7 Beschreibungsmittel Programmablaufplan und Struktogramm

Digitale Steuerungsaufgaben, deren Lösung auf der Anwendung eines Algorithmus basieren, können vorteilhaft mit grafischen Ablaufstrukturen beschrieben werden. Ein Algorithmus ist dabei eine Berechnungsregel, welche aus mehreren elementaren Schritten besteht, die in einer bestimmten Reihenfolge ausgeführt werden müssen. Die Beschreibung eines Algorithmus erfolgt durch die Aufzählung der auszuführenden Schritte, sowie der Vorschrift, in welcher Reihenfolge die einzelnen Schritte durchgeführt werden müssen. Der sich dabei ergebende Programmablauf kann mit grafischen Darstellungsmethoden als Programmablaufplan PAP (Flow-Chart) oder Struktogramm STG (PSD Program Structure Diagram) beschrieben werden.

Die Standards der beiden Darstellungsmethoden sind:

- DIN 66261: Sinnbilder für Struktogramme nach Nassi-Shneidermann.
- DIN 66001: Richtlinien zur Gestaltung von Programmablaufplänen.
- ISO/IEC 8631: 1989 Information technology-Program constructs and conventions for their representation.
- ISO 5807: 1985 Information processing-Documentation symbols and conventions for date, program and system flowcharts, program network charts and system resources charts.

Die Analyse von digitalen Steuerungsprogrammen, denen ein Algorithmus zu Grunde liegt, hat gezeigt, dass sich immer wieder die drei folgenden Ablaufstrukturen (Strukturblöcke) ergeben:

- Folge (Sequenz): die Verarbeitung von Schritten nacheinander;
- Auswahl (Selektion): die Auswahl von bestimmten Schritten;
- Wiederholung (Iteration): die Wiederholung von Schritten.

Mit diesen drei elementaren Ablaufstrukturen können die Beschreibungsmittel Programmablaufplan PAP oder Struktogramm STG nach Festlegung des Algorithmus zur Lösung von Steuerungsaufgaben bzw. Analyse von bestehenden Steuerungsprogrammen eingesetzt werden.

Die Darstellung von Programmabläufen erfolgt durch *Sinnbilder*. Bei der Methode Programmablaufplan sind dies: Operation (Kasten), Verzweigung (Route), Ablauflinie und Zusammenführung (von Ablauflinien), die zu so genannten *Programmkonstrukten* zusammengesetzt werden. Bei der Methode Struktogramm sind es zweipolige *Strukturblöcke* (1 Eingang, 1 Ausgang), deren Bedeutung man leichter versteht, wenn man sie als Ersatz für PAP-Programmkonstrukte einführt. Der Programmablauf wird durch die Auswahl der Sinnbilder und deren Schachtelung dargestellt.

Das schon etwas ältere Darstellungsmittel Programmablaufplan ist in Verruf geraten, da sehr unübersichtliche Reihenfolgen des Programmablaufs insbesondere durch eine Vielzahl von Sprüngen entstehen können. Geeignet ist der Programmablaufplan jedoch noch immer zur Darstellung maschinennaher Sprachen. Deshalb lässt sich die Anweisungsfolge eines Steuerungsprogramms, das in der Programmiersprache Anweisungsliste AWL geschrieben ist, durch einen Programmablaufplan gut beschreiben. Für eine allgemeine Darstellung des verwendeten Algorithmus einer Steuerungsaufgabe und für die Darstellung von Programmen, die in der Programmiersprache SCL geschrieben ist aber in jedem Fall das Struktogramm als Beschreibungsmittel das Geeignetere.

7.1 Programmablaufplan

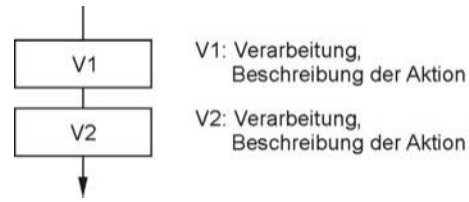
In Programmablaufplänen werden die Schritte symbolisch durch Sinnbilder dargestellt und der Steuerungsablauf durch Ablauflinien hergestellt. Die wichtigen Darstellungselemente des Programmablaufplans lassen sich in die Ablaufstrukturen Verarbeitung, Folge, Auswahl und Wiederholung unterteilen.

7.1.1 Programmkonstrukt Verarbeitung

Die Darstellung des Programmkonstrukts besteht nur aus einem Block, der genau einmal ausgeführt wird.

7.1.2 Programmkonstrukt Folge

Die Darstellung der Programmkonstrukts Folge oder Sequenz enthält zwei oder mehrere Verarbeitungsteile, die genau je einmal ausgeführt werden, wenn das Element abgearbeitet wird.

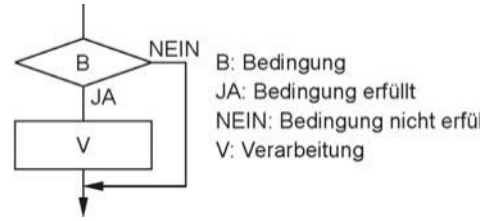


7.1.3 Programmkonstrukt Auswahl

Mit dem Programmkonstrukt Auswahl werden alternative Verarbeitungen beim logischen Ablauf des Programms möglich. Hierbei ergeben sich drei unterschiedliche Formen.

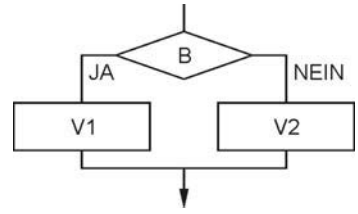
a) Bedingte Verarbeitung

Das Programmkonstrukt besteht aus einem Verarbeitungsteil und einem Steuerungsteil mit einer Bedingung. Die Bedingung bestimmt, ob der Vereinbarungsteil ausgeführt wird, wenn das Element abgearbeitet wird.



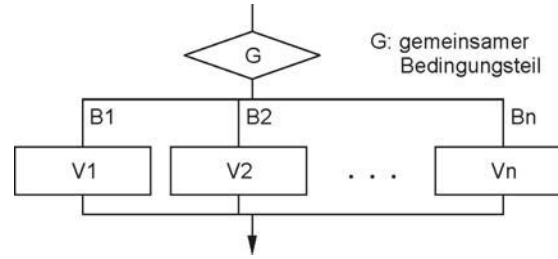
b) Einfache Alternative

Das Programmkonstrukt besteht aus zwei Verarbeitungsteilen und einem Steuerungsteil mit einer Bedingung. Der Steuerungsteil gibt mit der Bedingung an, welcher der beiden Verarbeitungsteile ausgeführt wird, wenn das Element abgearbeitet wird.



c) Mehrfache Alternative

Das Programmkonstrukt besteht aus mindestens drei Verarbeitungsteilen und einem Steuerungsteil mit der gleichen Anzahl einander ausschließender Bedingungen. Der Steuerungsteil gibt mit diesen Bedingungen an, welcher der Verarbeitungsteile ausgeführt wird, wenn das Element abgearbeitet wird.

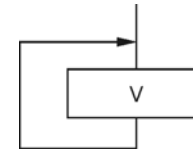


7.1.4 Programmkonstrukt Wiederholung

Mit dem Programmkonstrukt Wiederholung, auch Schleife oder Iteration genannt, wird ein Verarbeitungsteil solange wiederholt, wie es der Steuerungsteil vorgibt. Wird eine Wiederholung in Abhängigkeit von einer Bedingungsprüfung ausgeführt, so kann diese vor oder nach dem Verarbeitungsteil stehen.

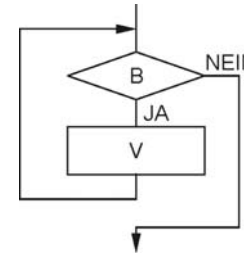
a) Wiederholung ohne Bedingungsprüfung

Das Programmkonstrukt enthält nur einen Verarbeitungsteil. Dieser wird endlos wiederholt ausgeführt, wenn das Element abgearbeitet wird.



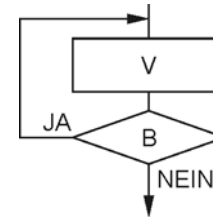
b) Wiederholung mit vorausgehender Bedingungsprüfung

Das Programmkonstrukt besteht aus einem Verarbeitungsteil und einem Steuerungsteil mit einer Bedingung. Die Bedingung bestimmt, ob bzw. wie häufig der Verarbeitungsteil ausgeführt wird, wenn das Element abgearbeitet wird. Möglich ist auch, dass der Verarbeitungsteil überhaupt nicht ausgeführt wird.



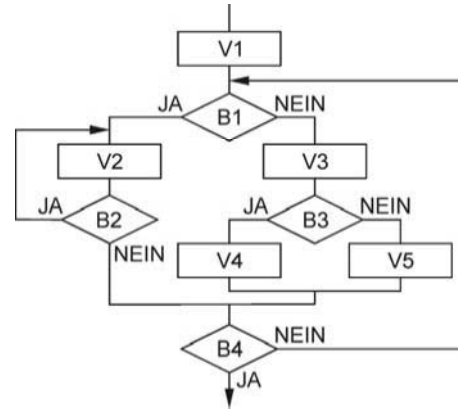
c) Wiederholung mit nachfolgender Bedingungsprüfung

Das Programmkonstrukt besteht aus einem Verarbeitungsteil und einem Steuerungsteil mit einer Bedingung. Mit der Bedingung wird bestimmt, ob bzw. wie häufig der Verarbeitungsteil nach der ersten Ausführung wiederholt wird, wenn das Element abgearbeitet wird.



7.1.5 Kombination der Programmkonstrukte

Die einzelnen Programmkonstrukte können beliebig miteinander kombiniert werden. Der damit mögliche Spaghetticode (die Wege des Ablaufplans gehen wie Spaghetti durcheinander), der zu unübersichtlichen Darstellungen führt, widerspricht den Regeln einer strukturierten Programmierung. Die Einführung von so genannten *Strukturblöcken* zur Bildung von Programmen mit abgeschlossenen Zweigen führt zu einer besseren Übersichtlichkeit und leichteren Verständlichkeit von Programmablaufplänen.



Ein Strukturblock ist dabei eine abgeschlossene funktionale Einheit, welche keine Überlappung mit anderen Strukturblöcken zulässt. Jeder Strukturblock hat einen Eingang und einen Ausgang und besteht aus einem Steuerungsteil und einem oder mehreren Verarbeitungsteilen. In einem Strukturblock kann zur Detaillierung jeder Verarbeitungsteil wieder durch einen Strukturblock ersetzt werden. Der Programmablauf wird durch Schachtelung von Strukturblöcken gebildet.

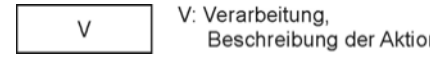
7.2 Struktogramm

Wie Programmablaufpläne haben auch Struktogramme das Ziel, den Algorithmus der Steuerungsaufgabe und den Ablauf der Operationen grafisch anschaulich darzustellen. Die Aussagen eines Struktogramms erfolgen mit Sinnbildern nach Nassi-Shneiderman und erläuternden Texten in den Sinnbildern. Der Steuerungsablauf wird durch die Auswahl der Sinnbilder sowie deren Schachtelung dargestellt. Die Texte beschreiben inhaltlich die Bedingungen und Verarbeitungen. Die Sinnbilder des Struktogramms stellen einen Ersatz für die Programmkonstrukte dar und werden als Strukturblöcke bezeichnet.

Die äußere Form des Sinnbildes ist immer ein Rechteck. Die Unterteilung innerhalb des Rechtecks erfolgt nur durch gerade Linien. Die obere Linie eines jeden Sinnbildes bedeutet den Beginn des Strukturblockes, die untere Linie das Ende. Die Strukturblöcke lassen sich wieder in die Grundstrukturen Verarbeitung, Folge, Auswahl und Wiederholung unterteilen.

7.2.1 Strukturblock Verarbeitung

Die Darstellung des Elementarblocks besteht nur aus einem Rechteck innerhalb dessen die Verarbeitung des Blocks beschrieben ist.



7.2.2 Strukturblock Folge

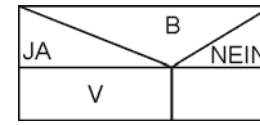
Die Darstellung der Strukturblocks Folge oder Sequenz enthält zwei oder mehrere aneinander gereihte Elementarblöcke

7.2.3 Strukturblock Auswahl

Mit dem Strukturblock Auswahl werden alternative Verarbeitungen beim logischen Ablauf des Programms möglich. Die Bedingung, nach der die Programmauswahl vorgenommen wird, kann entweder ein logischer Ausdruck oder ein Vergleichsausdruck sein. Der gemeinsame Bedingungsteil besteht bei der Mehrfachauswahl aus einer Fallabfrage, bei der einer der möglichen Fälle erfüllt sein muss. Für den Strukturblock Auswahl gibt es drei unterschiedliche Formen.

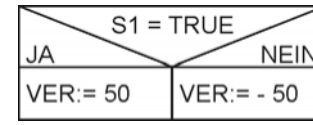
a) Einfachauswahl

Der Strukturblock besteht aus einem Verarbeitungsteil und einem Steuerungsteil mit einer Bedingung. Die Bedingung bestimmt, ob der Vereinbarungsteil ausgeführt wird, wenn der Block abgearbeitet wird.



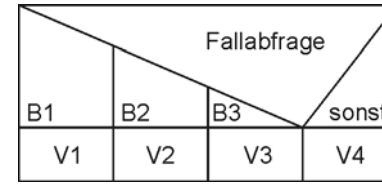
b) Zweifachauswahl

Der Strukturblock besteht aus zwei Verarbeitungsteilen und einem Steuerungsteil. Der Steuerungsteil gibt mit der Bedingung an, welcher der beiden Verarbeitungsteile ausgeführt wird, wenn der Block abgearbeitet wird.



c) Mehrfachauswahl oder Fallunterscheidung

Der Strukturblock besteht aus mindestens drei Verarbeitungsteilen und einem Steuerungsteil mit der gleichen Anzahl einander ausschließender Bedingungen. Der Steuerungsteil gibt mit diesen Bedingungen an, welcher der Verarbeitungsteile ausgeführt wird, wenn der Block abgearbeitet wird.



Ist bei der Fallabfrage gewährleistet, dass einer der aufgezählten Fälle vorkommt, kann die Spalte „sonst“ entfallen.

7.2.4 Strukturblock Wiederholung

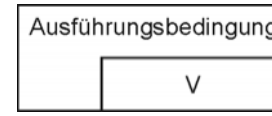
Mit dem Strukturblock Wiederholung, auch Schleife oder Iteration genannt, wird ein Verarbeitungsteil solange wiederholt, wie es der Steuerungsteil vorgibt. Wird eine Wiederholung in Abhängigkeit von einer Bedingungsprüfung ausgeführt, so kann diese vor oder nach dem Verarbeitungsteil stehen.

a) Wiederholung ohne Bedingungsprüfung (Endlosschleife)

Der Strukturblock enthält nur einen Verarbeitungsteil. Dieser wird endlos wiederholt ausgeführt, wenn der Block abgearbeitet wird.

b) Wiederholung mit vorausgehender Bedingungsprüfung

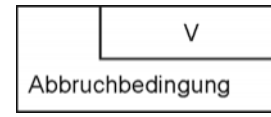
Der Strukturblock besteht aus einem Verarbeitungsteil und einem Steuerungsteil. Mit der Ausführungsbedingung wird bestimmt, ob bzw. wie häufig der Verarbeitungsteil ausgeführt wird, wenn der Block abgearbeitet wird.



Möglich ist auch, dass der Verarbeitungsteil überhaupt nicht ausgeführt wird. Eine Sonderform der Ausführungsbedingung ist die Zählschleife. Bei dieser Schleife wird eine Variable bei jedem Durchlauf des Verarbeitungsteils um einen bestimmten Wert aufwärts oder abwärts gezählt, bis eine bestimmte vorgegebene Grenze erreicht ist.

c) Wiederholung mit nachfolgender Bedingungsprüfung

Der Strukturblock besteht aus einem Verarbeitungsteil und einem Steuerungsteil. Mit der Abbruchbedingung wird bestimmt, ob bzw. wie häufig der Verarbeitungsteil nach der ersten Ausführung wiederholt wird, wenn der Block abgearbeitet wird.

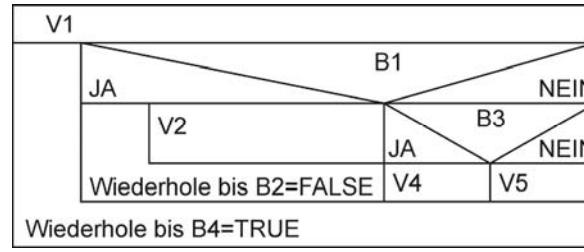


Zusammenfassend sind für den Aufbau eines Strukturblocks folgende Regeln zu beachten:

- Ein Block ist eine abgeschlossene funktionale Einheit mit einer klar definierten Aufgabe, einem Anfangszustand und einem Endzustand.
- Ein Block besitzt immer einen Eingang und einen Ausgang.
- Der Steuerfluss läuft in einem Block immer von oben nach unten.

7.2.5 Kombination der Strukturblöcke

Die einzelnen Strukturblöcke können beliebig miteinander verknüpft werden. Die Verknüpfung erfolgt durch senkrechtes Aneinanderreihen der Blöcke. Beim Aneinanderreihen der Blöcke muss kantendeckend gearbeitet werden, d. h., die Ausgangskante des einen Blocks ist zugleich die Eingangskante des folgenden Blocks.



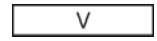
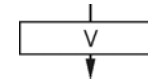
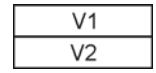
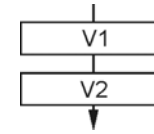
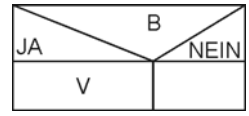
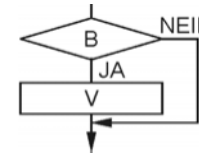
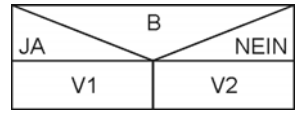
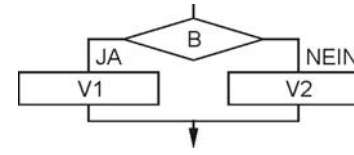
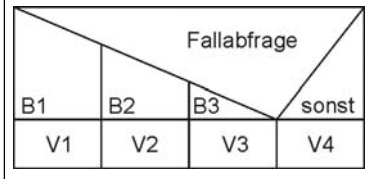
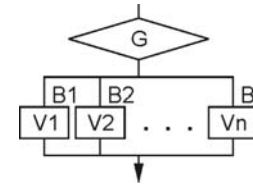
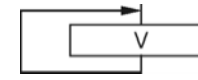
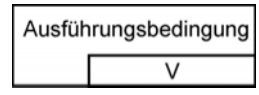
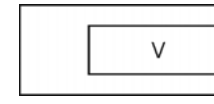
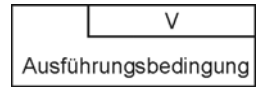
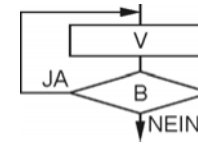
Da Struktogramme stets von oben nach unten gelesen werden, sind Ablauflinien überflüssig. Das gesamte Struktogramm kann als einziger Strukturblock zur Beschreibung der Aufgabe betrachtet werden, der in weitere Strukturblöcke unterteilbar ist. Somit kann zunächst die logische Grobstruktur dargestellt werden, welche dann bis zur Codierfähigkeit in Steuerungsanweisungen schrittweise verfeinert wird. Es entsteht so ein hierarchischer Aufbau, der zur Überschaubarkeit der Programmlogik beiträgt. Die schrittweise Verfeinerung im Sinne einer hierarchischen Gliederung (Top-down) ist deshalb möglich, weil jeder Strukturblock eindeutig mit seinem Anfang und Ende bestimmt ist.

Der hauptsächliche Unterschied zwischen der Darstellung in Struktogrammen und Programmablaufplänen besteht in dem Ziel, Sprünge möglichst zu vermeiden. Unterstützt wird dieses Ziel mit der eindeutigen Festlegung von nur einem Eingang (obere Linie) und einem Ausgang (untere Begrenzungslinie) für den Strukturblock. Durch diese Zweipoligkeit sind Sprünge aus oder in Strukturblöcke nicht mehr darstellbar.

7.3 Zusammenstellung der Sinnbilder für Struktogramm und Programmablaufplan

Die nachfolgende Tabelle zeigt eine Zusammenstellung der Sinnbilder für Struktogramme nach Nassi-Shneidermann mit Gegenüberstellung der Programmkonstrukte des Programmablaufplans.

Tabelle der Sinnbilder für Programmablaufplan und Struktogramm:

Ablaufstruktur	Struktogramm	Programmablaufplan
Verarbeitung		
Folge		
Bedingte Verarbeitung		
Einfache Alternative		
Mehrfache Alternative		
Wiederholung ohne Bedingungsprüfung		
Wiederholung mit vorausgehender Bedingungsprüfung		
Wiederholung mit nachfolgender Bedingungsprüfung		

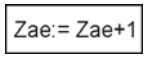
7.4 AWL-Programmierung nach Vorlage von Programmablaufplan oder Struktogramm

Das Ziel der Beschreibungsmittel Programmablaufplan oder Struktogramm ist die von der Programmiersprache unabhängige, strukturierte, übersichtliche Darstellung der Steuerungsaufgabe und des Programmablaufs. Bei der Umsetzung der beiden Beschreibungsmittel in ein Steuerungsprogramm muss der Operationsvorrat des Automatisierungsgerätes und der Programmiersprache berücksichtigt werden. Da die Programmiersprache Anweisungsliste AWL eine maschinennahe Sprache ist, müssen Verzweigungen und Wiederholungen mit Sprungbefehlen zu Marken innerhalb des Steuerungsprogramms umgesetzt werden. Trotz dieses Nachteils bei der Umsetzung eines Struktogramms oder Programmablaufplans in die Programmiersprache AWL bleibt jedoch das Ziel der beiden Beschreibungsmittel erhalten, den Lösungsalgorithmus der Steuerungsaufgabe übersichtlich darzustellen. Die einzelnen Strukturblöcke des Struktogramms sind mit den Programmkonstrukten des Programmablaufplans gleichzusetzen. Die Umsetzung eines Strukturblocks bzw. eines Programmkonstruktes in die Programmiersprache AWL ist deshalb identisch. Im Folgenden ist an Beispielen gezeigt, wie die einzelnen Strukturblöcke bzw. Programmkonstrukte in die Anweisungsliste umgesetzt werden können.

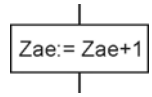
7.4.1 Verarbeitung

Beispiel: Die Variable „Zae“ (Integer) soll um den Wert 1 vergrößert werden.

Struktogramm



Programmablaufplan



STEP 7-AWL

```
L  Zae
+ 1
T  Zae
```

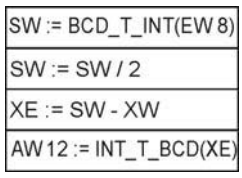
CoDeSys-AWL

```
LD  ZAE
ADD 1
ST  ZAE
```

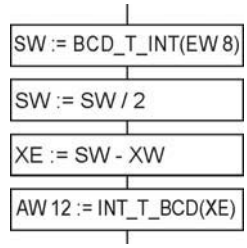
7.4.2 Folge

Beispiel: Der BCD-codierte Wert des Eingangswortes EW 8 soll in eine Integer-Zahl gewandelt und das Ergebnis der Variablen „SW“ zugewiesen werden. Die Variable „SW“ wird dann durch zwei dividiert. Von diesem Wert wird die Variable „XW“ abgezogen und das Ergebnis der Variablen „XE“ zugewiesen. Der Wert von „XE“ soll dann in eine BCD-Zahl gewandelt und an das Ausgangswort AW 12 gelegt werden. *Hinweis:* Die Umwandlung von BCD nach INTEGER ist wegen der durchzuführenden Rechenoperation erforderlich.

Struktogramm



Programmablaufplan



STEP 7-AWL

```
CALL  FC19
 BCDW:=EW8
 SBCD:=#HO1
 INTW:=#SW
L  #SW
L  2
/I
L  #XW
-I
T  #XE
CALL  FC41
 INTW:=#XE
 BCDW:=AW12
 SBCD:=#HO2
```

CoDeSys-AWL

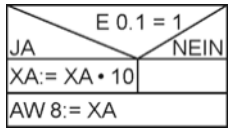
```
LD %IW8
FC19 HO1
ST SW

LD SW
DIV 2
SUB XW
ST  XE

LD  XE
FC20
ST FC20_OUT
LD FC20_OUT.BCDW
ST %QW12
```

7.4.3 Auswahl

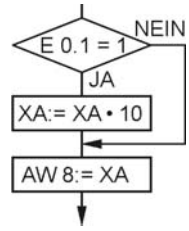
a) Einfachauswahl

Beispiel: Bei „1“-Signal am Eingang E 0.1 wird der Wert der Variablen „XA“ mit 10 multipliziert und der Variablen „XA“ wieder zugewiesen. Der Wert der Variablen „XA“ wird dann dem Ausgang AW 8 zugewiesen.

Struktogramm



Programmablaufplan



STEP 7-AWL

```
      UN E 0.1
      SPB M001
      L  #XA
      L  10
      *I
      T  #XA
M001:L  #XA
      T AW 8
```

CoDeSys-AWL

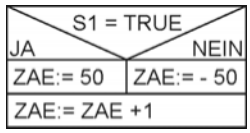
```
LDN  %IX0.1
JMPC M001
LD   XA
MUL  10
ST   XA
M001:
LD  XA
ST  %QW8
```

Hinweis: Bei der Umsetzung einer Einfachauswahl in die Anweisungsliste AWL wird ein Sprungbefehl gespart, wenn die Bedingung oder Abfrage negiert wird, wie es oben dargestellt ist. Bei erfüllter Bedingung wird dann der Verarbeitungsteil der Einfachauswahl einfach übersprungen und das Programm kann nach der Sprungmarke fortgesetzt werden.

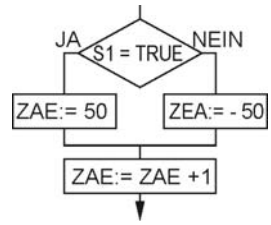
b) Zweifachauswahl

Beispiel: Hat die Variable „S1“ den binären Wert „TRUE“, wird der Variablen „ZAE“ 50 zugewiesen. Ist der binäre Wert „FALSE“ ist die Zuweisung –50. Danach wird zur Variablen ZAE der Wert +1 addiert.

Struktogramm



Programmablaufplan



STEP 7-AWL

```
      U  #S1
      SPB M001
      L  -50
      T  #ZAE
      SPA M002
M001:NOP 0
      L  50
      T  #ZAE
M002:NOP 0
      L  #ZAE
      +1
      T  #ZAE
```

CoDeSys-AWL

```
LD  S1
JMPC M001
LD  -50
ST  ZAE
JMP M002
M001:
LD  50
ST  ZAE
M002:
LD  ZAE
ADD 1
ST  ZAE
```

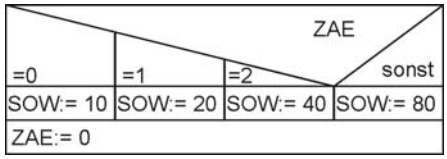
Hinweis: Bei der Umsetzung einer Zweifachauswahl in die Anweisungsliste AWL muss mit einem absoluten Sprung über den Verarbeitungsteil des „JA-Zweiges“ gesprungen werden. Steht die Abfrage am Ende eines Programms kann der absolute Sprung durch die Baustein-Ende-Anweisung BEA ersetzt werden.

c) Mehrfachauswahl

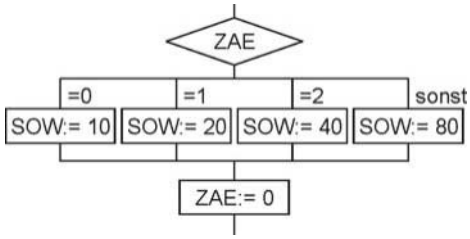
Die Umsetzung der Mehrfachauswahl kann mit mehreren hintereinander ausgeführten Zweifachauswahl-Abfragen ausgeführt werden.

Beispiel: Bei Zählerstand ZAE = 0 wird der Variablen SOW der Wert 10, bei ZAE = 1 der Wert 20, bei ZAE = 2 der Wert 40 und bei allen anderen Zählerständen der Wert 80 zugewiesen. Danach wird die Zählervariable ZAE auf den Wert 0 gesetzt.

Struktogramm:



Programmablaufplan:



STEP 7-AWL

```
      L  #ZAE
      L 0
      ==I
      SPB M001
      L  #ZAE
      L  1
      ==I
      SPB M002
      L  #ZAE
      L  2
      ==I
      SPB M003
      L  80
      T  #SOW
      SPA M004
M001:L  10
      T  #SOW
      SPA M004
M002:L  20
      T  #SOW
      SPA M004
M003:L  40
      T  #SOW
M004:L 0
      T  #ZAE
```

CoDeSys-AWL

```
LD  ZAE
EQ  0
JMPC M001
LD  ZAE
EQ  1
JMPC M002
LD  ZAE
EQ  2
JMPC M003
LD  80
ST  SOW
JMP  M004
M001:
LD  10
ST  SOW
JMP  M004
M002:
LD  20
ST  SOW
JMP  M004
M003:
LD  40
ST  SOW
M004:
LD  0
ST  ZAE
```

Umsetzung der Mehrfachauswahl mit der Sprungleiste SPL bei STEP 7

STEP 7 bietet mit der Operation Sprungleiste SPL ein Befehl zur Programmierung von Fallunterscheidungen an. Die Zielsprungleiste, die maximal 255 Einträge enthalten kann, beginnt unmittelbar nach der Operation SPL und endet vor der Sprungmarke, die der Operand SPL angibt. Jedes Sprungziel besteht aus einer Operation SPA. Solange der AKKU-Inhalt kleiner ist als die Anzahl der Sprungziele zwischen SPL-Anweisung und Sprungmarke, springt die Operation SPL auf eine der Operationen SPA.

Umsetzung des Beispiels der Mehrfachauswahl mit der Operation Sprungleiste SPL:

Anweisungsliste

```
      L   #ZAE
      SPL M003
      SPA M000
      SPA M001
      SPA M002
M003: L   80
      T   #SOW
      SPA M004
M000: L   10
      T   #SOW
      SPA M004
M001: L   20
      T   #SOW
      SPA M004
M002: L   40
      T   #SOW
M004: NOP 0
```

Hinweise:

Die Verwendung der Sprungleistenanweisung SPL setzt voraus, dass die Auswahlwerte Integer-Zahlen von 0 bis n sind.

Die Zielsprungleiste muss aus den Operationen „Springe-Absolut“ SPA bestehen, die sich vor der Sprungmarke befinden, die vom Operanden der Anweisung SPL angegeben wird. Andere Operationen innerhalb der Sprungleiste sind unzulässig.

ZAE:	Zuweisung:
0	SOW:= 10
1	SOW:= 20
2	SOW:= 40
sonst	SOW:= 80

7.4.4 Wiederholung

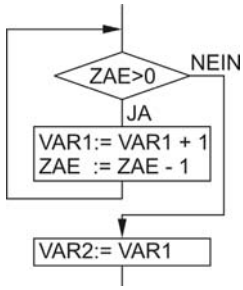
a) Wiederholung ohne Bedingungsprüfung

Mit diesem Strukturblock- bzw. Programmkonstrukt-Typ könnte man das zyklisch abzuarbeitende gesamte Steuerungsprogramm einer SPS beschreiben.

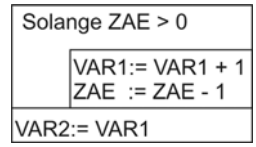
b) Wiederholung mit vorausgehender Bedingungsprüfung

Beispiel: Zur Variablen VAR1 (INT) wird sooft +1 dazu addiert, wie der Wert in der Variablen ZAE (INT) angibt. Danach wird die Variable VAR1 der Variablen VAR2 zugewiesen.

Programmablaufplan



Struktogramm



STEP 7-AWL

```
M001: L #ZAE
      L 0
      >I
      SPBN M002

      L #VAR1
      + 1
      T #VAR1

      L #ZAE
      + -1
      T #ZAE
      SPA M001
M002: L VAR1
      T VAR2
```

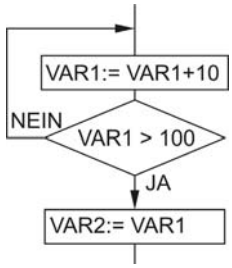
CoDeSys-AWL

```
M001:
      LD  ZAE
      GT  0
      NOT
      JMPC  M002

      LD  VAR1
      ADD 1
      ST  VAR1

      JMP M001

M002:
      LD  VAR1
      ST  VAR2
```

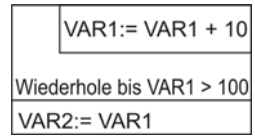
c) Wiederholung mit nachfolgender Bedingungsprüfung

Beispiel: Zur Variablen VAR1 (INT) wird solange der Wert 10 dazu addiert, bis die Variable VAR1 größer als 100 ist. Danach wird die Variable VAR1 der Variablen VAR2 zugewiesen.

Programmablaufplan



Struktogramm



STEP 7-AWL

```
M001: L VAR1
      + 10
      T VAR1
      L 100
      >I
      SPBN M001

      L #VAR1
      T #VAR2
```

CoDeSys-AWL

```
M001:
      LD  VAR1
      ADD 10
      ST  VAR1

      LD  VAR1
      GT  100
      NOT
      JMPC  M001

      LD  VAR1
      ST  VAR2
```

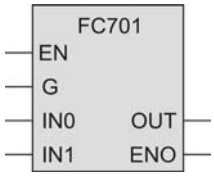
Bei STEP 7 besteht für die Wiederholung mit nachfolgender Bedingungsprüfung die Möglichkeit, die LOOP-Anweisung anzuwenden. Voraussetzung dafür ist jedoch, dass als Abbruchbedingung der Wert 0 einer Zählvariablen ZAE (INT) vorliegt. Bei jedem Schleifendurchlauf wird dann die Zählvariable um 1 bis zum Wert 0 vermindert. Es entfällt dabei die Subtraktion der Variablen ZAE und die Vergleichsabfrage, ob die Variable ZAE den Wert 0 hat.

7.5 Beispiele

■ **Beispiel 7.1: Bedingte Variablenauswahl (FC 701: SEL)**

Es ist eine Funktion FC 701 für die eigene Programmbibliothek zu entwerfen, die abhängig von einem binären Funktionseingang G einen der beiden Eingangs-Variablenwerte (INO bzw. IN1) an den Ausgang OUT legt. Die Eingangsvariablen IN0 und IN1 der Funktion haben den Datentyp WORD. Der Ausgabewert hat dann ebenfalls das Datenformat WORD.

Übergabeparameter:



Beschreibung der Parameter:

G: Datenformat BOOL;
G = FALSE: OUT := IN0
G = TRUE: OUT := IN1

IN0, IN1 und OUT: Datenformat WORD

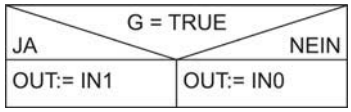
Zum Test der Funktion 701 wird an den Eingang IN0 das Eingangswort eines Zifferneinstellers EW und an den Eingang IN1 der Wert 16#1111 gelegt. Mit dem Schalter S1 wird dann bestimmt, welcher der beiden Werte an der Ziffernanzeige AW erscheint.

Zuordnungstabelle der Eingänge und Ausgänge:

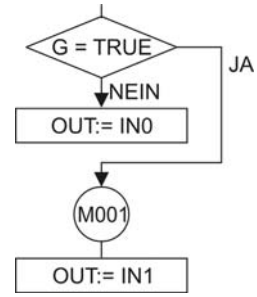
Eingangsvariable	Symbol	Datentyp	Logische Zuordnung	Adresse
Wahlschalter	S1	BOOL	Betätigt S1 = 1	E 0.1
Zifferneinsteller	EW	WORD	BCD-Code	EW 8
Ausgangsvariable				
Ziffernanzeige	AW	WORD	BCD-Code	AW 12

Nachfolgend sind das Struktogramm und der Programmablaufplan der Funktion FC 701 angegeben. Der Programmablaufplan ist dabei so gezeichnet, dass die Reihenfolge der Anweisungen und die Sprungmarke des AWL-Programms erkennbar sind.

Struktogramm:



Programmablaufplan:



STEP 7 Programm: AWL-Quelle

```
FUNCTION FC701: VOID   VAR_OUTPUT          BEGIN           M001:NOP 0;
VAR_INPUT               OUT : WORD ;       U  #G;               L  #IN1;
 G  : BOOL ;           END_VAR             SPB M001;            T  #OUT;
 IN0: WORD ;                               L  #IN0;
 IN1: WORD ;                               T  #OUT;        END_FUNCTION
END_VAR                                    BEA ;
```

CoDeSys Programm:

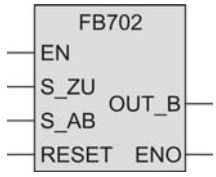
Obwohl es in CoDeSys die Select-Operation SEL gibt, welche die Aufgabe der Funktion FC 701 erfüllt, ist nachfolgend zu Übungszwecken die CoDeSys-AWL der Funktion FC 701 dargestellt.

```
FUNCTION FC701 : WORD    LD  G                    M001:
VAR_INPUT                JMPC  M001               LD  IN1
   G: BOOL;                                       ST  FC701
   IN0: WORD;            LD IN0
   IN1: WORD;            ST FC701
END_VAR                  RET
```

■ Beispiel 7.2: Stufenschalter (FB 702: STUFE)

Es ist ein Funktionsbaustein FB 702 zu entwerfen, der das stufenweise Zu- und Abschalten von bis zu acht Kompensations- bez. Leistungsstufen ermöglicht. Der Funktionsbaustein kann beispielsweise bei einer Blindstromkompensationsanlage verwendet werden, um bei entsprechenden Bedingungen jeweils Kompensationsgruppen hinzu- oder abzuschalten. Ein 0→1 Signalwechsel am Eingang S_ZU führt zu der Zuschaltung einer Kompensations- bzw. Leistungsstufe am Ausgang OUT_B des Funktionsbausteins. Ein 0→1 Signalwechsel am Eingang S_AB führt dagegen zum Abschalten einer Stufe. Mit einem „1"-Signal am Eingang RESET wird der Funktionsbausteinausgang OUT_B auf 0 zurückgesetzt.

Übergabeparameter:



Beschreibung der Parameter:

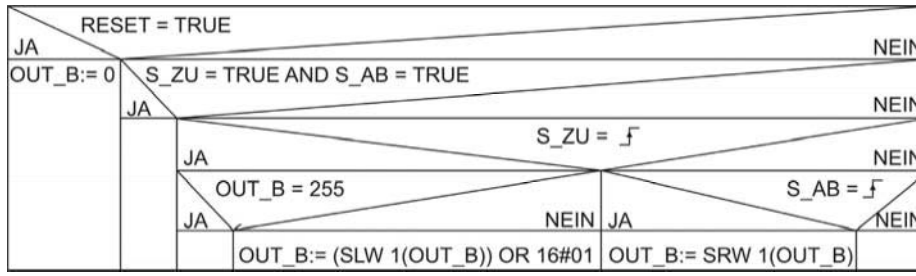
S_ZU (BOOL):	Zuschalten einer Leistungsstufe
S_AB (BOOL):	Abschalten einer Leistungsstufe
RESET (BOOL):	Löschen aller Leistungsstufen
OUT_B (BYTE):	Ausgangsbyte des Funktionsbausteins, dessen Bits mit einem „1"-Signal die Anzahl der eingeschalteten Leistungsstufen angeben.

Zum Test des Funktionsbausteins FB 702 wird an die Eingänge S_AUF, S_AB bzw. RESET jeweils ein Taster S1, S2 bzw. S3 gelegt. Das Ausgangsbyte XA_B des Funktionsbausteins wird einem Ausgangsbyte AB der SPS zugewiesen.

Zuordnungstabelle der Eingänge und Ausgänge:

Eingangsvariable	Symbol	Datentyp	Logische Zuordnung		Adresse
Taster Leistungsstufe zuschalten	S1	BOOL	Betätigt	S1 = 1	E 0.1
Taster Leistungsstufe abschalten	S2	BOOL	Betätigt	S2 = 1	E 0.2
Taster RESET	S3	BOOL	Betätigt	S3 = 1	E 0.3
Ausgangsvariable					
Leistungsstufen	AB	BYTE	Bitkombination		AB 4

Das nachfolgende Struktogramm zeigt den Programmaufbau des Funktionsbausteins FB 702.

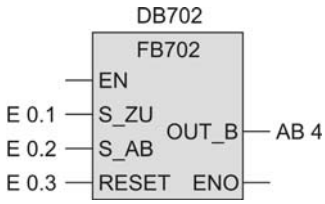


Liegt eine 0→1 Flanke am Eingang S_ZU vor, wird ein „1"-Signalwert von rechts in das Ausgangsbyte OUT_B geschoben. Bei einer 0→1 Flanke am Eingang S_AB wird das Ausgangsbyte OUT_B um eine

Stelle nach rechts geschoben. Ist die RESET-Taste betätigt, werden alle Bits des Ausgangsbytes zurückgesetzt. Für die Flankenauswertung der beiden Eingänge S_ZU und S_AB müssen die beiden Flankenoperanden FO1 bzw. FO2 als statische Lokalvariablen eingeführt werden.

STEP 7 Programm:

Aufruf FB 702 im OB 1:



FB 702 AWL-Quelle:

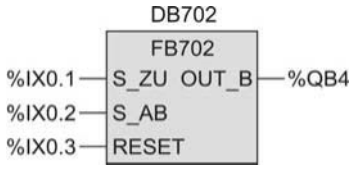
```
FUNCTION_BLOCK
FB702

VAR_INPUT
 S_ZU : BOOL ;
 S_AB : BOOL ;
 RESET : BOOL ;
END_VAR
VAR_OUTPUT
 OUT_B : BYTE ;
END_VAR

VAR
 FO1 : BOOL ;
 FO2 : BOOL ;
END_VAR
```

```
BEGIN
U  #RESET;
SPBN M001;
L  0;
T  #OUT_B;
BEA ;

M001: NOP 0;
U  #S_ZU;
U  #S_AB;
BEB ;

U  #S_ZU;
FP #FO1;
SPBN M002;

L  #OUT_B;
L  255;
==I ;
BEB ;
```

```
L  #OUT_B;
SLW 1;
L  B#16#1;
OW ;
T  #OUT_B;
BEA ;

M002: NOP 0;
U     #S_AB;
FP    #FO2;
NOT   ;
BEB   ;
L  #OUT_B;
SRW 1;
T  #OUT_B;

END_FUNCTION_
BLOCK
```

CoDeSys Programm:

Aufruf FB 702 im PLC_PRG:



FB 702 AWL:

```
FUNCTION_BLOCK
FB702
VAR_INPUT
 S_ZU: BOOL;
 S_AB: BOOL;
 RESET: BOOL;
END_VAR
VAR_OUTPUT
 OUT_B: BYTE;
END_VAR
VAR
 FO1: R_TRIG;
 FO2: R_TRIG;
END_VAR
```

```
LDN RESET
JMPC M001
LD  0
ST  OUT_B
RET

M001:
LD  S_ZU
AND S_AB
RETC

CAL
FO1(CLK:=S_ZU)
LD  FO1.Q
NOT
JMPC M002
LD  OUT_B
EQ  255
RETC
```

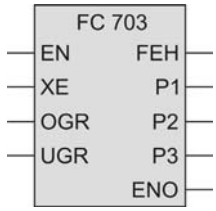
```
LD  OUT_B
SHL 1
OR 1
ST  OUT_B
RET

M002:
CAL
FO2(CLK:=S_AB)
LD   FO2.Q
NOT
RETC

LD  OUT_B
SHR 1
ST  OUT_B
```

■ **Beispiel 7.3: Vergleicher mit Dreipunktverhalten (FC 703: COMP_3)**

Mit drei Meldeleuchten soll angezeigt werden, ob ein Wert am Eingang XE (REAL) in einem unteren, mittleren oder oberen Bereich liegt. Die drei Bereiche werden durch eine untere Grenze UGR und eine obere Grenze OGR bestimmt. Eine Anzeigeleuchte FEH zeigt an, wenn versehentlich die untere Grenze größer oder gleich als die obere Grenze vorgegeben wurde. Alle anderen Anzeigen sind in diesem Fall ausgeschaltet.

Der Vergleicher soll mit der Funktion FC 703 realisiert werden.

Übergabeparameter:



Beschreibung der Parameter:

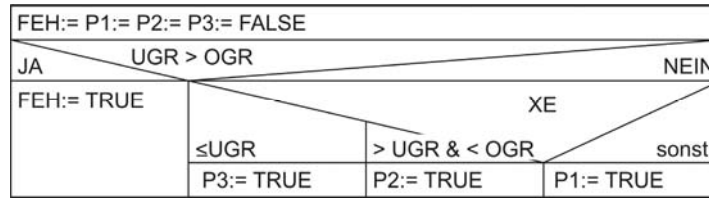
XE (REAL):	zu prüfender Wert
OGR (REAL):	obere Grenze
UGR (REAL):	untere Grenze
FEH (BOOL):	Anzeige UGR > OGR
P1 (BOOL):	Anzeige XE ≥ OGR
P2 (BOOL):	Anzeige UGR < XE < OGR
P3 (BOOL):	Anzeige XE ≤ UGR

Zum Test der Funktion FC 703 wird über ein Merkerdoppelwort MD 20 eine Gleitpunktzahl an den Eingangsparameter XE der Funktion gelegt. Die obere und untere Grenze des zu bestimmenden Bereichs werden durch die Merkerdoppelwörter MD 40 (OGR) und MD 30 (UGR) bestimmt. An die Ausgänge der Funktion FC 703 (FEH, P1, P2 und P3) werden Anzeigeleuchten gelegt.

Zuordnungstabelle der Merker und Ausgänge für den Test der Funktion:

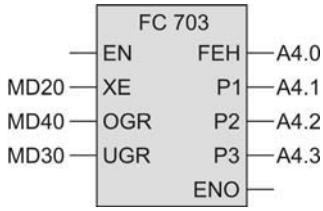
Merkervariable	Symbol	Datentyp	Logische Zuordnung		Adresse
Zu überprüfender Wert	XIN	REAL	Gleitpunktzahl		MD 20
Bereichsobergrenze	OGR	REAL	Gleitpunktzahl		MD 40
Bereichsuntergrenze	UGR	REAL	Gleitpunktzahl		MD 30
Ausgangsvariable					
Anzeige Fehler	FEH	BOOL	Anzeige leuchtet	FEH = 1	A 4.0
Anzeige XIN ≥ OGR	P1	BOOL	Anzeige leuchtet	P1 = 1	A 4.1
Anzeige UGR < XIN < OGR	P2	BOOL	Anzeige leuchtet	P2 = 1	A 4.2
Anzeige XIN ≤ UGR	P3	BOOL	Anzeige leuchtet	P3 = 1	A 4.3

Struktogramm der Funktion FC 703:



STEP 7 Programm:

Aufruf FC 703 im OB 1:



FC 703 AWL-Quelle:

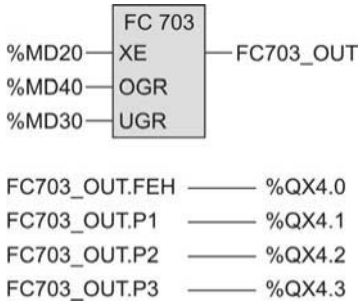
```
FUNCTION
FC703:VOID

VAR_INPUT
  XE : REAL ;
  OGR : REAL ;
  UGR : REAL ;
END_VAR
VAR_OUTPUT
  FEH : BOOL ;
  P1 : BOOL ;
  P2 : BOOL ;
  P3 : BOOL ;
END_VAR
```

```
BEGIN
CLR ;
=  #FEH;
=  #P1;
=  #P2;
=  #P3;

L  #UGR;
L  #OGR;
>=R ;
=  #FEH;
BEB ;
```

```
L  #XE;
L  #UGR;
<=R ;
=  #P3;
BEB ;

L  #XE;
L  #OGR;
<R ;
=  #P2;
BEB ;

SET ;
=  #P1;
END_FUNCTION
```

CoDeSys Programm:

Aufruf FC 703 im PLC_PRG:



FC 703 AWL:

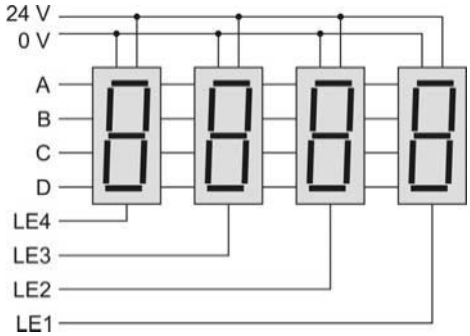
```
TYPE FC_OUT:
STRUCT
 FEH:BOOL;
 P1:BOOL;
 P2:BOOL;
 P3:BOOL;
END_STRUCT
END_TYPE
```

```
FUNCTION
FC703:FC_OUT
VAR_INPUT
 XE: REAL;
 OGR: REAL;
 UGR: REAL;
END_VAR

LD  FALSE
ST  FC703.FEH
ST  FC703.P1
ST  FC703.P2
ST  FC703.P3
```

```
LD  UGR
GE  OGR
ST  FC703.FEH
RETC
LD  XE
LE  UGR
ST  FC703.P3
RETC
LD  XE
LT  OGR
ST  FC703.P2
RETC
LD  TRUE
ST  FC703.P1
```

■ Beispiel 7.4: Multiplex-Ziffernanzeige (FB 704: BCD_MPA)

Ein vierstelliger BCD-Wert soll an einer Multiplex-Ziffernanzeige ausgegeben werden.



Eine Multiplex-Ziffernanzeige ist eine Anzeige mit speicherndem Verhalten. Die Dateneingänge A, B, C und D der vier Dekaden werden parallel geschaltet. Über jeweils einen eigenen Eingang LE (Latch Enable) wird eine Stelle der Ziffernanzeige dazu veranlasst, den auf den Datenleitungen augenblicklich anstehenden Wert in den Speicher und damit in die Anzeige zu übernehmen. Damit ist es möglich, die Anzahl der Leitungen zur Anzeigeeinheit und somit auch die belegten Ausgänge der SPS merklich zu reduzieren.

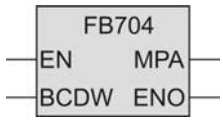
Bild 7.1: Multiplexanzeige

Anzahl der erforderlichen Leitungen L:

Muliplex-Ziffernanzeige:	BCD-Ziffernanzeige ohne Speicher:
L = 4 + n; (n = Anzahl der Dekaden)	L = 4 · n; (n = Anzahl der Dekaden)
Für n = 4 : Anzahl der Leitungen L = 8	Für n = 4 : Anzahl der Leitungen L = 16

Für die Ansteuerung des Ausgangsbytes der vierstelligen Multiplexanzeige ist ein Funktionsbaustein FB 704 für die eigene Programmbibliothek zu entwerfen.

Übergabeparameter:



Beschreibung der Parameter:

BCDW (WORD): BCD-Wert, der angezeigt werden soll

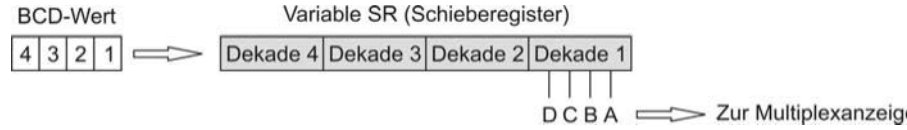
MPA (BYTE): Ausgabebyte für die Multiplex-Ziffernanzeige

Zum Test des Funktionsbausteins wird an den Eingang BCDW der Wert eines vierstelligen Ziffern-einstellers gelegt. An den Byte-Ausgang MPA (A ... D, LE1 ... LE2) wird das Ausgangsbyte gelegt, an das eine Multiplex-Ziffernanzeige angeschlossen ist.

Zuordnungstabelle der Eingänge und Ausgänge:

Eingangsvariable	Symbol	Datentyp	Logische Zuordnung		Adresse
Zifferneinsteller	EW	WORD	BCD-Code		EW 8
Ausgangsvariable					
Wert 1 der Ziffer	A	BOOL	Wert 2^0		A 4.0
Wert 2 der Ziffer	B	BOOL	Wert 2^1		A 4.1
Wert 4 der Ziffer	C	BOOL	Wert 2^2		A 4.2
Wert 8 der Ziffer	D	BOOL	Wert 2^3		A 4.3
Freigabe der 4. Ziffer	LE1	BOOL	Daten einlesen	LE4 = 0	A 4.4
Freigabe der 3. Ziffer	LE2	BOOL	Daten einlesen	LE3 = 0	A 4.5
Freigabe der 2. Ziffer	LE3	BOOL	Daten einlesen	LE2 = 0	A 4.6
Freigabe der 1. Ziffer	LE4	BOOL	Daten einlesen	LE1 = 0	A 4.7
Zusammenfassung der Ausgangsvariablen					
Anschlüsse Multiplexanzeige	AB	BYTE			AB 4

Damit jede Dekade zur richtigen Zeit die Daten übernehmen und speichern kann, ist eine zeitlich versetzte Ansteuerung der LE-Eingänge in Verbindung mit den dazugehörigen Daten erforderlich. Dazu wird der auszugebende BCD-Wert einer Variablen SR zugewiesen, deren vier rechte Bits die erste Ziffer angeben. Diese vier Bits werden auf die Datenleitungen A,B,C und D gelegt und der entsprechende LE-Eingang aktiviert. Ist die erste Ziffer in der Anzeige gespeichert, wird das Bitmuster der Variablen SR um vier Stellen nach links geschoben. Damit liegt die nächste Ziffer an den Datenleitungen A,B,C und D. Der Vorgang wiederholt sich dann, bis alle 4 Ziffern ausgegeben sind.



Ein Zeitablauf für die Ausgabe der vier Dekaden ist im nachfolgenden Zeitdiagramm dargestellt. Dekade 1 = 1 (Takt 1 bis Takt 3) bedeutet dabei, dass auf die Datenleitungen A, B, C und D die BCD-Werte der ersten Dekade gelegt werden. Entsprechendes gilt für Dekade 2 = 1 (Takt 4 bis Takt 6) usw.

Zeitablaufdiagramm für die Ansteuerung des Funktionsbaustein-Ausgangs MPA:

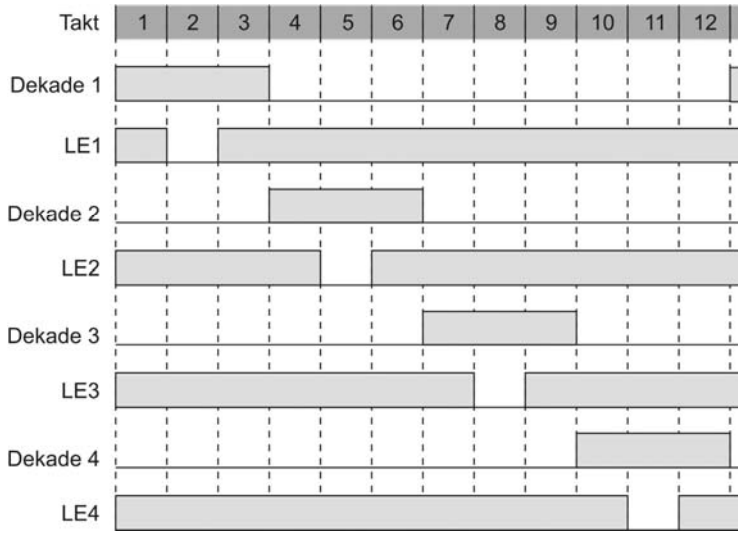
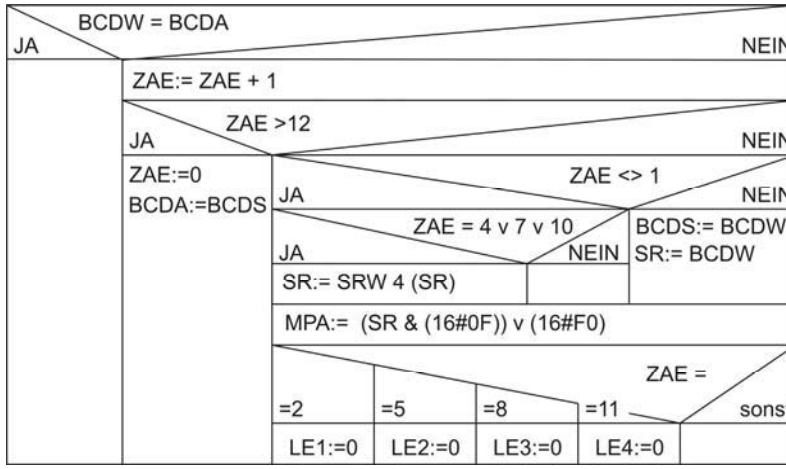


Bild 7.2:
Zeitablaufdiagramm

Aus dem Zeitdiagramm ist zu entnehmen, dass nach zwölf Takten alle vier Ziffern in die Anzeige geschrieben sind. Ein Takt entspricht im folgenden Struktogramm einem Durchlauf. Verbal können die erforderlichen Operationen innerhalb der zwölf Takte wie folgt beschrieben werden:

Takt	Operation
1:	Die auszugebende vierstellige BCD-Zahl wird in ein Schieberegister (SR) und in einen Speicher (BCDS) geladen. Die letzten vier Bits des Schieberegisters (SR) werden mit den Datenleitungen A, B, C und D verbunden.
2:	Der Freigabe-Ausgang LE1 wird auf „0“-Signal gelegt. Damit wird die auf den Datenleitungen liegende erste Ziffer in die erste Dekade der Anzeige übernommen.
3:	Der Freigabe-Ausgang LE1 wird wieder auf „1“-Signal gelegt. Auf der Datenleitung liegt weiterhin der Wert der ersten Ziffer. Mit der „0 – 1“-Flanke auf LE1 wird die Ziffer in der ersten Dekade innerhalb der Anzeige gespeichert.
4, 7 u. 10:	Der Inhalt des Schieberegisters (SR) wird um vier Bits nach rechts geschoben. Auf den Datenleitungen A, B, C und D liegt dann die Ziffer für die jeweils nächste Dekade.
5, 8 u. 11:	Der Freigabe-Ausgang für die jeweils nächste Dekade wird auf „0“-Signal gelegt. Damit erfolgt die Übernahme der jeweiligen Ziffer in die Anzeige.
6, 9 u. 12:	Alle Freigabe-Ausgänge werden wieder auf „1“-Signal gelegt. Damit wird der auf der Datenleitung liegende Wert an der entsprechenden Stelle in der Anzeige gespeichert.
13:	Der 13. Takt, der nach dem Zeitdiagramm nicht aufgeführt ist, dauert ein Programmzyklus und wird genutzt, um den Taktzähler ZAE zurückzusetzen und um den im Takt 1 gespeicherten Wert der Variablen BCDS an die Variable BCDA zu übergeben. Der Wert, der in der Variablen BCDA jeweils gespeichert ist, wurde von dem Funktionsbaustein vollständig an die Multiplexanzeige ausgegeben. Hat der Eingang BCDW des Funktionsbausteins den gleichen Wert wie die Variable BCDA, ist keine neuerliche Ausgabe erforderlich.

Darstellung des Steuerungsprogramms für den Funktionsbaustein FB 704 im Struktogramm:



Hinweise: Die im Struktogramm zusätzlich aufgeführten Variablen BCDA, BCDS und SR sind im Funktionsbaustein bei STEP 7 als statische Lokaldaten zu deklarieren. Mit der Anweisung MPA:= (SR & (W#16#0F) v (16#F0) werden die letzten vier Bits des Schiebespeichers dem Ausgangsbyte MPA zugewiesen und alle Bits der Freigabe-Ausgänge LE auf „1“-Signal gesetzt.

Der Aufruf des Funktionsbausteins FB 704 muss zeitgesteuert erfolgen, damit der begrenzten Reaktionszeit einer Ausgabebaugruppe mit einer Schaltfrequenz von 100 Hz Rechnung getragen wird. Dazu wird

an den Eingang EN des Funktionsbausteins FB 704 die Flankenauswertung eines Taktgenerators mit der Periodendauer von 20 ms gelegt. Die Übernahme eines vollständigen BCD-Wertes in die gemultiplexte Anzeige dauert dann 12 x 20 ms = 240 ms.

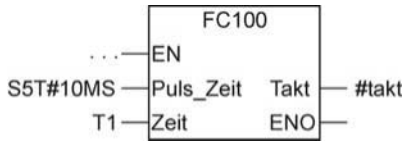
Lösung für STEP 7

STEP 7 Programm (AWL-Quelle) des Funktionsbausteins FB 704:

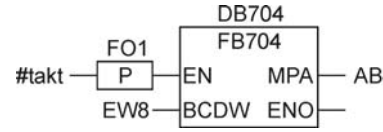
```
FUNCTION_BLOCK FB704
VAR_INPUT                VAR_OUTPUT              VAR
  BCDW : WORD ;            MPA : BYTE ;           BCDA : WORD ;
END_VAR                  END_VAR                  BCDS : WORD ;
                                                  SR : WORD ;
                                                  ZAE : INT ;
                                                 END_VAR
BEGIN
L  #BCDW;          L  #BCDW;          M003: NOP 0;       T  #MPA;
L  #BCDA;          T  #BCDS;          L  #SR;            BEA;
==I ;              T  #SR;            L  W#16#F;         M005: NOP 0;
BEB ;              SPA M003;          UW ;               L  #ZAE;
L  #ZAE;           M002: NOP 0;       L  W#16#F0;        L  8;
+  1;              O( ;               OW ;               <>I;
T  #ZAE;           L  #ZAE;           T  #MPA;           SPB M006;
                   L  4;              L  #ZAE;           L  #MPA;
L  12;             ==I;               L  2;              L  B#16#DF;
<=I;               )  ;               <>I;               UW ;
SPB M001;          O( ;               SPB M004;          T  #MPA;
                   L  #ZAE;           L  #MPA;           BEA;
L     0;           L  7;              L  B#16#7F;        M006: NOP 0;
T  #ZAE;           ==I;               UW ;               L  #ZAE;
                   )  ;               T  #MPA;           L  11;
L  #BCDS;          O( ;               BEA;               <>I;
T  #BCDA;          L  #ZAE;           M004: NOP 0;       BEB;
BEA;               L  10;             L  #ZAE;           L  #MPA;
                   ==I;               L  5;              L  B#16#EF;
M001: NOP 0;       )  ;               <>I;               UW ;
L  #ZAE;           SPBN M003;         SPB M005;          T  #MPA;
L  1;              L  #SR;            L  #MPA;
<>I;               SRW4;              L  B#16#BF;
SPB M002;          T  #SR;            UW ;               END_FUNCTION_BLOCK
```

Aufruf (FUP) der Bausteine im OB 1:

Netzwerk 1



Netzwerk 2



Lösung für CoDeSys

CoDeSys AWL des Funktionsbausteins FB 704:

```
FUNCTION_BLOCK FB704
VAR_INPUT                VAR_OUTPUT              VAR
 BCDW: WORD;              MPA: BYTE;             BCDA: WORD;
END_VAR                  END_VAR                 BCDS: WORD;
                                                 SR: WORD;
                                                 ZAE: INT;
                                                END_VAR
```

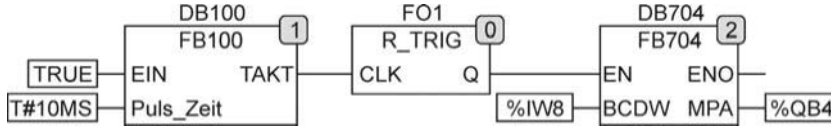
```
LD  BCDW
EQ  BCDA
RETC
LD  ZAE
ADD 1
ST  ZAE
LD  ZAE
LE  12
JMPC M001
LD  0
ST  ZAE
LD  BCDS
ST  BCDA
RET
M001:
LD  ZAE
NE  1
JMPC M002
```

```
LD  BCDW
ST  BCDS
ST  SR
JMP M003
M002:
LD  ZAE
EQ  4
OR (ZAE
EQ  7
)
OR (ZAE
EQ  10
)
NOT
JMPC M003
LD  SR
SHR 4
ST  SR
```

```
M003:
LD  SR
AND 15
OR  240
WORD_TO_BYTE
ST  MPA
LD  ZAE
NE  2
JMPC M004
LD  MPA
AND 127
ST  MPA
RET
M004:
LD  ZAE
NE  5
JMPC M005
LD  MPA
```

```
AND  91
ST  MPA
RET
M005:
LD  ZAE
NE  8
JMPC M006
LD  MPA
AND 223
ST  MPA
RET
M006:
LD  ZAE
NE  11
RETC
LD  MPA
AND 239
ST  MPA
```

Aufruf (CFC) der Bausteine im PLC-Programm:



■ **Beispiel 7.5: Umwandlung BCD_TO_REAL für 4 Ziffern (FC 705: BCD4_REAL)**

Für die eigene Programmbibliothek ist eine Funktion FC 705 (BCD4_REAL) zu entwerfen, die einen BCD-Wert von –9 999 bis + 9 999 in eine Gleitpunktzahl wandelt. Das Vorzeichen des BCD Wertes wird mit einem Schalter S als boolesche Variable am Eingang SBCD der Funktion FC 21 vorgegeben.

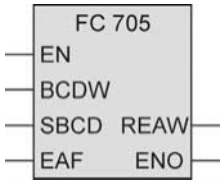
Der Gleitpunktzahlenbereich, in den der BCD-Wert eines vier stelligen Zifferneinstellers aufgelöst werden soll, wird durch die Angabe eines Ausgabefaktors als Zehnerpotenz an einem weiteren Funktionseingang vorgegeben. Der Wert des Zifferneinstellers multipliziert mit dem Ausgabefaktor ergibt dann die REAL-Zahl. Zur Vermeidung von unsinnigen Eingaben für die Zehnerpotenz, (beispielsweise 20) ist der Exponent der Zehnerpotenz als Integerzahl am Eingang EAF der Funktion anzugeben.

Beispiele:

BCD-Wert des Zifferneinstellers	Ausgabefaktor	Exponent des Ausgabefaktors	Ergebnis: $\text{Realzahl} = \text{BCD-Wert} \cdot \text{Faktor}$
1234	10	1	12 340.0
1234	1	0	1234.0
1234	0,01	-2	12.34
1234	0,0001	-4	0.1234

Zum Test der Funktion FC 705 wird an den Eingang BCDW des Code-Bausteins ein Eingangswort gelegt, an das ein vierstelliger Zifferneinsteller angeschlossen ist. Der Ausgangsparameter REALW wird einem Merkerdoppelwortwort MD zugewiesen.

Übergabeparameter:



Beschreibung der Parameter:

BCDW:	WORD	BCD-Wert mit 4 Dekaden im Datenformat WORD
SBCD:	BOOL	Vorzeichen des BCD-Wertes („0“ = positiv, „1“ = negativ)
EAF:	INT	Exponent des Ausgabefaktors
REAW:	REAL	Gewandelter BCD-Wert als Gleitpunktzahl

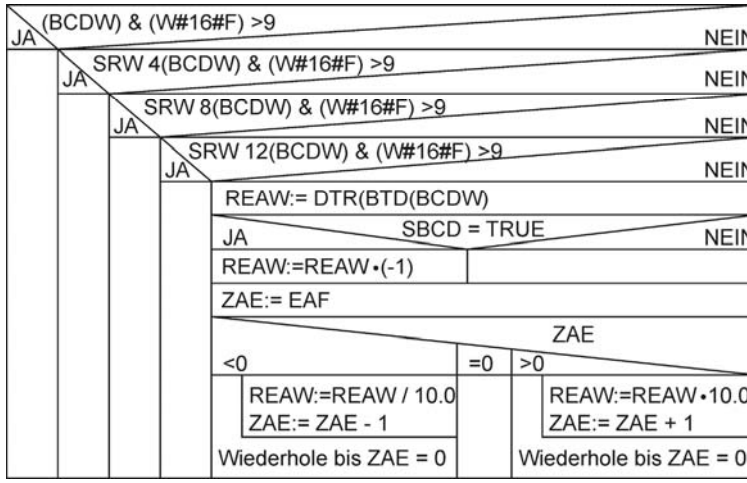
Zuordnungstabelle der Eingänge und Merker:

Eingangsvariable	Symbol	Datentyp	Logische Zuordnung	Adresse
Vorzeichen (Schalter)	S	BOOL	Betätigt S = 1	E 0.0
Zifferneinsteller	EW	WORD	4-stelliger BCD-Wert	EW 10
Exponent Ausgabefaktor	EAF	INT	INTEGER-Wert	EW 12
Merkervariable				
Gleitpunktzahl	MD	REAL	REAL-Format	MD 10

Lösung in STEP 7

Da es in STEP 7 die Operation „Bilden eines Exponentialwertes zur Basis 10“ nicht gibt, wird zur Berechnung der Gleitpunktzahl im vorgegebenen Bereich die vierstellige Zahl des Zifferneinstellers solange mit 10 multipliziert bzw. dividiert, wie es der an den Funktionseingang EAF gelegte Wert angibt.

Struktogramm der Funktion FC 705:



Das Struktogramm zeigt, dass zunächst geprüft wird, ob bei einer Umschaltung am Zifferneinsteller eine Pseudotetrade auftritt. In diesem Falle wird dieser Bearbeitungszyklus des Bausteins beendet und der bisherige Real-Wert am Funktionsausgang REAW ausgegeben.

Bei Auftreten einer Pseudotetrade würde die SPS beim Ausführen der Operation BTD in den STOPP-Zustand gehen.

STEP 7 Programm (AWL-Quelle):

```
FUNCTION FC705 : VOID
NAME : BCD4_REAL
VAR_INPUT                          VAR_OUTPUT          VAR_TEMP
 BCDW : WORD ;     EAF : INT          REAW : REAL ;       ZAE : INT ;
 SBCD : BOOL ;     END_VAR          END_VAR             END_VAR
BEGIN
L  #BCDW;      L  W#16#F;     T  #REAW;      ==I ;          M002: NOP 0;
L  W#16#F;     UW ;                          BEB ;          L  #REAW;
UW ;           L  9;          U  #SBCD;                     L  1.000e+01;
L  9;          >I ;           SPBN  M001;    M003: NOP 0;   /R ;
>I ;           BEB ;          L  #REAW;      L  #REAW;      T  #REAW;
BEB ;          L  #BCDW;      L  -1.00e+0;   L  1.000e+01;  L  #ZAE;
L  #BCDW;      SRW 12;        *R ;           *R ;           +  1;
SRW 4;         L  W#16#F;     T  #REAW;      T  #REAW;      T  #ZAE;
L  W#16#F;     UW ;                          L  #ZAE;       L  0;
UW ;           L  9;          M001: NOP 0;   +  -1;         ==I ;
L  9;          >I ;           L  #EAF;       T  #ZAE;       BEB ;
>I ;           BEB ;          T  #ZAE;       L  0;          SPA M002;
BEB;           L  #BCDW;      L  0;          ==I;
L  #BCDW;      BTD ;          >I ;           BEB ;
SRW 8;         DTR ;          SPB M002       SPA M003;      END_FUNCTION
```

CoDeSys Programm:

Die Auswertung des vorgegebenen Gleitpunktzahlenbereichs am Funktionseingangs EAF (Exponent des Faktors, mit der die BCD-Zahl multipliziert wird) ist bei CoDeSys mit der Funktion EXPT möglich, die den Multiplikationswert direkt mit der Formel Multiplikationswert = 10^{EAF} berechnet. Die Umwandlung der BCD-Zahl in eine Gleitpunktzahl, ist mit dem im Beispiel 6.9 angegebenen Verfahren durchzuführen, da es die Operation BCD_TO_REAL bei CoDeSys nicht gibt.

CoDeSys AWL:

```
FUNCTION FC705:REAL
VAR_INPUT                )               ADD(BCDW
 BCDW: WORD;             AND(BCDW        SHR 4           LDN  SBCD
 SBCD: BOOL;             SHR 8           AND 16#F        JMPC M001
 EAF: INT;               AND 15          MUL 10          LD  INTW
END_VAR                  LE 10           )               MUL -1
VAR                      )               ADD(BCDW        ST  INTW
 INTW: INT;              AND(BCDW        SHR 8
END_VAR                  SHR 12          AND 16#F        M001:
                         AND 15          MUL 100         LD  INTW
LD  BCDW                 LE  10          )               INT_TO_REAL
AND 15                   )               ADD( BCDW       ST  FC705
LE  10                   NOT             SHR 12
AND(BCDW                 RETC            AND 15          LD   10
SHR 4                                    MUL 1000        EXPT EAF
AND 15                   LD  BCDW        )               MUL  FC705
LE 10                    AND 16#F        ST  INTW        ST   FC705
```

■ **Beispiel 7.6: Umwandlung REAL_TO_BCD für 4 Ziffern (FC 706: BCD4_REAL)**

Für die eigene Programmbibliothek ist eine Funktion FC 706 (REAL_BCD4) zu entwerfen, die eine Gleitpunktzahl (Realzahl) am Funktionseingang REAW in einen BCD-Wert (BCDW) mit wählbarem Ziffernbereich wandelt. Damit lassen sich mit einer BCD-Anzeige auch sehr kleine oder sehr große Zahlen in Verbindung mit einem Ausgabefaktor darstellen. Die Funktion FC 706 benötigt deshalb einen weiteren Eingang, um den Ausgabefaktor als Zehnerpotenz vorzugeben. Die Gleitpunktzahl multipliziert mit dem Ausgabefaktor ergibt dann den 4-stelligen BCD-Wert am Ausgang der Funktion.

Je nach Ausgabefaktor können Ziffern der Gleitpunktzahl links oder rechts abgeschnitten werden. Werden durch einen ungünstig gewählten Ausgabefaktor Stellen der Gleitpunktzahl links abgeschnitten, wäre der ausgegebene BCD-Wert nicht sinnvoll. Ebenso falsch wäre ein BCD-Wert null, wenn die Gleitpunktzahl verschieden von null ist. In beiden Fällen soll ein weiterer Ausgang FEH der Funktion den Anzeigefehler durch ein 1-Signal melden und am Funktionsausgang BCDW der Wert 16#0000 erscheinen.

Zur Vermeidung von unsinnigen Eingaben für den Ausgabefaktor, (keine Zehnerpotenz, sondern beispielsweise 5) ist am Eingang EAF der Exponent der Zehnerpotenz des Ausgabefaktors als Integerzahl anzugeben.

Beispiele für den BCD-Wert und den Fehler FEH bei unterschiedlichen Ausgabefaktoren:

Realzahl REAW	Ausgabe-faktor	Exponent EAF	Berechnung des Ausgabewertes: Realzahl · Ausgabefaktor	Ausgabe BCDW	Fehler FEH
12.345	0,01	–2	12.345 · 0.01 = 0.12345	0000	TRUE
12.345	1	0	12.345 · 1 = 12.345	0012	FALSE
12.345	100	2	12.345 · 100 = 1234.5	1234	FALSE
12.345	1000	3	12.345 · 1000 = 12345	2345	TRUE
0.0	x	x	0.0 · x = 0	0000	FALSE

Zum Test der Funktion FC 706 wird der Code-Baustein aufgerufen und REAL-Werte über „Variable steuern“ in das am Eingang REAW der Funktion liegende Merkerdoppelwort MD bzw. INTEGER-Werte

in das am Eingang EAF liegende Merkerwort MW geschrieben. Der Ausgang der Funktion wird einem Ausgangswort AW zugewiesen, an das eine 4-stellige Ziffernanzeige angeschlossen ist.

Übergabeparameter: Beschreibung der Parameter:

REAW:	REAL	Gleitpunktzahl
EAF:	INT	Exponent des Ausgabefaktors
BCDW:	WORD	BCD-Wert mit 4 Dekaden im Datenformat WORD
SBCD:	BOOL	Vorzeichen des BCD-Wertes („0“ = positiv, „1“ = negativ)
FEH:	BOOL	Falscher Ausgabewert

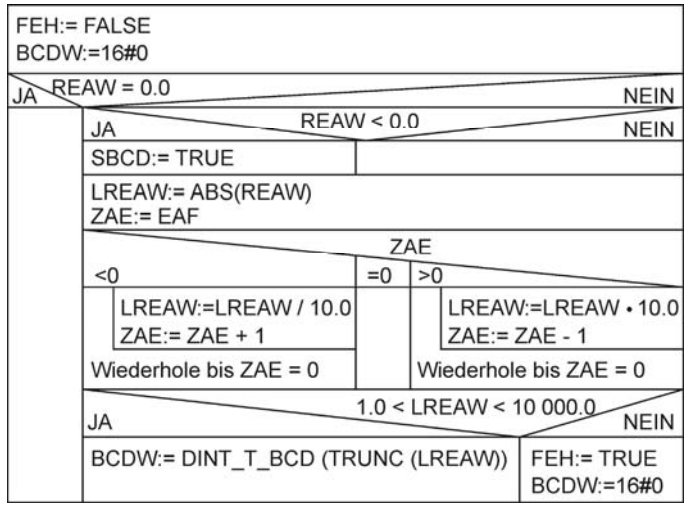
Zuordnungstabelle der Merker und Ausgänge:

Merkervariable	Symbol	Datentyp	Logische Zuordnung	Adresse
Real-Wert	REAW	REAL	Gleitpunktzahl	MD 10
Ausgabefaktor	EAF	INT	INTEGER-Wert	MW 20
Ausgangsvariable				
Ziffernanzeige	BCDW	WORD	4-stelliger BCD-Wert	AW 12
Vorzeichen-Anzeige	P_SBCD	BOOL	Leuchtet P_SBCD = 1	A 4.0
Ausgabefehler	P_FEH	BOOL	Sinnlose Ausgabe P_FEH = 1	A 4.1

Lösung in STEP 7

Wie im vorherigen Beispiel 7.5 erfolgt die Auswertung des Exponenten des Ausgabefaktors durch wiederholte Multiplikation bzw. Division mit 10.

Struktogramm der Funktion FC 706:



Vor Auswertung des Funktionseingangs EAF, wird abgefragt, ob die Gleitpunktzahl negativ ist. Falls JA, wird dem Ausgang SBCD der boolesche Wert TRUE zugewiesen. Mit der Operation ABS wird der Betrag der Gleitpunktzahl gebildet.

STEP 7 Programm (AWL-Quelle):

```
FUNCTION FC706 : VOID
NAME : BCD4_REAL
VAR_INPUT                VAR_OUTPUT              VAR_TEMP
 REALW : REAL;            BCDW : WORD ;           LREAW : REAL ;
 EAF : INT                SBCD : BOOL ;           ZAE : INT ;
END_VAR                   FEH  : BOOL ;          END_VAR
                          END_VAR
```

```
BEGIN
CLR ;          L  #EAF;       +  -1;         T  #ZAE;       L  #LREAW;
=  #FEH;       T  #ZAE;       T  #ZAE;       L  0;          L  9.999e+03;
L  0;          L  0;          L  0;          ==I ;          >R ;
T  #BCDW;      <I ;           ==I ;          SPB M002;      ) ;
L  #REAW;      SPB M001;      SPB M002;      SPA M001;      =  #FEH
L  0.00e+00;   ==I ;          SPA M003;                     BEB
==R   ;        SPB  M002;     M001: NOP 0;   M002: NOP 0;
BEB   ;        M003: NOP 0;   L  #LREAW;     O(;            L  #LREAW;
<R ;           L  #LREAW;     L  1.0e+01;    L  #LREAW;     TRUNC ;
=  #SBCD;      L 1.00e+01;    /R ;           L 1.00e+000;   DTB ;
L  #REAW;      *R ;           T  #LREAW;     <R ;           T  #BCDW;
ABS ;          T  #LREAW;     L  #ZAE;       ) ;
T  #LREAW;     L  #ZAE;       +  1;          O( ;           END_FUNCTION
```

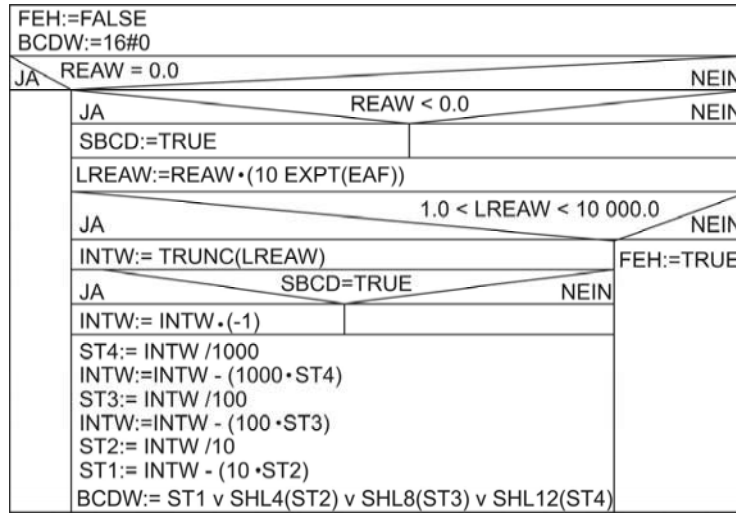
Lösung in CoDeSys

Das nebenstehende Struktogramm zeigt die Anweisungsfolge des CoDeSys Programms. Die Berechnung des Ausgabewertes kann wieder mit der Operation EXPT ausgeführt werden. Der lokale Realwert wird dabei nach der Formel berechnet:

$$LREAW = REAW \cdot 10^{EAF}$$

Für die Umwandlung einer Ganzzahl in eine BCD-Zahl wird wieder der bereits in Beispiel 6.10 dargestellte Algorithmus verwendet.

Struktogramm:



CoDeSys AWL:

```
TYPE FC706_OUT:   ST2: WORD;     EXPT EAF          LD  INTW     ST  ST2
STRUCT            ST1: WORD;     )                 DIV 1000     LD  INTW
 BCDW:WORD;       LREAW:REAL;    ST LREAW          ST  ST4      SUB( ST2
 SBCD:BOOL;       _INT_0:INT;    LD LREAW          LD  INTW     MUL  10
 FEH:BOOL;      END_VAR          LT 1.0            SUB( ST4     )
END_STRUCT                      OR( LREAW         MUL 1000     ST  ST1
END_TYPE         LD 0           GT 9999.0         )            LD  ST1
                 ST FC706.BCDW  )                 ST  INTW     OR( ST2
FUNCTION         LD FALSE       ST FC706.FEH      LD  INTW     SHL 4
FC706:FC706_OUT  ST FC706.FEH   RETC              DIV 100      )
VAR_INPUT        LD REAW        LD LREAW          ST  ST3      OR( ST3
 REAW: REAL;     EQ 0.0         TRUNC             LD  INTW     SHL 8
 EAF:INT;        RETC           ST INTW           SUB(ST3      )
END_VAR          LD REAW        LD INTW           MUL 100      OR( ST4
VAR              LT 0.0         MUL -1            )            SHL 12
 INTW:INT;       ST FC706.SBCD  ST _INT_0         ST  INTW     )
 ST4: WORD;      LD REAW        LD FC706.SBCD     LD  INTW     ST
 ST3: WORD;      MUL ( 10       SEL INTW,_INT_0   DIV 10       FC706.BCDW
                                ST  INTW
```