaltglieder sind **Schließer**, **Öffner** und **Wechsler**.



Taster mit Handbetätigung,
allgemein
(Ruhezustand)



(Betätigungszustand)



- ┌--- Betätigung, allgemein
- ┐--- Betätigung durch Drücken
- └--- Betätigung durch Ziehen
- ┘--- Betätigung durch Drehen
- └--- Betätigung durch Kippen
- mechanische Betätigung

Abbildung 13 Darstellung von Tastern

Schalter

Bei Wegfall der Betätigungskraft bleibt der Schaltzustand EIN oder AUS erhalten.



Schalter, unbetätigt

Abbildung 14 Darstellung eines Schalters

Taster

Bei Wegfall der Betätigungskraft bleibt der Schaltzustand nicht erhalten.

Befehlstaster

Dienen zur Befehlsgabe an eine Steuerung (z.B. START, STOP). Die Farbe ihrer Betätigungseinrichtung ist festgelegt und gibt Aufschluss über die Funktion.

Das **Betriebsmittelkennzeichen** der in der Steuerungstechnik verwendeten Schalter und Taster ist S.

Grenztaster

Grenztaster haben im Wesentlichen die Aufgabe, Verbirgbewegungen zu begrenzen.

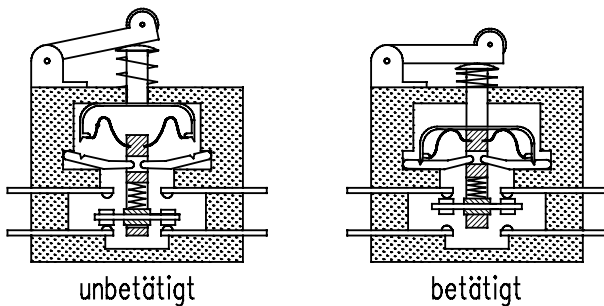
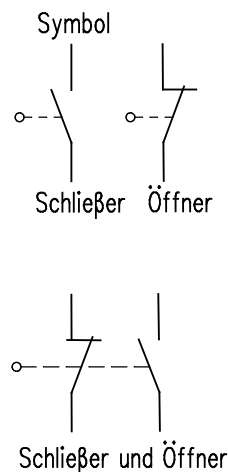


Abbildung 15 Grenztaster



Kombination von Öffner und Schließer

Beide Schaltglieder (Öffner und Schließer) werden gemeinsam mechanisch betätigt. Dabei sind zwei Möglichkeiten zu unterscheiden:

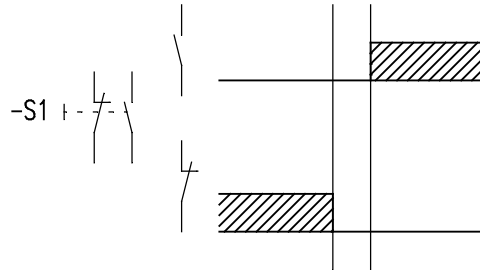


Abbildung 16 Schaltglieder nicht überdeckend

1. Die beiden Schaltglieder arbeiten nicht überdeckend. Zuerst öffnet der Öffner, dann schließt der Schließer.

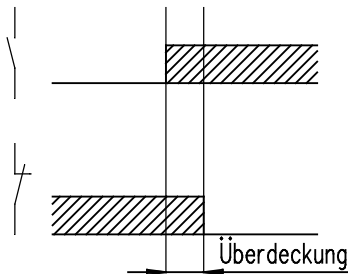


Abbildung 17 Schaltglieder überdeckend

2. Die Schaltglieder arbeiten überdeckend. Der Schließer ist schon geschlossen, bevor der Öffner unterbricht.

Schützsaltungen ohne Speicherverhalten (Tippbetrieb)

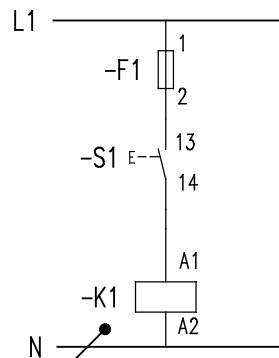


Abbildung 18 Tippbetrieb

- L1 Außenleiter
- N Neutraleiter
- F1 Steuersicherung (Anschlüsse 1, 2)
- S1 Drucktaster (Anschlüsse 13, 14)
- K1 Schütz (Anschlüsse A1, A2)

Man spricht in der Steuerungstechnik von einem **Stromlaufplan** des Steuerstromkreises in **aufgelöster Darstellung**. Die einzelnen Stromwege (Strompfade) werden **senkrecht** zwischen L1 und N angeordnet. Die Steuersicherung dient dem Leitungsschutz des Steuerstromkreises.

Wenn der Schließer S1 (Anschlussbezeichnung 13, 14) betätigt wird, ist der Stromkreis zur Schützspule K1 geschlossen; das Schütz zieht an. Solange der Spulenstrom fließt, bleibt das Schütz angezogen. Anders formuliert: Nur wenn der Taster S1 betätigt wird, zieht das Schütz K1 an. Man spricht dann von **Tippbetrieb**. Die Schützschialtung hat **kein Speicherverhalten**.

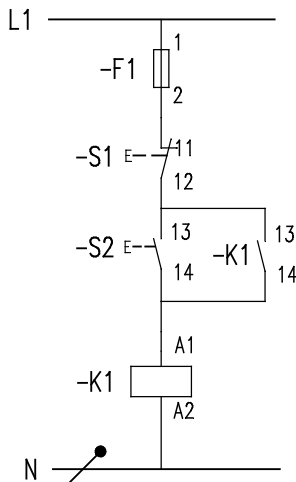
Schützschaltung mit Speicherverhalten (Selbsthaltung)

Abbildung 19 Selbsthaltung

Ein Hilfsschaltglied des Schützes K1 ist parallel zum Schließer S2 geschaltet. Wenn S2 betätigt wird, zieht das Schütz K1 an. Dadurch wird das Hilfsschaltglied (Schließer 13, 14) des Schützes K1 geschlossen. Der Schließer S2 ist dann überbrückt. Wenn er nicht mehr betätigt wird, kann der Spulenstrom weiterhin fließen. Das Schütz bleibt angezogen, es geht in **Selbsthaltung**. Die Schützschaltung hat **Speicherverhalten**. Zum Ausschalten des Schützes muss der Spulenstrom unterbrochen werden. Dies geschieht durch Betätigung des Öffners S1.

Lehrbeispiel 1

Für die Steuerung einer Umwälzpumpe soll eine Schützschaltung entwickelt werden.

Die Pumpe wird von einem Drehstrommotor angetrieben (Käfigläufermotor). Sie wird durch den Taster S2 (Schließer) eingeschaltet und durch den Taster S1 (Öffner) ausgeschaltet.

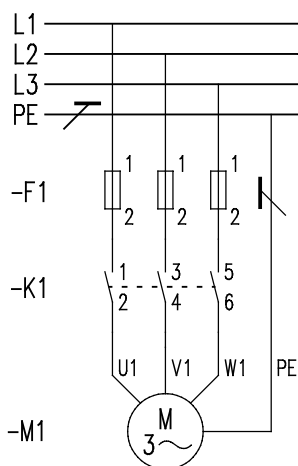
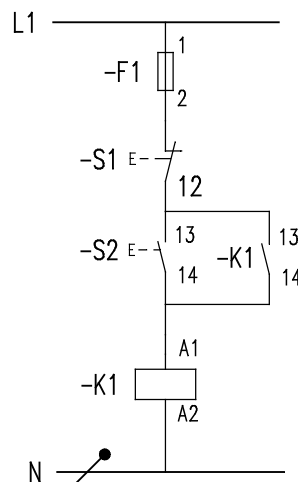
Antriebsmotor für Pumpe,
HauptstromkreisPumpe,
Steuerstromkreis

Abbildung 20 Pumpenantrieb mit Selbsthaltung

Im Hauptstromkreis schalten die Hauptschaltglieder des Schützes K1 den Antriebsmotor der Umwälzpumpe. Die Schmelzsicherungen F1 übernehmen den Leitungsschutz des Hauptstromkreises. Der Steuerstromkreis verdeutlicht die Selbsthaltung: S2 EIN, S1 AUS.

Motorschutz

Wenn beim Pumpenantrieb nach Lehrbeispiel 1 z.B. die Pumpe blockiert, wird der Antriebsmotor überlastet. Ohne weitere Maßnahmen würde der Motor zerstört werden. Solche Maßnahmen sind **Motorschutzeinrichtungen**.

Motorschutzeinrichtungen schützen den Elektromotor bei **Überlastung**, **Nichtanlauf** und **bei Ausfall eines Außenleiters**.

Motorschutzschalter können gleichzeitig als Schaltgerät und Schutzgerät eingesetzt werden. Im einfachsten Fall sind sie dreipolige Motorschalter mit **thermischer Überlastauslösung** und Handbetätigung.

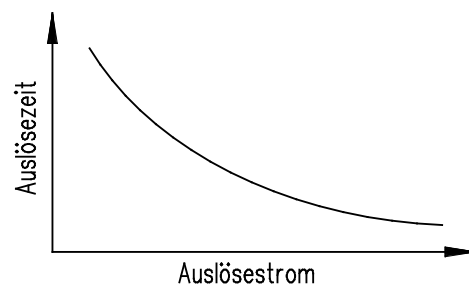
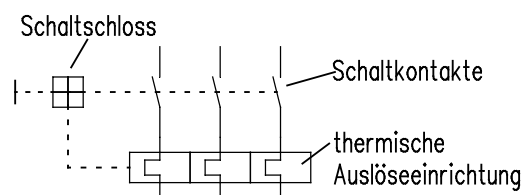


Abbildung 21 Motorschutzeinrichtung, thermisch

Motorschutzschalter mit ausschließlicher thermischer Auslösung sind nicht für den Kurzschlusschutz geeignet. Es sind dann Schmelzsicherungen für den Kurzschlusschutz vorzusehen.

Die **thermische Auslösung erfolgt** z.B. durch **Thermobimetalle**. Ein Widerstandsdraht, der vom Motorstrom durchflossen wird, ist auf einen **Bimetallstreifen** aufgebracht, der bei Stromfluss erwärmt wird. Je höher der Strom, desto größer die Erwärmung, desto stärker die Krümmung des Metallstreifens.

Bei einer bestimmten (einstellbaren) Krümmung des Bimetalls wird eine Sperre im Schaltschloss geöffnet, der Motor wird über die Schaltkontakte vom Netz getrennt. Die Bimetallauslösung ist **träge**. Bei Überlastung des Motors wird nicht sofort, sondern erst nach einer bestimmten Zeit abgeschaltet. Der relativ hohe Anlassstrom des Motors führt also nicht zum Ansprechen der Schutzeinrichtung.

Je größer die Überlastung, desto schneller löst der Motorschutz aus. Auskunft hierüber gibt die **Strom-Zeit-Kennlinie** des Motorschutzschalters.

Motorschutzschalter sind häufig zusätzlich mit einem **elektromagnetischen Kurzschlussauslöser** ausgestattet. Beim Erreichen eines fest eingestellten Ansprechwertes wird das Schaltschloss unverzüglich entklinkt. Der Ansprechwert beträgt etwa $12 \cdot I_e$, wobei I_e der höchstmögliche Einstellwert des thermischen Überlastauslösers ist.

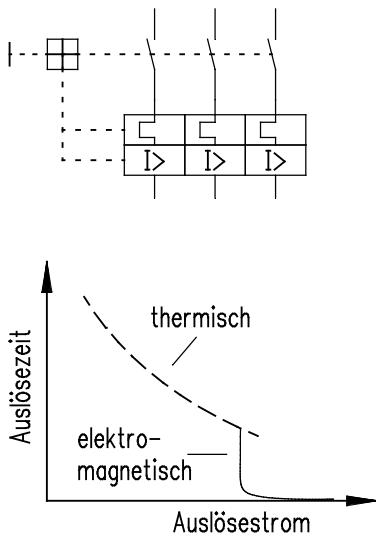


Abbildung 22 Motorschutzeinrichtung, thermisch und elektromagnetisch

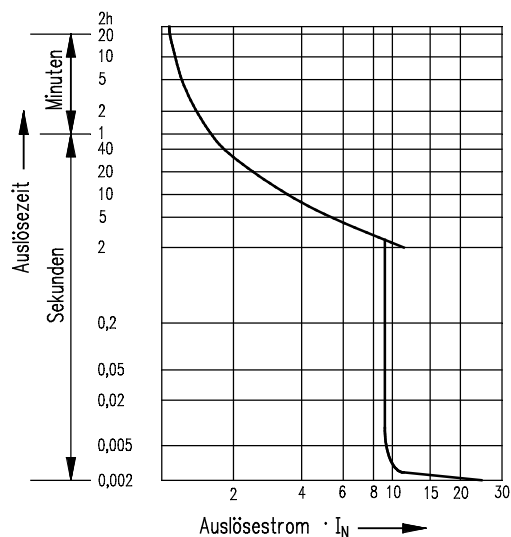


Abbildung 23 Auslösekennlinie

Wenn zum Schalten des Motors ohnehin ein Schütz zur Verfügung steht (siehe Lehrbeispiel 1), braucht die Motorschutzeinrichtung nicht zwingend Schaltkontakte zu haben. Solche Motorschutzeinrichtungen werden thermische **Überstromrelais** (sie haben keinen magnetischen Auslöser) genannt.

Thermische Überstromrelais (auch Bimetallrelais genannt), sind dreipolig ausgeführt. Sie „erkennen“ den Überlastfall. Die Trennung des Motors vom Netz erfolgt durch das Schütz.

Bei Ansprechen des Bimetallrelais wird der **Steuerstromkreis** (Spulenstromkreis) unterbrochen. Der Öffner 95-96 (Abbildung 25) des Bimetallrelais schaltet den Spulenstrom des Schützes ab.

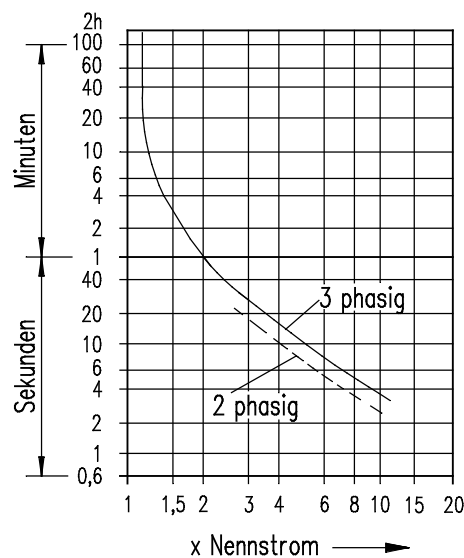
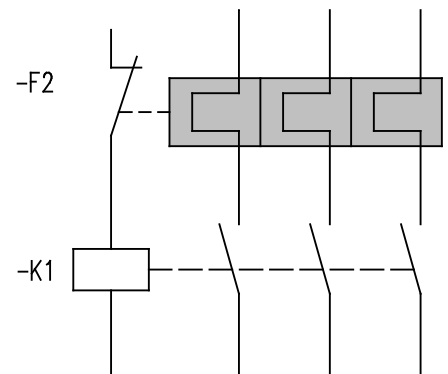


Abbildung 24 Motorschutz



Motorschutzeinrichtungen sind auf den **Nennstrom** des Motors einzustellen. Bei Motoren mit Stern-Dreieck-Anlauf auf $0,58 \cdot I_N$, wenn sich der Motorschutz in den Zuleitungen zu den Wicklungen befindet, also vom Strangstrom durchflossen wird.

Lehrbeispiel 2

Für den Pumpenantrieb nach Lehrbeispiel 1 ist ein Bimetallrelais vorzusehen, das den Motorschutz übernimmt.

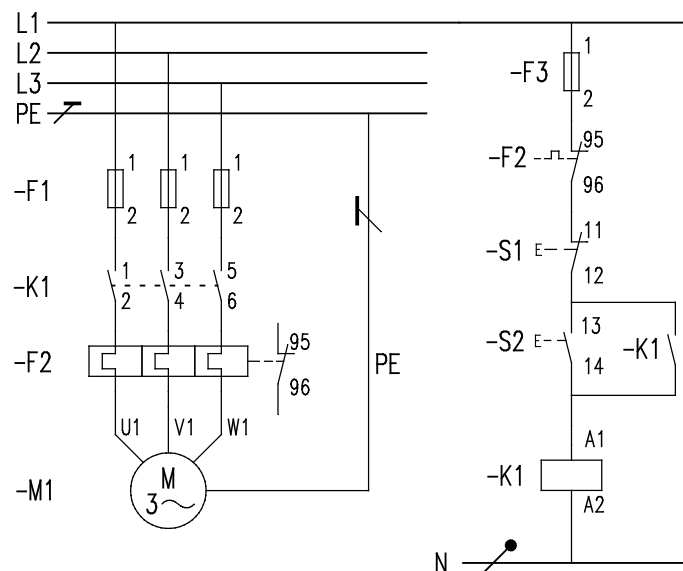


Abbildung 25 Wirkungsweise des Motorschutzes

Wenn der Motor in Betrieb ist (Schütz K1 angezogen) und der Überlastfall eintritt, macht der Öffner 95, 96 des Bimetallrelais auf. Als würde der Austaster S1 betätigt, wird dadurch der Spulenstrom unterbrochen. Das Schütz fällt ab, der Motor wird abgeschaltet.

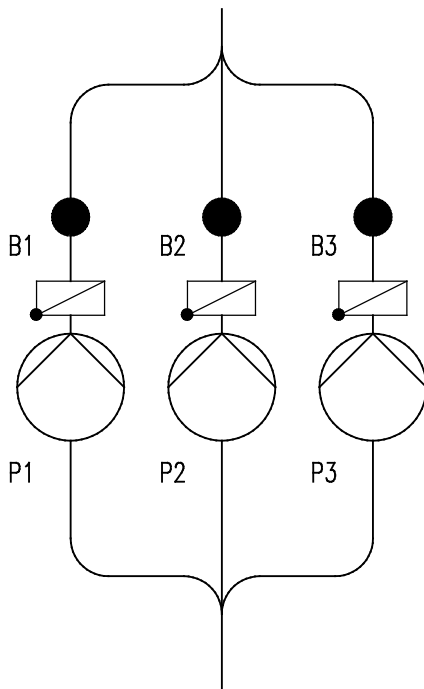
Lehrbeispiel 3

Abbildung 26 Pumpensteuerung

Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben.

Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst. Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal); wenn keine Strömung erkannt wird, somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung betätigt werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten
 S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten
 S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten
 S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Hauptstromkreis

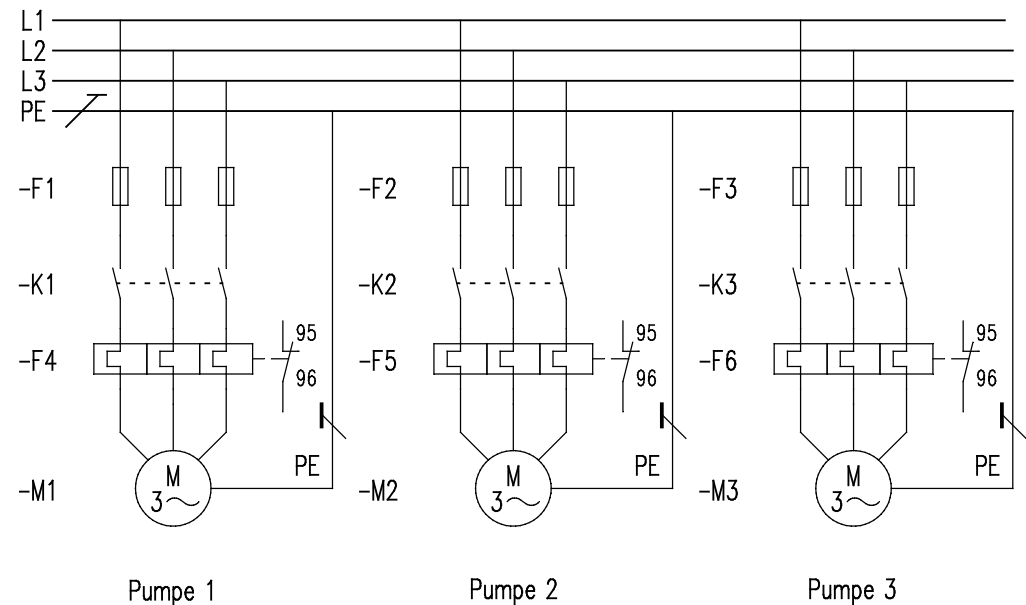


Abbildung 27 Pumpenantrieb, Hauptstromkreis

Steuerstromkreis

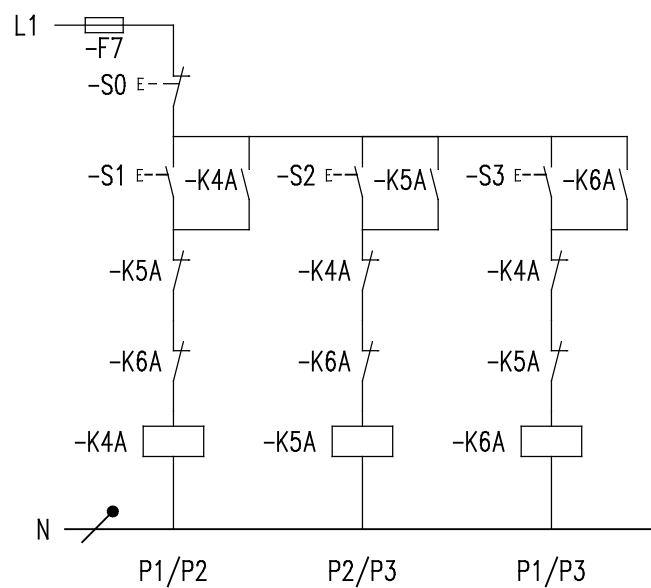


Abbildung 28 Pumpenantrieb, Steuerstromkreis

Bei den Schützen K4A, K5A, K6A handelt es sich um Hilfsschütze, was durch den Zusatzbuchstaben „A“ deutlich gemacht wird.

Wirkungsweise

Wenn S1 betätigt wird, zieht das Hilfsschütz K4A an und hält sich selbst. Der Öffner von K4A bei den Hilfsschützen K5A und K6A sorgt dafür, dass bei angezogenem Schütz K4A kein weiteres Hilfsschütz anziehen kann. Allgemein kann man sagen, dass bei dieser Schaltung immer nur ein Hilfsschütz angezogen sein kann. Die beiden anderen Hilfsschütze sind zu diesem Zeitpunkt gesperrt (verriegelt).

Damit ein anderes Hilfsschütz anziehen kann, muss zuvor mithilfe des Austasters S0 das momentan angezogene Hilfsschütz abfallen.

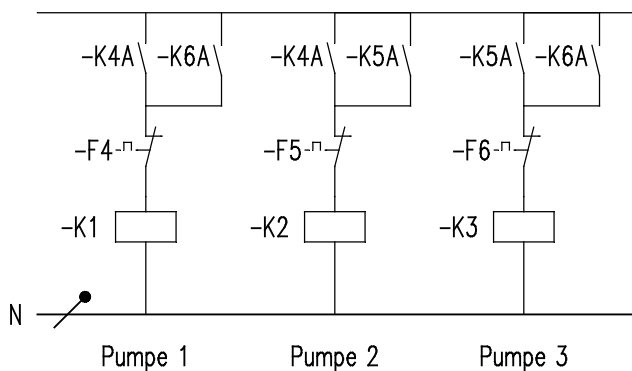


Abbildung 29 Ansteuerung der Pumpen

Dargestellt ist die Fortsetzung des Steuerstromkreises. Hier sind die Hauptschütze für die Pumpen gezeigt. Zunächst sind die thermischen Bimetallrelais F4...F6 zu beachten. Bei Überlastung des Antriebsmotors wird die jeweilige Pumpe ausgeschaltet.

Wenn das Hilfsschütz K4A anzieht, werden die Pumpen 1 und 2 eingeschaltet.
 Wenn das Hilfsschütz K5A anzieht, werden die Pumpen 2 und 3 eingeschaltet.
 Wenn das Hilfsschütz K6A anzieht, werden die Pumpen 1 und 3 eingeschaltet.

Nun ist noch die Steuerung der jeweiligen Reservepumpe zu ergänzen. Folgende Möglichkeiten sind dabei zu berücksichtigen:

K4A angezogen → P1 und P2 sollen arbeiten.
 Wenn dann
 $B1 = 1$ (P1 defekt)
 ODER
 $B2 = 1$ (P2 defekt),
 soll P3 als Reservepumpe arbeiten.

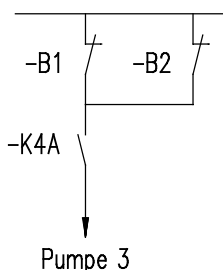


Abbildung 30 Pumpe 3 ist Reservepumpe

Wenn das Schütz K4A angezogen ist und mindestens Strömungswächter B1 oder Strömungswächter B2 keine Strömung erkennt (Öffner geschlossen, Schließer geöffnet), muss Pumpe 3 als Reservepumpe zugeschaltet werden.

K5A angezogen → P2 und P3 sollen arbeiten.
Wenn dann
B2 = 1 (P2 defekt)
ODER
B3 = 1 (P3 defekt),
soll P1 als Reservepumpe arbeiten.

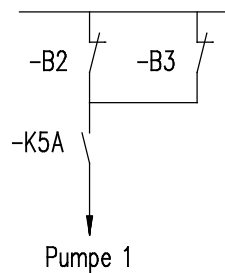


Abbildung 31 Pumpe 1 ist Reservepumpe

K6A angezogen → P1 und P3 sollen arbeiten.
Wenn dann
B1 = 1 (P1 defekt)
ODER
B3 = 1 (P3 defekt)
soll P2 als Reservepumpe arbeiten.

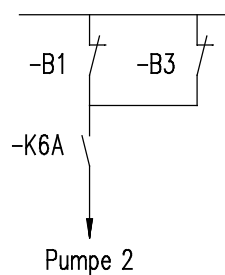


Abbildung 32 Pumpe 2 ist Reservepumpe

Diese Elemente werden nun in den Stromlaufplan für die Schütze K1...K3 einbezogen.

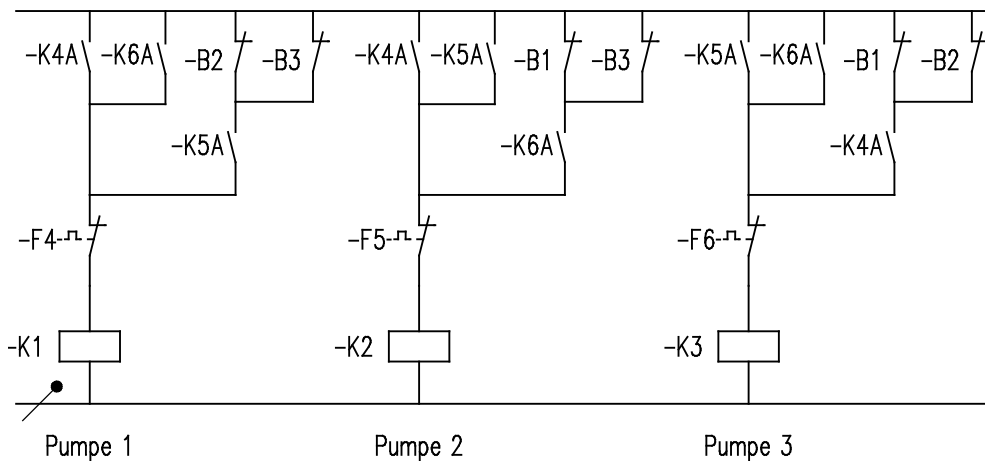


Abbildung 33 Pumpensteuerung mit Reservepumpen

Annahme:

Schütz K5A ist angezogen. Pumpe 2 und Pumpe 3 arbeiten. Wenn beide Strömungswächter (B2 und B3) eine Strömung erfassen, sind die Öffner B2 und B3 geöffnet. Ferner wird nun angenommen, dass die Pumpe 2 nicht mehr arbeitet. Der Öffner B2 des Strömungswächters schaltet dann die (Reservepumpe) Pumpe 1 zu.

Ein kleines „Problem“ soll allerdings nicht verschwiegen werden. Wenn z.B. mit S1 die Pumpen 1 und 2 eingeschaltet werden, ist in den beiden Pumpenzweigen keine Strömung vorhanden. Somit können die Strömungswächter natürlich auch keine Strömung erfassen. Folglich wird also auch die „Reservepumpe“ P3 anlaufen, bis die Strömungswächter in den Pumpenzweigen 1 und 2 Strömung erfassen. Danach arbeiten dann nur noch die beiden gewünschten Pumpen.

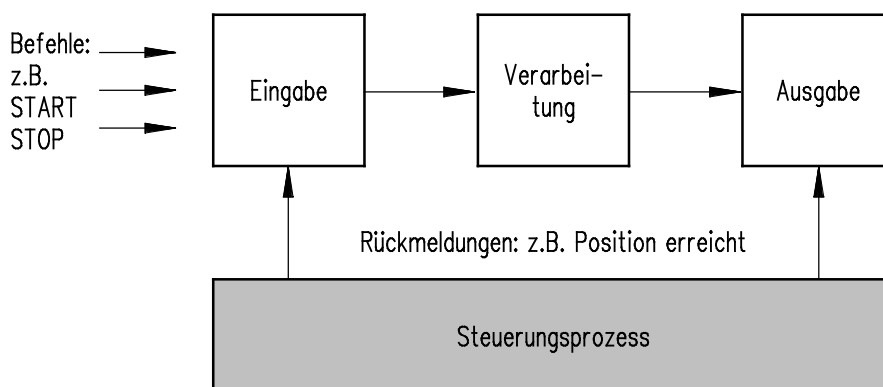
1.3 EVA-Prinzip

Abbildung 34 EVA-Prinzip

Wie alle informationsverarbeitenden Systeme arbeiten auch Steuerungen nach dem **EVA-Prinzip**.

Eingabe
Verarbeitung
Ausgabe

Die Bauelemente der **Eingabe-Ebene** sind bei allen Steuerungen praktisch identisch. Es sind dies zum Beispiel Schalter, Taster, Lichtschranken, Temperaturfühler, Strömungswächter. Diese Bauelemente liefern der Steuerung Informationen über den Fortgang des Steuerungsprozesses.

Die **Verarbeitungs-Ebene** kann ganz unterschiedlich ausgeführt sein. Möglich ist z.B. die Realisierung in Kontakttechnik (Schützschaltung), mithilfe von Logikbausteinen in fester „Verdrahtung“ oder mithilfe einer programmierbaren Logik (speicherprogrammierbare Steuerung).

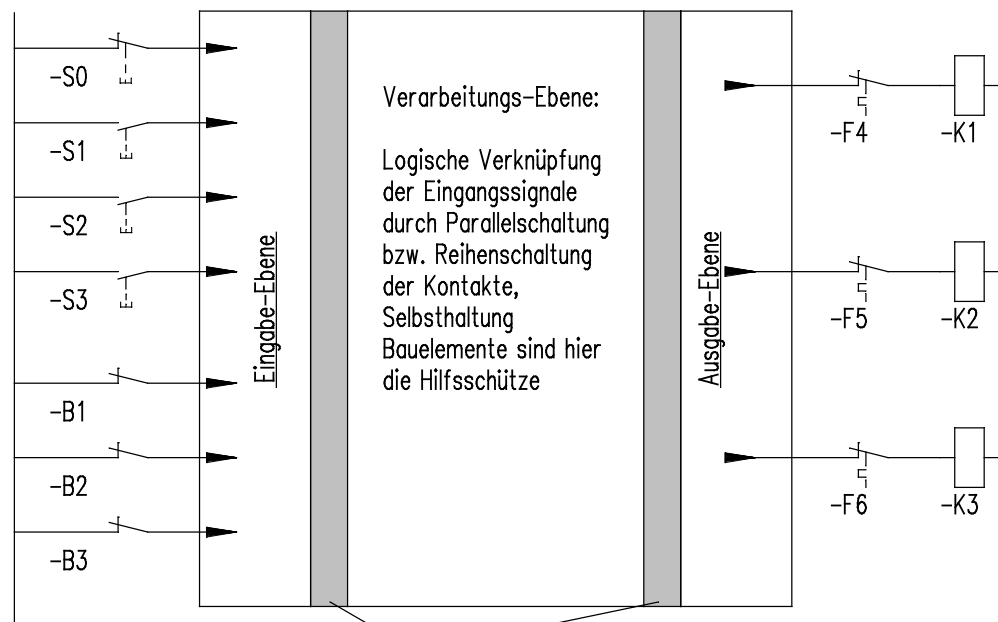
Auch die Bauelemente der **Ausgabe-Ebene** sind praktisch bei allen Steuerungen identisch. Zum Beispiel sind dies Lastschütze, Magnetventile, Relais, Meldelampen.

Zwischen der Eingabe-Ebene und der Verarbeitungs-Ebene und zwischen der Verarbeitungs-Ebene und der Ausgabe-Ebene sind u.U. **Leistungsschnittstellen** erforderlich, deren Aufgaben im Wesentlichen die **Trennung des Energieniveaus** (z.B. durch Optokoppler) und die **Pegelanpassung** (z.B. Eingabe-Ebene 24 V, Verarbeitungs-Ebene 5 V) sind.

Lehrbeispiel

Für die Heizung eines Bürogebäudes werden drei Pumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein. Die drei Pumpen werden von Drehstrom-Käfigläufermotoren angetrieben. Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst. Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung an die Steuerung. Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Stellen Sie die drei Ebenen der Pumpensteuerung dar!



Leistungsschnittstellen sind bei der Schützsteuerung nicht erforderlich, da die Spannung in allen Ebenen 230 V AC beträgt.

Abbildung 35 Anschluss der Betriebsmittel an die Steuerung

1.4 SPS-Steuerung

Die **speicherprogrammierbare Steuerung** (SPS) hat mittlerweile ab einem bestimmten Leistungsbereich die Schützsteuerung nahezu völlig verdrängt. Nur für relativ einfache Steuerungsaufgaben werden heute noch Schützsaltungen verwendet. Dies bedeutet allerdings keineswegs, dass Schütze im SPS-Zeitalter keine Bedeutung mehr haben. Als Hauptschütze (Lastschütze) übernehmen sie bei SPS-Steuerungen die Schaltung der Verbraucher.

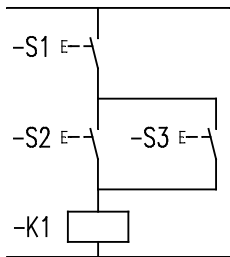


Abbildung 36 Schützsaltung

Die dargestellte Steuerung hat folgende Funktion:

Wenn der Taster S1 betätigt wird **UND** zusätzlich der Taster S2 **ODER** der Taster S3 betätigt wird, zieht das Schütz K1 an. Dies gilt natürlich auch, wenn beide Taster S2 **UND** S3 betätigt werden.

Die Steuerungsfunktion kann als logische Verknüpfung angesehen werden, die durch Verdrahtung (Verbindung der Taster und der Schützspule) gebildet wird. Es wird deshalb von einer **verbindungsprogrammierten Steuerung** (VPS) gesprochen. Jede Funktionsänderung ist mit einer Verdrahtungsänderung verbunden, was einen mehr oder weniger hohen Aufwand bedeuten kann.

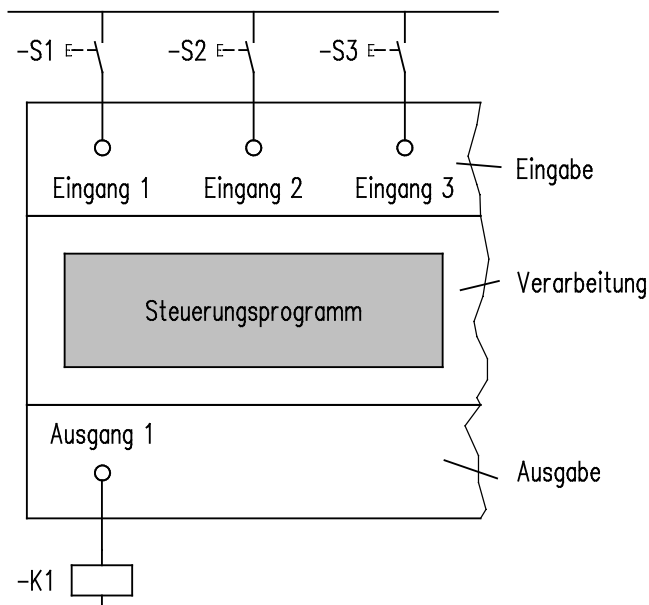


Abbildung 37 Speicherprogrammierbare Steuerung

Die Betriebsmittel Taster und Schütz erkennt man natürlich auch beim SPS-Anschluss wieder. Die Taster S1...S3 sind an die Eingänge 1...3 angeschlossen. An Ausgang 1 ist die Schützspule K1 angeschlossen.

Die bei der VPS ersichtliche Verdrahtung wird bei der SPS durch ein **Steuerungsprogramm** ersetzt. Verdrahtung wird also durch **Software** ersetzt. Das Steuerungsprogramm könnte folgendermaßen aussehen.

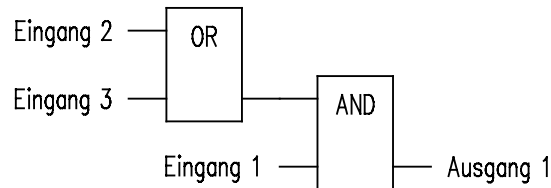


Abbildung 38 Steuerungsprogramm

Ohne Verdrahtungsänderung könnten durch Änderung des Steuerungsprogramms (Softwareänderung) zum Beispiel auch folgende Funktionen verwirklicht werden.

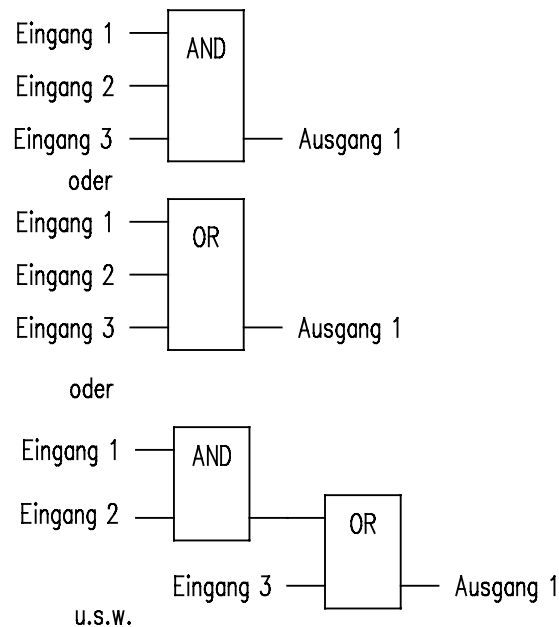


Abbildung 39 Darstellung von Steuerungsprogrammen

Die SPS ist also außerordentlich flexibel. Änderungen der Aufgabenstellung sind relativ problemlos durch **Softwareänderung** zu verwirklichen.

Solche Softwareänderungen (Programmänderungen) sind mithilfe eines Programmiergerätes bzw. eines handelsüblichen Personalcomputers möglich (entsprechende Programmiersoftware vorausgesetzt).

Neben der **Flexibilität** sind weitere Vorteile der speicherprogrammierbaren Steuerung (SPS):

- standardisierte Systeme
- geringer Raumbedarf
- Zuverlässigkeit
- einfache Dokumentation
- zeitsparende Projektierung

Zentraler Bestandteil der speicherprogrammierbaren Steuerung ist das **Automatisierungsgerät**, das im Wesentlichen aus folgenden Elementen besteht:

- Zentralbaugruppe mit Mikroprozessor und Speicher
- Eingänge mit Pegelanpassung und galvanischer Trennung
- Ausgänge mit Pegelanpassung und galvanischer Trennung
- Spannungsversorgung

Bei den Automatisierungsgeräten wird zwischen **Kompaktsteuerungen** und **modularen Steuerungen** unterschieden.

Kompaktsteuerungen beinhalten die Zentralbaugruppe sowie Ein- und Ausgänge in einem gemeinsamen Gehäuse. Im Allgemeinen ist die Spannungsversorgung extern zuzuführen.

Modulare Steuerungen bestehen aus einzelnen Komponenten, die nach Bedarf zusammengestellt werden können. Spannungsversorgung und Zentralbaugruppe sind obligatorisch. Ein- und Ausgabebaugruppen können bedarfsgerecht ergänzt werden.

Spannungsversorgung	Zentralbaugruppe	Eingabebaugruppe 0	Eingabebaugruppe 1	Eingabebaugruppe 2	Eingabebaugruppe 3	Ausgabebaugruppe 4	Ausgabebaugruppe 5	Ausgabebaugruppe 6
		Eingänge I 0.0 ⋮ I 0.7	Eingänge I 1.0 ⋮ I 1.7	Eingänge I 2.0 ⋮ I 2.7	Eingänge I 3.0 ⋮ I 3.7	Ausgänge Q 4.0 ⋮ Q 4.7	Ausgänge Q 5.0 ⋮ Q 5.7	Ausgänge Q 6.0 ⋮ Q 6.7

Abbildung 40 Aufbau eines modularen Automatisierungsgerätes, Beispiel

Es bedeuten:

I:	Input	Eingang, auch E anzutreffen
Q:	Output, Q ersatzweise für O, denn der Buchstabe O ist für das logische ODER vergeben	Ausgang, auch A anzutreffen

Operandenangabe bei modularen Steuerungen

I 3.6

- Kanalnummer
- Steckplatznummer
- Baugruppentyp
(hier Eingabe-
baugruppe)

Der **Operand** gibt Antwort auf die Frage:
Womit ist etwas zu tun?

Mögliche Antwort: Mit dem Eingang I3.6.

Abbildung 41 Operandenangabe, Beispiel

Prinzipiell kann jeder Steckplatz des modularen Systems mit beliebigen Baugruppentypen bestückt werden. Einschränkungen gelten oftmals für die Spannungsversorgung und die Zentralbaugruppe. Im Einzelfall geben die Handbücher der Hersteller hierüber Auskunft.

Signalzustände

Annahme: An Eingang I2.0 (Baugruppe-Nr. 2, Eingang 0) ist ein Schließer angeschlossen.

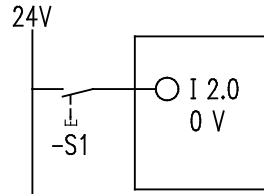


Abbildung 42 Schließer unbetätigt

Wenn der Schließer nicht betätigt wird, liegt am Eingang I2.0 des Automatisierungsgerätes keine Spannung an (0 V). Man sagt dann auch, der Eingang I2.0 führt den **Signalzustand „0“**.

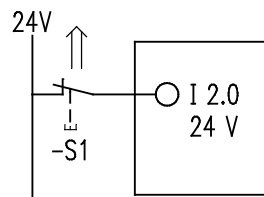


Abbildung 43 Schließer betätigt

Wenn der Schließer S1 betätigt wird, nimmt der Eingang I2.0 die Spannung 24 V an. Man sagt, der Eingang I2.0 führt den **Signalzustand „1“**.

Allgemein gilt in der Steuerungstechnik folgende Festlegung:

keine Spannung	Signalzustand „0“ „0“-Signal
Spannung	Signalzustand „1“ „1“-Signal

Tabelle 1 Digitale Signalzustände

Obige Angaben der Signalzustände sind stark vereinfacht.

Exakt kann z.B. gelten:

„0“-Signal: – 35 V ... + 4,5 V

„1“-Signal: + 13 V ... + 35 V

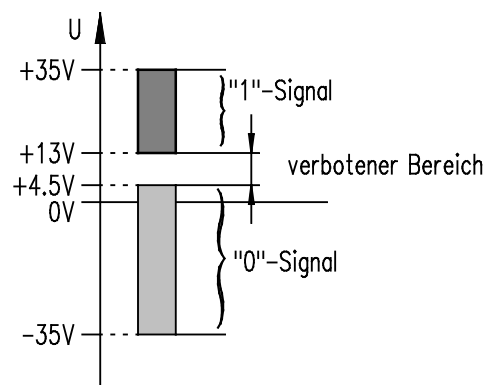


Abbildung 44 Trennung der Signalzustände

Die verbotene Zone (4,5 ... 13 V) dient zur eindeutigen Trennung der beiden Signalzustände.

SPS-Hardware

Der grundsätzliche Aufbau der SPS-Hardware entspricht dem EVA-Prinzip. Digitale oder analoge Signale werden eingegeben, nach einem festgelegten Programm verarbeitet und als Verarbeitungsergebnis in digitaler oder analoger Form ausgegeben.

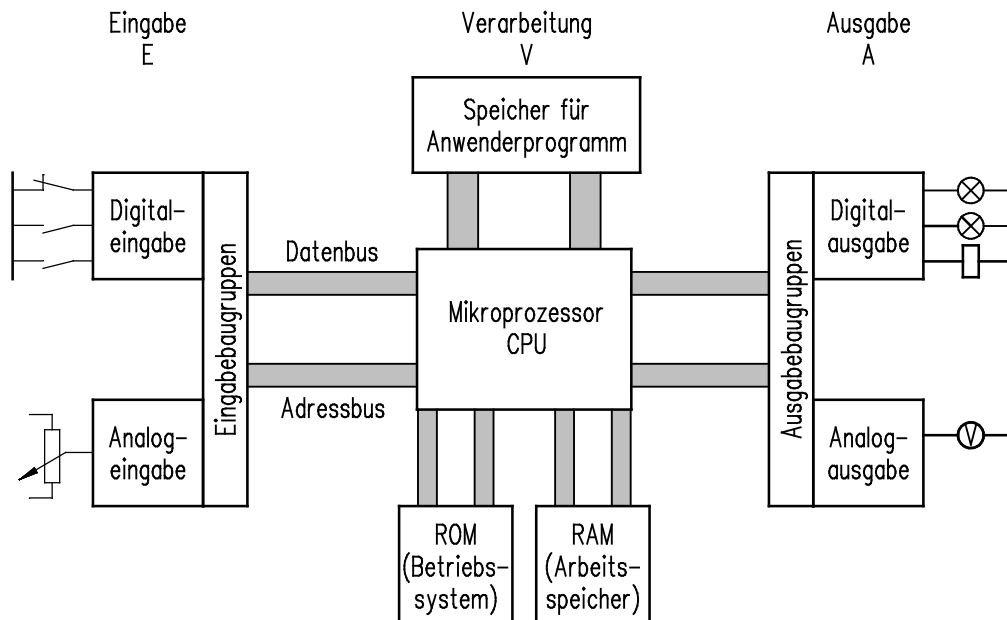


Abbildung 45 SPS-Hardware, Prinzip

Kernstück des SPS-System ist der **Mikroprozessor** (CPU; Central Processing Unit). Der Mikroprozessor organisiert den Datenverkehr des Systems, führt die Programm-anweisungen durch (z.B. logische Verknüpfungen) und gibt somit dem System die erforderliche „Intelligenz“.

Das **Betriebssystem** ist in einem **ROM** abgelegt (Read Only Memory, Nur-Lese-Speicher). Es ermöglicht dem Anwender die Arbeit mit dem SPS-System. Zum Beispiel:

- gibt das System gezielte Fehlermeldungen aus und nimmt Daten entgegen.
- wird der Datenverkehr zwischen den einzelnen Funktionsgruppen des Automatisierungsgerätes über das Bussystem gesteuert.
- führt das Rechenwerk die Anwenderprogramme aus.

Für all diese Aufgaben sind Programme erforderlich, die vom Hersteller der Hardware als **Betriebssystem** mitgeliefert werden und dem Anwender unmittelbar zur Verfügung stehen.

Im **Arbeitsspeicher** (RAM; Random Access Memory, Schreib-Lese-Speicher) werden z.B.

- die sich ändernden Eingangssignale,
- die internen Ergebnisse von logischen und mathematischen Operationen

gespeichert. Es wird in diesem Zusammenhang auch von einem **Speicher** gesprochen.

Der **Speicher für das Anwenderprogramm** beinhaltet die für die Lösung der jeweiligen Aufgabenstellung erforderliche Software. Das Anwenderprogramm bestimmt, welche Aufgabe von der standardisierten SPS-Hardware übernommen wird. Bei glei-

cher Hardware kann dies eine Ampelsteuerung, die Steuerung einer Werkzeugmaschine oder eines Logistiksystems sein.

Der Speicher für das Anwenderprogramm muss **nullspannungssicher** sein. Bei Spannungsausfall darf das Anwenderprogramm nicht verloren gehen. Möglich ist z.B. ein batteriegepuffertes RAM oder auch ein EEPROM (elektrisch löschbares und programmierbares ROM).

EEPROMS werden elektrisch gelöscht und sind anschließend byteweise neu programmierbar.

Bei den **Eingabebaugruppen** wird zwischen

- **Digitaleingabe**
Digitale Signale werden erfasst, zwischengespeichert und zwecks Verarbeitung an die CPU weitergeleitet.

und

- **Analogeingabe**
Analoge Signale werden erfasst, in digitale Signale umgewandelt und an die CPU weitergeleitet

unterschieden.

Ebenso wird bei den **Ausgabebaugruppen** zwischen Digital- und Analogausgabe unterschieden.

Programmbearbeitung bei der SPS

Die Begriffe Betriebssystem und Anwenderprogramm wurden bereits vorab erwähnt.

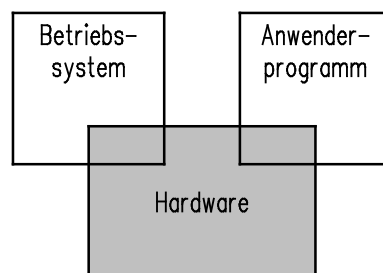


Abbildung 46 Softwarekonzept der SPS

Das **Betriebssystem** (die Betriebssoftware) hat eine große Hardwarenähe. Diese Software muss direkt mit der Hardware korrespondieren und wird demzufolge vom Hardwarehersteller entwickelt und mit der Hardware ausgeliefert. Unter anderem übernimmt das Betriebssystem folgende Aufgaben:

- Ausgabe auf Drucker
- Zusammenwirken mit dem Programmiergerät bzw. PC
- Eine Reihe von Hilfsprogrammen zum komfortablen Programmieren und Testen
- Unterstützung der SPS-Programmiersprachen

Annahme: Die Signalzustände an den Eingängen I1.0 und I1.1 sollen UND-verknüpft werden. Das Verknüpfungsergebnis soll dem Ausgang Q4.0 zugewiesen werden.

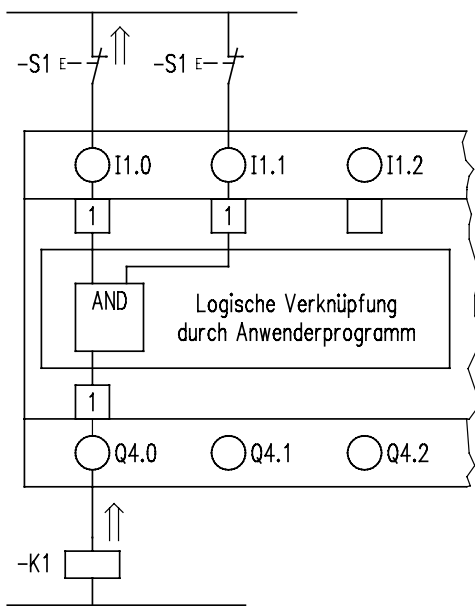


Abbildung 47 Logische Verknüpfung durch SPS-Programm

Kennzeichnend für die Abarbeitung eines SPS-Programms ist die zyklische Arbeitsweise. Nach Durchlauf des Programms wird es sofort wieder erneut durchlaufen. Das SPS-Programm arbeitet in einer **gewollten** Endlosschleife.

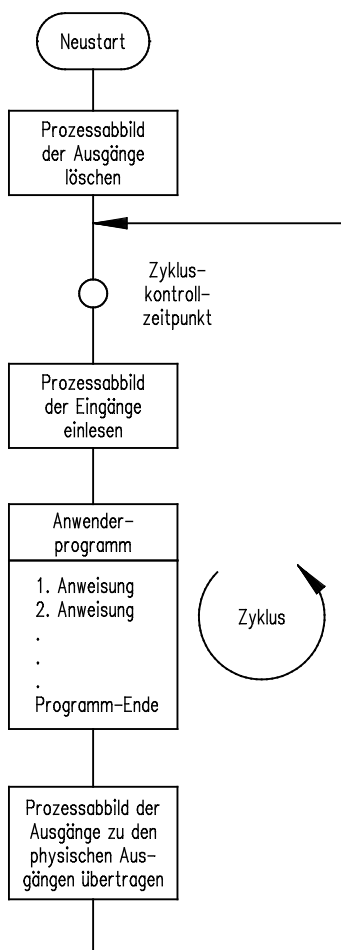


Abbildung 48 Programmbearbeitung einer SPS, allgemein

Die Abarbeitung des SPS-Programms erfolgt nach dem **Prozessabbild**. Dies bedeutet: Wenn das **Prozessabbild der Eingänge** gelesen wird, werden die an den Eingängen anliegenden Signalzustände in den Zwischenspeicher für die Eingangssignale transportiert (Signalspeicher). Mit diesen abgespeicherten Signalzuständen wird das Anwenderprogramm bearbeitet. Etwaige Änderungen der Eingangssignale haben zu diesem Zeitpunkt keinen Einfluss auf die Bearbeitung des Anwenderprogramms.

Wenn das Anwenderprogramm vollständig bearbeitet wurde, wird das **Prozessabbild der Ausgänge** gebildet. Die sich bei der Programmbearbeitung ergebenden Ergebnisse werden dann in den Zwischenspeicher für die Ausgangssignale (Signalspeicher) transportiert und von dort aus zu den Ausgängen übertragen.

Danach wird das Prozessabbild der Eingänge erneut eingelesen, und die beschriebenen Vorgänge wiederholen sich **zyklisch**.

Zu beachten: Während der sequenziellen Programmbearbeitung haben Änderungen der Eingangssignale keinen Einfluss auf die Programmbearbeitung (gearbeitet wird mit dem Prozessabbild der Eingänge, das zu einem früheren Zeitpunkt gebildet wurde). Die Ergebnisse der Programmbearbeitung werden erst nach Beendigung des jeweiligen Programmdurchlaufes an die Ausgänge weitergegeben.

Damit kommt der Zeit, die für einen Programmdurchlauf benötigt wird, große Bedeutung zu, da sie sozusagen die Reaktionsgeschwindigkeit der SPS bestimmt. Dies ist die Zeit, die zwischen der Änderung eines Eingangssignals und der entsprechenden Reaktion an den Ausgängen verstreicht.

Die **Zykluszeit**, die Zeit, die zwischen zweimaligem Passieren des **Zykluskontrollzeitpunktes** verstreicht, ist eine wichtige Kenngröße der SPS-Hardware. Die Zykluszeit wird ständig überwacht. Wenn ein Maximalwert überschritten wird, bringt das Betriebssystem die Steuerung in den STOP-Zustand.

Während der Programmbearbeitung werden Änderungen der Eingangssignale nicht berücksichtigt, und es werden keine neuen Ausgangssignale ausgegeben.

Treten gleichzeitig gegebene **gegensätzliche** Anweisungen auf, so hat die Anweisung Vorrang, die als letzte im Programm steht, da sie im Zwischenspeicher für die Ausgangssignale gespeichert wird.

Wenn zum Beispiel am Anfang des Programms der Ausgang Q4.0 eingeschaltet werden soll (erst nach Beendigung der Programmbearbeitung) und zu einem späteren Zeitpunkt beim gleichen Programmdurchlauf der Ausgang Q4.0 ausgeschaltet werden (oder ausgeschaltet bleiben) soll, so wird die näher am Programmende stehende Anweisung ausgeführt.

Kommen wir nun zurück auf die dargestellte UND-Funktion.

Die Bearbeitung dieser Aufgabe erfolgt folgendermaßen.

1. Signalzustände in Zwischenspeicher für die Eingangssignale einlesen.
2. Signalzustand von I1.0 in den Akkumulator transportieren (laden).
3. UND-Verknüpfung zwischen dem Akkumulatorinhalt und dem Signalzustand von Eingang I1.1 (Zwischenspeicher) bilden.
Das Verknüpfungsergebnis steht im Akkumulator.
4. Inhalt des Akkumulators in den Zwischenspeicher für die Ausgangssignale transportieren (speichern).
5. Inhalt des Zwischenspeichers für die Ausgangssignale an die Ausgänge (an den Steuerungsprozess) ausgeben.

Danach wiederholen sich die Vorgänge.

2 Realisierung einer Steuerung mit Verknüpfungselementen (KOP, AWL, FBS)

Lernbereich

Die Beschreibung eines steuerungstechnischen Problems mit Umgangssprache lässt bereits viele Elemente von logischen Verknüpfungen erkennen.

Ziel dieses Lernbereichs ist die Vermittlung von standardisierten Beschreibungssprachen für Steuerungsaufgaben mit Verknüpfungselementen.

Durch die Darstellung einer typischen Steuerungsaufgabe am Beispiel einer Säge mit Absaugvorrichtung werden logische Verknüpfungen formuliert.

Nach der Erarbeitung der erforderlichen logischen Funktionen und deren normierte Darstellung wird als Abschluss des Lernbereichs das Projekt 1 „Säge mit Absaugvorrichtung“ gelöst.

Projekt 1: Säge mit Absaugvorrichtung

Für eine Kreissäge mit einer Absaugvorrichtung ist eine Steuerung zu entwerfen.

Funktionsbeschreibung

Die Säge und die Absaugvorrichtung werden über Taster ein- und ausgeschaltet. Dabei soll die Absaugvorrichtung beim Ausschalten eine einstellbare Zeit nachlaufen. Es wird folgende Zuordnung getroffen:

Betriebsm.	Ein-/Ausgang	Symb. Name	Kommentar
S1	E1	EIN_1	EIN-Taster, Schließer
S2	E2	AUS_1	AUS-Taster, Öffner
K1	A1	SAEGE	Schütz für die Säge
K2	A2	ABSAUG	Schütz für die Absaugvorrichtung

Tabelle 2 Zuordnungstabelle

Signalzuordnung der Ausgänge:

- A1 = 1 - Säge eingeschaltet
- A1 = 0 - Säge ausgeschaltet
- A2 = 1 - Absaugvorrichtung in Betrieb
- A2 = 0 - Absaugvorrichtung außer Betrieb

Für dieses steuerungstechnische Problem ist eine Lösung mit Verknüpfungselementen geeignet. Hierfür werden im Folgenden die Grundlagen bereitgestellt und erarbeitet.

2.1 Boolesche Grundfunktionen

UND-Funktion

Der Ausgang der UND-Funktion nimmt den booleschen Wert TRUE (den Signalzustand „1“) an, wenn sämtliche Eingänge den booleschen Wert TRUE haben.

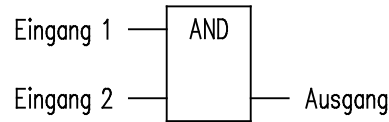


Abbildung 49 Darstellung einer UND-Funktion

Darstellung als Anweisungsliste (AWL)

Hinweise:

Der Marke (dem Label) muss ein Doppelpunkt folgen (hier TEST:)

Marke	Operator	Operand	Kommentar
TEST:	LD	%IX1.6	(*Eingang 1.6*)
	AND	%IX2.4	(*Eingang 2.4*)
	ST	%Q4.0	(*Ausgang 4.0*)

Tabelle 3 Darstellung einer AWL

Jede **Anweisung**, bestehend aus **Operator** und **Operand**, muss in einer neuen Zeile beginnen. Das Einfügen von **Leerzeilen** zwischen den Anweisungen ist zulässig.

Die hier verwendeten **Operatoren** bedeuten:

LD	(load, laden)	Setzt das Aktuelle Ergebnis (AE) dem Operanden gleich
AND	(UND)	UND-Funktion
ST	(store, speichern)	Speichert das Aktuelle Ergebnis (AE) auf die Operandenadresse

Das **Aktuelle Ergebnis (AE)** ist mit dem Akkumulator des Mikroprozessors vergleichbar. Bei herkömmlichen Systemen wird in diesem Zusammenhang von einem **Verknüpfungs-Ergebnis-Register (VKE)** gesprochen. Im AE sind die Zwischenergebnisse bei der Programmbearbeitung abgespeichert.

Die hier verwendeten **Operanden** bedeuten:

I	(Input; Eingang)	Eingang Speicherort
Q	(Ausgang)	Ausgang Speicherort
X		Bit-Größe (TRUE oder FALSE, „1“ oder „0“); auf die Angabe von „X“ kann auch verzichtet werden

Dem Operanden ist ein **Prozentzeichen (%)** voranzustellen

Kommentare des Anwenders müssen an Anfang und Ende mit der Zeichenkombination (* bzw. *) versehen werden.

Verdeutlichung der Programmabarbeitung

Es wird angenommen, dass die beiden Schließer S1 und S2 betätigt sind. Sie liefern dann den booleschen Wert TRUE an die Eingänge IX1.0 und IX1.1.

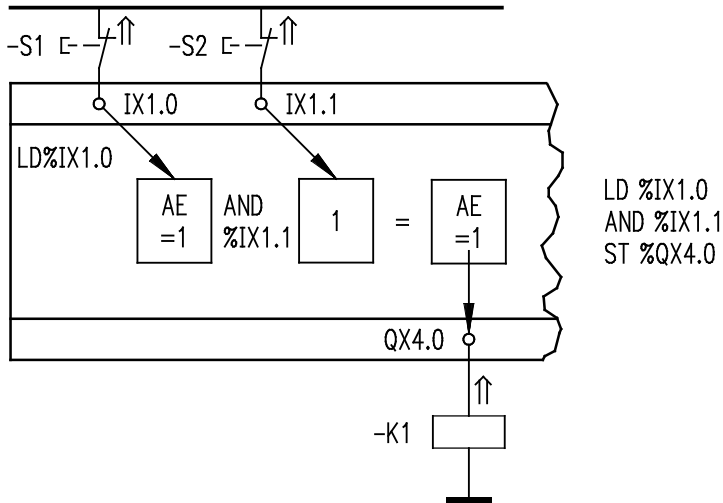


Abbildung 50 Abarbeitung eines Programms

Bei der Bearbeitung der Anweisungsliste laufen folgende Vorgänge ab:

LD %IX1.0

Der boolesche Wert am Eingang IX1.0 wird gelesen und in das Aktuelle Ergebnis (AE) geladen (AE = 1).

AND %IX1.1

Der momentane Inhalt des AE („1“) wird mit dem an Eingang IX1.1 anliegenden booleschen Wert („1“) UND-verknüpft. Mit dem Verknüpfungsergebnis („1“) wird das AE überschrieben.

ST %QX4.0

Der momentane Inhalt des AE („1“) wird auf die Operandenadresse QX4.0 gespeichert. Der Ausgang QX4.0 nimmt also den booleschen Wert „1“ an. Das Schütz K1 zieht an.

Darstellung in Funktionsbaustein-Sprache (FBS)

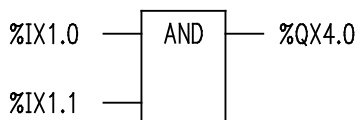
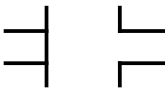
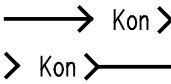


Abbildung 51 UND-Funktion in FBS-Darstellung

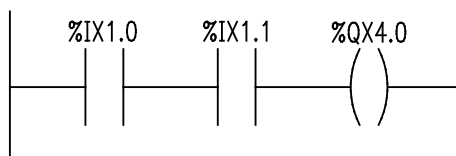
Element	Darstellung
Horizontale Linie	—
Vertikale Linie	
Linienverbindung (horizontal u. vertikal)	⊕
Linienkreuzung (ohne Verbindung)	⊖
Ecken (verbunden und nicht verbunden)	
Konnektor	

Die **Funktionsbaustein-Sprache** (FBS) nach IEC 61131-3 ist vergleichbar mit dem klassischen Funktionsplan.

Der **Konnektor** (Verbinder) ist die Fortsetzung einer verbundenen Linie. Dadurch wird eine übersichtliche Darstellung ermöglicht.

Tabelle 4 Darstellung von Linien (FBS) nach IEC 61131-3

Darstellung als Kontaktplan (KOP)



Der Kontaktplan (engl.: ladder diagram) hat eine enge Verwandtschaft mit dem Bereich elektromagnetischer Relaisysteme. Eine weitgehende Beschränkung auf die Bearbeitung **boolescher** Variablen ist kennzeichnend für den KOP.

Abbildung 52 UND-Funktion in KOP-Darstellung

Die Netzwerke des KOP werden i.Allg. in der Reihenfolge von oben nach unten ausgewertet, so wie sie im KOP grafisch angeordnet sind.

Ein **Netzwerk** wird dadurch bestimmt, dass das Aktuelle Ergebnis (AE) nicht über die Netzwerkgrenze hinaus wirksam ist. Es kann gesagt werden, dass ein Netzwerk durch eine **Ergebnisbildung** abgeschlossen wird.

Regeln für die KOP-Darstellung:

- Die „Stromflussrichtung“ im Kontaktplan ist von links nach rechts.
- Ein Netzwerk in Kontaktplandarstellung wird links durch eine senkrechte Linie (linke Stromschiene) und rechts durch eine senkrechte Linie (rechte Stromschiene) begrenzt.
- Verbindungslinien dürfen waagerecht und senkrecht verlaufen.
- Der Zustand der Verbindungslinien kann mit EIN oder AUS bezeichnet werden, den booleschen Werten TRUE bzw. FALSE entsprechend. Die linke Stromschiene hat immer dem Zustand TRUE. Für die rechte Stromschiene gibt es keinen definierten Zustand.

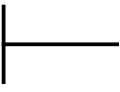
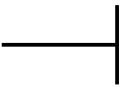

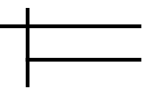
Element	Darstellung
Linke Stromschiene mit angebundener horizontaler Verbindung	
Rechte Stromschiene mit angebundener horizontaler Verbindung	
Horizontale Verbindung	
Vertikale Verbindung mit angebundenen horizontalen Verbindungen	

Tabelle 5 Stromschienen und Verbindungen im KOP nach IEC 61131-3

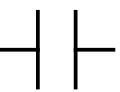
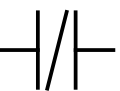


Element	Darstellung
Schließer (Abfrage auf den booleschen Wert "1")	
Öffner (Abfrage auf den booleschen Wert "0")	
Spule (Ergebniszuweisung)	
Negative Spule (Negierte Ergebniszuweisung)	

Tabelle 6 Kontakte und Spulen im KOP nach IEC 61131-3

Bei **Kontakten** wird zwischen **Öffner** und **Schließer** unterschieden.

Schließer in Ruhestellung: FALSE, Signalzustand „0“.
 Öffner in Ruhestellung: TRUE, Signalzustand „1“.

Ein **betätigter Öffner** liefert den Signalzustand „0“ (FALSE). Er wird auf den Signalzustand „0“ abgefragt.

Hinweise:

- Ein **Kontakt** übergibt einen booleschen Wert an die horizontale Verbindung seiner rechten Seite. Er verändert den Wert der zugehörigen booleschen Variablen nicht.
- Eine **Spule** kopiert den Zustand der Verbindung auf der linken Seite ohne Veränderung auf die rechte Seite.
- Eine **negative Ergebniszuweisung** wird in der Programmiersprache AWL durch STN ausgedrückt.

Lehrbeispiel 1

Das Programm in FBS-Darstellung ist als Anweisungsliste und als Kontaktplan darzustellen!

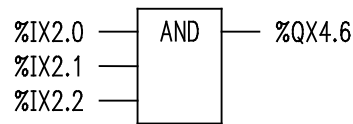


Abbildung 53 UND-Funktion in FBS

AWL:

```
LD  %IX2.0
AND %IX2.1
AND %IX2.2
ST  %QX4.6
```

KOP:

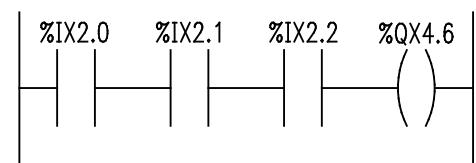


Abbildung 54 Kontaktplan einer UND-Funktion

ODER-Funktion

Der Ausgang einer ODER-Funktion nimmt den booleschen Wert TRUE an, wenn mindestens ein Eingang den booleschen Wert TRUE hat.

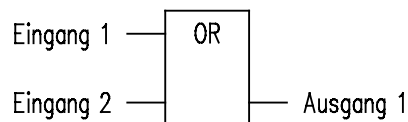


Abbildung 55 ODER-Funktion

Darstellung als AWL

```
LD  %IX2.0 (*Eingang 1*)
OR  %IX2.1 (*Eingang 2*)
ST  %QX4.0 (*Ausgang1*)
```

Hinweis:
OR (ODER) ODER-Funktion

Darstellung in FBS

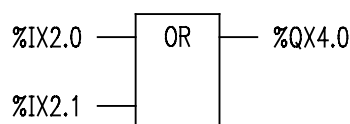


Abbildung 56 ODER-Funktion in FBS

Darstellung als KOP

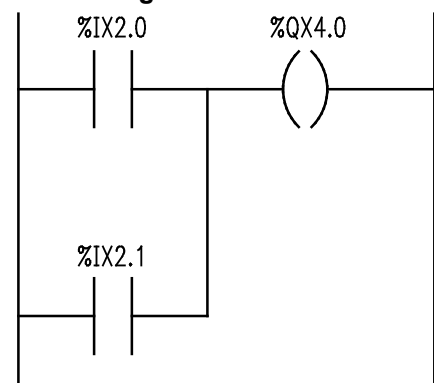


Abbildung 57 ODER-Funktion in KOP

NICHT-Funktion (Negation)

Der Ausgang der NICHT-Funktion (Negation) nimmt stets den entgegengesetzten booleschen Wert wie der Eingang an.

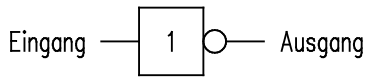


Abbildung 58 NICHT-Funktion

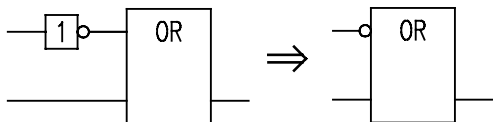
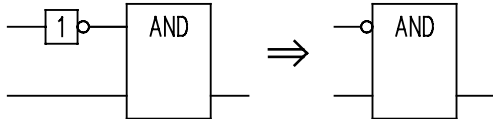


Abbildung 59 Darstellung der NICHT-Funktion in Verbindung mit der UND- bzw. ODER-Funktion

Darstellung als AWL

LDN %IX2.0 (*Eingang 1*)
 ANDN %IX2.1 (*Eingang 2*)
 ST %QX4.0 (*Ausgang 1*)

Die booleschen Werte an den Eingängen IX2.0 und IX2.1 werden negiert. Wenn beide Eingänge den booleschen Wert FALSE führen, nimmt der Ausgang den booleschen Wert TRUE an.

Der **Operator** der NICHT-Funktion ist N. Er tritt in Verbindung mit anderen Operatoren auf (z.B. LDN, ANDN, ORN, STN).

Darstellung in FBS

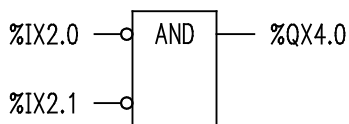


Abbildung 60 Negation in FBS

Darstellung als KOP

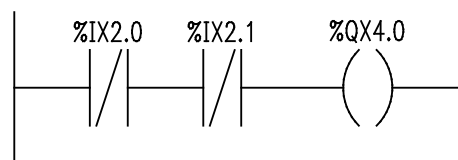


Abbildung 61 Negation in KOP

Lehrbeispiel 2

Dargestellt ist ein Steuerungsprogramm in FBS. Stellen Sie das Programm als Anweisungsliste und als Kontaktplan dar!

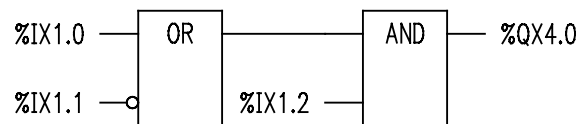


Abbildung 62 Funktionsbausteinsprache

KOP:

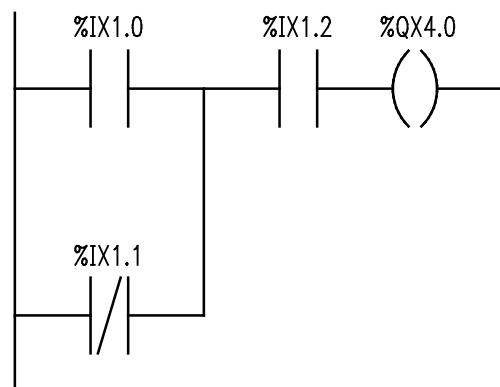


Abbildung 63 Kontaktplan

AWL:

```

LD  %IX1.0
ORN %IX1.1
AND %IX1.2
ST  %QX4.0
  
```

Variablen

Variablen sind vom Anwender definierte **Bezeichner**. Sie werden als Platzhalter für die Daten des SPS-Programms verwendet.

Unter einem **Bezeichner** wird nach IEC 61131-3 eine Folge von Buchstaben, Ziffern und Unterstrichzeichen (_) verstanden. Diese Folge muss mit einem Buchstaben, einer Ziffer oder einem Unterstrich beginnen und darf keine Leerzeichen enthalten.

Unabhängig vom jeweils verwendeten Steuerungssystem müssen nach IEC 61131-3 zumindest sechs Zeichen unterstützt werden. So müssen auf alle Fälle die Bezeichner

AUST_1 (*Austaster 1*)

und

AUST_2 (*Austaster 2*)

eindeutig unterschieden werden können. Moderne Programmiersysteme unterstützen aber wesentlich mehr Zeichen. Die Handbücher der Hersteller geben darüber Aufschluss.

Beispiele für Bezeichner:

```

EIN_1
1_AUSGANG
PUMPE_EIN
_MAGNET_AUS
  
```


IEC 61131-3 kennt keine globalen Zuordnungslisten, wie bei der „herkömmlichen“ Programmierung üblich. **Variablen** werden zur Informationsspeicherung und Informationsverarbeitung verwendet. Ihr Speicherort braucht nicht vom Anwender manuell bestimmt werden. Die Variable wird vom Programmiersystem automatisch verwaltet und hat einen fest zugeordneten **Datentyp**.

Der **Datentyp** bestimmt, welche **Werte** die Variable annehmen kann und wie viel **Speicherplatz** für die Variable zu reservieren ist.

Datentyp	Schlüsselwort	Bits
boolesche Daten	BOOL	1
ganze Zahl	INT (integer)	16
reelle Zahl	REAL	32
Datum	DATE	systemabhängig
Zeitdauer	TIME	systemabhängig
8-Bit-Folge	BYTE	8
16-Bit-Folge	WORD	16
32-Bit-Folge	DWORD	32
64-Bit-Folge	LWORD	64
Zeichenfolge	STRING	systemabhängig

Tabelle 7 Datentyp nach IEC 61131-3 (Auszug)

Schlüsselwörter sind nach IEC 61131-3 eindeutige Kombinationen von Zeichen, die als individuelle syntaktische Elemente angewendet werden. Schlüsselwörter dürfen nicht für andere Zwecke (z.B. als Variablennamen) verwendet werden.

Boolesche Daten werden mit dem Wert „0“ oder „1“ dargestellt; sie können auch durch die **Schlüsselwörter** FALSE oder TRUE angegeben werden.

Die im Programm (IEC 61131-3 spricht von einer Programm-Organisationseinheit POE) verwendeten Variablen müssen **deklariert** werden. Für diese Variablendeklaration sind folgende **Schlüsselwörter** von Bedeutung.

Schlüsselwort	Variablengebrauch
VAR	innerhalb einer Programm-Organisationseinheit
VAR_INPUT	von außerhalb kommend; nicht innerhalb der POE änderbar
VAR_OUTPUT	von der POE nach außen geliefert
VAR_IN_OUT	von außen kommend; kann innerhalb der POE geändert werden und wird dann nach außen geliefert
CONSTANT	Konstante
AT	Zuweisung des Speicherortes

Tabelle 8 Schlüsselwörter für Variablendeklaration nach IEC 61131-3 (Auszug)

Lehrbeispiel 3

Für die dargestellte Programm-Organisationseinheit (POE) soll die Variablendeklaration durchgeführt werden.

Die POE ist hier als Anweisungsliste dargestellt.

```
LD      %IX2.0  (*Eingang 1*)
ANDN    %IX2.1  (*Eingang 2*)
OR      %IX2.2  (*Eingang 3*)
ST      %QX4.0  (*Ausgang 1*)
```

In obiger Anweisungsliste sind die Variablen **direkt dargestellt**. Der **Speicherort** ist dann **direkt ersichtlich**. Diese Variablendarstellung ist allenfalls bei einer relativ geringen Variablenanzahl sinnvoll. Für die **direkte Darstellung von Variablen** ist ein besonderes **Symbol** notwendig, das bei den vorangegangenen Darstellungen schon mehrfach verwendet wurde.

Das **Symbol** besteht aus

- einem Prozentzeichen %
- der Speicherortangabe
- der Größenangabe
- einer oder mehrerer ganzer Zahlen, die durch Punkte getrennt sind.

Speicherort Eingang	I
Speicherort Ausgang	Q
Speicherort Merker	M
Einzelbit-Größe	X oder keine Angabe
Byte-Größe (8 Bit)	B
Wort-Größe (16 Bit)	W
Doppelwort-Größe (32 Bit)	D
Langwort-Größe (64 Bit)	L

Tabelle 9 Speicherortangabe bei direkt dargestellten Variablen (IEC 61131-3)

```
%IX3      Eingangsbit; kann auch so dargestellt werden: %I3
%QB5      Ausgangsbyte 5 (8 Bit)
%QD6      Ausgangs-Doppelwort 6 (32 Bit)
%IX1.2.3.4 Eingangsbit: Bus 1, Baugruppenträger 2, Modul 3, Kanal 4
```

Der **Speicherort** sollte stets zum letztmöglichen Zeitpunkt und in möglichst hoher Ebene festgelegt werden, da dadurch die Flexibilität der Programm-Organisationseinheit wesentlich gesteigert wird.

Die in diesem Lehrbeispiel dargestellte Anweisungsliste mit **direkt dargestellten Variablen** ist nur bei der hier gewählten Hardware-Beschaltung einsetzbar. Bei anderer Hardwarebeschaltung (anderer Anschluss der Sensoren und Aktoren an die Ein- und Ausgänge) wird das Programm nicht mehr wunschgemäß arbeiten.

Flexibler ist da schon die **symbolische Adressierung** der Variablen.


```
VAR
  EIN_1 AT %IX2.0 : BOOL;
  EIN_2 AT %IX2.1 : BOOL;
  EIN_3 AT %IX2.2 : BOOL;
  AUS_1 AT %QX4.0 : BOOL;
END_VAR
```

Die Variablendeklaration mit Speicherortzuweisung wird in die Schlüsselwörter VAR... END_VAR eingeschlossen. Durch das Schlüsselwort **AT** erfolgt die Zuweisung des Speicherortes. Die Variable EIN_1 ist also z.B. dem Speicherort %IX2.0 zugewiesen worden.

Zu beachten ist die Schreibweise der Variablendeklaration mit Doppelpunkt und Semikolon. Die Angabe BOOL (engl.: BOOLEAN) besagt, dass es sich um eine boolesche Variable (TRUE oder FALSE) handelt.

Noch einmal in übersichtlicher Form:

```
EIN_1 AT %IX2.0 : BOOL ;
```



- Semikolon
- Boolesche Variable
- Doppelpunkt
- Bit-Eingang
- Speicherortzuweisung
- Variablenname

Abbildung 64 Deklaration

Damit kann die Anweisungsliste wie folgt geschrieben werden.

```
VAR                                     (*Deklarationsteil*)
  EIN_1 AT %IX2.0 : BOOL;
  EIN_2 AT %IX2.1 : BOOL;
  EIN_3 AT %IX2.2 : BOOL;
  AUS_1 AT %QX4.0 : BOOL;
END_VAR

LD      EIN_1      (*Eingang 1*)
ANDN    EIN_2      (*Eingang 2*)
OR      EIN_3      (*Eingang 3*)
ST      AUS_1      (*Ausgang 1*)
```

Programm-Organisationseinheiten (POEs)

Programm-Organisationseinheiten sind kleine, voneinander unabhängige Einheiten des SPS-Programms nach IEC 61131-3. Sie können grundsätzlich mit den Bausteinen „herkömmlicher“ Programmiersprachen verglichen werden.

Jede POE besteht aus **Deklarationsteil** und **Anweisungsteil**. POEs sind selbstständige Programmelemente, die praktisch unabhängig von anderen Einheiten vom Compiler übersetzt werden können.

Nach dem Übersetzungsvorgang lassen sich die Einheiten zu einem Gesamtprogramm verbinden (Linken). Dies hat den wesentlichen Vorteil, dass die Problemlösungen **modularisierbar** sind und gleichartige Module für gleichartige Aufgabenstellungen universell eingesetzt werden können (Mehrfachverwendbarkeit).

Schlüsselwort	POE	Kommentar
PROGRAM	Programm	Hierbei handelt es sich um ein Hauptprogramm, das den Zugriff auf die SPS-Peripherie ermöglicht. Hier erfolgt die Deklaration von globalen Variablen und Zugriffspfaden.
FUNCTION_BLOCK	Funktionsbaustein	Hat Ein- und Ausgangsvariablen; wird sehr häufig bei der Programmerstellung genutzt und ist instanzierbar.
FUNCTION	Funktion	Einfacher SPS-Baustein, der den Operationsvorrat bei der Programmierung erweitert.

Tabelle 10 POEs nach IEC 61131-3 mit zugehörigen Schlüsselwörtern

Funktion

Die Funktion ist eine Programm-Organisationseinheit, die bei Ausführung genau ein Datenelement liefert. Dabei kann sie beliebig viele **Eingangsparameter** haben. Bei gleichen Eingangssignalen liefert die Funktion stets das gleiche Ergebnis. Funktionen haben **kein Speicherverhalten**, kein „Gedächtnis“.

Funktionen können **lokale Variablen** (auf die Funktion selbst begrenzte Variablen) verwenden. Es ist allerdings zu beachten, dass diese nach Beendigung der Funktion verloren gehen.

Direkt dargestellte Variablen können in Funktionen nicht deklariert werden. Funktionen können in **Textform** oder **grafisch** dargestellt werden.

Lehrbeispiel 4

(*Textform einer Funktion*)

$X := A * B + C * D;$

Beachten Sie bitte die Wertzuweisung ($:=$) und das Semikolon (;)

(*Grafische Form der Funktionsdarstellung*)

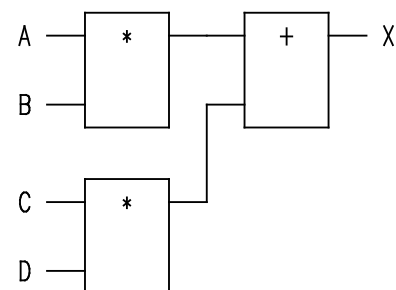


Abbildung 65 Funktion

Deklaration von Funktionen

Dem Schlüsselwort **FUNCTION** folgt der **Funktionsname**, ein **Doppelpunkt** und der **Datentyp** des Wertes, den die Funktion liefert. Zum Beispiel

FUNCTION TEST : REAL

Der Funktionsname lautet TEST, der Rückgabewert ist vom Datentyp REAL.

Lehrbeispiel 5

```
FUNCTION RECHNUNG : REAL
  VAR_INPUT
    A, B, C, D : REAL;
  END_VAR
  RECHNUNG := A * B + C * D;
END_FUNCTION
```

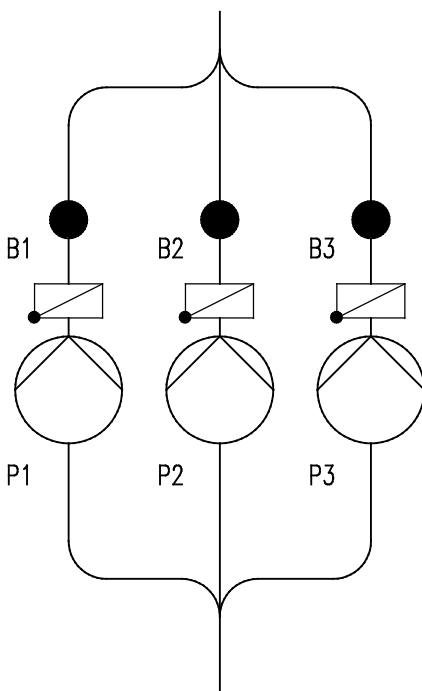
Lehrbeispiel 6

Abbildung 66 Umwälzpumpen

Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben.

Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst.

Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal). Wenn keine Strömung erkannt wird, liefert er somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung betätigt werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten
 S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten
 S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten
 S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Die Ansteuerung von Pumpe 1 erfolgt beispielsweise wie dargestellt.

Hierbei handelt es sich um ein komplexes Netzwerk, das nun Grundlage für ein SPS-Programm sein soll.

K4A, K5A und K6A sind Hilfsschütze, die im SPS-Programm durch **Merker** dargestellt werden können. Merker sind 1-Bit-Speicherelemente, die den booleschen Wert TRUE oder FALSE annehmen können. Die Merker signalisieren hier, welche zwei Pumpen miteinander arbeiten. Es gilt folgende Zuordnung:

K4A: Pumpe 1 und Pumpe 2 arbeiten
 K5A: Pumpe 2 und Pumpe 3 arbeiten
 K6A: Pumpe 1 und Pumpe 3 arbeiten

Möglich ist die **direkte Darstellung**, z.B.

K4A → %MX0.4
 K5A → %MX0.5
 K6A → %MX0.6

oder auch die **symbolische Darstellung**, z.B.

K4A → MERK_4
 K5A → MERK_5
 K6A → MERK_6

Weiterhin werden folgende Variablennamen vereinbart:

B2 → STROEM_PUMPE_2
 B3 → STROEM_PUMPE_3
 F4 → THERM_MOT_SCH_P1
 K1 → PUMPE_1

Programm in FBS

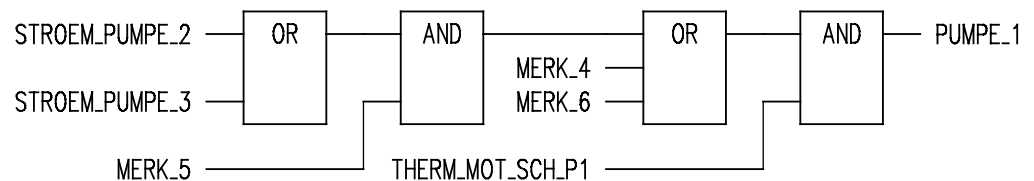


Abbildung 67 Steuerung Pumpe 1

Hinweis:

- Die Strömungswächter werden auf den Signalzustand „1“ abgefragt. Wenn keine Strömung gemessen wird, liefern die Strömungswächter den Signalzustand „1“ (den booleschen Wert TRUE) an die Steuerung.
- Der thermische Motorschutz wird ebenfalls auf den Signalzustand „1“ abgefragt. Wenn der Motor nicht überlastet wurde, liefert der Öffnerkontakt (95, 96) des Motorschutzes den Signalzustand „1“, den booleschen Wert TRUE an die Steuerung.

Programm als AWL

```

LD      STROEM_PUMPE_2  (*Strömungswächter von Pumpe 2*)
OR      STROEM_PUMPE_3  (*Strömungswächter von Pumpe 3*)
AND     MERK_5           (*Pumpe 2 und Pumpe 3 sollen arbeiten*)
OR      MERK_4           (*Pumpe 1 und Pumpe 2*)
OR      MERK_6           (*Pumpe 1 und Pumpe 3*)
AND     THERM_MOT_SCH_1  (*Motorschutz*)
ST      PUMPE_1          (*Pumpe 1 arbeitet*)

```

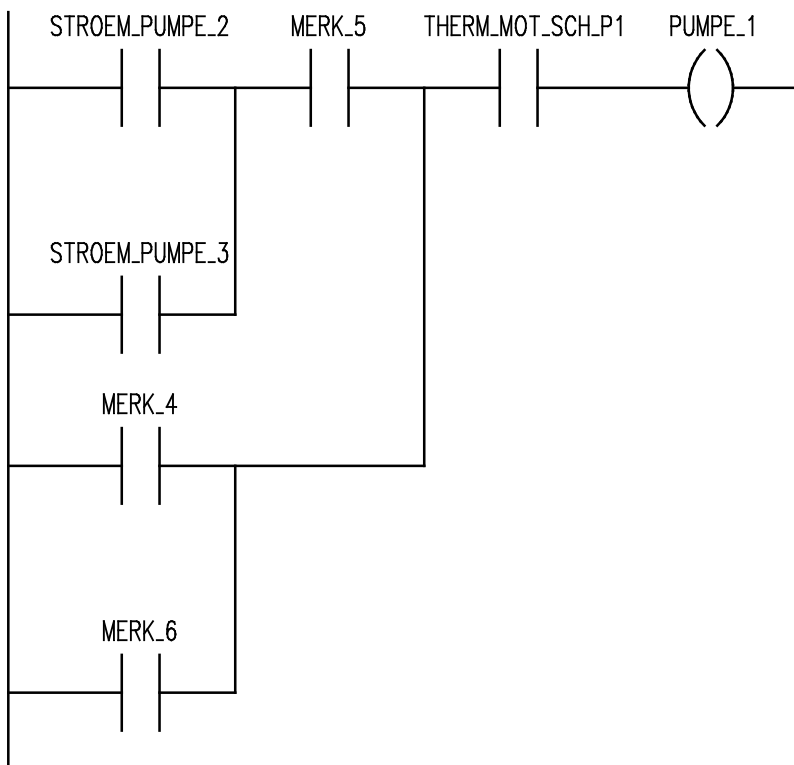
Programm in KOP-Darstellung

Abbildung 68 Steuerung von Pumpe 1

In allen Fällen ist folgende Variablendeklaration vorzunehmen:

```

VAR_INPUT
    STROEM_PUMPE_2, STROEM_PUMPE_3 : BOOL;
    THERM_MOT_SCH_P1 : BOOL;
END_VAR

```

```

VAR_OUTPUT
    PUMPE_1 : BOOL;
END_VAR

```

```

VAR
    MERK_4, MERK_5, MERK_6 : BOOL;
END_VAR

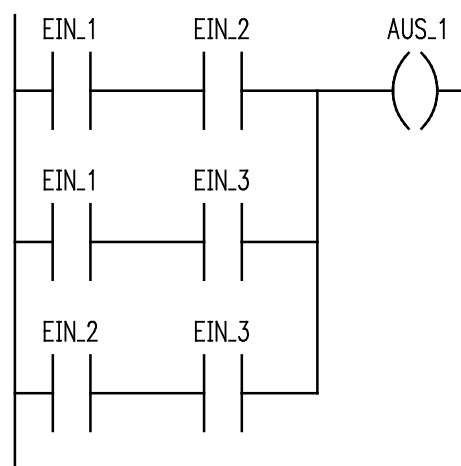
```

Hinweis

Die einzelnen am Markt befindlichen Programmiersysteme unterstützen die Variablendeklaration und übrigen auch die Zuordnung der Hardwareadressen (Ein- und Ausgänge) zu den Variablennamen in außerordentlich komfortabler Weise. In jedem Fall wesentlich anwenderfreundlicher, als in IEC 61131-3 beschrieben. Daher muss diesen Dingen bei der Grundlagenbearbeitung keine so hohe Bedeutung beigemessen werden.

Lehrbeispiel 7

Das Netzwerk ist in der Programmiersprache AWL darzustellen.

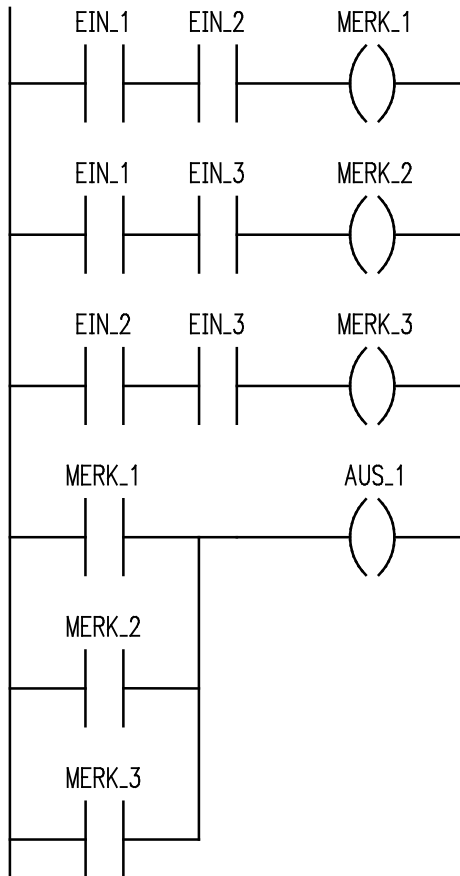


Das Netzwerk kann nicht „direkt“ programmiert werden. Entweder erfolgt die Programmierung mithilfe von **Klammern** oder **Merkern**.

Abbildung 69 Netzwerk

Programmierung mit Merkern

Die Ergebnisse der drei UND-Funktionen werden in Merkern zwischengespeichert. Es ergibt sich dann folgende Darstellung.



Die drei Merker werden anschließend ODER-verknüpft.

Die **AWL** lautet:

```
LD EIN_1
AND EIN_2
ST MERK_1
```

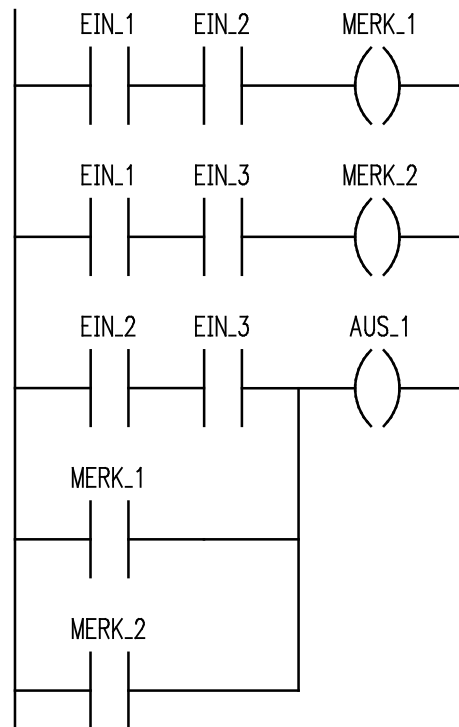
```
LD EIN_1
AND EIN_3
ST MERK_2
```

```
LD EIN_2
AND EIN_3
ST MERK_3
```

```
LD MERK_1
OR MERK_2
OR MERK_3
ST AUS_1
```

Abbildung 70 Verwendung von drei Merkern

Das Netzwerk kann auch mit zwei Merkern bearbeitet werden.



Anweisungsliste:

```
LD  EIN_1
AND EIN_2
ST  MERK_1
```

```
LD  EIN_1
AND EIN_3
ST  MERK_2
```

```
LD  EIN_2
AND EIN_3
OR  MERK_1
OR  MERK_2
ST  AUS_1
```

Abbildung 71 Verwendung von zwei Merkern

Programmierung mit Klammern:

```
LD  EIN_1
AND EIN_2
OR( EIN_1
AND EIN_3
)
OR( EIN_2
AND EIN_3
)
ST  AUS_1
```

2.2 Speicher

Speicher sind in der Steuerungstechnik immer dann erforderlich, wenn die **Befehlsausführungsdauer** länger als die Dauer der **Befehlsgabe** ist. Zum Beispiel kann eine kurze Betätigung eines Starttasters einen länger andauernden Betrieb eines Elektromotors bewirken.

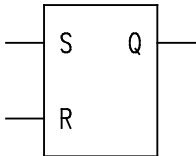


Abbildung 72 Speichersymbol

In der Steuerungstechnik ist der **RS-Speicher** von großer Bedeutung, wobei **zwischen vorrangigem Rücksetzen und vorrangigem Setzen** unterschieden wird.

RS-Speicher

Im Speichersymbol bedeutet:

- S** Setzeingang des Speichers
- R** Rücksetzeingang des Speichers
- Q** Speicherausgang

Ein zumindest kurzzeitig am Setzeingang S anliegender boolescher Wert TRUE (Signalzustand „1“) setzt den Speicherausgang Q dauerhaft. Ein zumindest kurzzeitig am Rücksetzeingang anliegender boolescher Wert TRUE setzt den Speicherausgang wieder zurück. Ein gesetzter Speicherausgang hat den booleschen Wert TRUE, ein rückgesetzter Speicherausgang demnach den booleschen Wert FALSE.

Eingang		Ausgang
S	R	Q
0	0	keine Änderung
1	0	1
0	1	0
1	1	unbestimmt

Tabelle 11 Funktion des Speichers

Besondere Beachtung verdient der Fall, wenn **gleichzeitig** $S = 1$ und $R = 1$. Dies ist eine **widersprüchliche Befehlsgabe**, da dem Speicher gleichzeitig „mitgeteilt“ wird:

- Setze den Ausgang Q ($S = 1$)
- Setze den Ausgang Q zurück ($R = 1$)

Keinesfalls darf es dem Zufall überlassen bleiben, welchen booleschen Wert der Ausgang bei **widersprüchlicher Befehlsgabe** annimmt. Daher wird zwischen **vorrangigem Rücksetzen** und **vorrangigem Setzen** unterschieden.

Vorrangiges Rücksetzen

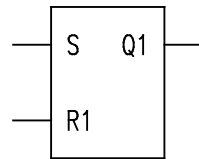


Abbildung 73 RS-Speicher mit vorrangigem Rücksetzen

Wenn gleichzeitig $S = 1$ und $R = 1$, nimmt der Ausgang den booleschen Wert FALSE („0“) an.

Bei widersprüchlicher Befehls-gabe setzt sich also der Rücksetzeingang durch. Dargestellt wird das vorrangige Rücksetzen im Symbol durch die Ziffer „1“ hinter dem „R“ (R1).

Vorrangiges Setzen

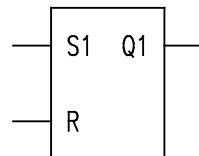


Abbildung 74 RS-Speicher mit vorrangigem Setzen

Wenn gleichzeitig $S = 1$ und $R = 1$, nimmt der Ausgang den booleschen Wert TRUE („1“) an. Bei widersprüchlicher Befehls-gabe setzt sich also der Setzeingang durch.

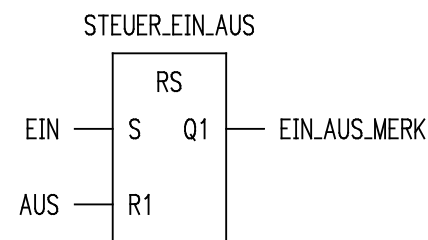


Abbildung 75 Speicherinstanz

Speicher zählen zu den **Standard-Funktionsbausteinen** nach IEC 61131-3 und werden **instanziert**. Unter **Instanzierung** versteht man, dass der Funktionsbaustein (hier Speicher) mit jeweils unterschiedlichen Parametern mehrfach genutzt werden kann. Die jeweilige Instanz wird durch den **Instanzennamen** (Instanzname) beschrieben.

Im Gegensatz zu Funktionen haben Funktionsbausteine ein „Gedächtnis“. Man sagt, sie haben **statische lokale Daten**, die über den Aufruf des Funktionsbausteins hinaus erhalten bleiben können.

Darstellung eines Speichers

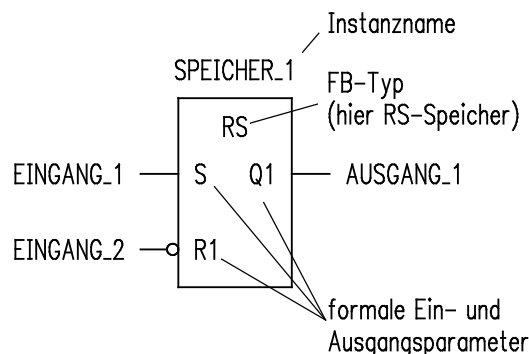


Abbildung 76 Speicherdarstellung

Der Rücksetzeingang ist in diesem und den folgenden Beispielen negiert, da es sich häufig um einen Öffner handelt.

Es ist folgende Variablendeklaration vorzunehmen:

```
VAR_INPUT
  EINGANG_1, EINGANG_2 : BOOL;
END_VAR
```

```
VAR_OUTPUT
  AUSGANG_1 : BOOL;
END_VAR
```

```
VAR
  SPEICHER_1 : RS;      (*vorrangiges Rücksetzen*)
END_VAR
```

Das Speicherverhalten (hier: vorrangiges Rücksetzen, RS) wird bei der Deklaration der FB-Instanz SPEICHER_1 bestimmt.

Bei einem Speicher mit vorrangigem Setzen würde die Deklarationszeile lauten:

```
SPEICHER_1 : SR;      (*vorrangiges Setzen*)
```

Anweisungsliste und Kontaktplan

Vorrangiges Rücksetzen

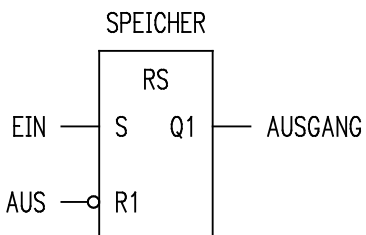


Abbildung 77 Vorrangiges Rücksetzen

```
LD   EIN
ST   SPEICHER.S      (*Setzeingang des Speichers*)
LDN  AUS
ST   SPEICHER.R1     (*Rücksetzeingang des Speichers, vorrangig*)

CAL  SPEICHER        (*Instanzaufruf*)

LD   SPEICHER.Q1     (*Speicherausgang*)
ST   AUSGANG
```

Vorrangiges Setzen

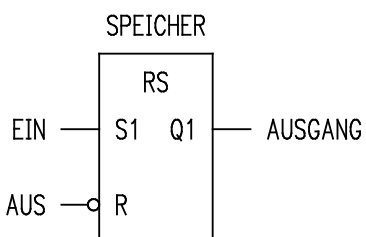


Abbildung 78 Vorrangiges Setzen

LD	EIN	
ST	SPEICHER.S1	(*Setzeingang des Speichers, vorrangig*)
LDN	AUS	
ST	SPEICHER.R	(*Rücksetzeingang des Speichers*)
CAL	SPEICHER	(*Instanzaufruf*)
LD	SPEICHER.Q1	(*Speicherausgang*)
ST	AUSGANG	

Den formalen Eingangsparametern des FB Speicher (S, R) werden die Aktualparameter (EIN, AUS) zugeordnet. Mit diesen Parametern wird die Instanz SPEICHER aufgerufen (CAL SPEICHER). Der boolesche Wert des formalen Ausgangsparameters Q1 wird dem Aktualparameter AUSGANG zugeordnet.

Vorrangiges Rücksetzen

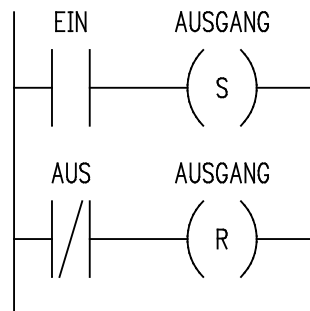


Abbildung 79 KOP-Darstellung, vorrangiges Rücksetzen

Vorrangiges Setzen

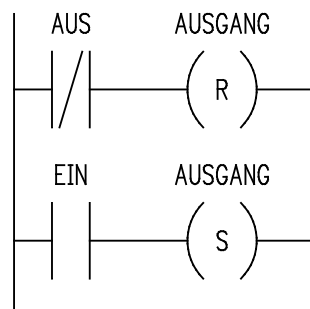


Abbildung 80 KOP-Darstellung, vorrangiges Setzen

Zu beachten ist die **Reihenfolge** von S und R bei der Kontaktplandarstellung.

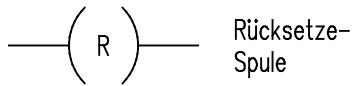
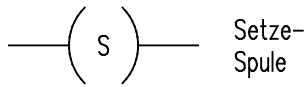
Darstellung von Spulen im Kontaktplan

Abbildung 81 Darstellung von Spulen

Operatoren der Speicherfunktion

- S** Setzt booleschen Operator auf „1“
R Setzt booleschen Operator auf „0“ zurück

Eingangsoperatoren von Funktionsbaustein „Speicher“

- S1, R** Funktionsbaustein-Typ SR (vorrangiges Setzen)
S, R1 Funktionsbaustein-Typ RS (vorrangiges Rücksetzen)

Lehrbeispiel 1

Dargestellt ist ein Steuerungsprogramm in der Programmiersprache FBS.

Das Programm soll als Anweisungsliste und als Kontaktplan dargestellt werden!

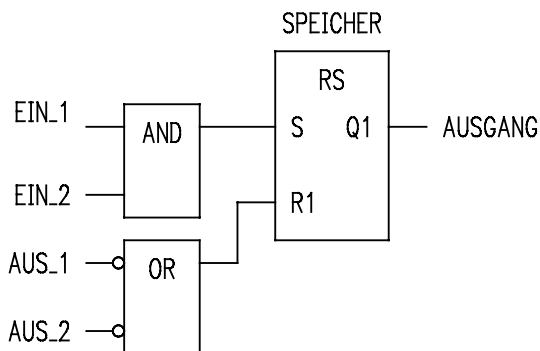


Abbildung 82 FBS-Darstellung

KOP

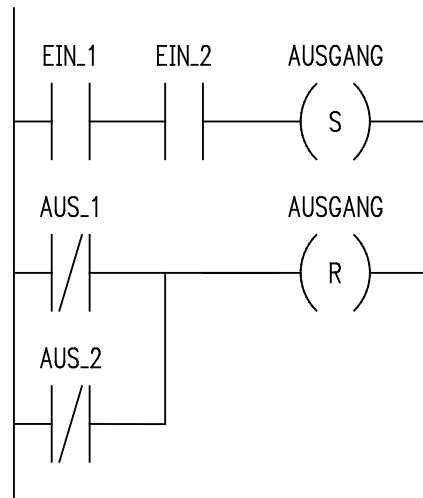


Abbildung 83 KOP-Darstellung

AWL

```
LD EIN_1
AND EIN_2
ST SPEICHER.S

LDN AUS_1
ORN AUS_2
ST SPEICHER.R1

CAL SPEICHER

LD SPEICHER.Q1
ST AUSGANG
```

Lehrbeispiel 2

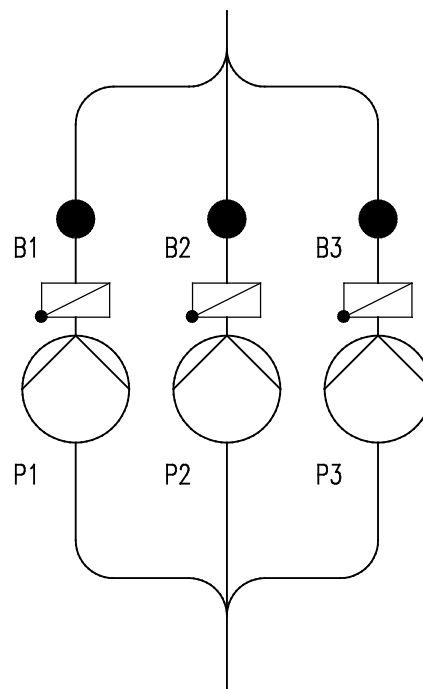


Abbildung 84 Pumpensteuerung

Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben. Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst.

Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal); wenn keine Strömung erkannt wird, somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung geöffnet werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten
 S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten
 S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten
 S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Für die Pumpensteuerung wurden drei Hilfsschütze verwendet, die jeweils die zwei gewünschten Pumpen in Betrieb nehmen. Diese Schütze werden in Selbsthaltung betrieben, d.h. sie haben Speicherverhalten.

S0	AUS	Austaster, Öffner
S1	P1_P2_EIN	Pumpe 1 und Pumpe 2 ein, Schließer
S2	P2_P3_EIN	Pumpe 2 und Pumpe 3 ein, Schließer
S3	P1_P3_EIN	Pumpe 1 und Pumpe 3 ein, Schließer
K4A	MERK_4	
K5A	MERK_5	
K6A	MERK_6	

Die Steuerungsaufgabe ist in den Programmiersprachen FBS, AWL und KOP darzustellen.

FBS

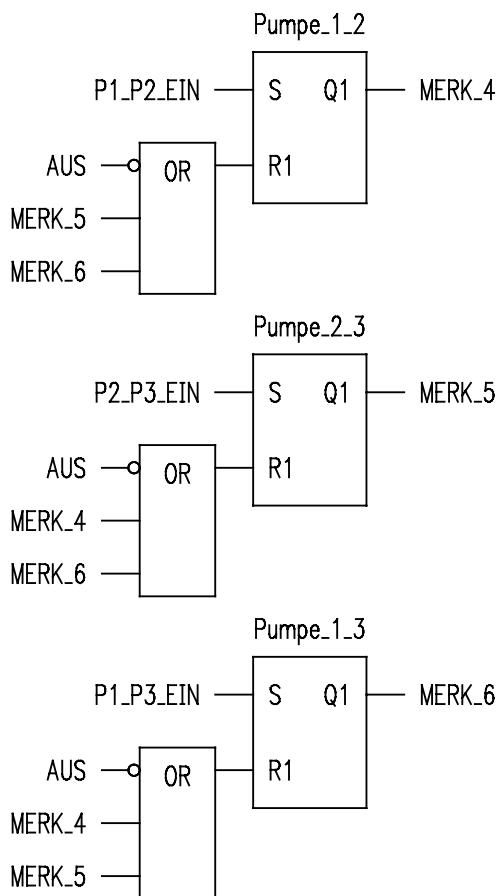


Abbildung 85 Pumpensteuerung, FBS-Darstellung

AWL

```
LD    P1_P2_EIN
ST    PUMPE_1_2.S
LDN   AUS
OR    MERK_5
OR    MERK_6
ST    PUMPE_1_2.R1
```

```
CAL   PUMPE_1_2
```

```
LD    PUMPE_1_2.Q1
ST    MERK_4
```

```
LD    P2_P3_EIN
ST    PUMPE_2_3.S
LDN   AUS
OR    MERK_4
OR    MERK_6
ST    PUMPE_2_3.R1
```

```
CAL   PUMPE_2_3
```

```
LD    PUMPE_2_3.Q1
ST    MERK_5
```

```
LD    P1_P3_EIN
ST    PUMPE_1_3.S
LDN   AUS
OR    MERK_4
OR    MERK_5
ST    PUMPE_1_3.R1
```

```
CAL   PUMPE_1_3
```

```
LD    PUMPE_1_3.Q1
ST    MERK_6
```

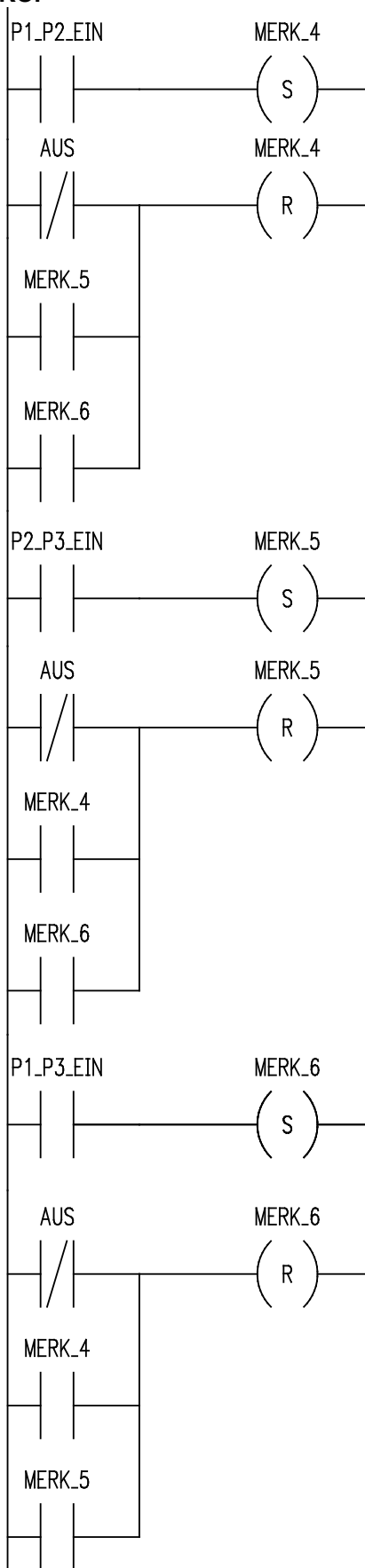
KOP

Abbildung 86 Kontaktplandarstellung der Pumpensteuerung

Nun ist die **Parameterversorgung** der einzelnen Speicherinstanzen ein wenig umständlich. Zumindest erfordert sie einige Anweisungszeilen. Einfacher und übersichtlicher ist da folgendes Verfahren.

```
CAL  PUMPE_1_2 (S := P1_P2_EIN, R1 := NOT AUS OR MERK_5 OR MERK_6)
LD   PUMPE_1_2.Q1
ST   MERK_4
```

Die Parameterversorgung erfolgt hier gemeinsam mit dem Instanzaufruf. Setz- und Rücksetzbedingungen werden durch Komma getrennt. Vorrangiges Rücksetzen (R1) ist gewährleistet.

Lehrbeispiel 3

Die Anweisungsliste nach Lehrbeispiel 2 ist wie oben gezeigt umzuschreiben.

```
CAL  PUMPE_1_2 (S:= P1_P2_EIN, R1 := NOT AUS OR MERK_5 OR MERK_6)
LD   PUMPE_1_2.Q1
ST   MERK_4
```

```
CAL  PUMPE_2_3 (S := P2_P3_EIN, R1 := NOT AUS OR MERK_4 OR MERK_6)
LD   PUMPE_2_3.Q1
ST   MERK_5
```

```
CAL  PUMPE_1_3 (S := P1_P3_EIN, R1 := NOT AUS OR MERK_4 OR MERK_5)
LD   PUMPE_1_3.Q1
ST   MERK_6
```

Zweifellos ist diese Form der AWL-Darstellung relativ übersichtlich.

2.3 Zeitverzögerungen

Für den verzögerten Beginn oder die verzögerte Beendigung von Aktionen sind **Zeitverzögerungen** (Zeitglieder, Timer) erforderlich. Dabei wird grundsätzlich zwischen **Einschaltverzögerung** und **Ausschaltverzögerung** unterschieden.

Auch Zeitglieder sind nach IEC 61131-3 **Standard-Funktionsbausteine**, die **instanziert** werden müssen.

Einschaltverzögerung

Das Zeitglied wird gestartet, die Zeit läuft ab. Wenn die Zeit abgelaufen ist, wird die gewünschte Aktion ausgelöst.

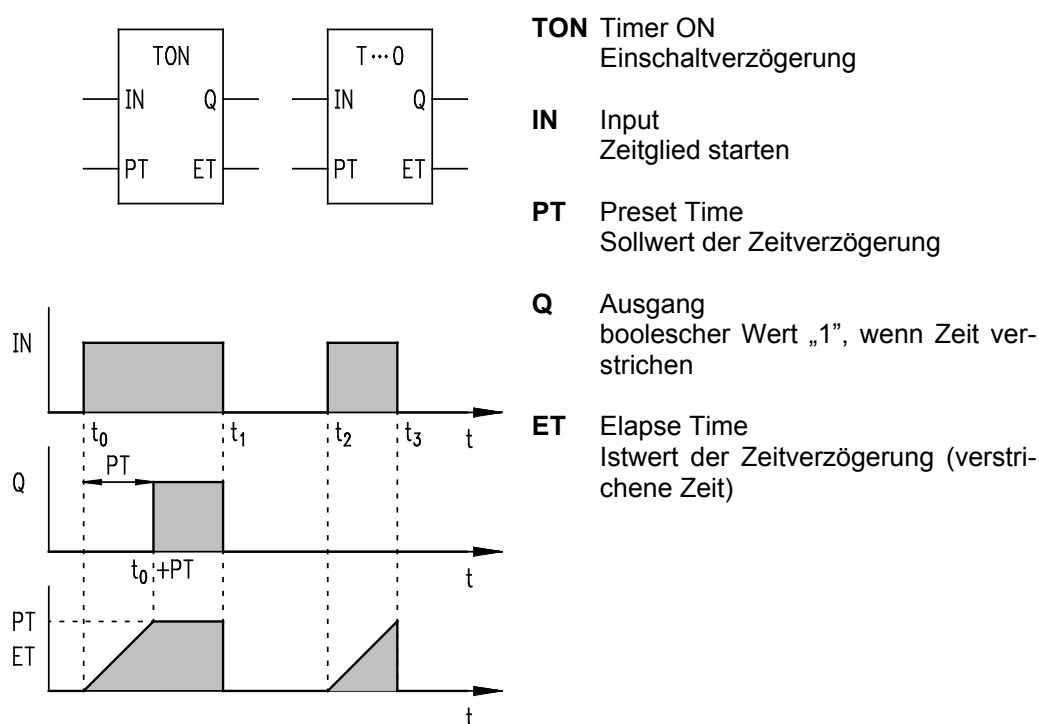


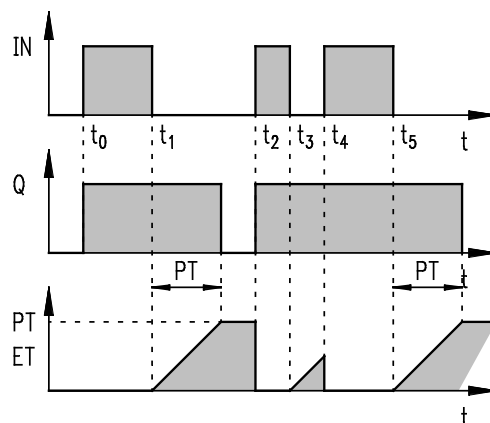
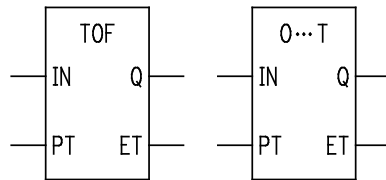
Abbildung 87 Einschaltverzögerung TON

Dem **Zeitdiagramm** können wichtige Informationen entnommen werden:

- Damit die Zeitverzögerung gestartet und die Zeit ablaufen kann, muss der Eingang IN den booleschen Wert „1“ führen.
- Wenn die gewünschte Zeit (PT) verstrichen ist, nimmt der Ausgang Q den booleschen Wert „1“ an.
- Wenn der boolesche Wert am Eingang (IN) wieder „0“ ist, nimmt auch der Ausgang Q wieder den booleschen Wert „0“ an.
- Wenn der boolesche Wert „1“ nicht so lange am Eingang IN anliegt wie die Zeitdauer PT vorgibt, wird der Ausgang Q nicht den booleschen Wert „1“ annehmen.

Ausschaltverzögerung

Beim Start des Zeitgliedes wird die gewünschte Aktion **unverzöglich** ausgelöst. Wenn die Zeit verstrichen ist, wird die ausgelöste Aktion wieder ausgeschaltet.

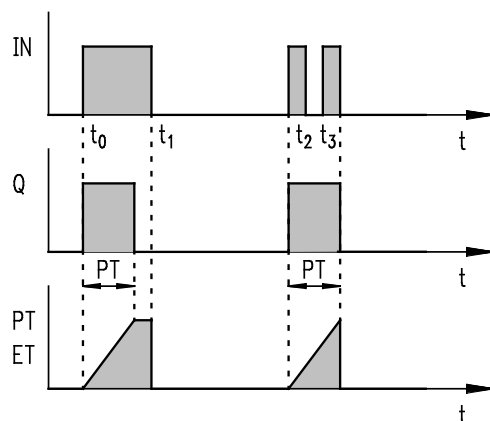
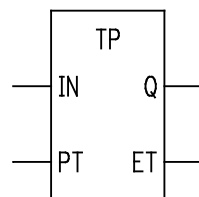


- Der boolesche Wert „1“ am IN-Eingang ruft den booleschen Wert „1“ am Q-Ausgang hervor.
- Sobald am IN-Eingang wieder der boolesche Wert „0“ anliegt, wird die Zeit PT ablaufen. Wenn PT abgelaufen ist, nimmt der Ausgang Q wieder den booleschen Wert „0“ an.
- Wenn während der Zeit PT am Eingang IN wieder der boolesche Wert „1“ angelegt wird, behält der Ausgang Q den booleschen Wert „1“.

Abbildung 88 Ausschaltverzögerung TOF

Puls

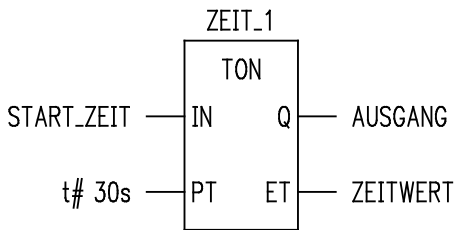
Schon ein kurzer **Puls** am IN-Eingang bringt den Ausgang Q für die Zeitdauer PT auf den booleschen Wert „1“.



Zu beachten ist, dass während der Laufzeit von PT ein weiterer Puls am Eingang IN keine Verlängerung des Ausgangssignals ergibt.

Abbildung 89 Puls

Programmierung von Zeitverzögerungen



Zeitverzögerungen (Timer) sind Standard-Funktionsbausteine, die bei Bedarf, versehen mit dem jeweiligen Parametern, aufgerufen werden können.

Abbildung 90 Instanz ZEIT_1

Aufruf eines Timers in AWL

Deklaration der Variablen für ZEIT_1

```
VAR
  START_ZEIT : BOOL;    (*Eingang*)
  AUSGANG : BOOL;       (*Ausgang*)
  ZEITWERT : TIME;       (*Ausgang-Istwert*)
  ZEIT_1 : TON;          (*Standard-FB, Einschaltverzögerung*)
END_VAR
```

1. Möglichkeit (Deklaration wie oben)

Zunächst werden die Eingangsparameter des Timers geladen und gespeichert.

```
LD   t#30s                (*Zeitverzögerung 30 Sekunden*)
ST   ZEIT_1.PT             (*Zeitverzögerungswert laden*)
LD   START_ZEIT           (*Zeitverzögerung starten*)
ST   ZEIT_1.IN            (*wenn START_ZEIT = 1*)

CAL  ZEIT_1                (*Aufruf der Instanz*)

LD   ZEIT_1.Q              (*Wenn Zeit abgelaufen ist*)
ST   AUSGANG               (*nimmt AUSGANG den booleschen Wert „1“ an*)
LD   ZEIT_1.ET             (*Der Istwert des Zeitwertes*)
ST   ZEITWERT              (*wird der Variablen ZEITWERT zugewiesen*)
```

Zu beachten ist, dass die **Einschaltverzögerung TON** dem Timer ZEIT_1 im Deklarationsteil zugewiesen wird.

2. Möglichkeit (Deklaration wie oben)

Der Aufruf des Timer erfolgt mit der in Klammern geschriebenen Liste der Ein- und Ausgangsparameter.

```
CAL  ZEIT_1(IN := START_ZEIT, PT := t#30s)  (*Aufruf*)
LD   ZEIT_1.Q                                (*Ausgangsparameter*)
ST   AUSGANG
LD   ZEIT_1.ET
ST   ZEITWERT
```

3. Möglichkeit (Deklaration wie oben)

Aufruf unter Verwendung der Eingangsvariablen als Operatoren.

```
LD    t#30s                (*Eingangsparameter*)
PT    ZEIT_1
LD    START_ZEIT
IN    ZEIT_1

LD    ZEIT_1.Q              (*Ausgangsparameter*)
ST    AUSGANG
LD    ZEIT_1.ET
ST    ZEITWERT
```

Hinweise:

- Voraussetzung für das ordnungsgemäße Zeitverhalten der hier beschriebenen Timer ist, dass die Zykluszeit des SPS-Programms klein gegenüber der Zeitdauer PT ist, falls die Zeit im Zyklus nur einmal aufgerufen wird.
- Das Verhalten des Zeitgliedes bei Veränderung von PT während der Zeitoperation ist abhängig von der verwendeten SPS-Hardware, d.h. herstellerabhängig.

Lehrbeispiel 1

Stellen Sie das Programm als Anweisungsliste dar!

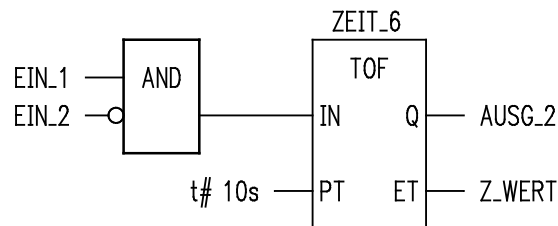


Abbildung 91 Instanz ZEIT_6

Variablendeklaration

```
VAR
    EIN_1, EIN_2 : BOOL;
    AUSG_2 : BOOL;
    Z_WERT : TIME;
    ZEIT_6 : TOF;
END_VAR
```

1. Möglichkeit

```
LD    t#10s
ST    ZEIT_6.PT
LD    EIN_1
ANDN  EIN_2
ST    ZEIT_6.IN

CAL    ZEIT_6

LD    ZEIT_6.Q
ST    AUSG_2
LD    ZEIT_6.ET
ST    Z_WERT
```


2. Möglichkeit

```

CAL   ZEIT_6 (IN := EIN_1 AND NOT EIN_2, PT := t#10s)
LD     ZEIT_6.Q
ST     AUSG_2
LD     ZEIT_6.ET
ST     Z_WERT

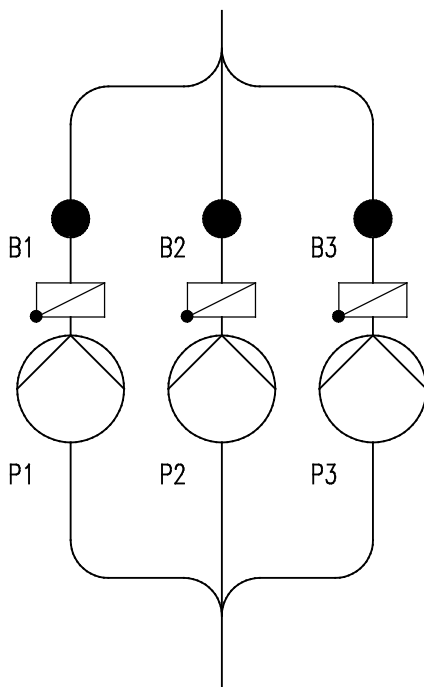
```

3. Möglichkeit

```

LD     t#10s
PT     ZEIT_6
LD     EIN_1
ANDN   EIN_2
IN     ZEIT_6
LD     ZEIT_6.Q
ST     AUSG_2
LD     ZEIT_6.ET
ST     Z_WERT

```

Lehrbeispiel 2

Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben.

Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst. Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal); wenn keine Strömung erkannt wird, somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung betätigt werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten
 S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten
 S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten
 S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Bei der Pumpensteuerung besteht das Problem, dass die Reservepumpe beim Einschalten auch anläuft, bis in beiden Arbeitspumpensträngen Strömung erkannt wird. Dieses Verhalten ist nicht zu akzeptieren. Daher soll beim Einschalten der Pumpen eine Zeit gestartet werden, und erst nach Ablauf dieser Zeit soll die jeweilige Reservepumpe eingeschaltet werden können.

Die Lösung kann folgendermaßen aussehen:

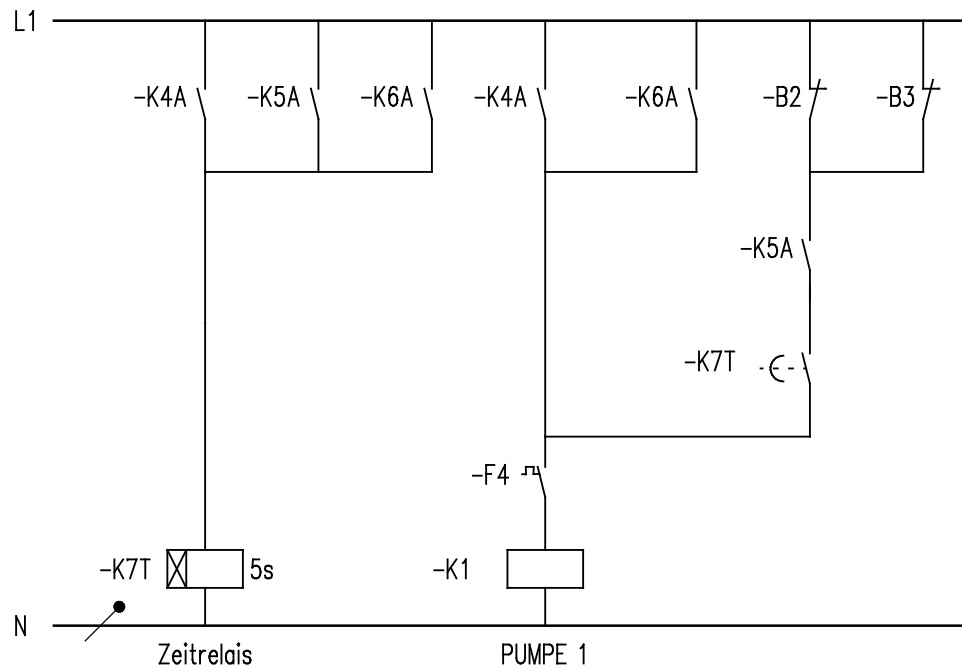


Abbildung 92 Anlaufzeit der Pumpen (hier Pumpe 1 dargestellt)

Annahme: Pumpe 1 ist Reservepumpe. Arbeitspumpen sind Pumpe 2 und Pumpe 3. Pumpe 2 und Pumpe 3 werden mithilfe des Hilfsschützes K5A speichernd eingeschaltet. K5A startet die Wartezeit K7T. Erst wenn diese Zeit abgelaufen ist, kann Pumpe 1 als Reservepumpe arbeiten. Vorausgesetzt natürlich, Pumpe 2 oder Pumpe 3 baut keine Strömung auf. Die Wartezeit muss so groß gewählt werden, dass die beiden jeweiligen Arbeitspumpen in dieser Zeit Strömung aufbauen können. Die jeweilige Reservepumpe läuft dann nicht an. Gewählt wird eine Zeit von 5 Sekunden.

Unter der Annahme, dass das Zeitrelais nur einen Wechslerkontakt (also eine Schließfunktion) hat, kann die dargestellte Gesamtschaltung eingesetzt werden.

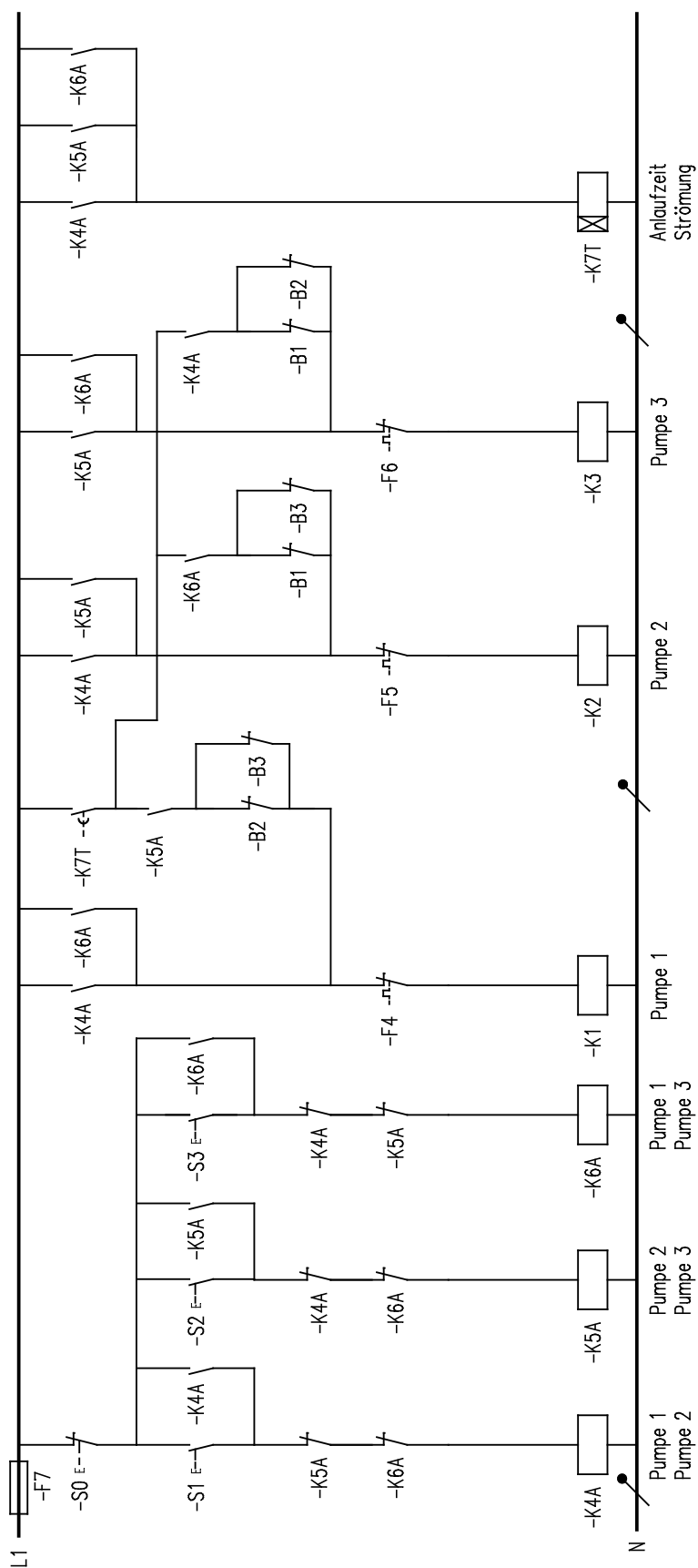


Abbildung 93 Gesamtschaltung

Erst wenn der Schließerkontakt von K7T nach Ablauf der Zeit geschlossen ist, wird die jeweilige Ansteuerung der Reservepumpe freigegeben.

Für den in Abbildung 92 dargestellten Steuerungsausschnitt soll nun das Programm in den Sprachen FBS und AWL entwickelt werden.

Programm in FBS

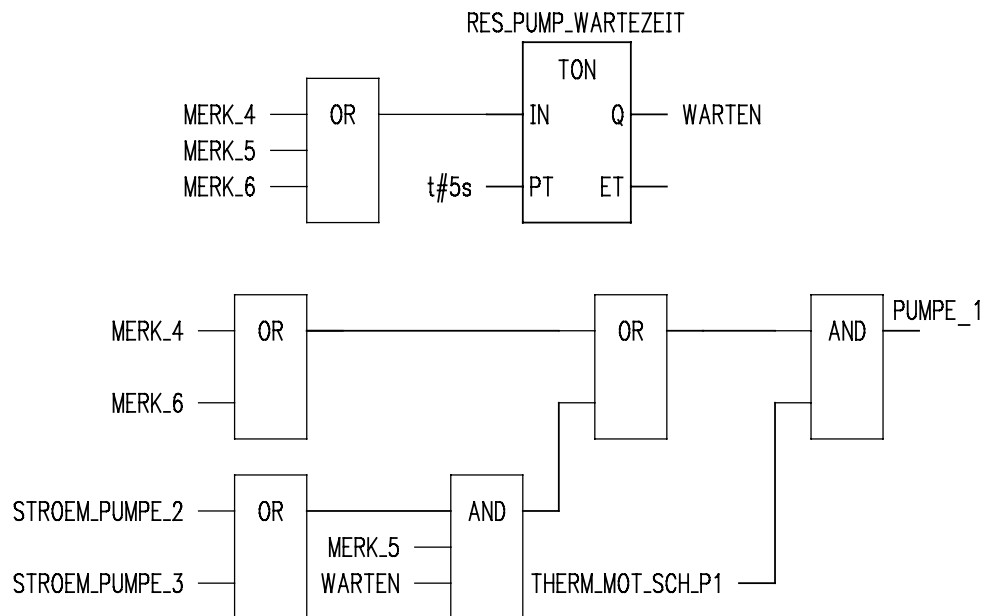


Abbildung 94 Anlaufzeit der Pumpen (hier Pumpe 1 dargestellt)

Programm als AWL

```

CAL RES_PUMP_WARTEZEIT (IN := MERK_4 OR MERK_5 OR MERK_6,
  PT := t#5s)
LD RES_PUMP_WARTEZEIT.Q
ST WARTEN

LD STROEM_PUMPE_2
OR STROEM_PUMPE_3
AND MERK_5
AND WARTEN
OR MERK_4
OR MERK_6
AND THERM_MOT_SCH_P1
ST PUMPE_1
  
```

Variablendeklaration:

```

VAR_INPUT
  STROEM_PUMPE_2, STROEM_PUMPE_3 : BOOL;
  THERM_MOT_SCH_P1 : BOOL;
END_VAR

VAR_OUTPUT
  PUMPE_1 : BOOL;
END_VAR
  
```

```

VAR
  MERK_4, MERK_5, MERK_6 : BOOL;
  WARTEN : BOOL;
  RES_PUMP_WARTEZEIT :TON;
END_VAR

```

2.4 Zähler

Durch Programmierung von **Zähloperationen** können Zählaufgaben direkt vom Mikroprozessor des Automatisierungsgerätes ausgeführt werden. Dabei kann **vorwärts** und **rückwärts** gezählt werden.

Auch Zähler werden (wie Speicher und Zeitglieder) als Standard-Funktionsbausteine **instanziiert**.

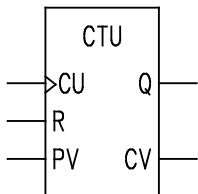


Abbildung 95 Vorwärtszähler

Vorwärtszähler (Aufwärtszähler)

Eine steigende Flanke am **Impulseingang** CU erhöht den aktuellen Zählerstand um den Wert 1.

Der boolesche Wert „1“ am **Löscheingang** R setzt den momentanen Zählerstand auf Null. Der Zähler kann dadurch für eine weitere Zählaufgabe vorbereitet werden.

Über den **PV-Eingang** wird der gewünschte Zählwert (Zählersollwert) eingegeben. Der Zählwert muss ganzzahlig sein.

Wenn am CU-Eingang so viele Impulse gezählt wurden, wie der Zählwert am PV-Eingang vorgibt, wird der **Zählausgang** Q den booleschen Wert „1“ annehmen. Dieser boolesche Wert bleibt erhalten, bis der Zähler gelöscht wird (R-Eingang).

Am **CV-Ausgang** kann der momentane Zählwert (Istwert) abgefragt werden. Es kann also festgestellt werden, wie viele Impulse bereits am CU-Eingang „eingelaufen“ sind. Auch der Istwert ist ganzzahlig.

CTU	CounTer Up	Aufwärtszähler
CU	Count Up	aufwärts zählen
R	Reset	Zähler löschen
PV	Preset Value	Zählwert, Sollwert
Q	Quit	Zählerausgang
CV	Current Value	augenblicklicher Zählwert, Istwert
LD	LoaD	Zählwert/Sollwert laden
CTD	CounTer Down	Abwärtszähler
CTUD	CounTer Up Down	Auf-/Abwärtszähler

Tabelle 12 Bezeichnungen beim Zähler

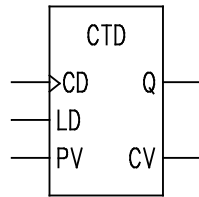


Abbildung 96
Rückwärtszähler

Rückwärtszähler (Abwärtszähler)

Beim booleschen Wert „1“ am **LD-Eingang** wird der Zähler-sollwert (PV-Eingang) übernommen.

Bei steigender Flanke am **Impulseingang CD** wird der aktuelle Zählerstand um 1 vermindert.

Wenn der Zählwert bis auf Null zurückgezählt wurde, nimmt der **Ausgang Q** den booleschen Wert „1“ an.

Durch erneutes Anlegen des booleschen Wertes „1“ am **LD-Eingang** wird der Zähler für einen weiteren Zählvorgang vorbereitet.

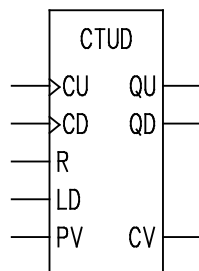


Abbildung 97
Vor- und Rückwärtszähler

Vor- und Rückwärtszähler

Beim **Vor- und Rückwärtszähler** (Auf- und Abwärtszähler) liegt eine Kombination der beiden Zähler-Grundtypen vor.

Bedeutung der Formalparameter (Ein- und Ausgänge) siehe oben.

Arbeitsweise des Vorwärtszählers (Prinzip)

Annahme:

Sollwert PV = 3

Istwert CV = 0 (Zähler gerade gelöscht und wieder R = 0)

1. Signalwechsel an CU: PV = 3
CV = 1

2. Signalwechsel an CU: PV = 3
CV = 2

3. Signalwechsel an CU: PV = 3
CV = 3

PV = CV (Sollwert = Istwert); der Zählerausgang Q nimmt den booleschen Wert „1“ an:

Q = 1

„1“-Signal an R (Löscheingang):

R = 1

Q = 0

CV = 0

Der Zähler wird gelöscht und für einen neuen Zählvorgang vorbereitet

Zähler für neuen Zählvorgang freigeben:

R = 0

CV = 0

PV = 3

Lehrbeispiel 1

Der dargestellte Aufwärtszähler ist als Anweisungsliste darzustellen!

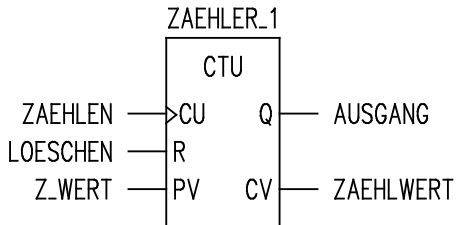


Abbildung 98 Vorwärtszähler, Aufwärtszähler

```

LD  ZAEHLEN
ST  ZAEHLER_1.CU

LD  LOESCHEN
ST  ZAEHLER_1.R

LD  Z_WERT
ST  ZAEHLER_1.PV

CAL ZAEHLER_1

LD  ZAEHLER_1.Q
ST  AUSGANG

LD  ZAEHLER_1.CV
ST  ZAEHLWERT

```

Selbstverständlich müssen nicht sämtliche Ein- und Ausgänge „beschaltet“ sein. Die jeweilige Steuerungsaufgabe bestimmt, was tatsächlich benötigt wird.

Zu erkennen ist, dass die Programmierung der Standard-Funktionsbausteine **Speicher**, **Timer** und **Zähler** deutliche Gemeinsamkeiten aufweist.

In der Deklaration wird der Aufwärtszähler bestimmt.

```

VAR
    ZAEHLER_1 : CTU;
    Z_WERT : INT;
END_VAR

```

Lehrbeispiel 2

Der dargestellte Abwärtszähler ist als Anweisungsliste darzustellen!

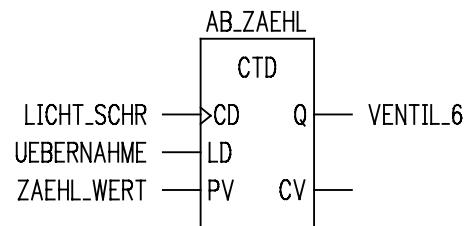


Abbildung 99 Rückwärtszähler, Abwärtszähler

```
LD  LICHT_SCHR
ST  AB_ZAEHL.CD
```

```
LD  UEBERNAHME
ST  AB_ZAEHL.LD
```

```
LD  ZAEHL_WERT
ST  AB_ZAEHL.PV
```

```
CAL AB_ZAEHL
```

```
LD  AB_ZAEHL.Q
ST  VENTIL_6
```

```
VAR
    AB_ZAEHL : CTD;
    ZAEHL_WERT : INT;
END_VAR
```


Lehrbeispiel 3

Der dargestellte Auf-/Abwärtszähler ist als Anweisungsliste darzustellen!

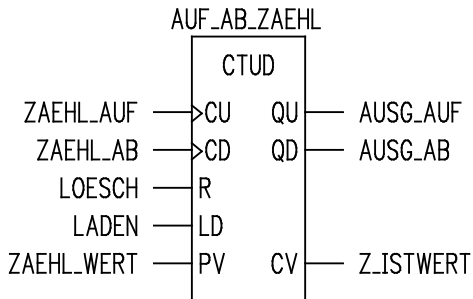


Abbildung 100 Auf-/Abwärtszähler

```

LD  ZAEHL_AUF
ST  AUF_AB_ZAEHL.CU

LD  ZAEHL_AB
ST  AUF_AB_ZAEHL.CD

LD  LOESCH
ST  AUF_AB_ZAEHL.R

LD  LADEN
ST  AUF_AB_ZAEHL.LD

LD  ZAEHL_WERT
ST  AUF_AB_ZAEHL.PV

CAL AUF_AB_ZAEHL

LD  AUF_AB_ZAEHL.QU
ST  AUSG_AUF

LD  AUF_AB_ZAEHL.QD
ST  AUSG_AB

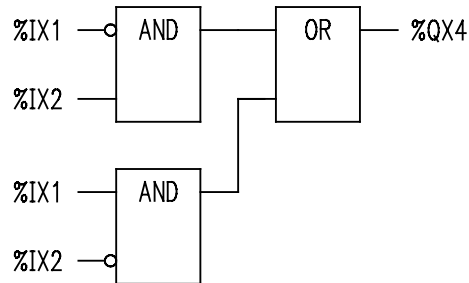
LD  AUF_AB_ZAEHL.CV
ST  Z_ISTWERT

```

Aufgaben

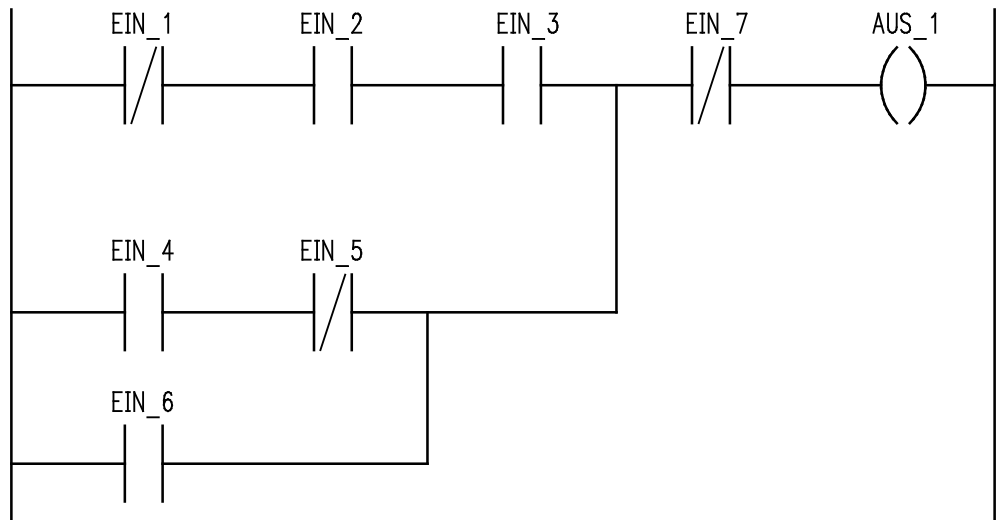
Aufgabe 1

Stellen Sie das Steuerungsprogramm als Anweisungsliste und als Kontaktplan dar!



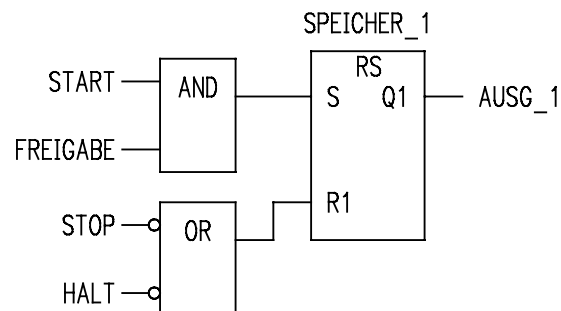
Aufgabe 2

Stellen Sie das Programm als Anweisungsliste dar!



Aufgabe 3

Entwickeln Sie die zugehörige Anweisungsliste!



Aufgabe 4

Erläutern Sie die Wirkungsweise des Programms!

```
VAR
    START : BOOL;
    AUS : BOOL;
    ZEIT_1 : TON;
END_VAR
```

```
LD t#60s
ST ZEIT_1.PT
LD START
ST ZEIT_1.IN
CAL ZEIT_1
LD ZEIT_1.Q
ST AUS
```

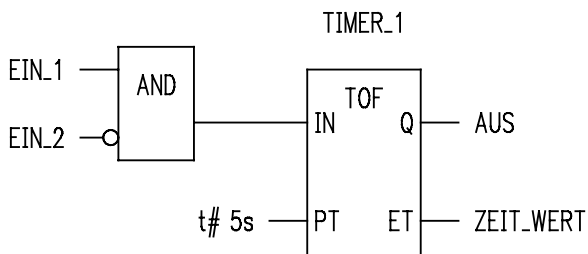
Aufgabe 5

Es gibt zwei weitere Darstellungsmöglichkeiten für das Programm nach Aufgabe 4.

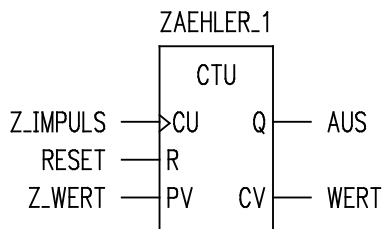
Stellen Sie das Programm in diesen Formen dar!

Aufgabe 6

Stellen Sie das Programm als Anweisungsliste (3 Möglichkeiten) dar!

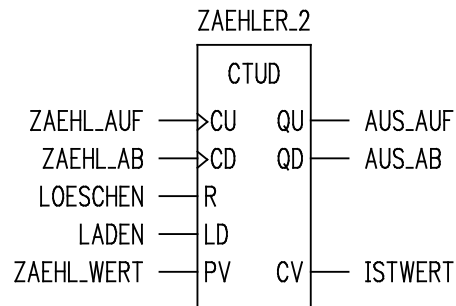
Aufgabe 7

Stellen Sie das Programm als Anweisungsliste dar!



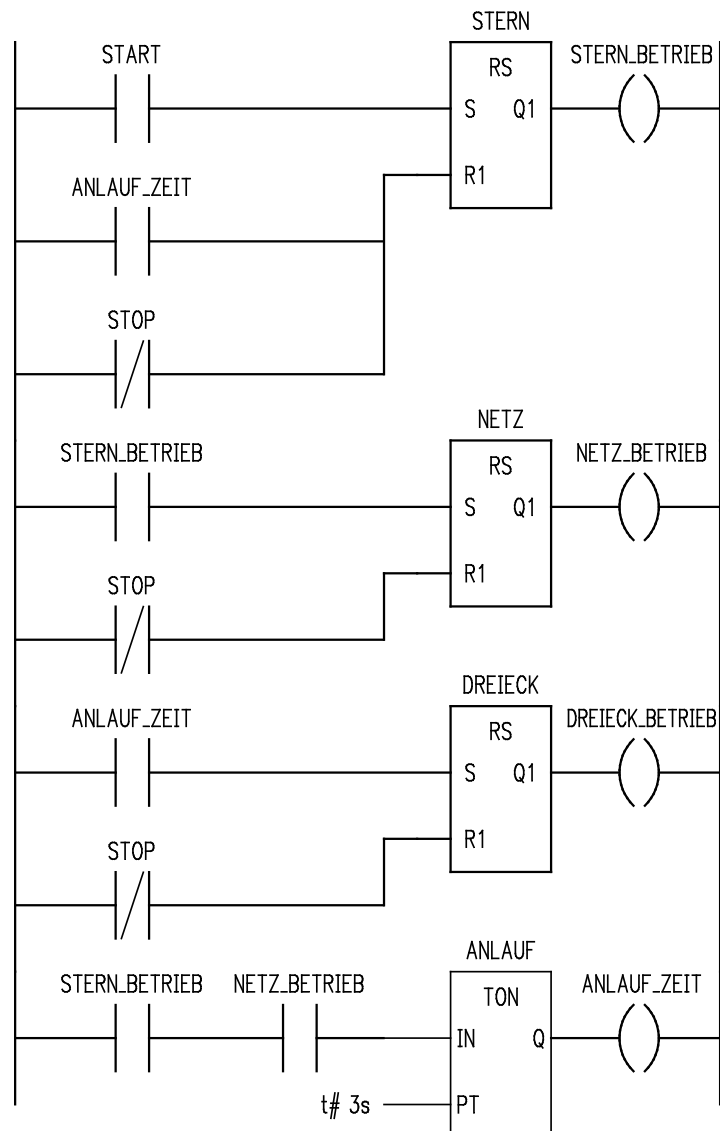
Aufgabe 8

Stellen Sie ZAEHLER_2 als Anweisungsliste dar!



Aufgabe 9

Stellen Sie das Programm als Anweisungsliste dar! Nehmen Sie auch die Variablen-deklaration vor!



Aufgabe

Nach der Erarbeitung der Grundlagen erstellen Sie zu dem technischen Problem „Säge mit Absaugvorrichtung“ eine Lösung in Funktionsbaustein-Sprache (FBS)!

Stellen Sie das Programm auch als KOP und AWL dar!

Erweiterung des Projektes 1

Zusätzlich ist eine NOT-AUS-Abschaltung vorzusehen, bei der die Säge und Absaugvorrichtung sofort ausgeschaltet werden.

Dazu gehört folgende Zuordnung:

Betriebsm.	Ein-/Ausgang	Symb. Name	Kommentar
S3	E3	AUS_2	NOT-AUS-Taster, Öffner

Tabelle 13 Zuordnungsliste

Ergänzen Sie die vorherigen Lösungen, um diese Funktion zu realisieren!

**Realisierung
Projekt 1
„Säge mit
Absaugvorrichtung“**

Lernbereich

3 Realisierung einer Steuerung durch eine Ablaufsprache (AS)

Steuerungstechnische Abläufe sind z.B. in vielen Produktionsmaschinen oder Verarbeitungsprozessen zu finden.

Ziel dieses Lernbereichs ist die Vermittlung einer standardisierten Beschreibungssprache für eine Ablaufsteuerung.

Durch die Darstellung eines typischen Steuerungsablaufes am Beispiel einer Verladeanlage werden die einzelnen Ablaufschritte und deren gesteuerte Abfolge erläutert.

Nach der Erarbeitung der erforderlichen Funktionen und deren normierte Darstellung wird als Abschluss des Lernbereichs das Projekt 2 "Verladeanlage" gelöst.

Projekt 2: Verladeanlage

Ein Kübelwagen soll zwischen Leerstation und Füllstation pendeln.

Funktionsablauf

Die Anlage wird über die Taster S1 und S2 ein und ausgeschaltet. Nach dem Einschalten soll der Kübelwagen zuerst in die Leerstation fahren und danach seinen Pendelbetrieb aufnehmen.

Das Leerventil (Y1) und das Füllventil (Y2) sind Magnetventile mit Federrückstellung und sollen zeitverzögert öffnen.

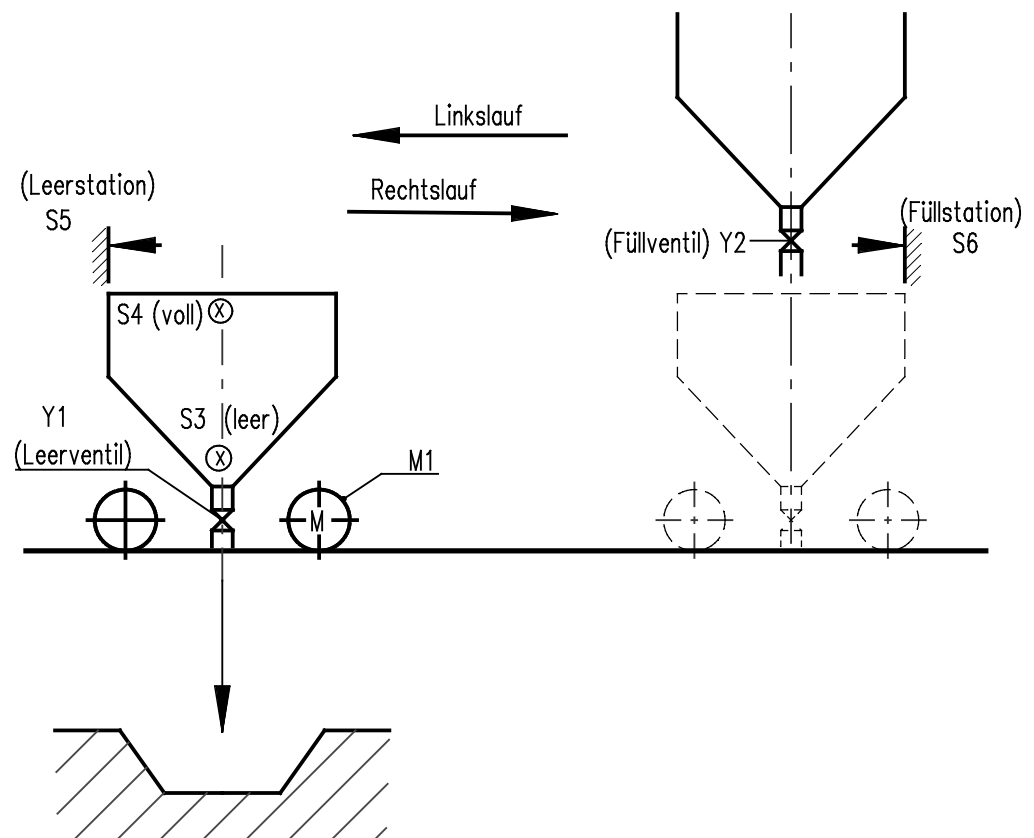


Abbildung 101 Verladeanlage

Betriebsm.	Ein-/Ausgang	Symbol. Name	Kommentar
S1	E1	EIN	EIN-Taster (Schließer)
S2	E2	AUS	AUS-Taster (Öffner)
S3	E3	LEER	S3 = 1 - Wagen leer S3 = 0 - Wagen nicht leer
S4	E4	VOLL	S4 = 1 - Wagen voll S4 = 0 - Wagen nicht voll
S5	E5	WALEER	S5 = 1 - Wagen in der Leerstation S5 = 0 - Wagen nicht in der Leerstation
S6	E6	WAFUELL	S6 = 1 - Wagen in der Füllstation S6 = 0 - Wagen nicht in der Füllstation
Y1	A1	VENTIL_1	A1 = 1 - Leerventil auf A1 = 0 - Leerventil zu
Y2	A2	VENTIL_2	A2 = 1 - Füllventil auf A2 = 0 - Füllventil zu
K1	A3	RECHTS	A3 = 1 - Wagen fährt nach rechts A3 = 0 - keine Wirkung
K2	A4	LINKS	A4 = 1 - Wagen fährt nach links A4 = 0 - keine Wirkung

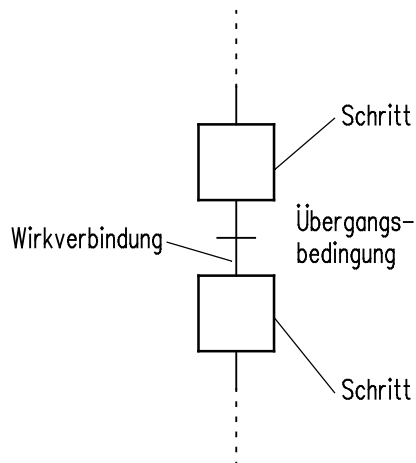
Tabelle 14 Zuweisungstabelle

Für dieses steuerungstechnische Problem ist eine Lösung im Form einer Ablaufsteuerung geeignet.

3.1 Normgerechte Darstellung von Ablaufsteuerungen

Bei der **Ablaufsprache** nach IEC 61131-3 wird zwischen der **grafischen** und der **textuellen** Variante unterschieden. Die **Ablaufsprache** (AS) ist den anderen Programmiersprachen übergeordnet. Mit ihrer Hilfe können umfangreiche Aufgabenstellungen in kleine überschaubare Einheiten zerlegt werden. Zumindest gilt dies für die grafische Variante, die in der Praxis auch überwiegend angewendet wird.

Die Ablaufsprache kann als Weiterentwicklung der Schrittkettenprogrammierung angesehen werden. Sie ist somit auch besonders für Steuerungsaufgaben mit schrittweisem Zustandsverhalten geeignet.



Die **Ablaufsteuerung** ist eine Steuerung mit zwangsweise **schrittweisem Ablauf**, bei dem der Übergang von einem Schritt auf den programmgemäß folgenden Schritt abhängig von **Übergangsbedingungen** erfolgt. Dabei werden **Schritte** und **Übergänge** durch **Wirkverbindungen** miteinander verbunden. Zur allgemeinen Verständlichkeit ist eine **normgerechte Darstellung** zwingend erforderlich.

Abbildung 102 Darstellung von Schritten und Übergangsbedingungen

Lehrbeispiel 1

Drei Umwälzpumpen sollen nacheinander in der Reihenfolge Pumpe 1, Pumpe 2, Pumpe 3 eingeschaltet werden können. Nach dem Einschaltvorgang sollen sämtliche Pumpen gemeinsam ausgeschaltet werden können.

Der Steuerungsablauf kann wie folgt beschrieben werden:

Taster S1 betätigt...

- Pumpe 1 einschalten

Taster S2 betätigt...

- Pumpe 2 einschalten

Taster S3 betätigt...

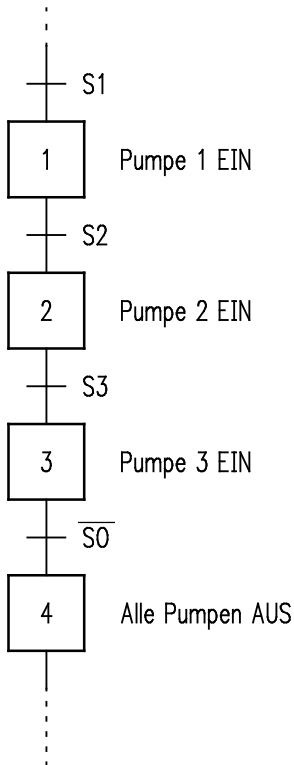
- Pumpe 3 einschalten

Taster S0 betätigt...

- Alle Pumpen ausschalten

Deutlich erkennbar ist die **klare Strukturierung** der Aufgabe. Die gesamte Steuerungsaufgabe besteht aus einer Reihe von Teilaufgaben, die stets in der gleichen Reihenfolge nacheinander abgearbeitet werden.

Nun kann die rein sprachliche Formulierung der Steuerungsaufgabe allerdings zu Missverständnissen führen. Die normgerechte grafische Darstellung nach IEC 61131-3 ist da schon eine wesentliche Verbesserung.



Es ist zu beachten, dass es sich bei S0 um einen Öffner handelt, der auf den Signalzustand „0“ abgefragt wird. Wenn der Öffner betätigt wird, liefert er den Signalzustand „0“ an die Steuerung. S0 ist daher zu negieren, was durch den übergesetzten Strich symbolisiert wird (S0̄).

Abbildung 103 Darstellung von Schritten und Transitionen

Darstellung von Schritten

Schritte (engl.: steps) beschreiben die unterschiedlichen **Beharrungszustände** einer Steuerung. Zu einem bestimmten Zeitpunkt kann ein Schritt entweder **gesetzt** (active) oder **nicht gesetzt** (inactive) sein. Der Schritt hat **Speicherverhalten** und ist einem bestimmten Prozessabschnitt funktionell zugeordnet. Prozessabschnitte ergeben sich aus technologischen oder programmbedingten Gründen. Der Schritt wird grafisch durch ein Rechteck (bevorzugt quadratisch) dargestellt.

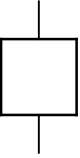
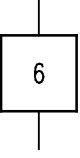
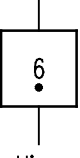
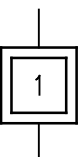
	Allgemeines Schrittsymbol Bevorzugt quadratisch; grundsätzlich jedoch beliebig.
	Schritte werden gekennzeichnet; die Kennzeichnung wird in das Schrittsymbol eingetragen. Hier: Schritt 6
 Hier: Schritt 6 gesetzt	Ein Punkt im Schrittsymbol besagt, dass der Schritt bei einem bestimmten Prozesszustand gesetzt wird. Es ist zu beachten, dass der Punkt nur der Kennzeichnung dient und nicht zum Schrittsymbol gehört.
	Darstellung eines Anfangsschrittes (hier Schrittnummer 1). Anfangsschritte (initial steps) werden zu Beginn des Steuerungsprozesses gesetzt. Sie kennzeichnen das Anfangsverhalten der Steuerung.

Tabelle 15 Darstellung von Schritten

Der **Anfangsschritt** wird auch als **Initialisierungsschritt** bezeichnet. Der Anfangsschritt (Initialisierungsschritt) kann ein beliebiger Schritt innerhalb der Ablaufkette sein. Wenn er keinen Vorgängerschritt hat, entfällt die oben eingehende Linie.

Wirkverbindungen

Wirkverbindungen (engl.: directed links), auch **Wirkungslinien** genannt, verbinden die einzelnen Schritte miteinander. Grundsätzlich können Wirkungslinien **senkrecht** oder **waagrecht** verlaufen; in Ausnahmefällen auch **schräg**, wenn dadurch die Übersichtlichkeit gesteigert werden kann.

Der **Ablauf** erfolgt definitionsgemäß **von oben nach unten** (eventuell von **links nach rechts**). Kann diese Definition nicht eingehalten werden, ist die Ablaufrichtung durch **Pfeile** zu kennzeichnen.



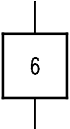
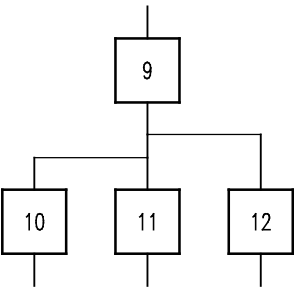
	Wirkverbindung mit definitionsgemäßem Ablauf von oben nach unten
	Wirkverbindung mit Ablauf von unten nach oben
 Schritt 12 Seite 4	Wenn eine Wirkverbindung unterbrochen werden muss, so wird die nächste Schritt-nummer angeben, ferner die Nummer des Blattes, auf dem dieser Schritt aufzufinden ist.
	Kreuzungen von Wirkverbindungen müssen vermieden werden, wenn sie sich auf den gleichen Ablauf beziehen.

Tabelle 16 Darstellung von Wirkverbindungen

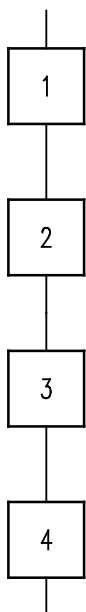


Abbildung 104 Wirkverbindungen; zwischen den Schritten 1 bis 4 senkrecht von oben nach unten

Hinweis:

Eine Voraussetzung für das Setzen eines Schrittes ist, dass sein unmittelbarer Vorgänger bereits gesetzt ist. Dadurch wird die Zwangsfolge bei der Schrittbe-arbeitung bewirkt. Zum Beispiel kann der 4. Schritt nur gesetzt werden, wenn sein unmittelbarer Vorgänger (der 3. Schritt) bereits gesetzt ist.

Übergänge und Übergangsbedingungen

Durch den **Übergang** (engl.: transition) werden zwei Schritte miteinander verbunden. Dargestellt wird der Übergang durch einen Strich, der in die Wirkungsverbindung zwischen den Schritten gezeichnet wird.

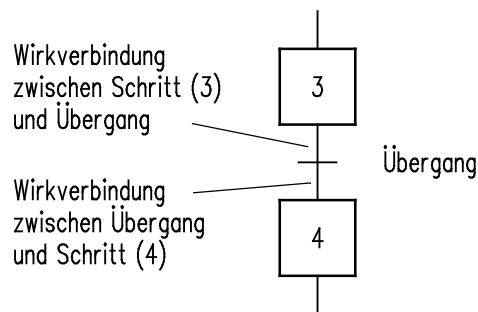


Abbildung 105 Darstellung von Übergängen

Dem Übergang wird eine **Übergangsbedingung** (engl.: transition condition) zugeordnet, die erfüllt sein kann oder nicht.

Für binäre Variable gilt:

0 oder **FALSE** Übergangsbedingung nicht erfüllt
1 oder **TRUE** Übergangsbedingung erfüllt

Übergangsbedingungen können dargestellt werden durch **Symbole**, **boolesche Gleichungen** oder **Texte**.

	<p>Die Übergangsbedingung ist erfüllt, wenn $a = 1$ ODER $b = 1 \text{ UND } c = 1$</p>
	<p>Die Übergangsbedingung ist erfüllt, wenn a geschlossen ODER b geschlossen UND c geschlossen</p>
<p>Boolesche Gleichung</p>	<p>Beachten Sie: $+$ kennzeichnet die ODER-Funktion \bullet kennzeichnet die UND-Funktion</p>
<p>Texte</p>	<p>a geschlossen ODER b geschlossen UND c geschlossen</p>

Tabelle 17 Darstellung von Übergangsbedingungen

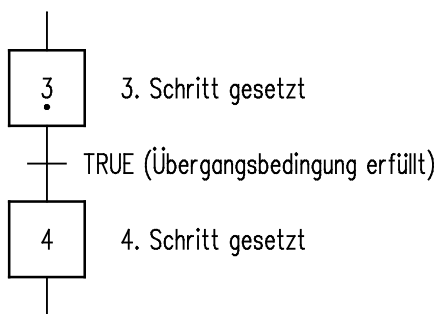


Abbildung 106 Setzen des Folgeschrittes

Der jeweilige **Folgeschritt** wird gesetzt, wenn

- der Vorgängerschritt gesetzt (aktiv) ist

UND

- die Übergangsbedingung erfüllt ist.

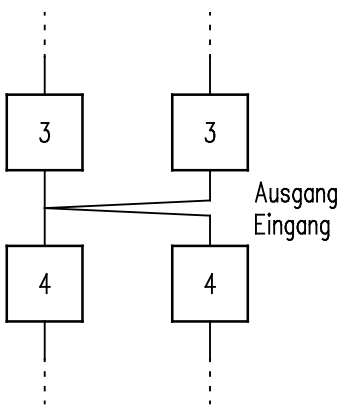


Abbildung 107 Wirkverbindung zwischen Schritten

Durch den gesetzten (aktiven) 3. Schritt wird der **Übergang** zum 4. Schritt freigegeben. Ein Übergang kann nur ausgelöst werden, wenn er freigegeben und die Übergangsbedingung erfüllt ist. Wurde der Übergang ausgelöst, wird der unmittelbar folgende Schritt gesetzt und der vorherige Schritt zurückgesetzt.

Trennt man die Wirkverbindungen zwischen den Schritten 3 und 4 gedanklich auf, so stellt der obere Teil der Wirkverbindung den Ausgang des 3. Schrittes und der untere Teil den Eingang des 4. Schrittes dar.

Lehrbeispiel 2

Drei Pumpen sollen nacheinander in Betrieb genommen werden können. Die Pumpe 1 wird mit S1, die Pumpe 2 mit S2 und die Pumpe 3 mit S3 eingeschaltet. Wenn alle drei Pumpen arbeiten, können mit S0 sämtliche Pumpen ausgeschaltet werden.

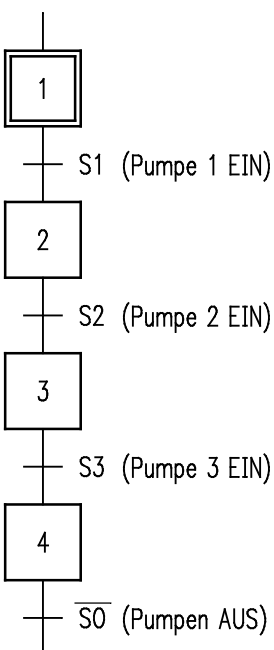


Abbildung 108 Ablaufkette mit vier Schritten

Die Ablaufkette besteht aus 4 Schritten (einschließlich Anfangsschritt).

Die Schritte 2 bis 4 schalten jeweils eine Pumpe speichernd ein. Wenn der Öffner S0 betätigt wird, erfolgt der Übergang von Schritt 4 nach Schritt 1. Der Öffner S0 muss auf den Signalzustand „0“ abgefragt werden; er ist daher zu negieren.

3.2 Programmierung von Schritten und Übergängen

Für die Programmierung von Ablaufsteuerungen wird die **Ablaufsprache (AS)** verwendet. Die Ablaufsprache **umfasst Hilfsmittel zur Gliederung einer SPS-Programm-Organisationseinheit** in eine Anzahl von Schritten und Transitionen, die in geeigneter Weise miteinander verbunden sind.

Jedem Schritt können eine oder mehrere **Aktionen** zugeordnet werden. Ist ein Teil einer Programm-Organisationseinheit in Ablaufsprache organisiert, dann muss die gesamte Programm-Organisationseinheit so aufgebaut werden.

Schritte

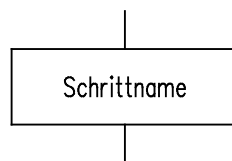


Abbildung 109 Schrittdarstellung

Der Schritt wird **grafisch** durch einen Block dargestellt, der einen **Schrittname** enthält.

In **Textform** kann ein Schritt durch den Ausdruck

```
STEP
...
END_STEP
```

dargestellt werden

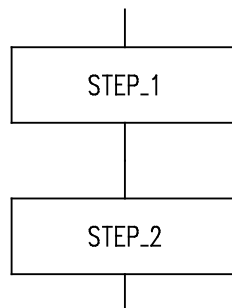


Abbildung 110 Verbindung zwischen Schritten

Die **Wirkverbindungen** (gerichtete Verbindung) können durch senkrechte Linien dargestellt werden, die oben und unten am Schritt angesetzt sind.

In **Textform** können Wirkungslinien (gerichtete Verbindungen) durch den Ausdruck

```
TRANSITION
...
END_TRANSITION
```

dargestellt werden.

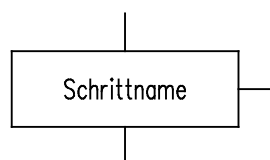


Abbildung 111 Schrittmarker

Ein Schritt ist entweder **gesetzt (aktiv)** oder **nicht gesetzt (inaktiv)**. Zur Beschreibung wird ein **Schrittmarker** verwendet, der durch

Schrittname.X

dargestellt wird.

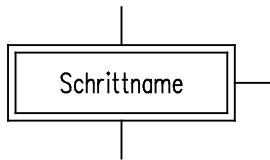


Abbildung 112 Anfangsschritt (Initialisierungsschritt)

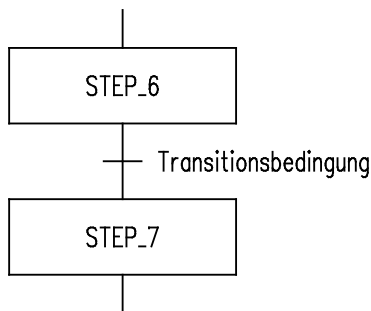
Wenn der Schritt **gesetzt** (aktiv) ist, hat der Schrittmerker den booleschen Wert **TRUE**, wenn er **nicht gesetzt** ist (inaktiv), den booleschen Wert **FALSE**.

Jedes AS-Netzwerk oder die entsprechende Textform **muss** einen **Anfangsschritt** (Initialisierungsschritt) haben.

Darstellung des **Anfangsschrittes** in **Textform**:

```
INITIAL_STEP
...
END_STEP
```

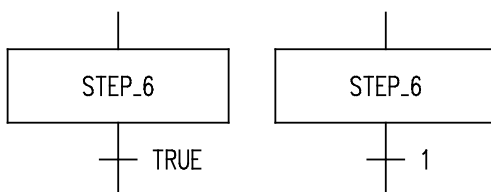
Übergänge (Transitionen)



Die **Transition** wird durch einen waagrechten Strich, der die senkrechte Wirkungsline durchkreuzt, dargestellt.

Jeder Transition muss eine **Transitionsbedingung** zugeordnet werden. Die Transitionsbedingung ist das Ergebnis eines **booleschen Ausdrucks** und kann den Wert **TRUE** oder **FALSE** haben.

Abbildung 113 Darstellung der Transitionsbedingung



- Transitionsbedingung nicht wahr: FALSE
- Transitionsbedingung wahr: TRUE

Wenn eine Transitionsbedingung **stets wahr** ist, wird sie durch TRUE bzw. „1“ dargestellt

Abbildung 114 Transitionsbedingung stets wahr

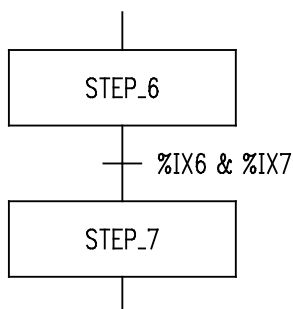


Abbildung 115 Transitionsbedingung in ST

Darstellung von Transitionsbedingungen nach IEC 61131-3

Transitionsbedingung in grafischer Form

- Die Transitionsbedingung kann durch einen **booleschen Ausdruck in ST-Sprache** auf der **rechten Seite** der Verbindung angegeben werden.

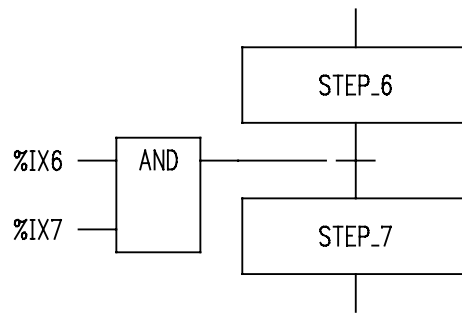


Abbildung 116 Transitionsbedingung in FBS

- Die Transitionsbedingung kann durch ein **Netzwerk in FBS** auf der **linken Seite** der Verbindung dargestellt werden.

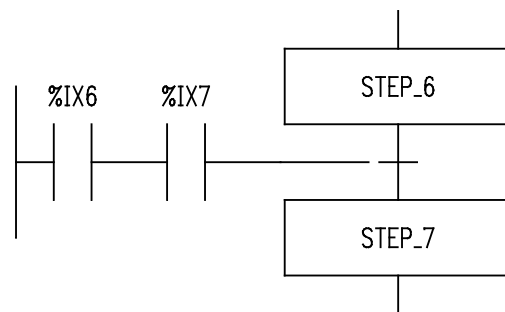


Abbildung 117 Transitionsbedingung in KOP

- Die Transitionsbedingung kann durch ein **Kontaktplan-Netzwerk** auf der **linken Seite** der Verbindung dargestellt werden.

- Die Transitionsbedingung kann durch einen **Konnektor** dargestellt werden:

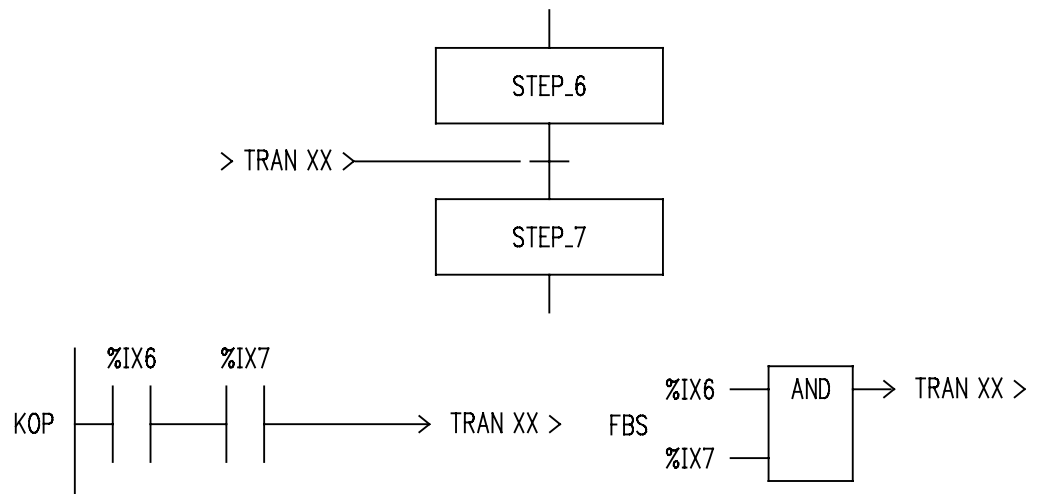


Abbildung 118 Verwendung eines Konnektors

Transitionsbedingung in Textform

Die Transitionsbedingung wird in **ST-Sprache** beschrieben durch

```

TRANSITION
(*Schritt-Rumpf*)
END_TRANSITION
    
```


Hierbei ist zu beachten:

- Den Schlüsselwörtern TRANSITION FROM folgt der **Schrittname des vorausgehenden Schrittes** (bzw. der vorausgehenden Schritte).
- Dem Schlüsselwort TO folgt der **Schrittname des nachfolgenden Schrittes** (bzw. der nachfolgenden Schritte).
- Dem Zuweisungsoperator „:=“ folgt ein **boolescher Ausdruck in ST-Sprache**, der die **Transitionsbedingung** beschreibt.
- Den Abschluss bildet das Schlüsselwort END_TRANSITION.

Lehrbeispiel 1

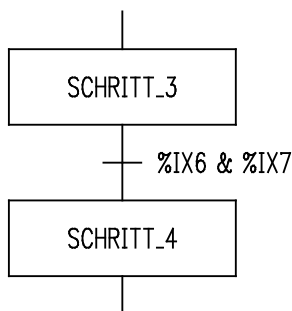


Abbildung 119 Transition in ST

```

STEP SCHRITT_3 : END_STEP
  TRANSITION FROM SCHRITT_3 TO SCHRITT_4
  := %IX6 & %IX7;
END_TRANSITION
STEP SCHRITT_4 : END_STEP
  
```

Der Übergang von SCHRITT_3 nach SCHRITT_4 erfolgt, wenn %IX6 & %IX7 den booleschen Wert TRUE ergibt.

Die Transitionsbedingung wird in **AWL-Sprache** dargestellt durch:

```

TRANSITION
(*Schrittrumpf*)
END_TRANSITION
  
```

Hierbei ist zu beachten:

- Den Schlüsselwörtern TRANSITION FROM folgt der **Schrittname des vorausgehenden Schrittes** (bzw. der vorausgehenden Schritte).
- Dem Schlüsselwort TO folgt der **Schrittname des nachfolgenden Schrittes** (bzw. der nachfolgenden Schritte). Danach wird ein Doppelpunkt „:“ geschrieben.
- In einer neuen Zeile werden die Anweisungen in AWL-Sprache geschrieben (Transitionsbedingung).
- Den Abschluss bildet das Schlüsselwort END_TRANSITION.

Lehrbeispiel 2

Übergangsbedingung wie Lehrbeispiel 1

Stellen Sie die Transitionsbedingung in AWL-Sprache dar!

```
STEP SCHRITT_3 : END_STEP
  TRANSITION FROM SCHRITT_3 TO SCHRITT_4:
    LD %IX6
    AND %IX7
    END_TRANSITION
STEP SCHRITT_4 : END_STEP
```

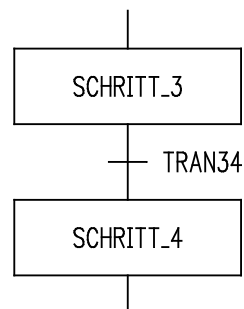


Abbildung 120 Transitionsname

Die Transitionsbedingung wird bei Verwendung eines **Transitionsnamens** auf der rechten Seite der Wirkungsline (gerichtete Verbindung) geschrieben.

Der **Transitionsname** (Bezeichner) muss in

```
TRANSITION
...
END_TRANSITION
```

eingebunden werden.

Transitionsbedingung in **ST-Sprache**:

```
TRANSITION TRAN34
:= %IX6 & %IX7;
END_TRANSITION
```

Transitionsbedingung in **AWL-Sprache**:

```
TRANSITION TRAN34:
LD %IX6
AND %IX7
END_TRANSITION
```

Transitionsbedingung in **KOP-Sprache**:

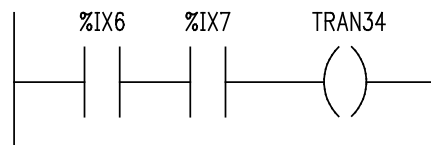


Abbildung 121 Transition in KOP

Transitionsbedingung in **FBS**:

```
TRANSITION TRAN34:
  %IX6 — [ AND ] —> TRAN34 >
  %IX7 — [ AND ]
END_TRANSITION
```

Abbildung 122 Transition in FBS

Lehrbeispiel 3

Programmieren Sie die Ablaufkette in Textform sowohl in ST- als auch in AWL-Sprache!

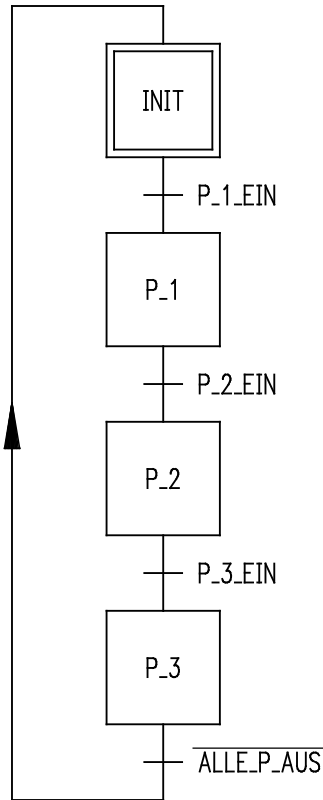


Abbildung 123 Ablaufkette

Programmierung in ST-Sprache

```

INITIAL_STEP INIT : END_STEP
  TRANSITION FROM INIT TO P_1
    := P_1_EIN;
  END_TRANSITION

STEP P_1 : END_STEP
  TRANSITION FROM P_1 TO P_2
    := P_2_EIN;
  END_TRANSITION

STEP P_2 : END_STEP
  TRANSITION FROM P_2 TO P_3
    := P_3_EIN;
  END_TRANSITION

STEP P_3 : END_STEP
  TRANSITION FROM P_3 TO INIT
    := NOT ALLE_P_AUS;
  END_TRANSITION
  
```

Beachten Sie:

- Die Schrittnamen lauten P_1 (Pumpe 1), P_2, P_3 und INIT.
- Die Transitionen sind durch Semikolon abzuschließen.

Programmierung in AWL-Sprache

```

INITIAL_STEP INIT : END_STEP
  TRANSITION FROM INIT TO P_1:
    LD P_1_EIN
  END_TRANSITION

STEP P_1 : END_STEP
  TRANSITION FROM P_1 TO P_2:
    LD P_2_EIN
  END_TRANSITION

STEP P_2 : END_STEP
  TRANSITION FROM P_2 TO P_3:
    LD P_3_EIN
  END_TRANSITION

STEP P_3 : END_STEP
  TRANSITION FROM P_3 TO INIT:
    LDN ALLE_P_AUS
  END_TRANSITION
  
```

Beachten Sie,

- dass in AWL-Sprache die Elemente
TRANSITION FROM... TO... :
durch einen Doppelpunkt abzuschließen sind.

3.3 Strukturen von Schrittketten

Ablaufketten bestehen aus einem ständigen Wechsel von **Schritten** und **Transitionen**. Es dürfen niemals zwei Schritte oder zwei Transitionen **direkt** aufeinander folgen.

Die Ablaufkette ist der einfachste Fall (die einfachste Struktur). Sie ist aus einer Folge von Schritten und Transitionen aufgebaut.

Nun ist es aber auch möglich, dass auf einen Schritt folgend **eine von mehreren Ketten** (alternativ) durchlaufen wird bzw. **mehrere Ketten gleichzeitig** (simultan) durchlaufen werden. Dies ermöglicht die relativ einfache Bearbeitung komplexer Steuerungsaufgaben.

Lineare Struktur (einfache Schrittkette)

Ständiger Wechsel **Schritt/Transition**:

Der Schritt STEP_3 wird in den aktiven Zustand versetzt, wenn der Schritt STEP_2 aktiv ist und die Transitionsbedingung %IX1 erfüllt ist.

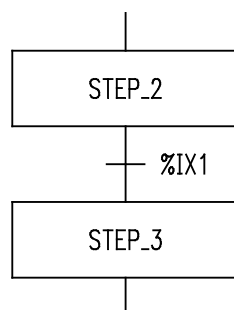


Abbildung 124 Lineare Struktur

Lineare Schrittketten (Ablaufketten) wurden bereits ausführlich bearbeitet. Daher braucht an dieser Stelle nichts mehr hinzugefügt zu werden.

Verzweigung

Bei der **Verzweigung** sind mehrere Abläufe möglich.

Durch den **Stern** wird angegeben, dass die Transitionen von **links nach rechts** bearbeitet werden.

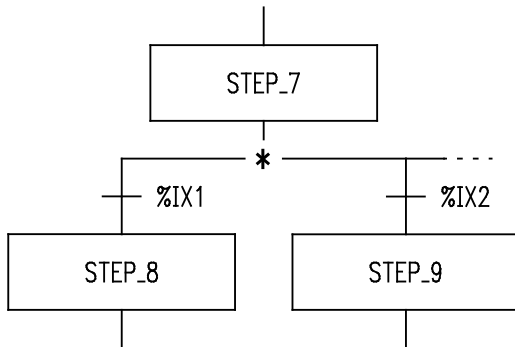


Abbildung 125 Verzweigung

Der Übergang von STEP_7 nach STEP_8 findet nur statt, wenn STEP_7 aktiv ist und Transitionsbedingung %IX1 erfüllt ist.

Der Übergang von STEP_7 nach STEP_9 findet nur statt, wenn STEP_7 aktiv ist und %IX2 = TRUE und %IX1 = FALSE ist.

Verzweigung mit Vorrang

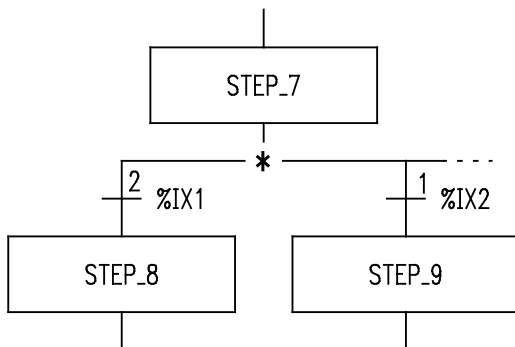


Abbildung 126 Verzweigung mit Vorrang

Die Schrittkettenzweige werden nummeriert. Der Zweig mit der **niedrigsten Ziffer** hat bei der Bearbeitung **Vorrang**.

Der Übergang von STEP_7 nach STEP_9 findet nur statt, wenn STEP_7 aktiv ist und die Transitionsbedingung %IX2 = TRUE ist. Der Übergang von STEP_7 nach STEP_8 findet nur statt, wenn STEP_7 aktiv ist und %IX1 = TRUE UND %IX2 = FALSE ist.

Verzweigung mit Ausschluss

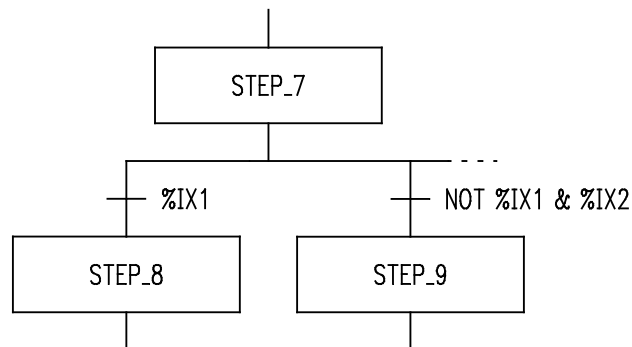


Abbildung 127 Verzweigung mit Ausschluss

Der Anwender hat selbst dafür zu sorgen, dass die Reihenfolge bei der Zweigbearbeitung eingehalten wird. Dies geschieht dadurch, dass die **Transitionsbedingungen sich gegenseitig ausschließen**.

Ein Übergang von STEP_7 nach STEP_9 findet nur statt, wenn %IX1 = FALSE und %IX2 = TRUE ist.

Zusammenführung

Am Ende jedes Schrittkettenzweiges steht eine **Transitionsbedingung**. Wenn die Transitionsbedingung wahr und der vorangehende Schritt aktiv ist, erfolgt der Übergang auf den auf die **Zusammenführung** folgenden Schritt.

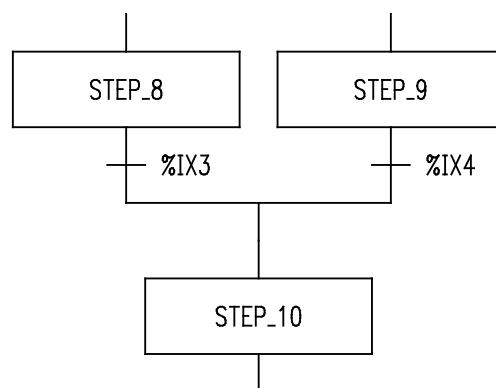


Abbildung 128 Zusammenführung

Ein Übergang von STEP_8 auf STEP_10 findet statt, wenn STEP_8 aktiv ist und %IX3 = TRUE.

Ein Übergang von STEP_9 nach STEP_10 findet statt, wenn STEP_9 aktiv ist und %IX4 = TRUE.

Ablaufaufspaltung (Parallelverzweigung)

Das Auslösen des Überganges (hier %IX6) führt zur **gleichzeitigen** Aktivierung mehrerer Abläufe.

Wenn zum Beispiel STEP_7 aktiv ist und die Transitionsbedingung %IX6 = TRUE, dann werden die Schritte STEP_8 und STEP_9 aktiviert. Danach laufen die beiden Schrittkettenzweige **unabhängig voneinander** ab.

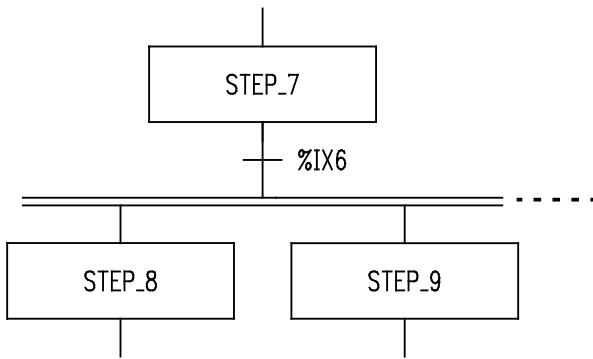


Abbildung 129 Ablaufspaltung

Zusammenführung (Ablaufsammlung)

Unter der waagerechten Doppellinie steht eine **gemeinsame** Transitionsbedingung. Der auf die Zusammenführung folgende Schritt kann nur aktiv werden, wenn **sämtliche** letzten Schritte der Zweige (hier STEP_8 und STEP_9) aktiv sind und die **gemeinsame** Transitionsbedingung (hier %IX7) wahr ist.

Der Übergang auf STEP_10 findet nur statt, wenn STEP_8 und STEP_9 aktiv sind und die gemeinsame Transitionsbedingung %IX7 = TRUE ist.

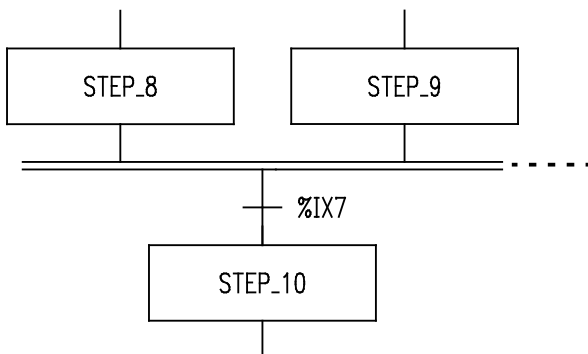


Abbildung 130 Ablaufsammlung

Sprung

Beim **Sprung** enthält mindestens ein Zweig **keine Schritte**. Wenn %IX1 = FALSE und %IX4 = TRUE (Sprungbedingung), werden die Schritte STEP_7 und STEP_8 übersprungen. Auf STEP_6 folgt dann unmittelbar STEP_9.

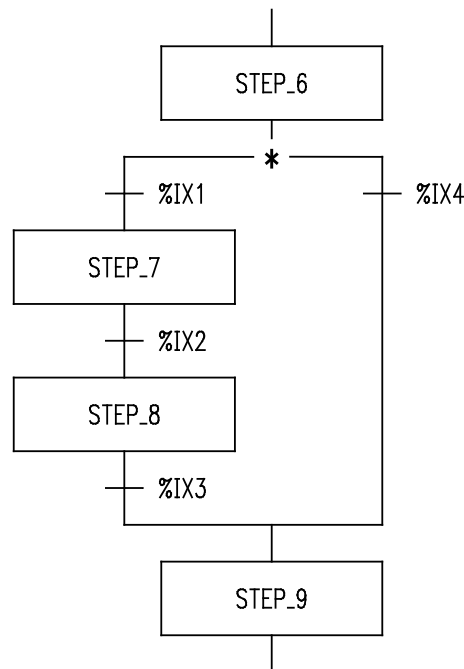


Abbildung 131 Sprung

Schleife

Bei der **Schleife** können bestimmte Schrittfolgen **mehrfach wiederholt** werden. Ein Übergang von STEP_8 nach STEP_7 findet statt, wenn %IX3 = FALSE und %IX4 = TRUE. Solange die Transitionsbedingungen diesen Zustand haben, werden die Schritte STEP_7 und STEP_8 **wiederholt**.

Erst wenn %IX3 = TRUE und %IX4 = FALSE, wird der Ablauf mit STEP_9 fortgesetzt.

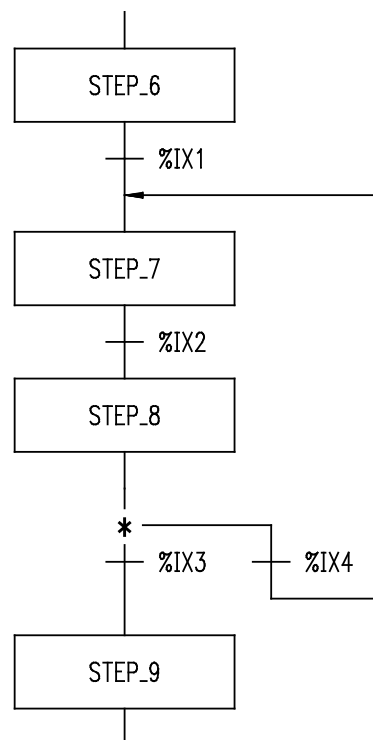


Abbildung 132 Schleife

Unsichere Netzwerke

Die Verbindung von Schritten und Transitionen bietet auch die Möglichkeit, **unsichere AS-Programme** zu entwickeln. Hierunter werden Programme verstanden, die aufgrund ihrer Logik „hängen bleiben“ oder bestimmte Programmteile niemals erreichen können.

Derartig programmierte Schrittketten sind fehlerhaft und müssen vom Programmiersystem oder SPS-Betriebssystem erkannt oder vermieden werden.

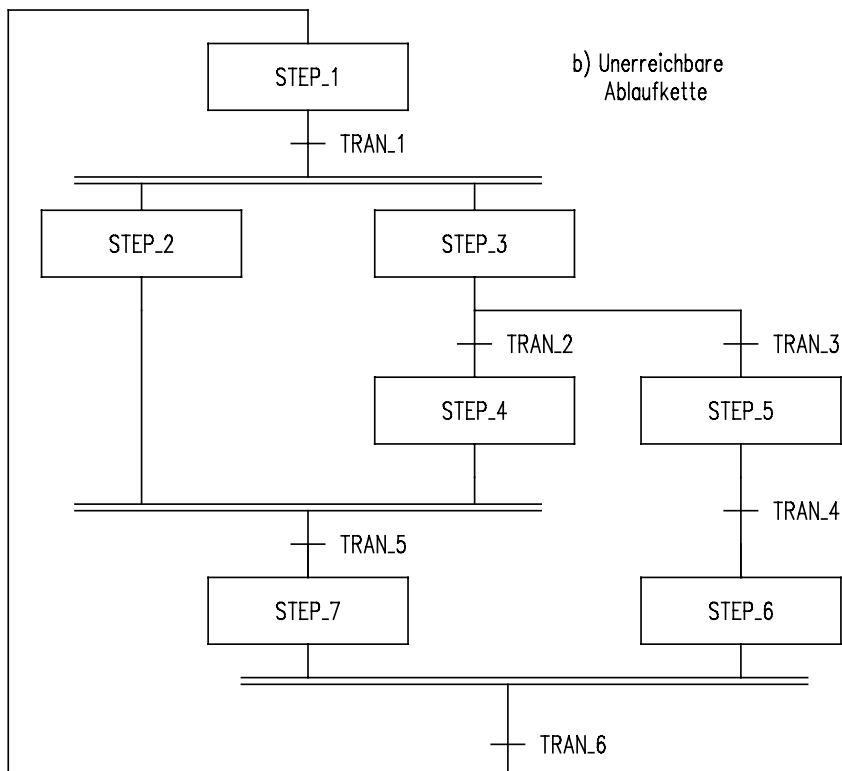
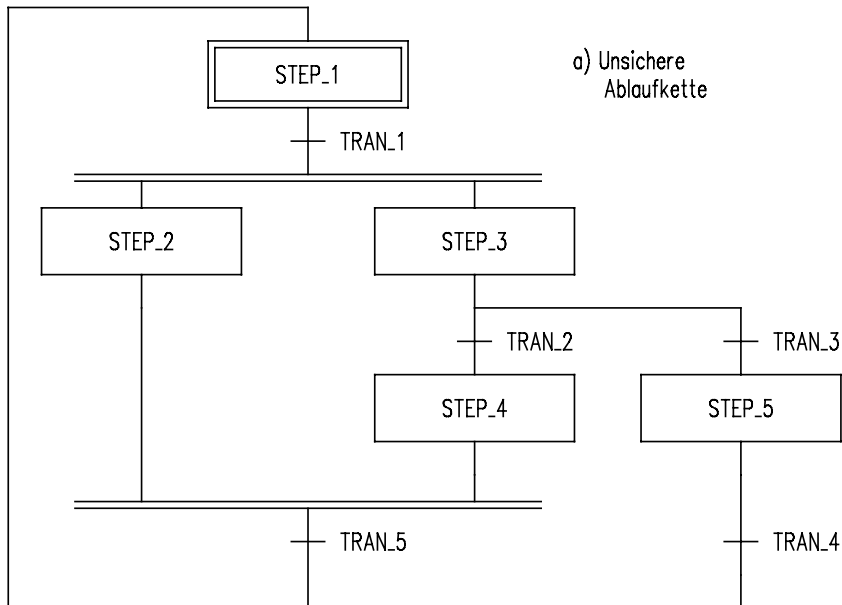


Abbildung 133 Fehler bei Ablaufketten

zu a) (Abbildung 133)

Angenommen, die Schritte STEP_2 und STEP_3 sind aktiv, die Transitionsbedingung TRAN_3 hat den booleschen Wert TRUE und die Transitionsbedingung TRAN_5 den booleschen Wert FALSE, kann der Schritt STEP_2 über STEP_5 und STEP_1 erneut aktiviert werden, obgleich er bereits aktiv ist.

Man sagt, das **Aktiv-Attribut** hat sich ohne Kontrolle vervielfältigt und bezeichnet dies als **unsicheres Netzwerk**.

zu b) (Abbildung 133)

Angenommen wird folgende Schrittfolge:

STEP_1 → STEP_3 → STEP_5 → STEP_6

Dann geht es nicht weiter, da TRAN_6 einen aktiven Schritt STEP_7 erwartet. STEP_7 ist aber nicht aktivierbar, da TRAN_5 auf STEP_4 wartet. TRAN_2 wird STEP_4 nicht aktivieren, da ja der Alternativzweig STEP_5 → STEP_6 bearbeitet werden soll. Der **notwendige Zustand** STEP_6 und STEP_7 aktiv, ist demnach **unerreichbar**.

Hinweise zur Programmierung

Ablaufaufspaltung (Parallelverzweigung)

Der Schritt PUMPE hat zwei Nachfolger (VENT_1 und VENT_2), die durch die Transitionsbedingung

%IX1 & %IX2

gleichzeitig aktiviert werden.

Es ist zu beachten, dass die beiden Folgeschritte, durch Komma getrennt, in Klammern geschrieben werden.

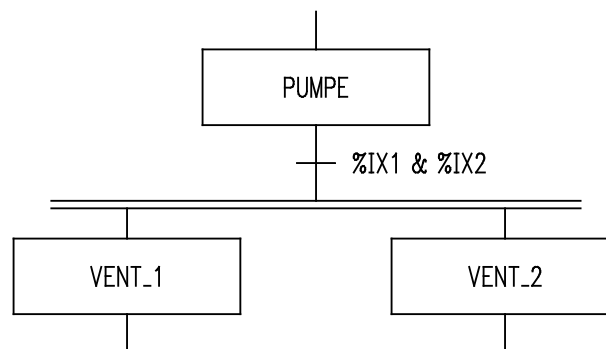


Abbildung 134 Ablaufaufspaltung

TRANSITION FROM PUMPE TO (VENT_1, VENT_2)

:= %IX1 & %IX2;

...

Zusammenführung (Ablaufsammlung)

Die beiden Schritte VENT_1 und VENT_2 haben **einen gemeinsamen Nachfolger** (HALT). Dieser Nachfolger wird aktiviert, wenn beide Vorgänger gesetzt sind und die Transitionsbedingung %IX3 wahr ist. Es ist zu beachten, dass die beiden Vorgänger, durch Komma getrennt, in Klammern geschrieben werden.

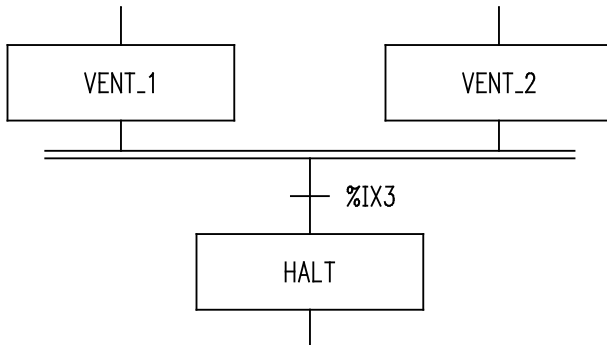


Abbildung 135 Ablaufzusammenführung

TRANSITION FROM (VENT_1, VENT_2) TO HALT
:= %IX3;

Lehrbeispiel

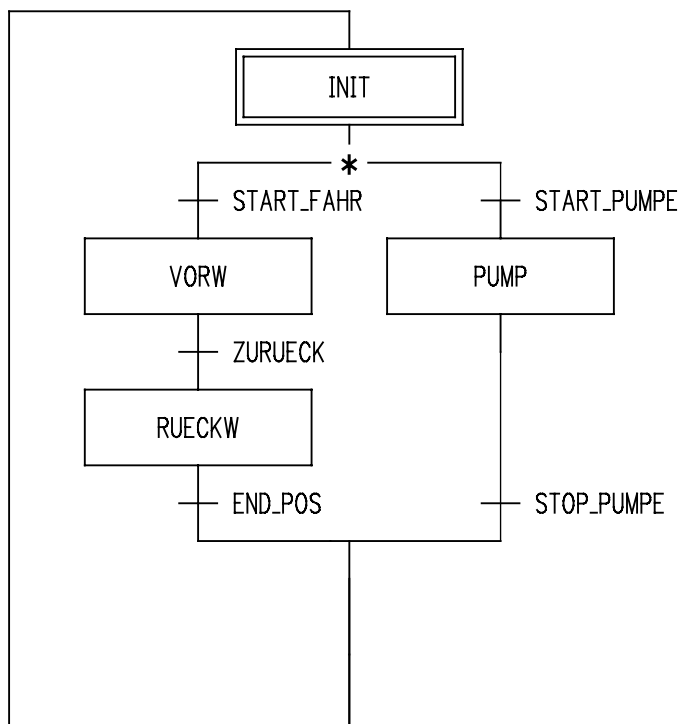


Abbildung 136 Verzweigung

Erstellen Sie das Programm in Textform!

Die Ablaufkette besteht aus zwei Zweigen. Der Stern gibt an, dass die Transitionen START_FAHR und START_PUMPE von links nach rechts bearbeitet werden. Bekanntlich ist der Initialisierungsschritt INIT zwingend erforderlich.

```

INITIAL_STEP INIT : END_STEP
TRANSITION FROM INIT TO VORW
:= START_FAHR;
END_TRANSITION

STEP VORW : END_STEP
TRANSITION FROM VORW TO RUECKW
:= ZURUECK;
END_TRANSITION

STEP RUECKW : END_STEP
TRANSITION FROM INIT TO PUMP
:= START_PUMPE & NOT START_FAHR;
END_TRANSITION

STEP PUMP : END_STEP
TRANSITION FROM RUECKW TO INIT
:= END_POS;
END_TRANSITION

TRANSITION FROM PUMP TO INIT
:= STOP_PUMPE;
END_TRANSITION

```

3.4 Befehle, Aktionen

Befehle rufen i.Allg. Aktionen hervor und werden grafisch durch **Befehlssymbole** dargestellt. Ein steuerndes System gibt einen **Befehl** aus, ein gesteuertes System führt eine **Aktion** aus.

Zum Beispiel:

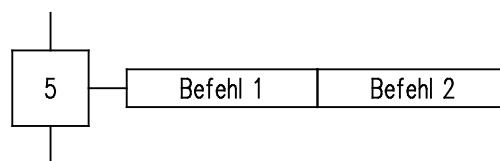
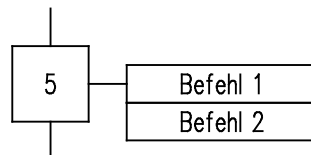
Befehl: Motor einschalten

Aktion: Motor wird eingeschaltet

Ein Befehl wird **ausgegeben**, wenn der Schritt, dem er zugeordnet ist, aktiv ist. Wenn der Schritt nicht mehr aktiv ist, wird der Befehl entweder

- nicht mehr ausgegeben (nicht gespeicherter Befehl)
- oder
- weiterhin ausgegeben (gespeicherter Befehl).

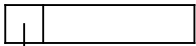
Darstellung von Befehlen



Wenn der 5. Schritt aktiv ist, werden die Befehle 1 und 2 ausgegeben. Durch die Anordnung der Befehle wird **keine Aussage** über die **Reihenfolge** der Bearbeitung gemacht.

Abbildung 137 Gleichwertige Darstellung von Befehlen

Befehlsarten (Aktionsarten)



Feld für Befehls-/Aktionsart

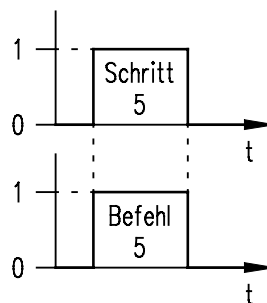
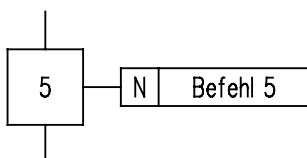
Abbildung 138 Befehlsart

Die Befehlsart (Aktionsart) wird in das linke Feld des Symbols eingetragen. Dabei sind genormte Bestimmungszeichen zu berücksichtigen.

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

Tabelle 18 Bestimmungszeichen für Befehle/Aktionen

Bestimmungszeichen N (oder kein Bestimmungszeichen)



Programm in FBS

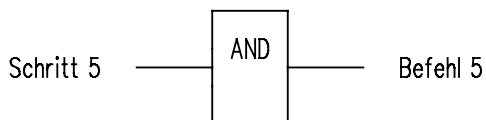
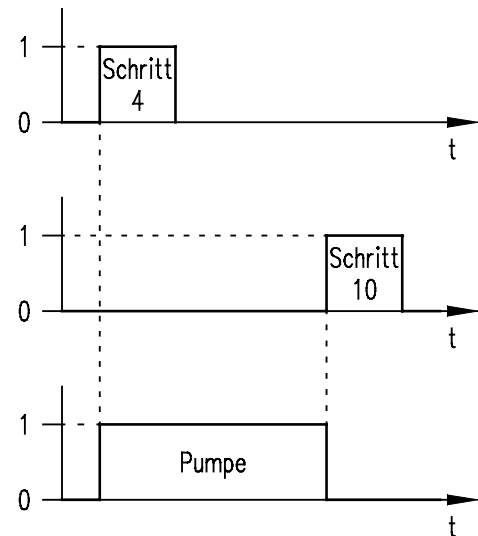
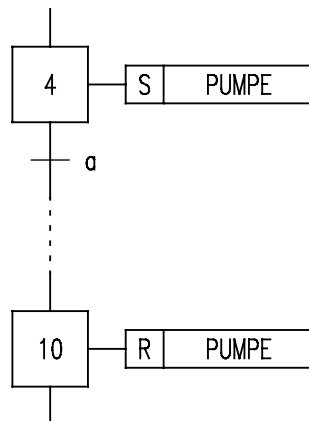


Abbildung 139 Bestimmungszeichen N

Der Befehl wird nur so lange ausgegeben, wie der 5. Schritt aktiv ist. N-Befehle wirken nur an dem Schritt, an dem sie „hängen“.

Bestimmungszeichen S und R



Programm in FBS

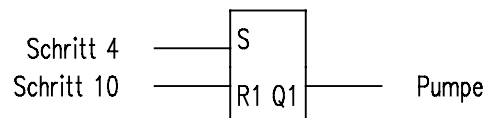
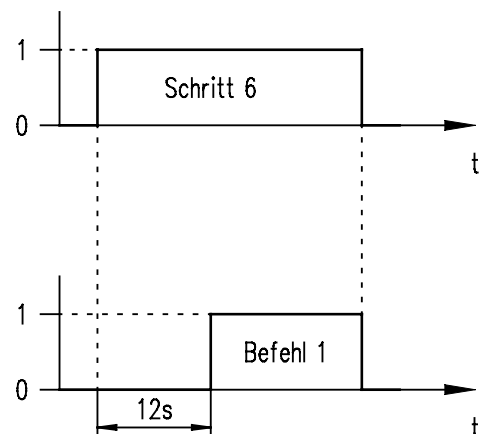
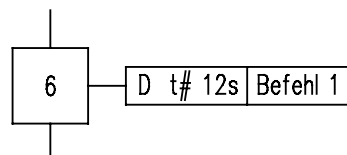


Abbildung 140 Bestimmungszeichen S und R

Sobald der 4. Schritt aktiv ist, wird die Pumpe eingeschaltet. Sobald der 10. Schritt aktiv ist, wird die Pumpe wieder ausgeschaltet. S-Befehle (S-Aktionen) wirken „schrittübergreifend“.

Bestimmungszeichen D



Programm in FBS

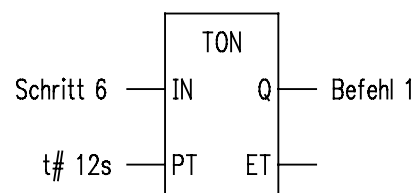


Abbildung 141 Bestimmungszeichen D

Ein Befehl mit dem Bestimmungszeichen D ist ein nichtspeichernder, verzögerter Befehl. Die Verzögerungszeit ist im Symbol angegeben. Der Befehl wird, nach dem der Schritt aktiv ist, um die angegebene Verzögerungszeit verspätet ausgegeben. Wenn der Schritt nicht mehr aktiv ist, wird auch der Befehl nicht mehr ausgegeben (nicht speichernd).

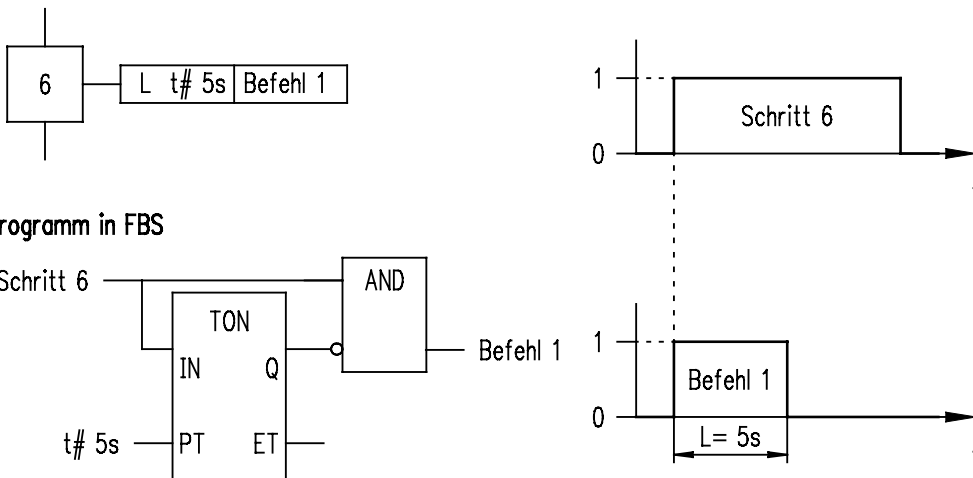
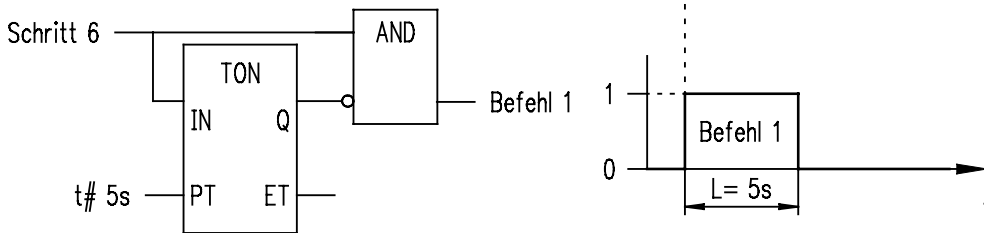
Bestimmungszeichen L**Programm in FBS**

Abbildung 142 Bestimmungszeichen L

Der Befehl mit dem Bestimmungszeichen L ist ein nichtspeichernder, zeitlich begrenzter Befehl. Die Zeit wird im Symbol angegeben. Wenn der Schritt aktiv ist, wird der Befehl unverzüglich ausgegeben. Nach Ablauf der Zeitbegrenzung wird die Befehlsausgabe beendet. Selbst wenn der Schritt noch gesetzt bleibt.

Programmierung von Aktionen/Befehlen

In der Regel gehört zu jedem Schritt mindestens eine **Aktion**. Eine Ausnahme bilden hier Schritte **ohne** Aktionen. Solche Schritte haben eine **Wartefunktion**. Sie warten, bis die Transitionsbedingung(en) des Nachfolgeschrittes erfüllt ist (sind).

Eine **Aktion** kann auf unterschiedliche Weise dargestellt werden.

Darstellung der Aktion durch eine boolesche Variable

Jede **boolesche Variable**, in einem VAR bzw. VAR_OUTPUT-Block deklariert, kann eine Aktion sein.

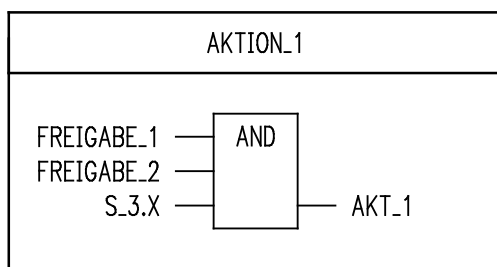
Darstellung der Aktion durch eine grafische Deklaration in FBS

Abbildung 143 Aktionsdarstellung

Darstellung der Aktion durch Textdeklaration in ST-Sprache

```

ACTION AKTION_1 :
AKT_1 := FREIGABE_1 & FREIGABE_2 & S_3.X;
END_ACTION

```

Darstellung der Aktion durch Textdeklaration in AWL-Sprache

```

ACTION AKTION_1 :
LD FREIGABE_1
AND FREIGABE_2
AND S_3.X
ST AKT_1
END_ACTION
    
```

Darstellung der Aktion durch Verwendung von AS-Elementen

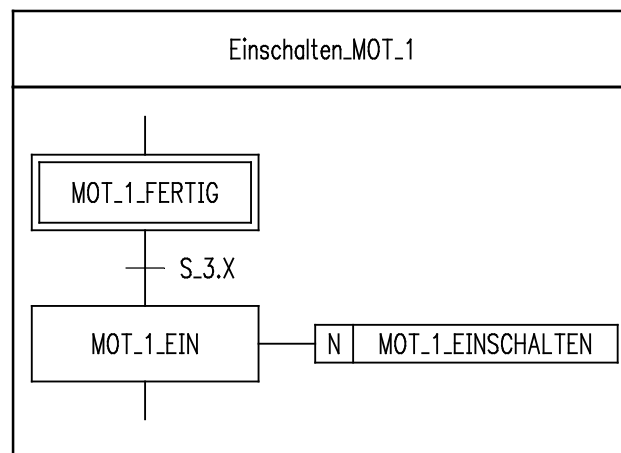


Abbildung 144 Aktion mit AS-Elementen

Aktionsblock in Textform

```

STEP S_3 :
AKTION_2 (D, t#5s, S1);
AKTION_3 (N);
END_STEP
    
```

Aktionsblock in Kontaktplandarstellung

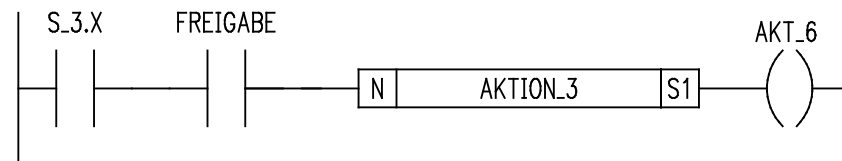


Abbildung 145 Aktionsblock (KOP)

Aktionsblock in FBS

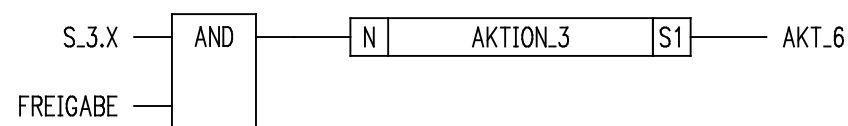


Abbildung 146 Aktionsblock (FBS)

Aktionsblock

Der **Aktionsblock** ist einem Schritt zugeordnet. Er wird ausgeführt, wenn der Schritt **aktiv** ist. Der Aktionsblock ist die grafische Darstellung einer Aktion mit **Bestimmungszeichen** und **Rückkopplungsvariable**.

Bestimmungszeichen	Aktionsname
--------------------	-------------

Tabelle 19 Einfacher Aktionsblock

Bestimmungszeichen	Aktionsname	Rückkopplungsvariable
Rumpf der Aktion: Aktion in einer beliebigen Sprache nach IEC 61131-3 (KOP, AWL, FBS, ST, AS)		

Tabelle 20 Aktionsblock mit angehängtem Anweisungsteil

Bestimmungszeichen	Aktionsname	Rückkopplungsvariable
--------------------	-------------	-----------------------

·
·
·

Aktionsname
Rumpf der Aktion: Aktion in einer beliebigen Sprache nach IEC 61131-3 (KOP, AWL, FBS, ST, AS)

Tabelle 21 Aktionsblock mit getrenntem Anweisungsteil; Verbindung über Aktionsnamen

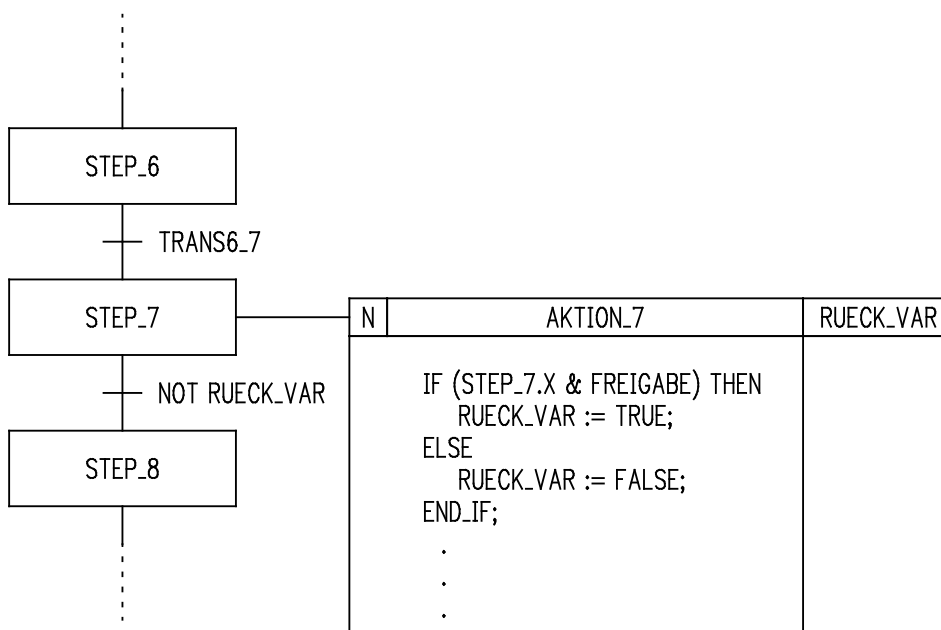


Abbildung 147 Aktionsblock in ST

Wenn die Transitionsbedingung **TRANS6_7** den booleschen Wert **TRUE** hat, wird der Schritt **STEP_7** aktiv geschaltet.

Das Bestimmungszeichen N des Aktionsblockes besagt, dass die Anweisungszeilen ausgeführt werden, solange der Schritt STEP_7 aktiv ist. Der Schritt STEP_7 bleibt aktiv, bis die Rückkopplungsvariable RUECK_VAR den booleschen Wert FALSE annimmt. Dies ist der Fall, wenn FREIGABE = FALSE. In diesem Fall wird der Rumpf der Aktion noch einmal durchlaufen und RUECK_VAR auf den booleschen Wert FALSE gesetzt.

Textdarstellung

```
TRANSITION FROM STEP_6 TO STEP_7
:= TRANS6_7;
END_TRANSITION
```

```
TRANSITION FROM STEP_7 TO STEP_8
:= NOT RUECK_VAR;
END_TRANSITION
```

```
ACTION AKTION_7
IF (STEP_7.X & FREIGABE) = TRUE THEN
    RUECK_VAR := TRUE;
ELSE
    RUECK_VAR := FALSE;
END_IF;
END_ACTION
```

Aktion mit Anweisungsteil in KOP-Sprache

Nach Aktivierung von STEP_6 nimmt die Variable VAR_1 den booleschen Wert TRUE an (dafür sorgt der Schrittmerker STEP_6.X). Insgesamt sind mehrere Netzwerke möglich, die in KOP programmiert werden.

Sobald der Übergang von STEP_6 nach STEP_7 erfolgt (Transitionsbedingung TRANS6_7 = TRUE), wird die AKTION_6 nochmals aufgerufen. Dann wird der Variablen VAR_1 der boolesche Wert der UND-Funktion von EIN_6 und EIN_7 zugewiesen, da STEP_6.X dann den booleschen Wert FALSE hat.

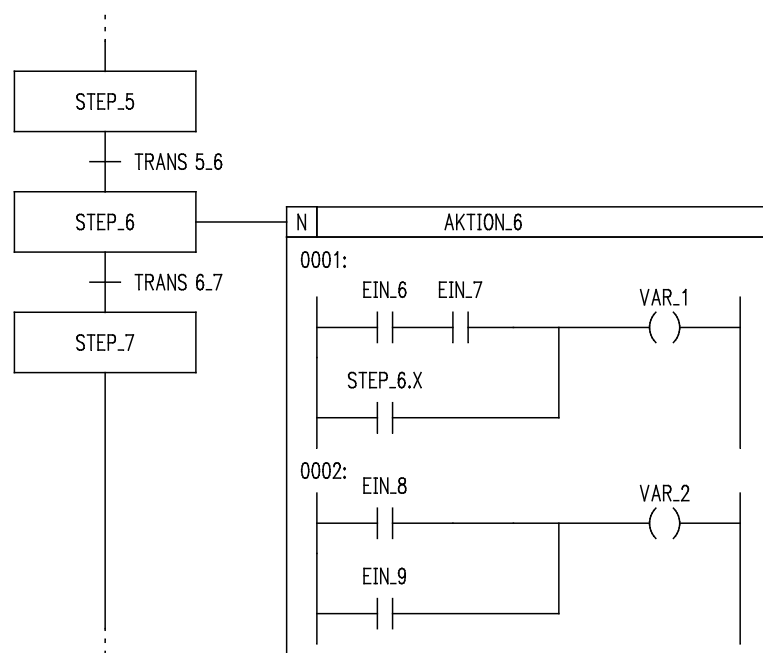


Abbildung 148 Aktionsblock in KOP

Aktion mit AS-Netzwerk

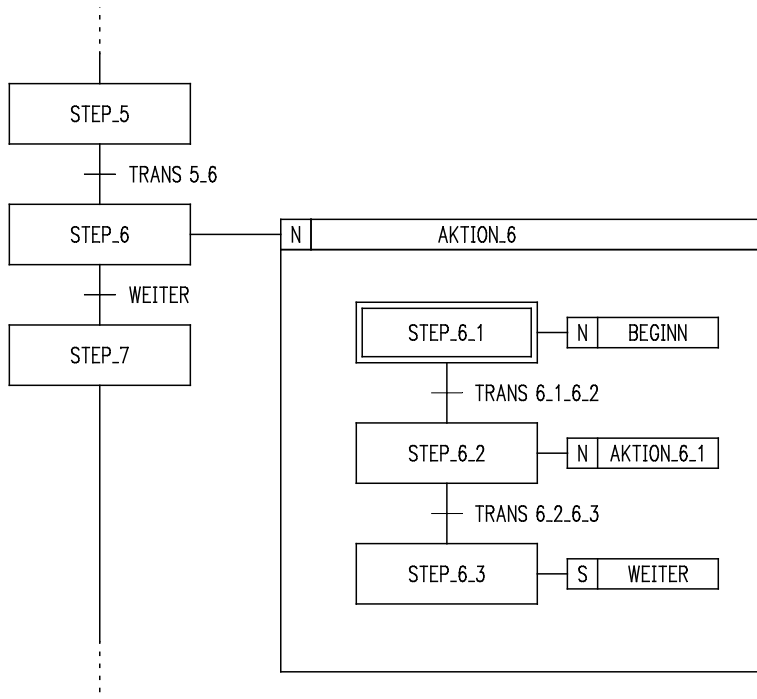


Abbildung 149 Aktionsblock in AS

Hier hat auch der Aktionsblock AS-Struktur. Es wird auch von **einer AS-Struktur der zweiten Ebene** gesprochen.

Wenn Schritt STEP_6 aktiv ist, wird die AS-Struktur von AKTION_6 bearbeitet. Sobald der Übergang von STEP_6 nach STEP_7 erfolgt ist, wird die AS-Struktur von AKTION_6 nicht mehr bearbeitet. Wenn unter allen Umständen die gesamte AS-Struktur durchlaufen werden soll, kann die Variable WEITER als letzte Aktion gesetzt werden und dann den Übergang nach STEP_7 ermöglichen.

Aktionssteuerung

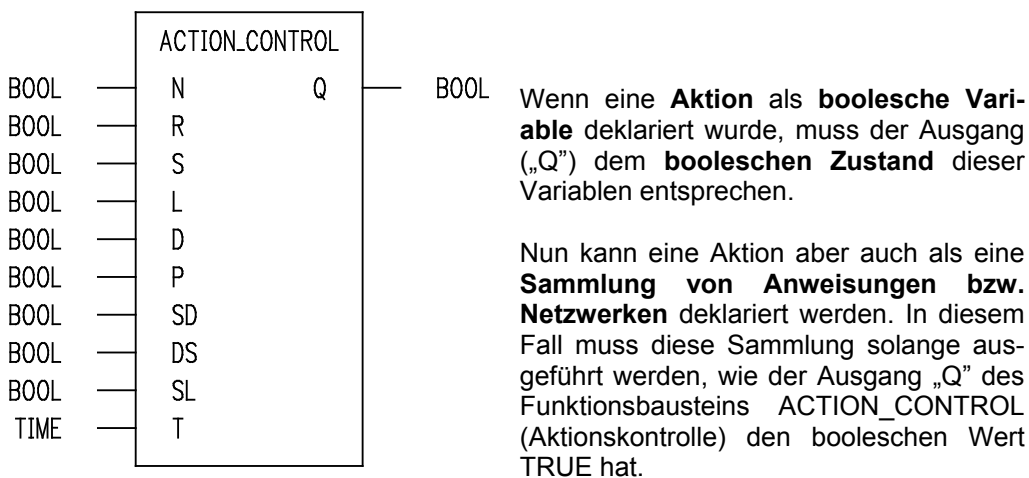


Abbildung 150 Funktionsbaustein ACTION_CONTROL

Die einzelnen Aktionen verfügen über eine ihnen direkt zugeordnete **Aktionskontrolle**. Sämtliche Aktionsblöcke, die eine bestimmte Aktion verwenden, beeinflussen die Aktionssteuerung dieser Aktion. Aus sämtlichen von den Aktionsblöcken gelieferten Ausführungsanforderungen wird eine Bedingung ermittelt, die bestimmt, ob die Aktion ausgeführt werden soll oder nicht.

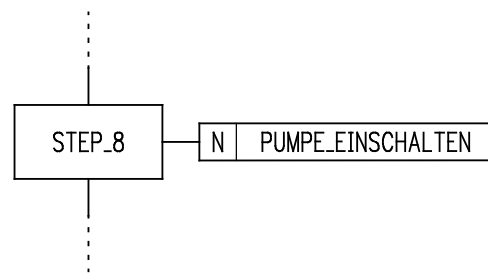
Im allgemeinen ist die **Aktionskontrolle** für den Anwender nicht sichtbar. Sie ist vielmehr Aufgabe des **SPS-Betriebssystems**.

Die **Aktionskontrolle** bestimmt in einem AS-Programm Beginn und Ende einer Aktion. Die Eingänge sind mit den Aktionsblöcken verbunden (Schrittmarker, Bestimmungszeichen), die die jeweilige Aktion umfassen. Der Ausgang ist ein boolescher Wert.

Zu beachten sind die 10 FB-Instanzen. Aus Gründen der Übersichtlichkeit wurden die Instanznamen weggelassen.

Sämtliche aktiven Schritte, die die Aktion verwenden, liefern den booleschen Wert TRUE an den Eingang, der im Aktionsblock als Bestimmungszeichen angegeben ist. Der Zeitparameter T wird dem jeweiligen Bestimmungszeichen mitgegeben.

Lehrbeispiel 1



Die Aktion PUMPE_EINSCHALTEN mit dem Bestimmungszeichen N soll nur an einer Stelle des Netzwerkes genutzt werden. Wenn der Schritt STEP_8 aktiv ist, nimmt der Schrittmarker den booleschen Wert TRUE an.

STEP_8.X = TRUE

Abbildung 151 N-Befehl/Aktion

Dieser Schrittmarker ist mit dem N-Eingang der Aktionskontrolle verbunden.

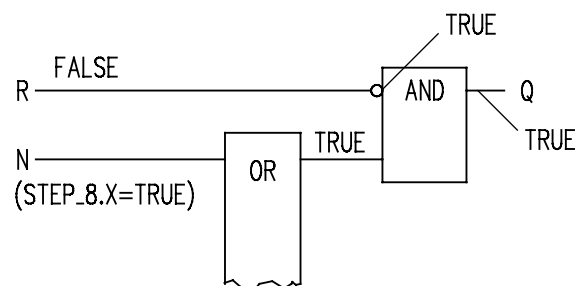


Abbildung 152 N-Befehl/Aktion

Somit liefert die ODER-Verknüpfung den booleschen Wert TRUE. Für diese Aktion ist kein RESET vorgesehen. Hat der negierte R-Eingang den booleschen Wert TRUE, hat das Ergebnis der UND-Verknüpfung ebenfalls den booleschen Wert TRUE (Q = TRUE). Die Aktion kann also ausgeführt werden.

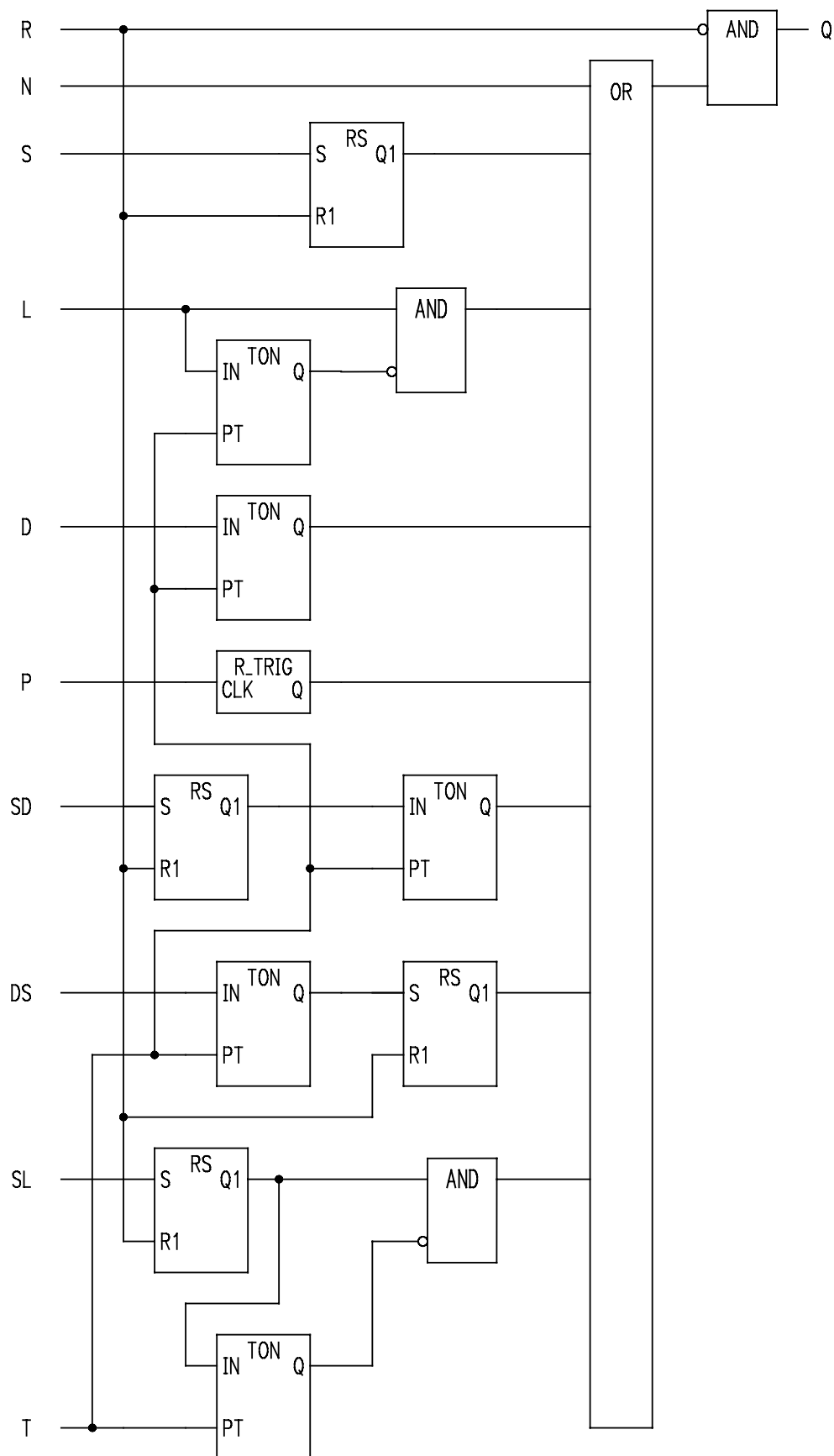


Abbildung 153 Aktionskontrolle; FB ACTION_CONTROL

Sobald der Schrittmarker STEP_8.X den booleschen Wert FALSE annimmt, hat der N-Eingang der Aktionskontrolle den booleschen Wert FALSE: Damit nimmt Q ebenfalls den booleschen Wert FALSE an, und die Aktion wird nicht mehr ausgeführt.

Lehrbeispiel 2

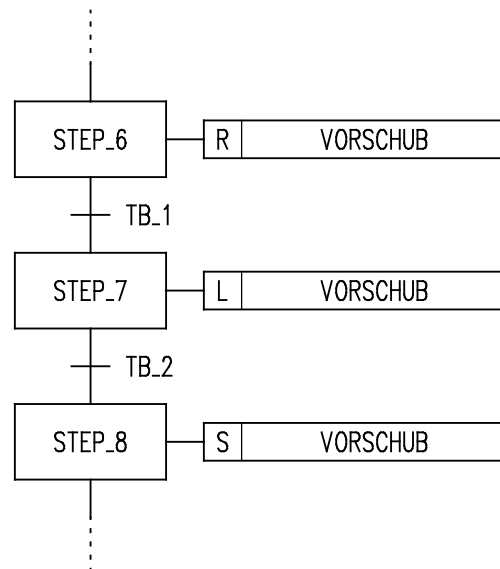


Abbildung 154 Mehrfachverwendung einer Aktion

Die Aktion VORSCHUB wird von drei unterschiedlichen Aktionsblöcken verwendet.

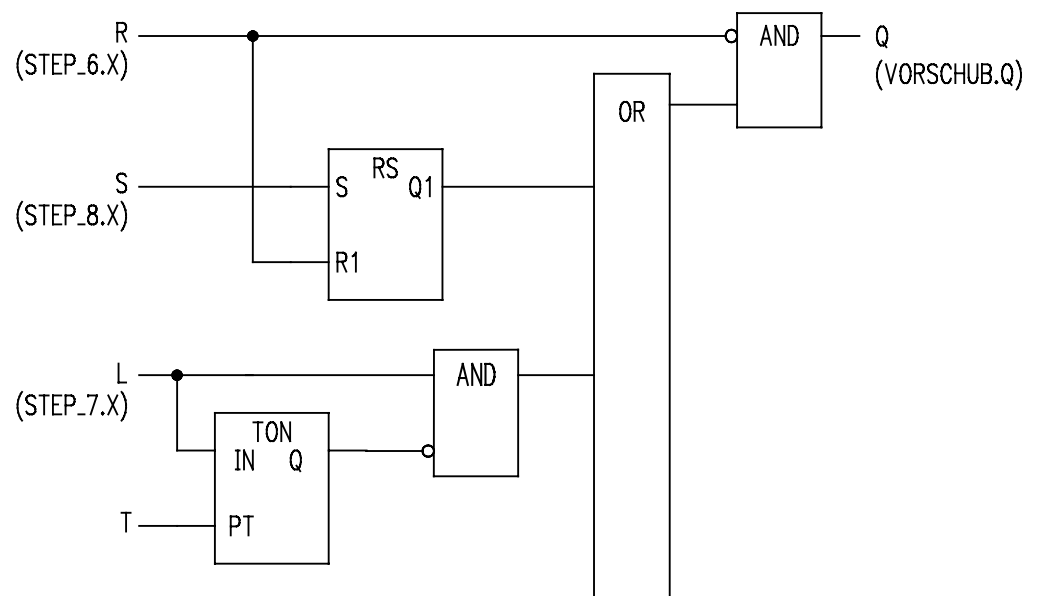


Abbildung 155 Bestimmung des booleschen Wertes von VORSCHUB.Q mittels FBS

Durch die Bestimmungszeichen der Aktion und durch die Schrittmarker wird der boolesche Wert von VORSCHUB.Q bestimmt.

Lehrbeispiel 3

Dargestellt ist ein Steuerungsausschnitt in AS-Darstellung, wobei auf die Deklaration verzichtet wurde.

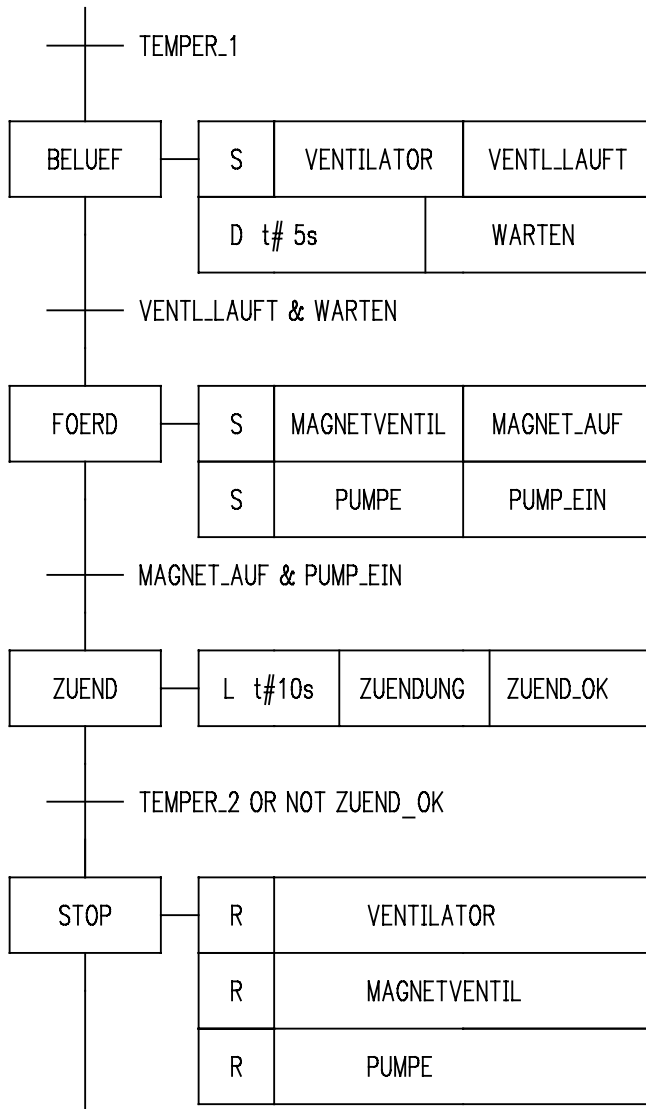


Abbildung 156 Steuerungsausschnitt in AS-Darstellung

Hinweise:

Initialisierungsschritt **START_HEIZ** ist nicht dargestellt.

Wenn die Temperatur **TEMPER_1** erreicht wird, wird der Belüftungsschritt **BELUEF** aktiviert. Dabei wird der Ventilator speichernd eingeschaltet. Eine Vorbelüftungszeit **WARTEN** von 5 Sekunden verstreicht.

Wenn der Ventilator in Betrieb ist (Rückmeldung **VENTL_LAUF**) und die Wartezeit verstrichen ist, erfolgt der Übergang vom Schritt **BELUEF** zum Schritt **FOERD** (Fördern). Das Magnetventil öffnet, und die Pumpe wird eingeschaltet.

Wenn beide Befehle am Schritt **FOERD** ausgegeben wurden, erfolgt der Übergang vom Schritt **FOERD** zum Schritt **ZUEND** (zünden). Dabei wird 10 Sekunden lang der zeitbegrenzte Befehl Zündung ausgegeben.

Wenn die Zündung erfolgreich war (Rückmeldung ZUEND_OK), verharret die Steuerung im Schritt ZUEND, bis die Temperatur TEMPER_2 erreicht wird.

War die Zündung nicht erfolgreich (ZUEND_OK = FALSE), wird nach 10 Sekunden der Übergang zum Schritt STOP vollzogen.

Wenn die Temperatur TEMPER_2 erreicht ist, oder die Zündung nicht gelang, erfolgt der Übergang vom Schritt ZUEND zum Schritt STOP. Die drei S-Befehle (Ventilator, Magnetventil, Pumpe) werden zurückgesetzt (R).

Darstellung des Steuerungsausschnittes in Textform

```
INITIAL_STEP START_HEIZ : END_STEP
TRANSITION FROM START_HEIZ TO BELUEF
:= TEMPER_1;
END_TRANSITION
```

```
STEP_BELUEF:
VENTILATOR (S, VENTL_LAUFT);
WARTEN (D, t#5s);
END_STEP
TRANSITION FROM BELUEF TO FOERD
:= VENTL_LAUFT & WARTEN;
END_TRANSITION
```

```
STEP_FOERD:
MAGNETVENTIL (S, MAGNET_AUF);
PUMPE (S, PUMP_EIN);
END_STEP
```

```
TRANSITION FROM FOERD TO ZUEND
:= MAGNET_AUF & PUMP_EIN;
END_TRANSITION
```

```
STEP_ZUEND:
ZUENDUNG (L, t#10s, ZUEND_OK);
END_STEP
```

```
TRANSITION FROM ZUEND TO STOP
:= TEMPER_2 OR NOT ZUEND_OK;
END_TRANSITION
```

```
STEP_STOP:
VENTILATOR (R);
MAGNETVENTIL (R);
PUMPE (R);
END_STEP
```

Zur Beachtung:

Im Beispiel werden mehrere **Rückkopplungsvariablen** verwendet (z.B. VENT_LAUFT). Die Rückkopplungsvariable hat den booleschen Wert TRUE, wenn die Aktion aktiv ist. Sie ist, wie andere Variablen auch, vom Anwender zu **deklarieren**. Wenn **keine** Rückkopplungsvariable definiert ist, wird ihre Aufgabe vom **Aktionsnamen** übernommen. Hieraus folgt:

Wenn Aktionsnamen an anderer Stelle als im Namensfeld des **Aktionsblockes** benutzt werden, müssen sie vom Anwender als boolesche Variable deklariert werden.

Schritte und zugeordnete Aktionen werden nach Aktivierung mindestens einmal durchlaufen.

Nach erfolgter Deaktivierung werden die Schritte und zugeordneten Aktionen erneut aufgerufen, damit ENDE-Bedingungen ausgewertet werden können.

Lehrbeispiel 4

Für eine Stanze soll ein Steuerungsprogramm entwickelt werden. Ein Arbeitsgang der Stanze besteht aus einer Abwärts- und Aufwärtsbewegung.

Wenn der Taster „EINS“ betätigt wird, findet **ein** Stanzvorgang statt. Wenn der Taster „ZWEI“ betätigt wird, finden **zwei** aufeinander folgende Stanzvorgänge statt.

S1 EIN_STANZ_VORG	Wahltaster, 1 Stanzvorgang, Schließer
S2 ZWEI_STANZ_VORG	Wahltaster, 2 Stanzvorgänge, Schließer
S3 STANZE_OBEN	Grenztaster Stanze oben, Öffner
S4 STANZE_UNTEN	Grenztaster Stanze unten, Öffner
M1 STANZE_AB	Stanze abwärts (Stanzen)
STANZE_AUF	Stanze aufwärts

Grafische Deklaration der Variablen

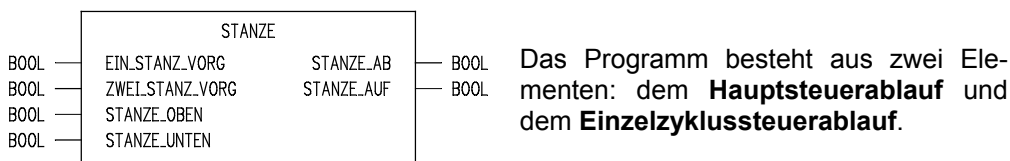


Abbildung 157 Grafischer Deklarationsblock

Im Hauptsteuerablauf wird bestimmt, ob ein oder zwei Stanzvorgänge gewünscht werden. Dies hängt davon ab, ob der Taster EIN_STANZ_VORG oder der Taster ZWEI_STANZ_VORG betätigt wird.

Taster EIN_STANZ_VORG betätigt:

Der Schritt ST_EINZ (Stanzen einzeln) wird aktiviert, wodurch der N-Befehl STANZ_VORGANG ausgegeben wird; der im Einzelzyklussteuerablauf als Transition verwendet wird.

Wenn diese Transition den booleschen Wert TRUE hat, wird der Einzelzyklussteuerablauf eingeleitet und ein Stanzvorgang vollzogen. Nach Beendigung des Stanzvorganges hat der Schrittmerker STOP.X den booleschen Wert TRUE, und der Schritt ENDE wird aktiviert.

Nur wenn die Taster EIN_STANZ_VORG und ZWEI_STANZ_VORG unbetätigt sind, ist ein weiterer Hauptsteuerablauf möglich.

Taster ZWEI_STANZ_VORG betätigt:

Der Schritt ST_ZWEI_1 wird aktiviert (1. Stanzvorgang). Wenn der Einzelzyklussteuerablauf bearbeitet wurde, wird der Warteschritt WARTEN aktiviert und anschließend der Schritt ST_ZWEI_2 (2. Stanzvorgang). Der weitere Ablauf entspricht dem oben gesagten.

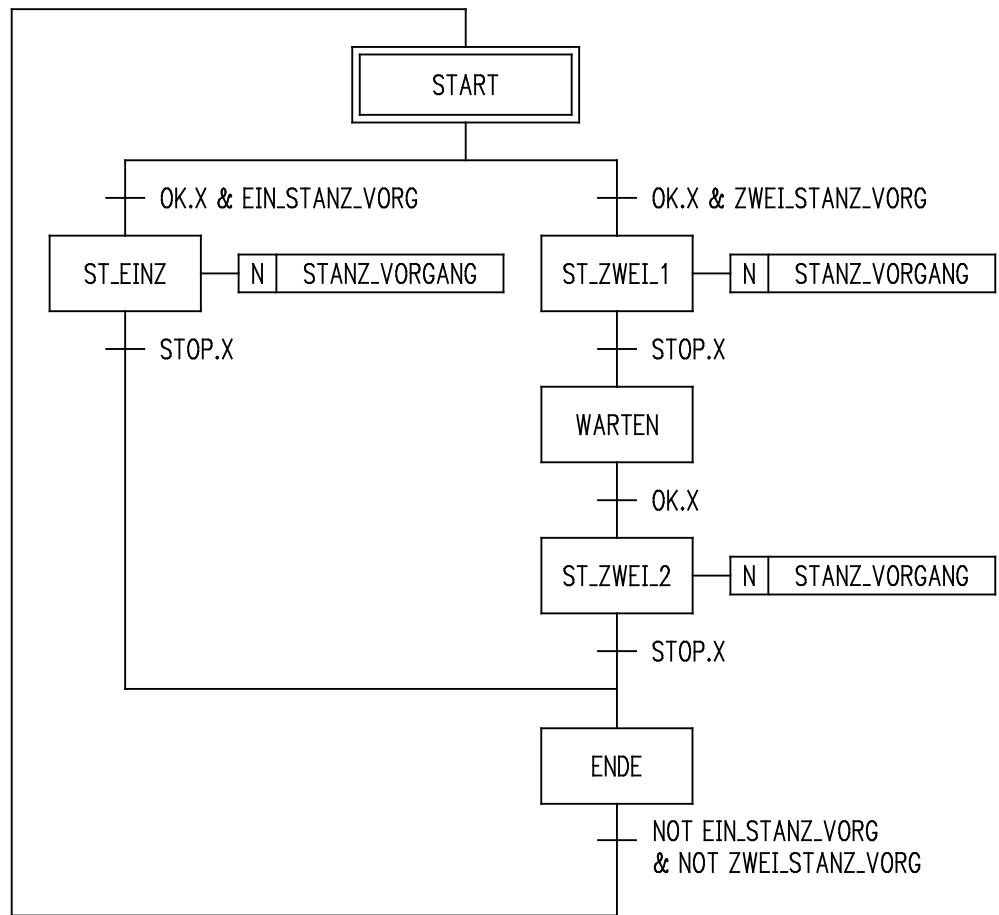


Abbildung 158 Hauptsteuerablauf

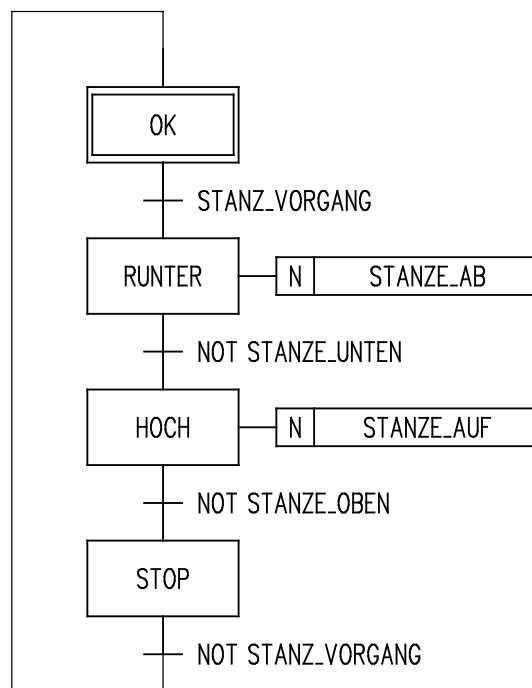
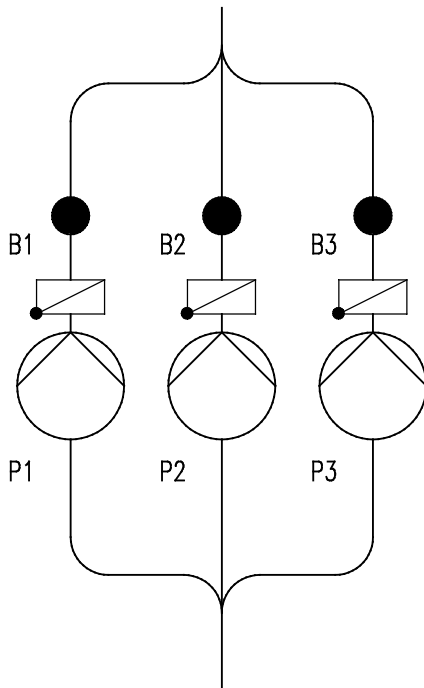


Abbildung 159 Einzelsteuerablauf

Lehrbeispiel 5

Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben.

Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst.

Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal); wenn keine Strömung erkannt wird, somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung geöffnet werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

Abbildung 160 Pumpensteuerung

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten

S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten

S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten

S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Die Problemlösung soll mithilfe der Ablaufsprache erfolgen.

S1 P1_2_EIN	Pumpe 1, 2 einschalten, Schließer
S2 P2_3_EIN	Pumpe 2, 3 einschalten, Schließer
S3 P1_3_EIN	Pumpe 1, 3 einschalten, Schließer
S0 ALLE_P_AUS	Alle Pumpen ausschalten
B1 STROEM_P1	Strömungswächter Pumpe 1, Öffner
B2 STROEM_P2	Strömungswächter Pumpe 2, Öffner
B3 STROEM_P3	Strömungswächter Pumpe 3, Öffner

K1 PUMPE1	Pumpe 1
K2 PUMPE2	Pumpe 2
K3 PUMPE3	Pumpe 3

Hinweise:

Die thermischen Auslöser (Überstromrelais) wirken ausschließlich elektromechanisch. Wenn hierdurch eine Pumpe abgeschaltet wird, erfasst dies der zugehörige Strömungswächter.

Die Strömungswächter arbeiten wie folgt:

Strömung vorhanden: „0“-Signal (Öffner betätigt)

Keine Strömung vorhanden: „1“-Signal (Öffner unbetätigt)

Arbeitsweise

Annahme: Pumpe 1 und Pumpe 2 sollen eingeschaltet werden. Es erfolgt dann der Übergang von Schritt INIT nach Schritt P_1_2, und die Pumpen 1 und 2 werden speichernd eingeschaltet. Wenn mindestens einer der Strömungswächter der eingeschalteten Pumpen keine Strömung erkennt, erfolgt der Übergang von Schritt P_1_2 nach Schritt P_3. Die Reservepumpe 3 wird dann eingeschaltet. Wird der Stoptaster betätigt, erfolgt der Übergang nach Schritt P_AUS, und alle Pumpen werden ausgeschaltet.

Da der Betrieb der Reservepumpe sicherlich nicht stets vorausgesetzt werden kann, ist ein Sprung von Schritt P_1_2 nach Schritt P_AUS notwendig. Die Arbeitspumpen können dann auch ausgeschaltet werden, ohne dass hierzu Schritt P_3 aktiviert werden muss.

Jeder der drei Zweige erfüllt prinzipiell die gleiche Aufgabe, wobei jeweils andere Pumpen Arbeits- und Reservepumpe sind.

Wenn alle Pumpen ausgeschaltet wurden, erfolgt der Übergang von Schritt P_AUS nach INIT. Der Ablauf kann dann erneut beginnen.

117

Allerdings ergibt sich beim Steuerungsablauf ein Problem: Bei der Inbetriebnahme zeigt sich, dass stets alle drei Pumpen eingeschaltet werden, obgleich keine der Pumpen defekt ist. Dies hat folgende Ursache:

Wenn z.B. die Pumpen 1 und 2 als Arbeitspumpen eingeschaltet werden, benötigen sie etwas Zeit, um die Strömung in den jeweiligen Pumpenzweigen „aufzubauen“. In diesem Zeitraum melden die Strömungswächter „keine Strömung“, und die Reservepumpe wird zusätzlich eingeschaltet.

Um dies zu verhindern, muss den Arbeitspumpen Zeit gegeben werden, Strömung „aufzubauen“. Erst nach Ablauf dieser Zeit darf die Reservepumpe im Bedarfsfall eingeschaltet werden können. Hierzu ist folgende Änderung notwendig, die für den linken Zweig (P1_2_EIN) exemplarisch dargestellt ist.

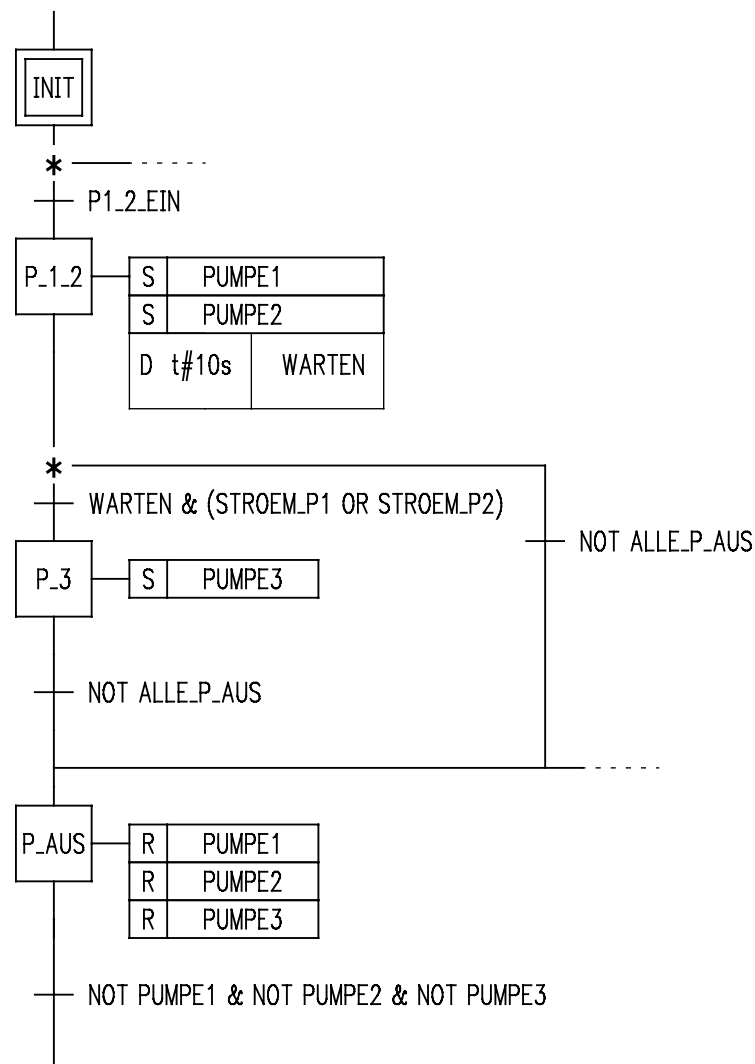
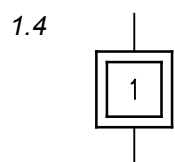
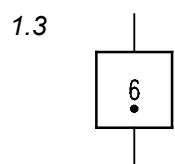
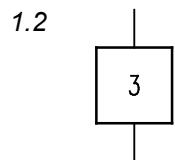
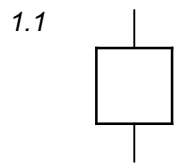


Abbildung 162 Anlaufzeit der Pumpen

Erst nach Ablauf der Zeit WARTEN kann im Bedarfsfall die Reservepumpe eingeschaltet werden.

Aufgabe 1

Erläutern Sie die dargestellten Symbole!

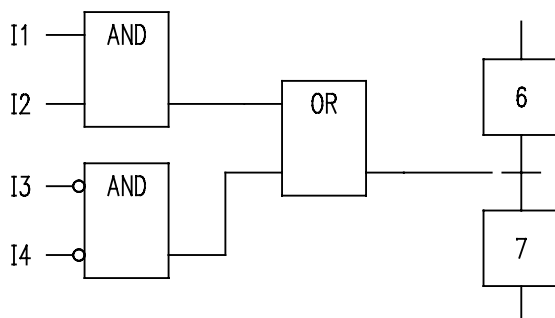


Aufgabe 2

Was versteht man unter einer Transition?

Aufgabe 3

Erläutern Sie die Wirkung der dargestellten Transition!



Aufgaben

Aufgabe 4

Dargestellt ist eine Transition in Textform.

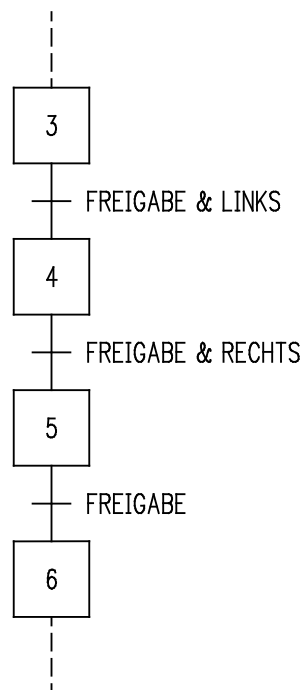
Schreiben Sie die Transition in grafischer Form!

```

STEP SCHRITT_6 : END_STEP
  TRANSITION FROM SCHRITT_6 TO SCHRITT_7
    := %IX6 & NOT %IX7;
  END_TRANSITION
STEP SCHRITT_7: END_STEP
  
```

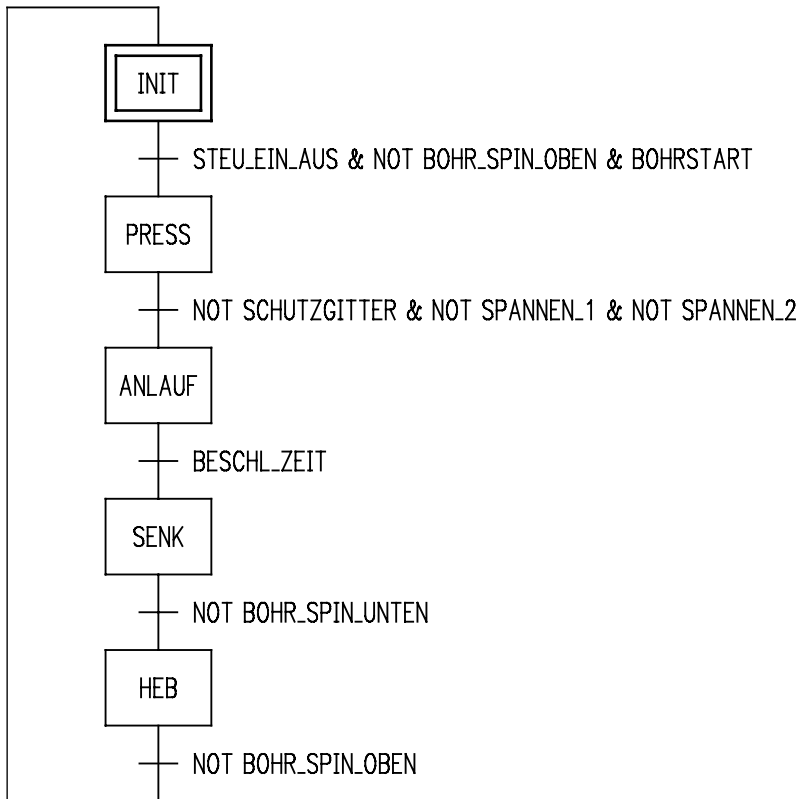
Aufgabe 5

Beschreiben Sie die nachstehend dargestellte Ablaufkette!



Aufgabe 6

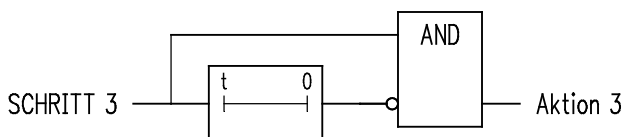
Stellen Sie die Ablaufkette in Textform dar!

Aufgabe 7

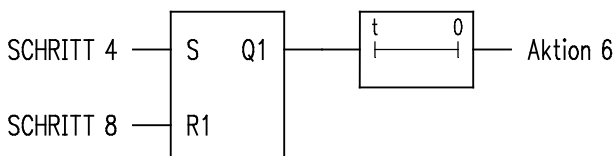
Unter welcher Voraussetzung ist die N-Aktion und unter welcher Voraussetzung ist die S-Aktion besonders sinnvoll einsetzbar?

Aufgabe 8

Geben Sie das Bestimmungszeichen der dargestellten Aktion an!
Stellen Sie das Symbol und das Signal-Zeit-Diagramm dar!

Aufgabe 9

Um welche Aktion handelt es sich? Geben Sie auch das Symbol an!



Aufgabe 10

Stellen Sie die DS-Aktion als Funktionsplan, Symbol und Signal-Zeit-Diagramm dar! Die programmierte Verzögerungszeit beträgt 15 s. Schritt 12 startet die Aktion, Schritt 20 nimmt sie wieder zurück.

Aufgabe 11

AS-Programm in Textdarstellung mit Deklaration.

Stellen Sie die grafische Form des Programms dar!

```

PROGRAM RUEHRWERK_1

  VAR_INPUT
    STEUERUNG_EIN , START_FUELL : BOOL ;
    VOLL , HALBVOLL , LEER : BOOL ;
  END_VAR

  VAR_OUTPUT
    ZULAUF_1 , ZULAUF_2 , RUEHREN : BOOL ;
    HEIZ_ZUL , HEIZ_ABL : BOOL ;
  END_VAR

  VAR
    HEIZZEIT : BOOL ;
    RUEHR_AUS : BOOL ;
  END_VAR

  INITIAL_STEP INIT : END_STEP

  TRANSITION FROM INIT TO FL_1
    := STEUERUNG_EIN & START_FUELL & LEER ;
  END_TRANSITION

  STEP FL_1 :
    ZULAUF_1 (N) ;
  END_STEP

  TRANSITION FROM FL_1 TO FL_2
    := NOT HALBVOLL ;
  END_TRANSITION

  STEP FL_2 :
    ZULAUF_2 (N) ;
    RUEHREN (S) ;
  END_STEP

  TRANSITION FROM FL_2 TO HEIZ
    := NOT VOLL ;
  END_TRANSITION

  STEP HEIZ :
    HEIZ_ZUL (N) ;
    HEIZ_ABL (N) ;
    HEIZZEIT (D, t#30m);
  END_STEP

```

```

TRANSITION FROM HEIZ TO ABPUMP
    := HEIZZEIT ;
END_TRANSITION

STEP ABPUMP :
    ABLAUF (N) ;
END_STEP

TRANSITION FROM ABPUMP TO AUS
    := LEER ;
END_TRANSITION

STEP AUS :
    RUEHREN (R, RUEHR_AUS) ;
END_STEP

TRANSITION FROM AUS TO INIT
    := RUEHR_AUS ;
END_TRANSITION

END_PROGRAM

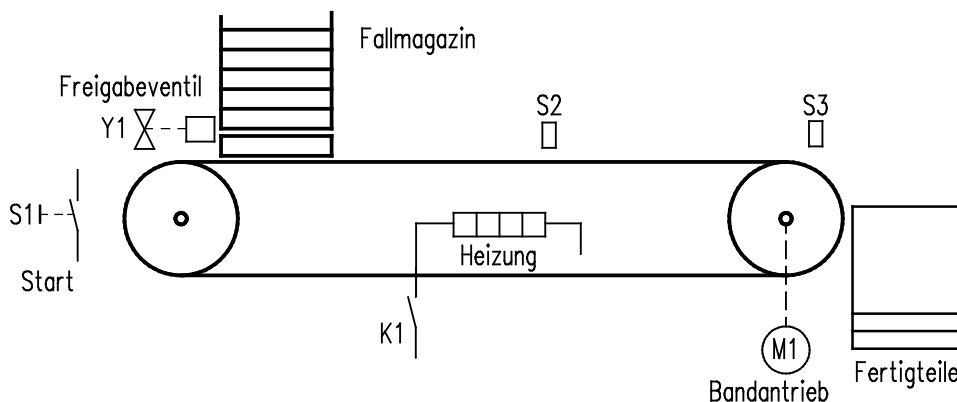
```

Aufgabe 12

Über ein Förderband werden Teile aus einem Fallmagazin zu einem Behälter transportiert. Bei Betätigung des Tasters S1 wird ein Teil aus dem Fallmagazin freigegeben; gleichzeitig wird das Band eingeschaltet.

Wenn die Heizposition (S2) erreicht ist, wird das Band angehalten. Dann wird 15 Sekunden die Heizung zur Feuchteentfernung eingeschaltet.

Danach wird das Transportband erneut eingeschaltet, damit das Teil in den Auffangbehälter (Endposition S3) gebracht werden kann. Bei Erreichen der Endposition stoppt das Band.



Bei erneuter Betätigung des Starttasters wiederholt sich der Vorgang.

Entwickeln Sie die grafische AS-Darstellung! Wählen Sie die Schritt- und Variablen-namen selbst.

**Realisierung
Projekt 2
„Verladeanlage“**Aufgabe

Nach der Erarbeitung der Grundlagen erstellen Sie zu dem technischen Problem „Verladeanlage“ eine Lösung in Form einer Ablaufsteuerung!

4 Realisierung einer Steuerung mittels strukturiertem Text (ST)

Bei einigen steuerungstechnischen Problemen wird eine SPS-Programmierung notwendig, die „strategische“ Entscheidungen vom Programm verlangt.

Ziel dieses Lernbereichs ist die Vermittlung einer höheren Programmiersprache, mit der es möglich ist, komplexe Programmstrukturen aufzubauen.

Durch die Darstellung eines typischen Steuerungsproblems am Beispiel eines Verkaufsautomaten werden Vorteile einer textorientierten höheren Programmiersprache deutlich.

Nach der Erarbeitung der erforderlichen Befehle und möglichen Programmstrukturen wird als Abschluss des Lernbereichs das Projekt 3 „Verkaufsautomat“ gelöst.

Projekt 3: Verkaufsautomat

Für einen Verkaufsautomaten für Blumengebinde ist eine Steuerung zu entwerfen.

Technologieschema:

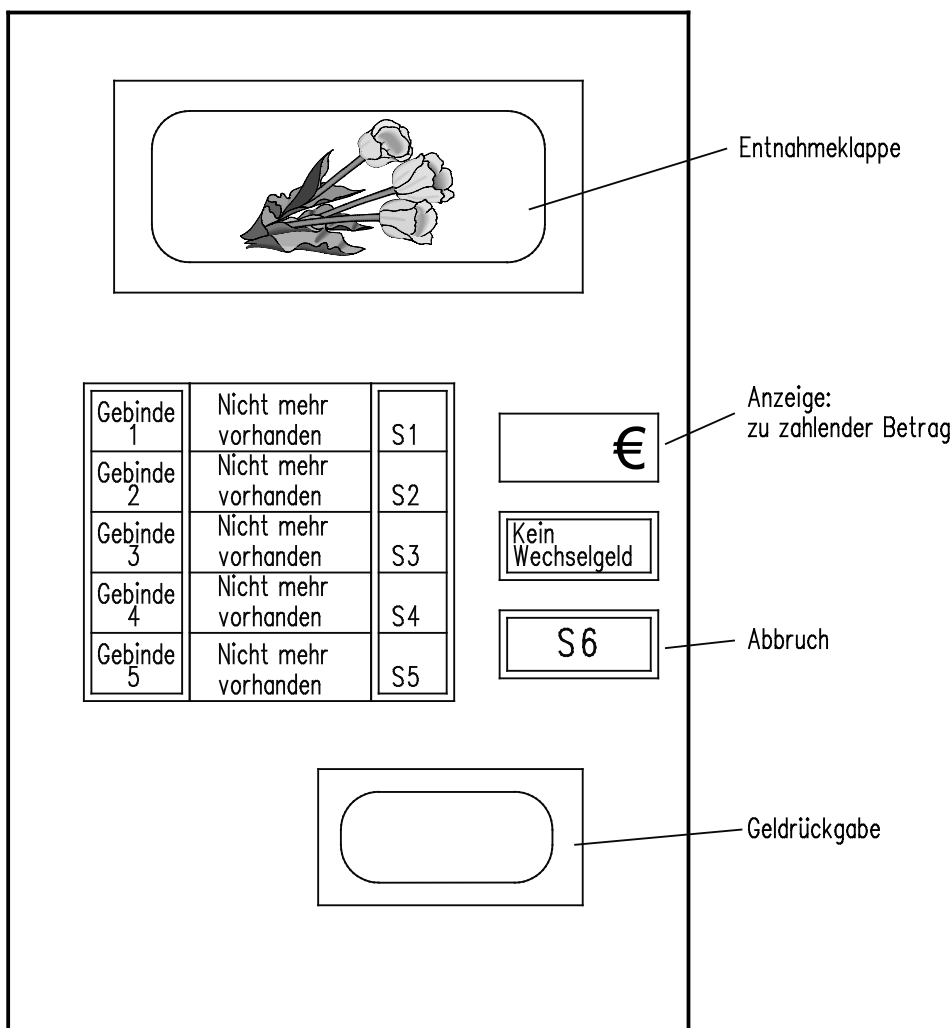


Abbildung 163 Verkaufsautomat

Lernbereich

Allgemeine Beschreibung

In dem Verkaufsautomaten befinden sich fünf Behälter mit Blumengebinden. Der Kunde kann anhand von Bildern eine Vorauswahl treffen. Zur direkten Betrachtung kann der Behälter mit der gewünschten Sorte auf Anforderung an das Sichtfenster der Entnahmeklappe gefahren werden. Gleichzeitig wird der erforderliche Betrag angezeigt.

Funktionsbeschreibung

Die Steuerung soll folgende Anforderungen erfüllen:

1. Steuerung für die Gebindeauswahl

Dem Benutzer stehen fünf Taster für die Auswahl der einzelnen Sorten zur Verfügung. Neben den Tastern befindet sich in einem Sichtfenster ein Bild des Gebindes und eine Anzeige, die aufleuchtet, wenn dieses Gebinde nicht mehr vorhanden ist. Betätigt ein Benutzer den entsprechenden Taster, wird das gewünschte Gebinde auf dem **kürzesten Weg** zur Entnahmeklappe gefahren. Um den kürzesten Weg zu berechnen, muss im Programm die aktuelle Position der Gebinde als Information vorliegen. Zur Bestimmung der aktuellen Position liefert der Induktiv-Sensor 2 Zählimpulse. Der Kunde kann nun bezahlen und das Gebinde entnehmen. Danach erhält er gegebenenfalls das Wechselgeld. Bis zur Entnahme des Gebindes kann der Vorgang über den Taster S6 abgebrochen werden. Nachdem ein Behälter geleert wurde, darf er nicht mehr angefahren werden.

Betriebsmittel	Symbol. Name	Kommentar
S1	E1	Auswahltaster für das Gebinde 1
S2	E2	Auswahltaster für das Gebinde 2
S3	E3	Auswahltaster für das Gebinde 3
S4	E4	Auswahltaster für das Gebinde 4
S5	E5	Auswahltaster für das Gebinde 5
S6	E6	Abbruchtaster
Magnet-verriegelung	A1	A1 = 1 Entnahmeklappe entriegelt A1 = 0 Entnahmeklappe verriegelt

Tabelle 22 Zuordnungsliste 1

2. Einschaltmodus

Beim Einschalten wird die Behältersteuerung an eine definierte Stelle fahren, die durch den Induktivsensor 1 markiert ist.

3. Geldroutine

Zugelassene Geldstücke sind 0,50 €, 1,- € und 2,- €.

Der Preis pro Gebinde beträgt 3,- €.

Bei Entnahme des Gebindes wird das Geld einbehalten und evtl. 1,- € zurückgezahlt. Wenn nicht 3,- € passend oder 4,- € gezahlt wurden, wird das eingezahlte Geld vollständig zurückgezahlt.

Die Münzprüfeinrichtung (MPE) gibt über drei Signalleitungen entsprechend der erkannten Münze ein kurzzeitiges „1“-Signal aus. Zudem fallen die jeweiligen Münzen in separate Schächte, von denen sie entweder in die Speicherschächte bei Entnahme des Gebindes oder in die Geldrückgabe bei Abbruch fallen.

Das Wechselgeld wird in ein spezielles Fach eingefüllt. Es besteht keine Verbindung zum Geldeinzug (Falschgeld). Nach dem Einfüllen wird durch die Betätigung des Tasters S13 die Anzahl auf 10 eingestellt.

Betriebsmittel	Symbolischer Name	Kommentar
MPE	E7	Kurzzeitiges "1"-Signal für 0,50 €-Stück
MPE	E8	Kurzzeitiges "1"-Signal für 1,- €-Stück
MPE	E9	Kurzzeitiges "1"-Signal für 2,- €-Stück
Induktivsensor 1	E10	Startposition
Induktivsensor 2	E11	Zählsensor
Geldverteilung	A2	Signal, welches das Geld einbehält
Geldverteilung	A3	Signal, welches das eingezahlte Geld zurückgibt
Wechselgeld	A4	Signal, das 1,- € zurückzahlt
S7	E12	Öffner E12 = 1 Klappe geschlossen E12 = 0 Klappe geöffnet
	A5	Rechtslauf
	A6	Linkslauf

Tabelle 23 Zuordnungsliste 2

4.1 Programmaufbau

Der **Strukturierte Text** (ST) ist eine Programmiersprache nach IEC 61131-3. Er kann als **höhere Programmiersprache** angesehen werden, da außerordentlich **mächtige Befehle** Verwendung finden.

Der Strukturierte Text setzt sich aus einer Folge von Anweisungen zusammen, die durch ein Semikolon voneinander getrennt sind. Eine Anweisung kann in mehreren Zeilen geschrieben werden; ebenso können mehrere Anweisungen (durch Semikolon getrennt) in einer Zeile stehen. Der Zeilenumbruch wird wie ein Leerzeichen behandelt.

Wesentliche **Vorteile** der Programmiersprache ST sind:

- Relativ kurze Programme
- Übersichtlicher Programmaufbau

Zumindest gilt dies für das SPS-Programm **vor** der Übersetzung.

Der Strukturierte Text hat allerdings auch **Nachteile**: Die übersetzten Programme sind im Allgemeinen länger und langsamer. Die Übersetzung in den Maschinencode ist nicht direkt beeinflussbar (höhere Programmiersprache). Vergleichbare höhere Programmiersprachen sind PASCAL und C.

4.2 Syntax

Elemente der Programmiersprache ST

Ausdruck

Der Ausdruck ist eine Teilanweisung, zusammengesetzt aus **Operatoren** und **Operanden**, die einen Wert liefert.

Beispiele

PUMPE1 AND NOT PUMPE2 AND PUMPE3

(PUMPE1 AND PUMPE2) OR (PUMPE2 AND PUMPE3)

Ein Ausdruck kann auch aus einem **einzigen Operanden** bestehen.

VENTIL_6

Mögliche Operanden sind:

- Konstanten
- Variablen
- Funktionsaufrufe
- andere Ausdrücke (z.B. 60-30)

Die Operanden werden in den Ausdrücken durch **Operatoren** verknüpft. Dabei ist wichtig, die **Rangfolge** der Operatoren zu berücksichtigen.

Rangstufe	Operation	Symbol	Erläuterung
1	Klammerung	(...)	$(12 * 2) + (6/2)$ Wert 27
2	Funktionsauswertung	Funktionsname (Argumentliste)	ADD (A,B,C)
3	Potenzierung	**	$2^{**}3$ Wert 8
4	Negation	-	- 5 Wert -5
5	Komplement	NOT	NOT TRUE Wert FALSE
6	Multiplikation	*	$6 * 2$ Wert 12
7	Division	/	$6/2$ Wert 3
8	Modulo	MOD	Rest bei Integerdivision $12 \text{ MOD } 10$, Wert 2
9	Addition	+	$30 + 12$ Wert 42
10	Subtraktion	-	$30 - 12$ Wert 18
11	Vergleich	<, >, >=, <=	$10 < 8$ Wert FALSE
12	Gleichheit	=	$12 = 12$ Wert TRUE
13	Ungleichheit	< >	$12 < > 12$ Wert FALSE
14	UND	AND, &	FALSE & TRUE Wert FALSE
15	Exklusiv-ODER	XOR	TRUE XOR FALSE Wert TRUE
16	ODER	OR	FALSE OR TRUE Wert TRUE

Tabelle 24 Rangfolge der Operatoren bei der Programmiersprache ST

Der Operator mit der höchsten Rangstufe ist zuerst anzuwenden, dann der Operator mit der nächsthöheren Rangstufe usw. Haben die Operatoren den **gleichen Rang**, werden Sie im Ausdruck **von links nach rechts** angewendet.

Beispiele

$16 - 5 * 2 + 10$

Unter Berücksichtigung der Rangfolge (Punktrechnung vor Strichrechnung) ergibt sich das richtige Ergebnis 16. Würde der Ausdruck ohne Rangfolge bearbeitet werden, ergäbe sich das falsche Ergebnis 32.

$(\text{PUMPE1 AND PUMPE2}) \text{ OR } (\text{PUMPE2 AND PUMPE3})$

Die UND-Funktion hat einen höheren Rang als die ODER-Funktion. Hierzu kommt noch, dass die Klammerung die höchste Rangstufe hat.

Anweisung

Eine Anweisung ist eine Zuweisung, wie sie in der Datenverarbeitung üblicherweise verwendet wird. Die Zuweisung weist das Ergebnis eines Ausdrucks einer Variablen zu.

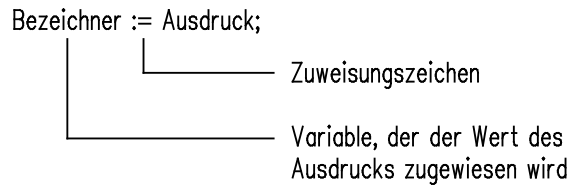


Abbildung 164 Anweisung

Unter einem **Bezeichner** wird nach IEC 61131-3 eine Folge von Buchstaben, Ziffern und Unterstrichzeichen verstanden. Diese Folge muss mit einem Buchstaben oder einem Unterstrich beginnen und darf keine Leerzeichen enthalten.

Beispiele

`TEMP_HOCH := TEMP1 OR TEMP2;` Das Ergebnis der booleschen Operation TEMP1 ODER TEMP2 wird der Variablen TEMP_HOCH zugewiesen. Wenn mindestens eine der beiden Variablen TEMP1 oder TEMP2 den booleschen Wert TRUE (den Signalzustand „1“) hat, wird der Variablen TEMP_HOCH der boolesche Wert TRUE zugewiesen.

`CV := CV + 1` Die Variable CV wird inkrementiert (um 1 erhöht).

`A := A + 2; B := B + 1; C := SIN (X);` Mehrere Anweisungen können, durch Semikolon getrennt, in einer Zeile geschrieben werden.

Aufruf eines Funktionsbausteins

Der Aufruf erfolgt über den **Namen** des Funktionsbausteins und die Angabe seiner in Klammern eingeschlossenen **Parameter**. Dabei ist die Reihenfolge der Parameter beliebig. Wenn bei einem Aufruf bestimmte Parameter nicht aktualisiert werden, gelten deren **Initialisierungswerte** oder die **Werte des vorangegangenen Aufrufs**.

Beispiel

`ZEIT_1 (IN := START_ZEIT, PT := t#30s);`
`AUS_1 := ZEIT_1.Q;`

Timer mit dem Instanznamen ZEIT_1. Wenn START_ZEIT = TRUE, wird die Zeit mit der Dauer 30 s gestartet. Das Ergebnis der Instanz ZEIT_1 wird dem Ausgang AUS_1 zugewiesen.

Funktionen der Operatoren

Funktionen können über ihre **Namen** und durch Angabe ihrer **Aktualparameter** aufgerufen werden. Die Aktualparameter werden in Klammern geschrieben und durch Kommata voneinander getrennt. Wahlweise können auch Formalparameter mit angegeben werden, dann ist die Reihenfolge der Parameteraufzählung bedeutungslos.

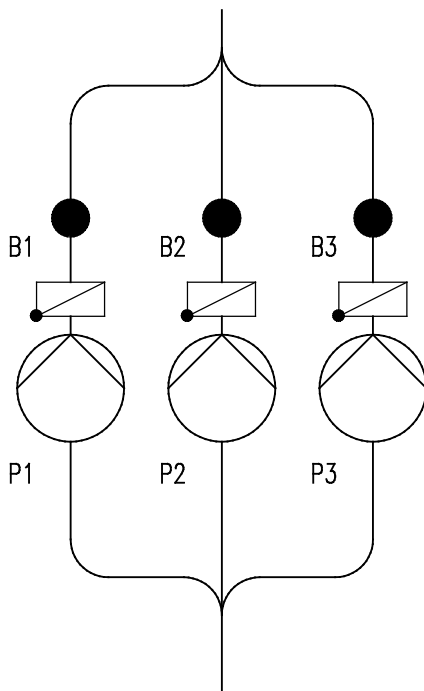
Beispiele

ADD (3, 4, 5); (*Angabe der Aktualparameter*)

ROL (IN := B, N := 3); (*Angabe der Formalparameter*)

ROL bedeutet „Rotate Left“ (Linksrotation); Verschiebung der Bits in einem Datenwort.

Lehrbeispiel



Für die Heizung eines Bürogebäudes werden drei Umwälzpumpen installiert, von denen immer zwei Pumpen gemeinsam arbeiten sollen. Wenn eine dieser beiden Arbeitspumpen ausfällt, schaltet sich automatisch die dritte Pumpe (Reservepumpe) ein.

Die drei Pumpen (P1...P3) werden von Drehstrom-Käfigläufermotoren angetrieben.

Die Strömung in den einzelnen Pumpenzweigen wird von Strömungswächtern (B1...B3) erfasst.

Wenn eine Strömung erkannt wird, liefert der betreffende Strömungswächter keine Spannung („0“-Signal); wenn keine Strömung erkannt wird, somit Spannung („1“-Signal). Verdrahtet sind also die Öffner der Strömungswächter, die bei Strömung geöffnet werden.

Damit keine Pumpe dauerhaft stillsteht, sollen immer zwei unterschiedliche Pumpen miteinander arbeiten können. Die dann gerade nicht benötigte Pumpe ist die Reservepumpe.

Abbildung 165 Pumpensteuerung

S1 betätigt: Pumpe 1 und Pumpe 2 arbeiten

S2 betätigt: Pumpe 2 und Pumpe 3 arbeiten

S3 betätigt: Pumpe 1 und Pumpe 3 arbeiten

S0 betätigt: Alle Pumpen aus

Jede Pumpe wird durch ein thermisches Motorschutzrelais vor Überlastung geschützt.

Das Programm ist in der Sprache ST zu erstellen!

S0	AUS	Austaster, Öffner
S1	P1_P2_EIN	Pumpe 1 und Pumpe 2 ein, Schließer
S2	P2_P3_EIN	Pumpe 2 und Pumpe 3 ein, Schließer
S3	P1_P3_EIN	Pumpe 1 und Pumpe 3 ein, Schließer

F4	THERM_P1	Thermisches Überstromrelais, Pumpe 1, Öffner
F5	THERM_P2	Thermisches Überstromrelais, Pumpe 2, Öffner
F6	THERM_P3	Thermisches Überstromrelais, Pumpe 3, Öffner
B1	STROEM_P1	Strömungswächter Pumpe 1, Öffner
B2	STROEM_P2	Strömungswächter Pumpe 2, Öffner
B3	STROEM_P3	Strömungswächter Pumpe 3, Öffner
K1	PUMPE_1	Pumpe 1 einschalten
K2	PUMPE_2	Pumpe 2 einschalten
K3	PUMPE_3	Pumpe 3 einschalten

FUNCTION_BLOCK PUMPENANTRIEB

VAR_INPUT (*Eingangsvariablen*)

AUS, P1_P2_EIN, P2_P3_EIN, P1_P3_EIN : BOOL;

THERM_P1, THERM_P2, THERM_P3 : BOOL;

STROEM_P1, STROEM_P2, STROEM_P3 : BOOL;

END_VAR

VAR_OUTPUT (*Ausgangsvariablen*)

PUMPE_1, PUMPE_2, PUMPE_3 : BOOL;

END_VAR

VAR

MERK_4 : BOOL; (*Pumpe 1 und 2 ein*)

MERK_5 : BOOL; (*Pumpe 2 und 3 ein*)

MERK_6 : BOOL; (*Pumpe 1 und 3 ein*)

ANLAUFZEIT : TON; (*Wartezeit Reserve*)

PUMPE_1_2 : RS; (*Instanz Pumpe 1/2*)

PUMPE_2_3 : RS; (*Instanz Pumpe 2/3*)

PUMPE_1_3 : RS; (*Instanz Pumpe 1/3*)

END_VAR

(*Pumpe 1 und Pumpe 2 einschalten*)

PUMPE_1_2 (S := P1_P2_EIN, R1 := NOT AUS OR MERK_5 OR MERK_6);

MERK_4 := PUMPE_1_2.Q1;

(*Pumpe 2 und Pumpe 3 einschalten*)

PUMPE_2_3 (S := P2_P3_EIN, R1 := NOT AUS OR MERK_4 OR MERK_6);

MERK_5 := PUMPE_2_3.Q1;

(*Pumpe 1 und Pumpe 3 einschalten*)

PUMPE_1_3 (S := P1_P3_EIN, R1 := NOT AUS OR MERK_4 OR MERK_5);

MERK_6 := PUMPE_1_3.Q1;

(*Anlaufzeit der Arbeitspumpen, die Strömung aufbauen müssen*)

ANLAUFZEIT (IN := MERK_4 OR MERK_5 OR MERK_6, PT := t#5s);

(*Pumpe 1*)

PUMPE_1 := MERK_4 OR MERK_6 OR (ANLAUFZEIT.Q & MERK_5 &
(STROEM_P2 OR STROEM_P3)) & THERM_P1;

(*Pumpe 2*)

PUMPE_2 := MERK_4 OR MERK_5 OR (ANLAUFZEIT.Q & MERK_6 &
(STROEM_P1 OR STROEM_P3)) & THERM_P2;

(*Pumpe 3*)

PUMPE_3 := MERK_5 OR MERK_6 OR (ANLAUFZEIT.Q & MERK_4 &
(STROEM_P1 OR STROEM_P2)) & THERM_P3;

END_FUNCTION_BLOCK

4.3 Programmierung von Verzweigungen

Alternativverzweigung

Anweisungen werden **abhängig von Bedingungen** ausgeführt.

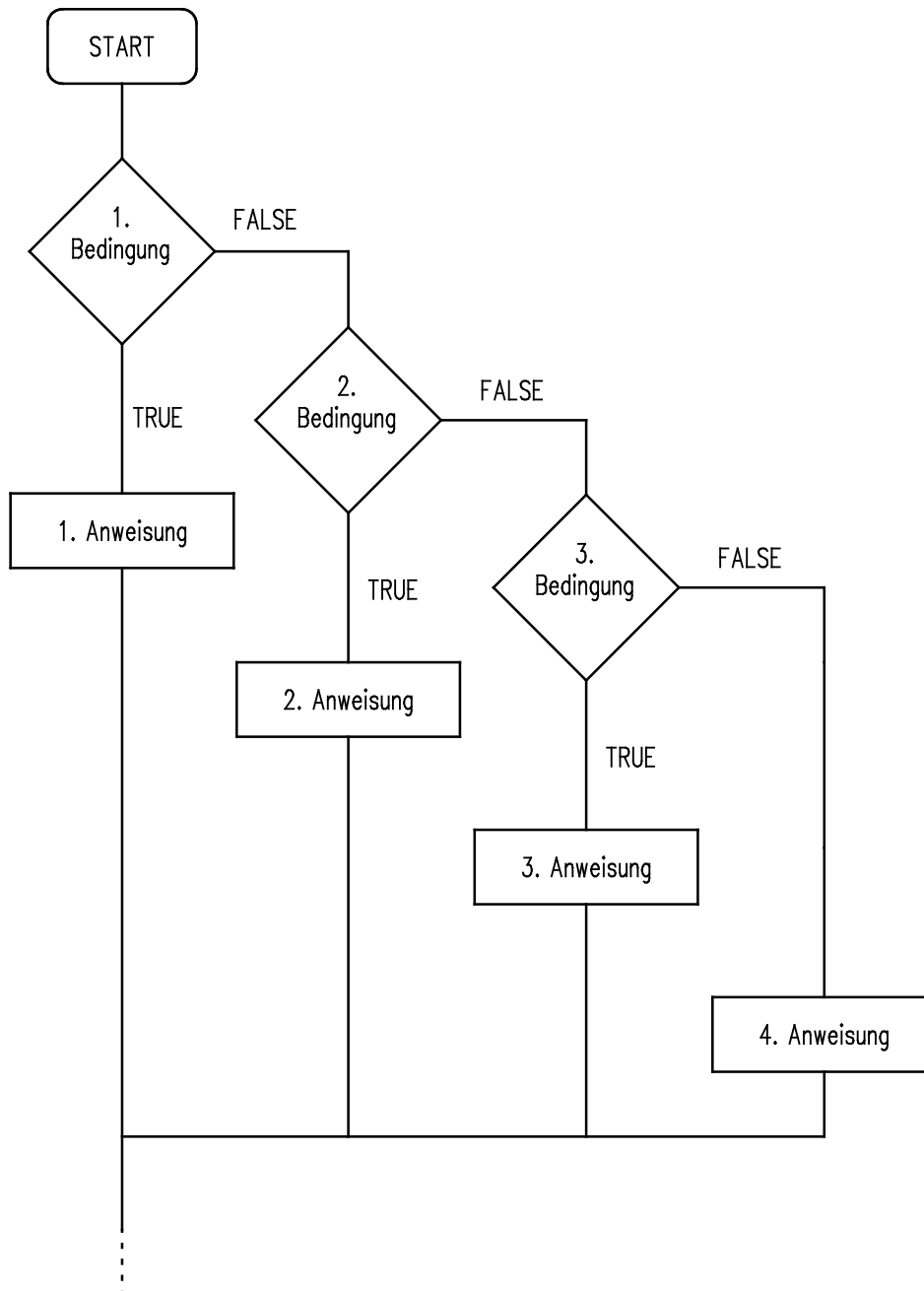


Abbildung 166 Verzweigung

Wenn die 1. Bedingung den booleschen Wert TRUE hat, wird die 1. Anweisung ausgeführt. Anderenfalls wird geprüft, ob die 2. Bedingung den booleschen Wert TRUE hat. Wenn dies zutrifft, wird die zweite Anweisung ausgeführt. Wenn nicht, wird geprüft, ob die 3. Bedingung den booleschen Wert TRUE hat. In diesem Fall wird die 3. Anweisung, ansonsten die 4. Anweisung ausgeführt.

Wenn also keine der drei Bedingungen erfüllt ist, wird die 4. Anweisung ausgeführt.

Die Programmierung dieser **Alternativverzweigung** in der Programmiersprache ST hat folgenden Aufbau.

```
IF          1. Bedingung THEN 1. Anweisung;
  ELSIF     2. Bedingung THEN 2. Anweisung;
  ELSIF     3. Bedingung THEN 3. Anweisung;
  ELSE                                     4. Anweisung;
END_IF;
```

Beispiel

```
IF    K > 0 THEN V := 1;
      ELSIF K := 0 THEN V := 2;
      ELSE V := 3;
END_IF;
```

Wenn die Variable K größer als Null ist, wird V = 1. Wenn K = 0, wird V = 2. Anderenfalls (d.h. K kleiner als Null) wird V = 3.

Lehrbeispiel

Erläutern Sie die Wirkungsweise des nachstehenden Programms!

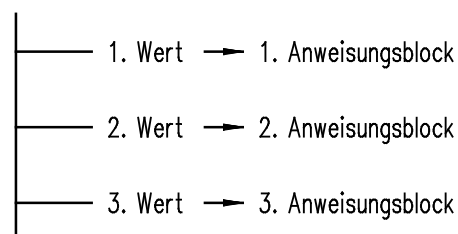
```
IF          TEMPERATUR < 0    THEN AUSGANG_1;
  ELSIF     TEMPERATUR > 10   THEN AUSGANG_2;
  ELSE                                     AUSGANG_3;
END_IF;
```

Wenn die Temperatur kleiner als Null ist, wird der AUSGANG_1 eingeschaltet. Wenn die Temperatur größer als 10 ist, wird der AUSGANG_2 eingeschaltet. Im Bereich größer oder gleich Null und kleiner oder gleich 10, wird der AUSGANG_3 eingeschaltet.

Multiauswahl CASE

Abhängig vom **Wert eines Ausdrucks** werden unterschiedliche Anweisungsblöcke bearbeitet.

Ausdruck



ELSE Anweisungsblock

Abbildung 167 Multiauswahl CASE

Wenn **keiner** der Werte zutrifft, wird der **ELSE-Anweisungsblock** bearbeitet. Sollte ein ELSE-Anweisungsblock **nicht** vorhanden sein, wird **kein** Anweisungsblock ausgeführt.

Struktur

CASE Ausdruck OF

```
1. Wert: 1. Anweisungsblock;  
2. Wert: 2. Anweisungsblock;  
3. Wert: 3. Anweisungsblock;  
ELSE Anweisungsblock;  
END_CASE;
```

Beispiel

CASE WERT_1

```
2: P := 0; C := 6;  
5: P := 6; C := 0;  
7,8: P := 6; C := 6;  
9,, 12: P := 0; C := 0;  
ELSE STOERUNG := 1;  
END_CASE;
```

Wenn WERT_1 = 2, dann P = 0 und C = 6.

Wenn WERT_1 = 5, dann P = 6 und C = 0.

Wenn WERT_1 = 7 oder 8, dann P = 6 und C = 6.

Wenn WERT_1 zwischen 9 und 12 liegt, dann P = 0 und C = 0.

Wenn WERT_1 keinen der obigen Werte annimmt, wird eine Störungsmeldung ausgegeben.

4.4 Programmierung von Schleifen

Schleifen werden programmiert, wenn Anweisungen **mehrfach wiederholt** werden sollen. Es wird zwischen der **Zählschleife FOR**, der **abweisenden Schleife WHILE** und der **nichtabweisenden Schleife REPEAT** unterschieden.

Zählschleife FOR

Von einem bestimmten **Anfangswert** ausgehend, wird eine ganzzahlige **Steuervariable** bei jedem Durchlauf des Anweisungsblocks um die angegebene **Schrittweite** erhöht oder erniedrigt (wenn die Schrittweite negativ ist), bis der ebenfalls angegebene **Endwert** erreicht ist.

Bei **ineinandergeschachtelten Schleifen** wird durch EXIT jeweils die **innerste Schleife** verlassen.

Die Zählschleife FOR...END_FOR eignet sich immer dann, wenn die **Anzahl der Schleifendurchläufe** im Vorfeld bekannt ist.

Bei der gewünschten Schrittweite 1 kann die Angabe der Schrittweite (BY 1) auch entfallen, da der **Standardwert** 1 ist.

Beispiel einer Zählschleife

```

FOR K := 1 TO 20 BY 1 DO
    :
END_FOR (* Ende der Zählschleife *)

```

Schrittweite
Endwert
Anfangswert
Steuervariable

Abbildung 168 Zählschleife

```

FOR K := 1 TO 20 BY 1 DO
    IF K := N THEN EXIT;
END_IF;
    WERT [K] := K + 5;
END_FOR;

```

Ausgehend vom Anfangswert 1, wird mit der Schrittweite 1 bis zum Endwert 20 der Anweisungsblock bis END_FOR durchlaufen. Die Anweisung EXIT veranlasst ein unverzügliches Verlassen der Schleife.

Die Anweisung EXIT

```

WHILE ...
    :
FOR ... TO ... BY ... DO
    :
IF ... THEN EXIT;
END_IF;
    :
END_FOR;
    :
END_WHILE;

```

Abbildung 169 Anweisung Exit

Wiederholungsanweisungen (REPEAT, WHILE, FOR) können vorzeitig mit Hilfe der Anweisung EXIT verlassen werden. Bei geschachtelten Anweisungen wird mit EXIT die innerste Wiederholungsanweisung verlassen und mit dem ersten Befehl der darüber liegenden Anweisung die Arbeit fortgesetzt.

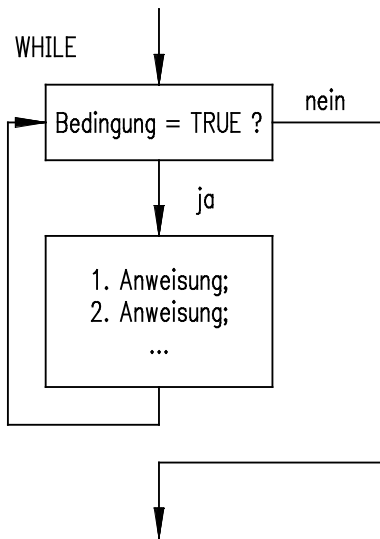
Abweisende Schleife WHILE

Abbildung 170 Abweisende Schleife

Bei der **abweisenden Schleife** wird der Anweisungsblock bei nicht **erfüllter Bedingung auch nicht einmal** durchlaufen. Die Wiederholungsbedingung wird **vor** dem Durchlaufen des Anweisungsblocks geprüft.

```

WHILE boolescher Ausdruck DO
    Anweisungsblock;
END_WHILE;
  
```

Solange der **boolesche Ausdruck** TRUE ist, wird der Anweisungsblock bearbeitet, der durch END_WHILE abgeschlossen wird.

Beispiel

```

K := 1;
WHILE K < 10 DO
    WERT [K] := K + 1;
    K := K + 2;
END_WHILE;
  
```

Solange K kleiner als 10 ist, wird der Anweisungsblock bearbeitet.

Nicht abweisende Schleife REPEAT

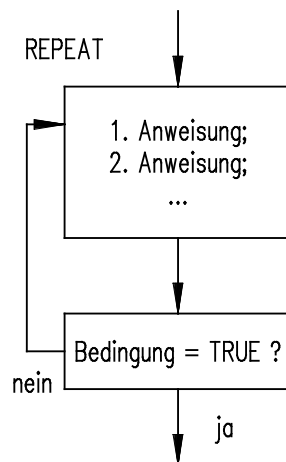


Abbildung 171 Nicht abweisende Schleife

Diese Schleife wird **mindestens einmal** durchlaufen. Die Schleifenbedingung wird **am Ende** des Anweisungsblocks geprüft.

```

REPEAT
  Anweisungsblock;
UNTIL boolescher Ausdruck
END_REPEAT;
  
```

Der Anweisungsblock wird ausgeführt, bis der **boolesche Ausdruck** den Wert FALSE annimmt.

Beispiel

```

K := 1
IF K < > V THEN
  REPEAT
    WERT [K] := K + 2;
    K := K + 1;
  UNTIL K > 20;
  END_REPEAT;
END_IF;
  
```

Schleifen dürfen keinesfalls als Warteschleifen auf externe Ereignisse verwendet werden. Bei unsachgemäßer Programmierung enden sie später als gedacht oder bilden sogar Endlosschleifen. Die Reaktionszeit der Steuerung wird dann unzulässig groß.

Aufgabe 1

Bestimmen Sie die Werte der Ausdrücke!

1.1 $(60 * 2 + 6) - 2 * (3 + 6)$

1.2 $(600 - (11 + 6 + 3 * 2) + 60) - 5 / 2 + 3 ** 4$

Aufgabe 2

Erläutern Sie die Wirkungsweise!

```
IF DRUCK > 20 THEN ROT;  
  ELSIF DRUCK < 10 THEN GELB;  
  ELSE GRUEN;  
END_IF;
```

Aufgabe 3

Erläutern Sie die Wirkungsweise!

```
CASE VAR_6  
  2:   A:=0 , B:=6;  
  6,7: A:=6 , B:=0;  
  9,,12: A:=6 , B:=6;  
      ELSE C:=1;  
END_CASE
```

Aufgabe 4

Unterscheiden Sie zwischen einer „abweisenden Schleife“ und einer „nicht abweisenden Schleife“!

Aufgabe 5

Um welche Schleife handelt es sich?

```
WERT :=1;  
WHILE WERT < 50 DO  
  VARIABLE [WERT] := WERT +1;  
  WERT := WERT + 2;  
END_WHILE;
```

Aufgaben

Aufgabe 6

Erläutern Sie die Arbeitsweise des Funktionsbausteins!

```

FUNCTION_BLOCK BLINKER
VAR_INPUT
    FREIGABE : BOOL;
    EIN : TIME;
    AUS : TIME;
END_VAR

VAR_OUTPUT
    BLINK_AUSGANG : BOOL;
END_VAR

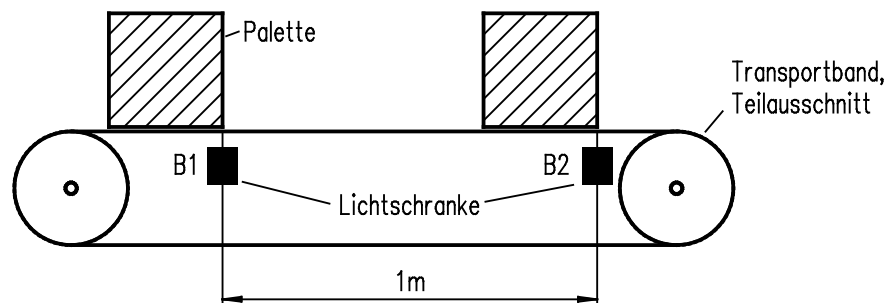
VAR
    HELL,DUNKEL : TP;
END_VAR

HELL ( IN := FREIGABE & NOT DUNKEL.Q , PT := EIN );
DUNKEL ( IN := NOT HELL.Q , PT := AUS );
BLINK_AUSGANG := HELL.Q;

END_FUNCTION_BLOCK
    
```

Aufgabe 7

Überwachung der Geschwindigkeit eines Transportbandes.
Auf einem Transportband werden Paletten transportiert. Wenn die Geschwindigkeit 0,2 m/s unterschreitet, soll eine Alarmmeldung gegeben werden.



Die beiden Lichtschranken sind in einem Abstand von 1 m angeordnet und haben Öffnerfunktion. Bei einer zulässigen Geschwindigkeit von 0,2 m/s benötigt die Palette von Lichtschranke zu Lichtschranke eine Zeit von 5 s. Bei Unterschreitung der Geschwindigkeit soll eine Alarrmeldung ausgegeben werden.

Erstellen Sie das Steuerprogramm in der Programmiersprache ST!

Der Deklarationsteil lautet wie folgt:

```

VAR_INPUT
    LICHT_SCHR_1 : BOOL; (*Lichtschranke1, Öffner*)
    LICHT_SCHR_2 : BOOL; (*Lichtschranke2, Öffner*)
END_VAR
    
```

```
VAR_OUTPUT
  ALARM_MELD : BOOL;    (*Alarmmeldung bei Zeitüberschreitung*)
END_VAR
```

```
VAR
  ZEIT_MERK : BOOL;      (*Von den Lichtschranken gelesener Merker*)
  ZEIT_SPEICH : SR;      (*Instanz für ZEIT_MERK*)
  LAUFZEIT : TON;        (*Instanz für Überwachungszeit*)
END_VAR
```

Aufgabe

Nach der Erarbeitung der Grundlagen erstellen Sie zu dem technischen Problem „Verkaufsautomat“ eine Lösung mit der Programmiersprache „Strukturierter Text“!

**Realisierung
Projekt 3
„Verkaufsautomat“**

Lösungen

Lösungsanhang

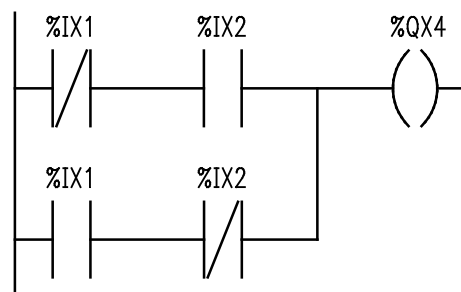
2 Realisierung einer Steuerung mit Verknüpfungselementen (KOP, AWL, FBS)

Aufgabe 1

Anweisungsliste:

```
LDN    %IX1
AND    %IX2
OR(    %IX1
ANDN   %IX2
)
ST     %QX4
```

Kontaktplan:



Aufgabe 2

```
LD      EIN_4
ANDN    EIN_5
OR      EIN_6
OR(     EIN_2
ANDN    EIN_1
AND     EIN_3
)
ANDN    EIN_7
ST      AUS_1
```

Aufgabe 3

```
LD      START
AND     FREIGABE
ST      SPEICHER_1.S
LDN     STOP
ORN     HALT
ST      SPEICHER_1.R1

CAL     SPEICHER_1

LD      SPEICHER_1.Q1
ST      AUSG_1
```

Andere Schreibweise:

```
CAL  SPEICHER_1 (S:=START AND FREIGABE, R1:=NOT STOP OR NOT HALT)
LD   SPEICHER_1.Q1
ST   AUSG_1
```

Aufgabe 4

Es handelt sich um eine Einschaltverzögerung (Verzögerungszeit 60 s). Wenn die Variable START den booleschen Wert TRUE annimmt, wird die Zeitverzögerung gestartet. Nach Ablauf der Verzögerungszeit nimmt die Variable AUS den booleschen Wert TRUE an.

Aufgabe 5

1. Darstellungsform

```
CAL  ZEIT_1 (IN:=START, PT:=t#60s)
LD   ZEIT_1.Q
ST   AUS
```

2. Darstellungsform

```
LD   t#60s
PT   ZEIT_1
LD   START
IN   ZEIT_1
LD   ZEIT_1.Q
ST   AUS
```

Aufgabe 6

1. Darstellungsform

```
LD   t#5s
ST   TIMER_1.PT
LD   EIN_1
ANDN EIN_2
ST   TIMER_1.IN

CAL  TIMER_1

LD   TIMER_1.Q
ST   AUS
LD   TIMER_1.ET
ST   ZEIT_WERT
```

2. Darstellungsform

```
CAL  TIMER_1 (IN:=EIN_1 AND NOT EIN_2, PT:=t#5s)
LD   TIMER_1.Q
ST   AUS
LD   TIMER_1.ET
ST   ZEIT_WERT
```

3. Darstellungsform

```
LD      t#5s
PT      TIMER_1
LD      EIN_1
ANDN    EIN_2
IN      TIMER_1
LD      TIMER_1.Q
ST      AUS
LD      TIMER_1.ET
ST      ZEIT_WERT
```

Allen Darstellungsformen ist folgende Deklaration voranzustellen:

```
VAR
    EIN_1: BOOL;
    EIN_2: BOOL;
    AUS: BOOL;
    ZEIT_WERT: TIME;
    TIMER_1: TOF;
END_VAR
```

Aufgabe 7

```
LD      Z_IMPULS
ST      ZAEHLER_1.CU
LD      RESET
ST      ZAEHLER_1.R
LD      Z_WERT
ST      ZAEHLER_1.PV

CAL      ZAEHLER_1

LD      ZAEHLER_1.Q
ST      AUS
LD      ZAEHLER_1.CV
ST      WERT
```

Aufgabe 8

```
LD      ZAEHL_AUF
ST      ZAEHLER_2.CU
LD      ZAEHL_AB
ST      ZAEHLER_2.CD
LD      LOESCHEN
ST      ZAEHLER_2.R
LD      LADEN
ST      ZAEHLER_2.LD
LD      ZAEHL_WERT
ST      ZAEHLER_2.PV

CAL      ZAEHLER_2

LD      ZAEHLER_2.QU
ST      AUS_AUF
LD      ZAEHLER_2.QD
ST      AUS_AB
LD      ZAEHLER_2.CV
ST      ISTWERT
```


Aufgabe 9

```
VAR_INPUT
    START, STOP: BOOL;
END_VAR
```

```
VAR_OUTPUT
    STERN_BETRIEB: BOOL;
    NETZ_BETRIEB: BOOL;
    DREIECK_BETRIEB: BOOL;
END_VAR
```

```
VAR
    ANLAUF_ZEIT: BOOL;
    STERN: RS;
    NETZ: RS;
    DREIECK: RS;
    ANLAUF: TON;
END_VAR
```

```
LD    START
ST    STERN.S
LD    ANLAUF_ZEIT
ORN   STOP
ST    STERN.R1
```

```
CAL    STERN
```

```
LD    STERN.Q1
ST    STERN_BETRIEB
LD    STERN_BETRIEB
ST    NETZ.S
LDN   STOP
ST    NETZ.R1
```

```
CAL    NETZ
```

```
LD    NETZ.Q1
ST    NETZ_BETRIEB
```

```
LD    ANLAUF_ZEIT
ST    DREIECK.S
LDN   STOP
ST    DREIECK.R1
```

```
CAL    DREIECK
```

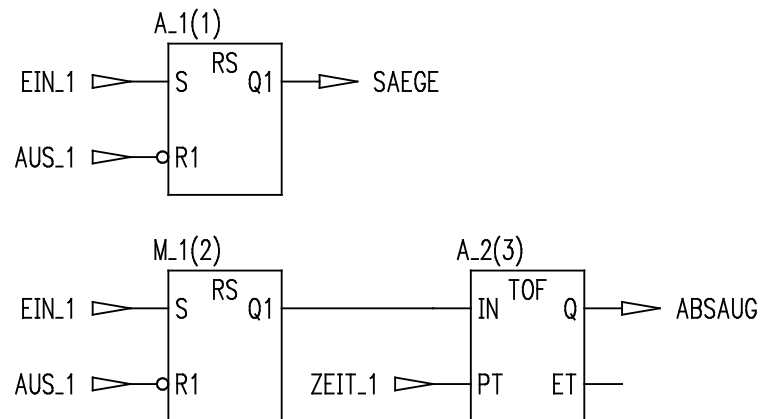
```
LD    DREIECK.Q1
ST    DREIECK_BETRIEB
```

```
LD    t#3s
ST    ANLAUF.PT
LD    STERN_BETRIEB
AND   NETZ_BETRIEB
ST    ANLAUF.IN
```

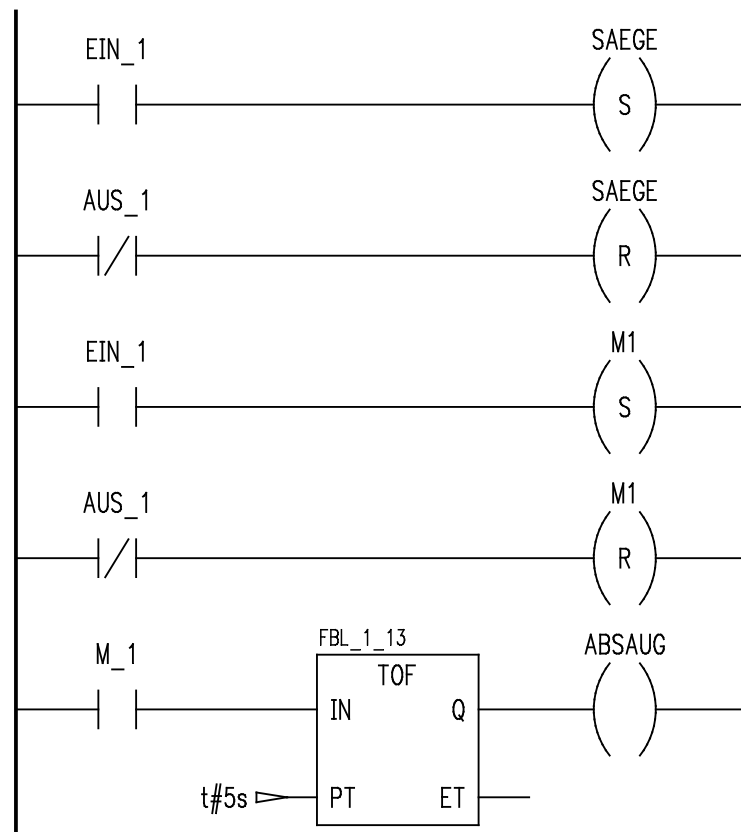
```
CAL    ANLAUF
```

```
LD    ANLAUF.Q
ST    ANLAUF_ZEIT
```

Projekt 1 „Säge mit Absaugvorrichtung“



Darstellung der Section SAEGE in KOP:



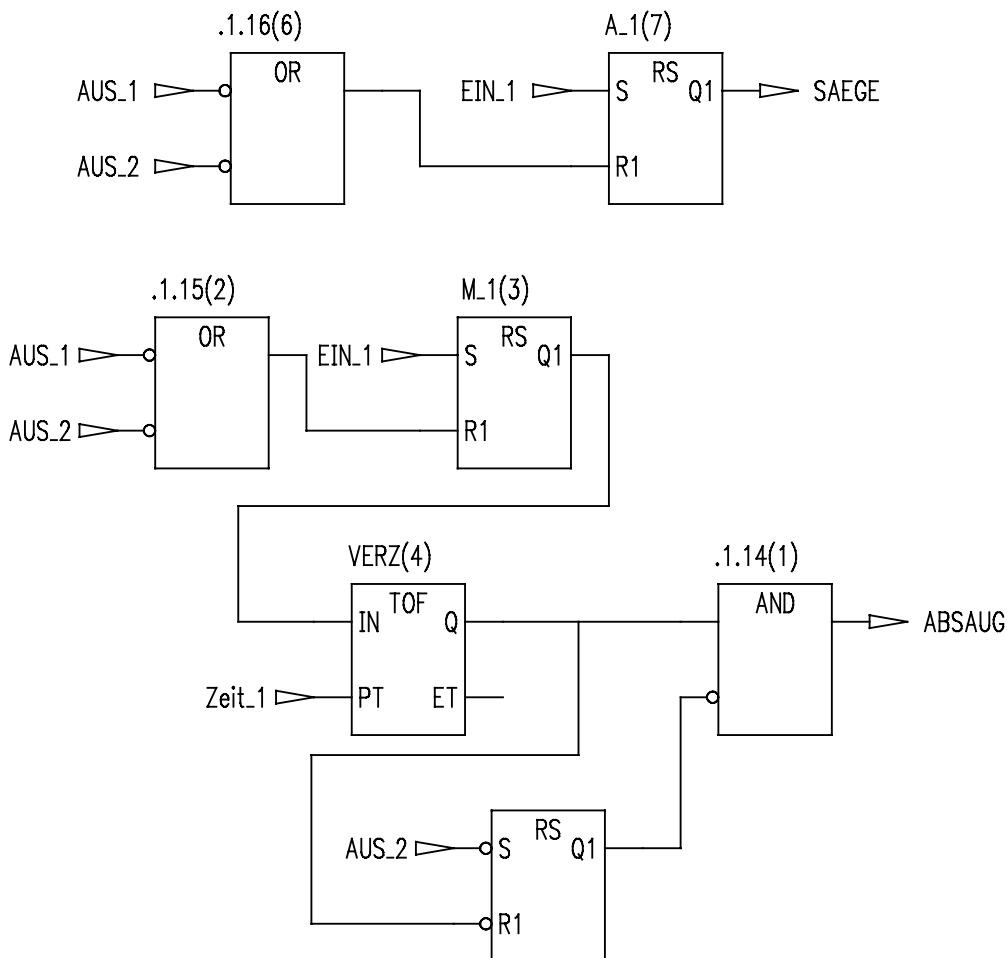
Darstellung mit AWL:

(* Darstellung der Section SAEGE in AWL *)

```

VAR
  VERZ : TOF;
END_VAR
LD  EIN_1
S   SAEGE
LDN AUS_1
R   SAEGE
LD  EIN_1
S   M1
LDN AUS_1
R   M1
CAL VERZ(IN:=M1,PT:=t#5s)
LD  VERZ.Q
ST  ABSAUG

```

NOT-AUS-Erweiterung

3 Realisierung einer Steuerung durch eine Ablaufsprache (AS)

Aufgabe 1.1

allgemeines Schrittsymbol

Aufgabe 1.2

Schrittsymbol mit Kennzeichnung (hier: Nummer 3)

Aufgabe 1.3

Schritt Nummer 6 in einen bestimmten Prozesszustand gesetzt

Aufgabe 1.4

Anfangsschritt (Initialisierungsschritt); hier Nummer 1

Aufgabe 2

Durch eine Transition (Übergangsbedingung) werden zwei Schritte miteinander verbunden. Wenn die Transition erfüllt ist, wird der nachfolgende Schritt aktiviert und der Vorgängerschritt deaktiviert.

Aufgabe 3

Die Transition zwischen den Schritten 6 und 7 ist erfüllt, wenn entweder

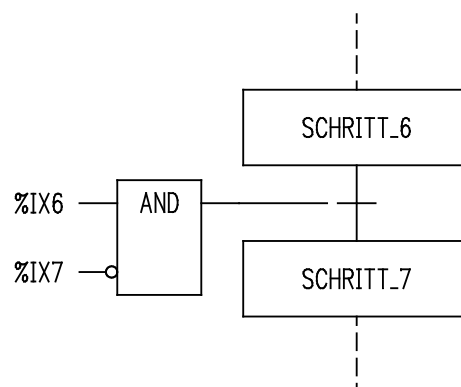
- I1 = „1“ und I2 = „1“

ODER

- I3 = „0“ und I4 = „0“

Wenn die Transition erfüllt ist, erfolgt der Übergang von Schritt 6 nach Schritt 7.

Aufgabe 4



Aufgabe 5

Der Schritt 5 kann nicht aktiv werden. Wenn die Übergangsbedingung (Transition)

- FREIGABE & RECHTS

erfüllt ist (den booleschen Zustand TRUE hat), dann ist auch die nachfolgende Transition

- FREIGABE

erfüllt.

Die Ablaufkette „rutscht“ von Schritt 4 direkt in Schritt 6.

Aufgabe 6

INITIAL_STEP INIT: END_STEP

TRANSITION FROM INIT TO PRESS
:= STEU_EIN_AUS & NOT BOHR_SPIN_OBEN & BOHRSTART;
END_TRANSITION

STEP PRESS: END_STEP

TRANSITION FROM PRESS TO ANLAUF
:= NOT SCHUTZGITTER & NOT SPANNEN_1 & NOT SPANNEN_2;
END_TRANSITION

STEP ANLAUF: END_STEP

TRANSITION FROM ANLAUF TO SENK
:= BESCHL_ZEIT;
END_TRANSITION

STEP SENK: END_STEP

TRANSITION FROM SENK TO HEB
:= NOT BOHR_SPIN_UNTEN;
END_TRANSITION

STEP HEB: END_STEP

TRANSITION FROM HEB TO INIT
:= NOT BOHR_SPIN_OBEN;
END_TRANSITION

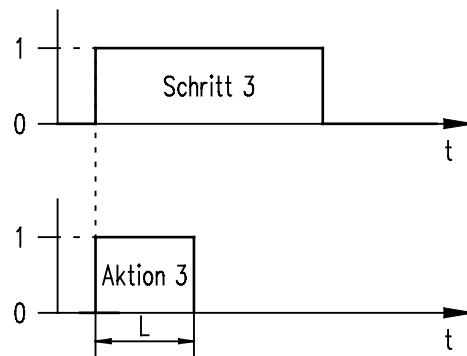
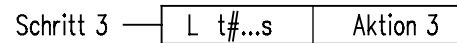
Aufgabe 7

Die N-Aktion wird nur an dem Schritt ausgeführt, an dem sie „hängt“.

Die S-Aktion wird über mehrere Schritte hinweg (schrittübergreifend) ausgeführt, bis sie ausdrücklich wieder zurückgenommen wird.

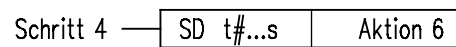
Aufgabe 8

Das Bestimmungszeichen lautet L (limited; zeitbegrenzt). Nach der programmierten Zeit wird die Aktion 3 zurückgenommen, auch wenn Schritt 3 weiterhin aktiv bleibt.

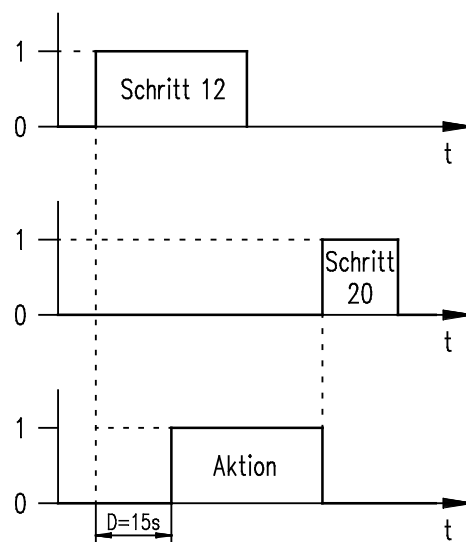
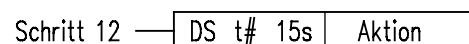
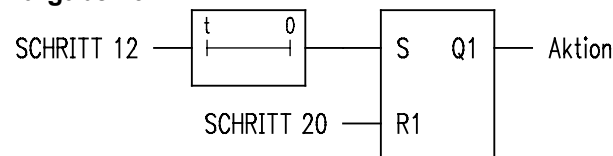


Aufgabe 9

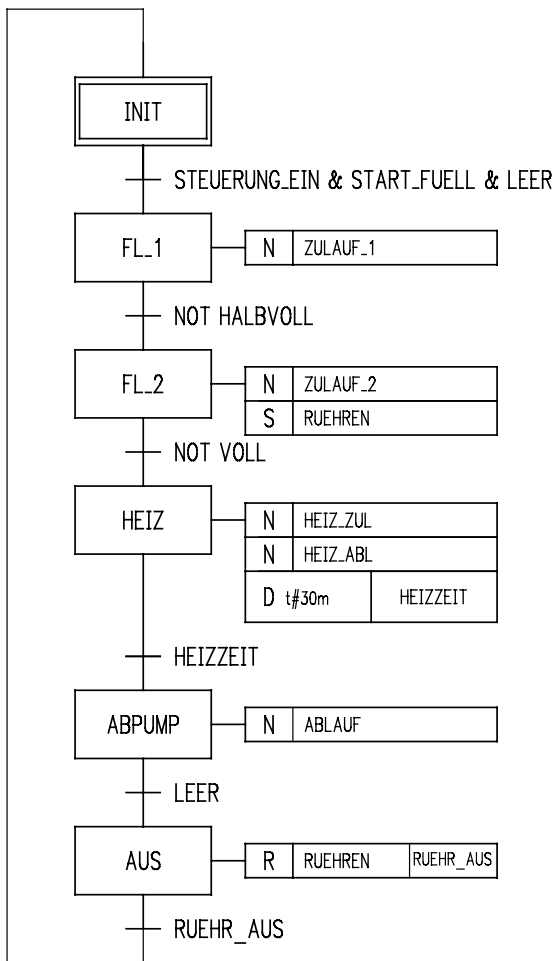
Es handelt sich um eine SD-Aktion. Die Aktion 6 wird um die programmierte Zeit verzögert nach Aktivierung von Schritt 4 speichernd gesetzt. Schritt 8 nimmt die Aktion wieder zurück.



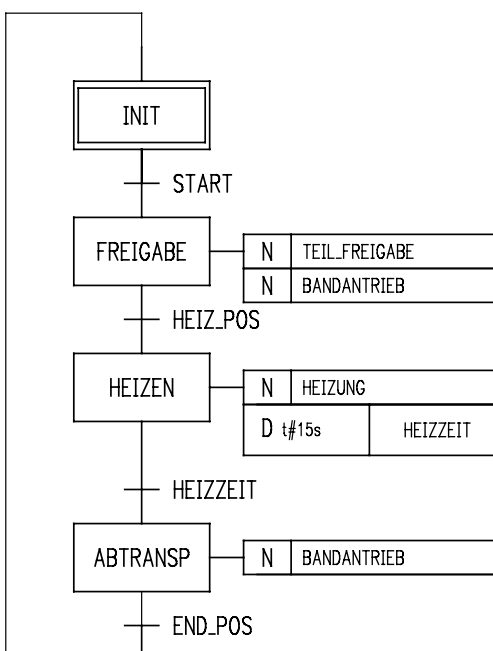
Aufgabe 10



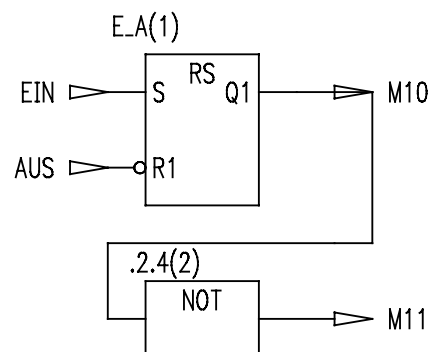
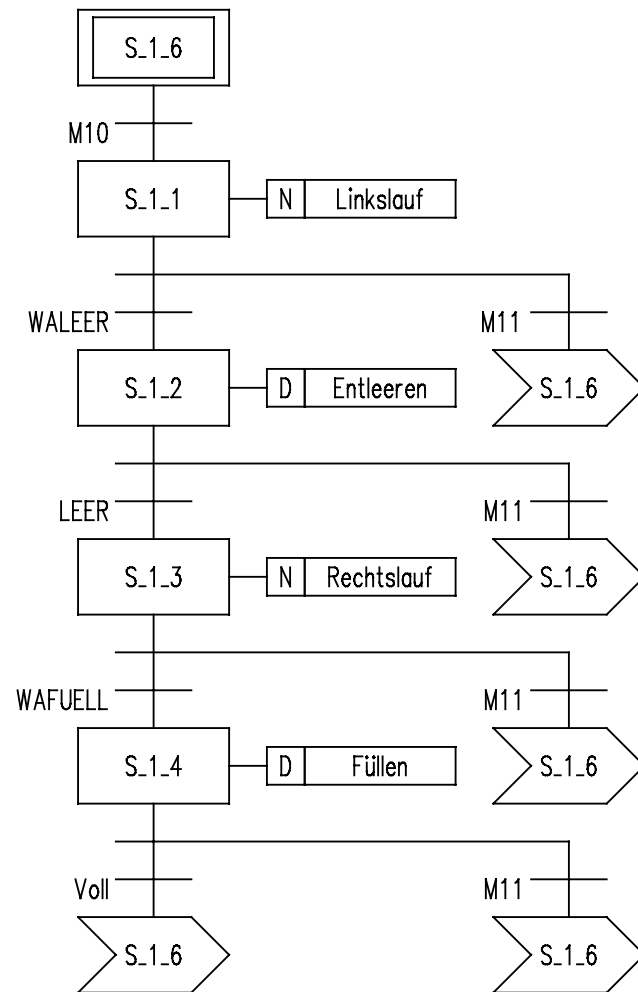
Aufgabe 11



Aufgabe 12



Projekt 2 „Verladeanlage“



4 Realisierung einer Steuerung mittels strukturiertem Text (ST)**Aufgabe 1.1**

108

Aufgabe 1.2

715,5

Aufgabe 2

Wenn DRUCK größer als 20 (bar) ist, nimmt die Variable ROT den booleschen Wert TRUE an. Wenn DRUCK kleiner als 10 (bar) ist, nimmt die Variable GELB den booleschen Wert TRUE an; ansonsten nimmt die Variable GRUEN den booleschen Wert TRUE an.

Aufgabe 3

Wenn VAR_6=2, dann A=0 und C=6.
Wenn VAR_6=6 oder 7, dann A=6 und B=0.
Wenn VAR_6 zwischen 9 und 12 liegt, dann A=6 und B=6.
Wenn keiner der oberen Werte zutrifft, dann C=1.

Aufgabe 4

Abweisende Schleife: Der Anweisungsblock wird bei nicht erfüllter Bedingung nicht durchlaufen. Bedingungsabfrage steht vor Anweisungsblock.

Nicht abweisende Schleife: Der Anweisungsblock wird mindestens einmal durchlaufen, auch bei nicht erfüllter Bedingung. Bedingungsabfrage steht nach Anweisungsblock.

Aufgabe 5

Es handelt sich um eine abweisende Schleife. Solange WERT kleiner als 50 ist, wird der Anweisungsblock bearbeitet.

Aufgabe 6

Es handelt sich um einen Blinkgeber mit zwei Zeitgliedern, die sich gegenseitig starten. Wenn das erste Zeitglied abgelaufen ist, wird das zweite Zeitglied gestartet und umgekehrt. Der Name dieses Funktionsbausteins (anwenderdefiniert) lautet BLINKER.

Die Variablen EIN und AUS enthalten die Verzögerungszeiten für die beiden Zeitglieder. Die Variable FREIGABE startet den BLINKER.

Aufgabe 7

FUNCTION_BLOCK UEBERWACHUNG
(*DEKLARATION WIE IN AUFGABENSTELLUNG*)

ZEIT_SPEICH (S1:=NOT LICHT_SCHR_1, R:=NOT LICHT_SCHR_2);
ZEIT_MERK:=ZEIT_SPEICH.Q1;

LAUFZEIT (IN:=ZEIT_MERK, PT:=t#5800ms);
ALARM_MELD:=LAUFZEIT.Q;

END_FUNCTION_BLOCK

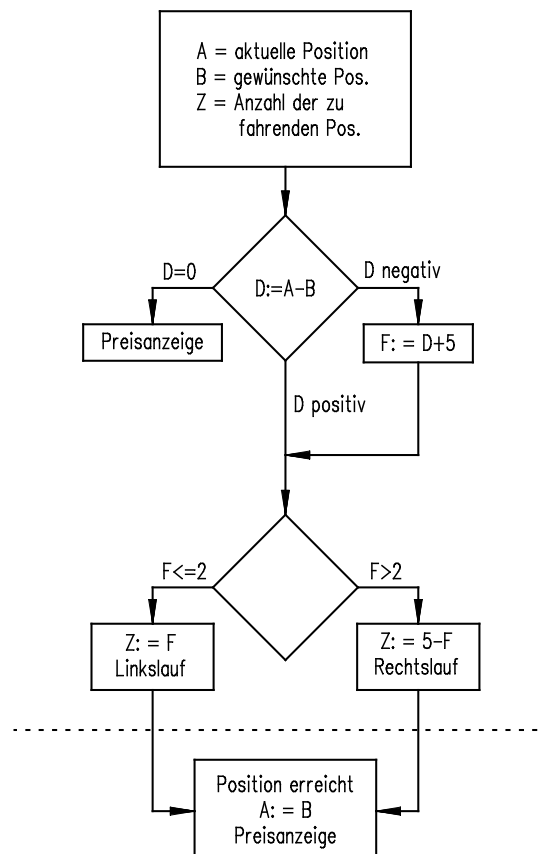
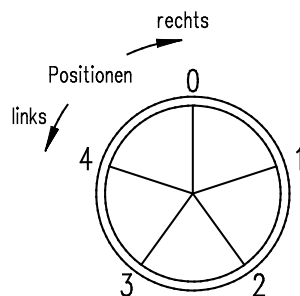
Hinweis:

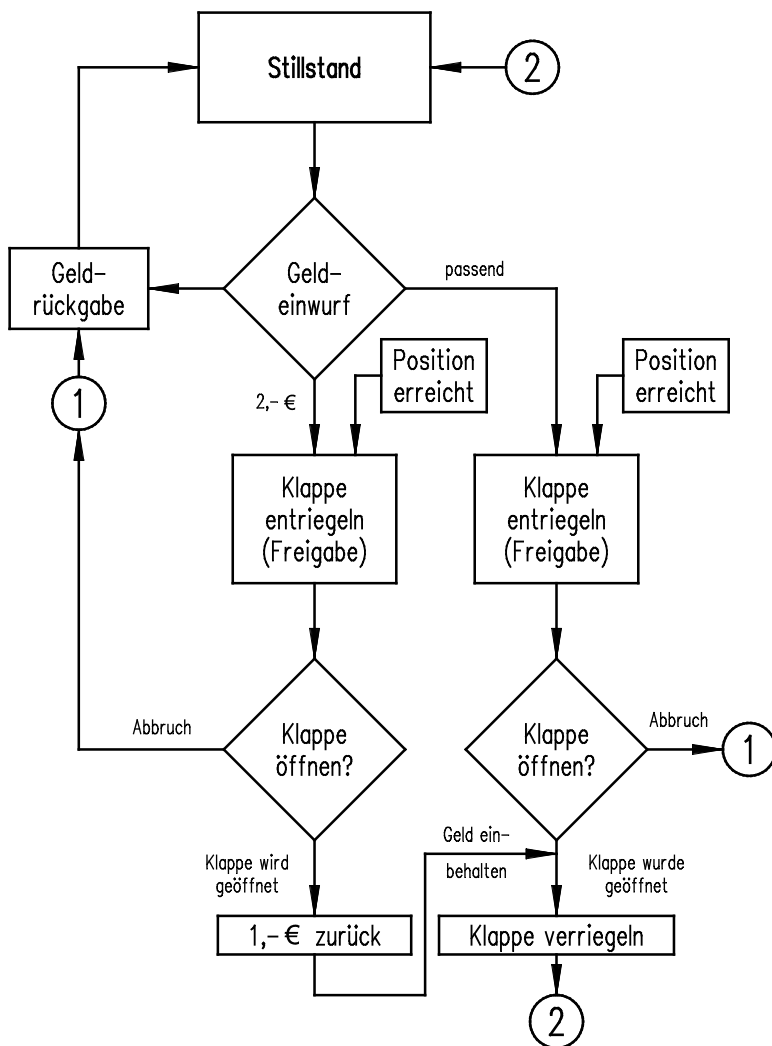
Bei der Lösung wurde nicht berücksichtigt, dass die Palette beim Durchlaufen der Lichtschranke diese während der gesamten Palettenlänge unterbricht. Wenn dies zu Problemen führt, kann mit Flankenerkennung gearbeitet werden. Bei Unterbrechung der Lichtschranke (Öffnerfunktion) wird eine abfallende Flanke geliefert.

Folgende Änderungen sind damit erforderlich:

VAR_INPUT
LICHT_SCHR_1: BOOL F_EDGE; (*fallende Flanke*)
LICHT_SCHR_2: BOOL F_EDGE; (*fallende Flanke*)
END_VAR

Projekt 3 „Verkaufsautomat“





(* Einschaltroutine *)
 IF NOT M_EIN THEN
 A1 := FALSE;
 A2 := FALSE;
 A3 := FALSE;
 A4 := FALSE;
 A5 := TRUE; (* Rechtslauf ein *)
 A6 := FALSE;
 A7 := FALSE;
 A8 := FALSE;
 A11 := FALSE;
 A12 := FALSE;
 A13 := FALSE;
 A14 := FALSE;
 A15 := FALSE;
 A := 0;
 E1 := FALSE;
 E2 := FALSE;
 E3 := FALSE;
 E4 := FALSE;
 E5 := FALSE;
 E6 := FALSE;
 E7 := FALSE;

```

E8 := FALSE;
E9 := FALSE;
E11 := FALSE;
E12 := FALSE;
E13 := FALSE;
Z := -1;
M11 := 0;

END_IF;

IF E10 AND NOT M_EIN THEN
  A5 := FALSE;
  M_EIN := TRUE;
  SUMME := 0;
  SELECT := 5;
END_IF;

IF M_EIN THEN (* Hauptprogramm *)

(* Gebindeauswahl *)
IF E1 AND NOT A11 THEN
  B := 0;
ELSIF E2 AND NOT A12 THEN
  B := 1;
ELSIF E3 AND NOT A13 THEN
  B := 2;
ELSIF E4 AND NOT A14 THEN
  B := 3;
ELSIF E5 AND NOT A15 THEN
  B := 4;
END_IF;

(* Richtungsauswahl *)
D := A - B;

IF D < 0 THEN
  F := D + 5;
ELSIF D = 0 THEN
  A7 := TRUE; (* Preisanzeige *)
  POS_ERR := TRUE;
END_IF;
W := F - 2;

IF W > 0 THEN
  Z := 5 - F;
  A5 := TRUE; (* Rechtslauf *)
  POS_ERR := FALSE;

ELSIF W <= 0 THEN
  Z := F;
  A6 := TRUE; (* Linkslauf *)
  POS_ERR := FALSE;
END_IF;

(* Fahren in die gewünschte Position *)
WHILE NOT (A - B = 0) DO
  IF Z > 0 THEN
    Z := Z - M11;
  END_IF;

```

```
IF Z=0 THEN
  A5 := FALSE;
  A6 := FALSE;
  A := B;
  POS_ERR := TRUE;
  A7 := TRUE;
END_IF;
END_WHILE;

(* Geldauswertung *)
IF E7 THEN
  SUMME := SUMME + 0,5;
ELSIF E8 THEN
  SUMME := SUMME + 1;
ELSIF E9 THEN
  SUMME := SUMME + 2;
END_IF;

(* Verzweigung zur Klappenfreigabe *)
IF SUMME - 3 = 0 THEN
  F_1 := TRUE; (* Freigabe 1 *)
  SUMME := 0;
ELSIF SUMME - 4 = 0 THEN
  F_2 := TRUE; (* Freigabe 2 *)
  SUMME := 0;
END_IF;

(* Klappenfreigabe und Geldeinzug *)
(* Routine für 3 € *)
IF F_1 AND POS_ERR THEN
  A1 := TRUE; (* Klappe freigeben *)
END_IF;

IF F_1 AND NOT E12 THEN
  A1 := FALSE;
  F_1 := FALSE;
  A2 := TRUE; (* Geld einbehalten *)
END_IF;

(* Routine für 4 € *)
IF F_2 AND POS_ERR THEN
  A1 := TRUE; (* Klappe freigeben *)
END_IF;

IF F_2 AND NOT E12 THEN
  A1 := FALSE;
  F_2 := FALSE;
  A2 := TRUE; (* Geld einbehalten *)
  A4 := TRUE; (* 1 € zurück *)
END_IF;

(* Beenden des Geldeinzuges und der Ausgabe des Wechselgeldes *)
IF A2 AND E12 THEN
  A4 := FALSE; (* Wechselgeld *)
  A2 := FALSE; (* Geldeinzug *)
  A7 := FALSE; (* Preisanzeige *)
END_IF;
```

```

(* Sperre der Gebindeauswahl *)
IF A2 AND NOT E12 THEN
  SELECT := A;
END_IF;

CASE SELECT OF 0: A11 := TRUE;
              1: A12 := TRUE;
              2: A13 := TRUE;
              3: A14 := TRUE;
              4: A15 := TRUE;
END_CASE;

(* Aufheben der Sperre *)
IF E8 OR NOT M_EIN THEN
  A11 := FALSE;
  A12 := FALSE;
  A13 := FALSE;
  A14 := FALSE;
  A15 := FALSE;
END_IF;

(* Abbruchroutine *)
IF E6 THEN
  SUMME := 0;
  A3 := TRUE; (* Geldrückgabe *)
  A7 := FALSE; (* Preisanzeige *)
  A1 := FALSE; (* Klappe verriegeln *)
ELSIF NOT E6 THEN
  A3 := FALSE;
END_IF;

(* Wechselgeld auffüllen *)
IF E13 THEN
  WECHSEL := 10; (* Zehn 1 €-Stücke aufgefüllt *)
  A8 := FALSE; (* Wechselgeld vorhanden *)
END_IF;

IF A4 THEN
  WECHSEL := WECHSEL - 1;
ELSIF WECHSEL = 0 THEN
  A8 := TRUE; (* Kein Wechselgeld vorhanden *)
END_IF;

END_IF; (* Ende Hauptprogramm *)

```