9 Interrupt–Behandlung

Beim Einsatz von Robotern in komplexen Fertigungsanlangen besteht die Notwendigkeit, daß der Roboter auf bestimmte externe oder interne Ereignisse gezielt und sofort reagieren kann und parallel zum Roboterprozeß Aktionen ausgeführt werden können. Das heißt, ein laufendes Roboterprogramm muß unterbrochen und ein Unterbrechungsprogramm bzw. –funktion gestartet werden. Nach Abarbeitung des Unterbrechungsprogramms soll, wenn nichts weiteres vereinbart ist, das unterbrochene Roboterprogramm wieder fortgesetzt werden.

Dieses gezielte Unterbrechen bzw. Starten eines Programms wird durch die Interrupt–Anweisung ermöglicht. Damit hat der Anwender die Möglichkeit, per Programm auf ein Ereignis zu reagieren, welches zeitlich nicht synchron zum Programmablauf auftritt.

Interrupts können von

- Geräten, wie Sensoren, Peripherieeinheiten, etc.,
- Fehlermeldungen
- dem Anwender oder durch
- Sicherheitsschaltungen

ausgelöst werden. Beispielsweise kann nach Drücken des Not–Aus–Schalters eine Interrupt–Routine aufgerufen werden, die bestimmte Ausgangssignale zurücksetzt (vorbereitetes Programm `IR_STOPM.SRC`).

NOTIZEN:

9.1 Deklaration

Bevor ein Interrupt aktiviert werden kann, müssen zunächst einmal die möglichen Unterbrechungsursachen und die jeweilige Reaktion des Systems darauf definiert werden.

INTERRUPT

Dies geschieht mit der Interrupt–Deklaration, bei der sie jeder Unterbrechung eine Priorität, ein Ereignis und die aufzurufende Interrupt–Routine zuordnen müssen. Die vollständige Syntax lautet:

```
INTERRUPT DECL Priorität WHEN Ereignis DO Unterprogramm
```

Zur Bedeutung der Argumente siehe Tab. 13.

Argument	Datentyp	Bedeutung
Priorität	INT	Arithmetischer Ausdruck, der die Priorität der Unterbrechung angibt. Es stehen die Prioritätsebenen 1…39 und 81…128 zur Verfügung. Die Werte 40...80 sind für eine automatische Prioritätsvergabe durch das System reserviert. Die Unterbrechung der Ebene 1 hat die höchste Priorität.
Ereignis	BOOL	Logischer Ausdruck, der das Unterbrechungsereignis definiert. Zugelassen sind: • eine boolsche Konstante • eine boolsche Variable • ein Signalname • ein Vergleich
Unterprogramm		Name des Interruptprogramms, das bei Auftreten des Ereignisses abgearbeitet werden soll.

Tab. 13 Argumente in der Interrupt–Deklaration

Die Anweisung

```
INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```

deklariert z.B. einen Interrupt der Priorität 4, der das Unterprogramm `UP1()` aufruft sobald der Eingang 3 auf High geht.

Die Interrupt–Deklaration ist eine Anweisung. Sie darf daher nicht im Vereinbarungsteil stehen!

Ein Interrupt wird erst ab der Programmebene erkannt, in der er deklariert ist. In darüber liegenden Programmebenen wird der Interrupt trotz Aktivierung nicht erkannt. Das heißt, ein in einem Unterprogramm deklarierter Interrupt, ist im Hauptprogramm nicht bekannt (s. Abb. 37).

GLOBAL

Wird der Interrupt hingegen als `GLOBAL` vereinbart so kann er in einem beliebigen Unterprogramm deklariert werden und verliert seine Gültigkeit beim Verlassen dieser Ebene nicht (s. Abb. 37).

```
GLOBAL INTERRUPT DECL 4 WHEN $IN[3]==TRUE DO UP1()
```

- Eine Deklaration kann jederzeit durch eine neue überschrieben werden.
- Ein GLOBALER Interrupt unterscheidet sich von einem normalen Interrupt dadurch, daß er nach Verlassen des Unterprogramms in dem er vereinbart wurde weiterhin gültig ist.
- Zur gleichen Zeit dürfen maximal 32 Interrupts deklariert sein.
- In der Interruptbedingung darf nicht auf Strukturvariablen oder –komponenten zugegriffen werden.
- Es dürfen keine Laufzeitvariablen als Parameter der Interruptroutine übergeben werden, außer es handelt sich um GLOBAL oder in der Datenliste vereinbarte Variablen.

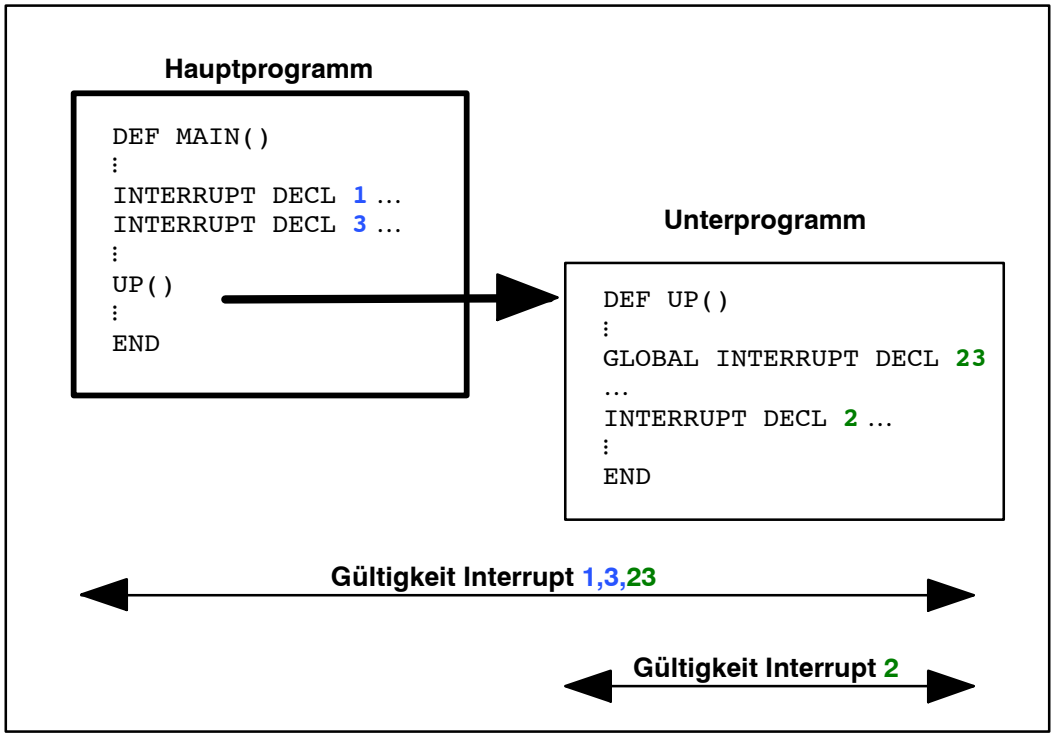


Abb. 37 GültigkeitsbereicheinesInterruptsabhängigvomDeklarationsortund–art

NOTIZEN:

9.2 Aktivieren von Interrupts

Interrupt einschalten

Nach der Deklaration ist ein Interrupt zunächst ausgeschaltet. Mit der Anweisung

```
INTERRUPT ON 4
```

wird der Interrupt mit der Priorität 4, mit

```
INTERRUPT ON
```

werden alle Interrupts eingeschaltet. Erst nach dem Einschalten kann eine Reaktion auf die definierte Unterbrechung erfolgen. Das Unterbrechungsereignis wird nun zyklisch überwacht.

flankengetriggert

Die Überprüfung des Ereignisses erfolgt dabei flankengetriggert, das heißt, ein Interrupt wird nur ausgelöst, wenn die logische Bedingung vom Zustand `FALSE` in den Zustand `TRUE` wechselt, nicht jedoch, wenn die Bedingung beim Einschalten bereits `TRUE` war.

Aus Rechenzeitgründen dürfen zur gleichen Zeit nur 16 Interrupts eingeschaltet sein. Dies ist insbesondere bei der globalen Aktivierung aller Interrupts zu beachten.

Interrupt ausschalten

In gleicher Weise wie das Einschalten funktioniert auch das Ausschalten einzelner oder aller Interrupts:

```
INTERRUPT OFF 4
```

oder

```
INTERRUPT OFF
```

Sperren / Freigeben

Mit den Schlüsselworten `ENABLE` und `DISABLE` können eingeschaltete Interrupts einzeln oder global freigegeben oder gesperrt werden.

Die Sperranweisung ermöglicht einen Schutz von bestimmten Programmteilen vor Unterbrechung. Ein gesperrter Interrupt wird zwar erkannt und gespeichert, aber nicht ausgeführt. Erst wenn eine Freigabe erfolgt, werden die aufgetretenen Interrupts in der Reihenfolge ihrer Priorität bearbeitet.

```
DISABLE 4
```

oder

```
DISABLE
```

Auf ein gespeichertes Ereignis wird nicht mehr reagiert, wenn der Interrupt vor der Auslösung ausgeschaltet wird. Falls ein Interrupt während er gesperrt ist mehrmals auftritt, wird er nach seiner Freigabe nur einmal ausgeführt.

Die Voraussetzungen für das Auslösen eines Interrupts sind:

- Der Interrupt muß deklariert sein (`INTERRUPT DECL` ...)
- Der Interrupt muß eingeschaltet sein (`INTERRUPT ON`)
- Der Interrupt darf nicht gesperrt sein
- Das zugehörige Ereignis muß eingetreten sein (flankengetriggert)

Priorität

Bei gleichzeitigem Auftreten von Interrupts wird erst der Interrupt höchster Priorität bearbeitet, dann die Interrupts niederer Priorität. Die Prioritätsstufe 1 hat dabei die höchste Priorität, Stufe 128 die niedrigste.

Bei Erkennen eines Ereignisses wird die aktuelle Ist–Position des Roboters gespeichert und die Interrupt–Routine aufgerufen. Der aufgetretene Interrupt sowie alle Interrupts niederer Priorität werden für die gesamte Dauer der Abarbeitung gesperrt. Bei der Rückkehr aus dem Interruptprogramm wird die sogenannte implizite Sperre, auch für den aktuellen Interrupt, wieder aufgehoben.

Der Interrupt kann nun also bei erneutem Auftreten (auch während des Interruptprogramms) wieder abgearbeitet werden. Soll dies verhindert werden, so muß der Interrupt explizit vor dem Rücksprung gesperrt oder ausgeschaltet werden.

Ein Interrupt kann nach dem ersten Befehl im Interruptprogramm durch Interrupts höherer Priorität unterbrochen werden. Im ersten Befehl hat der Programmierer somit die Möglichkeit, dies durch Sperren/Ausschalten eines oder aller Interrupts zu verhindern. Schaltet sich ein Interrupt im Interruptprogramm selbst ab, so wird dieses natürlich zu Ende abgearbeitet.

Nach Beendigung eines höher prioren Interrupts wird das unterbrochene Interruptprogramm an der Stelle fortgesetzt, an der es unterbrochen wurde.

- An ein Interruptprogramm können IN–Parameter übergeben werden.
- Soll ein lokales Interruptprogramm einen Parameter zurückliefern, so muß die Variable in der Datenliste des Hauptprogramms deklariert sein. Bei globalen Interruptprogrammen muß mit der Datenliste `$CONFIG.DAT` gearbeitet werden.
- Änderungen von `$TOOL` und `$BASE` im Interruptprogramm sind nur dort wirksam (Kommandobetrieb).
- Im Interruptprogramm gibt es keinen Rechnervorlauf, da es auf Kommandoebene läuft, d.h. es wird Satz für Satz abgearbeitet (`$ADVANCE`–Zuweisungen sind nicht zulässig). Ein Überschleifen ist somit nicht möglich.

Sonderfälle:

- Interrupts auf die Systemvariablen `$ALARM_STOP` und `$STOPMESS` werden auch im Fehlerfall bearbeitet, das heißt, trotz Roboterstop laufen die Interruptanweisungen ab (keine Bewegungen).
- Während eines Bedienstopps kann jeder deklarierte und aktivierte Interrupt erkannt werden. Nach einem erneuten Start werden die aufgetretenen Interrupts ihrer Priorität entsprechend abgearbeitet (falls sie freigegeben sind) und anschließend wird das Programm fortgesetzt.

Eine laufende Roboterbewegung wird beim Aufruf eines Interruptprogramms nicht abgebrochen. Während das Interruptprogramm bearbeitet wird, werden noch alle im unterbrochenen Programm aufbereiteten Bewegungen abgefahren. Wenn das Interruptprogramm während dieser Zeit komplett abgearbeitet ist, wird das unterbrochene Programm ohne Bewegungsstillstand, d.h. ohne Verlängerung der Bearbeitungszeit, fortgesetzt. Ist dagegen die Interruptaktion noch nicht beendet, bleibt der Roboter stehen bis nach dem Rücksprung die nächste Bewegung aufbereitet und fortgesetzt wird.

Befinden sich im Interruptprogramm selbst Bewegungsanweisungen, so hält das Interruptprogramm solange auf der ersten Bewegungsanweisung an bis der Vorlauf des Hauptprogramm abgearbeitet ist.

Den Einsatz von Interruptanweisungen und die Verwendung der speziellen Systemvariablen soll das folgende Beispiel erläutern. Hierin werden während einer Linearbewegung ständig zwei Sensoren (an den Eingängen 1 und 2) überwacht. Sobald ein Sensor einen Teil detektiert (auf High geht), wird ein Interruptunterprogramm aufgerufen, indem die Position des Teiles gespeichert wird und als Anzeige ein entsprechender Ausgang gesetzt wird. Die Roboterbewegung wird dabei nicht unterbrochen. Danach werden die erkannten Teile nochmals angefahren.

```
DEF  INTERRUP ( )

;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
INT I

;---------- Initialisierung ---------
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
FOR I=1 TO  16
   $OUT[I]=FALSE  ;alle Ausgaenge ruecksetzen
ENDFOR
INTERRUPT DECL 10 WHEN $IN[1]==TRUE DO SAVEPOS (1 )
INTERRUPT DECL 11 WHEN $IN[2]==TRUE DO SAVEPOS (2 )

;------------- Hauptteil ------------
PTP  HOME ;SAK-Fahrt

PTP  {X 1320,Y 100,Z 1000,A -13,B 78,C -102}

INTERRUPT  ON ;alle Interrupts aktivieren
LIN  {X 1320,Y 662,Z 1000,A -13,B 78,C -102} ;Suchstrecke
INTERRUPT  OFF 10 ;Interrupt 10 ausschalten
INTERRUPT  OFF 11 ;Interrupt 11 ausschalten

PTP  HOME

FOR I=1 TO  2
   IF $OUT[I] THEN
      LIN  TEIL[I] ; erkanntes Teil anfahren
      $OUT[I]=FALSE
      PTP  HOME
   ENDIF
ENDFOR

END

;----------- Interruptprogramm ----------
DEF  SAVEPOS (NR :IN )  ;Teil erkannt
INT NR
$OUT[NR]=TRUE           ;Merker setzen
TEIL[NR]=$POS_INT       ;Position speichern
END
```

Neben dem Basis–Paket (`BAS.SRC`) ist standardmäßig auch eine Datei `IR_STOPM()` auf der Steuerung vorhanden. Dieses Unterprogramm führt im Fehlerfall einige grundlegende Anweisungen aus. Dazu gehört neben einigen technologiespezifischen Aktionen auch das Rückpositionieren des Roboters auf die Bewegungsbahn. Während nämlich der Roboter nach Drücken des NOT–AUS–Schalters auf der Bahn bleibt, wird bei Schutzeinrichtungen, die direkt den Bediener betreffen (z.B. Schutztür), hardwareseitig ein nicht bahntreuer Stop ausgelöst.

Sie sollten daher im Initialisierungsteil Ihrer Programme stets die folgende Sequenz implementieren (befindet sich standardmäßig im Fold `BAS INI`):

```
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT  ON 3
```

In der Datei IR_STOPM() wird mit der Anweisung PTP $POS_RET das Rückpositionieren veranlaßt und somit wieder SAK hergestellt.

Weitere nützliche Systemvariablen für das Arbeiten mit Interrupts sind in Tab. 14 aufgeführt. Die Positionen beziehen sich immer auf die im Hauptlauf aktuellen Koordinatensysteme.

achsspezi-fisch	kartesisch	Beschreibung
$AXIS_INT	$POS_INT	Position, an der der Interrupt ausgelöst wurde
$AXIS_ACT	$POS_ACT	aktuelle IST–Position
$AXIS_RET	$POS_RET	Position, an der die Bahn verlassen wurde
$AXIS_BACK	$POS_BACK	Position des Startpunktes der Bahn
$AXIS_FOR	$POS_FOR	Position des Zielpunktes der Bahn

Tab. 14 Nützliche Systemvariablen bei der Interrupt–Behandlung

Die Positionen …_BACK und …_FOR sind bei Überschleifbewegungen davon abhängig wo sich der Hauptlauf befindet. Siehe dazu Abb. 38 bis Abb. 39.

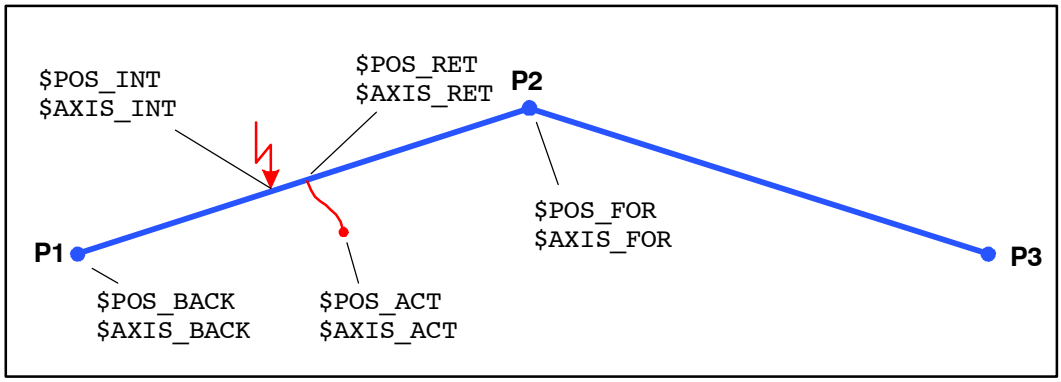


Abb. 38 Interrupt–Systemvariablen bei Genauhaltpunkten

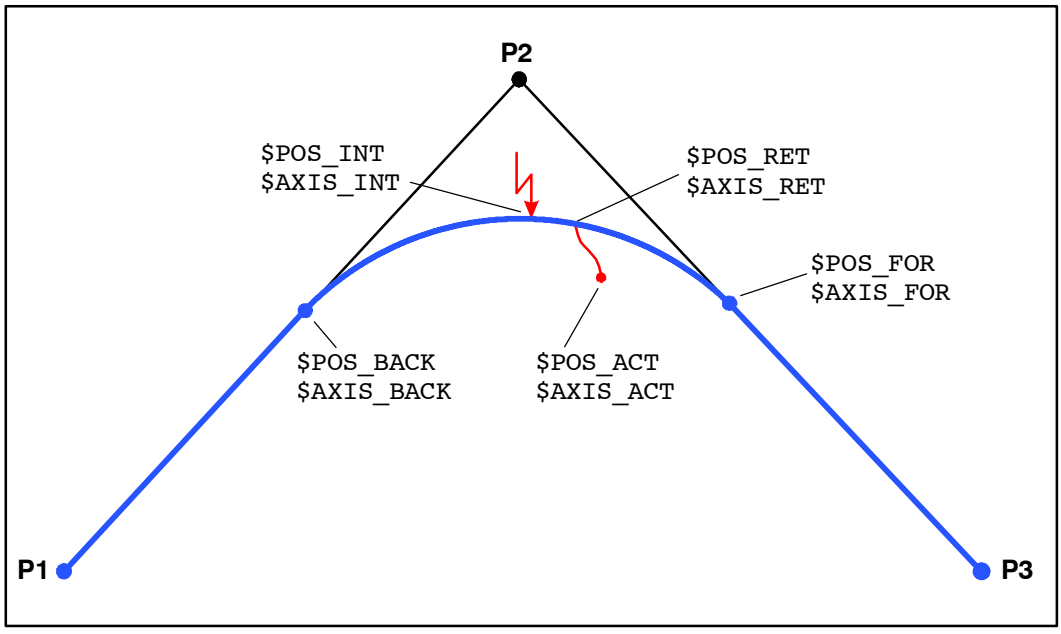


Abb. 39 Interrupt–Systemvariablen bei Interrupt in einem Überschleifbereich

9.3 Laufende Bewegungen anhalten

BRAKE

Sollen noch laufende Roboterbewegungen beim Eintreten eines Interrupts angehalten werden, so bedient man sich der BRAKE–Anweisung im Interruptprogramm. Programmiert man ohne Parameter

```
BRAKE
```

so bewirkt dies ein Abbremsen der Bewegung mit den programmierten Bahn– bzw. Achsbeschleunigungswerten. Das Verhalten ist das gleiche wie beim Betätigen der STOP–Taste. Die programmierte Bewegungsbahn wird dabei nicht verlassen.

Mit der Anweisung

```
BRAKE F
```

(brake fast) erreichen Sie ein kürzeren Bremsweg. Der Roboter wird dabei mit der maximalen bahntreuen Verzögerung abgebremst.

Die BRAKE – Anweisung darf nur in einem Interruptprogramm stehen. In anderen Programmen führt sie zu einem Fehlerstop.

Die BRAKE–Anweisung muß nicht direkt nach dem Aufruf, sondern kann an beliebiger Stelle im Interruptprogramm erfolgen. Ihre Wirkung hängt davon ab, ob zum Zeitpunkt ihrer Abarbeitung noch eine Bewegung des unterbrochenen Programms ausgeführt wird. Wenn der Roboter steht, hat die Anweisung keine Auswirkung. Eine noch laufende Bewegung des unterbrochenen Programms wird mit dem programmierten Bremsmodus angehalten. Der BRAKE–Befehl ersetzt allerdings nicht die HALT–Anweisung, wenn der Programmlauf angehalten werden soll. Die Bearbeitung des Interruptprogramms wird erst nach erfolgtem Stillstand des Roboters mit der darauffolgenden Anweisung fortgesetzt.

Nach dem Rücksprung in das unterbrochene Programm wird eine im Interruptprogramm mit `BRAKE` oder `BRAKE F` angehaltene Bewegung fortgesetzt!

NOTIZEN:

9.4 Abbrechen von Interrupt–Routinen

In Beispiel 8.1 werden während der Roboterbewegung durch 2 Initiatoren maximal 2 Gegenständen detektiert und deren Positionen zum späteren Anfahren aufgezeichnet.

Die Suchstrecke wird dabei auf jeden Fall komplett abgefahren, auch wenn die beiden Gegenstände schon erkannt sind. Um Zeit zu sparen, ist es wünschenswert die Bewegung sofort abzubrechen, wenn die maximale Anzahl von Teilen erkannt wurde.

RESUME

Das Abbrechen einer Roboterbewegung ist in der KR C1 mit der

RESUME

Anweisung möglich. RESUME bricht alle laufenden Interruptprogramme und auch alle laufenden Unterprogramme bis zu der Ebene ab, in der der aktuelle Interrupt deklariert wurde.

Wie die BRAKE–Anweisung, ist auch RESUME nur in einem Interruptprogramm zulässig.

Zum Zeitpunkt der RESUME–Anweisung darf der Vorlaufzeiger nicht in der Ebene stehen, in der der Interrupt deklariert wurde, sondern mindestens eine Ebene tiefer.

Da die Suchstrecke mit RESUME abgebrochen werden soll, muß die Suchbewegung in einem Unterprogramm programmiert werden. Dies geschieht im folgenden Beispiel in MOVEP(), das Interruptunterprogramm heißt IR_PROG().

Wichtig ist, daß in Unterprogrammen, die mit RESUME abgebrochen werden sollen, vor der letzten Zeile der Vorlauf angehalten wird. Nur dann ist gewährleistet, daß der Vorlaufzeiger bei RESUME nicht in der Ebene steht, in welcher der Interrupt deklariert wurde. In MOVEP() wurde dies mit der $ADVANCE=0 Zuweisung realisiert.

Im Interruptprogramm selbst wird – sobald 4 Teile durch einen Sensor an Eingang 1 erkannt sind – die Suchbewegung mittels BRAKE angehalten, und schließlich mit der RESUME–Anweisung abgebrochen (da neben IR_PROG() auch MOVEP() beendet wird). Ohne die BRAKE–Anweisung würde zunächst noch die Suchbewegung im Vorlauf abgearbeitet werden.

Nach RESUME wird das Hauptprogramm mit der Anweisung nach dem Unterprogrammaufruf, also $ADVANCE=3 (Vorlauf zurückstellen), fortgesetzt.

```
DEF  SEARCH ( )
;--------- Deklarationsteil ---------
EXT  BAS (BAS_COMMAND :IN,REAL :IN )
DECL AXIS HOME
;---------- Initialisierung ---------
INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
INTERRUPT  ON 3   ;standardmaessige Fehlerbehandlung
BAS (#INITMOV,0 ) ;Initialisierung von Geschwindigkeiten,
                  ;Beschleunigungen, $BASE, $TOOL, etc.
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
INTERRUPT DECL 11 WHEN $IN[1] DO IR_PROG ( )
I[1]=0            ;vordefinierten Zaehler auf 0 setzen
;------------- Hauptteil ------------
PTP  HOME ;SAK-Fahrt
INTERRUPT  ON 11
MOVEP ( )          ;Abfahren der Suchstrecke
$ADVANCE=3         ;Vorlauf zurueckstellen
INTERRUPT  OFF 11
GRIP ( )
PTP  HOME
END
;-------- Unterprogramm ------------
DEF  MOVEP ( ) ;Unterprogramm zum Abfahren der Suchstrecke
PTP  {X 1232,Y -263,Z 1000,A 0,B 67,C -90}
LIN  {X 1232,Y 608,Z 1000,A 0,B 67,C -90}
$ADVANCE=0     ;Vorlauf anhalten
END ;
;------ Interruptprogramm ---------
DEF  IR_PROG ( ) ;Teileposition speichern
;INTERRUPT  OFF 11
I[1]=I[1]+1
POSITION[I]=$POS_INT ;Abspeichern der Position
IF I[1]==4 THEN        ;4 Teile werden erkannt
   BRAKE               ;Anhalten der Bewegung
   RESUME              ;Abbrechen von IR_PROG & MOVE
ENDIF
;INTERRUPT  ON 11
END
;-------- Unterprogramm ------------

DEF  GRIP ( )         ;Greifen der erkannten Teile
INT POS_NR            ;Zaehlvariable
FOR POS_NR=I[1] TO  1  STEP  -1
POSITION[POS_NR].Z=POSITION[POS_NR].Z+200
LIN  POSITION[POS_NR] ;200mm ueber Teil fahren
LIN_REL  {Z -200}     ;Teil senkrecht anfahren
; Teil greifen
LIN  POSITION[POS_NR] ;wieder hoch fahren
LIN  {X 634,Y 1085,Z 1147,A 49,B 67,C -90}
; Teil ablegen
ENDFOR
END
```

Wenn die Gefahr besteht, daß ein Interrupt fälschlicherweise durch empfindliche Sensorik zweimal ausgelöst wird (”Tastenprellen”), so können Sie dies durch Ausschalten des Interrupts in der ersten Zeile des Interruptprogramms verhindern. Allerdings kann dann auch ein tatsächlich auftretender Interrupt während der Interruptverarbeitung nicht mehr erkannt werden. Vor dem Rücksprung muß der Interrupt – wenn er weiter aktiv sein soll – wieder eingeschaltet werden.

Falls, wie in obigem Beispiel, eine Bewegung mit `RESUME` abgebrochen wird, sollte die nachfolgende Bewegung keine `CIRC`–Bewegung sein, da der Anfangspunkt jedes mal ein anderer ist (unterschiedliche Kreise).

Bei der in Beispiel 9.2 programmierten Suchaktion werden die Eingänge im Interpolationstakt (z.Z. 12ms) abgefragt. Dabei besteht eine maximale Ungenauigkeit von ca. 12ms mal Bahngeschwindigkeit.

“Schnelles Messen”

Soll diese Ungenauigkeit vermieden werden, so darf man den Initiator nicht an die Anwendereingänge anschließen, sondern muß spezielle Eingänge (4 Stück) am Peripheriestecker X11 benutzen. Diese Eingänge können über die Systemvariablen `$MEAS_PULSE[1]...MEAS_PULSE[4]` angesprochen werden (Reaktionszeit 125µs).

Beim Einschalten des Interrupts darf der Meßimpuls nicht anliegen, sonst erscheint die entsprechende Fehlermeldung.

NOTIZEN:

9.5 Verwendung zyklischer Flags

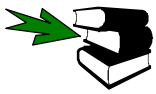
In der Anweisung zur Interrupt–Deklaration sind keine logischen Verknüpfungen zulässig.

Um kompliziertere Ereignisse definieren zu können, müssen Sie daher mit zyklischen Flags arbeiten, da nur diese eine stetige Aktualisierung ermöglichen.

Mit der Sequenz

```
:
$CYCFLAG[3] = $IN[1] AND ([$IN[2] OR $IN[3])
INTERRUPT DECL 10 WHEN $CYCFLAG[3] DO IR_PROG()
INTERRUPT ON 10
:
```

können Sie somit gleichzeitig 3 Eingänge überwachen und logisch miteinander verknüpfen.



Weitere Informationen finden Sie im Kapitel **[Variablen und Vereinbarungen]** Abschnitt **[Systemvariablen und Systemdateien]**.

NOTIZEN: