# 6 Programmablaufkontrolle

## 6.1 Programmverzweigungen

### 6.1.1 Sprunganweisung

Die einfachste Form der Programmverzweigung ist die der unbedingten Sprunganweisung. Sie wird ohne eine bestimmte Bedingung zu reflektieren auf jeden Fall ausgeführt. Durch die Anweisung

GOTO

```
GOTO MERKER
```

springt der Programmzeiger an die Position MERKER. Die Position muß allerdings mit

```
MERKER:
```

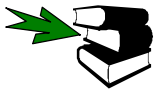
auch irgendwo im Programm definiert sein. Die Sprunganweisung selber läßt keine Rückschlüsse auf die damit erzeugte Programmstruktur zu. Deshalb sollte der Name der Sprungmarke möglichst so gewählt werden, daß die damit hervorgerufene Sprungaktion etwas verständlicher wird. So ist es z.B. ein Unterschied, ob man schreibt

```
GOTO MARKE_1
```

oder

```
GOTO KLEBERSTOP
```

Da die GOTO–Anweisung sehr schnell zu unstrukturierten und unübersichtlichen Programmen führt, und ferner jede GOTO–Anweisung durch eine andere Schleifenanweisung ersetzt werden kann, sollte möglichst wenig mit GOTO gearbeitet werden.



**Ein Beispiel zu “GOTO” finden Sie im Abschnitt 6.2.3.**

**NOTIZEN:**

### 6.1.2 Bedingte Verzweigung

IF Die strukturierte IF-Anweisung gestattet die Formulierung bedingter Anweisungen und die Auswahl aus zwei Alternativen. In der allgemeinen Form lautet die Anweisung

```
IF Ausführbedingung THEN
   Anweisungen
ELSE
   Anweisungen
ENDIF
```

Die Ausführbedingung ist ein boolscher Ausdruck. Abhängig von dem Wert der Ausführbedingung wird entweder der erste (THEN-Block) oder der zweite Anweisungsblock (ELSE-Block) ausgeführt. Der ELSE-Block ist optional und kann daher auch fehlen. Falls die Ausführbedingung=FALSE ist, wird keine Anweisung ausgeführt und hinter dem ENDIF weitergearbeitet.

Es sind beliebig viele Anweisungen zulässig. Die Anweisungen können insbesondere auch weitere IF-Anweisungen sein. Eine Schachtelung von IF-Blöcken ist also möglich. Jede IF-Anweisung muß jedoch mit einem eigenen ENDIF abgeschlossen werden.

In der folgenden Programmsequenz wird, sofern Eingang 10 auf FALSE ist, die HOME-Position angefahren. Ist Eingang 10 gesetzt, dann wird, wenn der Wert von Variable A größer als der von B ist, zunächst Ausgang 1 gesetzt und Punkt 1 angefahren. Unabhängig von A und B wird auf jeden Fall bei gesetztem Eingang 10 die Variable A um 1 erhöht und dann die HOME-Position angefahren:

```
...
INT A,B
...
IF $IN[10]==FALSE THEN
   PTP HOME
ELSE
   IF A>B THEN
      $OUT[1]=TRUE
      LIN PUNKT1
   ENDIF
   A=A+1
   PTP HOME
ENDIF
...
```

**NOTIZEN:**

### 6.1.3 Verteiler

Liegen mehr als 2 Alternativen vor, kann dies entweder mit einer geschachtelten IF-Konstruktion oder – wesentlich komfortabler – mit dem Verteiler SWITCH programmiert werden.

SWITCH

Die SWITCH-Anweisung ist eine Auswahlanweisung für verschiedene Programmzweige. Ein Auswahlkriterium wird vor der SWITCH-Anweisung mit einem bestimmten Wert belegt.

Blockkennung

Stimmt dieser Wert mit einer Blockkennung überein, so wird der entsprechende Programmzweig abgearbeitet und das Programm springt ohne Berücksichtigung der folgenden Blockkennungen zur ENDSWITCH-Anweisung vor. Stimmt keine Blockkennung mit dem Auswahlkriterium überein, so wird, falls vorhanden, ein DEFAULT-Block abgearbeitet. Anderenfalls wird mit der Anweisung nach der ENDSWITCH-Anweisung fortgefahren.

Es ist zulässig, einem Programmzweig mehrere Blockkennungen zuzuordnen. Umgekehrt ist es nicht sinnvoll eine Blockkennung mehrmals zu verwenden, da immer nur der erste Programmzweig mit der entsprechenden Kennung berücksichtigt wird.

Zulässige Datentypen des Auswahlkriteriums sind INT, CHAR und ENUM. Der Datentyp von Auswahlkriterium und Blockkennung muß übereinstimmen.

Die DEFAULT-Anweisung kann fehlen, darf aber innerhalb einer SWITCH-Anweisung nur einmal vorkommen.

Mit der SWITCH-Anweisung können Sie somit zum Beispiel in Abhängigkeit von einer Programmnummer verschiedene Unterprogramme aufrufen. Die Programmnummer könnte beispielsweise von der SPS an die digitalen Eingänge der KR C1 angelegt werden (s. Abschnitt 7.3 zur SIGNAL-Anweisung). Dadurch steht sie als Auswahlkriterium in Form eines Integer-Wertes zur Verfügung.


```
DEF MAIN()
...
SIGNAL PROG_NR $IN[1] TO $IN[4]
                ;In die INT-Variable PROG_NR wird von der SPS
                ;jetzt die gewuenschte Programmnummer abgelegt
...
SWITCH PROG_NR
   CASE 1              ;wenn PROG_NR=1
        TEIL_1()
   CASE 2              ;wenn PROG_NR=2
        TEIL_2()
        TEIL_2A()
   CASE 3,4,5          ;wenn PROG_NR=3, 4 oder 5
        $OUT[3]=TRUE
        TEIL_345()
   DEFAULT             ;wenn PROG_NR<>1,2,3,4,5
        ERROR_UP()
ENDSWITCH
...
END
```

In ähnlicher Weise ist das standardmäßig auf der Steuerung vorhandene CELL-Programm (CELL.SRC) aufgebaut.

## 6.2 Schleifen

Die nächste Grundstruktur zur Programmablaufkontrolle sind die Schleifen, die bis zum Eintritt einer bestimmten Bedingung die wiederholte Abarbeitung einer oder mehrerer Anweisungen beinhalten. Schleifen werden nach der Form der Bedingung und der Stelle der Abfrage auf Fortsetzung unterschieden.

Ein Sprung von außen in einen Schleifenkörper ist nicht erlaubt und wird von der Steuerung abgelehnt (Fehlermeldung).

### 6.2.1 Zählschleife

Zählschleifen werden so lange ausgeführt, bis eine Zählvariable entweder durch Hoch- oder Runterzählen einen bestimmten Endwert über- oder unterschreitet. In KRL steht dafür die FOR-Anweisung zur Verfügung. Mit

FOR

```
FOR Zähler = Start TO Ende STEP Schrittweite
    Anweisungen
ENDFOR
```

läßt sich somit sehr übersichtlich eine bestimmte Anzahl von Durchläufen programmieren.

Als **Start**-Wert und **Ende**-Wert des Zählers geben Sie jeweils einen Ausdruck vom Typ Integer an. Die Ausdrücke werden einmal vor Beginn der Schleife ausgewertet. Die INT-Variable **Zähler** (muß vorher deklariert sein) wird mit dem Startwert vorbesetzt und nach jedem Schleifendurchlauf um die Schrittweite erhöht oder vermindert.

Die **Schrittweite** darf keine Variable sein und darf nicht Null sein. Wenn keine Schrittweite angegeben ist, hat sie den Standardwert 1. Auch negative Werte sind für die Schrittweite zugelassen.

Zu jeder FOR-Anweisung muß es eine **ENDFOR**-Anweisung geben. Das Programm wird nach Beendigung des letzten Schleifendurchlaufs mit der ersten Anweisung hinter ENDFOR fortgesetzt.

Den Wert des Zählers können Sie sowohl innerhalb als auch außerhalb der Schleife benutzen. Innerhalb der Schleifen dient er zum Beispiel als aktueller Index für die Bearbeitung von Feldern. Nach dem Verlassen der Schleife hat der Zähler den zuletzt angenommenen Wert (also Ende+Schrittweite).

In folgendem Beispiel werden zunächst die Achsgeschwindigkeiten $VEL_AXIS[1]…$VEL_AXIS[6] auf 100% gesetzt. Danach werden die Komponenten eines 2-dimensionalen Feldes mit den berechneten Werten initialisiert. Das Ergebnis ist in Tab. 21 dargestellt.

```
DEF FOR_PROG()
...
INT I,J
INT FELD[10,6]
...
FOR I=1 TO 6
    $VEL_AXIS[I] = 100   ;alle Achsgeschwindigkeiten auf 100%
ENDFOR
...
FOR I=1 TO 9 STEP 2
    FOR J=6 TO 1 STEP -1
        FELD[I,J] = I*2 + J*J
        FELD[I+1,J] = I*2 + I*J
    ENDFOR
ENDFOR
       ;I hat jetzt den Wert 11, J den Wert 0
...
END
```

| Index | | I = | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **J =** | **6** | 38 | 8 | 42 | 24 | 46 | 40 | 50 | 56 | 54 | 72 |
| | **5** | 27 | 7 | 31 | 21 | 35 | 35 | 39 | 49 | 43 | 63 |
| | **4** | 18 | 6 | 22 | 18 | 26 | 30 | 30 | 42 | 34 | 54 |
| | **3** | 11 | 5 | 15 | 15 | 19 | 25 | 23 | 35 | 27 | 45 |
| | **2** | 6 | 4 | 10 | 12 | 14 | 20 | 18 | 28 | 22 | 36 |
| | **1** | 3 | 3 | 7 | 9 | 11 | 15 | 15 | 21 | 19 | 27 |

**Tab. 21** Ergebnis der Berechnung aus Beispiel 5.2

**NOTIZEN:**

### 6.2.2 Abweisende Schleife

WHILE

Die WHILE-Schleife fragt zu Beginn der Wiederholung nach einer Ausführbedingung. Sie ist eine abweisende Schleife, weil sie kein einziges Mal durchlaufen wird, wenn die Ausführbedingung von Anfang an schon nicht erfüllt ist. Die Syntax der WHILE-Schleife lautet:

```
WHILE Ausfuehrbedingung
      Anweisungen
ENDWHILE
```

Die Ausfuehrbedingung ist ein logischer Ausdruck, der eine boolsche Variable, ein boolscher Funktionsaufruf oder eine Verknüpfung mit einem boolschen Ergebnis sein kann.

Der Anweisungsblock wird ausgeführt, wenn die logische Bedingung den Wert TRUE hat, d.h. die Ausführbedingung erfüllt ist. Wenn die logische Bedingung den Wert FALSE hat, wird das Programm mit der nächsten Anweisung hinter ENDWHILE fortgesetzt. Jede WHILE-Anweisung muß deshalb durch eine ENDWHILE-Anweisung beendet werden.

Die Verwendung von WHILE wird aus Beispiel 5.3 ersichtlich.


```
DEF WHILE_PR()
...
INT X,W
...
WHILE $IN[4] == TRUE  ;Durchlauf solange Eingang 4 gesetzt ist
   PTP PALETTE
   $OUT[2] = TRUE
   PTP POS_2
   $OUT[2] = FALSE
   PTP HOME
ENDWHILE
...
X = 1
W = 1
WHILE W < 5;          ;Durchlauf solange W kleiner 5 ist
   X = X * W
   W = W + 1
ENDWHILE
                        ;W ist jetzt 5
...                     ;X ist jetzt 1$2$3$4 = 24
W = 100
WHILE W < 100          ;Durchlauf solange W kleiner 100 ist
   $OUT[15] = TRUE
   W = W + 1
ENDWHILE
...                   ;Schleife wird nie durchlaufen, W bleibt 100
END
```

**NOTIZEN:**

### 6.2.3 Nicht abweisende Schleife

REPEAT

Das Gegenstück zur WHILE-Schleife ist die REPEAT-Schleife. Bei REPEAT wird erst am Ende der Schleife nach einer Abbruchbedingung gefragt. Deshalb werden REPEAT-Schleifen auf jeden Fall einmal durchlaufen, auch wenn die Abbruchbedingung schon vor Schleifenanfang erfüllt ist.

```
REPEAT
   Anweisungen
UNTIL Abbruchbedingung
```

Die Abbruchbedingung ist analog zur Ausführbedingung der WHILE-Schleife ein logischer Ausdruck, der eine boolsche Variable, ein boolscher Funktionsaufruf oder eine Verknüpfung mit einem boolschen Ergebnis sein kann:


```
DEF REPEAT_P()
...
INT W
...
REPEAT
   PTP PALETTE
   $OUT[2]=TRUE
   PTP POS_2
   $OUT[2]=FALSE
   PTP HOME
UNTIL $IN[4] == TRUE   ;Durchlauf bis Eingang 4 gesetzt wird
...
X = 1
W = 1
REPEAT
   X = X * W
   W = W + 1
UNTIL W == 4           ;Durchlauf bis W gleich 4 wird
                       ;W ist jetzt 4
...                    ;X ist jetzt 1S2S3S4 = 24
W = 100
REPEAT
   $OUT[15] = TRUE
   W = W + 1
UNTIL W > 100           ;Durchlauf bis W groesser 100 wird
...                     ;mindestens 1 Schleifendurchlauf, d.h.
                        ;W ist jetzt 101, Ausgang 15 ist gesetzt
END
```

Mit WHILE und REPEAT haben Sie nun ein sehr mächtiges Werkzeug zur strukturierten Programmierung an der Hand, mit dem Sie die meisten GOTO-Anweisungen ersetzen können. Die Anweisungsfolge

```
...
X = 0
G = 0
MERKER:
X = X + G
G = G + 1
   IF G > 100 THEN
      GOTO FERTIG
   ENDIF
GOTO MERKER:
FERTIG:
...
```

läßt sich beispielsweise mit REPEAT sehr viel eleganter verwirklichen:

```
...
X = 0
G = 0
REPEAT
   X = X + G
   G = G + 1
UNTIL G > 100
...
```

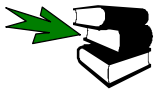
**NOTIZEN:**

### 6.2.4 Endlosschleife

LOOP Mit der LOOP-Anweisung lassen sich Endlosschleifen programmieren:

```
LOOP
   Anweisungen
ENDLOOP
```

Die wiederholte Ausführung des Anweisungsblocks läßt sich nur mittels der EXIT-Anweisung beenden.



Abschnitt 6.2.5 zur EXIT-Anweisung

### 6.2.5 Vorzeitige Beendigung von Schleifendurchläufen

EXIT Mit der EXIT-Anweisung können Sie jede Schleife vorzeitig beenden. Durch Aufruf von EXIT innerhalb des Anweisungsblocks der Schleife werden die Schleifendurchläufe sofort beendet und das Programm wird hinter der Schleifenendanweisung fortgesetzt.

Durch die Wahl geschickter Abbruch- oder Ausführbedingungen ist die EXIT-Anweisung in REPEAT- oder WHILE-Schleifen meist nicht notwendig. Für die Endlosschleife stellt EXIT allerdings die einzige Möglichkeit dar, die Schleifendurchläufe zu beenden. Dazu folgendes Beispiel:


```
DEF EXIT_PRO()
PTP HOME
LOOP                          ;Start der Endlosschleife
   PTP POS_1
   LIN POS_2
   IF $IN[1] == TRUE THEN
      EXIT                    ;Abbruch, wenn Eingang 1 gesetzt
   ENDIF
   CIRC HELP_1,POS_3
   PTP POS_4
ENDLOOP                       ;Ende der Endlosschleife
PTP HOME
END
```

**NOTIZEN:**

## 6.3 Warteanweisungen

WAIT Mit der WAIT-Anweisung können Sie das Anhalten des Programms bis zum Eintritt einer bestimmten Situation bewirken. Man unterscheidet zwischen dem Warten auf das Eintreten eines bestimmten Ereignisses und dem Einlegen von Wartezeiten.

### 6.3.1 Warten auf ein Ereignis

Mit der Anweisung

```
WAIT FOR Bedingung
```

können Sie den Programmlauf bis zum Eintreten des mit Bedingung spezifizierten Ereignisses anhalten:

- Wenn der logische Ausdruck Bedingung beim WAIT-Aufruf bereits TRUE ist, wird der Programmablauf nicht angehalten (es wird jedoch trotzdem ein Vorlaufstop ausgelöst).
- Ist Bedingung FALSE, so wird der Programmlauf angehalten, bis der Ausdruck den Wert TRUE annimmt.

Beispiele:

```
WAIT FOR $IN[14]          ;wartet bis Eingang 14 TRUE ist
WAIT FOR BIT_1 == FALSE   ;wartet bis die Var. BIT_1 = FALSE ist
```

Der Compiler erkennt nicht, wenn der Ausdruck durch eine fehlerhafte Formulierung nie den Wert TRUE annehmen kann. In diesem Fall wird der Programmablauf endlos angehalten, weil der Interpreter auf eine unerfüllbare Bedingung wartet.

### 6.3.2 Wartezeiten

Die WAIT SEC-Anweisung dient zum Programmieren von Wartezeiten in Sekunden:

```
WAIT SEC Zeit
```

Zeit ist ein arithmetischer REAL-Ausdruck, mit dem Sie die Anzahl der Sekunden angeben, für die der Programmlauf unterbrochen werden soll. Ist der Wert negativ, so wird nicht gewartet.

Beispiele:

```
WAIT SEC 17.542
WAIT SEC ZEIT*4+1
```

**NOTIZEN:**

## 6.4 Anhalten des Programms

HALT Wollen Sie den Programmablauf unterbrechen und die Bearbeitung anhalten, so programmieren Sie die Anweisung

```
HALT
```

Die zuletzt durchlaufene Bewegungsanweisung wird jedoch noch vollständig ausgeführt. Fortgesetzt wird der Programmlauf nur durch Drücken der Starttaste. Danach wird die nächste Anweisung nach HALT ausgeführt.

**Sonderfall: In einer Interrupt-Routine wird der Programmlauf durch eine HALT-Anweisung erst nach vollständiger Abarbeitung des Vorlaufs angehalten (s. Abschnitt 9 Interrupt).**

Ausnahme: Bei Programmierung einer BRAKE – Anweisung wird sofort gehalten.

**NOTIZEN:**

## 6.5 Quittieren von Meldungen

CONFIRM Mit der Anweisung

```
CONFIRM V_Nummer
```

können Sie quittierbare Meldungen programmgesteuert quittieren. Nach erfolgreicher Quittierung ist die mit der Verwaltungsnummer V_Nummer spezifizierte Meldung nicht mehr vorhanden.

Nach dem Aufheben eines Stop-Signals wird beispielsweise stets eine Quittungsmeldung ausgegeben. Vor dem Weiterarbeiten muß diese zunächst quittiert werden. Folgendes Unterprogramm erkennt und quittiert diese Meldung automatisch, sofern die richtige Betriebsart (kein Handbetrieb) gewählt ist, und der Stop-Zustand tatsächlich aufgehoben wurde (da ein Roboterprogramm nicht startbar ist, wenn eine Quittungsmeldung anliegt, muß das Unterprogramm in einem Submit-File laufen):


```
DEF AUTO_QUIT()
INT M
DECL STOPMESS MLD     ;vordefinierte Strukturtyp fuer Stopmel-
dungen
IF $STOPMESS AND $EXT THEN      ;Stopmeldung und Betriebsart
prüfen
   M=MBX_REC($STOPMB_ID,MLD)    ;aktuellen Zustand in MLD einle-
sen
   IF M==0 THEN         ;Ueberpruefen, ob Quittierung erfolgen
darf
      IF ((MLD.GRO==2) AND (MLD.STATE==1)) THEN
         CONFIRM MLD.CONFNO     ;Quittierung dieser Meldung
       ENDIF
   ENDIF
ENDIF
END
```

**NOTIZEN:**