PROZESS-SPS

Programmier-Handbuch

Inhaltsverzeichnis

SYMPAS-Programmierumgebung

1. Überblick

Unterstützt alle Stationen der Programm-entwicklung

SYMPAS Programmierumgebung für PROZESS-SPS-Programme. Mit Hilfe dieser Programmiersoftware kann eine Problemstellung, die von seiten eines zu steuernden Prozesses ensteht, in ein SYMPAS-Programm für alle PROZESS-SPS-Steuerungen umgesetzt werden. Die SYMPAS Programmierumgebung unterstützt alle wichtigen Stationen der Programmentwicklung - von dem Editieren, über den Syntax-Check bis hin zur Übertragung in die Steuerung und die Inbetriebnahme im integrierten Setup-Mode.

Hardware-voraus-setzung: PC, IBM-kompatibel

Hardwarevoraussetzung für den Einsatz von SYMPAS ist ein IBM-kompatibler Personal-Computer. Der PC dient dabei sowohl der Dateneingabe, als auch zur Überwachung von Programmabläufen und Registerzuständen, während der Inbetriebnahmephase.

PROZESS-SPS-Programme und Registersätze können auf Diskette oder Festplatte gespeichert und rückgeladen werden.

Der Personal-Computer wird nur solange benötigt, bis ein Programm in die Steuerung übertragen (plus eventueller Datensätze) und erfolgreich ausgetestet ist. Danach kann der PC wieder anderweitig verwendet werden.

Menü- und Fenster-struktur

Die Pull-Down-Menü- und Fenster-Struktur von SYMPAS verbindet maximale Übersichtlichkeit mit Bedienungs-komfort. Um dem professionellen Anwender ein zügiges Arbeiten zu ermöglichen, ist zusätzlich der Zugriff auf die wichtigsten Funktionen über Hotkeys vorhanden.

Mit (F1) Hilfe aufrufen

Der ständig in der Statuszeile eingeblendete Hilfstext und die über die F1-Taste aktivierbaren Hilfsfenster bieten kontextbezogene Hilfestellung.

2. SYMPAS - System

2.1 Hardware (Voraussetzungen)

Voraussetzung für die Benutzung der SYMPAS-Programmierumgebung sind:

- ein IBM - kompatibler Personalcomputer mit mindestens 512 kByte RAM und 2 Diskettenlaufwerken (oder 1 Laufwerk und Festplatte) und DOS-Betriebssystem.
- eine serielle Schnittstelle (COM1 oder COM2).
- ein Programmierkabel vom PC zur Steuerung vom Typ EM-PK.
- eine PROZESS-SPS-Steuerung - PASE-E, DELTA, NANO oder MIKRO.

2.2 Software

Aktuelle Informationen in der Datei LESEMICH

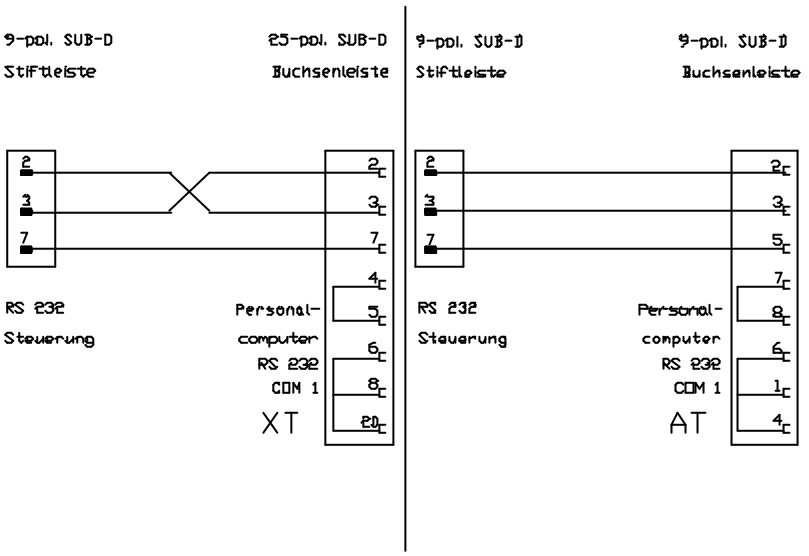
Eine Übersicht über die vorhandenen Dateien kann der Datei LESEMICH entnommen werden, die auf jeden Fall gelesen werden sollte. Denn sie enthält wichtige aktuelle Informationen, die nicht im Handbuch zu finden sind. Durch eingeben von TYPE A:LESEMICH wird diese Datei auf dem Bildschirm angezeigt, durch den DOS-Befehl PRINT A:LESEMICH wird die Datei ausgedruckt.

2.3 Hardware - Installation

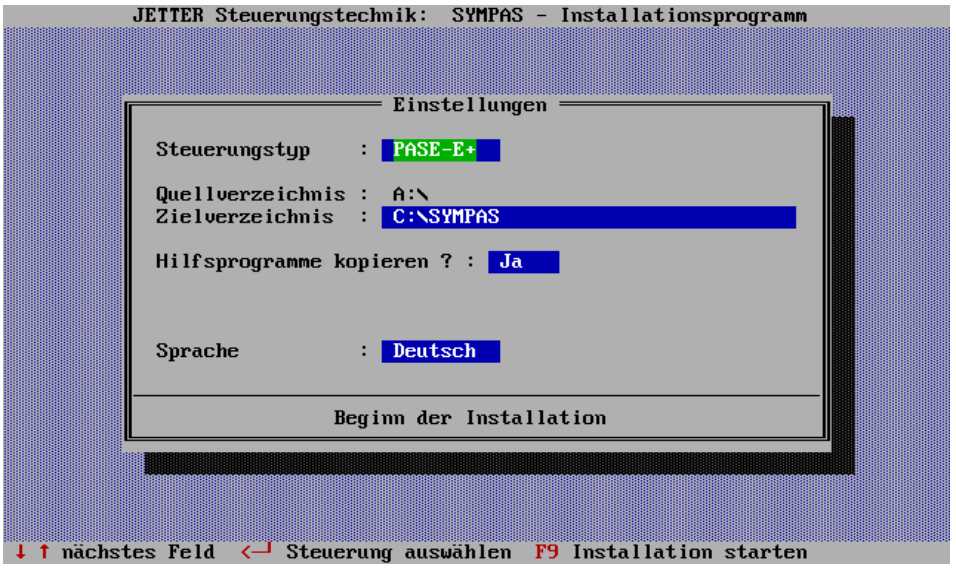
Um die SYMPAS-Programmierumgebung zusammen mit einer PROZESS-SPS zu benutzen, muss eine Verbindung mit der seriellen Schnittstelle (COM1 oder COM2) des PC hergestellt werden. Die Schnittstelle ist innerhalb der SYMPAS-Programmierumgebung konfigurierbar. Bei XT-kompatiblen Systemen hat COM1 einen 25-poligen SUB-D Stiftstecker, bei AT-kompatiblen Systemen hat COM1 einen 9-poligen SUB-D Stiftstecker. Es ist das Verbindungskabel EM-PK zu verwenden.

Programmier-kabel EM-PK

Das Programmierkabel EM-PK kann nach folgendem Muster auch selbst angefertigt werden.



2.4 Software-Installation



INSTALL.EXE installiert die SYMPAS-Programmier-umgebung

Die Software-Installation wird mit dem Programm INSTALL.EXE ausgeführt. Dieses Programm eröffnet auf der Festplatte oder einer Diskette ein Unterverzeichnis mit dem Namen SYMPAS (Defaulteinstellung). In dieses Verzeichnis werden alle relevanten Dateien kopiert. Welche Dateien und Unterverzeichnisse in das Unterverzeichnis SYMPAS kopiert werden, hängt von der Konfiguration ab, welche vor dem eigentlichen Installationsvorgang im oben abgebildeten Konfigurationsfenster vorgegeben werden kann. Zur Installation geben Sie die Zeile

`A:\INSTALL` oder `B:\INSTALL`

ein.

Mit den Cursor-Tasten ↓ und ↑ kann die Auswahlzeile aktiviert und nach dem Betätigen der RETURN-Taste <+ kann diese angewählte Zeile geändert werden. Nachdem Sie die Grundkonfiguration vorgegeben haben, wird mit der Funktionstaste F9 der Installationsvorgang gestartet. Folgende Einstellungen können in den Auswahlzeilen getroffen werden:

Steuerungstyp

Hier kann zwischen den Steuerungen PASE-E, DELTA, NANO und MIKRO gewählt werden. Unabhängig von der Installation kann der Steuerungstyp in der SYMPAS-Programmierumgebung jederzeit neu gewählt werden.

Zielverzeichnis

Hier kann der komplette Zielpfad angegeben werden. Soll die Programmierumgebung in einem anderen Unterverzeichnis, so ist diese Zeile entsprechend zu editieren. Zum Beispiel so:

```
C:\VERZEICHNIS
```

installiert SYMPAS in das Unterverzeichnis "Verzeichnis" auf das Laufwerk C.
Nachdem die Zeile mit der Cursor-Taste angewählt wurde, öffnet die ENTER-Taste <+ ein Fenster, in dem der Zielpfad editiert werden kann.

Hilfsprogramme kopieren ?

Hier kann festgelegt werden, ob die mitgelieferten Hilfsprogramme ebenfalls installiert werden sollen. Diese Hilfsprogramme sind in der Datei LESEMICH

dokumentiert, und können einem eigenen Verzeichnis zugeordnet werden.

Sprache

Hier kann definiert werden in welcher Sprache die Programmierumgebung bedient werden soll. Es kann zwischen deutsch und englisch gewählt werden. Auch nach der Installation kann jederzeit sowohl die Dialog- als auch die Programmsprache in SYMPAS selbst geändert werden.

Mit (F9) Installation starten

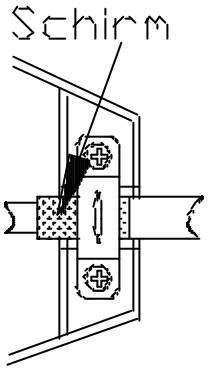
Mit der Funktionstaste F9 wird der Installationsvorgang entsprechend den Vorgaben gestartet und ausgeführt.

2.4 SYMPAS und mehrere vernetzte Steuerungen (JETWay-H)

JETWay-H: 126 Teilnehmer 115 kBaud

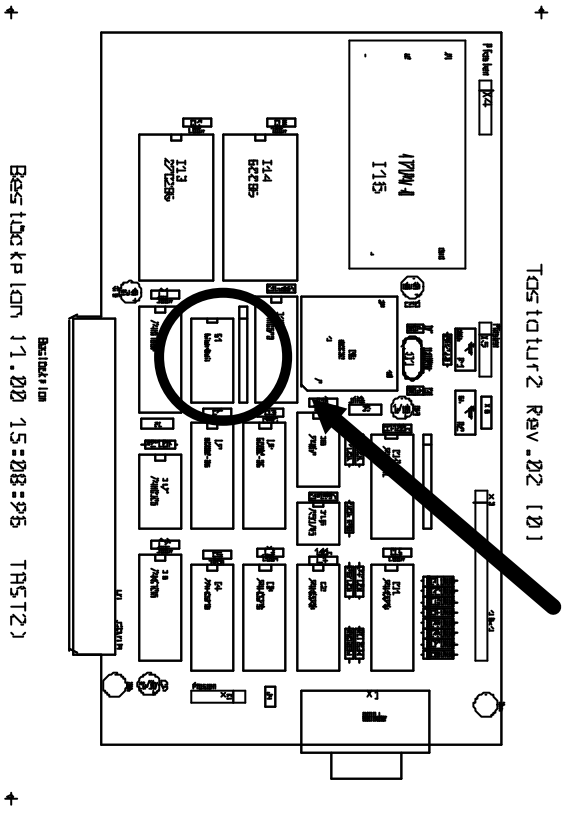
Folgende Vorteile bietet die Verwendung der JETWay-H-Schnittstelle als Programmierschnittstelle gegenüber der RS232-Schnittstelle:

- Es können bis zu 126 PROZESS-SPS von einem SYMPAS-Arbeitsplatz aus adressiert werden
- Es können Übertragungsraten bis zu 115kBaud realisiert werden.
- Größere Entfernungen möglich

JETWay-H-Kabel		
Verbindung auf Seite PROZESS-SPS	**Schirmung**	**Spezifikation max. Länge**
9 pol. Sub-D-Stecker oder 15 pol. Sub-D-Stecker	 **Schirm großflächig auflegen ! Metallisierte Gehäuse verwenden !**	RS485 max. Kabellänge: 400m
Pin	**Signal**	**Bemerkung**
7	Gnd	7
8	Daten +	8
9	Daten -	9

Die JETWay-H-Karte für den PC

Mit Hilfe der unten dargestellten Steckkarte für den PC kann die Verbindung zwischen SYMPAS und bis zu 126 PROZESS-SPS-Steuerungen über den JETWay-H verwirklicht werden.



Hier sind die DIL-Schalter zum definieren der Portadresse. Default ist 340h.

Abbildung 1: JETWay-H-Steckkarte für den PC

AUTOEXEC.BAT

In der AUTOEXEC.BAT Ihres PC ist folgende Zeile einzutragen (vorausgesetzt, Sie verwenden die Defaulteinstellung):

SET JETWAY_PORT=340h

DIL-SCHALTER

Sollten Sie eine abweichende Portadresse wählen wollen oder müssen, ist dies mit den oben dargestellten DIL-Schaltern auf der JETWay-H-Karte möglich.

DIL-Schalter auf der JETWay-H-Karte						
Port	Schalter 2	Schalter 3	Schalter 4	Schalter 5	Schalter 6	Schalter 7
300h	OFF	OFF	ON	ON	ON	ON
310h	OFF	OFF	ON	ON	ON	OFF
320h	OFF	OFF	ON	ON	OFF	ON
330h	OFF	OFF	ON	ON	OFF	OFF
340h*)	OFF	OFF	ON	OFF	ON	ON
350h	OFF	OFF	ON	OFF	ON	OFF
360h	OFF	OFF	ON	OFF	OFF	ON
*) Defaulteinstellung						

Folgende Zeile muss in der AUTOEXEC.BAT eingefügt werden

Entsprechend ist die Zeile in der AUTOEXEC.BAT zu ändern:

```
SET  JETWAY_PORT=x
```

Im SYMPAS-Menü "Spezial / Einstellungen" kann zwischen der Programmierschnittstelle über RS232 und über JETWay-H gewählt werden.

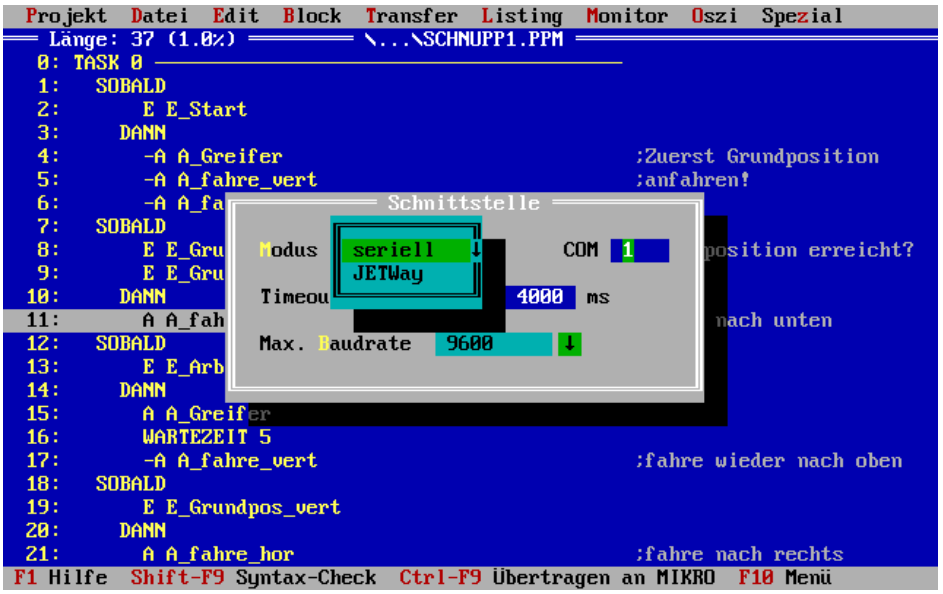


Abbildung 2: SYMPAS-Menü: Spezial / Schnittstelle

Hinweis:

Für das Fertigen dieses Kabels gelten folgende Mindestanforderungen:

Adernzahl:	**3**
Querschnitt:	**0,25²**
Stecker:	**SUB-D, metallisiert**
Schirmung:	**gesamt, nicht paarig**

Der Schirm muss auf beiden Seiten einen großflächigen Kontakt zu den Steckergehäusen haben.

3. Bedienung der SYMPAS-Programmierumgebung

3.1. Starten der SYMPAS-Programmierumgebung

Nachdem die Software, der Anleitung in *Kapitel 2.4 Software-Installation* folgend, installiert wurde, kann die Programmierumgebung gestartet werden. Die Eingabe von SYMPAS startet die Programmierumgebung.

C:\SYMPAS>**SYMPAS**

startet die Programmierumgebung SYMPAS.

Sinnvoll ist es im Unterverzeichnis SYMPAS ein weiteres Unterverzeichnis anzulegen z.B. PROJEKT1:

Sinnvolle Unterverzeichnisstruktur wählen

C:\SYMPAS**PROJEKT1**>

Außerdem ist in der AUTOEXEC.BAT die entsprechende Pfadangabe für den Aufruf von SYMPAS einzutragen.
Jetzt kann SYMPAS zum Beispiel folgendermaßen gestartet werden:

SYMPAS starten

C:\SYMPAS\PROJEKT1>**SYMPAS**

Alle Informationen, Dateien, etc., welche einen Bezug zu "Projekt1" haben werden nun in diesem Unterverzeichnis abgelegt. Auf diese Weise bleibt der Überblick - auch bei einer Vielzahl von Projekten - erhalten.

3.2 Beschreibung der Bildschirme

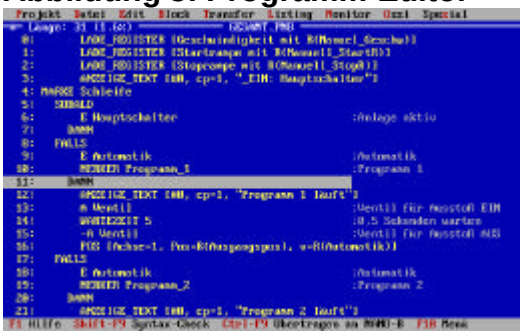
Mit (F4) zwischen Programm- und Symbol-editor wechseln

Nach dem Starten von SYMPAS sehen Sie folgenden Bildschirm vor sich. Der Programm-Editor dient zum Erstellen der PROZESS-SPS-Programme. Neben dem Programm-Editor existieren noch zwei weitere Bildschirme.
Der Symbol-Editor dient zum Erstellen der Symbolik innerhalb eines Programmes. Mit der Funktionstaste F4 schalten Sie zwischen Programm- und Symbol-Editor hin und her.

Mit (F7) und (F4) zwischen Inbetrieb-nahmeschirm und Editoren wechseln

Der Inbetriebnahme-Bildschirm unterstützt die Inbetriebnahme des Programmes, der Steuerung ansich und des gesteuerten Prozesses. Er wird mit der Funktionstaste F7 aktiviert. Mit F4 schalten Sie zum Programm-Editor bzw. zum Symbol-Editor.

Abbildung 3: Programm-Editor



(F4) (F4) (F7) (F4)

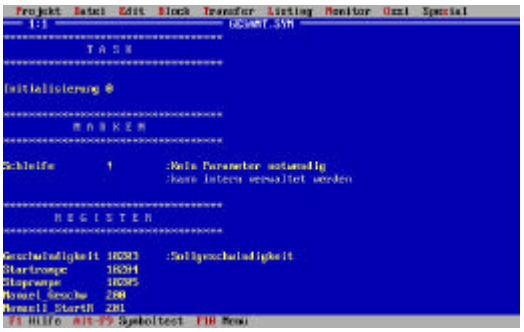


Abbildung 5: Symbol-Editor

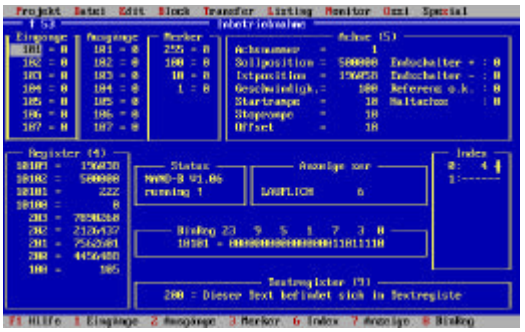


Abbildung 4 : Inbetriebnahme

Diese drei Bildschirme bilden die globale Struktur der Bedieneroberfläche von SYMPAS.
In der oberen Zeile des jeweiligen Bildschirmes befindet sich die Menüleiste. Diese Menüleiste ist für alle drei Bildschirme identisch, einige Funktionen beziehen sich aber auf einen bestimmten Bildschirm (z.B Inbetriebnahme) und haben für die anderen beiden keine Bedeutung (helle Darstellung der jeweiligen Auswahlzeilen).

Mit (ALT) und dem hervorgehobenen Buchstaben Menü oder Funktion aufrufen

Die Menüleiste wird mit der Funktionstaste F10 aktiviert. Mit den Cursortasten ← und → können Sie sich in der Menüleiste bewegen. Mit der Cursortaste ↓ öffnen Sie das entsprechende Pull-Down-Menü. Hier bewegen Sie sich mit den Cursortasten ↓ und ↑. Eine angewählte Auswahlzeile innerhalb eines Pull-Down-Menüs aktivieren Sie mit der Taste ENTER <+.

Verlassen können Sie das Pull-Down-Menü oder die Menüleiste mit der ESC-Taste. Mit der ESC-Taste können auch alle anderen aktivierten Funktionen abgebrochen werden.

Alle Funktionen, die von den drei Bildschirmen aus aufgerufen werden können, werden durch Hilfstexte unterstützt. Erstens befindet sich stets ein kontextbezogener Hilfstext in der Statuszeile, der letzten Zeile des Bildschirmes. Umfangreiche Hilfsinformation erscheint nach Betätigen der Funktionstaste F1 in einem Fenster. Auch diese Informationen sind kontextbezogen. Verlassen werden die Hilfsfenster mit der ESC-Taste.

Mit dem Programm-Editor werden die Programme für PROZESS-SPS-Steuerungen geschrieben. Mit dem Symbolik-Editor wird die Symbolik eines Programmes definiert. Mit dem Inbetriebnahme-Bildschirm schließlich wird das Programm in Verbindung mit der Steuerung und dem zu steuernden Prozess getestet und optimiert.

3.3 Programmeingabe

Befehls-eingabe über Kürzel

Die eigentlichen Befehle der Programmiersprache SYMPAS werden über die zwei Anfangsbuchstaben des Befehles in den Programm-Editor eingegeben. Durch Drücken der Taste "T" erscheint ein Auswahlfenster in dem alle mit dem Buchstaben "T" beginnenden SYMPAS-Befehle aufgeführt sind. Jetzt kann entweder durch die Cursortasten oder durch "A", den zweiten Buchstaben des Wortes TASK, der TASK-Befehl aktiviert werden. Nachdem die Parameternummer in einem Fenster eingegeben wurde, erscheint der Befehl auf dem Programm-Editor-Bildschirm. Nach diesem Schema wird ein Programm erstellt.
Die Taste "?" lässt ein Fenster erscheinen, in welchem alle zur Verfügung stehenden Befehle aufgelistet sind und von dort auch gewählt werden können.

Exemplarische Erstellung eines NANO-Programmes

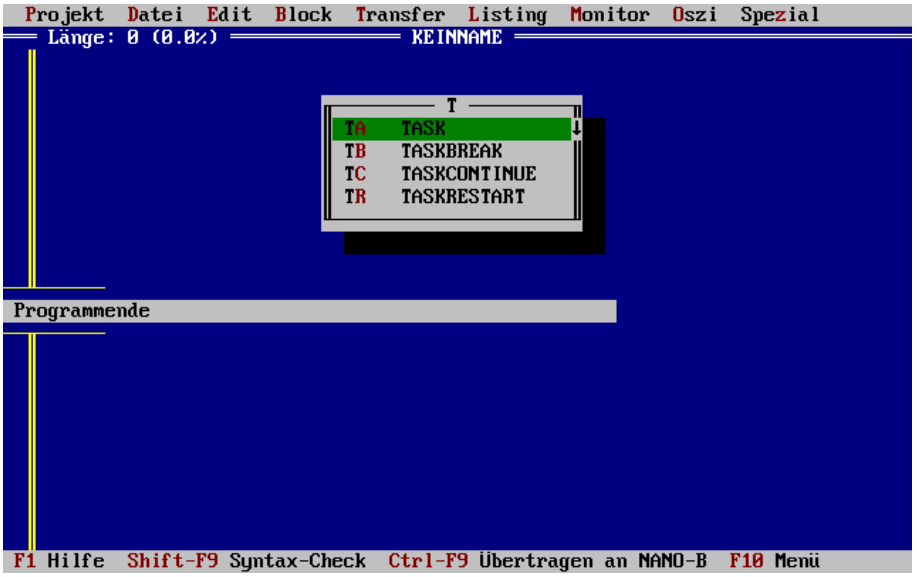
Nun soll exemplarisch ein Beispielprogramm (NANO) erstellt werden.

Nachdem Sie SYMPAS gestartet haben (*Kapitel 3.1. Starten der SYMPAS-Programmierumgebung*) erscheint der Programm-Editor. Als erstes öffnen Sie mit der Tastenkombination ALT-P das Pull-Down-Menü "Projekt". Mit der Cursortaste ↓ wird die Zeile "Projektdaten eingeben" angewählt und mit ENTER <+ aktiviert. SYMPAS öffnet nun ein Fenster in dem sie die globalen Projektdaten festlegen können. Geben Sie hier unter den Menüpunkten, "Programmname", "Kunde/Projekt", "Ort", "Version", "Symboldatei" die entsprechenden Informationen ein. Nachdem Sie die Angaben der letzten Zeile "Symboldatei" mit ENTER <+ quittiert haben, wird das Fenster geschlossen. Jetzt kann auf dem Programm-Editor Bildschirm mit der Eingabe eines Programmes begonnen werden. Hier gilt das oben bereits kurz vorgestellte Schema der Eingabe des Kürzels eines Befehles, um diesen in den Programmtext einzufügen.

Erster Befehl:

- Drücken der Taste "T" -> es erscheint ein Eingabefenster aller Befehle die mit dem Buchstaben "T" beginnen.

(T) öffnet ein Fenster aller Befehle, die mit "T" beginnen

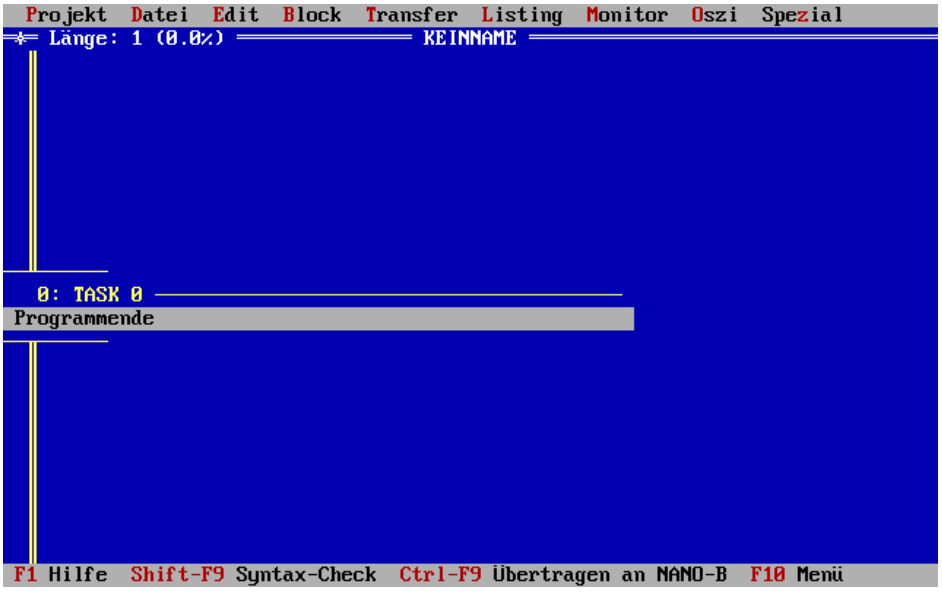


- Drücken der Taste "A" -> es erscheint ein Eingabefenster in dem die gewünschte Tasknummer festgelegt wird.

(A) öffnet ein Fenster in dem die Tasknummer spezifiziert wird

- drücken Sie jetzt die Taste "0" (null) und quittieren mit ENTER <+ -> der Befehl TASK0 erscheint auf dem Bildschirm.

Der TASK 0-Befehl erscheint auf dem Bildschirm



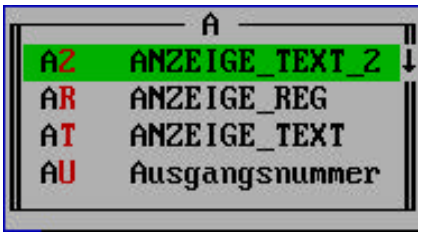
Hinweis:

Jedes Programm **muss** mit dem Befehl TASK 0 beginnen.

Zweiter Befehl:

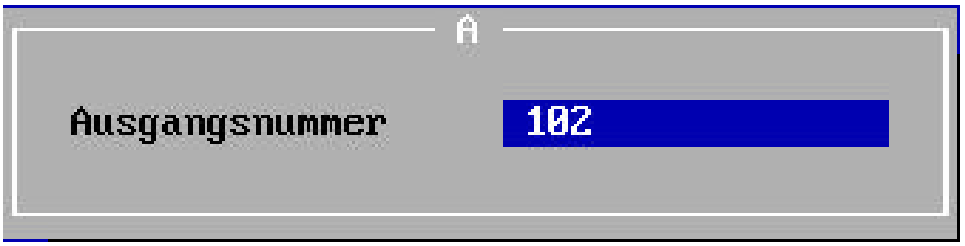
- Drücken der Taste "A" -> es erscheint ein Eingabefenster in dem alle Befehle aufgeführt werden die mit dem Buchstaben "A" beginnen.

(A) öffnet ein Fenster aller Befehle, die mit "A" beginnen



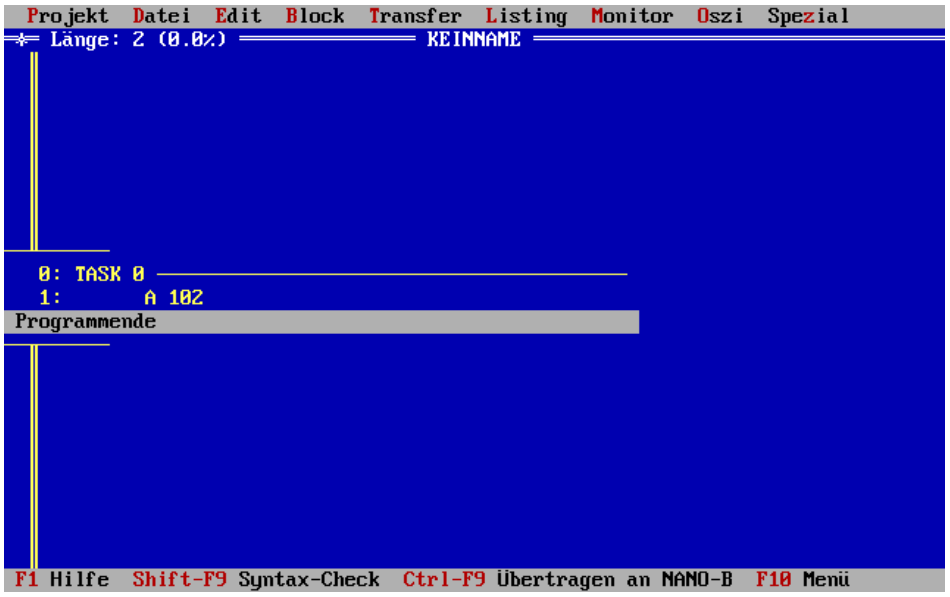
- Drücken der Taste "U" -> es erscheint ein Eingabefenster in den Sie den Ausgangsparameter eingeben.

(U) öffnet ein Fenster in dem der Ausgangs-parameter angegeben wird



- drücken Sie jetzt die Tasten "102" und quittieren mit ENTER <+ -> der A102-Befehl (Ausgang 102) erscheint auf dem Bildschirm. Dieser Befehl bewirkt, dass der Ausgang zwei gesetzt, aktiviert wird.

Der Befehl A 102 erscheint auf dem Bildschirm



Dritter Befehl:

- Drücken der Tasten "W" und "A" -> es erscheint ein Eingabefenster in dem die gewünschte Wartezeit in Vielfachen von 100ms eingegeben werden kann.

- geben Sie jetzt "10" ein und quittieren mit ENTER <+ -> Der Wartezeitbefehl mit dem Parameter "10" erscheint auf dem Bildschirm. Dieser Befehl bewirkt, dass die Steuerung 1 Sekunde wartet und dann mit der weiteren Ausführung des Programmes fortfährt.

Vierter Befehl:

- Drücken der Tasten "A" und "U" -> es erscheint ein Eingabefenster in dem die gewünschte Ausgangsnummer festgelegt wird.

- diesmal geben Sie "-102" ein und quittieren mit ENTER <+ -> der -A102-Befehl (-Ausgang102) erscheint auf dem Bildschirm. Dieser Befehl bewirkt, dass der Ausgang 102 zurückgesetzt wird.

Fünfter Befehl:

- drücken der Tasten "W" und "A" -> es erscheint ein Eingabefenster in dem die Wartezeit definiert werden kann.

- bestätigen Sie den Defaultwert aus der letzten Befehlseingabe "10" -> Dieser Befehl führt wiederum dazu, dass die Steuerung 1 Sekunde wartet.

Sechster Befehl:

- Drücken der Tasten "S" und "P" -> es erscheint ein Eingabefenster in dem das Sprungziel (Task oder Marke) festgelegt werden kann, zu welcher das Programm verzweigt.

- geben Sie "0" ein und quittieren mit ENTER <+ -> es erscheint der Befehl SPRUNG0 auf dem Bildschirm. Der Programmablauf schließt sich auf TASK0, das Programm bildet somit eine Endlosschleife.

Hinweis:

Jeder Task eines Programmes **muss** mit einem SPRUNG-Befehl in sich geschlossen werden.

Jetzt ist die Eingabe der Programmtextes abgeschlossen. Mit den Cursortasten ↑ und ↓ kann man sich durch den Programmtext bewegen. Die Delete-Taste löscht Programmzeilen. Eingefügt wird automatisch über die aktuelle Zeile, auf der der Cursor steht. Mit den Cursortasten ← und → kann zwischen den Bereichen für Programmtext und Kommentar gewechselt werden.

3.3.1 Tasten und Funktionen im Programm-Editor

Cursor-Bewegungen:

Taste:	**Funktion:**
Cursor hoch	Eine Zeile zurück
Cursor runter	Eine Zeile vor
Seite hoch	Seite zurück
Seite runter	Seite vor
Ctrl-Seite hoch	Zum Programmanfang
Ctrl-Seite runter	Zum Programmende
Cursor links	Befehls-Ebene
Cursor rechts	Kommentar-Ebene

Editor-Befehle:

Taste:	**Funktion:**
A..Z	Ein Befehl wird direkt aktiviert, wenn sein Anfangsbuchstabe nur einmal in der Befehlsliste vorkommt. Ansonsten wird ein Auswahlfenster angeboten, aus dem heraus der gewünschte Befehl entnommen werden kann. Außerdem existiert die Möglichkeit über zwei Anfangsbuchstaben eines Befehles diesen in den Programmtext einzufügen. Der Befehl wird vor der aktuellen Cursorposition eingetragen.
?	Eine Liste des kompletten Befehlsvorrates wird in Form eines Auswahlfensters angeboten.
SPACE	Der letzte eingetragene Befehl wird wiederholt.

ENTER	Parameter des aktuellen Befehls editieren.
BS(←)	Befehl vor aktueller Befehlszeile löschen.
DEL	Aktuellen Befehl löschen.

Block-Operationen:

Taste:	**Funktion:**
Ctrl-K B	Blockanfang markieren
Ctrl-K K	Blockende markieren
Ctrl-K V	Block verschieben
Ctrl-K C	Block kopieren
Ctrl-K Y	Block löschen
Ctrl-K R	Block von Disk lesen
Ctrl-K W	Block auf Disk schreiben
Ctrl-K P	Block drucken
Ctrl-K H	Block ausschalten
Ctrl-K L	Zeile markieren
Ctrl-Q B	Blockanfang suchen
Ctrl-Q K	Blockende suchen

Programm-Marken:

Taste:	**Funktion:**
Ctrl-K 0..9	Cursorposition 0 bis 9 im Programmtext merken.
Ctrl-Q 0..9	Cursorposition 0 bis 9 im Programmtext anspringen.

Sonstiges:

Taste:	**Funktion:**
Ctrl-S	zeigt Symolparameter der aktuellen Zeile solange an wie die Ctrl-Taste gedrückt bleibt.
Ctrl-M	zeigt Variableninhalte solange an wie die Ctrl-Taste gedrückt bleibt.

3.3.2 Programm übertragen

Mit (F2) das Programm speichern

Bevor das Programm in der Steuerung zum Laufen gebracht werden soll bietet sich ein Sichern, zum Beispiel auf Festplatte, an. Mit der Tastenkombination ALT-D öffnen Sie das Pull-Down-Menü "Datei". Mit der Cursortaste ↓ begeben sie sich auf die Auswahlzeile "Speichern". Mit der Taste ENTER <+ starten sie den Vorgang "Speichern" (kürzer über Hotkey F2). Speichern wird zu "Speichern unter...", wenn vorher kein Dateiname definiert wurde. Um ein Programm von Festplatte in den Programm-Editor zu laden, verwendet man die Auswahlzeile "Öffnen..." in dem selben Pull-Down-Menü.

Mit (CTRL) (F9) das Programm an die Steuerung übetragen

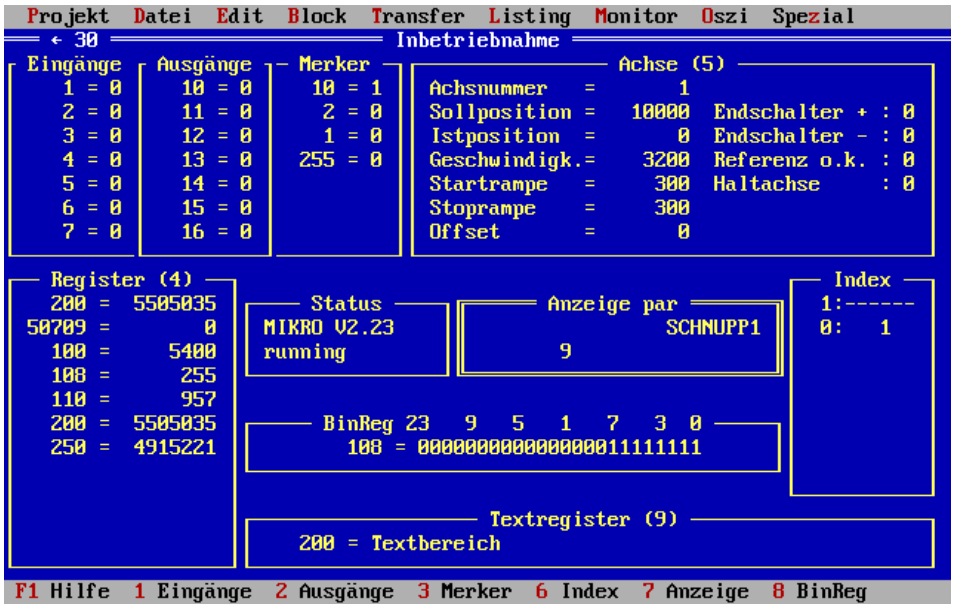
Nachdem ein Programm eingegeben wurde, kann es mit der Tastenkombination CTRL-F9 an die Steuerung übertragen werden. Nach dem Drücken der CTRL-F9 Tasten geschehen drei Dinge. Erstens wird das Programm an die Steuerung übertragen und dort gestartet. Zweitens wird der Bildschirm auf "Inbetriebnahme" umgeschaltet. Und drittens ertönt als Startzeichen ein akustisches Signal.

Sie haben seit *Kapitel 3. Bedienung der SYMPAS-Programmierumgebung* ein PROZESS-SPS-Programm in dem Programm-Editor erstellt, dieses auf Festplatte gesichert, an die Steuerung übertragen und dort automatisch gestartet. Die LED am Ausgang 102 wird nun als Blinklicht arbeiten mit gleichen Signalzeiten von jeweils einer Sekunde.

Mit der Funktionstaste F4 kommen Sie wieder in den Programmeditor, zum Beispiel um hier Änderungen am Programm vorzunehmen. Oder Sie bleiben weiterhin im Inbetriebnahme-Bildschirm, um die richtige Funktion eines Programmes zu überprüfen. Sicherlich wird ein solches zu überprüfendes Programm wesentlich komplexer sein als das eben

programmierte Blinklicht. In einem solchen Fall unterstützt der Inbetriebnahme-Modus von SYMPAS wirkungsvoll das Austesten des Programmes im Zusammenspiel mit der Steuerung und dem zu steuernden Prozess.

3.4 Der Inbetriebnahme-Bildschirm (Setup-Mode)



Der Inbetriebnahme-Bildschirm bietet eine Vielzahl von Funktionen, welche die Inbetriebnahme eines Programmes in Verbindung mit der Steuerung und dem zu steuernden Prozess unterstützen.

Mit (F1) Hilfe aufrufen

Auch hier erhalten Sie in der Statuszeile unmittelbare, und über das Hilfsfenster (Aktivierung mit Funktionstaste F1) umfangreiche, kontextbezogene Hilfestellung.

Im Inbetriebnahme-Modus können Eingänge, Ausgänge, Registerinhalte (als Zahl oder Text), Achsparameter angezeigt und geändert werden. Desweiteren können der Inhalt der Bediengeräte (LCD9, LCD16, etc.) sowie die gerade bearbeitete Zeilennummer des Programmes angezeigt werden.

3.4.1 Tasten und Funktionen im Inbetriebnahme-Bildschirm

Die Feldkennzahl zur Aktivierung der einzelnen Felder kann entweder der Statuszeile oder den Klammern hinter der Feldüberschrift entnommen werden.

Innerhalb eines aktiven Feldes gelten folgende Kommandos:

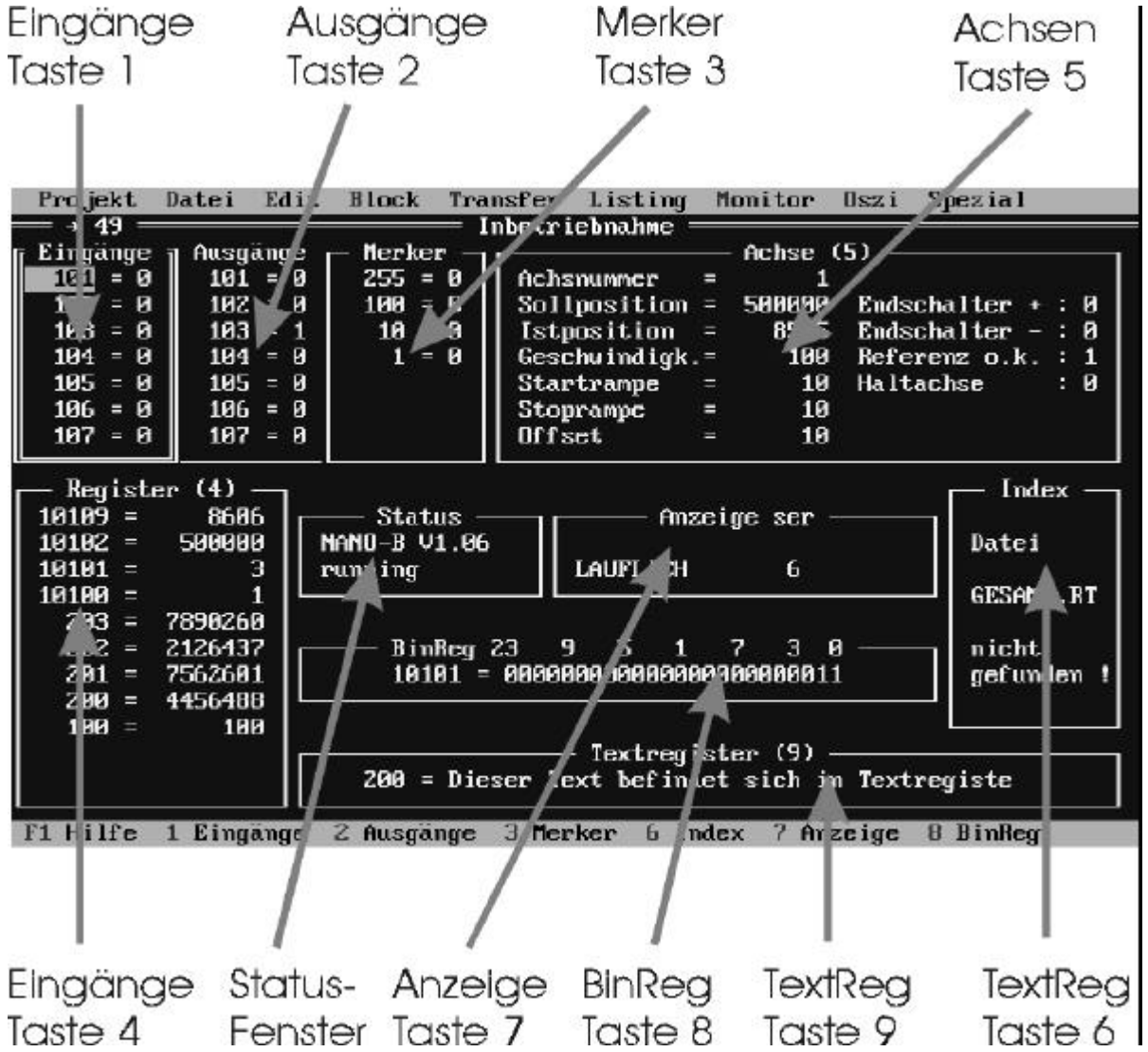
Taste:	**Funktion:**
INS	Einfügen einer neuen Zeile
DEL	Entfernen des aktuellen Eintrages
ENTER	o Eingabe eines neuen Wertes o Eingabe einer neuen Nummer
Cursor hoch	Ein Feldeintrag zurück
Cursor runter	Ein Feldeintrag vor
Cursor links, Cursor rechts	Umschalten zwischen Nummern- und Werteingabe (nur Ausgänge, Merker, Register, Binreg, Textregister)
Ctrl-Cursor	Anwahl der Bits 0 bis 23 im "Binreg"-Feld.
+	Inkrementieren des Cursorwertes
-	Dekrementieren des Cursorwertes

3.4.2 Beschreibung der Fenster

Zwischen den einzelnen Fenstern wird mit den Feldkenn-zahlen gewechselt

Zwischen den einzelnen Fenstern des Inbetriebnahme-Bildschirmes wird mit Hilfe der Feldkennzahlen gewechselt. So aktiviert die Taste "1" das Fenster für die Eingänge, die Taste "2" das Fenster für Ausgänge, "3" die Merker, "4" die Register, "6" das Indexfenster, "7" das Fenster für die Darstellung des Anzeigeninhaltes, "8" das Binärregister, in welchem der Inhalt eines beliebigen Registers binär dargestellt werden kann, und schließlich "9" das Textregister.

Folgende Funktionsfenster stehen zur Verfügung:



Eingangs-fenster

Taste (1)

- Drücken der Taste "1" -> jetzt ist das Eingangsfenster doppelt umrandet, und somit aktiv.
  Drücken der Taste "Insert" -> es erscheint ein Fenster, in welchem Sie die Nummer des darzustellenden Einganges festlegen und mit der Taste ENTER <+ quittieren.

Dasselbe gilt für Ausgänge:

Ausgangs-fenster

Taste (2)

- Drücken der Taste "2" -> jetzt ist das Ausgangsfenster doppelt umrandet, also aktiv.
  Drücken der Taste "Insert" -> es erscheint ein Fenster, in welchem Sie die Nummer des darzustellenden Ausganges festlegen und mit der Taste ENTER <+ quittieren. Diesen Vorgang können Sie solange fortsetzen bis das Fenster mit Ausgängen aufgefüllt ist. Mit den Cursortasten ↑ und ↓ kann man sich von Ausgang zu Ausgang bewegen, mit den Cursortasten ← und → kann man zwischen Ausgangsnummer und Ausgangsstatus hin und her schalten. Den Ausgang können Sie mit den Tasten "+" und "-" setzen oder rücksetzen. Ebenso können Sie mit den "+" und "-" Tasten die Ausgangsnummer de- oder inkrementieren
  Mit der "Delete" Taste löschen Sie die Darstellung des Ausganges auf dem sich der Cursor befindet. Mit der "Insert" Taste können Sie einen Ausgang über der aktuellen Cursorposition einfügen.

Dasselbe gilt für Merker.

Register-fenster

Taste (4)

- der Unterschied bei der Bedienung der Register in Fenster "4" besteht darin, dass dem Register ein Wert zugewiesen werden kann. Mit den Cursortasten ← und → kann zwischen der Registernummer und dem Registerinhalt gewechselt werden. Steht der Cursor auf dem Registerinhalt und Sie drücken ENTER <+ erscheint ein Fenster, in welchem der Registerinhalt geändert werden kann. Nachdem Sie den neuen Registerwert eingegeben haben quittieren Sie mit ENTER <+.

Achs-fenster

Taste (5)

- nachdem Sie das Achsfenster mit der Taste "5" geöffnet haben, betätigen Sie die Taste "Insert". Es erscheint ein Fenster, in welches die gewünschte Achsnummer eingegeben werden kann (quittieren mit ENTER <+). Danach werden alle Parameter in diesem Fenster entsprechend

dem Achszustand gesetzt. Mit den Cursortasten kann man zwischen den Parametern wechseln, durch drücken von ENTER <+ kann man in einem Fenster die Parameter editieren (mit ENTER <+ quittieren).

Index-fenster

Taste (6)

- in dem Indexfenster werden die Zustände der einzelnen Tasks des Programmes dargestellt. Drücken Sie die Taste "Insert" und geben Sie die Nummer des Tasks ein, dessen Darstellung gewünscht wird. Diesen Vorgang wiederholen Sie solange, bis alle relevanten Tasks sich im Fenster befinden.
  Die Darstellung der einzelnen Tasks folgt folgendem Schema:
  - die Tasknummer wie vom Anwender vorgegeben
  - :
  - Zeilennummer im Task, der gerade bearbeitet wird
  - falls zutreffend eine Statusbeschreibung des Tasks mit folgenden vier Zeichen:
    - o "+" Wartezeit; im Programm eingegebene Wartezeit.
    - o "i" Input; Programm wartet auf Bedienereingabe.
    - o "M" SOBALD_MAX
    - o "¦" Taskbreak; der Parallelzweig ist momentan unterbrochen.
    - o "----" Fehler; der angegebene Task existiert im Programm nicht.
    - o "Err" ungültige Programmzeile

Anmerkung:

Das Indexfeld funktioniert nur, wenn seit der Übertragung des Programmes SYMPAS nicht verlassen wurde; sonst wird "-1" angezeigt.

Anzeige fenster

Taste (7)

- um in das Anzeigefenster zu kommen drücken Sie die Taste "7". In diesem Fenster wird dargestellt, welchen Inhalt das angeschlossene Bediengerät (z.B. LCD9/10) augenblicklich anzeigt.

Binreg-fenster

Taste (8)

- mit der Taste "8" in das Binreg-Fenster wechseln. Dieses Fenster stellt den Inhalt eines Registers in binärer Form dar. Mit der Tastenkombination CTRL und einer der beiden Cursortasten ← und → kann ein einzelnes Bit angewählt und modifiziert (+ und -) werden. Zum Beispiel können Sie die Darstellung des Statusregisters 10100 des Slavemoduls SV1 wählen, indem Sie die "Insert" Taste betätigen, danach die gewünschte Registernummer eingeben und mit ENTER <+ quittieren.

Textregister-fenster

Taste (9)

- mit der Taste "9" wird das Textregister-Fenster aktiviert. Nach dem Drücken von "Insert" wird nach einer Registernummer gefragt. Geben Sie zum Beispiel das Register 200 ein. Mit den Cursortasten ← und → können Sie zwischen der Registernummer und dem entsprechenden Eingabetext wechseln. Befinden Sie sich mit dem Cursor auf dem Eingabetext, drücken Sie ENTER <+ um den Text zu editieren.
  Mit dieser Funktion können zum Beispiel Dialogtexte für das VIADUKT geschrieben werden (maximale Länge 40 Zeichen). Der Text wird ab dem Register 200 abgelegt. In den Bits 0 bis 7 wird eine Information über die Länge des Textes, in den Bits 8 bis 15 werden Statusinformationen, dann jedes Zeichen im ASCII-Format abgelegt (drei Zeichen pro Register).

Wenn Sie nach dem Starten des Beispielprogrammes aus *Kapitel 3.3 Programmeingabe* auf dem Inbetriebnahme-Bildschirm den Ausgang 102 darstellen, können Sie verfolgen, wie sich nach jeder

Sekunde der Status ändert. Bei komplexen Prozessen gelingt es mit diesem Hilfsmittel, kompakt und übersichtlich, den Status einer Vielzahl von Funktionen darzustellen, zu überwachen und auch zu modifizieren. Es können die Zustände von Achsen, Eingängen, Ausgängen, Merkern, Registern, Anzeigen etc. nebeneinander veranschaulicht werden.

- auf dem Inbetriebnahme-Bildschirm befindet sich unter dem Menüpunkt "Projekt" eine Statusanzeige zur allgemeinen Funktion des Schirmes.

  ®

  Der rotierende Pfeil zeigt an, dass der Inbetriebnahme-Schirm arbeitet. Sollten sich die dargestellten Daten nicht verändern, kann man hiermit sicherstellen, dass es sich um statische Zustände der einzelnen Eingänge, Ausgänge etc. handelt und um keine Funktionsfehler des Inbetriebnahme-Modus.

  Zahl

  Die Zahl hinter dem rotierenden Pfeil gibt die Zeit (in 1/100 Sekunden) an, die ein Refresh-Zyklus dauert. Das ist die Zeit, welche vergeht bis der Status aller Eingänge, Ausgänge, Merker, Register etc. in der Darstellung aktualisiert wurde.

3.5 Beschreibung der Menüs

Hier werden die einzelnen Pull-Down-Menüs, welche aus der Menüleiste heraus aktiviert werden können, beschrieben. Die Beschreibung wird in der Reihenfolge gegeben, in welcher die einzelnen Funktionen in den Pull-Down-Menüs der drei Bildschirme aufgeführt sind. Prinzipiell sind die Pull-Down-Menüs für alle drei Bildschirme - Programm-Editor, Symbol-Editor und Inbetriebnahme-Bildschirm - identisch. Einige Funktionen haben nur im Zusammenhang mit einem bestimmten Bildschirm Sinn, und sind über den anderen beiden Bildschirmen hell, nicht aktivierbar, dargestellt.

3.5.1 Tasten und Funktionen in den Pull-Down-Menüs

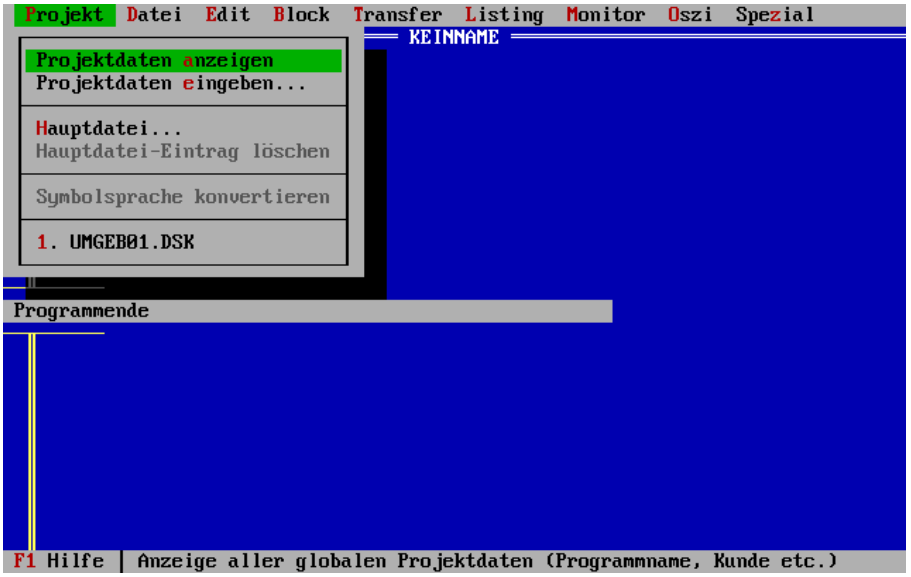
In der Hauptmenüleiste und den Pull-Down-Menüs bewegt man sich mit folgenden Tasten.

Taste:	**Funktion:**
Cursor hoch	Eine Auswahlzeile zurück
Cursor runter	Eine Auswahlzeile vor
Cursor links	Ein Menüpunkt zurück
Cursor rechts	Ein Menüpunkt vor
Home	Erste Auswahlzeile
End	Letzte Auswahlzeile
ENTER < +	Funktion unter Cursor aktivieren
ESC	Abbruch

Durch Drücken eines farblich hervorgehobenen Buchstabens oder einer hinter einem Menüpunkt

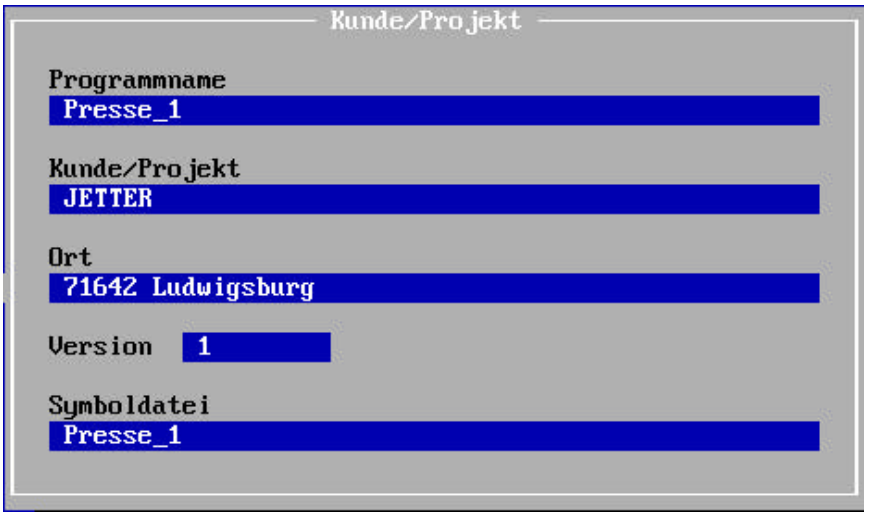
angeführten Funktionstaste kann der zugehörige Auswahlpunkt direkt aktiviert werden (Hotkeys).
Auswahlzeilen werden angekreuzt [x], indem die ALT-Taste zusammen mit dem farbig hervorgehobenen Buchstaben gedrückt wird.

3.5.2 Das Menü "Projekt"



Projektdaten anzeigen

Hier werden unter den Punkten
globale Informationen angezeigt.



Projektdaten eingeben

Hier können Sie unter den Punkten (siehe auch Abbildung oben)

- Programmname
- Kunde / Projekt
- Ort
- Version
- Symboldatei

die entsprechenden globalen Informationen eingeben. Mit der Taste ENTER <+ quittieren Sie die Eingaben.
Bei jeder Aktivierung des Programm-Editors wird die Versionsnummer automatisch inkrementiert. Sollte dies nicht erwünscht sein, so ist die Versionsnummer manuell zurückzusetzen.

Hauptdatei

Sollen INCLUDE-Dateien in den Programmtext eingebunden werden so ist eine Hauptdatei zu definieren, in welcher bis zu 32 INCLUDE-Dateien angegeben werden können. Eine Schachtelung von INCLUDE-Dateien ist nicht möglich. (Das Einfügen von INCLUDE-Dateien in den Symbol-Editor ist weiter hinten beschrieben). Mit den Tasten "#" und "i" öffnen Sie ein Fenster, in welchem der Name der INCLUDE-Datei definiert wird. Im Programm-Editor erscheint folgende Zeile:

```
#INCLUDE NAME
```

Eine INCLUDE-Datei entspricht in ihrem Aufbau einer normalen Programm-Datei. Somit können bestehende Programme oder Programmsequenzen zu einem Gesamtprogramm zusammengestellt werden. Eine weitere Möglichkeit besteht darin Programmgrößen zu realisieren die nicht mehr im PC-Speicher Platz finden. Durch das Unterteilen in verschiedene INCLUDE-Dateien, die in der Hauptdatei wieder logisch

zusammengefügt werden, wird die Beschränkung durch die Größe des PC-Speichers umgangen. Ausführliche Beschreibung in *Kapitel 3.7 INCLUDE-Dateien*.

Hinweis:

In der Hauptdatei dürfen maximal 32 INCLUDE-Dateien definiert werden.

Hauptdatei-Eintrag löschen

Löscht die Hauptdatei, die mit der Auswahlzeile "Hauptdatei..." definiert wurde.

Symbolsprache konvertieren

Konvertiert die in der SYMPAS-Symbolik verwendete Landessprache in eine andere Landessprache. Dazu wird im Symbol-Editor der alternative Ausdruck in eckigen Klammern angegeben.

Symbolsprache konvertieren

Vor der Konvertierung

In eckigen Klammern das alternative Symbol definieren

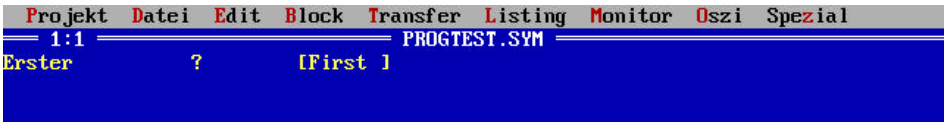


Abbildung 6: Symbol-Editor vor der Konvertierung

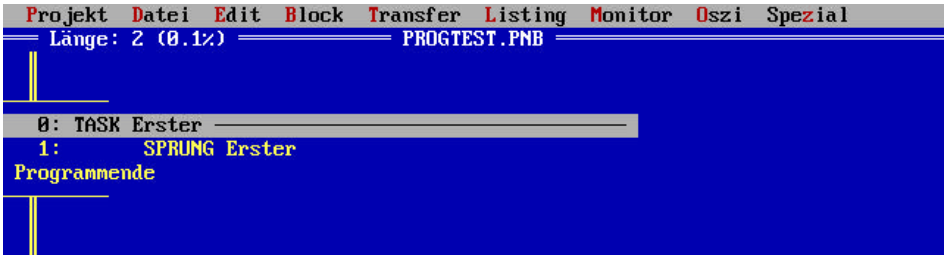


Abbildung 7: Programm-Editor vor der Konvertierung

Konvertierung im Menü "Projekt / Symbolsprache konvertieren" starten

Nach der Konvertierung

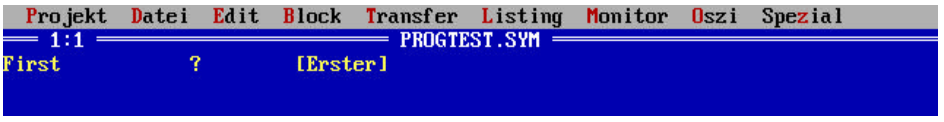


Abbildung 8: Symbol-Editor nach der Konvertierung

Die Programmiersprache im Menü "Spezial / Einstellungen " wählen

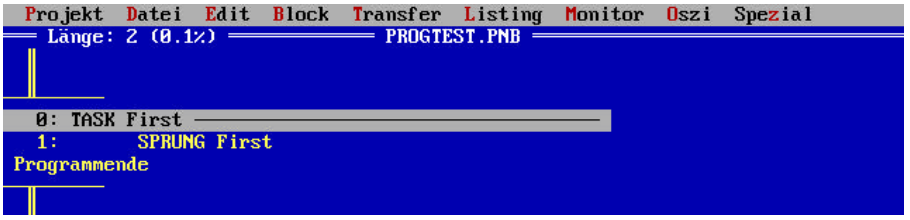
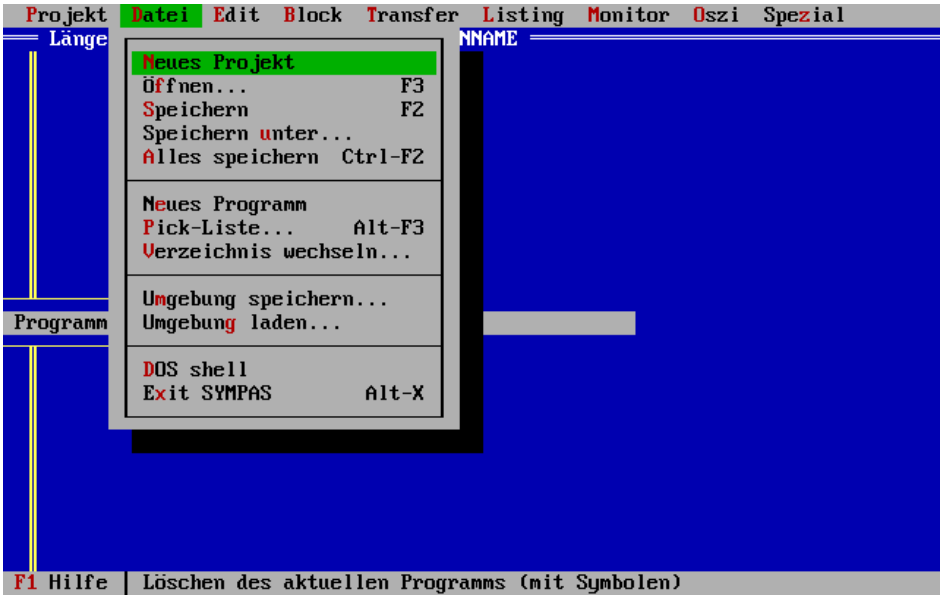


Abbildung 9: Programm-Editor nach der Konvertierung

3.5.3 Das Menü "Datei"



Neues Projekt

Um ein neues Programm zu beginnen kann der Programm- und Symbol-Editor rückgesetzt werden. Der derzeitige Inhalt beider Editoren geht bei diesem Schritt verloren und sollte vorher gesichert werden. Sie finden jetzt die gleichen Voraussetzungen, wie nach einem Neustart von SYMPAS vor.

Öffnen...

Mit diesem Menüpunkt werden Programme, oder Symboldateien von den Disketten- oder Festplatten-laufwerken geladen. Nachdem Sie diesen Punkt aufgerufen haben, erscheint ein Fenster, in welchem der komplette Pfad und der Dateiname eingegeben werden können. Gehen Sie nicht auf dieses Fenster ein, sondern drücken ENTER <+ zeigt ein Fenster alle verfügbaren Verzeichnisse und Dateinamen, unter

welchen Sie mit den Cursortasten auswählen können (mit ENTER <+ laden).

Speichern

Diese Funktion sichert den Programm-, oder Symbol-Editor auf das Laufwerk unter dem im Menü "Projektdaten eingeben" definierten Namen.

Speichern unter...

Diese Funktion sichert den Programm-, oder Symbol-Editor unter beliebigem Namen auf das Laufwerk. Nachdem "Speichern unter..." aktiviert wurde, öffnet sich ein Fenster, in dem der unter "Projektdaten" festgelegte Dateiname erscheint. Dieser Name (und Pfad) kann mit ENTER <+ bestätigt oder auch geändert und dann mit ENTER <+ abgeschlossen werden. Der angegebene Name wird zum aktuellen Namen.

Alles speichern

Diese Funktion sichert sowohl den Programm-Editor als auch den Symbol-Editor auf Diskette oder Festplatte. Ist keine Symboldatei im Fenster "Projektdaten eingeben" definiert worden, wird nur der Programm-Editor gespeichert. Es gelten die Einstellungen aus dem Fenster "Projektdaten eingeben...".

Neues Programm

Das Programm im Programm-Editor wird gelöscht. Der Inhalt des Symbol-Editors wird nicht verändert. (INCLUDE-Datei zum Projekt hinzufügen).

Pick-Liste...

Es können maximal 32 Dateinamen in ein Auswahlfenster, und die entsprechenden Dateien aus diesem heraus in den Programm-Editor geladen werden.
Eine Hauptdatei und bis zu 32 INCLUDE-Dateien können über diese Pickliste komfortabel bearbeitet werden. Mit SHIFT-F9 kann zwischen den beiden zuletzt bearbeiteten Dateien hin- und hergeschaltet werden. Mit ENTF wird ein Listeneintrag entfernt.

Verzeichnis wechseln...

Mit dieser Funktion wird das Verzeichnis oder Laufwerk gewechselt.

Umgebung speichern...

Verzeichnis mit ENTER wählen mit ALT-W wechseln

Mit dieser Funktion sichern Sie folgende Einstellungen unter einem definierbaren Dateinamen. (Mit ENTER das gewünschte Verzeichnis selektieren mit ALT-W in dieses wechseln.)

Programm-Editor:

- Programmname
- Cursorstellung
- Blockdaten
- Programmarken-Daten
- Schalterstellung "Symbolparameter anzeigen"
- Schalterstellung "Monitorfunktion"

Symbol-Editor:

- Symboldateiname
- Cursorstellung
- Blockdaten
- Programmarken

Inbetriebnahme-Bildschirm:

- hier wird das komplette Erscheinungsbild des Bildschirmes gespeichert. Der Zustand aller Fenster bis hin zur Cursorstellung wird festgehalten.

Die Extension der Umgebungsdateien ist .DSK.

Umgebung laden...

Diese Funktion lädt eine *.DSK-Datei. Siehe Umgebung speichern.

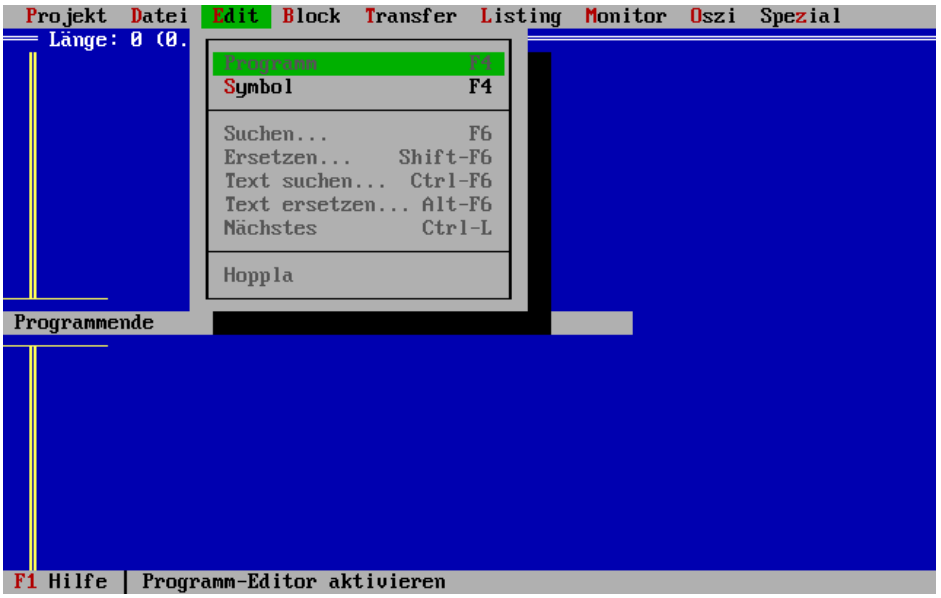
DOS shell

Hiermit unterbrechen Sie SYMPAS um auf DOS-Ebene zu operieren und können von dort mit dem DOS-Befehl EXIT wieder nach SYMPAS zurückkehren.

Exit SYMPAS

Mit dieser Funktion wird SYMPAS beendet und zur DOS-Ebene zurückgekehrt.

3.5.4 Das Menü "Edit"



Programm

Hiermit schalten Sie den Programm-Editor ein.

Symbol

Hier schalten Sie in den Symbol-Editor um.

Suchen...

Suchkriterien: Befehl einschließlich Parameter- oder Zeilennummer

Diese Funktion unterstützt das Suchen eines Befehles oder einer Programmzeile. Nachdem die Funktion aktiviert wurde, erscheint ein Fenster, in welchem der zu suchende Begriff (Suchkriterien: Befehl einschließlich der Parameter- oder Programmzeilenummer) angegeben werden kann. Indirekt-Level sind dabei keine Suchkriterien.

Gesucht wird ab der Cursorposition nach unten bis zum Programmende.

Ersetzen...

Dieser Punkt hängt mit der vorhergehenden Suchfunktion zusammen. Der Befehl verhält sich zunächst analog zu dem obigen Suchbefehl, bietet dann aber noch eine weitere Eingabezeile an, in welcher der Begriff einzutragen ist, durch welchen der Suchbegriff zu ersetzen ist. Indirekt-Level sind nicht möglich.

Text suchen...

Mit Hilfe dieser Funktion werden Texte gesucht. Diese können sich in den Kommentarbereichen befinden oder Parameterbestandteile von Befehlen sein (z.B. ANZEIGE_TEXT).

Text ersetzen...

Dieser Punkt hängt mit der vorhergehenden Suchfunktion zusammen. Außerdem kann ein Begriff definiert werden durch welchen der Suchbegriff zu ersetzen ist.

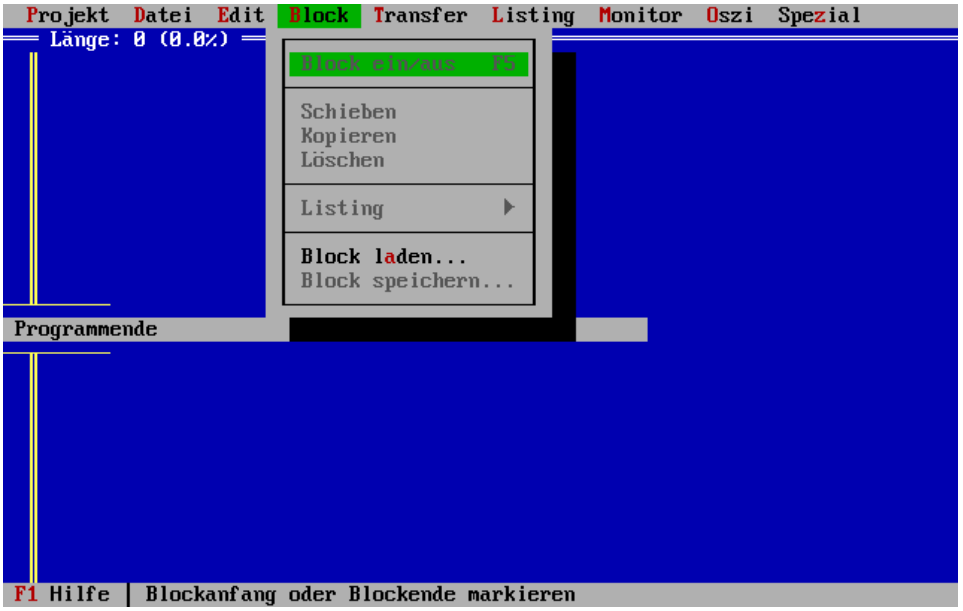
Nächstes

Auch dieser Befehl bezieht sich auf die Suchfunktion. Hier wird derselbe Befehl - wie in "Suchen" definiert - ein weiteres mal im Programmtext gesucht und gegebenenfalls ersetzt.

Hoppla

Dieser Punkt holt die zuletzt gelöschte Zeile zurück.

3.5.5 Das Menü "Block"



Block ein/aus

Hier kann die Blockfunktion ein- und ausgeschaltet werden. Nach dem Einschalten kann mit der Cursortaste ↓ der Block beginnend von der aktuellen Cursorposition markiert werden. Ist dies geschehen, wird der Markiermodus, durch wiederholtes Aufrufen dieser Funktion, ausgeschaltet. Der Block kann jetzt weiter verarbeitet werden.

Schieben (Ctrl K-V)

Verschiebt den Block von der Ausgangsposition vor die aktuelle Cursorposition. Der Block befindet sich nach der Ausführung dieses Befehles also nicht mehr an seiner ursprünglichen Position.

Kopieren (Ctrl K-C)

Kopiert den Block vor die aktuelle Cursorposition. Der Block befindet sich nach der Ausführung des Befehles sowohl an der Stelle, an welcher er markiert wurde, als auch oberhalb der aktuellen Cursorposition.

Löschen (Ctrl K-Y)

Löscht den markierten Block.

Listing

Mit Hilfe dieser Funktion können als Block markierte Programmsequenzen auf dem Drucker ausgegeben oder in einer Datei abgelegt werden. Diese Datei hat das gleiche Format wie die Datenübertragung an den Drucker, während des Druckvorganges. Nach dem Aktivieren der Auswahlzeile "Listing" erscheint ein Fenster, in welchem die Ausgabe auf Drucker oder Datei, die Blattlänge und der linke Rand definiert werden können. Die Defaultwerte beziehen sich auf den Ausdruck von Listings auf Endlospapier.

Block laden...

Hier wird ein Block von Diskette oder Festplatte geladen. Es wird ein Fenster geöffnet, in welchem der Name des zu ladenden Blockes angegeben werden kann. Ignoriert man dieses Fenster und drückt ENTER <+ erscheint ein Fenster mit den zur Auswahl stehenden Dateien. Eine Blockdatei ist, ihrem Aufbau nach, mit der Datei eines kompletten Programmes identisch.

Block speichern...

Der Block wird auf Diskette oder Festplatte gesichert. Nach dem Bestätigen der Auswahlzeile erscheint ein Fenster in dem der Name des Blockes festgelegt werden kann. Eine Blockdatei unterscheidet sich in ihrem Aufbau nicht von der Datei eines kompletten Programmes.

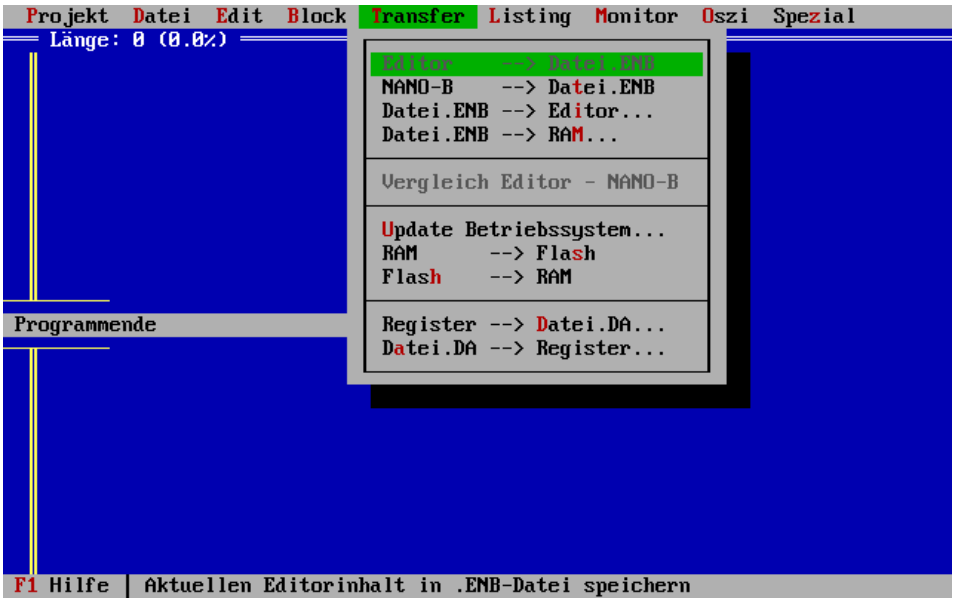
Hinweis:

Für die Auswahlzeilen "Block laden..." und "Block speichern..." kann ein Pfad auf ein Verzeichnis vorgegeben werden, der in dem erscheinenden Eingabefenster nicht mehr erneut definiert werden muss.
Dazu ist auf DOS-Ebene (z.B. in der AUTOEXEC.BAT) folgender Befehl zu geben

SET SYMPASBLOCKS="PFAD"

wobei für "Pfad" zum Beispiel C:\BLOCK stehen kann.

3.5.6 Das Menü "Transfer"



Extension beispielhaft für NANO-B; lautet je nach Steuerung verschieden

Editor -> Datei.ENB

Diese Funktion überträgt das Programm in eine Objektdatei deren Name in einem Fenster definiert werden kann.

NANO-B -> Datei.ENB

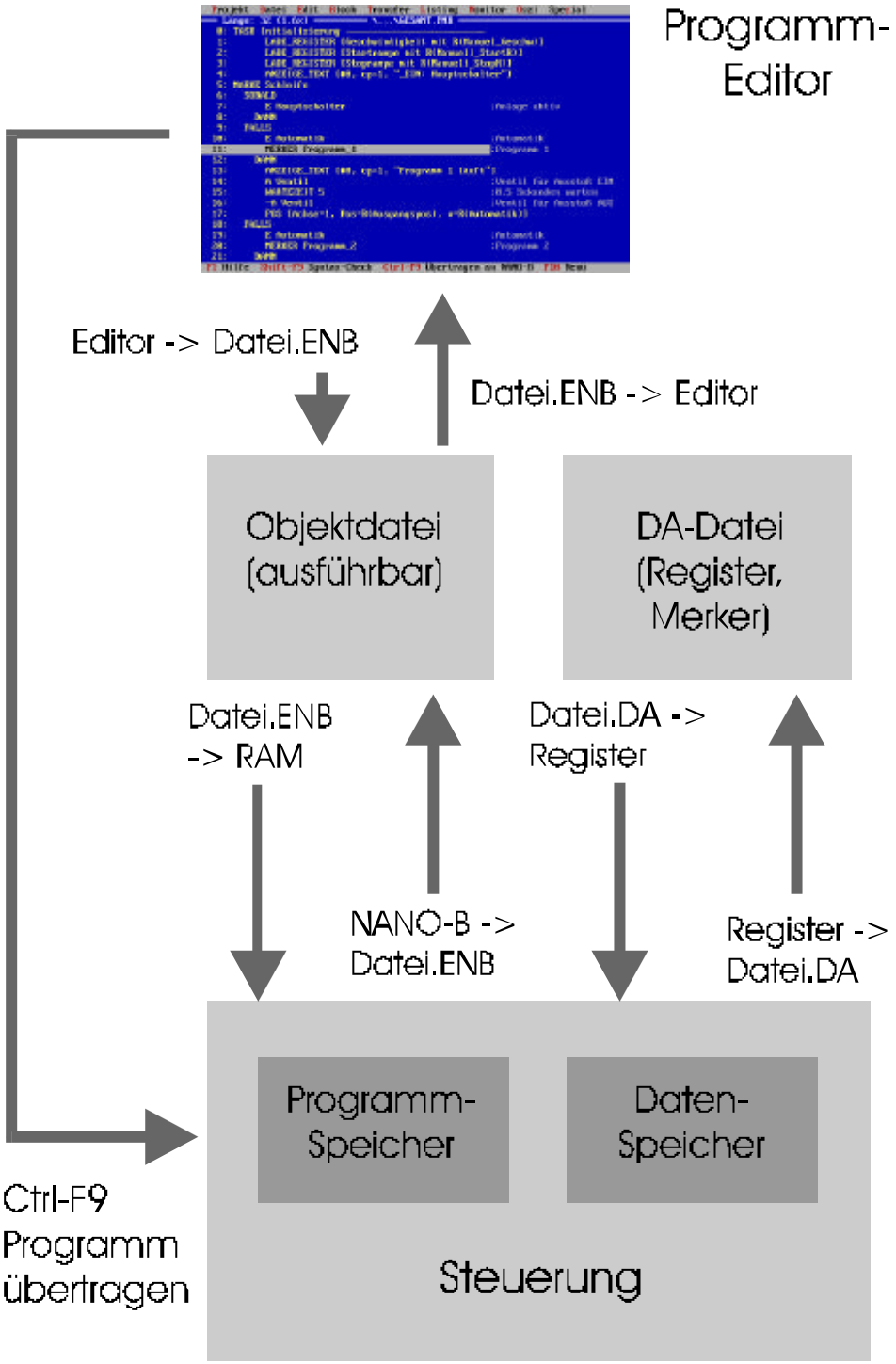
Die Übertragung eines Programmes vom RAM der Steuerung in eine Objektdatei deren Name in einem Fenster definiert werden kann.

Datei.ENB -> Editor...

Die Übertragung eines Programmes aus einer Objektdatei in den Programm-Editor.

SYMPAS Programm- und Systemdateien

(Extension *.ENB beispielhaft NANO-B)



Datei.ENB -> NANO-B...

Die Übertragung eines Programmes aus einer Objektdatei in das RAM der Steuerung. Das Programm wird übertragen aber nicht gestartet.

Vergleich Editor -> NANO-B

Vergleicht den Programm-Editor mit dem Programm im RAM der Steuerung. Ein Fenster gibt darüber Auskunft, inwieweit beide Programme identisch sind.

Register -> Datei.DA...

Mit dieser Funktion sind Sie in der Lage selbst definierte Register- und Merkerbereiche als Datei auf Festplatte zu sichern. Nach dem Aufruf der Funktion erscheint ein Fenster in dem Sie verschiedene Bereiche festlegen können. Diese Bereiche können sich auch im Typ voneinander unterscheiden. Es sind Register und Merkerbereiche möglich. Maximal sind 8 verschiedene Bereiche möglich. Nachdem alle gewünschten Bereiche festgelegt worden sind, verlassen Sie das Fenster mit der ersten Menüzeile "Auswahl beendet". Jetzt erscheint ein Fenster, in welchem Sie den Namen für die Datei festlegen können. SYMPAS fügt die Extension ".DA" an und legt die Datei auf der Festplatte ab.

Datei.DA -> Register...

Die im letzten Punkt beschriebene Datei, zum Speichern von Register- und Merkerbereichen, wird von Festplatte oder Diskette in die Steuerung geladen. Damit sind jetzt alle Register- und Merkerbereiche der Steuerung, welche in der Datei definiert sind, auf diesen Stand gebracht worden.

Die DA-Datei

DA-Datei speichert Register- und Merker-bereiche auf PC oder VIADUKT

Die DA-Datei ist eine ASCII-Datei die auf der Festplatte des PC oder des VIADUKT abgelegt werden und von dort wieder in eine Steuerung geladen werden können.

Exemplarisch kann eine DA-Datei folgendermaßen aussehen:

Kopf ->

```
SD1001
; NANO-B DATA FILE - JETTER
Steuerungstechnik 71642 Ludwigsburg
; C:\SYMPAS\BEISPIEL\BEISPIE1
```

Kopf-Definition

Register-bereich ->

```
RS 1 10
RS 2 20
RS 3 30
RS 4 40
RS 5 50
```

Register-liste

Merker-bereich ->

```
FS 1 0
FS 2 0
FS 3 0
FS 4 0
FS 5 0
```

Merker-liste

In obigen Beispiel wurden folgende Register- beziehungsweise Merkerbereiche in die DA-Datei "BEISPIE1.DA" gespeichert.

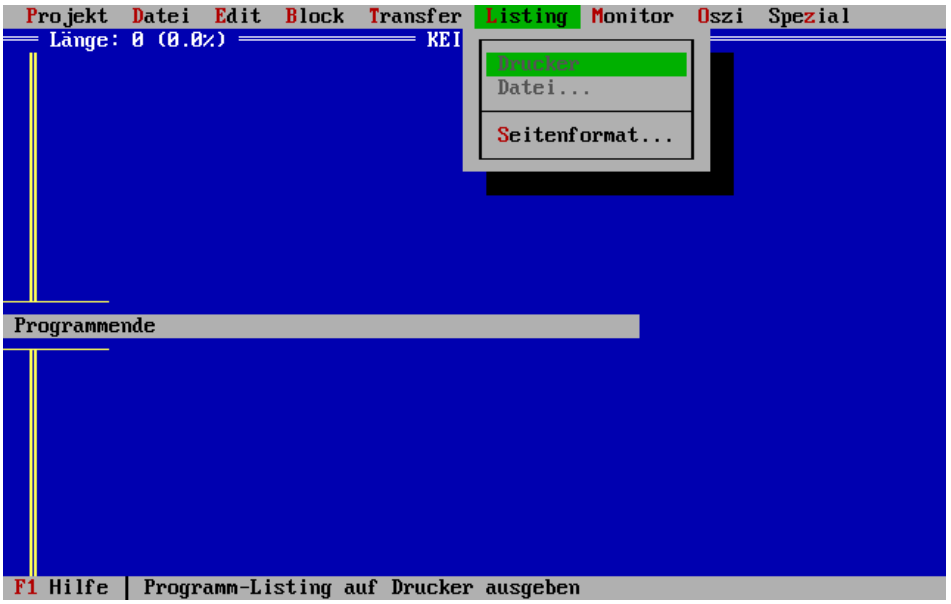
- Register 1 bis 5 mit entsprechendem Inhalt
- Merker 1 bis 5 mit entsprechendem Status

Aufbau DA-Datei

Die Register respektive die Merkerliste ist folgendermaßen aufgebaut.

- 1 Spalte Kennung: RS für Register, FS für Merker
- 2 Spalte Register- respektive Merkernummer
- 3 Spalte Registerinhalt respektive Merkerstatus

3.5.7 Das Menü "Listing"



Drucker

Der Inhalt des Programm- oder Symbol-Editors wird als Programmlisting auf dem Drucker ausgegeben.

Datei...

Der Inhalt des Programm- oder Symbol-Editors wird als Programmlisting in eine Datei geschrieben. Nach dem Aktivieren der Auswahlzeile "Datei" öffnet sich ein Fenster, in welchem Sie den Namen der Datei festlegen können, die das Programmlisting aufnimmt. Die Extension *.LST wird automatisch angefügt.

Seitenformat...

In einem Fenster können verschiedene Einstellungen zum Seitenformat vorgenommen werden.

Anzahl Zeilen

Hier kann die Blattlänge definiert werden. Der Defaultwert bezieht sich auf die Ausgabe von Listings auf Endlospapier.

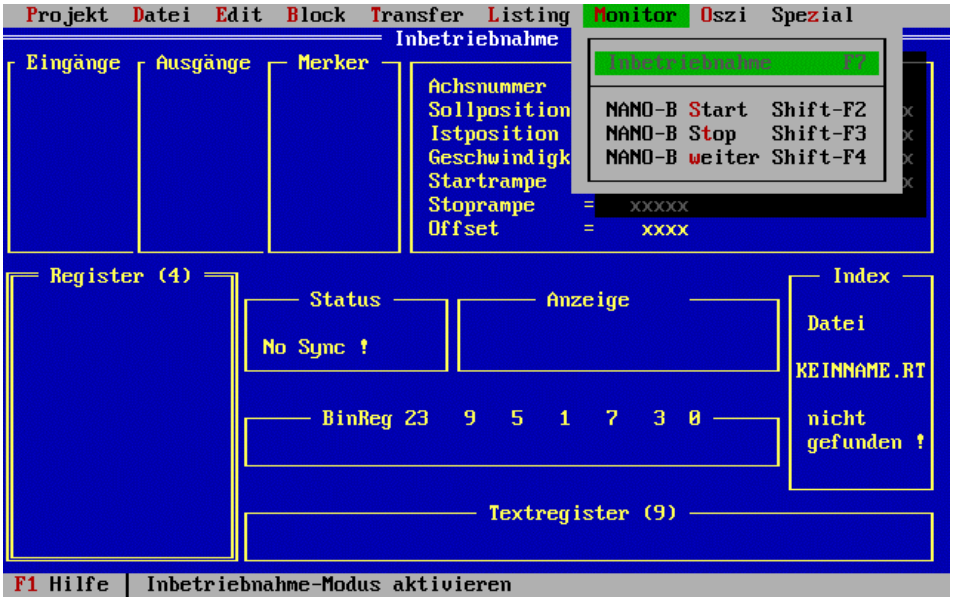
Heftrand

Hier wird die Breite des linken Randes des Listings definiert. Die einzugebende Zahl steht für die Anzahl vorangestellter Leerzeichen, bevor der eigentliche Text beginnt.

Seitenvorschub

Die Funktion (angekreuzt bedeutet EIN) erzeugt am Ende einer jeden ausgedruckten Listingseite einen Seitenvorschub. Seitenvorschub AUS gibt statt dessen Leerzeilen aus.

3.5.8 Das Menü "Monitor"



Inbetriebnahme

Diese Auswahlzeile schaltet auf den Inbetriebnahme-Bildschirm um.

NANO-B Start

Hiermit starten Sie das Programm im Steuerungs-RAM. Das Programm wurde vorher zum Beispiel mit der Auswahlzeile "Datei.EPR -> RAM" aus dem Pull-Down-Menü "Transfer" in den Steuerungs-RAM übertragen. Mit dieser Funktion wird es gestartet.

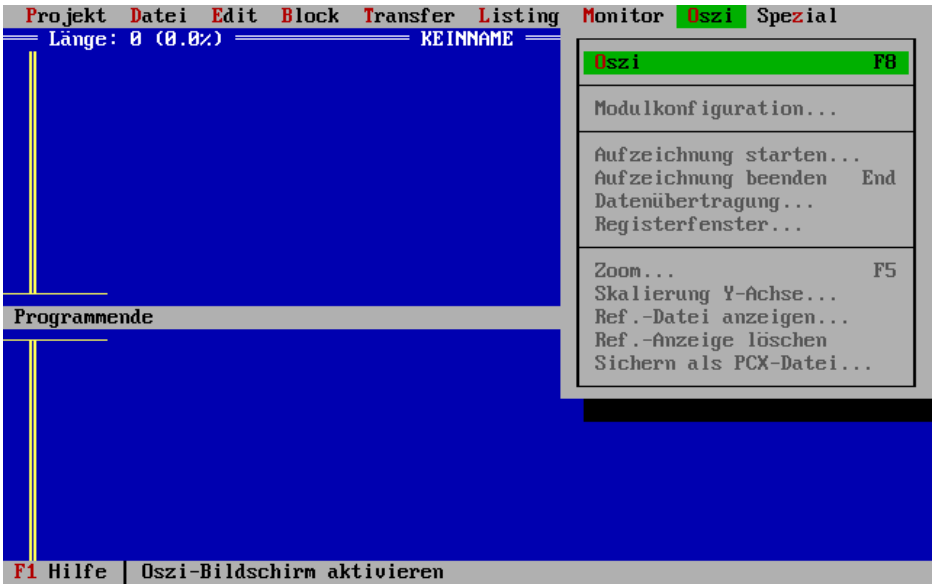
NANO-B Stop

Die Abarbeitung des Anwenderprogrammes wird gestoppt.

NANO-B weiter

Die Abarbeitung des Anwenderprogrammes wird an der Stelle fortgesetzt an der sie unterbrochen wurde. Im Gegensatz dazu würde die Funktion "NANO-B Start" am Anfang des Programmes beginnen.

3.5.9 Das Menü "Oszi"



Aufzeichnung beliebiger Register verschiedener Reglermodule

Mit Hilfe der Oszi-Funktion können beliebige Register folgender Module aufgezeichnet werden

- PASE-E SV4 Plus (Servoregler)
- PASE-E DIMA3 (Digitaler Servoregler)
- PASE-E PID4 (Digitaler PID-Regler)
- PASE-E AD16 (Analoge Eingangskarte)

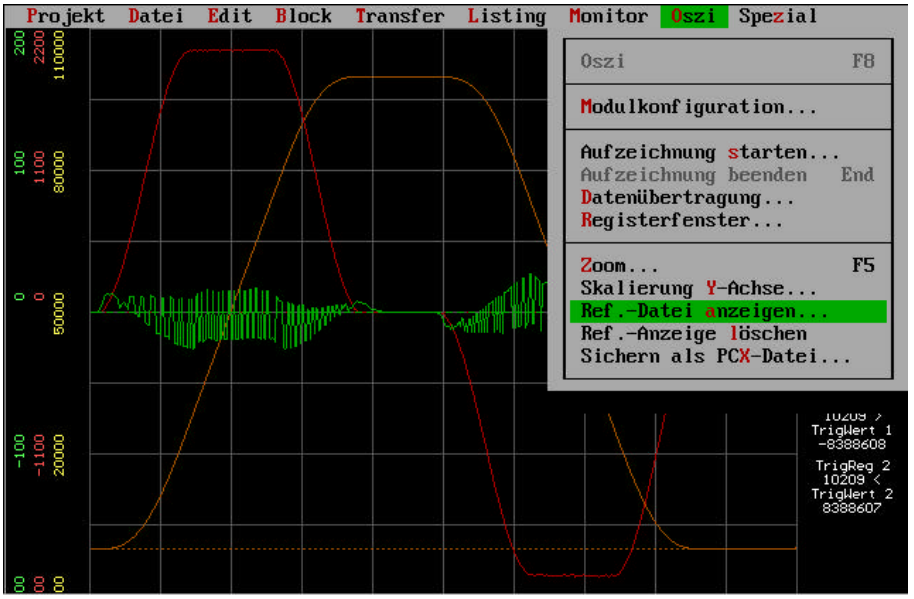
- NANO SV (Servoregler)
- NANO PID (Digitaler PID-Regler)

Da alle Registerinhalte mitprotokolliert werden können, ist es möglich die Geschwindigkeit und Position einer Achse über der Zeit oder den Graphen eines Analogeinganges darzustellen um nur einige Beispiele zu nennen.

3 Graphen können gleichzeitig dargestellt werden

Gleichzeitig können bis zu 3 Graphen auf dem Oszi-Bildschirm dargestellt werden.

Mit (F8) in den Oszi-Bildschirm

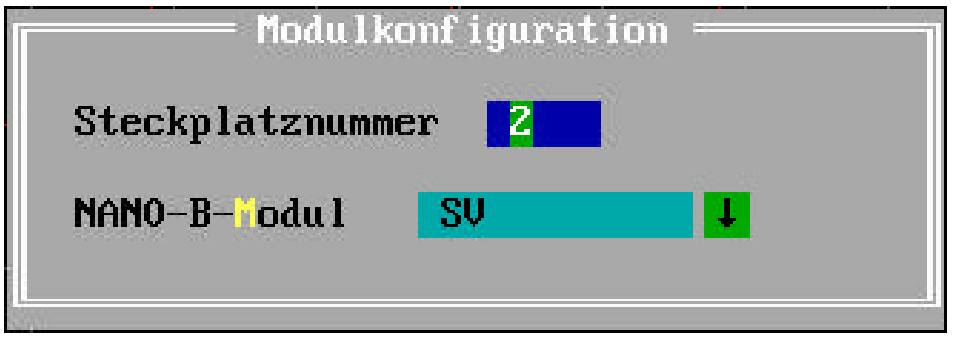


Im Menü "Oszi" stehen folgende Funktionen zur Verfügung.

Modulkonfiguration...

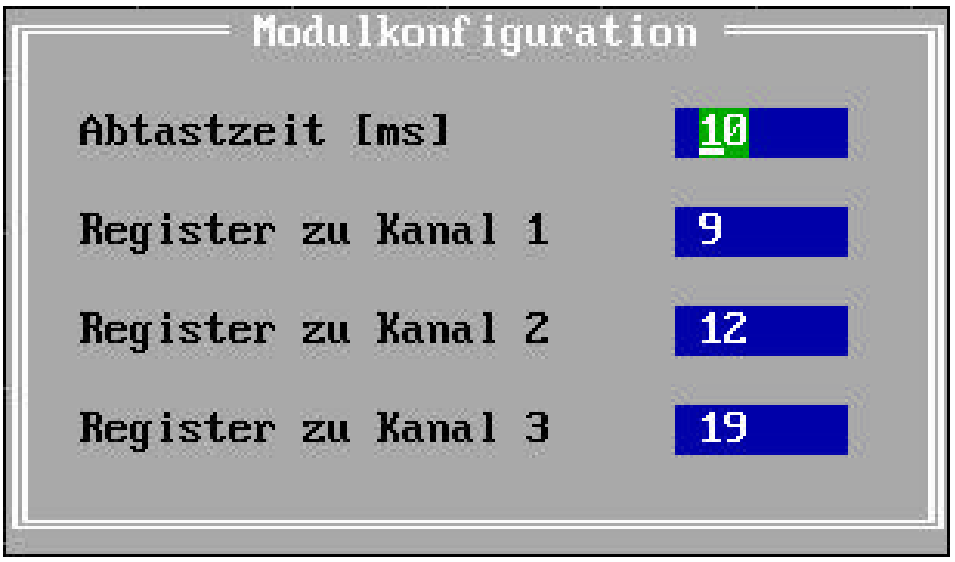
Steckplatz-nummer und Modulart eingeben

Zunächst erscheint ein Fenster, in welchem die Nummer und Art des aufzuzeichnenden Modules angegeben wird.



Zur Eingabe der eigentlichen Modulkonfiguration öffnet sich nun ein weiteres Fenster.

Abtastzeit definieren und Register / Kanal-zuordnung treffen



Hier definieren Sie die Abtastzeit und geben an, welches Register der Reglermodule auf welchem Kanal (Kurve) des Oszi-Bildschirmes dargestellt werden soll.

Aufzeichnung starten...

Es erscheint folgendes Fenster

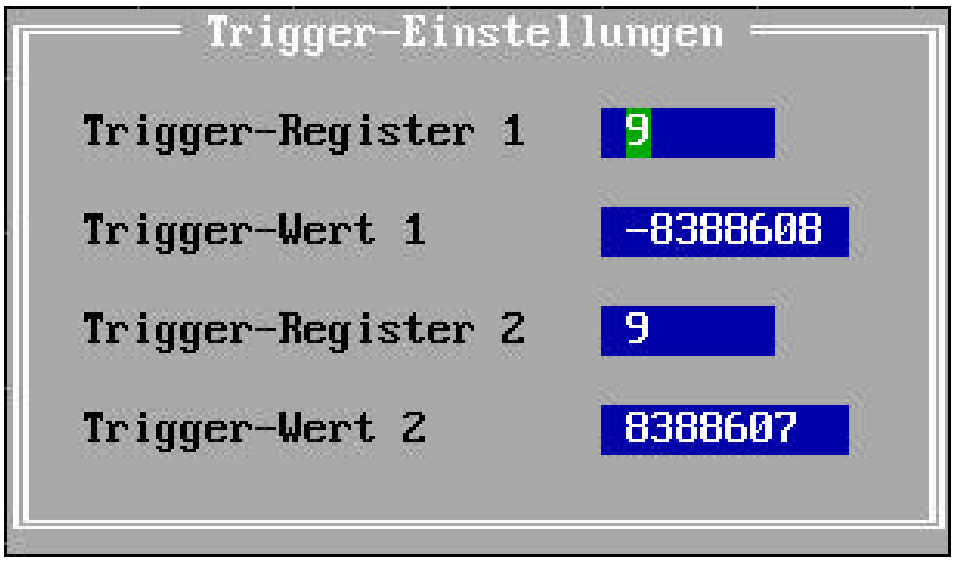
Start Startet die Aufzeichnung der unter "Modulkonfiguration" zugeordneten Register.

Bedingter Start

Startet die Aufzeichnung der unter "Modulkonfiguration" zugeordneten Register, in Abhängigkeit der Bedingungen die in der Auswahlzeile "Trigger-Einstellungen" festegelgt werden können.

Trigger-Einstellungen

Es wird folgendes Fenster geöffnet



Beide Trigger-Bedingungen müssen erfüllt sein

Wobei die Bedingung bei Trigger-Register 1 ist:

- Trigger-Register 1 > Trigger-Wert 1

Die Bedingung bei Trigger-Register 2 ist:

- Trigger-Register 2 < Trigger-Wert 2

Beide Trigger-Bedingungen müssen erfüllt sein.

Aufzeichnung beenden

Beendet die Aufzeichnung der Registerwerte.

Datenübertragung...

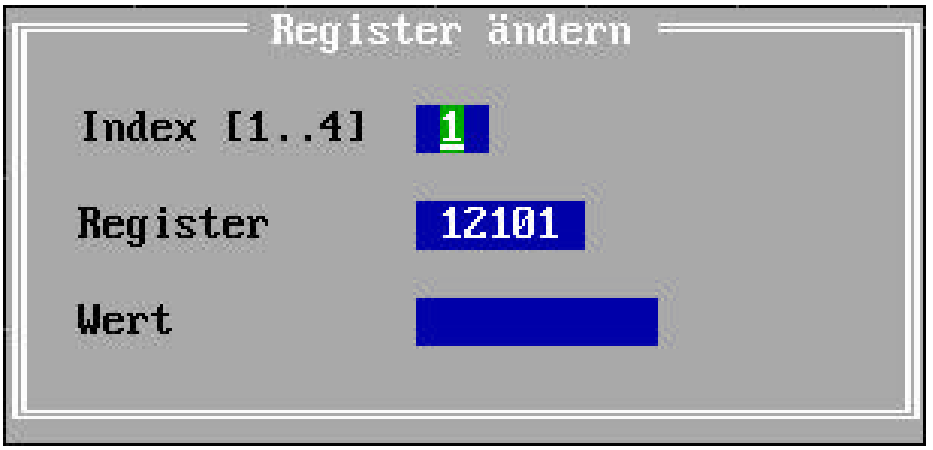
Anzahl der darzustellenden Kurven eingeben

In folgendem Fenster definieren Sie wie viele Kurven aus dem Speicher der Steuerung in SYMPAS eingelesen respektive auf dem Oszi-Bildschirm dargestellt werden.

Registerfenster...

Die Inhalte 4 zusätzlicher Register können angezeigt werden

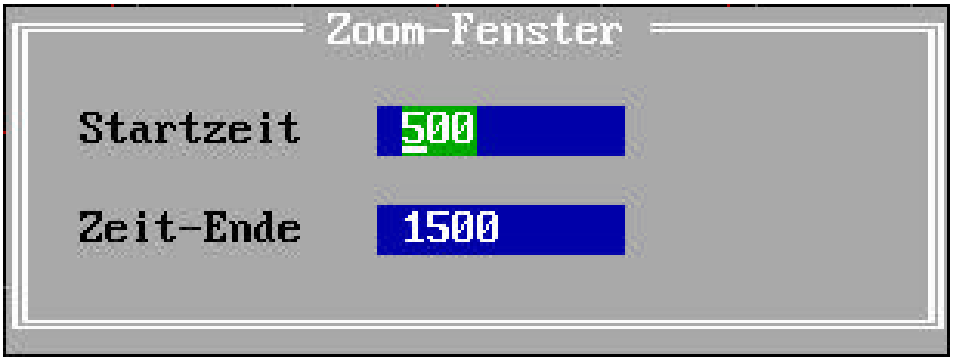
Mit dieser Funktion können in der rechten oberen Ecke des Oszi-Bildschirmes bis zu 4 beliebige Steuerungsregister angezeigt und geändert werden.



Zoom...

Mit Zoom die X-Achse skalieren

Geben Sie an welcher Bereich der Zeitachse über den gesamten Bildschirm dargestellt werden soll.



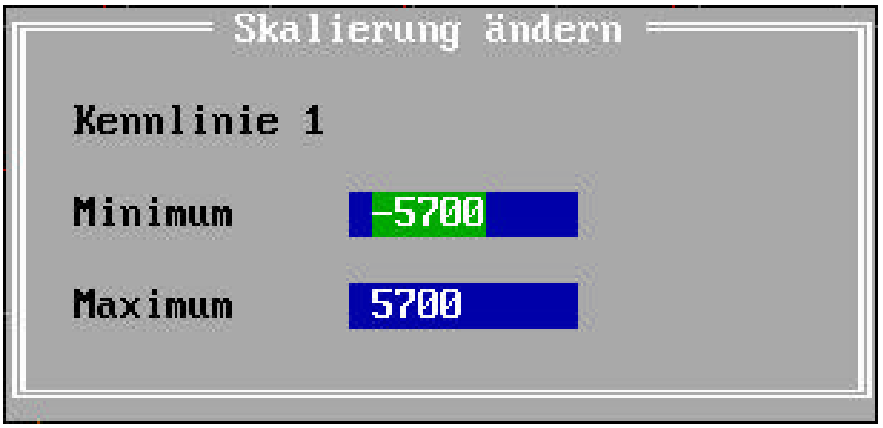
Skalierung Y-Achse...

Die Y-Achsen-Skalierung jeder Kennlinie kann separat gewählt werden

Definieren Sie zunächst für welche Kennlinie die Darstellung der Y-Achse skaliert werden soll.

Geben Sie jetzt den neuen Wertebereich, der dargestellt werden soll.

Der Darstellungs-bereich jeder Kennlinie kann gewählt werden



Ref.-Datei anzeigen...

Die Bildschirmdarstellung kann mit "Datei / Speichern" abgespeichert werden (*.SCP). Mit "Ref.-Datei anzeigen..." wird eine Referenzdatei, die sich auf der Platte befindet wieder auf dem Bildschirm dargestellt.

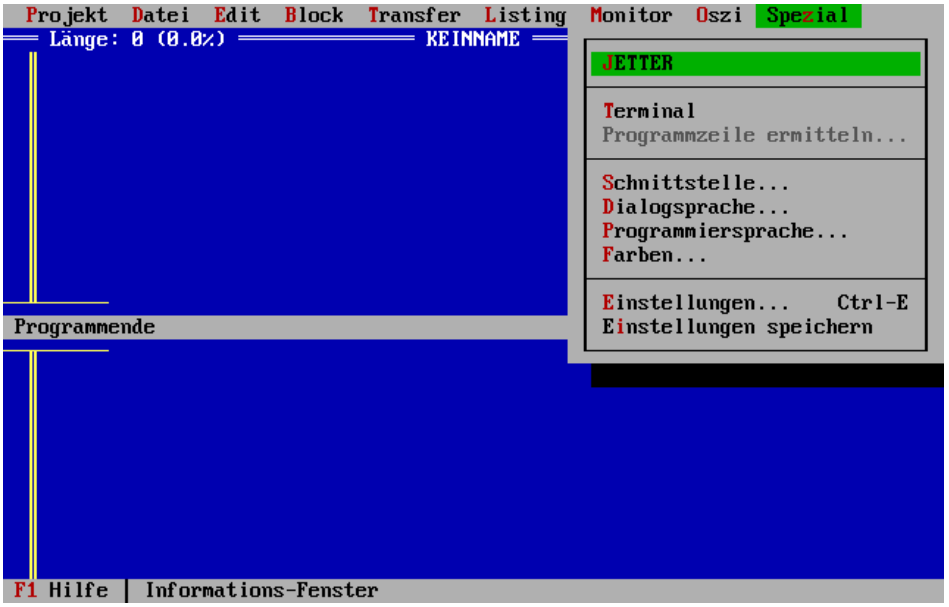
Ref.-Anzeige löschen

Die Bildschirmdarstellung der Referenzdatei (*.SCP) wird wieder gelöscht.

Sichern als PCX-Datei...

Die Bildschirmdarstellung wird als PCX-Datei auf Platte abgelegt.

3.5.10 Das Menü "Spezial"



JETTER

Hier erscheint ein Informationsfenster. Aus diesem Fenster können folgende Informationen entnommen werden:

- Programmversion
- Unsere Telefonnummer
- Aktuelle Schnittstelle: COM1, COM2 oder JETWay
- Steuerungs-Status: Online oder Offline
- Umgebung speichern: EIN oder AUS
  Dieser Punkt gibt über die Schalterstellung der Funktion "Umgebung automatisch speichern" Auskunft.
- Syntax-Check: EIN oder AUS
  Dieser Punkt gibt über die Schalterstellung der Funktion "Syntax-Check" Auskunft.
- Größe des noch verfügbaren RAM im PC

Terminal

Mit dieser Funktion wird ein Terminal simuliert. In dem oberen Bildschirmabschnitt befinden sich die über die Schnittstelle gesendeten Daten, in dem unteren Bildabschnitt werden die Daten angezeigt die an SYMPAS zurückgesandt werden.
Diese Funktionen bleiben der firmeninternen Nutzung vorbehalten.

Programmzeile ermitteln

Aktuelle Programmzeile ermitteln (Taskpointer, Spezialfunktion).

Schnittstelle...

Schnittstelle

Timeoutzeit

Baudrate

In dem erscheinenden Fenster können Sie die Schnittstelle festlegen, welche die Verbindung zur Steuerung bildet. Es kann zwischen COM1 und COM2 des PC oder der JETWay-Schnittstelle gewählt werden (siehe *Kapitel 2.4 SYMPAS und mehrere vernetzte Steuerungen (JETWay-H)).* Außerdem kann eine Timeoutzeit für die gewählte Schnittstelle angegeben werden. Mit dieser Funktion kann der Wechsel zwischen den verschiedenen Bildschirmen (Programm-Editor, Symbol-Editor, Inbetriebnahme-Bildschirm) bei nicht angeschlossener Steuerung beschleunigt werden. Außerdem wird die Baudrate für DA-Datei respektive Programmübertragung angegeben werden.

Dialogsprache...

Hier wählen Sie die Dialogsprache, z.B. die Sprache der Pull-Down-Menüs, Hilfsfenster, etc. Es kann zwischen englisch und deutsch gewählt werden.

Programmiersprache...

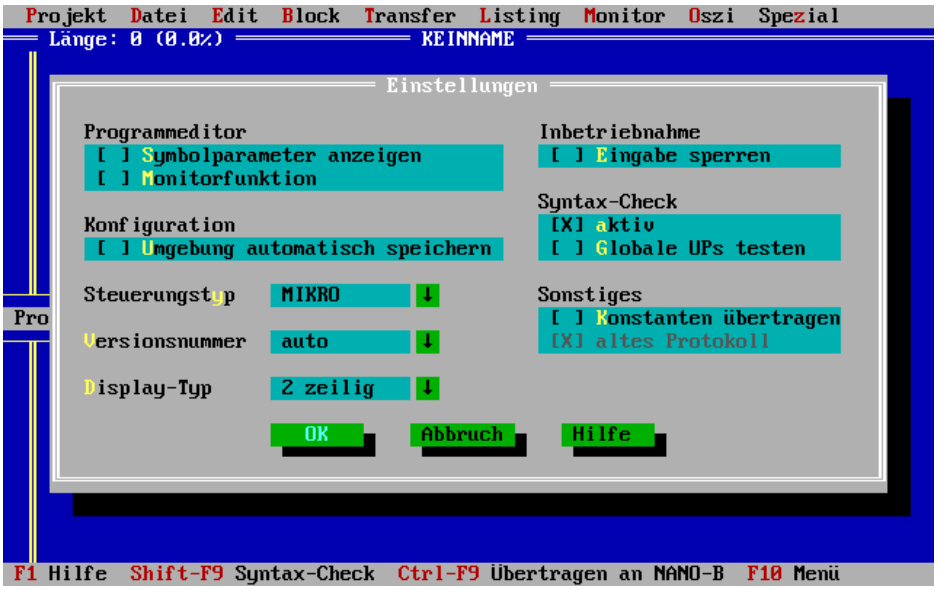
Hier wählen Sie die Programmiersprache, also die Sprache in welcher die Befehle der Programmiersprache dargestellt werden. Es kann zwischen englisch und deutsch gewählt werden.

Farben...

In dieser Auswahlzeile können die Farbeinstellungen für die gesamte Programmierumgebung getroffen werden. Nach dem Bestätigen der Auswahlzeile erscheint ein Fenster, in dem wiederum 4 Unterfenster angewählt werden können. Mit der TAB-Taste werden die Fenster "Gruppe", "Detail", "Vordergrund" und "Hintergrund" der Reihe nach aktiviert (sichtbar durch doppelte Umrandung). Mit der SHIFT-TAB-Tastenkombination werden die Fenster in umgekehrter Reihenfolge aktiviert. In den einzelnen Fenstern bewegt man sich mit den Cursortasten. Für jeden Punkt des Fensters "Gruppe" finden sich im Fenster "Detail" ein oder mehrere Unterpunkte, welchen Sie in den Fenstern "Vordergrund" und "Hintergrund" eine entsprechende Farbe zuweisen können. Mit der ESC-Taste wird die Farbeinstellung abgebrochen, mit der Taste ENTER <+ quittiert. Unterhalb des Fensters "Hintergrund" befindet sich ein Test-Text, an welchem die Farbeinstellungen exemplarisch dargestellt werden.

Einstellungen...

In einem Fenster können folgende Einstellungen getroffen werden.



Programmeditor

- Symbolparameter anzeigen (Ctrl-Alt-S)

  Bei der Verwendung von symbolischen Begriffen für Parameter der Programmiersprachen-Befehle blendet diese Funktion den Zahlwert des Parameters zusätzlich in den Programmtext ein.
  Befindet sich in dem Programm folgende Zeile

  ```
  REG rAnzahlTeile
  ```

  so fügt diese Funktion in die Zeile darunter

  ```
  REG rAnzahlTeile
      100
  ```

  die Nummer des Steuerungs-Registers hinzu. Hierdurch kann die Zuweisung zwischen Symbolik

und physikalisch existenten Registern überprüft werden.

- Monitorfunktion (Ctrl-Alt-M)

  Die Registerinhalte werden im Programm-Editor angezeigt.
  Existiert zum Beispiel folgende Programmzeile

```
LADE_REGISTER [100 mit R(200)]
```

  so würde die "Monitor ein" Funktion zu folgendem Ergebnis führen

```
LADE_REGISTER [100 mit R(200)]
               0         23
```

  Die Inhalte der entsprechenden Register werden in der Zeile darunter angezeigt und ständig aktualisiert.

Konfiguration

- Umgebung automatisch speichern

  Ist dieser Schalter aktiv werden bei jedem Beenden von SYMPAS alle Umgebungseinstellungen (wie in *Kapitel 3.5.3 Das Menü "Datei"* beschrieben) unter dem Namen SYMPAS.DSK gesichert. Bei einem Neustart von SYMPAS werden alle Einstellungen restauriert, wenn der Schalter "Umgebung automatisch speichern" mit "Einst. speichern" festgehalten wird.

- Steuerungstyp

  Hier wird der gewünschte Steuerungstyp eingestellt.

Mit den Einstellungen

- Nummer
- Auto
- Ignorieren

Kompatibilität zu alten Versionen

- Versionsnummer

  Hier wird die Versionsnummer des Betriebssystems der oben angeführten Steuerung eingegeben.
  Auto: Fehlermeldung, wenn bei Programmübertragung Steuerungsversion den verwendeten Befehlssatz nicht beinhaltet (zu alt ist)
  Nummer: Fehlermeldung, wenn bei Programmübersetzung Steuerungsversion den verwendeten Befehlssatz nicht beinhaltet (zu alt ist)

- Display-Typ

  Hier wird angegeben, ob mit einem 2 oder einem 4 zeiligen LCD-Display gearbeitet wird. (Nur Inbetriebnahme-Bildschirm).

Inbetriebnahme

- Eingabe sperren

  Registerinhalte beziehungsweise die Zustände von Eingängen, Ausgängen und Merkern können dargestellt aber nicht geändert werden.

Syntax-Check

- aktiv

Mit diesem Schalter wird die Funktion "Syntax-Check" ein- und ausgeschaltet. Der Syntax-Check überprüft das Programm auf die Richtigkeit folgender Kriterien:

- TASK0 vorhanden ?
- unvollständiges Auskommentieren
- doppelte Marke oder Task
- bedingte Sprunganweisung nicht abgeschlossen
- unvollständige bedingte Sprunganweisung
- Klammerfehler
- Taskbefehle ohne zugehörigen Task
- Sprungbefehl ohne zugehörige Marke
- Unterprogramm ohne zugehörige Marke
- RÜCKSPRUNG ohne UNTERPROGRAMM-Befehl
- Fehler Befehlssyntax
- Abgeschlossenheit Task
- Anzahl Unterprogrammebenen (20 möglich)
- Hauptprogramm läuft in Unterprogramm
- Sprung in anderen Task
- lokales Unterprogramm von fremden Task aufgerufen
- Fehler Arithmetik-, Boole'sche-Syntax

Ist der Schalter der Funktion "Syntax-Check" EIN, so wird bei folgenden Aktionen ein Syntax-Check durchgeführt:

- vor dem Übertragen des Programmes mit der Auswahlzeile "Datei.ENB... -> RAM" im Pull-Down-Menü "Transfer".

- vor dem automatischen Übertragen und Starten des Programmes aus dem Programm-Editor mit der Tastenkombination CTRL-F9.

Unabhängig von der Schalterstellung wird der Syntax-Check durchgeführt:

- mit der Tastenkombination SHIFT-F9 im Programm-Editor.

- Globale UPs testen

  Globale Unterprogramme stehen per Definition am Ende des letzten Task. Abweichende Platzierung mahnt der Syntax-Check an. Wird diese Funktion deaktiviert können globale Unterprogramme beliebig plaziert werden.

Sonstiges

- Konstanten übertragen

  Die in der Symbolik definierten Konstanten werden an die Steuerung übertragen.

- altes Protokoll

  Für die Programmierschnittstelle (RS232) wird das alte Protokoll verwendet. (nur PASE-E PLUS)

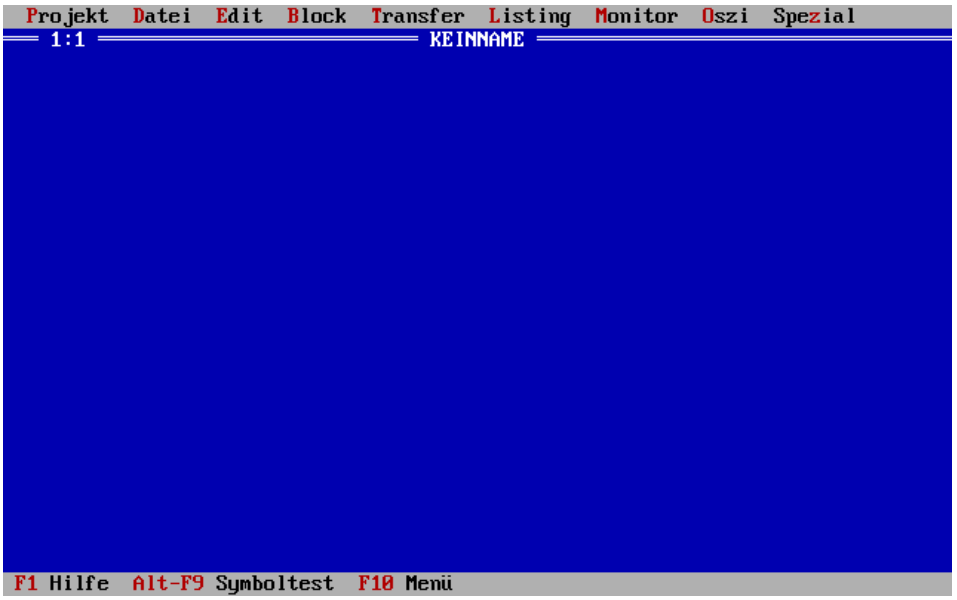
Einst. speichern

Mit der Funktion "Einstellungen speichern" werden folgende Einstellungen festgehalten:

- Dialogsprache

- Programmiersprache
- Schnittstelle
- Zeilen pro Seite
  Diese Einstellung bezieht sich auf die Ausgabe von Block- und Programmlistings.
- Heftrand
  Diese Einstellung bezieht sich auf die Ausgabe von Block- und Programmlistings.
- Schalterstellungen
  - o Umgebung automatisch speichern
  - o Syntax-Check
  - o Seitenvorschub bei Seitenende
  - o Timeout
- Farben

3.6 Symbolische Programmierung - der Symbol-Editor



Mit (F4) zwischen Programm- und Symbol-editor wechseln

Mit der Funktionstaste F4 oder der entprechenden Auswahlzeile im Pull-Down-Menü "Edit" gelangen Sie in den Symbol-Editor. In diesem Editor wird eine Datei angelegt, in der sämtlichen Befehlsparametern der Programmier-sprache symbolische Namen zugewiesen werden können. So wird zum Beispiel aus dem Eingang "E 102" der Eingang "E eStart". Jeder numerische Parameter der Befehlssprache kann durch eine solche - symbolische - Namenszuweisung ersetzt, und damit die Übersichtlichkeit des Programmes gesteigert, und der Pflegeaufwand nach der Fertigstellung verringert werden.

Die Reihenfolge sollte folgendermaßen eingehalten werden. Zunächst erstellen Sie die komplette Symboldatei, um dann im Programm-Editor das dazugehörige Programm zu schreiben.

3.6.1 Tasten und Funktionen im Symbol-Editor

Bei der Erstellung der Symbolik eines Programmes sind einige Regeln zu beachten.

Alle zur Verfügung stehenden ASCII-Zeichen ab der Ordinalzahl 32 können eingegeben werden.

Gültige Symbole müssen in Spalte 1 beginnen und durch mindestens ein Leerzeichen vom darauffolgenden Parameter getrennt sein. Die Symbollänge ist auf 15 Zeichen beschränkt.

Parameter vom Typ Textsequenz (ANZEIGE_TEXT) müssen durch eines der folgenden Zeichen umschlossen sein: " , ' oder # (z.B.: "Hello World"). Die Stringlänge ist auf 24 Zeichen beschränkt.

INCLUDE-Dateien können nach folgendem Muster eingebunden werden:

```
#INCLUDE Dateiname
```

Auch bei Include-Dateien ist der Beginn in der ersten Spalte verbindlich.

Kommentaren muss ein Semikolon oder mindestens ein Leerzeichen vorangestellt werden.

Cursor Bewegungen:

Taste:	**Funktion:**
Cursor hoch	Eine Zeile zurück
Cursor runter	Eine Zeile vor
Seite hoch	Seite zurück
Seite runter	Seite vor
Ctrl-Seite hoch	Zur ersten Zeile
Ctrl-Seite runter	Zur letzten Zeile
Cursor links	Eine Spalte zurück
Cursor rechts	Eine Spalte vor
Home	Zeilenanfang
End	Zeilenende
Ctrl-Cursor links	Vorheriges Wort
Ctrl-Cursor rechts	Nächstes Wort

Editor Befehle:

Taste:	**Funktion:**
ENTER	Neue Zeile
BS	Zeichen vor Cursor löschen
DEL	Zeichen unter Cursor löschen
Ctrl-Y	Zeile löschen

Blockoperationen:

Taste:	Funktion:
Ctrl-K B	Blockanfang markieren
Ctrl-K K	Blockende markieren
Ctrl-K V	Block verschieben
Ctrl-K C	Block kopieren
Ctrl-K Y	Block löschen
Ctrl-K R	Block von Diskette lesen
Ctrl-K W	Block auf Diskette schreiben
Ctrl-K P	Block drucken
Ctrl-K H	Block ausschalten
Ctrl-K L	Zeile markieren
Ctrl-Q B	Blockanfang suchen
Ctrl-Q K	Blockende suchen

Programm-Marken:

Taste:	Funktion:
Ctrl-K 0..9	Es werden 0 bis 9 Cursorpositionen im Symboltext gemerkt.
Ctrl-Q 0..9	Die Cursorpositionen 0 bis 9 werden im Symboltext angesprungen.

3.6.2 Erstellen einer Symboldatei (im Symbol-Editor)

Die numerischen Parameter der Programmiersprache können durch symbolische Namen ersetzt werden. Das trägt zur Übersichtlichkeit des Programmes bei, und erleichtert das Einarbeiten in den Quelltext. Wichtig ist, dass nur der numerische Teil des Parameters durch Symbolik ersetzt wird. Alle anderen Bestandteile des Parameters bleiben bestehen.

`REG 100` wird zu

`REG` **`rStückzahl`**

Eine Symboldatei wird nach folgendem Schema angefertigt:

- ein gültiges Symbol beginnt in der ersten Spalte. Beginnt eine Zeile mit einem ";" oder einem Leerzeichen " " wird diese als Kommentarzeile interpretiert.

`rStückzahl`

- diesem symbolischen Parameter `rStückzahl` wird jetzt sein numerisches Äquivalent zugeordnet. Dieses befindet sich in der selben Zeile und ist durch mindestens ein Leerzeichen " " vom symbolischen Namen zu trennen.

`rStückzahl` **`100`**

- jetzt kann noch ein Kommentar angefügt werden. Er ist durch mindestens ein Leerzeichen oder Semikolon vom Parameter zu trennen.

```
rStückzahl    100 ;Kommentar: Dem Symbol
                  "rStückzahl" wird der numerische
                  Parameter "100" zugeordnet.
```

Soweit geschieht die Definition im Symbol-Editor. Existiert nun im Programm (im Programm-Editor) eine Befehlszeile

REG 200 = REG rStückzahl

so wird dem Register REG 100 das physikalisch in der Steuerung existiert die symbolische Bezeichnung "Stückzahl" zugewiesen.

Nachdem das Symbollisting erstellt wurde, kann dieses im Programmeditor die Programmerstellung folgendermaßen unterstützen. Im Programmeditor wird zum Beispiel ein Register mit symbolischer Namenszuweisung eingegeben. Nach der Eingabe des Kürzels "RE" erscheint ein Fenster, in welchem die Registernummern bzw. die symbolische Namens-zuweisung erfolgt. Gibt man hier den Anfangs-buchstaben des symbolischen Namens und danach die Tastenkombination SHIFT-? ein, so erscheint ein Fenster aller symbolischen Namen, welche mit diesem Buchstaben beginnen. Hier können Sie jetzt bequem über die Cursortasten die entsprechende Bezeichnung auswählen. Somit ist sichergestellt, dass einmal getippte Symbolik nicht ständig neu eingegeben werden muss.

Beispiel für eine Symboldatei:

```
Symbollisting von "prog01" V1, 28.04.1996  15:13      Seite 1


JETTER PROZESS-SPS NANO--B

Kunde/Projekt: sympas handbuch
Ort          : Ludwigsburg
Datum        : 28.04.1996  15:13
Version      : 1


;**** TASK ****
;
tSteuer_Task     0       ;Der Task steuert den Programmablauf
tAuto_Task       1       ;Automatik-Task
tAnzeige_Istpos  2       ;Anzeige-Task Istposition
tNOTAUS          3       ;NOTAUS-Task


;**** MARKEN ****
;
sSchleife        40      ;Marke 40
sFahre_links       41        ;Marke 41: Programmsequenz Fahrt nach
links
sFahre_rechts      42        ;Marke 42: Programmsequenz Fahrt nach
rechts
sRef_Fahrt       43      ;Marke 43: Programmsequenz Referenzfahrt


;**** EINGÄNGE ****
;
eNot_Aus_Schalte 105     ;Schalter Notausbedingung
eSchutztür       106     ;Schalter Notausbedingung Schutztür offen
eAutomatik       107     ;Schalter Automatik/Hand
eStart_Taster    108     ;Taster "Start'
eStop_Taster     201     ;Taster "Stop"
eReffahrt_Taster 202     ;Taster "Referenzfahrt"


;**** REGISTER ****
;
rSM_Status       11100   ;Statusregister SM-Ansteuerung
rKommandoregiste 11101   ;Kommandoregister SM-Ansteuerung
rSM_Geschwindigk    11103             ;Sollgeschwindigkeitsregister
SM-Ansteuerung
rIstposition     11109   ;Istpositionsregister SM-Ansteuerung


;**** MERKER ****
;
mReferenz_OK     1       ;Merker: Referenzfahrt hat stattgefunden
mAutomatik_Task  2       ;Steuermerker Automatik-Task
mPfeil_Links     217     ;LCD Cursortaste links
mPfeil_Rechts    218     ;LCD Cursortaste rechts
```

Hinweis:

Der Pfad der Symboldatei muss identisch zum Pfad der zugehörigen Programmdatei sein. Ausführlich wird der Zusammenhang zwischen Programmdatei und Symboldatei an dem Demonstrationsbeispiel in *Kapitel 6. Demonstrationsbeispiel: Handling-System* dargestellt.

3.7 INCLUDE-Dateien

SYMPAS-Programme respektive Programmteile können mit Hilfe des INCLUDE-Befehles in ein bestehendes SYMPAS-Programm eingebunden werden. Somit kann aus einer Bibliothek von SYMPAS-Modulen ein Gesamtprogramm zusammengestellt werden. Sowohl in Programm- als auch in Symboldateien können #INCLUDE-Befehle enthalten sein. Die maximale Anzahl der INCLUDE-Dateien ist auf 32 je Editor begrenzt.

Programme strukturieren

Funktions-bibliotheken

Maximale Programm-länge vergrößern

INCLUDE-Dateien werden eingesetzt um

- Programme modular zu strukturieren
- Funktionen in INCLUDE-Dateien zu Befehlbibliotheken zusammenzufassen
- Einschränkungen der maximalen Programmlänge in den Editoren zu umgehen

3.7.1 INCLUDE-Dateien im Programm-Editor

#INCLUDE-Befehl als Platzhalter für den Text der INCLUDE-Datei

INCLUDE-Dateien werden mit dem #INCLUDE-Befehl in den Text der Hauptdatei eingefügt. Diese Befehlszeile fungiert als Platzhalter für den Programmtext, der in der INCLUDE-Datei steht. Genau dieser in der INCLUDE-Datei enthaltene Programmtext befindet sich logisch an der Stelle in der Hauptdatei, an welcher der #INCLUDE-Befehl mit dem Namen der entsprechenden Datei eingefügt ist.

Hauptdatei definieren

32 INCLUDE-Dateien sind möglich

Im Menü "Datei / Hauptdatei..." wird die Hauptdatei definiert. In dieser Hauptdatei werden die INCLUDE-Dateien eingefügt.

Keine Schachtelung von INCLUDE-Dateien

Abbildung 10: In der Hauptdatei können bis zu 32 INCLUDE-Dateien definiert werden

Der #INCLUDE-Befehl

Der #INCLUDE-Befehl bindet die INCLUDE-Datei ein

Mit dem #INCLUDE-Befehl wird die INCLUDE-Datei logisch in den Programmtext der Hauptdatei eingebunden.

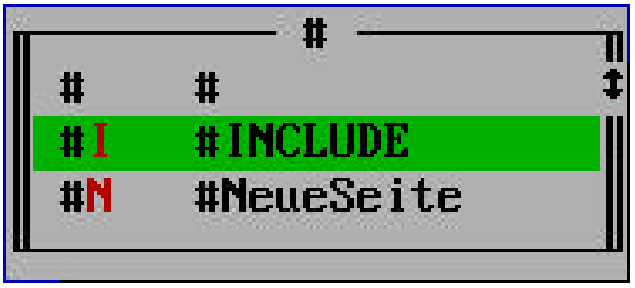


Abbildung 11: Der #INCLUDE-Befehl bindet die INCLUDE-Datei ein

Hinweis:

- 32 INCLUDE-Dateien können in der Hauptdatei definiert werden.
- Die Schachtelung von INCLUDE-Dateien ist nicht zulässig. In einer INCLUDE-Datei dürfen keine weiteren INCLUDE-Dateien angegeben werden.

Beispiel:

Das Ergebnis sieht dann zum Beispiel so aus:

Hauptdatei: GESAMT.PNB

INCLUDE-Datei PUMPE01

```
Projekt  Datei  Edit  Block  Transfer  Listing  Monitor  Oszi  Spezial
Länge: 35 (1.7%)                    \...\GESAMT.PNB
 22:       ANZEIGE_TEXT [#0, cp=1, "Programm 2 läuft"]
 23:       A aGebläse                          ;Gebläse einschalten
 24:       WARTEZEIT 10                        ;1 Sekunde warten
 25:       POS [Achse=1, Pos=R(rAblage), v=R(rAutomatik)] ;Teil ablegen
 26:    SOBALD
 27:       HALTACHSE Achse=1
 28:      DANN
 29:       -A aGebläse                         ;Gebläse aus
 30:       WARTEZEIT 10                        ;Warte 1 Sekunde
 31:       ANZEIGE_TEXT [#0, cp=1, "Achse in Pos"]
 32: TASK tPumpenSteuerun
 33: #INCLUDE  PUMPE01
 34:       SPRUNG tPumpenSteuerun
Programmende
F1 Hilfe  Shift-F9 Syntax-Check  Ctrl-F9 Übertragen an NANO-B  F10 Menü
```

Abbildung 12: In die Hauptdatei GESAMT.PNB wurde die INCLUDE-Datei PUMPE01 eingefügt

Mit der Pickliste wählen welche Datei im Programm-Editor erscheint

Die Pickliste

Mit der Pickliste (Datei / Pickliste...) kann komfortabel zwischen der Hauptdatei und den INCLUDE-Dateien umgeschaltet werden. Dazu laden Sie sich zunächst alle notwenigen Dateien (Neue Datei) in die Pickliste. Danach erscheint immer die Datei im Programm-Editor die Sie in der Pickliste anklicken.

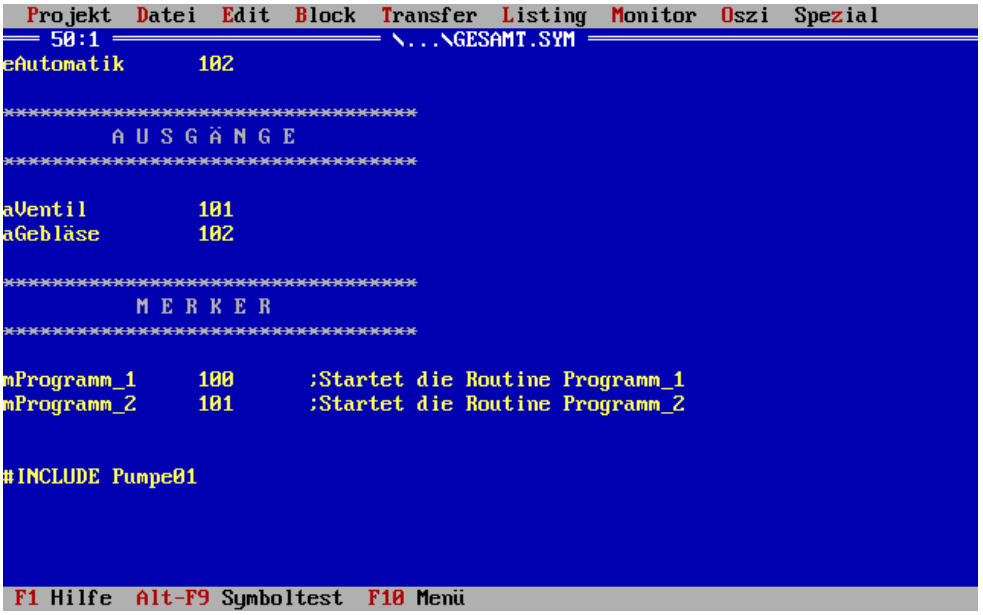
Abbildung 13: Mit "Neue Datei" Dateien in die Liste aufnehmen. Die Datei, die Sie in der Liste anklicken erscheint auf dem Bildschirm.

3.7.2 INCLUDE-Dateien im Symbol-Editor

Mit #INCLUDE lassen sich weitere Symbol-dateien in den Symboltext einfügen

In eine Symboldatei können weitere Dateien als INCLUDE-Dateien eingebunden werden. So kann eine Bibliothek vorgefertigter, anwendungsbezogener Symboldateien angelegt werden, die bei Bedarf in den Symboltext eingebunden werden. Im Symbol-Editor steht zum Beispiel folgende Zeile:

#INCLUDE-Befehl



INCLUDE-Dateien können mit Pfad angegeben werden

Die INCLUDE-Datei kann mit Pfad angegeben werden. Eine Verschachtelung in mehrere Ebenen ist nicht möglich. Die entsprechenden Informationen werden den Dateien auf Festplatte entnommen und erscheinen nicht im Symbol-Editor.

Hinweis:

- 32 INCLUDE-Dateien können in der Symboldatei definiert werden.
- Die Schachtelung von INCLUDE-Dateien ist nicht zulässig. In einer INCLUDE-Datei dürfen keine weiteren INCLUDE-Dateien angegeben werden.

INCLUDE-Dateien:

Dateiname und Datei selber müssen in Ordnung sein

Bei der Verwendung von INCLUDE-Dateien sind zwei Punkte zu beachten. Einmal muss der Dateiname mit dem auf der Festplatte existenten identisch sein. Zum zweiten muss der Inhalt der entsprechenden INCLUDE-Datei in Ordnung sein. Sollten sich Fehler in der INCLUDE-Datei oder dem Dateinamen befinden, ist das Verlassen des Symbol-Editors nicht mehr möglich. Die entsprechenden Includezeilen sind im Fehlerfall auszukommentieren.

Hinweis:

Kann der Symbol-Editor wegen einer fehlerhaften INCLUDE-Datei nicht mehr verlassen werden, sind die Fehler in der INCLUDE-Datei auszukommentieren (mit ;). Die dann fehlerbereinigte INCLUDE-Datei kann verlassen werden.

Aufbau INCLUDE-Datei ist identisch zum Aufbau Symboldatei

Abschließend sei darauf hingewiesen, dass der Aufbau einer INCLUDE-Datei identisch zu einer Symboldatei ist. Somit kann jede existente, und auf Festplatte gesicherte, Symboldatei als INCLUDE-Datei in eine weitere Symboldatei eingebunden werden; diese darf jedoch keine weiteren INCLUDE-Dateien enthalten.

3.8 Fehlermeldungen

SYMPAS unterstützt mit folgenden Fehlermeldungen den Programm-Editor, den Symbol-Editor und die Programmierung der PROZESS-SPS-Steuerungen im allgemeinen.

Mit (SHIFT) (F9) den Syntax-Check aufrufen

Die Überprüfung des Programmes und der zugehörigen Symboldatei geschieht mit dem Syntax-Check. Dieser wird (sofern aktiviert; *Kapitel 3.5.10 Das Menü "Spezial"*) vor der Übertragung des Programmes in die Steuerung, oder im Programm-Editor mit der Tastenkombination SHIFT-F9 aktiviert.

Mit (ALT) (F9) den Symboltext überprüfen

Außerdem existiert im Symbol-Editor die Möglichkeit, mit der Tastenkombination ALT-F9, den Symboliktext auf syntaktische Richtigkeit überprüfen zu lassen.

Ein Fenster gibt über die Art und Anzahl der Fehler Auskunft. Nach dem Bestätigen mit ENTER <+ steht der Cursor an der Fehlerstelle im Programm-Editor. Eine rote Fehlerzeile gibt weitere Informationen.
Wurden in dem oben erwähnten Fenster mehrere Fehler gemeldet, so können Sie durch entsprechend häufiges Aufrufen des Syntax-Check mit SHIFT-F9 einen Fehler nach dem anderen korrigieren. Eine kontextbezogene Hilfestellung erscheint jeweils in der roten Fehlerzeile.

Dabei sind folgende Fehlermeldungen möglich:

Fehlermeldungen Symbolfehler:

1 Symbol nicht gefunden

Ein im Programmtext existentes Symbol wurde in der Symboldatei nicht definiert.

2 Symbol existiert bereits

Ein Symbol wurde mehrmals definiert. Z.B. zweimal EINGANG102.

3 Ungültiger Parameter

Einem Symbol wurde im Symbol-Editor ein ungültiger Parameter zugewiesen: numerischer Parameter: max. ± 8388606.

4 Außerhalb zulässigem Wertebereich

Der Befehlsparameter befindet sich außerhalb des zulässigen Wertebereiches.

5 Doppeltes Symbol mit unterschiedlichem Datentyp

Ein und dasselbe Symbol wurde für einen numerischen und einen Textparameter definiert.

6 Textausdruck nicht korrekt

Ein ungültiger Stringparameter wurde definiert:
Stringparameter müssen in " oder ' oder # Zeichen eingeschlossen werden. Ihre Länge darf maximal 24 Zeichen betragen.

Fehlermeldungen Syntax-Check:

9 Funktionsdefinition (X) nicht gefunden

Eine im Programmtext aufgerufene Funktion wurde im Programmkopf nicht definiert.

10 Funktionsaufruf ungleich Definition

Anzahl der Parameter in Aufruf und Definition ungleich.

11 Zu viele Marken

Zu viele relative Marken (von SYMPAS selbst verwaltet).

13 1. Anweisung muss TASK0 sein

Der erste Befehl in einem PROZESS-SPS-Programm muss TASK 0 sein.

14 TASK(X) existiert bereits

Der Task mit der Nummer "X" wurde im Programm bereits definiert.

15 MARKE(X) existiert bereits

Die Marke mit der Nummer "X" wurde im Programm bereits definiert.

16 TASK nicht gefunden

Ein als TASKBREAK, TASKCONTINUE, TASKRESTART definierter Task existiert nicht.

17 TASK(X) ergibt keine Endlosschleife

Der Task mit der Nummer "X" ist in sich nicht durch einen Befehl SPRUNG abgeschlossen. Jeder Task muss in sich geschlossen sein, eine Endlosschleife bilden.

20 Sprungmarke nicht gefunden

Die dem SPRUNG(X) zugeordnete MARKE(X) oder TASK(X) existieren nicht.

21 SPRUNG in anderen TASK

Es ist nicht möglich Sprungbefehle in andere Parallelzweige vorzunehmen.

22 SPRUNG in Funktion erlaubt

Sprünge in Funktionen hinein (von außen) sind nicht zulässig.

23 SPRUNG aus Funktion nicht erlaubt

Sprünge aus Funktionen hinaus (nach außen) sind nicht zulässig.

24 Globale Unterprogramme nur am Programmtextende

Es wird zwischen lokalen und globalen Unterprogrammen unterschieden. Lokale Unterprogramme werden nur von einem Task genutzt, und befinden sich am Taskende.
Globale Unterprogramme werden von verschiedenen Tasks in Anspruch genommen und müssen sich am Ende des gesamten Programmtextes, also hinter dem

Text des letzten Tasks befinden. Sollten Sie diese Struktur nicht einhalten können oder wollen, kann der Syntax-Check mit dem entsprechenden Schalter deaktiviert werden (Globale UPs testen). Es erscheint jetzt keine Fehlermeldung; eine abweichende Programmstruktur ist lauffähig, aber weniger übersichtlich.

25 Nur 20 Unterprogrammebenen zulässig

Es sind 20 Unterprogrammebenen zulässig.

26 Rücksprung ohne Unterprogrammaufruf

SYMPAS ist auf einen Befehl RÜCKSPRUNG ohne vorhergehenden UNTERPROGRAMM-Aufruf gestoßen.

27 Hauptprogramm geht in Unterprogramm über

Das Hauptprogramm läuft in ein Unterprogramm hinein.

28 Unterprogramm nicht gefunden

Zu einem UNTERPROGRAMM(X) Aufruf existiert keine entprechende MARKE(X).

30 SOBALD hier nicht erlaubt

Programm-Syntax lässt an dieser Stelle keinen Befehl SOBALD zu.

31 FALLS hier nicht erlaubt

Programm-Syntax lässt an dieser Stelle keinen Befehl FALLS zu.

32 SONST ohne FALLS..DANN

SYMPAS ist auf SONST ohne vorhergehende FALLS..DANN Anweisung gestoßen.

33 SONST, FALLS, SOBALD, DANN zu weit weg von FALLS

Der Programmtext innerhalb einer bedingten Entscheidung, also zwischen FALLS und DANN, oder aber zwischen FALLS und SONST oder der zu FALLS gehörenden Abschluss-Anweisung - DANN, FALLS, SOBALD - ist zu lang. Abhilfe schaffen Sie durch Kürzen des entsprechenden Programmtextes.

34 FALLS, SOBALD, DANN zu weit weg von SONST

Der Programmtext innerhalb einer bedingten Entscheidung, hier zwischen SONST und der entsprechenden Abschluss-Anweisung - DANN, FALLS, SOBALD - ist zu lang.

35 DANN erwartet

An dieser Stelle erwartet der Compiler ein DANN.

37 Nur in Eingangsbedingung erlaubt

Diese Operatoren sind nur zwischen FALLS (SOBALD) und DANN erlaubt.

38 Nur als Ausgangsanweisung erlaubt

Diese Operatoren sind nur nach DANN und SONST erlaubt.

39 Fehler innerhalb des Ausdrucks

Die Operatoren =, +, -, *, /, WODER, WUND, WXODER, ISTPOS, ZD, ZB, ZH wurden an dieser Stelle im falschen Zusammenhang gebraucht.

41 Zahl oder Variable erwartet

An dieser Stelle erwartet der Kompiler eine Zahl oder Variable.

42 "=" erwartet

An dieser Stelle erwartet der Kompiler ein Gleichheitszeichen.

43 Boole'scher Ausdruck erwartet

An dieser Stelle erwartet der Kompiler einen Boole'schen Ausdruck.

44 Arithmetischer Vergleichsoperator erwartet

An dieser Stelle erwartet der Kompiler einen arithmetischen Vergleichsoperator.

45 ")" ohne "("

Klammern nicht vollständig gesetzt.

46 ")" erwartet

Klammern nicht vollständig gesetzt.

47 11. Klammerebene nicht erlaubt

Maximal 10 Klammerebenen sind zulässig.

50 Funktionsdefinition vor TASK 0

Funktionen müssen vor dem ersten Task (TASK 0) definiert werden.

51 END_DEF ohne DEF_FUNKTION

END_DEF wurde ohne DEF_FUNKTION spezifiziert. END_DEF schließt eine Funktionsdefinition mit DEF_FUNKTION ab.

52 END_DEF erwartet

Eine Funktionsdefinition wurde nicht mit END_DEF abgeschlossen.

53 RÜCKSPRUNG erwartet

Vor einem END_DEF fehlt ein RÜCKSPRUNG.

Sonstige Fehler:

55 Unbekannter Befehl

Der Kompiler ist auf einen unbekannten Befehl gestoßen.

56 Steuerungs-Speicher voll!

Der Programmspeicher der Steuerung ist zu klein für das zu übertragende Programm.

57 "}" ohne "{"

Kommentarklammern nicht vollständig gesetzt.

58 "}" nicht gefunden

Kommentarklammern nicht vollständig gesetzt.

59 Datei kann nicht geöffnet werden

DOS-Fehler im Zusammenhang mit Include-Dateien ('Datei nicht gefunden' oder 'Too many open Files').

60 Out of Memory

Die Include-Datei passt nicht in den PC-Speicher.

61 Keine Verschachtelung zulässig

In einer Include-Datei befindet sich ein weiteres #INCLUDE.

62 Maximal 8 Include-Dateien erlaubt

Maximal dürfen 8 Include-Dateien in einer Hauptdatei definiert werden.

63 Include-Datei nur erlaubt wenn Hauptdatei definiert

Include-Dateien können nur in einer Hauptdatei definiert werden.

64 Unerwartetes Dateiende

Systemfehlermeldung.

65 Sprunglänge größer 32kByte

Eine von SYMPAS verwaltete Marke ist zu weit vom SPRUNG-Befehl entfernt. Entfernung reduzieren.

66 Steuerungs-Version x.xx wird verwendet

Ein Befehl wurde verwendet, der eine neuere als in "Einstellungen..." definierte Betriebssystemversion erfordert.

3.9 Dateien, Extensions, etc.

Die Zusammenstellung der Dateien, welche sich im Lieferumfang von SYMPAS befinden, entnehmen Sie der aktuellen LESEMICH Datei. Diese Datei befindet sich auf der SYMPAS-Diskette und kann auf DOS-Ebene ausgelesen werden.

Über alle Dateien die beim Arbeiten mit SYMPAS von diesem generiert werden gibt folgende Übersicht Auskunft.

NAME.PPE (PASE-E), **NAME.PPM** (MIKRO),
NAME.PPD (DELTA), **NAME.PNA** (NANO-A),
NAME.PNB (NANO-B)

Das sind die Bezeichnungen für die Programmdateien, in welchen der Programmtext gespeichert wird.

NAME.BKE (PASE-E), **NAME.BKM** (MIKRO),
NAME.BKD (DELTA), **NAME.BNA** (NANO-A),
NAME.BNB (NANO-B)

So werden die Sicherheitskopien für die entsprechenden Programmdateien bezeichnet.

NAME.SYM

Das sind die Symboldateien zu den Programmdateien. Der Name der Programmdatei muss nicht mit dem Namen der Symboldatei identisch sein. Die Symboldateien müssen sich in dem Verzeichnis der zugehörigen Programmdateien befinden.

NAME.BKS

So werden die Sicherheitskopien der Symboldateien bezeichnet.

SYMPAS.CFG

Die Konfigurationsdatei. In dieser Datei werden alle Einstellungen festgehalten, die mit der Auswahlzeile "Einst. speichern" im Pull-Down-Menü "Spezial" angesprochen werden.

SYMPAS.DSK

Diese Deskdatei wird von SYMPAS beim Aufstarten berücksichtigt wenn der Schalter "Umgebung automatisch speichern" in der Datei SYMPAS.CFG auf EIN gespeichert wurde. Nach den Vorgaben der SYMPAS.DSK wird die Umgebung beim Aufstarten restauriert.

NAME.DSK

Die Deskdatei. In dieser Datei werden alle Einstellungen festgehalten, die mit der Auswahlzeile "Umgebung sichern" im Pull-Down-Menü "Datei" angesprochen werden. Hier kann sich der Benutzer neben der Datei SYMPAS.DSK noch andere Dateien zur Speicherung der Umgebungs-Konfiguration anlegen.

NAME.SUE (PASE-E), **NAME.SUM** (MIKRO),
NAME.SUD (DELTA), **NAME.SNA** (NANO-A),
NAME.SNB (NANO-B)

In dieser Datei werden die Einstellungen des Inbetriebnahme-Bildschirmes festgehalten.

NAME.LST

In dieser Datei werden Druckerausgaben, welche in eine Datei umgeleitet werden gespeichert.

NAME.RT

Systemdatei, deren Existenz für die Funktion des Indexfensters im Inbetriebnahme-Bildschirm notwendig ist.

NAME.EP (PASE-E), **NAME.EPR** (PASE-M),
NAME.EPD (DELTA), **NAME.ENA** (NANO-A),
NAME.ENB (NANO-B)

Objektdatei. Diese Datei wird mit der Auswahlzeile „Editor -> Datei.EP“ im Pull-Down-Menü „Transfer“ erzeugt.

NAME.DA

Register- und Merkerbereichs Datei. Mit der Auswahlzeile "Register -> Datei.DA..." im Pull-Down-Menü "Transfer" können Sie selbst definierte Register- und Merkerbereiche der Steuerung auf Diskette oder Festplatte, in die oben genannten Dateien, sichern.

NAME.SIT

Sympas Include Table enthält bereits übersetzte Symbole in binärer Form.

3.10 Verschiedenes

3.10.1 Indirekte Adressierung

Mit (CTRL) (R) oder (SPACE) die indirekt Level im Definitions-fenster aktivieren

Indirekte Adressierung wird in dem geöffneten Fenster definiert, in welchem der oder die Parameter der Befehle definiert werden. Hierzu sind die Tastenkombination CTRL-R oder die Taste SPACE zu betätigen. Jetzt erscheint vor der Parameterzeile ein "R" nach zweimaligen Betätigen "RR", sofern in diesem Befehl doppelt indirekte Adressierung möglich ist.

3.10.2 Kommentare

Es gibt drei Möglichkeiten Kommentare im Programm-Editor einzugeben.

- drücken Sie die Taste ";" und geben den entsprechenden Kommentar als Programmzeile ein. Quittieren mit ENTER < +.

- um rechts neben dem Programmtext Kommentare anzubringen drücken Sie die Cursortaste → und geben den entsprechenden Kommentar ein.

- außerdem existiert die Möglichkeit geschweiften Klammern { ... } Kommentare im Programmtext anzubringen. Alle Zeichen, welche sich zwischen den geschweiften Klammern befinden, werden vom Kompiler als Kommentar interpretiert und nicht übersetzt (Auskommentieren von Programm-passagen).

3.10.3 Aufruf mit dem Schalter /o (Laptop, Notebook)

SYMPAS ist in Overlay-Technik programmiert und belegt somit möglichst wenig Platz im Arbeitsspeicher Ihres PC. Allerdings macht diese Technik häufige Zugriffe auf die Festplatte des PC notwendig. Dies ist im Normalfall unkritisch, Sie werden von diesen Aktivitäten wenig mitbekommen.

Sollten Sie SYMPAS jedoch auf einem Diskettenlaufwerk (Zugriffszeit problematisch), oder in einem LAP-Top oder Notebook (bei Akkubetrieb Reduktion der Betriebszeit) installiert haben, so können Sie SYMPAS auf DOS-Ebene mit

SYMPAS /o

aufrufen. Jetzt wird der Overlay-Puffer im Arbeitsspeicher des PC so groß gemacht, dass keine Zugriffe auf Festplatte mehr nötig werden, sich sämtliche Programmteile von SYMPAS ständig im RAM befinden. Somit sind die oben angesprochenen Probleme vom Ansatz her ausgeräumt.

SYMPAS /oxxxx

Overlaybereich um xxxx Bytes vergrößern.

Alternativ zur Verwendung des Schalters können 20 freie 16kByte Blöcke EMS auf dem PC installiert werden (siehe DOS-Handbuch).

3.10.4 Das Programme NOSYMPAS.EXE

Das Programm NOSYMPAS.EXE ist eine reduzierte Versionen des Programmes SYMPAS.EXE).
Die Programme sind für Endkunden bestimmt, denen nur sehr eingeschränkte Manipulationsmöglichkeiten eingeräumt werden sollen. Programme können geladen werden (von Festplatte und aus dem Steuerungs-RAM) jedoch nicht editiert. Wie gewohnt ist der Zugriff auf den Inbetriebnahme-Bildschirm vorgesehen, hier sind jedoch nach dem Aufruf zunächst keine Datenmanipulationen möglich. Nachdem die Sperre im Dialog "Einstellungen" entriegelt wurde, können die Registerwerte, Eingänge, Ausgänge, etc. der PROZESS-SPS-Steuerung geändert werden.

Bei der Unterstützung eines Kunden kann auch folgender Punkt hilfreich sein. Programmname (z.B. *.PNB) und/oder Setupdateiname (z.B. *.SNB) können bereits beim Start von SYMPAS von DOS-Ebene aus angegeben werden.

```
NOSYMPAS /NANOB PROG01.PPE SETUP05.SUE
```

Dabei geht NOSYMPAS.EXE wie folgt vor:

- *.PNB laden
- *.SNB laden
- Inbetriebnahme-Bildschirm aktivieren, wobei die Werteingabe bzw. Änderung zunächst gesperrt ist. (Mit Auswahlzeile "Eingabe sperren" im Pull-Down-Menü "Spezial / Einstellungen..." freigeben.)

3.10.5 Umschalten zum DOS-Bildschirm

Die Tastenkombination Alt-F5 holt den DOS-Bildschirm in den Vordergrund, der vor dem Aufruf von SYMPAS beziehungsweise nach dem Verlassen der DOS-Shell mit EXIT aktuell war. Ein beliebiger Tastendruck wechselt wieder zum SYMPAS-Bildschirm.

3.10.6 Password

Aktivierung

Um ein Password zu definieren ist SYMPAS der Aufrufparameter /p (/pv) mitzugeben. Geschieht dies, wird zu Beginn von SYMPAS das Password in einem Fenster eingegeben. Es kann zwischen zwei Varianten gewählt werden:

/p = die eingegebenen Zeichen werden nicht lesbar dargestellt; zur Sicherheit muss die Eingabe einmal wiederholt werden.

/pv = die eingegebenen Zeichen können gelesen werden; die Eingabewiederholung entfällt.

Definitionen

- ein Password darf maximal 8 Zeichen, muss mindestens 5 Zeichen lang sein.
- erscheint bei der Eingabe des Passwords keine Fehlermeldung so ist das Password gültig.

Anwendung

- das Password wird verschlüsselt in den Programmkopf einer Objektdatei oder direkt in die Steuerung (Ctrl-F9) aufgenommen.
- wird mit der Auswahlzeile "Datei.ENB -> Editor" ein Programm rückübersetzt und wurde zuvor ein Password definiert und in den Programmkopf aufgenommen, so muss dieses jetzt angegeben werden, sonst ist keine Übersetzung des Programmes aus einer EPROM-Datei möglich. Ist das Password, welches beim Programmstart von SYMPAS eingegeben wurde, identisch mit dem Password im Programmkopf des Steuerungsprogrammes so wird keine erneute Passwordeingabe gefordert.

3.10.7 SYMPAS ab 3.09 und MIKRO vor 2.10

Wird eine SYMPAS-Version ab 3.09 zusammen mit der MIKRO mit einer Betriebssystemversion kleiner 2.10 betrieben so ist der Syntax-Check von SYMPAS nicht in der Lage eine Überschreitung der 3 maximal zulässigen Unterprogrammebenen zu erkennen.

3.10.8 SYMPAS und PASE-J (bis Version 4.04)

Wird eine Steuerung des Typs PASE-J verwendet so ist SYMPAS mit folgender Aufrufzeile zu starten:

SYMPAS ***/J***

3.10.9 SYMPAS im Netzwerk (PASE-E bis Version 4.04)

Wird SYMPAS im Netzwerk eingesetzt so ist mit folgender DOS-Kommandozeile

```
SET SYMPAS_CONFIG=Pfad
```

der Pfad für die Konfigurationsdatei SYMPAS(_M).CFG anzugeben.

3.10.10 Sonstige Kommandozeilenparameter (Aufrufschalter)

/L	kontrastreiche Farbpalette für Laptops.
/S	Symbole in Auswahlfenstern alphabetisch sortiert.
/B	Eingabefenster erscheinen unterhalb der Cursorzeile.
/Bxxxx	Eingabe einer festen Baudrate (die in der Steuerung eingestellt ist).
/PASEE+	Einstellung SYMPAS: PASE-E PLUS
/MIKRO	Einstellung SYMPAS: MIKRO
/DELTA	Einstellung SYMPAS: DELTA
/NANOA	Einstellung SYMPAS: NANO-A
/NANOB	Einstellung SYMPAS: NANO-B (Default wenn keine Steuerung über Schalter definiert ist.

II. SYMPAS-Programmierung

1. Überblick

Zur Lösung einer **Steuerungsaufgabe** mit einer PROZESS-SPS gehört neben der geeigneten Zusammenstellung der Steuerungshardware ("Hardware - Konfiguration") ein auf die Problemstellung zugeschnittenes Programm. Das Programm veranlasst die Steuerung dazu, die gestellte Steuerungsaufgabe zu erfüllen.

SYMPAS orientiert sich am Bewegungs-ablauf der Maschine

Die Programmiersprache der PROZESS-SPS ist in gewissem Sinne unkonventionell. Sie orientiert sich am Bewegungs-ablauf der zu steuernden Maschine bzw. des zu kontrollierenden Prozesses und nicht, wie sonst üblich, z.B. an einem Kontaktplan.

Dies ist ein gewaltiger Unterschied: Zur Programmierung einer Steuerungsaufgabe ist es **nicht** notwendig, sich zunächst über die Lösung mit herkömmlichen Mitteln (Schütze, Hilfsschütze ...) Gedanken zu machen. Vielmehr kann der Bewegungsablauf nahezu direkt in die Programmiersprache der PROZESS-SPS umgesetzt werden.

Klartext-sprache beschreibt den Prozess

Zusätzlich zu der reinen Beschreibungssprache sind auch Fließkomma-Arithmetik, Datenverwaltung und Multitasking mit bis zu 32 Tasks vorhanden.

Bei der Bearbeitung des Programms unterscheidet sich die PROZESS-SPS von konventionellen Steuerungen dadurch, dass **kein** zyklischer Speicherdurchlauf stattfindet. Dadurch ist die Reaktionszeit unabhängig von der Programmlänge, da nur die zur Fortsetzung des Steuerungsablaufs notwendigen

Eingangsbedingungen ständig geprüft werden, alle anderen jedoch nicht.

2. Grundsätzliches zur Programmierung

2.1 Prinzipieller Programmaufbau

Da sich die Programmiersprache von den genormten SPS-Sprachen deutlich unterscheidet, hat dies entsprechende Auswirkungen auf den prinzipiellen Programmaufbau.

Kein zyklischer Speicher-durchlauf sondern Multitasking

Der grundlegendste Unterschied zu herkömmlichen SPS-Sprachen besteht darin, dass hier kein zyklischer Speicherdurchlauf und somit keine quasiparallele Abarbeitung aller Eingangsbedingungen stattfindet, sondern jeder der max. 32 voneinander unabhängigen (und somit quasigleichzeitig arbeitenden) Programmteile in sich sequentiell arbeitet.

Damit wurde der Tatsache Rechnung getragen, dass in der Regel in einer Anlage zwar mehrere parallele Teilvorgänge gesteuert werden müssen, diese jedoch in aller Regel in sich sequentiell ablaufen.

Der Programmaufbau sollte sich daher möglichst direkt an dieser Aufteilung orientieren. Es empfiehlt sich, ein Grundprogramm zu definieren, das als Hauptprogramm fungiert und über Merker (Flags) die weiteren "Unter"-Programme aktiviert und verknüpft.

Zusätzlich sind gewöhnlich noch Parallelprogramme zur Verarbeitung asynchroner Kommandos, beispielsweise von Bedienelementen, vom Leitrechner, oder vom VIADUKT notwendig.

Da bei dieser Sprache die Verarbeitungszeit nicht von der Programmlänge, sondern von der Anzahl der

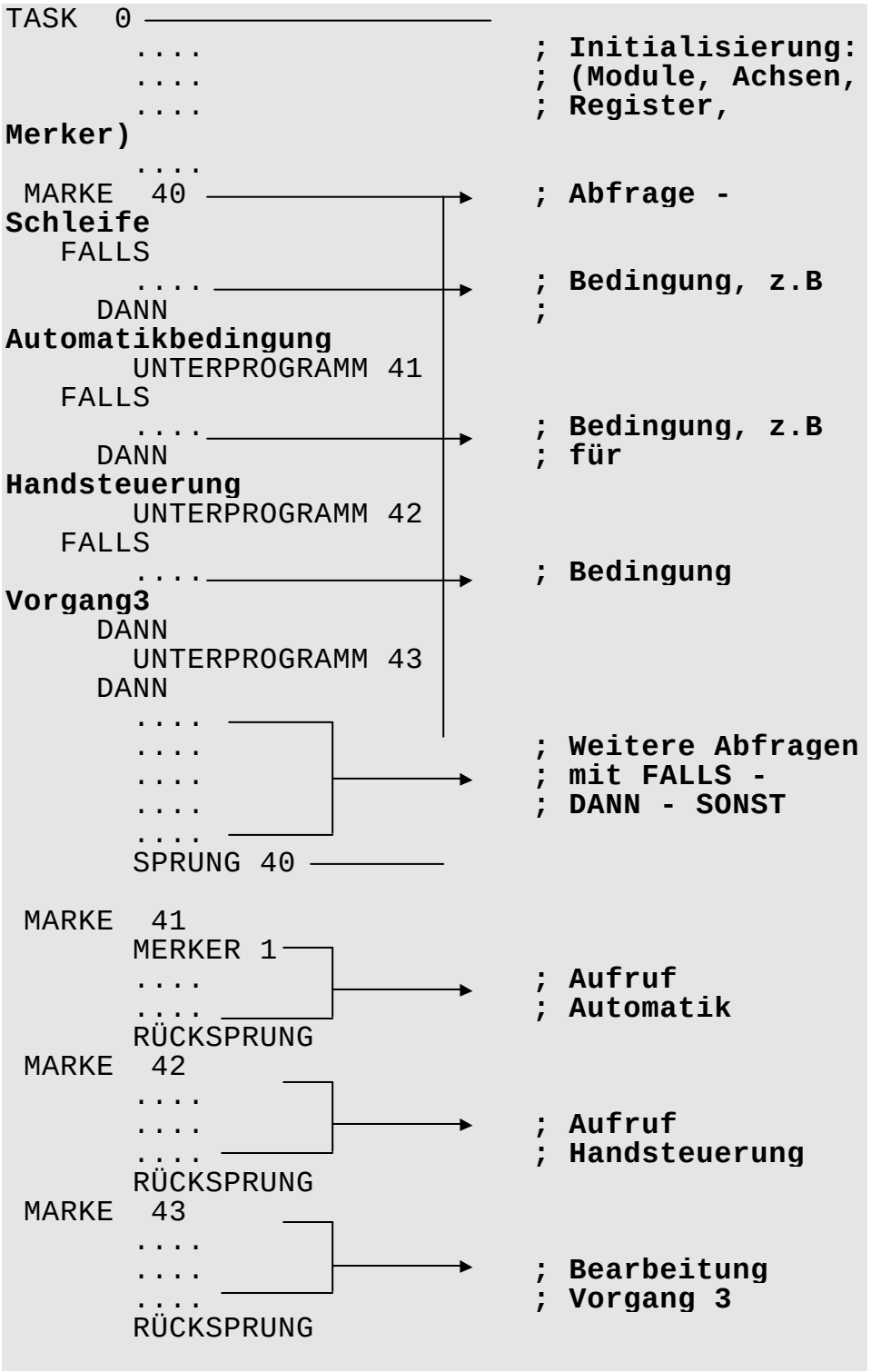
verwendeten Parallelprogramme (Task) abhängt, sollten davon nicht unnötig viele benutzt werden.

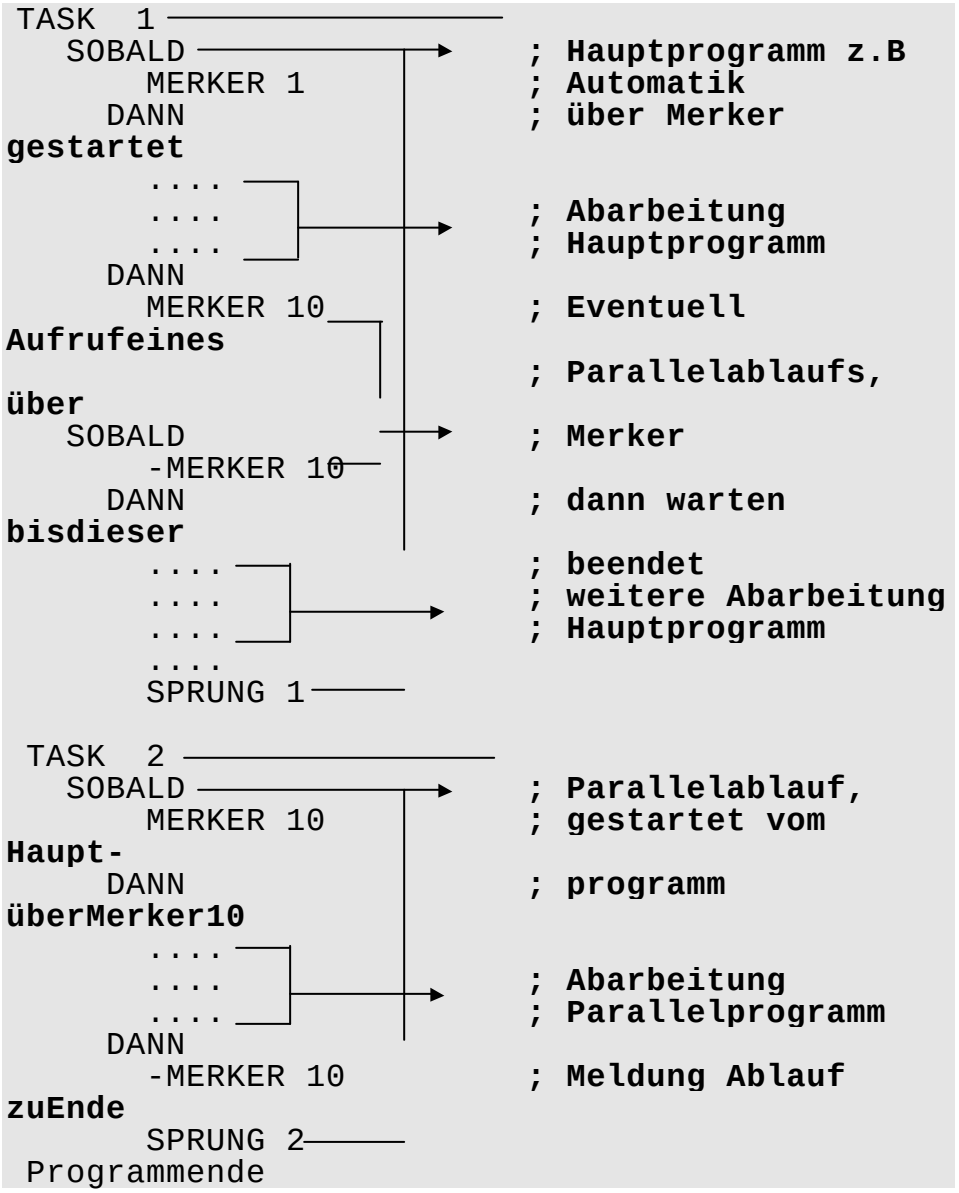
Grundbefehl SOBALD . . . DANN . . .

Der Grundbefehl **SOBALD**...(Bedingung)... **DANN** ...(Ausgabe).. kann deshalb besonders gut zur direkten Ablaufbeschreibung verwendet werden, da (im Gegensatz zum IF/THEN-Befehl bei BASIC oder PASCAL) **gewartet** wird, bis die Bedingung erfüllt ist, und erst danach die Ausgabe erfolgt. Dies bedeutet, dass im Gegensatz zu herkömmlichen SPS-Sprachen die Reihenfolge der Eingabe gleichzeitig der Reihenfolge des Ablaufs entspricht.

Soll jedoch nicht gewartet werden, sondern nur eine logische Entscheidung erfolgen, so wird der Befehl **FALLS . . . DANN . . .** (**SONST**)... verwendet. Dieser entspricht direkt dem BASIC-Befehl IF..THEN...(ELSE)... , d.h. es erfolgt (je nach log. Zustand der Bedingung) eine der beiden Ausgaben, welche jedoch auch leer sein können. (Der SONST - Zweig kann sogar einschließlich des "SONST" ganz weggelassen werden.)

Ein **prinzipieller Programmaufbau** kann beispielsweise wie folgt gestaltet sein:




TASK 1
SOBALD
MERKER 1
DANN
gestartet
....
....
....
DANN
MERKER 10
Aufrufeines
über
SOBALD
-MERKER 10
DANN
bisdieser
....
....
....
....
SPRUNG 1
TASK 2
SOBALD
MERKER 10
Haupt-
DANN
überMerker10
....
....
....
DANN
-MERKER 10
zuEnde
SPRUNG 2
Programmende
; Hauptprogramm z.B
; Automatik
; über Merker
; Abarbeitung
; Hauptprogramm
; Eventuell
; Parallelablaufs,
; Merker
; dann warten
; beendet
; weitere Abarbeitung
; Hauptprogramm
; Parallelablauf,
; gestartet vom
; programm
; Abarbeitung
; Parallelprogramm
; Meldung Ablauf

Beschreibung der einzelnen parallelen Zweige (Tasks) und deren Zusammenwirken:

TASK 0

TASK 0 muss immer vorhanden sein

Im TASK 0 wird zunächst eine Initialisierung der Zusatzkarten und der im Programm verwendeten Ausgänge, Merker und Register vorgenommen. Bei Achsen ist im Normalfall eine Referenzfahrt nötig, welche auch hier untergebracht werden kann.

TASK 0: Initialisierung, Abfrage schleife für Eingänge, Tasten, etc.

Anschließend folgt eine Reihe von FALLS - Bedingungen. Dabei können zum Beispiel Eingänge abgefragt werden. Diese **Abfrage Schleife** wird ständig durchlaufen, es darf also keine SOBALD - Anweisung darin stehen. Auf diese Weise erreicht man, dass wenn eine dieser Bedingungen wahr wird, dies sofort erkannt wird und dementsprechend gehandelt werden kann.

Es wird dann sofort zur entsprechenden Marke (MARKE 41, MARKE 42 etc.) gesprungen, wo diejenigen Anweisungen stehen, die abgearbeitet werden müssen. Dabei kann es sich auch um die Auslösung eines Automatik-Steuerungs-Tasks oder ähnlichem handeln. Es dürfen jedoch keine Anweisungen, deren Ausführung lange dauert, an dieser Stelle untergebracht werden. Ansonsten wird die Abfrage-Schleife nicht mehr durchlaufen und die neuen Eingaben werden nicht mehr berücksichtigt.

Bei "Aufruf Automatik" könnte zum Beispiel ein Task gestartet werden, welcher eine automatische Steuerung ausführt. Ebenso könnte bei "Aufruf Handautomatik" ein anderer Task gestartet werden, welcher eine von Hand kontrollierte Steuerung ausführt.

TASK 1

TASK 1: Automatik-Task, Steuerung der Haupt-funktionen

In diesem Task steht das **Haupt - Steuerungsprogramm.**
Dieses Hauptprogramm wird mit dem MERKER 1 gestartet. Es könnte aus dem TASK 0 (MARKE 41) gestartet werden. So könnte die Bedingung für diesen Vorgang eine Start-Taste (Eingang) sein. Sobald dieser Eingang dann durch Drücken der Taste aktiviert wird der MERKER 1 gesetzt und schon ist die Start-Funktion ausgelöst.

Das Hauptprogramm arbeitet dann seine Befehle ab, startet eventuell ein weiteres paralleles Programm (hier mit MERKER 10) und wird dann durch definierte Bedingungen zum Ende kommen. Man muss sich dieses Hauptprogramm, wenn es sich wirklich um eine Automatik handelt, als Endlos-Schleife vorstellen. Durch bestimmte Abbruchbedingungen, zum Beispiel eine Stop-Taste oder eine Fehlermeldung, wird dann diese Endlosschlaufe verlassen und das Ende erreicht.

Vom Ende wird wieder zum Anfang des Tasks gesprungen, so dass der Task wieder bereit ist, um neu aufgerufen zu werden. Ist der MERKER 1 in der Zwischenzeit nicht zurückgesetzt worden, so läuft die Steuerung gleich wieder los, was unter Umständen auch erwünscht ist. Ansonsten müsste man den Merker vor dem Sprung zum Anfang noch zurücksetzen.

TASK 2

Dieser Task stellt ein Beispiel eines einfachen über Merker aufgerufenen Tasks dar, und steuert zum Beispiel die Bedienerführung.

2.1.1 Regeln zum Programmaufbau - Taskstruktur

Definition eines Parallelzweigs (Tasks)

32 Task ermöglichen parallel ablaufende Programmteile

Zur Programmierung von komplexeren Abläufen und zur Realisierung von Unterprogrammen gibt es bei der PROZESS-SPS die Möglichkeit, mehrere parallel laufende Programmteile zu schreiben. Insgesamt sind 32 voneinander unabhängige Programmteile möglich. Diese können über MERKER (FLAG) oder REGISTER auch miteinander verknüpft (also abhängig gemacht) werden. Einen dieser Programmteile nennt man **"Parallelzweig" oder "Task".**

TASK 0 muss immer vorhanden sein

Ein Parallelzweig beginnt immer mit einem TASK-Befehl. Die TASK für Parallelzweige haben die Nummern 0 bis 31 (je einschließlich). TASK 0 ist immer vorhanden. Wenn kein weiterer TASK mit einer Nummer kleiner 32 eingegeben wird, so ist der "Parallelzweig" Nr. 0 der einzige und somit existiert keine Parallelverarbeitung in der Steuerung.

Nummerierung aufsteigend und lückenlos

Wird ein Parallelzweig benötigt, dann muss ein TASK mit der Nummer 1 eingetragen werden. Für alle weiteren benötigten Parallelzweige muss in aufsteigender Reihenfolge und ohne eine Nummer auszulassen die nächste Nummer gewählt werden.

TASK werden im Editor mit einer Linie markiert. Diese Linie soll die Trennung zwischen zwei Programmen verdeutlichen.

SPRUNG-Ziele nur innerhalb eines Task. Niemals tasküber-greifend

Ein Parallelzweig ist ein in sich abgeschlossenes Programm. Das heißt, dass es aus einem Parallelzweig heraus keinen SPRUNG an eine MARKE geben darf, die auch gleichzeitig von einem anderen Parallelzweig benutzt wird. Am Ende eines Parallelzweigs muss ein SPRUNG an eine MARKE innerhalb des Parallelzweigs stehen. Andernfalls läuft der Parallelzweig "in den nächsten"; dies führt zu einer Fehlfunktion des Programms.

Globale Unter-programme an das Programmtex t-ende

Die Ausnahme bilden Unterprogramme und Funktionen. Sie können durchaus von einem anderen Parallelzweig aus aufgerufen werden. Im Prinzip ist sogar der gleichzeitige Aufruf desselben Unterprogramms von verschiedenen Parallelzweigen aus möglich - es sollte dann aber am Ende des letzten Tasks stehen (globale Unterprogramme an den Schluss des gesamten Programmtextes).

TASK, die am Anfang eines Parallelzweigs stehen (0..31) dürfen wie alle anderen Marken auch, als Ziele für Sprunganweisungen (SPRUNG) benutzt werden.

Der Programmaufbau ohne Multi-Tasking, also ohne parallele Zweige, kann zum Beispiel folgendermaßen aussehen:

```
      TASK  0 ---------------------------------
--
            ...
            ...
            ...               ; Programmtext
            ...
            ...
      MARKE  50
            ...
            ...               ; Programmtext
            ...
            SPRUNG  50
```

Die Parameter nummern des TASK oder MARKE-Befehles können symbolische Namen haben

Wobei am Ende zur Marke 50 gesprungen wird. Die Nummer dieser Marke kann natürlich eine beliebige Zahl von 32 bis 999 oder ein symbolischer Name sein. Die Sprunganweisung am Schluss muss zu einer vorhandenen MARKE geschehen. Es kann auch an den Anfang, also direkt zu TASK 0 gesprungen werden.

Beispiel für die Strukturierung eines Programms mit drei Parallelzweigen:

```
      TASK  0 ----------------------------------
--
            ...
      MARKE  101        ; Alle gesetzten Marken haben
                  natürlich
                        ; nur einen Sinn,
      MARKE  105        ;.wenn sie angesprungen
                  werden.
                        ; Die Sprünge können
      MARKE  102        ; dabei beliebig und kreuz und
                  quer,
                        ; auch nach vorne
            ...               ; programmiert werden,
                  jedoch nur
                        ; innerhalb desselben
            ...               ; Parallelzweigs (Tasks).
            SPRUNG 101
      TASK  1 ----------------------------------
--
            ...
      MARKE  200
      MARKE  201
            ...
            SPRUNG 201
            ...
            SPRUNG 1
      TASK  2 ----------------------------------
--
            ...
            ...
            ...
            SPRUNG 2
```

Parallelbearbeitung (Taskswitching)

Regeln zum Taskwechsel

Die parallele Bearbeitung der einzelnen Parallelzweige des Anwenderprogramms wird vom Betriebssystem der Steuerung verwaltet. Es gibt gewisse Regeln, nach denen die Bearbeitung der einzelnen Programmteile aufgenommen und der Wechsel von einem auf den nächsten Parallelzweig vorgenommen wird. Diese Regeln zu kennen, ist für den fortgeschrittenen Programmierer interessant:

Regel 1: Die Bearbeitung des Anwenderprogramms beginnt immer mit dem TASK 0.

Regel 2: Der Parallelzweigwechsel erfolgt immer bei den folgenden Befehlen oder Situationen:

WARTEZEIT

BEDIENEREINGABE

Bei einer SOBALD - Bedingung, die nicht erfüllt ist

Regel 3: Folgende drei weiteren Bedingungen für einen Taskwechsel können vom Anwender mittels Merker (Merkernummer im Steuerungs-handbuch) beeinflusst werden. In eckigen Klammern steht der Default nach Reset.

- **Nach einer bestimmten Zeit (Taskswitch-Timeout) [ein]**
- **Beim SPRUNG - Befehl [ein]**
- **Bei einer nicht erfüllten FALLS - Bedingung [aus]**

Zur Regel 3:

a) Taskswitch-Timeout (Merker 2056)

Taskwechsel nach einer definierten Zeit

Nach einer bestimmten Zeit wird der Task, sobald der aktuelle Befehl beendet ist, gewechselt, ohne dass eine andere Bedingung erfüllt sein muss.
Diese Zeit, welche im Spezialregister "Task-Timeoutzeit" (in Einheiten von ms) gespeichert ist, kann vom Anwender verändert werden.

Nach Reset ist der Merker 2056 = 1, also gesetzt
ist das Spezialregister "Task-Timeoutzeit" mit 20 (ms) geladen

Ohne Änderung von Merker 2056 und Spezialregister "Task-Timeoutzeit" ist also dieser Taskswitchtimeout eingeschaltet. Es wird also ein Task nach höchstens 20 ms gewechselt.

Anmerkung:

Ein Ausgangszweig kann auch durch Timeout nicht unterbrochen werden und wird daher noch komplett abgearbeitet.

b) Beim SPRUNG - Befehl (Merker 2057)

Taskwechsel vor dem SPRUNG-Befehl

Ist der Merker 2057 gesetzt, so wird bei jedem SPRUNG - Befehl der Task gewechselt.

Auch dieser Merker ist nach Reset = 1, also gesetzt.

c) Nicht erfülltes FALLS (Merker 2058)

Taskwechsel vor nicht erfülltem FALLS

Ist der Merker 2058 gesetzt, so wird bei jeder FALLS - Bedingung, wenn sie nicht erfüllt ist, ein Taskwechsel durchgeführt. (Der DANN - Zweig wird also gegenüber dem SONST - Zweig bevorzugt.)

Dieser Merker ist nach Reset = 0, also nicht gesetzt.

d) Merker 2056 bis 2058 <---> Spezialregister "Taskswitch-Bedingungen"

Überlagerung Merker 2056 bis 2058 mit Spezialregister

Diese drei Merker sind mit dem Spezialregister "Taskswitch-Bedingungen" überlagert, wobei die drei Merker die niederwertigsten drei Bits des Registers darstellen (Merker 2056 <--> Bit 0 usw.) und die höherwertigeren Bits des Registers nicht benutzt werden.

Die angegebenen Werte nach Reset ergeben für das Spezialregister "Taskswitch-Bedingungen" den Wert 3.

Durch den Befehl

```
LADE_REGISTER [ Taskswitch-Bedingungen mit 0]
```

können alle drei Bedingungen von Regel 3 ausgeschaltet werden.

2.1.2 Spezialregister / -merker zur Tasksteuerung

Merker 2056

Ist dieser Merker gesetzt, **so wird nach einer bestimmten Zeit,** welche im Spezialregister "Task-Timeoutzeit" steht, **der Task** ohne dass eine andere Bedingung zutrifft **gewechselt.** Natürlich kann nicht mitten in einem Befehl gewechselt werden, jedoch gleich anschließend. Nach Reset ist dieser Merker gesetzt, also die Taskwechselbedingung eingeschaltet.

Spezialregister "Task-Timeoutzeit"

Dieses Register legt die obige Zeit in Einheiten von ms (Millisekunden) fest. Nach Reset ist dieses Register auf 20 gesetzt, was zur Folge hat, das nach höchstens 20 ms der Task gewechselt wird.

Merker 2057

Ist dieser Merker gesetzt, so wird **bei jeder SPRUNG-Anweisung der Task gewechselt.** Dieser Merker ist nach Reset gesetzt, es wird also ohne Veränderung dieses Merkers der Task bei jedem SPRUNG-Befehl gewechselt.

Merker 2058

Wenn dieser Merker gesetzt ist, so wird **bei jeder FALLS-Bedingung, wenn diese nicht erfüllt ist, der Task gewechselt.** Dies bedeutet also eine Bevorzugung des DANN-Zweigs gegenüber dem SONST-Zweig.
Nach Reset ist dieser Merker nicht gesetzt.

Das **Spezialregister "Taskswitch-Bedingungen"** ist mit den Merkern 2056 bis 2063 überlagert, wobei jene Merker von 2059 bis 2063 nicht benutzt sind. Es werden also nur die drei Bits mit der kleinsten Wertigkeit verwendet. Durch Zuweisung eines bestimmten Wertes auf dieses Register können in einem Befehl alle

Taskwechselbedingungen so wie gewünscht festgelegt werden. Nach Reset beträgt der Wert dieses Registers, wie aus den Werten der obigen Merkern folgt, 3.

Im folgenden Beispiel werden die gleichen Taskwechsel - Bedingungen auf zwei verschiedene Arten gesetzt:

Nummerierung exemplarisch: DELTA

Beispiel:

```
LADE_REGISTER [ 61467 mit 4]            -MERKER
2056
                                      -MERKER
2057
                                       MERKER
2058
```

Links werden durch Zuweisung des Wertes 4 auf das Register 61467 die Merker 2056, 2057 und 2058 mit den Werten 0,0 und 1 besetzt. Rechts erfolgt dasselbe über die drei MERKER - Befehle.

2.2 Symbolische Programmierung

Symbolischer Name statt nummerischer Parameter:

Die numerischen Parameter der Programmiersprache können durch symbolische Namen ersetzt werden. Das trägt zur Übersichtlichkeit des Programmes bei, und erleichtert das Einarbeiten in den Quelltext. Wichtig ist, dass nur der numerische Teil des Parameters durch Symbolik ersetzt wird. Alle anderen Bestandteile des Parameters bleiben bestehen.

E eStart

statt

E 101

So wird zum Beispiel aus

Eingang 101 -> Eingang Start

oder aus

Register 100 -> Register Stückzahl

Statt der wenig aussagekräftigen numerischen Parameter können Symbolnamen verwendet werden, und damit die Transparenz und Verständlichkeit des Programmes erhöht werden.

Siehe auch die Ausführungen zum Symbol-Editor im *Kapitel SYMPAS-Programmierumgebung*.

2.2.1 Empfehlungen zur Symbol-Notation

Typisierung der Symbolnamen

Es ist sinnvoll für die Wahl der Symbolnamen einige Regeln zu vereinbaren. Die Typisierung der Symbolnamen für Eingänge, Ausgänge, Merker, Register, etc. sollte im Symbolnamen erfolgen. Also zum Beispiel

eStart für einen Starteingang, oder

rZielposition für ein Zielpositionsregister einer Achse

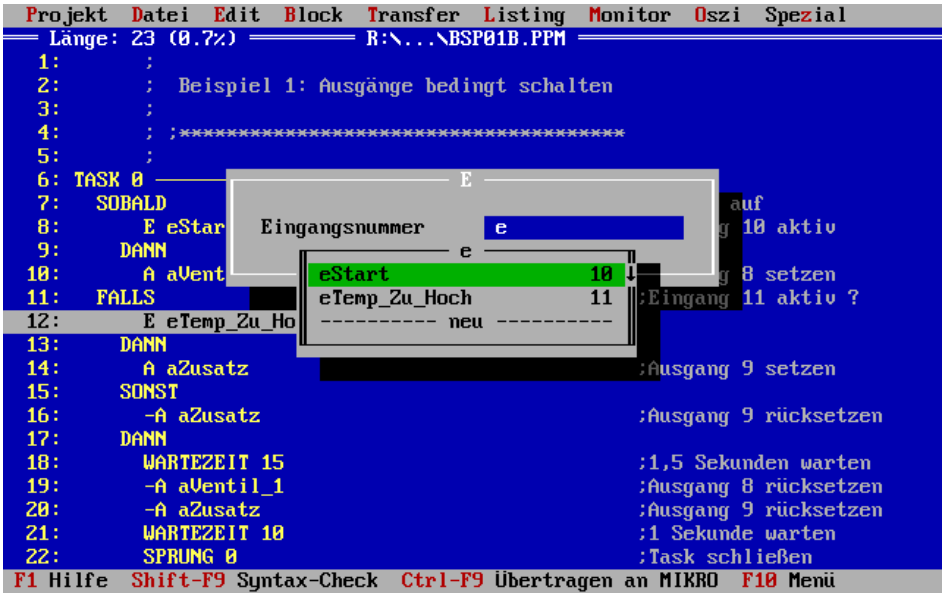
Das Voranstellen der typisierenden Kleinbuchstaben vor den eigentlichen Symbolnamen hat zwei wesentliche Vorteile

Erhöhte Lesbarkeit

- die Lesbarkeit des Programmkodes wird erhöht, da jeder Symbolnamen nicht nur über seine Bedeutung bzw. Funktion (durch den Namen an sich) sondern auch über seinen Typ (Eingang, Ausgang, Merker, Register, etc.) Auskunft gibt.

Liste aller Symbole gleichen Typs

- bei der Namenseingabe besteht die Möglichkeit durch Eingabe von **e?** zum Beispiel alle Eingänge im Dialogfenster anzeigen zu lassen. Sie müssen sich die Symbole also nicht mehr ausdrucken oder merken, sondern erhalten eine bequeme Online-Hilfestellung.



Empfohlene Kürzel zur Typisierung der Symbol-namen

Folgende vorangestellte Kleinbuchstaben zur Typisierung werden empfohlen

Kürzel	Typ	Untertyp
e	Eingang	-
a	Ausgang	-
m ms	Merker	- Spezialmerker
r rs rf rm rt rv	Register	- Spezialregister Fließkommareg Slaveregister auf Modulen, Karten Textregister VIADUKT-Register
s sm se su	Marken	- für SOBALD_MAX Fehlerunterpro-gramme Unterprogramme
t	Task	-
z ze zk	Zahl	- Fehlermeldung Kommandos

zb zv		Definition eines Registerbit VIADUKT-Masken-Aufruf
ax	Achsen	

2.2.2 Beispiele zur Symbol-Notation

Folgende Programmsequenz veranschaulicht die oben getroffenen Empfehlungen zur Symbol-Notation.

```
 0:       ; ************************************
 1:       ;
 2:       ;  Beispiel 1: Ausgänge bedingt schalten
 3:       ;
 4:       ; ;***********************************
 5:       ;
 6: TASK 0 ──────────────────────
 7:   SOBALD                      ;Warten auf
 8:       E eStart                ;Eingang 10 aktiv
 9:     DANN
10:       A aVentil_1             ;Ausgang 8 setzen
11:   FALLS                       ;Eingang 11 aktiv ?
12:       E eTemp_Zu_Hoch
13:     DANN
14:       A aKühl                 ;Ausgang 9 setzen
15:     SONST
16:       -A aKühl                ;Ausgang 9 rücksetzen
17:     DANN
18:       WARTEZEIT 15            ;1,5 Sekunden warten
19:       -A aVentil_1            ;Ausgang 8 rücksetzen
20:       -A aKühl                ;Ausgang 9 rücksetzen
21:       WARTEZEIT 10            ;1 Sekunde warten
22:       SPRUNG 0                ;Task schließen
Programmende
```

Dabei wurden folgende Symboldefinitionen vorgenommen

```
eStart          10       ;Eingang: Starteingang
eTemp_Zu_Hoch   11       ;Eingang: Temperatursensor
aKühl           9        ;Ausgang: Kühlaggregat
aVentil_1       8        ;Ausgang: Einlassventil
```

2.3 Anmerkung zu den Programmbeispielen

Hinweis:

In den folgenden Programm- und Befehlsbeispielen wird sowohl symbolische als auch numerische Programmierung eingesetzt. Auf Symbolik wird verzichtet, wenn es sinnvoll ist den Bezug zur Hardware direkt über die Registernummer herzustellen oder aus didaktischen Gründen immer dann, wenn die symbolische Darstellung den Sachverhalt unnötig komplizieren würde.

Bei numerischer Angabe werden die Registernummern der PROZESS-SPS NANO-B verwendet; Ausnahmen werden gekennzeichnet.

3. Die Programmiersprache

In diesem Kapitel werden alle Befehle beschrieben, welche zur Programmierung der Steuerung zur Verfügung stehen. Die Befehle werden zuerst beschrieben und dann wird anhand eines (oder mehrerer) Beispiele aufgezeigt, wie man sie verwenden kann.

3.1 Übersicht der Befehle

In der folgenden Liste sind alle Befehle aufgeführt.

Arithmetische und Boole'sche Zeichen:

> = < + - * / ( ) #

(*Kapitel 3.3 Boole'sche Ausdrücke* und *Kapitel 3.4 Arithmetische Ausdrücke*)

Kommentarzeichen:

; (*Kapitel 3.11.3 Das Kommentarzeichen*)

PROZESS-SPS-Befehlssatz		
Kürzel	**Befehl**	**Bemerkung**
AR	ANZEIGE_REG	Registerinhalte auf LCD, Drucker ausgeben
AT	ANZEIGE_TEXT	Texte auf LCD, Drucker ausgeben
A2	ANZEIGE_TEXT_2	in Abhängigkeit eines Registers kann zwischen zwei alternativen Texten gewählt werden (z.B. Zweisprachigkeit).
AU	AUSGANGSNUMMER	Setzen, Rücksetzen, Abfragen eines digitalen Ausganges
BE	BEDIENEREINGABE	Eingabe von Registerwerten durch den Bediener mit Hilfe des LCD
BL	BIT_LÖSCH	löscht das Bit eines Registers, oder fragt ein Bit auf 0 ab
BS	BIT_SETZ	setzt das Bit eines Registers, oder fragt ein Bit auf 1 ab
DA	DANN	FALLS..**DANN**..SONST, SOBALD..DANN
DF	DEF_FUNKTION	Markiert den Beginn einer Funktions-definition
ED	END_DEF	Markiert das Ende einer Funktions-definition
EI	EINGANGSNUMMER	Abfragen eines digitalen Einganges
F	FALLS	**FALLS**..DANN..SONST
G	GRENZEN	1. Ermittelt ob Register sich innerhalb bestimmter Grenzen befindet (**Abfrage**) 2. Zwingt Register innerhalb bestimmte Grenzen (**Anweisung**)
H	HALTACHSE	1. Ermittelt ob Achse angehalten

		wurde (**Abfrage**) 2. Hält Achse an (**Anweisung**)
I	ISTPOS	ermittelt die Istposition einer Achse
K	KOPIERE	kopiert einen Registerbereich
LE	LEER	dieser Befehl zeigt keine Wirkung hat aber eine Ausführungszeit (Testzwecke)
LM	LÖSCHE_MERKER	löscht einen Merkerbereich
LR	LADE_REGISTER	lädt ein Register mit einem Wert (direkt, indirekt, doppelt indirekt).
MA	MARKE	Sprungmarke für den Programmfluss
ME	MERKER	Setzten, Rücksetzten, Abfragen eines Merkers
NH	N-HOLE-REGISTER	lädt ein Register einer Slave-Steuerung in den Speicher der Master-Steuerung, JETWay, Feldbus
NI	NICHT	logisch NICHT (invertiert eine Eingangs- bedingung)
NS	N-SENDE-REGISTER	überträgt ein Register der Master-Steuerung in den Speicher einer Slave-Steuerung, JETWay, Feldbus
O	ODER	logisch ODER (Eingangsbedingung)
P	POS	positioniert eine Achse mit Geschwindigkeit **v** auf die Position **Pos**
RD	REGDEC	verringert den Wert eines Registers um 1
RE	REG	Registerbefehl, z.B. REG 100 = 1234
RI	REGINC	erhöht den Wert eines Registers um 1
RL	REG_LÖSCHEN	setzt einen Registerbereich auf 0
RN	REGNULL	setzt ein Register auf 0, oder fragt ein Register auf 0 ab
RÜ	RÜCKSPRUNG	schließt Unterprogramm oder Funktion ab
SB	SOBALD	**SOBALD..**DANN
SF	SPEZIALFUNKTION	ruft verschiedene Spezialfunktionen

		auf, z.B. Trigonometrie
SM	SOBALD_MAX	**SOBALD_MAX..**DANN, zusätzlich kann eine Zeit angegeben werden nach deren vertreichen ein Unterprogramm (z.B. Fehlerbehandlung) angesprungen wird
SO	SONST	FALLS..DANN..**SONST**
SP	SPRUNG	steuert den Programmfluss
SZ	STARTE-ZEIT	startet ein Zeitregister
TA	TASK	Marke für den Taskbeginn
TB	TASKBREAK	hält einen Task an
TC	TASKCONTINUE	setzt angehaltenen Task fort
TR	TASKRESTART	startet angehaltenen Task ab Beginn
U	UNTERPROGRAMM	ruft ein Unterprogramm auf
WA	WARTEZEIT	der Task unterbricht für eine bestimmte Zeit die Task-Ausführung
WO	WODER	ODER-Verknüpfung von Registern
WU	WUND	UND-Verknüpfung von Registern
WX	WXODER	Exklusiv-ODER-Verknüpfung von Registern
ZE	ZEIT-ENDE	fragt Zeitregister ab

Prozess-SPS-Befehlssatz - Zahlen

Kürzel	Befehl	Bemerkung
ZB	Zahl, binär	die Zahleneingabe erfolgt binär: b01010101010101010101010101
ZD	Zahl, dezimal	die Zahleneingabe erfolgt dezimal: 1234
ZH	Zahl, hexadezimal	die Zahleneingabe erfolgt hexadezimal: hFA23CD

3.2 Grundbefehle

3.2.1 Wartebedingung SOBALD ... DANN

Syntax:

```
SOBALD
    <Bedingung>
  DANN
```

SOBALD wartet bis Bedingung erfüllt

Bedeutung:

Es wird gewartet, bis <Bedingung> erfüllt ist und erst dann mit der folgenden Anweisung (nach DANN) weitergemacht.

Elementar-bedingungen:

Eingang, Ausgang, Merker, Registerbit, arithmetischer Vergleich

Bei der Bedingung handelt es sich entweder um einen Merker, einen Eingang, einen Ausgang, ein bestimmtes Bit eines Registers oder das Ergebnis eines arithmetischen Vergleiches.

Diese "Elementarbedingungen" können beliebig zu einem sogenannten Boole'schen Ausdruck verknüpft werden, dessen Ergebnis dann als Bedingung gilt. Dabei sind sogar Klammern zugelassen. Ist die Ordnung nicht mit Klammern anders festgelegt, so wird der Boole'sche Ausdruck von vorne nach hinten durchgearbeitet und dann das Ergebnis als Bedingung interpretiert. (Siehe auch Boole'sche Ausdrücke *Kapitel 3.3 Boole'sche Ausdrücke).*

Beispiele:

```
1)    SOBALD
          E eStart
          MERKER mTaskFreigabe
        DANN
```

Sobald der Eingang E eStart aktiv ist und der MERKER mTaskFreigabe gesetzt ist, wird das Programm mit dem Befehl, der nach DANN folgt, fortgesetzt.

Hinweis:

Steht bei einem boole'schen Ausdruck nichts zwischen zwei oder mehr Elementarbedingungen, so wird dies automatisch als UND-Verknüpfung interpretiert.

```
2)    SOBALD
          REG rSpannung
          =
          50
        DANN
```

Sobald das Register REG rSpannung den Wert 50 hat, wird die Programmausführung fortgesetzt. (Der Wert von REG rSpannung kann zum Beispiel in einem anderen Task verändert werden, oder einen analogen Spannungswert repräsentieren.)

3.2.2 Wartebedingung SOBALD_MAX ... DANN

Syntax:

```
SOBALD_MAX      [Max.Zeit=<Zeit>,      Unterprog.=
<Unterprog.>]
      <Bedingung>
    DANN
```

SOBALD wartet bis Bedingung erfüllt ist. Zusätzlich: Timeoutzeit

Bedeutung:

Es wird gewartet, bis <Bedingung> erfüllt ist und erst dann mit der folgenden Anweisung (nach DANN) weitergemacht.
Ist die Max.Zeit abgelaufen bevor die Bedingung erfüllt ist, so wird das Unterprogramm aufgerufen.

Bei der Bedingung handelt es sich entweder um einen Merker, einen Eingang, einen Ausgang, ein bestimmtes Bit eines Registers oder das Ergebnis eines arithmetischen Vergleiches.

Beispiele:

```
1)                    SOBALD_MAX[Max.Zeit=z5s,
Unterprog.=seFehler]
          REG rDruckInZylinder
          >
          50
        DANN
          ...
          ...
    MARKE seFehler
          -A aHydraulikPumpe
          ANZEIGE_TEXT[#0,cp=1,"_Fehler
beheben"]
          ANZEIGE_TEXT[#0, cp=25, "Danach F1"]
      SOBALD
          MERKER mTasteF1
        DANN
          A aHydraulikPumpe
```

```
          ANZEIGE_TEXT[#0, cp=1, "_"]
          RÜCKSPRUNG
```

Sobald das Register REG rDruckInZylinder größer als 50 ist, wird mit den Anweisungen nach DANN fortgefahren. Befindet sich der Task länger als 5 Sekunden (Max.Zeit = z5s) an der nicht erfüllten SOBALD-Bedingung wird die Fehlerroutine seFehler aufgerufen. Nach dem die Fehlerroutine abgearbeitet worden ist kehrt diese mit dem RÜCKSPRUNG-Befehl an die SOBALD-Bedingung zurück. Der Task wartet also wieder bis REG rDruckInZylinder größer als 50 wird - was jetzt der Fall sein sollte, denn die Fehlerroutine hat den Fehler korrigiert.

Hinweis:

Nach dem die Fehlerroutine abgearbeitet worden ist kehrt diese mit dem RÜCKSPRUNG-Befehl an die SOBALD-Bedingung zurück.

```
2)                    SOBALD_MAX[Max.Zeit=z2s,
Unterprog.=seNotAus]
          E ePneuAchse
        DANN
          ...
          ...
          ...
    MARKE seNotAus
          -A aPneuAchse
          HALTACHSE Achse=X
          HALTACHSE Achse=Y
          HALTACHSE Achse=Z
          A aWarnton
          ANZEIGE_TEXT[#0, cp=1,"_NotAus"]
          ...
          ...
          RÜCKSPRUNG
```

Sobald der Eingang ePneuAchse aktiv wird, wird mit den Anweisungen nach DANN fortgefahren. Befindet sich der Task länger als 2 Sekunden (Max.Zeit = z2s) an der nicht erfüllten SOBALD-Bedingung wird die

Fehlerroutine seNotAus aufgerufen. Nach dem die Fehlerroutine abgearbeitet worden ist kehrt diese mit dem RÜCKSPRUNG-Befehl an die SOBALD-Bedingung zurück.

3.2.3 Verzweigungsbedingung FALLS ... DANN ... (SONST)

Syntax:

```
FALLS
    <Bedingung>
  DANN
    <1. Anweisung>
  SONST
    <2. Anweisung>
  DANN oder SOBALD oder FALLS
```

Bedeutung

Ist <Bedingung> erfüllt, so wird die 1. Anweisung ausgeführt. Ist <Bedingung> nicht erfüllt, wird die 2. Anweisung ausgeführt.
Anschließend wird nach dem zweiten DANN, einem folgenden SOBALD oder einem neuerlichen FALLS mit der Bearbeitung des Programms fortgefahren.

Für <Bedingung> muss wiederum ein beliebiger Boolescher Ausdruck - wie bei der SOBALD-Anweisung - stehen.

Die 1. Anweisung und die 2. Anweisung können beide aus mehreren Befehlen bestehen. Sie können beide aber auch fehlen.

Hinweis:

- Der DANN-Zweig wird durch ein SONST oder DANN, SOBALD, FALLS abgeschlossen.
- Der SONST-Zweig wird durch DANN, SOBALD, FALLS abgeschlossen.

- Beide werden **nicht** durch MARKE abgeschlossen.

Beispiele:

1)
```
FALLS
   E eAnschlag
 DANN
   MERKER mTeilPosOK
 SONST
   -MERKER mTeilPosOK
 DANN (oder FALLS oder SOBALD)
   ...
```

Dieses Beispiel zeigt, wie die FALLS - DANN - SONST - Struktur im Normalfall aussieht.
Ist der Eingang E eAnschlag aktiv (<Bedingung> ist erfüllt), dann wird der MERKER mTeilPosOK gesetzt, die Anweisung im SONST-Zweig übersprungen und nach dem zweiten DANN fortgefahren.
Ist E eAnschlag nicht aktiv (<Bedingung> nicht erfüllt), so wird die Anweisung im DANN-Zweig übersprungen, der Merker zurückgesetzt und anschließend bei der Anweisung nach dem zweiten DANN fortgefahren.

2)
```
FALLS
   MERKER msTasteF1
   ODER
   MERKER msTasteF2
 DANN
 SONST
   A aLampe
 DANN (oder FALLS oder SOBALD)
 ...
```

Hier wird bei erfüllter Bedingung (MERKER msTasteF1 oder MERKER msTasteF2 gesetzt) nichts gemacht (DANN - Zweig ist leer). Ist die Bedingung jedoch nicht erfüllt, so wird der Ausgang A aLampe eingeschaltet (SONST - Zweig).

3)
```
FALLS
```

```
      E eStart
    DANN
      A aHydAchse
      A aWarnton
      LADE_REGISTER [ rLänge mit 25]
    SONST
    DANN (oder FALLS oder SOBALD)
    ...
```

Ist der Eingang E eStart aktiv, dann werden die Ausgänge A aHydAchse und A aWarnton gesetzt und es wird der Wert 25 ins Register REG rLänge geladen.
Ist jedoch E eStart nicht aktiv, so wird direkt zum zweiten DANN gesprungen und mit der nächsten Anweisung weitergemacht.

4)
```
SOBALD
    MERKER mStation1
  DANN
FALLS
    E eTaste1
  DANN
    A aPumpe
    WARTEZEIT 5Sekunden
    -A aPumpe
  SONST
    A aSignalTon
    WARTEZEIT 2Sekunden
    -A aSignalTon
FALLS
    MERKER mFehler
    E eAnschlag
  DANN
    A RelaisAus
  SONST
SOBALD
    -E eAnschlag
  DANN
    SPRUNG 100
```

In Beispiel 4) ist ein kleiner Ausschnitt aus einem Programm dargestellt, in dem noch einmal die Struktur der SOBALD-DANN-Anweisung und der FALLS-DANN-SONST-Verzweigung dargestellt ist.

Der SYMPAS-Editor unterstützt die Erkennung dieser Strukturen durch verschiedene Einrückpositionen der Befehle.

3.2.4 Der WARTEZEIT-Befehl

Der Befehl

WARTEZEIT <Wartezeit>

dient zur Programmierung einer festen Zeit, während der die Ausführung des Task angehalten werden soll. Es soll einfach gewartet werden bis die angegebene Zeit abgelaufen ist. Die Wartezeit ist auch der einzige Parameter, welcher hier eingegeben werden muss.

Indirekte Adressierung der Wartezeit ist möglich

Die Angabe der Wartezeit kann auch indirekt über Register erfolgen.

Im Normalfall wird der **Parameter Wartezeit** in Einheiten von **100 ms** angegeben. Eine Wartezeit von 10 entspricht demnach einer Sekunde.

Einheit WARTEZEIT Parameter: 100ms

Diese Einheit kann durch Ändern des Spezialregisters "Anwender-Zeitbasis" definiert werden. Dabei ist jedoch Vorsicht geboten, denn sehr kleine Einheiten führen dazu, dass das Betriebssystem mit sehr hohem Verwaltungsaufwand beschäftigt ist. Wie das genau geschieht ist bei der Beschreibung der Zeitregister nachzuschlagen. An dieser Stelle soll jedoch nicht näher darauf eingegangen werden, da es nur für ganz spezielle Anwendungen gebraucht wird.

Beispiel:

```
ANZEIGE_TEXT [#0, cp=0, "_Wunderbar !"]
WARTEZEIT  50
ANZEIGE_TEXT [#0, cp=0, "_"]
```

Bei diesem Beispiel wird auf dem Bediengeräte ein Text angezeigt, 5 Sekunden gewartet und schließlich die Anzeige wieder gelöscht.

Zeitregister

Im Zusammenhang mit der Wartezeit ist es wichtig, auch die Zeitregister zu erwähnen: Es handelt sich dabei um die Register, in welche der Parameterwert des Wartezeitbefehls geschrieben wird. Sobald ein solches Register nicht mehr Null ist, wird es nach jeder Zeiteinheit um eins verkleinert. Der Befehl Wartezeit besteht lediglich darin, dieses Register zu laden und anschließen zu warten, bis es Null ist.

Zu den Zeitregistern: Jeder Task hat sein eigenes Zeitregister (dessen Nummer im entsprechenden Steuerungshandbuch nachgeschlagen werden kann). Das folgende Beispiel zeigt die Task-Zeitregister exemplarisch für die NANO-B.

Nummerierung exemplarisch: NANO-B

Beispiel: Task-Zeitregister

Task 0	Register 2300
Task 1	Register 2301
...	
...	
Task 31	Register 2331

In manchen Anwendungsfällen will man nun ein solches Zeitregister aktivieren, jedoch trotzdem mit Anweisungen im Programm fortfahren. Dies wird dadurch erreicht, dass mit dem LADE_REGISTER - Befehl das jeweilige Zeitregister geladen wird. Später kann das Zeitregister dann durch einfaches Abfragen in einen Vergleich eingebaut werden.

Hinweis:

Es muss darauf geachtet werden, dass man nicht sowohl den WARTEZEIT-Befehl wie auch das Zeitregister desselben Tasks gleichzeitig verwendet, denn dadurch können Fälle auftreten, wo eine Wartezeit nicht mehr fertig wird.

Diese Gefahr kann jedoch leicht ausgeschlossen werden: Man nimmt stattdessen einfach das Zeitregister eines Tasks, indem nie ein WARTEZEIT - Befehl vorkommt. Auf diese Art kann im selben Task sowohl der Befehl wie auch ein (oder sogar mehrere) Zeitregister verwendet werden.

Beispiele:

1a)
```
TASK  0 -------------------------
        ...
        ...
        WARTEZEIT  10
        ...
```

1b)
```
TASK  0 -------------------------
        ...
        ...
        LADE_REGISTER[rsTaskzeitReg   mit
10]
     SOBALD
        REGNULL rsTaskzeitReg
       DANN
        ...
```

Diese beiden Programme sind bezüglich ihrer Funktion genau gleich. Zuerst wird das Zeitregister gesetzt und dann gewartet, bis es Null ist. Genau das ist die Funktion des WARTEZEIT - Befehls.

3.3 Boole'sche Ausdrücke

Boolean sind entweder wahr (1) oder falsch (0)

Bei diesen Ausdrücken handelt es sich immer um Bedingungen, die entweder erfüllt sind oder nicht. Erfüllt hat dabei den Wert 1 und nicht erfüllt den Wert 0. Die Boole'schen Ausdrücke kommen immer in einem SOBALD oder FALLS - Zweig als Eingangsbedingungen vor, also nie nach einem DANN oder SONST!

Alles was zwischen einem FALLS oder SOBALD und dem nächsten DANN steht, wird von der Steuerung als ein Boolescher Ausdruck aufgefasst.

Elementar-bedingungen

Dabei gibt es sehr einfache Ausdrücke, welche zum Teil aus einem Befehl bestehen, und im folgenden **Elementarbedingungen** genannt werden sollen. Dazu gehören:

- Merker
- Eingänge
- Ausgänge
- einzelne Bits von Registern
- arithmetische Vergleiche (auch REG_NULL)
- HALTACHSE

Merker, Eingänge, Ausgänge und Registerbits können miteinander (logisch) verknüpft werden.

Dabei sind folgende Zeichen zugelassen:

= < > ( ) #

und folgende Befehle zur Verknüpfung:

NICHT ODER XODER UND

Elemetar-bedingungen per Default UND-verknüpft

Wenn zwischen zwei Eingangsbedingungen kein Verknüpfungsbefehl steht, so werden sie automatisch **UND**-verknüpft! In Ausgangsanweisungen muss UND verwendet werden.

Hinweis:

Drei Klammerebenen sind bei boole'schen Ausdrücken erlaubt.

3.3.1 Formulierung der Elementarbedingungen

Merker

Merker haben den Wert 1 oder 0

Ein Merker ist im Prinzip ein **Ein-Bit-Register.** Er kann den Wert 1 oder 0 haben. Ist der Merker gesetzt, so entspricht dies dem Wert 1. Ist er nicht gesetzt, so entspricht dies dem Wert 0. Durch Eingeben eines Minuszeichens vor der Merkernummer im Eingabefeld kann sehr leicht auf den invertierten Wert des Merkers und damit auf die Verneinung zugegriffen werden.

Beispiele:

```
1)    FALLS
          MERKER mStarteTask
```

MERKER mStarteTask gesetzt? (= 1 ?)

```
2)    FALLS                          FALLS
          -MERKER PosOk                  NICHT
                                         MERKER
PosOK
```

MERKER PosOK nicht gesetzt? (= 0 ?)

Beide Formulierungen haben genau die gleiche Auswirkung.

Eingänge und Ausgänge

Zugriff auf Ein- und Ausgänge z.B. mit E 101 oder -A 108

Auch diese haben entweder den Wert 1 oder 0 (gesetzt oder nicht gesetzt). Wie bei Merkern, kann auch hier durch ein vorangestelltes Minuszeichen auf den nicht aktiven Eingang bzw. Ausgang zugegriffen werden.

Beispiele:

```
1)    FALLS
          E eStart
```

Eingang `E eStart` aktiv? (=1 ?)

```
2)    FALLS                          FALLS
          -A aLampe                      NICHT
                                         A aLampe
```

Ausgang `A aLampe` nicht aktiv? (= 0 ?)
Beide Formulierungen haben dieselbe Bedeutung.

Einzelne Bits von Registern

Zugriff auf Registerbits mit den Befehlen BIT_SETZ und BIT_LÖSCH

Auf diese wird mit den Befehlen `BIT_SETZ` und `BIT_LÖSCH` zugegriffen (siehe auch Kapitel 3.6.4 Registerbit-Befehle für diese Befehle).
`BIT_SETZ` ist 1, falls das angegebene Bit den Wert 1 hat, sonst 0.
`BIT_LÖSCH` ist 1, falls das angegebene Bit den Wert 0 hat (gelöscht ist) sonst ist es 0.

Beispiele:

Register 1 habe den Wert 100110 (binär) = 38.

```
BIT_SETZ [Reg.1, Bit 2]  ist 1          (Bit ist gesetzt)
BIT_LÖSCH [Reg.1, Bit 2] ist 0          (Bit ist nicht gelöscht)
BIT_LÖSCH [Reg.1, Bit 4] ist 1          (Bit ist gelöscht)
BIT_SETZ [Reg.1, Bit 0]  ist 0          (Bit ist nicht gesetzt)
```

Arithmetische Vergleiche

Auch arithmetische Vergleiche sind wahr oder falsch

Auch diese sind immer entweder wahr oder falsch. Dabei erhält wahr den Wert 1 und falsch den Wert 0.

Beispiele:

1)
```
FALLS                          FALLS
   REG  rZähler                          REGNULL rZähler
   =                            DANN
   0
 DANN
```

Falls das Register REG rZähler den Wert 0 hat, so ist dieser Ausdruck wahr (= 1). Ist das REG rZähler nicht 0, so ist der Ausdruck falsch (= 0). Dieses Beispiel entspricht genau dem Befehl REGNULL rZähler. Dieser fragt nämlich, ob das angegebene Register 0 ist.

2)
```
FALLS
   REG rAnzahl
   >
   10
  DANN
```

Falls der Inhalt von Register REG rAnzahl größer als 10 ist, so ist dieser Ausdruck wahr, ist der Wert des Registers jedoch kleiner oder gleich 10, so wird der Ausdruck falsch (= 0).

Hinweis:

Der REG-Befehl oder arithmetische und logische Verknüpfungen von Registern oder Ergebnisse von Funktionen **ohne** Vergleichsoperator werden implizit mit 0 verglichen.

Beispiel:

```
   FALLS                              FALLS
         REG rTestReg                              REG
rTestReg
     DANN                                 #
                                          0
                                       DANN
```

Anmerkung:
Bei den arithmetischen Vergleichen ist es möglich, rechts und links vom Vergleichsoperator beliebige **arithmetische Ausdrücke** und **Wortverarbeitungsbefehle** zu verwenden (siehe hierzu *Kapitel 3.4 Arithmetische Ausdrücke).*

Beispiel:

```
      SOBALD
          REG 100
          >
          REG 1000
          *
          135
          +
          REG 1001
          WUND
          h00F080
          DANN
```

Es wird an dieser Programmstelle so lange gewartet, bis der Wert in Register 100 größer ist, als das Ergebnis der arithmetischen und logischen Verknüpfungen auf der rechten Seite.

Hinweis:

Die Kombinationen "> =" bzw. "< =" sind **nicht** zulässig.

3.3.2 Beispiele von verknüpften Ausdrücken

1)
```
FALLS
    MERKER 1
    MERKER 2
    ODER
    E 101
  DANN
```

Boole'scher Ausdruck:
(MERKER 1 UND MERKER 2) ODER E 101.
Dieser Ausdruck ist wahr, wenn Merker 1 und Merker 2 beide gesetzt sind oder wenn der Eingang E 101 aktiv ist.

2a)

```
      LADE_REGISTER [ rZähler mit 10]
MARKE  sSchleife
      ...
      REGDEC rZähler
  FALLS
      NICHT
      REGNULL rZähler
    DANN
      SPRUNG sSchleife
    DANN
      ...
```

2b)

```
      LADE_REGISTER [ rZähler mit 10]
MARKE sSchleife
      ...
      REGDEC rZähler
  FALLS
      REGNULL rZähler
    DANN
```

```
SONST
  SPRUNG sSchleife
DANN
  ...
```

In diesem Beispiel ist eine Schleife realisiert, welche 10 mal durchlaufen wird. Dazu wird zunächst der Schleifenzähler (Register REG rZähler) mit der Anzahl der Schleifendurchläufe geladen und innerhalb der Schleife um 1 erniedrigt (REGDEC rZähler). Am Ende der Schleife wird der Schleifenzähler (REG rZähler) geprüft, ob er bereits 0 erreicht hat. Ist dies nicht der Fall, so muss die Schleife nochmals durchlaufen werden (Sprung nach MARKE sSchleife).
Dabei wird der Vergleich auf zwei verschiedene Arten gemacht. Im einen Fall wird die Bedingung verneint und deshalb vom DANN-Zweig gesprungen, im anderen Fall wird vom SONST-Zweig die Schleife geschlossen. Beide Programme sind in Bezug auf ihre Funktion identisch.

3a)
```
SOBALD
    NICHT
    (
    E 101
    ODER
    -E 102
    )
    REG 100
    <
    20
  DANN
```

Sobald der Boole'sche Ausdruck '1' (also wahr) ist kann weiter gemacht werden bei DANN.

Zum Ausdruck:

```
NICHT (E 101 ODER E -102) UND REG 100 < 20
```

Damit dieser Ausdruck wahr ist, müssen beide Teilausdrücke (vor und nach dem UND) wahr sein.

Dabei ist der vordere Teilausdruck etwas verwirrend, doch bei genauerer Betrachtung findet man heraus: Dieser Teilausdruck ist dann wahr, wenn (E 101 ODER E -102) falsch - **E 101 inaktiv und E 102 aktiv** - ist.
Der hintere Teilausdruck ist erfüllt, wenn der Inhalt von **Register REG 100 kleiner als 20** ist.

Damit der gesamte Ausdruck wahr wird, müssen drei Elementarausdrücke erfüllt sein:

E 101=0 UND E 102=1 UND REG 100<20

3b)
```
SOBALD
     -E 101
     E 102
     REG 100
     <
     20
```

Dieses Programmstück hat also genau die gleiche Funktion wie das obige.

Man sieht hiermit auch, dass durch Vereinfachung solcher Ausdrücke sehr oft eine bessere Übersicht erreicht werden kann. Andererseits ist es manchmal, vom Bezug zur Maschine her, einfacher verständlich, wenn man den komplizierteren Ausdruck beibehält.

3.4 Arithmetische Ausdrücke

Diese Befehle können sowohl zur Bildung einer Eingangsbedingung nach SOBALD oder FALLS (arithmetischer Vergleich; siehe vorheriges *Kapitel 3.3 Boole'sche Ausdrücke*) als auch in einer Ausgangsanweisung (Zuweisung eines Rechenergebnisses auf ein Register) verwendet werden.

Der Aufbau und die Auswertung einer Formel ist in beiden Fällen identisch, mit Ausnahme dessen, dass bei einem arithmetischen Vergleich links vom Vergleichsoperator eine Zahl oder Verknüpfung stehen darf, eine Zuweisung jedoch immer nur auf ein Register erfolgen kann.

Die Zuweisung eines Wertes erfolgt durch das Gleichheitszeichen

Zuweisung: =

Zur Beschreibung von arithmetischen / logischen Ausdrücken dienen folgende Befehle :

arithmetische Operatoren: + - * /

logische Operatoren: WUND, WODER, WXODER

Zahlen: Binäre Zahlen
Dezimale Zahlen
Hexadezimale Zahlen

Variable: REG <Reg.Nummer>

3.4.1 Zahlen

Zahlen können auf drei verschiedene Arten eingegeben werden:

1. Dezimalzahlen mit **ZD** (Zahl dezimal)
2. Binärzahlen mit **ZB** (Zahl binär)
3. Hexadezimalzahlen **ZH** (Zahl hexadezimal)

Im Programmtext erfolgt die Unterscheidung der drei Zahlenformate dadurch, dass den Binärzahlen 'b' und den Hexadezimalzahlen 'h' vorangestellt wird.

Die Eingabe von Fließkomma-zahlen geschieht indirekt

Es können nur **ganze Zahlen** eingegeben werden. Die direkte Eingabe einer Fließkommakonstanten ist nicht möglich.

Entsprechend dem internen Format für Ganzzahlen (23 Bit und Vorzeichen) können siebenstellige Dezimalzahlen, sechsstellige Hexadezimalzahlen und vierundzwanzig-stellige Binärzahlen eingegeben werden.

3.4.2 Arithmetische Ausdrücke

Die Rechenzeichen **+ - / *** können direkt auf der Tastatur gewählt und damit eingegeben werden.

Hinweis:

10 Klammerebenen dürfen bei arithmetischen Ausdrücken verwendet werden !

Syntax:

```
REG x
=
REG y
+
REG z
/
5
-
20
```

Bedeutung:

Dem Register x wird der Wert der folgende Rechnung zugewiesen:

((Register y + Register z) / 5) - 20.

Wie man sieht, wird **nicht nach der Regel "Punktrechnung vor Strichrechnung"** gerechnet, sondern es wird einfach ein Schritt nach dem anderen, in der Reihenfolge, in der die Operationen dastehen, durchgearbeitet.

Die Steuerung nimmt immer ein Operationszeichen und die nächstfolgende Zahl (Operand) und wertet diese "Teilrechnung" aus, erhält also ein Zwischenergebnis. Anschließend geht sie von diesem Zwischenergebnis aus, nimmt das nächste Operationszeichen und den nächsten Operanden und wertet diese Teilrechnung wieder aus. Dies geht solange bis sie zum letzten Zahlenwert gelangt.

Also ein Ausdruck der Form: X = A + B - C / D - E * F

bedeutet folgendes: X = (((A + B - C) / D) - E) * F

Bei den folgenden Beispielen wird noch zwischen Ganzzahlregistern und Fließkommaregistern unterschieden, da sie zum Teil andere Möglichkeiten anbieten.

3.4.3 Zuweisungen auf Ganzzahlregistern

Nachkommastellen fallen bei Zuweisung auf Ganzzahlregister weg

Bei Zuweisungen auf Ganzzahlregister werden Nachkommastellen einfach weggelassen, d.h. es wird nicht gerundet!

Also: Reg 0 = 10 / 3 = 3,333...

-> Reg 0 enthält den Wert 3

Reg 0 = - 10 / 3 = - 3,333...

-> Reg 0 enthält den Wert -3

Beispiele:

1)
```
REG 100
=
h0000A8
+
b00000000000001001001110
```

Dem Register REG 100 wird die Summe aus hA8 (=168) und b1001001110 (=590) zugewiesen. Dies ergibt 758, was anschließend der Wert von REG 100 ist. (REG 100 = 758)

2)
```
REG 200
=
100
/
3
```

Dem Register REG 200 wird der Wert 100 / 3 (= 33.333...) zugewiesen. Da REG 200 nur ganze Zahlen speichern kann, fallen die Nachkommastellen weg und das Register hat anschließend der Wert 33. (REG 200 = 33)

3)

Man möchte die Zuweisung A = A - (B + C) / 2 berechnen, was ja nicht auf diese Weise (Punkt vor Strich) eingegeben werden kann.

Lösung:

Der Ausdruck (Rechnung) muss umgestellt werden, sodass die Steuerung bei der Bearbeitung des Ausdrucks (Schritt für Schritt) die gewünschte Reihenfolge ausführt.

Umformung: A - (B + C) / 2 = 0 - B - C / 2 + A

Der rechts stehende Ausdruck entspricht genau dem benötigten Ausdruck
(REG 1 = A, REG 2 = B, REG 3 = C). Weil man eine Rechnung nicht mit einem "-" (Minuszeichen) beginnen kann wurde hier noch ein kleiner Trick angewendet (0 - ...).

```
REG 1
=
0
-
REG 2
-
REG 3
/
2
+
REG 1
```

Das REG 1 (A) wird um den Wert (REG 2 + REG 3) / 2 (Durchschnitt von B und C) verkleinert. Der neue Wert wird wieder im REG 1 gespeichert.

3.4.4 Zuweisungen auf Gleitkommaregister

Die Gleitkommaregister werden im *Kapitel 3.6 Register und Merker* und *Kapitel 4.1 Grundsätzliches über Register und Merker)* genauer beschrieben. Hier nur einige kurze Hinweise:

Gleitkommaregister speichern Zahlen von -10^{15} bis $+10^{15}$

Die Genauigkeit der Rechnungen liegt bei 7 zählenden Stellen, da die Zahlen in einem 32 Bit breiten Speicher gespeichert werden.

Die verschiedenen PROZESS-SPS stellen folgende Fließkommaregisterbereiche zur Verfügung.

PROZESS-SPS	Fließkommaregisterbereich
PASE-E Plus	8960 bis 9215
DELTA	62208 bis 62463
NANO B	? bis ?
NANO A	-
MIKRO	-

In einem Programm können jedoch keine Gleitkommazahlen eingegeben werden. (z.B. DELTA-Register: REG 62208 = 1,456 kann **nicht** direkt eingegeben werden.)

Hinweis:

Durch die Zuweisung REG x = 1456 / 1000 kann der Wert 1,456 in ein Fließkommaregister geladen werden.

Die Regeln für die Bearbeitung von Ausdrücken sind genau gleich wie bei den Ganzzahlregistern. 10 Klammerebenen, keine Punkt vor Strichrechnung.

Beispiele :

Nummerierung der Register am Beispiel DELTA

1)
```
REG 62208
=
12345
/
1000
```

Dem Gleikommaregister (DELTA) REG 62208 wird der Wert 12,345 (= 12345 / 1000) zugewiesen.

2)
```
REG 62248
=
REG 0
/
REG 1
+
100
*
REG 62208
```

Dem Gleitkommaregister (DELTA) 62248 wird der Wert der folgenden Berechnung zugewiesen:

```
                REG 0
REG 62248 = ( ------- + 100 )  *  REG 62208
                REG 1
```

3.5 Task, Marken, Sprünge und Unterprogramme

3.5.1 Task, Marken und Sprünge

Die folgenden Befehle gehören zusammen und dienen zur Realisierung von unbedingten Sprüngen in einem Programm. Im Zusammenhang mit dem FALLS-Befehl können daraus bedingte Sprünge gemacht werden.

TASK **MARKE** **SPRUNG**

Die Marken kennzeichnen bestimmte Punkte im Programm, die mit einem Sprungbefehl angesprungen werden können.

TASK 0 bis 31

Task müssen mit 0 beginnen und lückenlos aufsteigend nummeriert werden

Diese Marken dienen zur **Kennzeichnung von parallelen Zweigen**. Sie müssen im Programm in aufsteigender Reihenfolge eingesetzt werden. Also zuerst TASK 0, dann TASK 1 usw. Es darf kein Task ausgelassen werden. Sie werden folgendermaßen dargestellt:

```
TASK  0 -----------------
```

MARKE 32 bis 999

Marken dienen als Sprungmarken oder kennzeichnen den Beginn eines Unterprogrammes

Diese Marken dienen rein als Knotenpunkte oder im Zusammenhang mit Unterprogrammen als Kennzeichnung für den Unterprogrammanfang. Sie werden ohne den waagrechten Strich dargestellt:

```
MARKE  32
```

Sprünge

SPRUNG-Befehle können indirekt definiert werden

Nach einer SPRUNG - Anweisung wird die Ausführung des Programms an der Marke fortgesetzt, deren Nummer im Sprungbefehl als Parameter angegeben ist. Die Markennummer kann auch indirekt angegeben werden, also mit REG x, was bedeutet: Der Sprung soll zu derjenigen Marke erfolgen, deren Nummer im Register REG x gespeichert ist.
Es muss vor allem darauf geachtet werden, dass nicht auf eine Marke gesprungen wird, die einem anderen Task zugeordnet ist.

Hinweis:

Legen Sie für jeden Task spezielle Markennummern fest. Also zum Beispiel

TASK 0: Marken 100 bis 199

```
TASK  1:   Marken 200 bis 299
TASK  2:   Marken 300 bis 399
....
```

Syntax:

```
TASK  0 ---------------------
...
MARKE  100
...
...
SPRUNG 100
...
...
SPRUNG 0
```

Bedeutung

Die Programm-ausführung beginnt immer bei TASK 0

Die Programmausführung beginnt bei TASK 0 und arbeitet die Anweisungen ab, bis der Befehl SPRUNG 100 erreicht wird. Damit wird zur MARKE 100 gesprungen. Steht diese Sprunganweisung im Ausgangszweig einer FALLS-Bedingung, so wird der Sprungbefehl eventuell nicht ausgeführt und das Programm läuft weiter, bis der SPRUNG 0 erreicht wird.

Hinweis:

Es muss darauf geachtet werden, dass dieser zweite Sprung nicht aus irgendeinem Grund übersprungen wird. Dies hättet zur Folge, wenn mehrere Tasks existieren, dass der TASK 0 in den TASK 1 hineinläuft.

Beispiel:

```
1)          LADE_REGISTER [ rZähler mit 10]
      MARKE  sSchleife
             ...
             ...
             ...
             REGDEC rZähler
        FALLS
             NICHT
             REGNULL rZähler
```

```
          DANN
            SPRUNG sSchleife
          DANN
            ...
```

In diesem Beispiel ist eine Schleife realisiert, welche 10 mal durchlaufen wird.

Das Register REG rZähler, welchem zu Beginn der Wert 10 zugewiesen wird, wird bei jedem Schleifendurchlauf um eins verkleinert (REGDEC rZähler). Anschliessend wird noch verglichen, ob es den Wert 0 bereits erreicht hat. Wenn nicht, so wird wieder zum Beginn der Schleife (MARKE sSchleife) gesprungen. Ansonsten wird das Programm nach dem zweiten DANN fortgesetzt.

3.5.2 Unterprogramme

Die Befehle

UNTERPROGRAMM **RÜCKSPRUNG**

gehören ebenfalls zusammen und dienen der Realisierung von Unterprogrammen.

Unterprogramme sind Programmteile, welche von einem beliebigen Punkt im Programm angesprungen werden können. Diese Teilprogramme werden dann abgearbeitet und nach Beendigung wird wieder an jenen Ort im Programm gesprungen, von wo das Unterprogramm aufgerufen wurde.

Hinweis:

20 Unterprogrammebenen sind zulässig.

Die Verwendung von Unterprogrammen führt zu übersichtlicheren und kompakteren Programmtext

Dadurch, dass der Rücksprung aus einem Unterprogramm stets zu der Programmstelle zurückführt, von der das Unterprogramm aufgerufen wurde, brauchen Programmteile, welche oft und an verschiedenen Stellen im Programm benötigt werden, nur einmal geschrieben werden. Dies führt zu einer Einsparung von Speicherplatz und trägt zu einer besseren Übersicht bei.

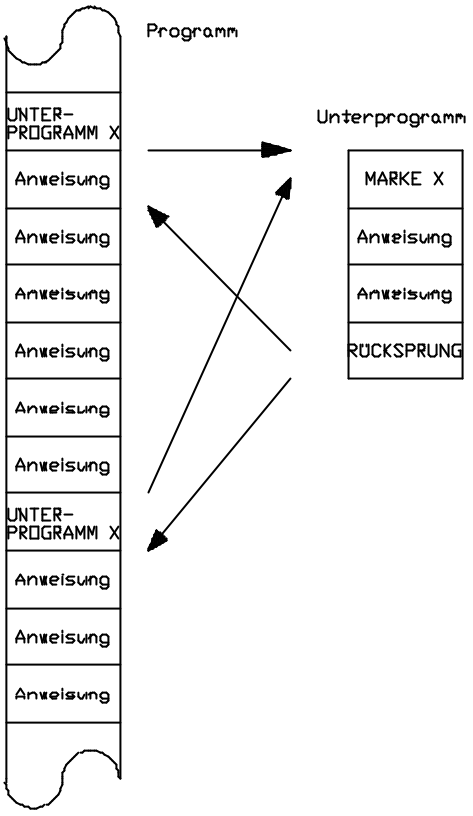


Abbildung 14: Unterprogamme

Syntax:

```
DANN
  UNTERPROGRAMM seFehlerBehandlung
  ...
  ...

MARKE seFehlerBehandlung
  ...
  ...
  ...
RÜCKSPRUNG
```

Bedeutung

Kommt das Programm zum Befehl UNTERPROGRAMM seFehlerBehandlung, dann wird sofort zur MARKE seFehlerBehandlung gesprungen. Dort werden die Anweisungen, abgearbeitet, bis der Befehl RÜCKSPRUNG erreicht wird. Sobald dieser auftritt, wird wieder zu der Stelle zurückgesprungen, wo das Unterprogramm aufgerufen wurde und mit dem folgenden Befehl fortgefahren.

Das Unterprogramm kehrt nach Ausführung an den Ort des Aufrufes zurück

Der Unterschied zum SPRUNG - Befehl liegt darin, dass sich das Programm den Ort merkt, von wo es weggesprungen ist, um anschließend wieder dorthin zurückzuspringen.

Indirekter Aufruf von Unterprogrammen

Auch beim UNTERPROGRAMM - Befehl ist die indirekte Adressierung möglich. So kann ein Unterprogramm folgendermaßen aufgerufen werden:

```
UNTERPROGRAMM R(100)
```

oder

```
UNTERPROGRAMM R(sZeiger)
```

Dabei wird dasjenige Unterprogramm aufgerufen, welches bei der Marke beginnt, deren Nummer in Register REG 100 bzw. REG sZeiger gespeichert ist.

Regeln im Zusammenhang mit Unterprogrammen

1. Es sind 20 **Unterprogrammebenen** zugelassen.

2. Aus einem Unterprogramm darf nicht hinausgesprungen werden. Es muss immer mit dem Rücksprungbefehl beendet werden. Ansonsten kann kein neues Unterprogramm mehr aufgerufen werden.

3. Ein und dasselbe Unterprogramm darf von verschiedenen Parallelzweigen angesprungen werden. Je nach Programm sogar gleichzeitig.

Beispiel:

```
      TASK  Initialisierung
--------------------
            ...
            ...
            UNTERPROGRAMM sGlobal
            ...
            ...
      TASK  Automatik
--------------------------
            ...
            ...
            ...
            UNTERPROGRAMM sGlobal
            ...
      TASK  Ein-/Ausgabe
----------------------
            ...
            UNTERPROGRAMM sGlobal
            ...
            SPRUNG 2


     MARKE  sGlobal
            ...
            ...        * Unterprogrammtext *
            ...
            RÜCKSPRUNG
            Programmende
```

Es wird in diesem Beispiel aus allen drei Zweigen einmal das Unterprogramm am Ende des letzten Task (TASK Ein-/Ausgabe) aufgerufen. Dies ist wie beschrieben durchaus zulässig, kann jedoch bei **gleichzeitigem** Aufruf, je nach Unterprogramm zu unerwünschten Effekten führen.

Zum Beispiel wenn im Unterprogramm verschiedene Register angezeigt werden sollen, was mit der indirekten Registerangabe (beim ANZEIGE_REG - Befehl) ja kein Problem ist, so kann das in diesem Fall zur Anzeige von einem "falschen" Register führen.

Hinweis:

Globale Unterprogramme, also Unterprogramme die von mehreren Task aufgerufen werden, müssen am Ende des letzten Task stehen.

Bei gleichzeitigem Aufruf des gleichen Unterprogramms ist also Vorsicht geboten. Wenn nötig kann eine Koordination der Unterprogrammaufrufe auch über Merker geregelt werden.

3.5.3 Funktionen

Die Befehle

DEF_FUNKTION[<Funktion>, xy] **END_DEF**

Funktionen werden im Programmkopf definiert

Der Programmierer definiert, wie bei Hochsprachen gewohnt, die Funktion im Kopf des Programmes und kann diese dann im Programmtext beliebig aufrufen. Funktionen sind den Unterprogammen ähnlich jedoch wesentlich vielseitiger.

- Funktionen können mit Übergabeparametern aufgerufen werden.
- Funktionen können mit Rückgabeparametern definiert werden.
- Funktionen haben lokale Variablen und Marken.
- Funktionen können in Bedingungen als Boolean oder in Ausgangsanweisungen eingesetzt werden.
- Funktionen unterscheiden sich im Aufruf nicht von Systembefehlen.

Befehlsbibliotheken können mit Hilfe der Funktionen erstellt werden

Der Programmierer kann sich mit Funktionen eigene, anwendungsspezifische Befehlsbibliotheken erstellen.

Definieren der Funktion

Funktionen werden vor TASK 0 definiert

Die Funktionen werden am Programmanfang (vor dem TASK 0) definiert. Zunächst werden die Rahmendaten in einem Definitionsfenster (Aufruf mit dem Tastenkürzel (D) dann (F)) spezifiziert.

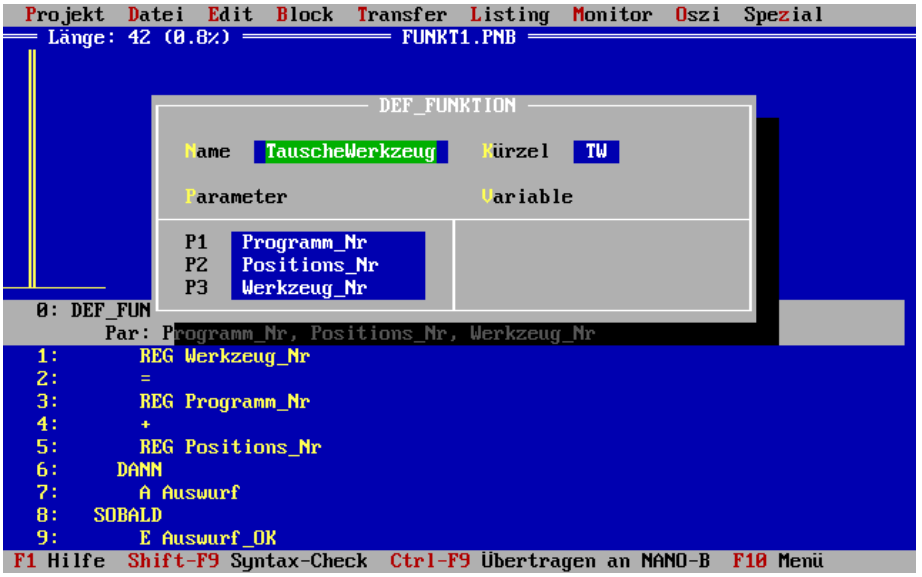


Abbildung 15: Mit Hilfe der Definitionsfenster werden die Funktionen parametriert

Definieren des Funktionstextes

Funktionen werden mit END_DEF abgeschlossen

Nach dem Quittieren mit (¢) wird der Funktionkopf auf dem Bildschirm sichtbar, danach können Sie den Funktionstext eingeben. Der Funktionstext wird mit END_DEF abgeschlossen.

Funktionsaufruf im Programmtext

Mit dem Befehlskürzel werden Funktionen im Programmtext aufgerufen

Mit dem Funktionskürzel das im Definitionsfenster eingeben wurde (siehe Abbildung oben) wird die Funktion im Programmtext aufgerufen.

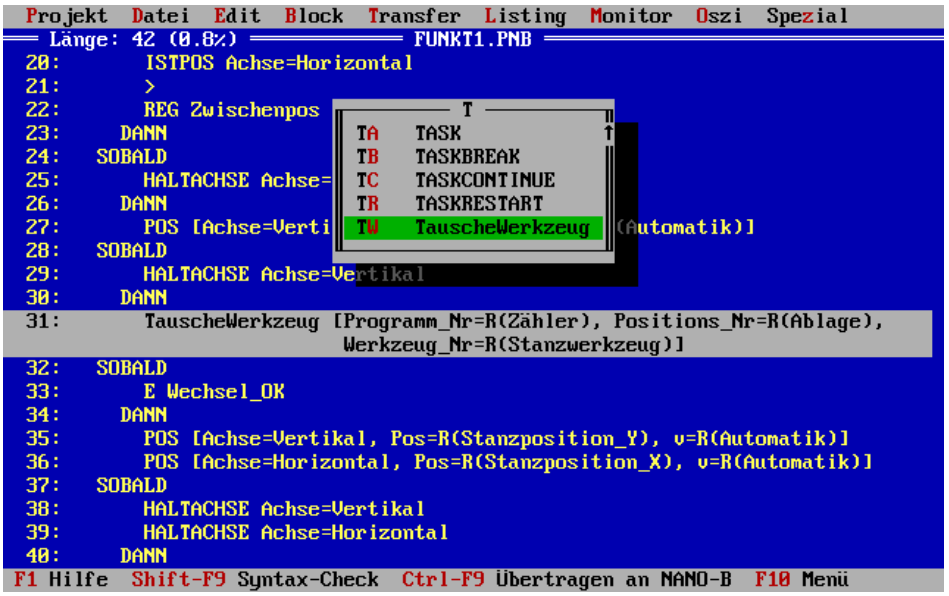


Abbildung 16: Der Aufruf erfolgt über Funktions- (Befehls-) Kürzel, wie von einem Befehlsaufruf gewohnt

Beispiel 1 (in einer Ausgangsanweisung):

Mit DEF_FUNKTION beginnt die Funktion

Mit END_DEF endet die Funktions-definition

```
0: DEF_FUNKTION [TauscheWerkzeug, TW]
         Par: Programm_Nr, Positions_Nr,
              Werkzeug_Nr
   1:        REG rWerkzeug_Nr
   2:        =
   3:        REG rProgramm_Nr
   4:        +
   5:        REG rPositions_Nr
   6:      DANN
   7:        A rAuswurf
   8:    SOBALD
   9:        E rAuswurf_OK
  10:      DANN
  11:        A rWerkz_Einlegen
  12:        RÜCKSPRUNG
  13: END_DEF
  14: TASK tInitialisierung ------------------
--
  15:    SOBALD
  16:        E eRückmeldung
  17:      DANN
  18:        POS [Achse=axHorizontal,

Pos=R(rAblegen),v=R(rAutomatik)]
```

```
 19:   SOBALD
 20:       ISTPOS Achse=axHorizontal
 21:       >
 22:       REG rZwischenpos
 23:     DANN
 24:   SOBALD
 25:       HALTACHSE Achse=axHorizontal
 26:     DANN
 27:       POS
[Achse=axVertikal,Pos=R(rAblegen)
                 v=R(rAutomatik)]
 28:   SOBALD
 29:       HALTACHSE Achse=axVertikal
 30:     DANN
 31:       TauscheWerkzeug
           [Programm_Nr=R(rZähler),
            Positions_Nr=R(rAblage),
            Werkzeug_Nr=R(rStanzwerkzeug)]
 32:   SOBALD
 33:       E eWechsel_OK
 34:     DANN
 35:       POS [Achse=axVertikal,
                Pos=R(rStanzposition_Y),
                v=R(rAutomatik)]
 36:       POS [Achse=axHorizontal,
                Pos=R(rStanzposition_X),
                v=R(rAutomatik)]
 37:   SOBALD
 38:       HALTACHSE Achse=axVertikal
 39:       HALTACHSE Achse=axHorizontal
 40:     DANN
 41:       A aStanzen1
 42:       ...
Programmende
```

Mit Tausche-Werkzeug wird die Funktion aufgerufen

Beispiel 2 (in einer Eingangsbedingung):

In diesem Beispiel wird abgefragt, ob die Startbedingung `StartBedErfüllt` erfüllt ist. Sobald Dies der Fall ist startet der TASK `tAusbruchStanze`.

```
0: DEF_FUNKTION [StartBedErfüllt, SE]
         Par: rMindestPos
   1:   FALLS
   2:       E eTürGeschlossen
   3:       E eStartSignal
   4:       MERKER mGlobaleFreigabe
   5:       REG rMindestPos
   6:       >
   7:       1000
   8:     DANN
   9:       REG StartBedErfüllt
  10:       =
  11:       1
  12:     SONST
  13:       REG StartBedErfüllt
  14:       =
  15:       0
  16:     DANN
  17:       RÜCKSPRUNG
  18: END_DEF
  19: TASK tAusbruchStanze --------------------
--
  20:   SOBALD
  21:       StartBedErfüllt
                 [rMindestPos=rIstPosition]
  22:     DANN
  23:       ...
 Programmende
```

3.6 Register und Merker

Die folgenden Befehle dienen zum Umgang mit Registern und Merkern und werden in diesem Kapitel erklärt.

LADE_REGISTER

KOPIERE

SPEZIALFUNKTION Nr.x

REGDEC

REGINC

REGNULL

REG_LÖSCHEN

BIT_SETZ

BIT_LÖSCH

MERKER

LÖSCHE_MERKER

3.6.1 Grundsätzliche Informationen zu den Registern

Register-definition durch nummerischen Parameter oder symbolischen Variablen-namen

Die Register sind die Zahlenspeicher der PROZESS-SPS. Sie können wie Variablen benutzt werden. Man kann ihnen einen Wert zuweisen und später wieder darauf zurückgreifen. Es wird zwischen Ganzzahlregistern, Gleitkommaregistern und Spezialregistern unterschieden. Alle Register werden durch eine Nummer oder einen symbolischen Variablennamen gekennzeichnet.

Bei allen Registern hat man die Möglichkeit, sie indirekt zu adressieren. Das heißt die Nummer eines gewünschten Registers befindet sich in einem anderen Register. Siehe dazu Befehl LADE_REGISTER.

Ganzzahlregister:

Bei diesen Registern handelt es sich um 24 Bit breite Register, in denen eine **ganze Zahl** zwischen **-8388608 und 8388607** gespeichert wird.

Die verschiedenen PROZESS-SPS stellen folgende Ganzzahlregisterbereiche zur Verfügung.

PROZESS-SPS	Ganzzahlregisterbereich
PASE-E Plus	0 bis 8191 **mit Speichererweiterung zusätzlich** MEM128: 200000 bis 232767 MEM512: 200000 bis 331071 MEM1024:

	200000 bis 462143
DELTA	0 bis 20479
NANO-C	0 bis 1999 und 20000 bis 27999
NANO-B	0 bis 1999
NANO-A	0 bis 199
MIKRO	100 bis 1099

Auch als Parameter bei verschiedenen Befehlen können diese Ganzzahlregister angegeben werden. Statt einem bestimmten Parameter wird dann zum Beispiel R100 angegeben. Für den Befehl hat dies zur Auswirkung, dass er den aktuellen Speicherinhalt des Registers REG 100 als Parameter benutzt.

Gleitkommaregister:

Diese Register sind 32 Bit breit und speichern reelle Zahlen, also Fließkommazahlen in einem Bereich von

$$\mathbf{-10^{15}\ bis\ +10^{15}.}$$

Betragsmäßig liegt die kleinst mögliche Zahl bei $1{,}2*10^{-15}$

Aus der Zahlendarstellung von 32 Bit, ergibt sich die Genauigkeit der Berechnungen zu 7 zählenden Dezimalstellen.

Die verschiedenen PROZESS-SPS stellen folgende Fließkommaregisterbereiche zur Verfügung.

PROZESS-SPS	Fließkommaregisterbereich
PASE-E Plus	8960 bis 9215
DELTA	62208 bis 62463
NANO-C	65024 bis 65279
NANO B	-
NANO A	-
MIKRO	-

Spezialregister:

Die Spezialregister und somit die Funktionen des Betriebssystems sind mit Vorsicht zu behandeln

Die Spezialregister beinhalten Parameter, welche vom Betriebssystem genutzt werden. Diese bieten vielfältige Möglichkeiten, auf das Verhalten der Steuerung Einfluss zu nehmen und sollten aus diesem Grund auch nur vorsichtig angewandt werden!

Slaveregister:

Diese Register befinden sich auf intelligenten Modulen bzw. Karten der verschiedenen PROZESS-SPS. Sie dienen der Kommunikation zwischen der CPU und dem Prozessor des Moduls respektive der Karte. So werden vom Steuerungs-Programm Befehle und Parameter in diese Register geschrieben und Statusmeldung ausgelesen. Der Zugriff auf die Slaveregister erfolgt durch die gleichen Befehle wie für alle anderen Register. Einige dieser Register speichern Zustände oder momentane Werte (z.B. die Istposition einer Achse) und sind aus diesem Grund als **nur lese** (read only) Speicher ausgeführt. Das Beschreiben dieser Register ist nicht erlaubt !

PROZESS-SPS	Slaveregisterbereich
PASE-E Plus	Steckpl.1: 11100 bis 11799 Steckpl.2: 12100 bis 12799 ... Steckpl.32:42100 bis 42799
DELTA	Steckpl.1: 21000 bis 24999 Steckpl.2: 31000 bis 34999 Steckpl.3:. 41000 bis 44999
NANO-B	CPU: 11100 bis 11999 Modul 2: 12200 bis 12999 Modul 3: 13300 bis 13999 Modul 4: 14400 bis 14999
NANO-A	-
MIKRO	1100 bis 1149

3.6.2 Befehle, um Register zu laden

Der Befehl

LADE_REGISTER [x mit a]

dient zum Laden von Zahlenwerten (oder Inhalten anderer Register) in ein Register.

Beschreibung:

Im oben dargestellten Befehl gibt x die Nummer des Registers an, in das die Zahl a hineingeschrieben werden soll.

Indirekte und doppelt indirekte Adressierung

Durch Drücken (1, 2 mal) der (SPACE) Taste werden die Indirektlevel gewählt

Für das "x" und das "a" im obigen Befehl kann aber nicht nur eine Zahl stehen, sondern man kann auch ein Register spezifizieren : Man muss nur durch Drücken der Space-Taste der Registernummer ein "R" voranstellen.
Wird "Ry" anstatt "x" geschrieben, so wird der Wert "a" in das Register geschrieben, dessen Nummer im Register y steht.
Steht "Rb" anstelle von "a", hat dies zur Folge, dass nicht der Wert selbst, sondern der Inhalt des spezifizierten Registers in das Register x (oder Ry) geladen wird.

Gibt man für "a" nun "RR" (2 mal Space-Taste) und dann eine Zahl (b) ein

```
LADE_REGISTER [ x mit RR(b)]
```

so hat dies folgende Auswirkung: Zuerst wird der Wert des Registers mit der Nummer b gelesen. Dieser Wert dient nun als Registernummer. Also wird im Register, welches diesen Wert als Nummer hat, ein neuer Wert

gelesen und schließlich dieser neue Wert im Register x abgespeichert.

Beispiele:

1) Laden einer Zahl in ein Register

```
LADE_REGISTER [ rNeuePosition mit 1280]
```

Der Wert 1280 wird ins REG rNeuePosition geladen.

2) Kopieren eines Registers auf ein anderes

```
LADE_REGISTER [ rSpannung mit R(rSpannung1)]
```

Der Wert, welcher sich im Register REG rSpannung1 befindet wird ins REG rSpannung geladen. Mit anderen Worten, der Inhalt von REG rSpannung1 wird ins REG rSpannung kopiert.

3a) Laden mit doppelt indirekter Adressierung

```
LADE_REGISTER [ rSpannung mit RR(rU_Zeiger)]
```

Derjenige Wert, welcher im Register mit der Nummer, die im Register REG rU_Zeiger ist, wird ins REG rSpannung geladen.

3b) Zahlenbeispiel zur doppelt indirekten Adressierung:

Registerbelegung	Wert
REG 64	211
REG 211	70035
REG 5000	4711
REG 4711	beliebig

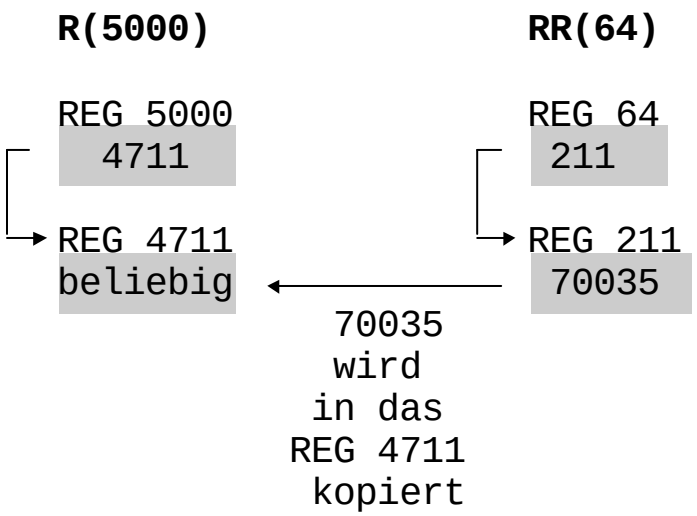
mit dieser Belegung wird nun folgender Befehl ausgeführt:

```
LADE_REGISTER [R(5000)  mit RR(64)]
```

Daraus resultieren folgende Registerwerte:

Register 64 = 211 (bleibt gleich)
Register 211 = 70035 (bleibt gleich)
Register 5000 = 4711 (bleibt gleich)
Register 4711 = R5000 = RR64 = R211 = 70035

Schaubild:



Der Befehl

KOPIERE [n=<Anz. Reg> von <Quellreg> nach <Zielreg>]

dient dazu, um ganze Registerblöcke zu kopieren. Man muss nur die Anzahl der Register angeben, die Nummer des ersten Registers, welches kopiert werden soll, und schließlich noch die Nummer des Registers, wo das erste Register hinkopiert werden soll.

Indirekte Parameter-angabe möglich

Alle diese drei Parameter können auch **einfach indirekt** angegeben werden.

Beispiel:

```
KOPIERE [n=5, von 100 nach 200]
```

Reg.Nr.	Inhalt	Ausführung	Reg.Nr.	Inhalt
100	77	---------->	200	77
101	3198	---------->	201	3198
102	791	---------->	202	791
103	86320	---------->	203	86320
104	13629	---------->	204	13629

Die obige Darstellung soll veranschaulichen, was beim Kopiere-Befehl passiert: Es werden in diesem Fall fünf Register kopiert (n=5). Das erste zu kopierende Register ist das REG 100 und dieses wird aufs REG 200 kopiert. In der Darstellung wurden beliebige Werte für die REG 100 bis REG 104 angenommen. Wichtig ist, dass nach dem Kopier-Befehl dieselben Werte in den REG 200 bis REG 204 erscheinen.

Beispiele:

1) **KOPIERE [n=100, von 0 nach 1000]**

Die Register REG 0 bis REG 99 werden auf die Register REG 1000 bis REG 1099 kopiert.

2) **KOPIERE [n=100, von 700 nach 650]**

REG 700 bis REG 799 werden auf REG 650 bis REG 749 kopiert. Hier bekommen also die Register REG 700 bis REG 749 neue Werte und die alten gehen "verloren". (Sie sind jetzt in den Registern REG 650 bis REG 699 gespeichert.)

SPEZIAL-FUNKTION 1 zur Initialisierung von Achsen

Der Befehl

SPEZIALFUNKTION [#1, p1=a, p2=b]

dient auch zum Kopieren von Registerinhalten. Es können max 99 Register kopiert werden, die Zielregister müssen in diesem Fall jedoch nicht in einem festen Block sein. Es können vielmehr auch einzelne Register geladen werden.

Es gibt noch zwei weitere Spezialfunktionen, welche im *Kapitel 3.11.4 Spezialfunktionen* erklärt sind. Die Spezialfunktion 1 dient wie bereits erwähnt zum Kopieren und wird deshalb hier beschrieben.

Die Spezialfunktion 1 wurde im Hinblick auf das Initialisieren von Achskarten entwickelt, weil dort sehr viele Register geladen werden müssen. Die Funktion hat zwei Parameter: Der erste (p1) gibt die Nummer des ersten Registers eines Beschreibungsblocks an, (p2) gibt das erste zu beschreibende Register (b) an. Die Parameter p1 und p2 können auch indirekt adressiert werden. Dieser Beschreibungsblock enthält alle Angaben darüber, wieviele und welche Register kopiert werden sollen und zwar nach folgenden Konventionen:

Beschreibungsblock

Reg.Nr.	Inhalt
a	Anzahl zu kopierender Register
a + 1	1. Offset Registernummer
a + 2	Inhalt des 1. Registers
a + 3	2. Offset Registernummer
a + 4	Inhalt des 2. Registers
a + 5	usw.

Für jedes zu beschreibende Register werden zwei Register des Beschreibungsblocks gebraucht. Zum Beispiel Register a+1 und Register a+2. Diese beiden Zahlen haben zur Folge, dass ins Register mit der Nummer b+R(a+1) der Wert R(a+2) kopiert wird. Dieser Sachverhalt kann am besten anhand eines Beispiels eingesehen werden:

Beispiel:

```
SPEZIALFUNKTION [ #1, 100, 1000]
```

Der Beschreibungsblock beginnt also bei Register 100. Für das Beispiel sollen folgende Werte in den Registern 100 und Folgende sein:

Reg 100 = 4	Reg 103 = 11	Reg 106 = 912
Reg 101 = 10	Reg 104 = 199	Reg 107 = 19
Reg 102 = 4500	Reg 105 = 15	Reg 108 = 9999

Nach Ausführung der Spezialfunktion enthalten folgende Register die Werte:

Reg 1010 = 4500
Reg 1011 = 199
Reg 1015 = 912
Reg 1019 = 9999

Der Befehl

REG_LÖSCHEN [vonReg<erstes Reg> bis <letztesReg>]

dient zum Löschen von Registerinhalten. Es kann ein beliebig großer Registerblock zu Null gesetzt werden. Es werden alle Register einschließlich dem ersten Register und dem letzten Register zu Null gesetzt.

Beispiel:

```
REG_LÖSCHEN [100  bis 200]
```

Alle Register von 100 bis 200 werden zu Null gesetzt.

3.6.3 Rechnen mit Registern

Das Rechnen mit Registern ist ausführlich im Kapitel über arithmetische Ausdrücke (*Kapitel 3.4 Arithmetische Ausdrücke*) geschildert. Hier sollen vorallem noch die Befehle

REG <RegNr>

REGNULL <RegNr>

REGDEC <RegNr>

REGINC <RegNr>

erläutert werden. Bei allen diesen vier Befehlen ist es möglich, die Registernummer, welches jeweils der einzige anzugebende Parameter ist, indirekt zu spezifizieren. Also es kann für RegNr zum Beispiel R100 stehen. Dies bedeutet, dass für den Befehl dasjenige Register gewählt wird, dessen Nummer im Register 100 steht.

Der Befehl

REG

Dieser Befehl greift auf den Wert eines Registers direkt zu und kann wie eine Variable behandelt werden. In einer Ausgangsanweisung wird dem Register, welches auf der linken Seite des Gleichheitszeichens steht, ein Wert zugewiesen. In einer Eingangsbedingung wird der Inhalt des Registers gelesen. Die rechts des Gleichheitszeichens stehenden Registerzugriffe bewirken in beiden Fällen ein Lesen des Registers.

Beispiele:

1)
```
DANN
   REG 1
   =
   REG 105
   *
   25
```

Dieses Beispiel zeigt eine Zuweisung (Ausgangsanweisung eingeleitet durch DANN). Dabei wird das Register REG 105 gelesen und sein Inhalt mit der Zahl 25 multipliziert. Das Ergebnis dieser Rechenoperation wird im Register 1 gespeichert. Der Inhalt von REG 105 bleibt unverändert erhalten.

2)
```
FALLS
     REG 1
     =
     REG 105
     *
     25
   DANN
```

In diesem Fall steht der Ausdruck REG 1 = REG 105 * 25 nicht in einer Ausgangsanweisung, sondern er bildet eine Eingangsbedingung. Bei diesem Programmteil wird der Wert des Registers 1 nicht verändert. Er wird nur mit dem Produkt REG 105 * 25 verglichen. (siehe

auch Boolesche Ausdrücke, *Kapitel 3.3 Boole'sche Ausdrücke*)

Der Befehl REGNULL setzt ein Register zu 0 oder fragt ab, ob ein Register 0 ist:

REGNULL <RegNr>

Dieser Befehl hat als Eingangsbedingung (nach FALLS oder SOBALD) folgende Bedeutung. Er soll anhand des folgenden Beispiels erläutert werden:

Beispiel:

```
FALLS                    FALLS
   REGNULL 49               REG 49
  DANN                      =
                            0
                          DANN
```

Diese beiden Programmteile führen dieselbe Funktion aus. Rechts wird der Vergleich als allgemeiner arithmetischer Vergleich ausgeführt und links wird der spezielle Befehl REGNULL benutzt.

Die Befehle

REGDEC REGINC

Diese beiden Befehle dienen dazu, ein Register um 1 zu erniedrigen (dekrementieren) respektive um 1 zu erhöhen (inkrementieren). Diese Funktionen werden oft in Schleifen zum Erhöhen oder Verringern von Zählern und Zeigern verwendet.

Beispiele:

1a)
```
DANN
```

1b)
```
DANN
```

```
REGDEC 100
```

```
REG 100
 =
 REG 100
 -
 1
```

Diese beiden Programmteile haben dieselbe Funktion. Bei beiden wird der Inhalt des Registers 100 um 1 verkleinert.

2a)

```
DANN
  REGINC 88
```

2b)

```
DANN
  REG 88
  =
  REG 88
  +
  1
```

Auch hier haben beide Programmteile genau dieselbe Auswirkung. Das Register 88 wird um 1 erhöht.

3)

```
      LADE_REGISTER [ rZähler mit 10]
MARKE  55
       ...
       REGDEC rZähler
   FALLS
       REGNULL rZähler
     DANN
     SONST
       SPRUNG 55
     DANN
```

Auf diese Art kann nun eine Schleife realisiert werden, welche eine bestimmte Anzahl von Durchläufen ausführt. In der Schleife wird das "Zählregister" immer um eins dekrementiert und schließlich verglichen ob es 0 ist (`REGNULL rZähler`). Falls es Null ist, so wird beim ersten DANN nichts gemacht, also unverzüglich zum zweiten DANN gesprungen und mit dem weiteren Programm fortgefahren. Ist das Register 1 jedoch noch nicht 0, so wird zurück zum Schleifenanfang gesprungen.

3.6.4 Registerbit-Befehle

Mit den Befehlen

BIT_SETZ

BIT_LÖSCH

können einzelne Bits von Registern abgefragt, gesetzt oder gelöscht werden.

Dabei bedeutet der Befehl

BIT_SETZ [Reg. <RegNr>, Bit <BitNr>]

als **Ausgangsanweisung** (nach DANN oder SONST):
Das beschriebene Bit soll gesetzt werden, es soll ihm also der Wert 1 zugewiesen werden.

als **Eingangsbedingung** (nach FALLS oder SOBALD):
Es wird abgefragt, ob das beschriebene Bit gesetzt ist, also ob es den Wert 1 hat.

BIT_LÖSCH [Reg. <RegNr>, Bit <BitNr>]

als **Ausgangsanweisung** (nach DANN oder SONST):
Das beschriebene Bit wird zu Null gesetzt.

als **Eingangsbedingung** (nach FALLS oder SOBALD):
Es wird abgefragt, ob das beschriebene Bit Null ist. Ist es Null, so wird der Ausdruck wahr, ansonsten wird er falsch.

Die Registernummer kann auch hier indirekt angegeben werden, die Bitnummer jedoch nicht.

Bezeichnung der Bits (Numerierung):

Ganzzahlregister (24 Bit):

23	22	21	20	19	18	17	16	15	14	13	12	...

...	11	10	9	8	7	6	5	4	3	2	1	0

Bit 0 ist das Bit mit der kleinsten Wertigkeit.
Bit 23 ist dasjenige mit der größten Wertigkeit

Die interne Zahlendarstellung erfolgt im Zweierkomplement

Die interne Zahlendarstellung erfolgt im Zweierkomplement. Beispiel:

+10	=	0000...1010
+1	=	0000.....01
0	=	0.....0
-1	=	1111.....11
-10	=	1111...0110

Bit 23 ist bei negativen Zahlen 1

Bit 23 ist bei negativen Zahlen 1.

Die Wertigkeit kann so berechnet werden: $W = 2^{BitNr}$

Beispiele:

1)
```
DANN
   BIT_SETZ [Reg.12, Bit 3]
```

Das vierte Bit von Register 12 wird gesetzt, das heißt es hat anschließend den Wert 1. Sind alle anderen Bits dieses Registers Null, so hat das Register 12 nachher den Wert 8.

2)
```
LADE_REGISTER [1 mit 0]
 BIT_SETZ [Reg.1, Bit 9]
 BIT_SETZ [Reg.1, Bit 8]
 BIT_SETZ [Reg.1, Bit 6]
 BIT_SETZ [Reg.1, Bit 3]
 BIT_SETZ [Reg.1, Bit 2]
 ...
```

Dieses Programm setzt bestimmte Bits vom Register 1. Der daraus folgende Wert für das Register 1 kann wie folgt (Aufsummierung der "gesetzten Wertigkeiten") berechnet werden:

$2^9 + 2^8 + 2^6 + 2^3 + 2^2 = 512 + 256 + 64 + 8 + 4 =$ 844

3a)
```
SOBALD
    BIT_SETZ [Reg.1, Bit12]
  DANN
    ...
```

3b)
```
SOBALD
    NICHT
    BITLÖSCH[Reg.1, Bit12]
  DANN
    ...
```

Diese Programme haben genau die gleiche Auswirkung. Es wird gewartet, bis das 12. Bit vom Register 1 gesetzt (nicht gelöscht) ist.

3.6.5 Merker und Merker -Befehle

Merker haben den Wert 1 oder 0

Merker sind im Prinzip Speicher, welche jedoch nur ein Bit breit sind. Sie können also entweder den Wert 1 oder 0 haben. Aus diesem Grund sagt man auch ein Merker ist gesetzt (= 1) oder gelöscht (= 0). Wie die Register sind auch die Merker mit Nummern gekennzeichnet. Die Numerierung der Merker kann im entsprechenden Steuerungshandbuch nachgeschlagen werden. Die Nummern von 2048 bis 2303 sind für Spezialmerker, welche vom System gebraucht werden, reserviert. Die für den Anwender nützlichen Spezialmerker sind im *Kapitel 4.1.2 Merker* beschrieben.

Überlagerter Merker-Register-Bereich

Außerdem gibt es einen Merkerbereich der mit einem Registerbereich überlagert ist. Zur Numerierung siehe das entsprechende Steuerungshandbuch. Zur näheren Ausführung siehe *Kapitel 4.1 Grundsätzliches über Register und Merker*.

Es stehen folgende Merkerbefehle zur Verfügung:

MERKER

LÖSCHE_MERKER

Der Befehl

MERKER <MerkerNr>

hat folgende Bedeutung:

als **Ausgangsanweisung** (nach DANN oder SONST):

Der Merker wird gesetzt, erhält also den Wert 1.
Durch Eingeben eines "-" (Minuszeichen) vor der Merkernummer kann der Merker gelöscht werden.

als **Eingangsbedingung** (nach SOBALD oder FALLS):

Es wird abgefragt, ob der Merker gesetzt ist. Das Ergebnis dieser Abfrage entspricht dem Wert des Merkers, mit der Bedeutung wahr (= 1) oder falsch (= 0).

Auch hier kann ein Minuszeichen vor der Markennummer eingegeben werden, womit man auf den inversen Wert des Merkers zugreifen kann. Es wird also abgefragt, ob der Merker gelöscht ist. (Das Ergebnis ist also genau der inverse Wert des Merkers.)

Beispiele:

1)
```
DANN
   -MERKER 2
```

Diese Anweisung löscht den Merker 2. Der Merker 2 hat also anschließend den Wert 0.

2)
```
SOBALD
     -MERKER 61
   DANN
```

Es wird gewartet bis der Merker 61 den Wert 0 hat. Dieser Merker kann zum Beispiel von einem anderen Task verändert also gelöscht werden. Ist der Merker bereits 0, wenn das Programm zu diesem Befehl kommt, dann wird unverzüglich mit dem weiteren Programmablauf fortgefahren.

Der Befehl

LÖSCHE_MERKER

dient zum Löschen von ganzen Merker-Blöcken. Das folgende Beispiel soll das noch illustrieren.

Beispiel:

```
LÖSCHE_MERKER [1 bis 300]
```

Dieser Befehl löscht alle Merker von 1 bis 300, das heißt alle Merker von 1 bis 300 haben nachher den Wert 0.

3.7 Eingänge und Ausgänge

Eingänge und Ausgänge dienen zur Ein- und Ausgabe von binären Signalen.

3.7.1 Eingänge

Die Eingänge sind dem Programm direkt als binäres Signal, also als 0 oder 1 zugänglich. Die Eingänge können zum Beispiel an einem Schalter angeschlossen, welcher dann im Programm über den Eingangsbefehl abgefragt werden kann.

Die Numerierung der Eingänge kann dem entsprechenden Steuerungshandbuch entnommen werden.

Eingänge können abgefragt aber nicht gesetzt werden

Die Eingänge haben nur in Eingangsbedingungen eine Funktion. Sie können also nicht von der Software gesetzt oder sonst irgendwie beeinflusst werden. Nur Abfragen sind möglich.

Dies geschieht sehr einfach mit dem Eingangsbefehl:

E 101

Dieser Befehl fragt ab, ob der Eingang 101 gesetzt ist. Durch Eingabe eines Minuszeichens vor der Eingangsnummer hat man wie bei Merkern auf das invertierte Signal Zugriff. Die Nummer des Eingangs kann dabei auch indirekt angegeben werden.

Beispiel:

```
SOBALD
    -E 108
  DANN
```

Dieser Programmteil wartet, bis der Eingang 108 nicht mehr gesetzt ist. Sobald also der Eingang nicht mehr aktiv ist, wird mit dem Programm weitergemacht.

Mehrere in Registern zusammen-gefasste Eingänge

Über spezielle Register können mehrere Eingänge auf einmal eingelesen werden. Die Registernummern können dem entsprechenden Steuerungshandbuch entnommen werden. Außerdem werden sie exemplarisch in *Kapitel 4.1 Grundsätzliches über Register und Merker* dargestellt.

3.7.2 Ausgänge

Bei den Ausgängen handelt es sich um digitale Schaltausgänge zur Ansteuerung von Ventilen, Schützen, Leuchtmeldern oder Ähnlichem.

Die Numerierung der Ausgänge kann dem entsprechenden Steuerungshandbuch entnommen werden.

Der Befehl, um einen Ausgang zu schalten oder abzufragen ist:

A 101

Der Befehl hat folgende Auswirkung:

als **Ausgangsanweisung** (nach DANN oder SONST):
Der Ausgang 101 wird gesetzt (eingeschaltet)

als **Eingangsbedingung** (nach FALLS oder SOBALD):
Abfrage des Ausgangs: Ist der Ausgang gesetzt?
(Dabei handelt es sich um eine interne Logikabfrage. Das bedeutet, dass z.B. Kurzschlüsse des Leistungsausganges auf diese Art nicht erkannt werden können.)

Durch Eingabe eines Minuszeichens vor der Ausgangsnummer kann auch hier auf den inversen Wert zugegriffen werden, das heißt es kann abgefragt werden, ob ein Ausgang nicht gesetzt ist oder als Ausgangsanweisung wird der Ausgang abgeschaltet.

Auch hier kann die Nummer des Ausgangs indirekt spezifiziert werden.

Beispiele:

```
1)    FALLS
          E 101
        DANN
          A 201
        SONST
          -A 201
        DANN
          ...
```

Hier wird zuerst der Eingang 101 abgefragt. Ist dieser gesetzt, so soll auch der Ausgang 201 gesetzt werden. Ist der Eingang 101 nicht gesetzt, so soll der Ausgang 201 zurückgesetzt werden.

Anmerkung:

Das Setzen eines bereits aktiven Ausgangs, sowie das Zurücksetzen eines nicht aktiven Ausgangs haben keine Auswirkung.

```
2)    SOBALD
          -A 202
        DANN
```

Sobald der Ausgang 202 nicht mehr aktiv ist, soll mit dem Programm weitergemacht werden.

Auch bei den Ausgängen gibt es Register, über welche man gleichzeitigen Zugriff auf 8, 16 oder 24 Ausgänge hat. Die Numerierung kann dem entsprechenden Steuerungshandbuch entnommen werden.

3) Das Register 2540 (NANO-B) stimmt mit den Ausgängen A 101 bis A 108 überein.

Beispiele

Register-nummer NANO-B

1)

```
REG 2540
=
b00000000000000000011111111
```
(16 Nullen, 8 Einsen)

2)

```
LADE_REGISTER [2540 mit 255]
```

Mit diesen Befehlen werden alle Ausgänge (1 bis 8) gesetzt. (Es leuchten also anschließend alle 8 Leuchtdioden) Mit der Registerzuweisung kann direkt der binäre Wert eingegeben werden oder - wenn gewünscht - kann das wie oben rechts mit dem LADE_REGISTER Befehl direkt, also mit einem Befehl gelöst werden. Dazu muss zuerst der binäre Wert in den dezimalen Wert umgerechnet werden. (Siehe auch *Kapitel 3.11.4 Spezialfunktionen)*

3.8 Anzeige Befehle und Bedienereingabe

In diesem Kapitel werden die Befehle zu den Bediengeräten beschrieben. Diese Geräte sind in einem separaten Handbuch beschrieben.

Die Befehle sind folgende:

ANZEIGE_TEXT

ANZEIGE_REG

BEDIENEREINGABE

3.8.1 Anzeige von Texten

Der Befehl

ANZEIGE_TEXT[#<GeräteNr>,cp=<Cursorpos> "<Text>"]

dient zum Ausgeben eines Textes auf den Bediengeräten oder einem Drucker.

Bedeutung der Parameter

Gerätenummer

Für diesen Parameter kann 0 bis 9 eingegeben werden.

Bediengerät

#0 bis #4

es wird ein Bediengerät angesteuert

Drucker

#8

diese Gerätenummer veranlasst die Steuerung den Text auf einen Drucker auszugeben.

Frei programmierbare Schnittstelle

#9

Der Ausgabe erfolgt über die frei programmierbare PRIM-Schnittstelle.

Die separate Anzeige auf bis zu 4, gleichzeitig angeschlossene Bediengeräte ist möglich (Beschreibung kann angefordert werden).

Cursorposition

Dieser Parameter gibt die Cursorposition an, an der das erste Zeichen des Textes stehen soll. Dabei sind Werte von 0 bis 127 möglich. Die entsprechenden Cursorpositionen können dem Bediengerätehandbuch entnommen werden.

Beispiel zum LCD9:

Erste Zeile der Anzeige: Cursorpositionen von **1 bis 24**

Zweite Zeile der Anzeige: Cursorpositionen von **25 bis 48**

Die Cursorposition **0** hat eine spezielle Bedeutung: Wird die Cursorposition 0 gesetzt, so wird der Text an den zuletzt ausgegebenen Text angehängt. Der Cursor

steht genau da, wo er nach Beendigung des letzten Anzeigebefehls stehengeblieben ist.
(Mit einem Spezialregister kann die Funktion der Cursorposition 0 verändert werden. Siehe dazu die Beschreibung der Spezialregister im Steuerungshandbuch.)

Text

_ und $ sind Kontrollzeichen

Hier kann der Text angegeben werden, welcher angezeigt werden soll. Dabei dienen die beiden Zeichen "_" und "$" als Kontrollzeichen:

_ löscht die Anzeige

"_" Dieses Zeichen hat zur Folge, dass zuerst die Anzeige gelöscht wird und anschließend der angegebene Text, beginnend bei der Cursorposition 1 (unabhängig vom eingegebenen Parameter) angezeigt wird. Dieses Zeichen ist nur sinnvoll, wenn es am Anfang des Textes steht, weil sonst der vordere Textteil zuerst auf die Anzeige gebracht wird, jedoch gleich wieder gelöscht wird. Das Zeichen hat die Bedeutung **DELSCR** (Delete Screen/Anzeige löschen). Soll dieses Zeichen angezeigt werden, so kann in einem Spezialregister der Steuerung der Zeichencode für DELSCR geändert werden.

Beim Drucken hat dieses Zeichen die Bedeutung **FORM FEED** (Blatt - Vorschub).

$ löscht den Rest der Zeile ab Cursorposition

"**$**" Dieses Zeichen hat zur Folge, dass der Rest der Zeile, ab der momentanen Cursorposition gelöscht wird. Dieses Zeichen wird auch als **DELEOL** (Delete end of line/löschen bis Zeilenende) bezeichnet und kann ebenfalls durch ein anderes Zeichen ersetzt werden (Siehe Spezialregisterbeschreibung im Steuerungshandbuch).

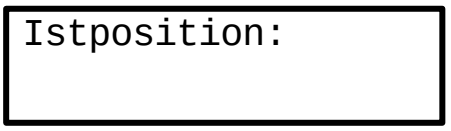
Beim Drucken hat dieses Zeichen die Bedeutung **LINE FEED** (Zeilen - Vorschub).

Beispiele:

1)

```
ANZEIGE_TEXT [#0, cp=0, "_Istposition:"]
```

Dieser Befehl löscht zuerst die gesamte LC - Anzeige und schreibt anschließend "Istposition:" auf die obere Zeile der Anzeige (Cursorposition = 1). Bei der Cursorposition könnte ebensogut irgend eine andere Zahl stehen, denn diese wird nach dem Anzeige-Lösch Zeichen (DELSCR) nicht mehr berücksichtigt. Die Anzeige sieht dann folgendermaßen aus:



2)

```
ANZEIGE_TEXT [#0, cp=25, "Sollposition:$"]
```

Dieser Befehl schreibt an der angegebenen Cursorposition 25, also das erste Zeichen der zweiten Anzeige-Zeile, den Text "Sollposition:" und löscht anschließend den Rest dieser Zeile.

3)

```
ANZEIGE_TEXT [#0, cp=0, " FEHLER"]
```

Dieser Befehl schreibt, beginnend bei der momentanen Cursorposition den Text " FEHLER" . Der Text wird also einfach an den zuletzt geschriebenen Text angehängt.

4)

```
ANZEIGE_TEXT [#8, cp=1, "Das geht auf den Drucker"]
```

Dies hat zur Folge, dass der Text "Das geht auf den Drucker", auf dem Drucker gedruckt wird. Es wird am Anfang der Zeile gedruckt. Einzelheiten zur Druckerausgabe sind im Steuerungshandbuch beschrieben. Bei Ausgabe auf PRIM wird die Cursorposition ignoriert.

3.8.2 Anzeige von Registerinhalten

Der Befehl

ANZEIGE_REG[#<GeräteNr>,cp=<Cursorpos>Reg=<RegNr>]

dient zum Ausgeben eines Registerwertes auf die Bediengeräte Module oder einem Drucker.

Die Parameter **Gerätenummer und Cursorposition** haben genau die gleiche Funktion wie bei dem ANZEIGE_TEXT - Befehl (siehe oben). Im Weiteren muss hier noch eine **Registernummer** angegeben werden. Es handelt sich dabei natürlich um die Nummer des Registers, dessen Wert man angezeigt haben will. Dabei ist auch die indirekte Adressierung möglich.

Für das Anzeigen eines Registers können nun noch zwei Parameter verändert werden, welche in den Spezialregistern "Feldbreite für Integerdarstellung" und "Ausrichtung, links/rechts" gespeichert werden. Die Nummern der beiden Spezialregister können dem jeweiligen Steuerungshandbuch entnommen werden.

Es sind hier die Standardeinstellungen (nach Reset) dargestellt:

Einstellung nach Reset

Spezialreg. "Feldbreite für Integerdarstellung" = 1

Es werden 8 Stellen für die Registeranzeige benutzt, Minuszeichen ganz vorne und Ziffern rechtsbündig in den restlichen 7 Plätzen.

Einstellung nach Reset

Spezialregister "Ausrichtung, links/rechts" = 0

Rechtsbündige Anzeige

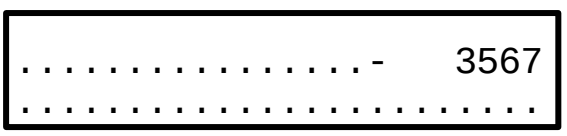
PROZESS-SPS (nicht NANO-A) ermöglichen auch die Darstellung von Gleitkommazahlen auf den Bediengeräten; Spezialregister "Feldbreite für Gleitkommadarstellung" (Wertebereich 1..14).

Beispiele:

1)

```
ANZEIGE_REG [#0, cp=17, Reg=100]
```

Mit dieser Anweisung wird das Register 100 auf die LC - Anzeige gebracht. Sind die Spezialregister "Feldbreite für Integerdarstellung" und "Ausrichtung, links/rechts" seit Reset nicht verändert worden, so wird das Register am Ende der ersten Anzeige-Zeile angezeigt und zwar wie nachfolgend dargestellt (Annahmen: Anzeige war vor dem Befehl leer und Register 100 = -3567).



Die Punkte bezeichnen diejenigen Stellen, welche nach den Befehlen noch den "alten" Inhalt haben.

2)

```
      ANZEIGE_TEXT   [#0,   cp=25,   "Istposition
:$"]
      ANZEIGE_REG [#0, cp=41, Reg=11109]
```

Hier sieht man wie man die beiden Anzeige - Befehle nützlich kombinieren kann. Es wird dabei zuerst in die zweite Zeile (links) der Text "Istposition :" geschrieben und der Rest der zweiten Zeile gelöscht (Dollarzeichen "$"). Mit dem zweiten Befehl wird rechts unten auf der Anzeige das Register 11109 angezeigt. Dieses Register speichert die Istposition der Schrittmotorachse auf dem Grundmodul NANO-B (Annahmen: Die Istposition der Achse 1 habe den Wert 5400.

```
.........................
Istposition:         5400
```

Die Punkte bezeichnen diejenigen Stellen, welche nach den Befehlen noch den "alten" Inhalt haben.

3) Das folgende Beispielprogramm soll zeigen, wie es möglich ist, ein Protokoll von Werten direkt auf einem Drucker zu führen.

```
      TASK  5 --------------------------
            LADE_REGISTER [ rFeldbreite mit 2]
            ANZEIGE_TEXT [#3, cp=1 "$"]
            ANZEIGE_TEXT                [#3,cp=1
"Sollposition"]
            ANZEIGE_TEXT              [#3,cp=21
"Istposition"]
            ANZEIGE_TEXT       [#3,       cp=41
"Drehzahl$"]
      MARKE  100
        SOBALD
            MERKER 1
```

```
          DANN
            ANZEIGE_REG        [#3,        cp=3,
Reg=rSollPos]
            ANZEIGE_REG       [#3,       cp=23,
Reg=rIstPos]
            REG 1
            =
            30
            *
            REG rIstPos
            /
            REG AnzahlGeberLinien
            ANZEIGE_REG [#3, cp=43, Reg=1]
            -MERKER 1
            SPRUNG 100
```

Dieses Programm wurde in einem Parallelzweig (5) programmiert. Dies ist auch sehr sinnvoll, denn auf diese Art kann von einem beliebigen anderen Parallelzweig einfach durch Setzen des Merkers 1 das Ausdrucken einer Zeile des Protokolls ausgelöst werden. Zu Beginn dieses Tasks (Parallelzweigs) wird die Titelzeile gedruckt. Dies geschieht vor der eigentlichen Schleife, so dass in diesem Fall drei Spalten gedruckt werden. In der ersten steht die Sollposition, in der zweiten die Istposition und in der dritten Spalte steht die momentane Drehzahl in U/min (falls 1000 Inkremente je Umdrehung ausgewertet werden).

NANO-A kein Drucker-anschluss

Der Protokollausdruck könnte wie folgt aussehen:

```
   Sollposition         Istposition           Drehzahl
          15000                8433               2450
      -   4800                1206           -   1207
         250000              250000                  0
```

3.8.3 Abfrage von Registerwerten durch das Programm

Der Befehl

BEDIENEREINGABE[#<GerNr>,cp=<Cursorpos>,Reg=<RegNr>]

dient zum Einlesen von Registerwerten, welche über die Tastatur des Anzeige- und Tastaturmoduls eingegeben werden können.

Für die beiden Parameter **Gerätenummer und Cursorposition** gilt dasselbe wie bei dem ANZEIGE_TEXT -Befehl, mit folgenden Änderungen. Bei der Gerätenummer kann natürlich nicht ein Drucker angewählt werden, also ist die Gerätenummer 8 in diesem Fall zu vermeiden. Wird die Cursorposition 0 eingegeben, so wird bei der Bedienereingabe der Wert aus Spezialregister "Absolute Cursorposition für BEDIENEREINGABE" als Cursorposition gewählt. Ist dieser jedoch auch 0 (und das ist der Wert, den das Register nach Reset hat), so wird an der momentanen Cursorposition eingelesen.

Indirekte Adressierung des Zielregisters möglich

Bei der **Registernummer** handelt es sich um die Nummer desjenigen Registers, dem der eingegebene Wert zugewiesen werden soll. Auch hier ist eine einfach indirekte Registerangabe möglich.

Wichtig bei der BEDIENEREINGABE ist noch zu wissen, dass im Normalfall für die Eingabe 8 Zeichen zur Verfügung stehen. Dieser Wert, welcher im Spezialregister "Feldbreite BEDIENEREINGABE" gespeichert ist, kann auch geändert werden. (Siehe *Kapitel 3.8.4 Spezialregister für die Bedienereingabe*)

Beispiel:

Kombination von ANZEIGE_TEXT und BEDIENER-EINGABE

Um eine sinnvolle Bedienerführung zu realisieren, wird meist die BEDIENEREINGABE mit dem ANZEIGE_TEXT-Befehl kombiniert.

```
      ANZEIGE_TEXT [#0, cp=1, "_Neue Position
?"]
      BEDIENEREINGABE [#0, cp=17, Reg=100]
```

Diese beiden Befehle bewirken, dass auf der Anzeige oben links der Text "Neue Position ?" erscheint und anschließend auf die Eingabe einer Zahl gewartet wird. Diese Zahl, welche im Register 100 gespeichert wird, dient als neue Sollposition für eine Positionierung.

3.8.4 Spezialregister für die Bedienereingabe

Hinweis:

Die Spezialregisternummern sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein allgemeiner Überblick über die Spezialregister gegeben.

Spezialreg "Feldbreite Fließkommadarstellung"

Feldbreite des ANZEIGE_REGISTER-Befehles bei Gleitkommazahlen. Wertebereich 1..14, Wert nach Reset: 8.

Spezialregister "Feldbreite für Integerdarstellung"

Dieses Register enthält die Anzahl Zeichen, welche bei einem ANZEIGE_REG - Befehl angezeigt werden sollen. Nach Reset wird der Wert auf 1 gesetzt, was einer Anzeige von 7 Stellen entspricht.

Spezialregister "Linksbündige Zahlendarstellung"

Dieses Register enthält die Information, ob ein angezeigtes Register auf dem Bediengerät linksbündig oder rechtsbündig geschrieben werden soll. Der Wert nach Reset beträgt hier 0.

Die folgenden Tabellen zeigen die verschiedenen Kombinationsmöglichkeiten der beiden Register und

deren Auswirkung auf das Anzeigeformat. Die Cursorposition entspricht immer dem vordersten Zeichen. Der "*" (Stern) steht im Folgenden für die Ausgabe eines Leerzeichens. Das "+" steht als Platzhalter für ein positives Vorzeichen, es wird jedoch in Wirklichkeit eine Leerstelle angezeigt.

a) Spezialregister **"Linksbündige Zahlendarstellung"** = 0; Default nach Reset

RegWert	1234	-345	7654321	-1234567
0	***+1234	****-345	+7654321	-1234567
1	+***1234	-****345	+7654321	-1234567
2	+**1234	-***345	+654321	-234567
3	+*1234	-**345	+54321	-34567
4	+1234	-*345	+4321	-4567
5	+234	-345	+321	-567
6	+34	-45	+21	-67
7	+4	-5	+1	-7
8	+	-	+	-

b) Spezialregister **"Linksbündige Zahlendarstellung"** = 1

RegWert	1234	-345	7654321	-1234567
0	+1234	-345	+7654321	-1234567
1	+	-	+	-
2	+1	-3	+7	-1
3	+12	-34	+76	-12
4	+123	-345	+765	-123
5	+1234	-345*	+7654	-1234
6	+1234*	-345**	+76543	-12345
7	+1234**	-345***	+765432	-123456
8	+1234***	-345****	+7654321	-1234567

c) Spezialregister **"Linksbündige Zahlendarstellung"** = 2 entspricht Punkt a) jedoch wird keine Vorzeichenstelle angezeigt.

Spezialregister "Feldbreite BEDIENEREINGABE

Der Wert dieses Registers gibt die Feldbreite an, die dem Bediener an der LC-Anzeige zur Verfügung, wenn der Befehl BEDIENEREINGABE ausgeführt wird. Der Wert dieses Register nach Reset beträgt 8, das heißt dem Bediener stehen 8 Zeichen zur Eingabe zur Verfügung. Dabei ist eines dieser Zeichen (das Vorderste) für das Vorzeichen reserviert und wird nur durch ein Vorzeichen besetzt.

3.8.4.1 Steuerzeichen für den Text - Anzeige Befehl

Default nach Reset: $

Spezialregister "Lösche bis Zeilenende Zeichen"

Dieses Register enthält den ASCII-Code des **DELEOL** (**DEL**ete **E**nd **O**f **L**ine) Zeichens. Nach Reset hat es den Wert 36, was der ASCII-Code von "$" (Dollarzeichen) ist. Will man zum Beispiel in der Anzeige eines Textes ein Dollarzeichen verwenden, so kann mit diesem Register durch das Eingeben einer anderen Zahl die Funktion von "$" auf ein anderes Zeichen übertragen werden.

Default nach Reset: _

Spezialregister "Lösche Anzeige Zeichen"

Dieses Register enthält den ASCII-Code des **DELSCR** (**DEL**ete **SCR**een) Zeichens. Dieses Zeichen veranlasst das Löschen der LC - Anzeige. Nach Reset hat das Register den Wert 95, was der ASCII-Code von "_" (Unterstreichen) ist. Auch dieses Zeichen kann durch ein beliebiges Anderes ersetzt werden.

Register-nummerierung DELTA

Beispiel:

```
LADE_REGISTER [ 61462 mit 38]
ANZEIGE_TEXT [ #0, cp=0, "&"]
```

Diese beiden Befehle löschen die Anzeige. Zuerst wird das DELSCR-Zeichen geändert und nachher im ANZEIGE_TEXT - Befehl gleich angewendet.

3.8.4.2 Steuerregister für die Cursorposition auf der LC-Anzeige

Spezialregister "Absolute Cursor Position AT, AR"

Der Wert dieses Registers gibt die Cursorposition beim ANZEIGE_TEXT oder ANZEIGE_REG - Befehl an, falls im Befehl die Cursorposition 0 angegeben wurde. Ist der Wert dieses Register auch 0, was nach Reset der Fall ist, so wird der anzuzeigende Text einfach an die letzte Anzeige angehängt. Die momentane Cursorposition wird also nicht verändert.

Spezialregister "Absolute Cursor Position BE"

Dieses Register hat genau die gleiche Funktion wie das Obige, aber für den BEDIENEREINGABE-Befehl. Es bestimmt die Cursorposition, an der die Eingabe stattfindet, wenn bei einer Bedienereingabe die Cursorposition 0 eingegeben wurde. Auch dieser Wert ist nach Reset 0, was ein Anhängen an die letzte Ein- oder Ausgabe zur Folge hat.

Registernummerierung DELTA

Beispiel:

```
LADE_REGISTER [ 61649 mit 25]
BEDIENEREINGABE [ #0, cp=0, Reg=100]
```

Das Register für die indirekte Angabe der Cursorposition bei der Bedienereingabe wird zunächst mit dem Wert 25 geladen. Der nachfolgende Befehl erlaubt, am Anfang der zweiten Zeile der Anzeige eine Zahl einzugeben, die dann Register 100 zugewiesen wird.

3.8.4.3 LCD-Anzeigezeit

Die Anzeige der Bediengeräte arbeitet mit zwei Anzeigeebenen. Normalerweise wird Ebene eins angezeigt, auf die auch vom Anwenderprogramm zugegriffen wird; das heißt, hier werden alle ANZEIGE_TEXT-, ANZEIGE_REG- und BEDIENEREINGABE-Befehle dargestellt.
Durch Betätigen der Tasten 'R' und 'F' können über das Anzeigemodul Register, Merker und Ein- und Ausgänge abgefragt und geändert werden. Dies erfolgt in der Anzeigeebene zwei (Monitorscreen genannt).
Nach welcher Zeit wieder auf die Ebene eins zurückgeschaltet wird, kann festgelegt werden durch das

Spezialreg "Anzeigezeit für Monitorfunktionen"

Dieses Register gibt die Zeit in Sekunden an, wie lange ein über die Tastatur abgefragtes Register, Merker, Eingang oder Ausgang auf der Anzeige bleibt, das heißt, wie lange der Monitorscreen angezeigt wird, bevor auf die Anzeige des Anwenderprogramms zurückgeschaltet wird.
Nach Reset hat dieses Register den Wert 3, also drei Sekunden.

3.8.4.4 Eingabeerlaubnis für Merker- und Registeränderungen über die Bediengeräte-Tastatur

Eine Reihe von Spezialregistern dienen zur Definition von Register-Bereichen, welche dann über die Tastatur geändert werden dürfen. Dabei bilden immer zwei aufeinanderfolgende Register ein Paar, welches einen Block definiert. Die ersten drei Paare beziehen sich auf Register, das vierte Paar auf Merker.

Spezialregister "Erstes änderbares Register - Bereich 1"
"Letztes änderbares Register - Bereich 1"

Das Spezialregister "Erstes änderbares Register - Bereich 1" (Wert nach Reset = 0) gibt das unterste Register an, welches geändert werden darf und Spezialregister "Letztes änderbares Register - Bereich 1" (Wert nach Reset = 59999) das Oberste. (erster Register - Eingabeerlaubnis - Bereich)

Spezialregister "Erstes änderbares Register- Bereich 2"
"Letztes änderbares Register - Bereich 2"

Ebenso zweiter Register - Bereich (Werte nach Reset: beide Register 0)

Spezialregister "Erstes änderbares Register - Bereich 3"
"Letztes änderbares Register - Bereich 3"

Ebenso dritter Register - Bereich (Werte nach Reset: beide Register 0)

Spezialregister "Erster änderbarer Merker" "Letzter änderbarer Merker"

Das Spezialregister "Erster änderbarer Merker" (Wert nach Reset = 0) gibt den untersten Merker an, welcher geändert werden kann, Spezialregister "Letzter änderbarer Merker" (Wert nach Reset = 59999) den obersten.

Hinweis:

Die Werte nach Reset sind so gesetzt, dass man alle Register und alle Merker über die Tastatur ändern kann. In den meisten Fällen empfiehlt es sich aber, wichtige Register- und Merkerblöcke, welche wichtige Werte enthalten, auf diese Art zu schützen. So kann verhindert werden, dass durch einen Tippfehler an der Tastatur wichtige Werte verloren gehen.
Die Eingabeerlaubnis für Merker, Eingänge, Ausgänge und Register kann auch generell verboten werden. (Siehe dazu die Spezialmerker im Steuerungshandbuch.)

Register-nummerierung DELTA

Beispiel:

```
LADE_REGISTER [ 61697 mit 49]
LADE_REGISTER [ 61698 mit 100]
LADE_REGISTER [ 61699 mit 199]
LADE_REGISTER [ 61703 mit 300]
```

Unter der Annahme, dass die übrigen Register nach Reset nicht geändert wurden, ergeben diese Zuweisungen folgenden Zustand:

Register 0 bis 49: Eingabe erlaubt
Register 100 bis 199: Eingabe erlaubt
Merker 1 bis 300: Eingabe erlaubt

Alle anderen Register und Merker sind über die Tastatur nicht mehr veränderbar. Somit sind alle Spezialregister, auch die Achsmodulregister geschützt.

3.8.4.5 Einschränkung der Monitorfunktionen

Spezialregister "Einschränkung der Monitorfunktionen"

Mit diesem Register sind die Merker 2096 (Bit0) bis 2103 (Bit7) überlagert.

0 = Funktion gesperrt, 1 = Funktion verfügbar

Bit0 = 0	R, I/O Tasten ohne Monitorfunktion (aber Merker wird gesetzt)
Bit0 = 1	R, I/O Taste mit Monitorfunktion
Bit1 = 0	R, I/O Taste ohne Funktion Merkereingabe
Bit1 = 1	R, I/O Taste mit Funktion Merkereingabe
Bit2 = 0	R, I/O Taste ohne Funktion Ausgangsnummer-eingabe
Bit2 = 1	R, I/O Taste mit Funktion Ausgangsnummer-eingabe
Bit3 = 0	R, I/O Taste ohne Funktion Eingangsnummer-eingabe
Bit3 = 1	R, I/O Taste mit Funktion Eingangsnummer-eingabe
Bit4 = 0	= Taste kann keine Registerinhalte verändern
Bit4 = 1	= Taste kann Registerinhalte verändern
Bit5 = 0	= Taste kann keine Merker verändern
Bit5 = 1	= Taste kann Merker verändern
Bit6 = 0	= Taste kann keine Ausgänge verändern
Bit6 = 1	= Taste kann Ausgänge verändern
Bit7 = 0	= Taste greift nicht auf Eingänge zu
Bit7 = 1	= Taste greift auf Eingänge zu

3.8.4.6 Zeit für die Bedienereingabe

Einstellung nach Reset: keine zeitliche Limitierung

Spezialregister "Maximale Zeit für BEDIENEREINGABE"

Dieses Register gibt dem Betriebssystem an, wieviel **Zeit** (in Sekunden) der **Bediener für eine BEDIENEREINGABE** zur Verfügung hat.

Nach Reset hat dieses Register den Wert 0, was folgende Bedeutung hat: es wird ohne zeitliche Bedingung gewartet, bis der Bediener seine Eingabe gemacht hat und diese mit ENTER abgeschlossen hat. Die Zahl wird dann eingelesen und mit Verarbeitung des Programms weitergemacht. Durch Setzen des Registers zum Beispiel auf 10 wird erreicht, dass der Befehl BEDIENEREINGABE nach 10 Sekunden abgebrochen wird.

Merker 2053

Dieser Merker ist im Zusammenhang mit dem Spezialregister "Maximale Zeit für BEDIENEREINGABE" zu sehen. Nach einer BEDIENEREINGABE kann über diesen Merker abgefragt werden, ob die Eingabe vom Bediener ordnungsgemäß beendet wurde oder ob die Bedienereingabe durch ein 'Timeout' abgebrochen wurde.

Merker 2053 = 1 bedeutet einen Abbruch durch Timeout.

3.9 Steuerungsbefehle für Achsen

In diesem Kapitel werden alle zur Verfügung stehenden Befehle im Umgang mit Achsen erklärt. Es handelt sich dabei um die folgenden Befehle:

POS

ISTPOS

HALTACHSE

Durch die Verwendung dieser Makrobefehlsaufrufe für die Steuerung einer Achse wird die Programmentwicklung wesentlich erleichtert; das Programm wird übersichtlicher und leichter lesbar.

3.9.1 Die Positionierung

Für das Positionieren einer Achse, also das Fahren einer Achse an einen bestimmten Ort, steht der Befehl :

```
POS [Achse<AchsNr>, Pos<Sollpos>, v<Sollgeschw>]
```

Es müssen drei Parameter eingegeben werden, welche dazu dienen, der Achskarte mitzuteilen, wohin gefahren werden soll und mit welcher Endgeschwindigkeit. Alles weitere wird dann von der Achskarte selbst erledigt. Damit die richtige Achse angesteuert wird, muss zuerst die Achsnummer eingegeben werden.

Hinweis:

- **Zwischen die wiederholte Ausführung von POS-Befehlen ist eine Wartezeit von 100 ms einzufügen.**
- **Zwischen dem wiederholten Parametrieren der Sollgeschwindigkeit ist eine Wartezeit von 100 ms einzufügen.**

Die Bedeutung der Parameter (alle Parameter können indirekt adressiert werden):

Alle Parameter können indirekt adressiert werden

Achsennummer

Die Achsnumerierung ist dem entsprechenden Steuerungshandbuch zu entnehmen.

Sollposition

Alternativ zum POS-Befehl können die Positions- und Geschwindigkeitsregister direkt mit LADE_REG beschrieben werden

Hier kann eine beliebige Zahl eines Ganzzahlregisters eingegeben werden (also von -8388608 bis 8388607). Ob all diese Werte sinnvoll sind, hängt von der ganzen Anlage ab. Im Normalfall ist der wirklich sinnvolle Bereich für eine Anwendung kleiner. (Siehe dazu auch Beschreibungen der entsprechenden Servoregler.)

Auf jeden Fall gibt diese Zahl die Sollposition an, das heißt also jene Position, zu der die Achse fahren soll. Der Positionseintrag im POS-Befehl entspricht einem direkten Beschreiben des Sollpositionsregisters.

Sollgeschwindigkeit

Die eingegebene Zahl gibt die maximale Geschwindigkeit für diese Positionierung an.

Sowohl bei der Sollposition wie auch bei der Sollgeschwindigkeit kann eine doppelt indirekte Registerangabe stehen. Also zum Beispiel RR50. Für Erklärungen zu diesem doppelt indirekten Registerzugriff siehe LADE_REGISTER Befehl, *Kapitel 3.6.2 Befehle, um Register zu laden*.

Damit kann die Bedeutung des Befehls POS in einfache Worte gefasst werden:

" **Achse x** : fahre mit der **Geschwindigkeit v** zur **Position Pos** "

Bei Servo: sinusförmige Start- und Stoprampen

Beim Anfahren wird die Geschwindigkeit allmählich (sinusquadratförmig) erhöht und beim Abbremsen vor der Zielposition wird sie ebenfalls allmählich (auch sinusquadratförmig) verringert.
Wie schnell dieses "allmählich", das heißt wie groß die Steilheit der Sinusquadratfunktion sein soll, kann über weitere Parameter (Start- und Stoprampen), welche direkt in die dafür vorgesehenen Register geladen werden, eingestellt werden.

Register-nummern beispielhaft an NANO-B

Im Register 1xy05 steht die **Startrampe**.

Im Register 1xy06 steht die **Stoprampe**.

Anmerkung:

Außer der Start- und Stoprampe müssen in den meisten Fällen noch weitere Parameterregister auf den Achskarten initialisiert werden. Dazu werden bevorzugt die folgenden Befehle verwendet:

LADE_REGISTER
SPEZIALFUNKTION [#1, . . .]

KOPIERE

Diese Befehle sind ausführlich im *Kapitel 3.6.2 Befehle, um Register zu laden beschrieben*.

Zur Positionierung ist auch der Befehl

HALTACHSE Achse<AchsNr>

notwendig. Dieser Befehl kann als Eingangsbedingung und als Ausgangsanweisung verwendet werden. Die Achsnummer kann auch einfach indirekt angegeben werden.

HALTACHSE verzögert mit der Startrampe

Als **Ausgangsanweisung** (nach DANN oder SONST) bedeutet er:

> Die Achse soll unverzüglich (d.h. ohne Rampe) anhalten und die Position soll an der momentanen Istposition geregelt werden.

Als **Eingangsbedingung** (nach FALLS oder SOBALD) bedeutet er:

> Hat die Achse die Zielposition (respektive das Zielfenster) erreicht? Wenn ja, gibt der Befehl eine 1 als boole'schen Wert (wahr), wenn nein, eine 0 (falsch) zurück.

Wird ein Minuszeichen vor der Achsnummer eingegeben, so kehrt dieser Befehl seine Bedeutung um.

> In einer **Ausgangsanweisung** heißt das, dass eine angehaltene Achse wieder in Bewegung gesetzt wird, falls ihre Sollposition noch nicht erreicht ist.

In einer **Eingangsbedingung** kann abgefragt werden, ob eine Achse noch läuft.

Anmerkung:

Beim Positionierbefehl muss noch folgendes berücksichtigt werden. Sobald der Befehl eingelesen und an die Achskarte weitergegeben ist, hat der Prozessor auf der CPU seine Arbeit erledigt und geht deshalb zum nächsten Befehl über. Die Positionierung der Achse selbst übernimmt der Mikroprozessor auf dem Achsreglermodul selbständig.
Aus diesem Grund wird im Normalfall mit Hilfe des HALTACHSE-Befehls abgefragt, ob die Achse bereits im Zielfenster angekommen ist bevor die nächste Positionierung gestartet wird. Es ist jedoch auch ausdrücklich zulässig, während einer laufenden Positionierung eine weitere zu starten. In diesem Fall wird die Achse auf die zuletzt übergebene Sollposition fahren ohne zwischendurch anzuhalten.
Das Zielfenster ist ein wählbarer Bereich um die Zielposition. Er kann mit dem Register 7 eingestellt werden.

Initialisieren der Achsregler

Nachdem die Versorgungsspannung der Steuerung eingeschaltet wurde, werden alle Register der Achsregler mit ihren Anfangswerten besetzt (Resetwerte). Bei verschiedenen SV-Modulen wird bei diesem Reset auch ein Relais, welches den analogen Geschwindigkeits - Sollwertausgang nach außen durchschaltet, abgeschaltet. Durch Laden der Zahl 1 ins Kommandoregister (1) des gewünschten Modules kann dieses Relais eingeschaltet werden:

Referenzieren der Achse

Das Positionieren ist erst möglich, wenn eine Referenzposition geladen ist. Zum Einstieg kann diese Referenzposition auch einfach gesetzt werden. Dies ist durch Beschreiben des Kommandoregisters mit der Zahl 3 möglich:

```
LADE_REGISTER [ rKommando mit 3]
```

Im Normalfall wird ein spezieller, genau definierter Referenzpunkt durch eine Referenzfahrt gesucht und dann geladen. Beim obigen Befehl wird einfach die momentane Position als Referenzpunkt geladen.

Beispiele:

1)
```
LADE_REGISTER [ rKommando mit 3]
LADE_REGISTER [rKOmmando mit 1]
POS [Achse=21, Pos=10000, v=500]
```

Zuerst wird das Relais eingeschaltet, dann die Referenz gesetzt und schließlich wird positioniert:

"Achse 21 auf dem ersten Achsmodul : Fahre mit der Geschwindigkeit 500 ‰ zur Position 10000 !"

Die Achse 21 soll mit der Geschwindigkeit 500 zur Position 10000 fahren. Es werden also lediglich die Sollposition (10000) und die Sollgeschwindigkeit (500) in die entsprechenden Register auf der Achskarte geladen.
Die Achskarte stellt die Sollgeschwindigkeit an ihrem Ausgang folgendermaßen ein:
Mit einer sinusquadratförmigen Rampe (Schrittmotor linear), steigt die Geschwindigkeit auf ihren Endwert von 500 an. Sobald erkannt wird, dass die Achse in der Nähe der Sollposition ist, wird die Geschwindigkeit wieder verringert, bis die Sollposition erreicht ist. Dort ist dann auch die berechnete Geschwindigkeit 0.

Detaillierte Informationen zu den Servoregler dem entsprechenden Steuerungshandbuch entnehmen

(Für eine genauere Beschreibung, auch von Start- und Stoprampe, kann im Servoregler- oder Schrittmotorkapitel des entsprechenden Steuerungshandbuches nachge-schlagen werden.)

```
2)        LADE_REGISTER [rStartRampe mit 50]
          LADE_REGISTER [rStopRampe mit 10]
          LADE_REGISTER [ rKommando mit 3]
          LADE_REGISTER [rKommando mit 1]
          POS [Achse21, Pos-40000, v1000]
      SOBALD
          HALTACHSE Achse21
        DANN
          POS [Achse21, Pos -30000, v200]
```

Zuerst werden die Rampen definiert (Start/Stop) dann die Referenz gesetzt und zuletzt das Relais eingeschaltet.
Zunächst wird mit einer Geschwindigkeit von 1000‰ zur Position -40000 gefahren. Nachdem die Position erreicht ist, wird der Befehl erteilt, mit der Geschwindigkeit 200‰ zur Position -30000 zu fahren.

```
3)         LADE_REGISTER [rKommando mit 3]
           LADE_REGISTER [rKommando mit 1]
           POS [Achse=21, Pos100000, v=300]
           WARTEZEIT  20
           HALTACHSE 21
```

Auch hier wird zuerst die Referenz gesetzt und dann das Relais eingeschaltet. Anschließend wird die Achse 21 (Achse auf erstem Achsmodul am Steckplatz 2) angewiesen, mit der Geschwindigkeit 300‰ zur Position 100000 zu fahren. Nach einer Wartezeit von 2 Sekunden (Angabe in Vielfachen von 100 ms), wird diese Fahrt mit dem HALTACHSE-Befehl abgebrochen.

3.9.2 Abfrage der momentanen Position

Um die momentane Position abzufragen braucht man den Befehl

ISTPOS

Dieser Befehl erlaubt einen eleganten Zugriff auf die momentane Position einer Achse.
Diese Position ist in einem Register des Achsmodules (Register 9) und könnte ebensogut durch Abfragen dieses Registers ermittelt werden. Die Angabe der Achsnummer kann auch indirekt erfolgen.

Beispiel:

```
          LADE_REGISTER [rKommando mit 3]
          LADE_REGISTER [rKommando mit 1]
          POS [Achse=21, Pos10000, v=500]
      SOBALD
          ISTPOS Achse21
          >
          8000
        DANN
          A 201
```

Zuerst wird die Referenz gesetzt, dann das Relais eingeschaltet und schließlich der Befehl gegeben, mit der Geschwindigkeit 500‰ zur Position 10000 zu fahren. Sobald die Istposition größer ist als 8000, wird der Ausgang A 201 gesetzt.

3.10 Task Befehle

Die in diesem Kapitel beschriebenen Befehle dienen zur gegenseitigen Steuerung von Tasks.

So kann man einen Task:

TASKBREAK	unterbrechen
TASKCONTINUE	weiterlaufen lassen
TASKRESTART	neu starten

3.10.1 Taskbreak

Die Anweisung

TASKBREAK #<TaskNr>

unterbricht die Bearbeitung des angegebenen Parallelzweigs (Tasks).

Der Parameter, welcher bei diesem Befehl angegeben werden muss, ist die Nummer desjenigen Parallelzweiges, welcher unterbrochen werden soll. Also eine Zahl von 0 bis 31.

Hinweis:

Zu beachten ist dabei, dass intelligente Slavemodule ihre Regelung/Positionierung **nicht** abbrechen! Sollte das gewünscht sein, muss die Positionierung explizit abgebrochen/ die Regelung unterbrochen werden.

Mit diesem Befehl kann zum Beispiel ein Automatikablauf an einer beliebigen Stelle unterbrochen werden. Anschließend kann ein Hand- oder NOTAUS- Programm bearbeitet werden.

3.10.2 Taskcontinue

Die Anweisung

TASKCONTINUE #<TaskNr>

veranlasst, dass ein unterbrochener Parallelzweig seine Bearbeitung fortsetzt.

3.10.3 Taskrestart

Die Anweisung

TASKRESTART #<TaskNr>

startet die Bearbeitung des angegebenen Parallelzweigs neu, also am Anfang des Tasks.

3.10.4 Beispiele zu den Taskbefehlen

```
      TASK  0 ---------------------------------
--
            ...
            ...         ; z. B. Handprogramm
            ...
      TASK  1 ---------------------------------
--
            ...
            ...         ;z.                   B.
Automatikprogramm
            ...
      TASK  2 ---------------------------------
---            ...
            ...         ; Referenzfahrtprogramm
            ...
      TASK  3 ---------------------------------
--
            ...
            ...         ;z.      B.      weitere
Programme
            ...
      TASK  4 ---------------------------------
--
        SOBALD
            E -101      ;Notausschalter
gedrückt
          DANN
            TASKBREAK #0
            TASKBREAK #1
            TASKBREAK #2
            TASKBREAK #3
            HALTACHSE Achse21
            HALTACHSE Achse31
            HALTACHSE Achse41
        SOBALD
            E 101       ; Notausschalter gelöst
          DANN
            HALTACHSE Achse-21
            HALTACHSE Achse-31
            HALTACHSE Achse-41
            TASKCONTINUE #0
            TASKCONTINUE #1
            TASKCONTINUE #2
            TASKCONTINUE #3
            SPRUNG 4
```

Für weitere Beispiele und generelle Informationen über Multitasking siehe *Kapitel 2.1 Prinzipieller Programmaufbau*

3.11 Verschiedene Befehle

In diesem Kapitel werden die Befehle

STARTE-ZEIT

ZEIT-ENDE

LEER

;

SPEZIALFUNKTION

GRENZEN

Wortverarbeitung WUND, WODER, WXODER

beschrieben.

3.11.1 Zeitbefehle

3.11.1.1 Die Befehle STARTE-ZEIT und ZEIT-ENDE?

Die Befehle haben die folgende Syntax:

STARTE-ZEIT [Register Nr., Wert (Zeit)]

ZEIT-ENDE? [Register Nr.]

Diese beiden Befehle werden hier gemeinsam beschrieben, da sie der gleichen Funktion zugehören, das heißt voneinander abhängig sind.

Der STARTE-ZEIT respektive ZEIT-ENDE? Befehl kann indirekt parametriert werden

Der Parameter des STARTE-ZEIT Befehls kann als Zahl oder indirekt, als Registernummer angegeben werden.

Mit Hilfe des STARTE-ZEIT und des ZEIT-ENDE? Befehls können zeitliche Überwachungen vorgenommen werden. Der Befehl STARTE-ZEIT enthält die gewünschte Zeit, das Register in dem dieser Wert abgelegt werden soll und er startet im laufenden Programm die Überwachungzeit. Der Befehl ZEIT-ENDE? dient zur Abfrage, das heißt er stellt fest, ob die im STARTE-ZEIT Befehl angegebene Zeit abgelaufen ist. Im Unterschied zu dem Befehl WARTEZEIT läuft das Programm während des bestimmten Zeit-Intervalls weiter, auch im gleichen Task. Diese Funktion kann z.B. benutzt werden um Vorgänge zeitlich zu begrenzen, wie etwa die Erwärmung eines Gegenstandes. Der Inhalt des angegebenen Registers steht in keinem direkten Zusammenhang mit der angegebenen Zeit. Es ist also nicht einfach möglich zu kontrollieren wieviel Zeit bereits verstrichen ist, bzw. welche Restzeit noch vorhanden ist!
Als Überwachungszeitregister können alle Anwender-register genutzt werden.

Zwischen dem STARTE-ZEIT-Befehl und dem zugehörenden ZEIT-ENDE?-Befehl darf auf das gewählte Register keine Zuweisung erfolgen, sonst liefert der ZEIT-ENDE?-Befehl kein sinnvolles Ergebnis zurück!

Interne Bearbeitung von STARTE-ZEIT, ZEIT_ENDE?

Beim STARTE-ZEIT-Befehl wird zum Inhalt eines wählbaren Zeitbasis-Registers die Befehl angegebene Zeit hinzuaddiert und die Summe in das im Befehl angegebene Register gespeichert. Die Addition wird als vorzeichenlose 22 Bit Operation durchgeführt. Dies

bedeutet, dass die maximale Überwachungszeit 4 Millionen Zeitinkremente betragen kann.

Der ZEIT-ENDE?-Befehl vergleicht den abgespeicherten Wert mit dem aktuellen Inhalt des Zeitbasis-Registers. Ist das Zeitbasis-Register noch kleiner als der gespeicherte Wert, so hat der ZEIT-ENDE?-Befehl das Ergebnis "falsch" (0). Ist das Zeitbasis-Register gleich oder größer als der gespeicherte Wert, so wird das Bit 23 des angegebenen Registers gesetzt (negativ) und das Ergebnis ist "wahr" (1; Zeit abgelaufen).

Daraus ergibt sich, dass nach dem STARTE-ZEIT-Befehl innerhalb von 4 Millionen Zeitinkrementen mindestens ein Mal der ZEIT-ENDE?-Befehl das Ergebnis "wahr" liefern muss, damit das Register negativ gesetzt werden kann, bevor ein Zahlenüberlauf stattfindet. Andererseits kann durch Setzen des benutzten Registers auf einen negativen Wert der Zustand "Zeit abgelaufen" erzwungen werden.

Die Nummer des Zeitbasis-Registers kann in einem Register angegeben werden. Nach dem Einschalten wird das "Laufzeitregister" (in Userinkrementen) verwendet. Es kann jedoch jedes Anwenderregister verwendet werden.

Beispiel:

```
    POS [Achse=21, Pos=..., v=...]
    STARTE-ZEIT [Reg=rÜberwachung, Zeit=100]
...
    WARTZEIT 20
...
...
SOBALD
    ZEIT-ENDE? Reg=rÜberwachung
    ODER
    HALTACHSE 21
  DANN
...
```

3.11.1.2 Spezialregister zu den Zeitbefehlen

Spezialregister "Anwender-Zeitbasis in ms"

Dieses Register legt die **Zeitinkremente** der Steuerung fest und zwar in Einheiten von ms (Millisekunden). Nach Reset ist dieser Wert auf 100, also das Zeitinkrement auf 100 ms festgelegt. Bei diesem Register können Werte von 1 bis 255 eingegeben werden, wobei Werte kleiner 10 nicht eingegeben werden sollten.

Beim WARTEZEIT Befehl wird das Zeitregister des Tasks mit der entsprechenden Zahl geladen und dann gewartet, bis dieses Zeitregister Null ist. Folgende zwei Programm-sequenzen sollen dies veranschaulichen (Beispiel NANO-B):

```
TASK  0                              TASK  0
      ...                                  ...
     LADE_REGISTER [2300 mit 10]     WARTEZEIT
10
  SOBALD                                  ...
      REGNULL 2300
    DANN
      ...
```

Diese beiden Programmteile haben genau die gleiche Funktion. In manchen Fällen mag es jedoch nützlich sein, mit einem Zeitregister zu arbeiten, weil dabei zwischen Laden des Zeitregisters und der Abfrage auf Null, noch weitere Befehle ausgeführt werden können.

Hinweis:

Der Umgang mit den Zeitregistern hat auch seine Tücken! So ist es sehr gefährlich und nicht empfehlenswert, in ein und demselben Task sowohl die WARTEZEIT-Funktion wie auch das Zeitregister des Tasks direkt zu laden. Auf diese Weise können zum Beispiel "unendliche" Wartezeiten resultieren. Abhilfe: STARTE-ZEIT- und ZEIT-ENDE-Befehle benutzen.

3.11.2 LEER

Der Befehl

LEER

hat nur betriebssysteminterne Bedeutung.

Da es sich beim LEER um einen "echten" Befehl handelt, wird er im Programm bearbeitet, was es in zeitkritischen Programmen ermöglicht, sehr kurze Verzögerungszeiten auszuführen.

3.11.3 Das Kommentarzeichen

Der **";" (Strichpunkt)** ist eigentlich kein Befehl, sondern dient lediglich dazu, in den Programmtext eine Kommentarzeile einfügen zu können.
Es wird dadurch möglich, ausführlichere Kommentare zu schreiben, als diejenige, die in der Kommentarspalte hinter einem Befehl eingetragen werden können.

Da es sich um einen reinen Kommentar handelt, wird die komplette Zeile beim Übersetzen des Quellprogramms durch den Steuerungs-Kompiler eliminiert. In der Steuerung wird somit kein Speicherplatz und keine Programmausführungszeit benötigt.

3.11.4 Spezialfunktionen

Spezial-funktionen nur bei PASE-E und DELTA

Über den Befehl

SPEZIALFUNKTION

können drei interne Steuerungs-Funktionen aufgerufen werden:

Funktion 1: **Initialisiere Achse**

Funktion 4: **BCD -> HEX-Wandlung**

Funktion 5: **HEX -> BCD-Wandlung**

Indirekte Adressierung möglich

Die Parameter p1 und p2 können bei allen Funktionen auch indirekt angegeben werden.

Initialisiere Achse

Für die Achsinitialisierung

SPEZIALFUNKTION [#1, p1=<Par1>, p2=<Par2>]

Diese Funktion ist zur Initialisierung von Achskarten gedacht. Sie kopiert Registerwerte von einem Speicherbereich zu einem anderen. Dieser Befehl ist ausführlich im *Kapitel 3.6.2 Befehle, um Register zu laden* beschrieben.

BCD -> HEX Wandlung

Die

SPEZIALFUNKTION [#4, p1=<Par1> p2=<Par2>]

Zum Beispiel BCD-Schalter können abgefragt werden

dient zur Umwandlung von binär codierten Dezimalzahlen (BCD) in Binärzahlen.
Diese kann zum Beispiel zur Ermittlung von Werten benutzt werden, die von BCD-Schaltern, die an eine Eingangskarte angeschlossen sind, eingelesen wurden.

Die beiden Parameter dieser Funktion sind:

Parameter 1 -> **Quellregisternummer**

Parameter 2 -> **Zielregisternummer**

Die Bits des Quellregisters werden als BCD-Zahl interpretiert, in eine Binärzahl umgewandelt und in das Zielregister geschrieben. Dabei stellen jeweils vier Bits des Quellregisters eine Dezimalstelle dar. Es können maximal vier Stellen verarbeitet werden.

BCD-Zahl im Quellregister:

Bit 0 bis 3 -> letzte Dezimalstelle ("Einer")

Bit 4 bis 7 -> zweitletzte Dezimalstelle ("Zehner")

Bit 8 bis 11 -> drittletzte Dezimalstelle ("Hunderter")

Bit 12 bis 15-> viertletzte Dezimalstelle ("Tausender")

Beispiel:

Das Register 100 habe den Wert:
0101 1000 0011 0110 = 22582

Der Wert der dadurch
gespeicherten BCD-Zahl beträgt jedoch:
5 8 3 6 = 5836

Der Befehl

```
SPEZIALFUNKTION [#4, p1=100, p2=101]
```

hat nun zur Folge, dass das Register 101 den Wert 5836 hat.

HEX -> BCD Wandlung

Die

SPEZIALFUNKTION [#5, p1=<Par1> p2=<Par2>]

Zum Beispiel BDC-kodierte Anzeigen können damit angesteuert werden

dient zur Umwandlung von Binärzahlen in binär codierte Dezimalzahlen (BCD). Sie entspricht somit der Umkehrung der Spezialfunktion 4.

Parameter 1	->	Quellregisternummer (Binäre Zahl)
Parameter 2	->	Zielregisternummer (BCD - Zahl)

3.11.5 Der Befehl GRENZEN

Ein sehr praktischer und Programm-Code sparender Befehl:

```
GRENZEN [Reg.Nr, untere Grenze, obere Grenze]
```

Der Befehl kann vielseitig verwendet werden:

GRENZEN in Eingangs-bedingung

1. GRENZEN nach FALLS oder SOBALD

Hier wird der Wert des durch den GRENZEN-Befehl spezifierten Registers überprüft, ob er in dem durch die untere und obere Grenze festgelegten Intervall liegt. Das Ergebnis dieser Operation ist **wahr (1)** oder **falsch (0)**.

GRENZEN in Ausgangs-anweisung

2. GRENZEN nach DANN oder SONST

Auch hier wird der Wert des durch den GRENZEN-Befehl spezifierten Registers überprüft, ob er in dem durch die Grenzen festgelegten Intervall liegt. Hier ist das Ergebnis jedoch folgendes:

a) **Der Wert liegt unterhalb des Intervalls:**
Hier wird der alte Wert durch den Wert der unteren Grenze ersetzt.

b) **Der Wert liegt oberhalb des Intervalls:**
Der alte Wert wird durch den oberen Grenzwert ersetzt.

c) **Der Wert liegt innerhalb des Intervalls:**
Der alte Wert wird beibehalten.

Die Grenzwerte können auch durch indirekte oder doppelt indirekte Adressierung angegeben werden.

3.11.6 Wortverarbeitung

In diesem Kapitel werden folgende Befehle erläutert:

WUND

WODER

WXODER

Mit diesen drei Befehlen können ganze Register bitweise miteinander logisch verknüpft werden.

Diese logischen Verknüpfungsanweisungen können in gleicher Weise angewendet werden wie die arithmetischen Operatoren + - * / .

Sie können im gleichen Ausdruck verwendet werden, wobei es in der Bearbeitung keine Prioritätsunterschiede gibt.

Nachfolgend werden die Befehle anhand von Beispielen erläutert.

WUND

```
1)    REG 0
      =
      b01010101010101010101010101
      WUND
      b00100100100100100100100100
```

Das erste Bit der ersten Zahl, wird mit dem ersten Bit der zweiten Zahl UND-verknüpft.
Das zweite Bit der ersten Zahl, wird mit dem zweiten Bit der zweiten Zahl UND-verknüpft.
usw.

Mit 0 werden die Bits rückgesetzt
Mit 1 werden die Bits beibehalten

Bei der **W**ortweisen **UND**-Verknüpfung wird nur dasjenige Ergebnisbit '1', wo die entsprechenden Bits der ersten Zahl **und** der zweiten Zahl gesetzt (= 1) sind. Die UND-Verknüpfung eines bestimmten Bits mit '0' setzt das Ergebnisbit auf '0'; bei der UND-Verknüpfung mit '1' wird der Zustand des Bits in das Ergebnis übernommen.

Das Resultat der jeweiligen Verknüpfung wird als entsprechendes Bit im Register REG 0 gespeichert, so dass im Register REG 0 der Wert b00000100000100000100000 1 = 266305 steht.

```
2)    REG 0
      =
      REG 1
      WUND
      h0000FF
```

Bei dieser Zuweisung werden die niederwertigsten acht Bits (h0000FF = b00000000000000011111111) des Registers REG 1 unverändert in das REG 0 übernommen. Die höherwertigeren Bits von REG 0 werden zu '0' gesetzt.

WODER

```
1)    REG 0
      =
      REG 1
      WODER
      b00000000000011110000111
```

Mit 1 werden die Bits gesetzt
Mit 0 werden die Bits beibehalten

Bei der **W**ortweisen **ODER**-Verknüpfung werden diejenigen Ergebnisbits gesetzt (=1), wo die entsprechenden Bits der ersten Zahl **oder** die Bits der zweiten Zahl **oder** die Bits beider Zahlen '1' sind.
Bei der ODER-Verknüpfung eines bestimmten Bits mit '0' wird der Zustand des Bits in das Ergebnisbit übernommen; die Verknüpfung mit '1' setzt das Ergebnis auf '1'.

Das Ergebnis, welches im Register REG 0 gespeichert wird sieht folgendermaßen aus: REG 0 = bxxxxxxxxxxxx1111xxxx1111. x bezeichnet dabei diejenigen Bits, welche vom REG 1 abhängig sind.

WXODER

```
1)    REG 100
      =
      46398
      WXODER
      123098
```

Mit 1 werden die Bits invertiert
Mit 0 werden die Bits beibehalten

Bei der **W**ortweisen E**X**klusiv-**ODER**-Verknüpfung werden diejenigen Bits des Ergebnisses auf '1' gesetzt, wo die entsprechenden Bits der beiden Zahlen unterschiedliche logische Zustände haben. Bei gleichen Zuständen wird das Ergebnis '0'.
Bei einer Exklusiv-ODER-Verknüpfung eines bestimmten Bits mit '0' wird der Zustand dieses Bits übernommen; bei der Verknüpfung mit '1' wird der inverse Wert des Bits in das Ergebnisbit geschrieben.

```
 46398 --> 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0
123098 --> 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 1 0 1 0
  EXODER   ----------------------------------------
 87524 <-- 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 1 0 0
```

Im Register REG 100 steht nach dieser Zuweisung der Wert 87524.

```
2)    REG 100
      =
      REG 100
      WXODER
      hFFFFFF
```

Mit dieser Zuweisung wird jedes Bit vom REG 100 invertiert.

3.12 Netzwerk - Befehle

Mono-Master-Netz

JETWay-R

126 Slaves können angeschlossen werden

Ein Netzwerk ist ein Zusammenschluss mehrerer Steuerungen, remote I/O oder Ventilinseln. Diese können über die folgenden zwei Befehle miteinander kommunizieren, das heißt, es können Registerwerte von der einen Einheit zu einer anderen übertragen werden. Bei diesem Netz handelt es sich um ein "Mono-Master-Netz". Das bedeutet, eine Einheit ist der Master (Nummer 1) und alle anderen sind Slaves (Nummer 2 ...). Diese Nummer kann in einem Register festgelegt werden. Wichtig ist zu beachten, dass beide folgenden Befehle nur im Programm der Master-Einheit eingegeben werden dürfen. Ansonsten sind sie nicht definiert.

Die Befehle

N-HOLE-REGISTER

N-SEND-REGISTER

dienen zum Holen von Registerwerten von einer anderen Steuerung in die Master-Steuerung und zum Senden von Registerwerten von der Master-Steuerung auf eine andere Steuerung.

Komfortabler Netzzugriff mit 50000er-Nummern

Netzbetrieb mit 50000er-Nummern (*Kapitel 3.12.3 Netzbetrieb mit 50000er-Nummern)*

3.12.1 Senden von Registerwerten an Slave-Steuerungen

Dies geschieht mit dem folgenden Befehl:

```
N-SEND-REGISTER      [An      <Netz      Nr>von
Reg.<Quellreg>an
                  Reg.<Zielreg>]
```

Dabei muss bei **Netz Nr** die Nummer der Steuerung stehen, welche man erreichen will. Also die Nummer der Slave-Steuerung, welche den Registerwert erhalten soll.

Indirekte Adressierung möglich

Bei **Quellregister-Nummer** muss die Nummer des Register angegeben werden, welches gelesen werden soll. Dies ist ein Register der Master-Steuerung. Dieser Parameter kann auch indirekt angegeben werden (zum Beispiel R10).

Bei **Zielregister-Nummer** muss die Nummer des Registers, der Slave-Steuerung angegeben werden. Also dasjenige Register, der Slave-Steuerung, in das der Wert hineingeschrieben werden soll. Auch diese Registernummer kann indirekt adressiert werden, wobei das Zeigerregister in der **Master-Steuerung** ist.

Spezialmerker zeigt Netzwerk-fehler an

Der Spezialmerker "Netzwerk-Timeout" zeigt dabei an, ob ein Fehler (Übertragungsfehler oder Timeout) bei der letzten Übertragung aufgetreten ist. Aus diesem Grunde sollte nach jeder Übertragung dieser Merker überprüft werden. Ist der Merker gesetzt, so ist ein Fehler aufgetreten und die Datenübertragung kann wiederholt werden.

Beispiel:

```
MARKE  100
    N-SEND-REGISTER  [An  2  von  Reg.100  an
Reg.200]
  FALLS
      MERKER rsNetzwerk-Timeout
    DANN
      SPRUNG 100
    DANN
      ...
```

Spezialmerker "Netzwerk-Timeout" zeigt Übertragungs-fehler an

Bei diesem Beispiel wird der Wert des Registers 100 von der Master-Steuerung an die Steuerung Nr. 2 gesendet und dort ins Register 200 geschrieben. Tritt dabei ein Fehler auf, so wird der Spezialmerker "Netzwerk-Timeout" gesetzt, sonst wird er zurückgesetzt; das Programm springt deshalb zur Marke 100 zurück und wiederholt den Sendevorgang. Bei diesem Beispiel wird das Senden des Registerwertes so oft wiederholt, bis die Übertragung fehlerfrei durchgeführt werden konnte.

3.12.2 Registerwerte von einer Slave-Steuerungen holen

Dies geschieht mit dem Befehl:

```
N-HOLE-REGISTER  [von  <Netz  Nr>Reg.<Quellreg>,
Reg
                    hier=<Zielreg>]
```

Die anzugebenden Parameter haben folgende Bedeutung:

Netz Nr

Ist die Netznummer der Slave-Steuerung, von der ein Register gelesen werden soll.

Quellregister - Nummer

Indirekte Adressierung möglich

Ist die Nummer desjenigen Registers, der Slave-Steuerung, welches gelesen werden soll (Indirekte Adressierung möglich).

Zielregister - Nummer

Ist die Nummer des Registers der "eigenen" Steuerung, wo der Wert hineingeschrieben werden soll (Indirekte Adressierung möglich).

Anmerkung:

Bei indirekter Adressierung sind die Register, in denen die Quell- und Zielregisternummern stehen, stets auf der **Master-Steuerung**.

Spezialmerker "Netzwerk-Timeout" meldet Netzwerkfehler

Auch hier wird bei einem Fehler der Spezialmerker "Netzwerk-Timeout" gesetzt.

Beispiele :

1)

```
   MARKE  100
     N-HOLE-REGISTER    [von    2Reg.100,    Reg
hier=200]
   FALLS
        MERKER rsNetzwerk-Timeout
      DANN
        SPRUNG 100
      DANN
        ...
```

Hier wird von der Slave-Steuerung mit der Netzwerknummer 2 das Register 100 gelesen und auf

das Register 200 der Master-Steuerung kopiert. Mittels Spezialmerker "Netzwerk-Timeout" wird geprüft, ob ein Fehler aufgetreten ist.
Ist ein Fehler aufgetreten, so wird das Register 100 in der Slavesteuerung erneut gelesen.

```
2)    LADE_REGISTER [100 mit 1000]
MARKE  100
      N-HOLE-REGISTER    [von    2Reg.R100,    Reg
hier=R100]
  FALLS
      MERKER rsNetzwerk-Timeout
    DANN
      SPRUNG 100
    DANN
      REGINC 100
  FALLS
      REG 100
      <
      1030
    DANN
      SPRUNG 100
    DANN
```

Dieses Programm kopiert einen ganzen Registerbereich (die Register 1000 bis 1029) von der Slave-Steuerung mit der Netzwerknummer 2 auf die Master-Steuerung; ebenfalls in den Registerbereich 1000 bis 1029.

Eine "Sammelmeldung", ob jemals ein Übertragungsfehler aufgetreten ist, findet sich im Spezialmerker "Sammelmeldung Netzwerkfehler". Dieser wird bei einem Fehler gesetzt, vom Betriebssystem aber nicht mehr zurückgesetzt. Das Zurücksetzen kann vom Anwenderprogamm aus erfolgen.

3.12.3 Netzbetrieb mit 50000er-Nummern

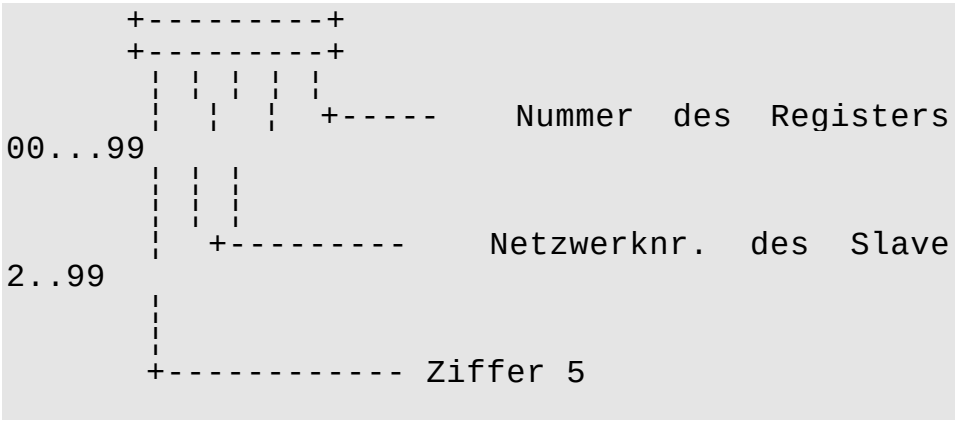
3.12.3.1 Ansprechen der Register

Gleicher Zugriff auf Master- und Slaveregister

Zugriffe auf Register einer Steuerung durch die Master-Steuerung unterscheiden sich nur durch die Parameternummer von einem internen LADE_REGISTER-Befehl. Abgesehen von dieser Nummer sind die Programmsequenzen für einen Zugriff auf ein internes Register und ein Slaveregister identisch.

Registernummer 00...99

Die Registernummer hat folgendes Schema:



Mit diesen Registernummern kann die Master-Steuerung auf ein 100 Register umfassendes Fenster in der Slave-Steuerung zugreifen.

Beispiel:

Das Register 62 in der Slave-Steuerung mit der Netznummer 32 wird von der Master-Steuerung aus mit dem Befehl

```
LADE_REGISTER [ 100 mit R(53262) ]
```

angesprochen.

Mit dem Nummern-offsetregister werden "Fenster" für den Slave-registerzugriff gesetzt

Soll auf ein Register zugegriffen werden, dessen Nummer größer als 99 ist, so ist in das Spezialregister "Nummernoffset Register" der Slave-Steuerung ein Nummeroffsetwert einzugeben. Dieser wird bei einem Zugriff der Master-Steuerung auf die Register der Slave-Steuerung zu der Registernummer im Programm der Master-Steuerung addiert.

Der Befehl

```
LADE_REGISTER [ 100 mit R(53262) ]
```

im Programm der Master-Steuerung zuzüglich einem Wert von 200 in Spezialregister "Nummernoffset Register" der Slave-Steuerung mit der Netznummer 32 greift effektiv auf das Register 262 der Slave-Steuerung zu.

Spezialregister "Nummeroffset Register"

Dieser Wert wird zu der Registernummer im Programm der Master-Steuerung addiert. Die Summe ergibt das Register, auf welches die Master-Steuerung in der Slave-Steuerung wirklich zugreift.

Wert nach Reset: 0

Hinweis:

Mit dem Befehl N-SENDE-REGISTER setzt man in der Slave-Steuerung das "Register-Fenster" mit dem Spezialregister "Nummernoffset Register" auf den gewünschten Bereich und kann dann mit den 50000er-Nummern in diesem "Fenster" arbeiten.

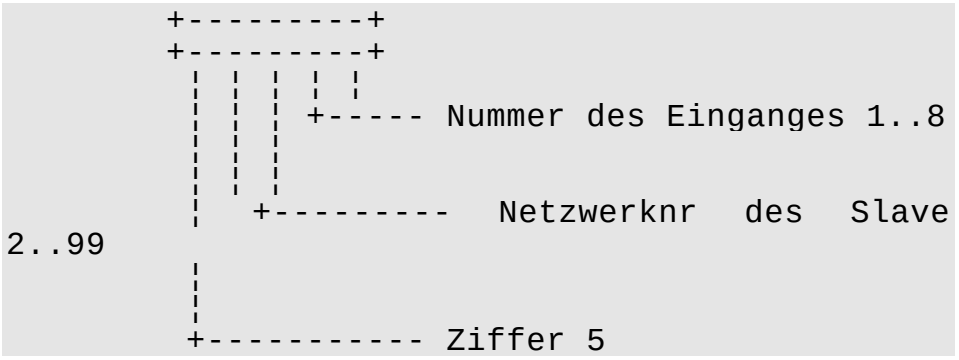
3.12.3.3 Ansprechen der Eingänge, Ausgänge und Merker

Ansprechen der Eingänge

Gleicher Zugriff auf Master- und Slave-eingänge

Zugriffe auf Eingänge der Slave-Steuerung durch die Master-Steuerung unterscheiden sich nur durch die Parameternummer von einem internen Master-Eingangsbefehl. Abgesehen von dieser Nummer sind die Programmsequenzen für einen Zugriff auf einen Master-Eingang und einen Slave-Eingang identisch.

Die Eingangsnummer hat folgendes Schema:



Mit dem Nummern-offsetregister werden "Fenster" für den Slave-eingangs-zugriff gesetzt

Auf die im Eingangsparameter definierte Eingangsnummer wird der Zahlenwert aus dem dazugehörigen Nummernoffsetregister für Eingänge addiert. Der resultierende Eingang wird angesprochen.

Spezialregister "Nummernoffset Eingänge":

Nummernoffset für Eingänge; das Register befindet sich auf der Slave-Steuerung.
Dieser Wert wird zu der Eingangsnummer im Programm der Master-Steuerung addiert. Die Summe ergibt den

Eingang, auf welchen die Master-Steuerung in der Slave-Steuerung wirklich zugreift.

Beispiel:

Der Eingang 108 in der Slave-Steuerung mit der Netzwerknummer 5 wird von der Master-Steuerung aus mit dem Befehl

```
EINGANG 50508
```

angeprochen.

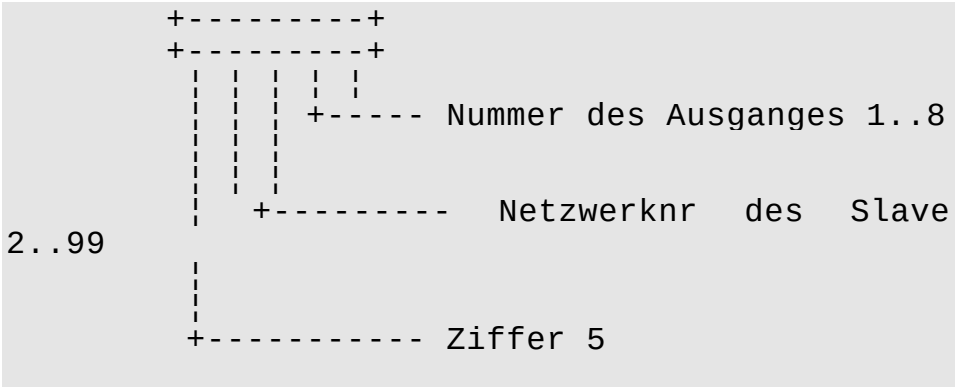
Zuvor muss im Spezialregister "Nummernoffset Eingänge" (auf der Slave-Steuerung) der Wert 100 eingetragen werden.

Ansprechen der Ausgänge

Gleicher Zugriff auf Master- und Slave-ausgänge

Zugriffe auf Ausgänge der Slave-Steuerung durch die Master-Steuerung unterscheiden sich nur durch die Parameternummer von einem internen Master-Ausgangsbefehl. Abgesehen von dieser Nummer sind die Programmsequenzen für einen Zugriff auf einen Master-Ausgang und einen Slave-Ausgang identisch.

Die Ausgangsnummer hat folgendes Schema:



Mit dem Nummern-offsetregister werden "Fenster" für den Slave-ausgangs-zugriff gesetzt

Auf die im Ausgangsparameter definierte Ausgangs-nummer wird der Zahlenwert aus dem dazugehörigen Nummernoffsetregister für Ausgänge addiert. Der resultierende Ausgang wird angesprochen.

Spezialregister "Nummernoffset Ausgänge":

Nummernoffset für Ausgänge; das Register befindet sich auf der Slave-Steuerung.
Dieser Wert wird zu der Ausgangsnummer im Programm der Master-Steuerung addiert. Die Summe ergibt den Ausgang, auf welchen die Master-Steuerung in der Slave-Steuerung wirklich zugreift.

Beispiel:

Der Ausgang 108 in der Slave-Steuerung mit der Netzwerknummer 5 wird von der Master-Steuerung aus mit dem Befehl

```
AUSGANG 50508
```

angeprochen.

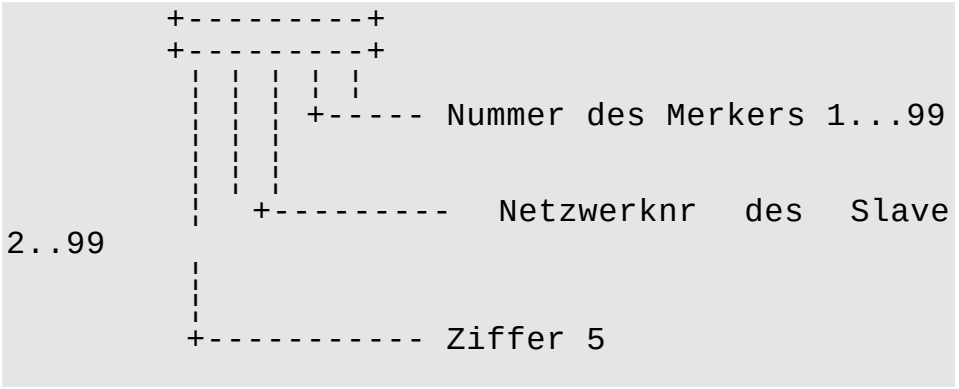
Zuvor muss im Spezialregister "Nummernoffset Ausgänge" (auf der Slave-Steuerung) der Wert 100 eingetragen werden.

Ansprechen der Merker

Gleicher Zugriff auf Master- und Slavemerker

Zugriffe auf Merker der Slave-Steuerung durch die Master-Steuerung unterscheiden sich nur durch die Parameternummer von einem internen Master-Merkerbefehl. Abgesehen von dieser Nummer sind die Programmsequenzen für einen Zugriff auf einen Master-Merker und einen Slave-Merker identisch.

Die Merkernummer hat folgendes Schema:



Mit dem Nummern-offsetregister werden "Fenster" für den Slave-merkerzugriff gesetzt

Auf die im Merkerparameter definierte Merkernummer wird der Zahlenwert aus dem dazugehörigen Nummern-offsetregister für Merker addiert. Der resultierende Merker wird angesprochen.

Spezialregister "Nummernoffset Merker":

Nummernoffset für Merker; das Register befindet sich auf der Slave-Steuerung.
Dieser Wert wird zu der Merkernummer im Programm der Master-Steuerung addiert. Die Summe ergibt den Merker, auf welchen die Master-Steuerung in der Slave-Steuerung wirklich zugreift.

Beispiel:

Der Merker 154 in der Slave-Steuerung mit der Netzwerknummer 12 wird von der Master-Steuerung mit dem Befehl

```
MERKER 51254
```

angesprochen.

Zuvor muss im Nummernoffsetregister für Merker (auf der Slave-Steuerung) der Wert 100 eingetragen werden.

3.12.4 Spezialregister / -merker zum Netzwerkbetrieb

Merker 2110

Zeigt Fehler beim letzten Netzwerkbefehl an (Prüfsumme oder Timeout).

Merker 2111

Zeigt Fehler bei einem Netzwerkbefehl seit Reset der Steuerung (akkumulierend Merker 2110).

Register "Netzwerknummer Netzwerk 1"

Netzwerknummer Netzwerk 1.

Register "Netzwerknummer Netzwerk 2"

Netzwerknummer Netzwerk 2.

Register "Netzwerkreaktionszeit"

Gibt die Netzwerkreaktionszeit in Millisekunden an. Die Zeit startet, wenn ein Netzwerkbefehl ausgeführt wird und endet, wenn die Antwort von der anderen Steuerung per Netz eingetroffen ist (Vor allem abhängig von der Belastung der anderen Steuerung). Dieses Funktion bezieht sich jeweils auf das Masternetz einer Steuerung - also das Netz einer Steuerung in welchem diese Master ist. Eine Steuerung kann zur selben Zeit in maximal einem Netz Master sein.

Register "Ausführungszeit Netzwerkbefehl"

Gibt die Bearbeitungszeit des letzten ausgeführten Netzwerkbefehls an (in Millisekunden). Dies ist die Zeit in Register "Netzwerkreaktionszeit" plus der Zeit, die vergeht, bis die Antwort der anderen Steuerung vom zugehörigen Programm-Task der eigenen Steuerung verarbeitet wurde und dieser Task weiterläuft (von Belastung der anderen und der eigenen Steuerung abhängig). Dieses Funktion bezieht sich jeweils auf das Masternetz einer Steuerung - also das Netz einer Steuerung in welchem diese Master ist. Eine Steuerung kann zur selben Zeit in maximal einem Netz Master sein.

Register "Timeout Netzwerkzugriff"

Timeoutzeit in Millisekunden. Voreinstellung: 250. Nach dieser Zeit wird ein Netzwerkbefehl abgebrochen. Die angesprochene Steuerung hat sich nicht gemeldet (evtl. abgeschaltet).

Register "Anzahl Prüfsummenfehler Netzwerk-empfang"

Prüfsummenfehler. Inhalt wird bei jedem Prüfsummenfehler um eins erhöht. Um die Qualität der Netzwerkübertragung beurteilen zu können, kann dieses Register von Zeit zu Zeit gelesen werden.

4. Beschreibung des Speichers

Dieser Überblick ist bewusst möglichst allgemein gehalten nicht konkret auf eine bestimmte Steuerung zugeschnitten

Der Speicherplatz, der dem Steuerungs-Anwender zur Verfügung steht, ist in Register und Merker aufgeteilt. Dieses Kapitel soll über die Aufteilung des gesamten Speichers Aufschluss geben. Dabei werden auch viele vom Betriebssystem benutzte Register beschrieben, welche dem Anwender in manchen Fällen hilfreich sein können. Es muss jedoch darauf hingewiesen werden, das beim Ändern von Spezialregistern Vorsicht geboten ist.

4.1 Grundsätzliches über Register und Merker

4.1.1 Register

Register sind die Speicher der Steuerung in der die Daten gehalten werden

Die Register sind die Speicher der Steuerung. In ihnen sind alle benötigten Zahlenwerte gespeichert. Auch der Anwender kann einen Teil dieser Register benutzen, und zwar um Werte abzuspeichern oder um mit diesen zu rechnen. Die Register können entweder vom Programm aus geladen werden, oder mit Hilfe von SYMPAS. Dabei können ganze Datenblöcke mittels PC in die Steuerung übertragen werden. Dies ist in vielen Fällen nützlich, da das Laden von Registern in einem komplexeren Programm sehr viel Platz rauben kann.

Es wird zwischen **Ganzzahlregistern**, **Gleitkommaregistern** und **Spezialregistern** unterschieden. Alle Register werden durch eine Nummer gekennzeichnet. Im folgenden werden diese drei Registerarten erklärt:

Ganzzahlregister

Registerbreite: 24 Bit -> 23 Bit plus Vorzeichen

Bei diesen Registern handelt es sich um 24 Bit breite Register, in denen eine **ganze Zahl** zwischen **-8388608 und 8388607** gespeichert wird. Das höchstwertige Bit speichert dabei das Vorzeichen der Zahl. Der Wert dieser Register kann auch als binäre Zahl (b . . . .) oder hexadezimale Zahl (h . . . .) angegeben werden. Siehe dazu Codierung weiter unten.

Die spezifischen Registernummern sind dem jeweiligen Steuerungshandbuch zu entnehmen

Die Registernummern sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein allgemeiner Überblick über die Register der PROZESS-SPS gegeben.

Registeraufbau:

Das 24ste Bit speichert das Vorzeichen, die anderen 23 Bits eine Zahl, welche dem binären Wert dieser 23 Bits entspricht:

Ist das Vorzeichenbit Null, so entspricht dieser Wert genau der Zahl des Registers.

Ist das Vorzeichenbit jedoch Eins, so kann die Zahl durch addieren dieses binären Wertes zu -8388608 (= -h800000) ermittelt werden.

Beispiele

Binäre Zahl (24 Bit):	Hex - Zahl	Dez - Zahl
b000000000000000001101111	h00006F	111
b011111111111111111111111	h7FFFFF	8388607
b100000000000000000000000	h800000	-8388608
b111111111111111111111111	hFFFFFF	-1
b100000000000000011010011	h8000D3	-8388397

Gleitkommaregister

NANO-A verfügt über keine Gleitkomma-register

Diese Register sind 32 Bit breit und speichern reelle Zahlen, also im Prinzip beliebige gebrochene Zahlen in einem Bereich von

$$\mathbf{-10^{15} \quad bis \quad +10^{15}}$$

Betragsmäßig liegt die kleinstmögliche Zahl bei 10^{-15}.

Die Genauigkeit der Berechnungen liegt bei 7 zählenden Stellen, weil nur soviele Stellen in den 32-Bit Registern gespeichert werden können.

Ihre Funktion ist vor allem das genaue Berechnen auch von gebrochenen Zahlen. Bei Zuweisung von gebrochenen Zahlen auf ein Ganzzahlregister gehen die Nachkommastellen immer verloren. Wird zum Beispiel der Wert -2,5 (Ergebnis einer Division) in ein Ganzzahlregister geladen, so steht dort der Wert -2.

Eine weitere wichtige Verwendung der Gleitkommaregister ist die Berechnung von Ausdrücken, in denen mit Ergebnissen größer als 8

Millionen gerechnet werden muss. In einem Ganzzahlregister kann dies zu praktisch undefinierten Werten führen. Das Beispiel unten soll dieses Problem veranschaulichen.

Die Registernummern sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein allgemeiner Überblick über die Register der PROZESS-SPS gegeben.

Beispiel: Einfache Zuweisung

```
REG 1
=
2
*
5'000'000
```

Bei der Zuweisung des Wertes 10'000'000 auf das Ganzzahlregister 1 passiert folgendes:

Die Zahl, welche ja als binäre Zahl vorliegt wird in das Register hineingeladen, doch weil die Zahl länger ist als das Register, gehen vorne Bits verloren, respektive das Bit 23 (Vorzeichen) wird besetzt. Das Ergebnis lautet wie folgt:

```
    10'000'000  =  h989680
                =
b1001'1000'1001'0110'1000'0000

      ->  Reg 1 = -6'777'216
```

Spezialregister

Es gibt im wesentlichen zwei Arten von Spezialregistern : Die einen befinden sich auf intelligenten

Erweiterungsmodulen und speichern Parameter oder Statusinformationen dieser Module (diese werden im Zusammenhang mit den jeweiligen Modulen in den entsprechenden Steuerungshandbüchern ausführlich beschrieben). Daneben gibt es solche Register, welche vom Betriebssystem der Steuerung benutzt werden.

Beim Umgang mit Spezial-registern ist Vorsicht geboten

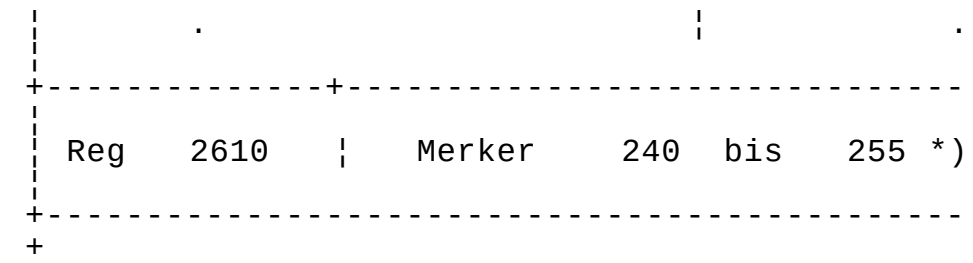
Die Spezialregisternummern sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein allgemeiner Überblick über die Register der PROZESS-SPS gegeben.

Register mit zusammengefassten Merkern:

Die Überlagerung wird hier beispielhaft an der Nummerierung der NANO-B dargestellt

Die Spezialregisternummern mit den Überlagerungen sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein Beispiel zur NANO-B dargestellt. Die Merker 0 bis 255 sind mit den Registern 2600 bis 2610 überlagert.

Reg	2600	Merker	0	bis	23
Reg	2601	Merker	24	bis	47
Reg	2602	Merker	48	bis	63
Reg	2603	Merker	64	bis	87
.					.
.					.



*) Register 2610:

Wie alle Register besteht das Register 2610 aus 24 Bit. Von diesen Bits sind nur **die ersten 16 überlagert mit den Merkern 240 bis 255.**

Überlagerung Merker-Register: Beispiele an den Registern 0, 1,2

Register 2600 - Merker 0 bis 23:

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0  Bit-
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0  Nummer
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  Merker-
7 7 7 7 7 7 7 7 7 7 6 6 6 6 6 6 6 6 6 5 5 5 5  Nummer
9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6
```

Die Bit- bzw. Merkernummern sind von oben nach unten zu lesen.

Register 2601 - Merker 24 bis 47:

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0  Bit-
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0  Nummer
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦
3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  Merker-
0 0 0 0 9 9 9 9 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8  Nummer
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

Die Bit- bzw. Merkernummern sind von oben nach unten zu lesen.

Register 2602 - Merker 48 bis 63:

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0  Bit-
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0  Nummer
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  Merker-
2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0  Nummer
7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4
```

Die Bit- bzw. Merkernummern sind von oben nach unten zu lesen.

Register mit zusammengefassten Eingängen oder Ausgängen

Einfacher Zugriff auf mehrere in Registern zusammen-gefasste Ein- oder Ausgänge

In verschiedenen Register der Steuerungen sind **8 oder 16 oder 24 Eingänge** in einem Register zusammen-gefasst. Gleiches gilt für die digitalen **Ausgänge**.

Die Spezialregisternummern mit den Überlagerungen sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein Beispiel zur DELTA dargestellt.

32 Register zu je 8 Eingängen

Diese 8 Eingänge werden in die Bits 0 bis 7 geschrieben, alle anderen Bits (8 bis 23) sind 0. Daraus folgt ein Wertebereich von 0 bis 255 für diese Register. Die **32 Register von 62464 bis 62495** entsprechen je 8 Eingängen:

Nummerierung beispielhaft: DELTA

RegNr	Eingänge		RegNr	Eingänge
62464	101 - 108		62480	301 - 308
62465	109 - 116		62481	309 - 316
62466	117 - 124		62482	317 - 324
62467	125 - 132		62483	325 - 332
62468	133 - 140		62484	333 - 340
62469	141 - 148		62485	341 - 348
62470	149 - 156		62486	349 - 356
62471	157 - 164		62487	357 - 364
62472	201 - 208		62488	401 - 408
62473	209 - 216		62489	409 - 416
62474	217 - 224		62490	417 - 424
62475	225 - 232		62491	425 - 432
62476	233 - 240		62492	433 - 440
62477	241 - 248		62493	441 - 448
62478	249 - 256		62494	449 - 456
62479	257 - 264		62495	457 - 464

32 Register zu je 16 Eingängen

Diese 16 Eingänge werden in die Bits 0 bis 15 geschrieben, alle anderen Bits (16 bis 23) sind 0. Daraus folgt ein Wertebereich von 0 bis 65535 für diese Register. Die **32 Register von 62528 bis 62559** entsprechen je 16 Eingängen:

Nummerierung am Beispiel von DELTA

RegNr	Eingänge		RegNr	Eingänge
62528	101 - 116		62544	301 - 316
62529	109 - 124		62545	309 - 324
62530	117 - 132		62546	317 - 332
62531	125 - 140		62547	325 - 340
62532	133 - 148		62548	333 - 348
62533	141 - 156		62549	341 - 356
62534	149 - 164		62550	349 - 364
62535	157 - 164		62551	357 - 364
62536	201 - 216		62552	401 - 416
62537	209 - 224		62553	409 - 424
62538	217 - 232		62554	417 - 432
62539	225 - 240		62555	425 - 440
62540	233 - 248		62556	433 - 448
62541	241 - 256		62557	441 - 456
62542	249 - 264		62558	449 - 464
62543	257 - 264		62559	457 - 464

32 Register zu je 24 Eingängen

Diese 24 Eingänge werden in die Bits 0 bis 23 geschrieben. Das Bit 23, welches demjenigen Eingang mit der jeweils höchsten Nummer entspricht, bestimmt das Vorzeichen des resultierenden Ganzzahlwertes. Der Wertebereich dieser Register ist gleich demjenigen aller Ganzzahlregister, also von -8388608 bis 8388607. Die **32 Register von 62592 bis 62623** entsprechen je 24 Eingängen:

Nummerierung am Beispiel von DELTA

RegNr	Eingänge		RegNr	Eingänge
62592	101 - 124		62608	301 - 324
62593	109 - 132		62609	309 - 332
62594	117 - 140		62610	317 - 340
62595	125 - 148		62611	325 - 348
62596	133 - 156		62612	333 - 356
62597	141 - 164		62613	341 - 364
62598	149 - 164		62614	349 - 364
62599	157 - 164		62615	357 - 364
62600	201 - 224		62616	401 - 424
62601	209 - 232		62617	409 - 432
62602	217 - 240		62618	417 - 440
62603	225 - 248		62619	425 - 448
62604	233 - 256		62620	433 - 456
62605	241 - 264		62621	441 - 464
62606	249 - 264		62622	449 - 464
62607	257 - 264		62623	457 - 464

Beispiele:

1)

```
LADE_REGISTER [ 62528 mit 255]
```

Hier wird der Wert 255 in das Register geladen , welches mit den Eingängen E 101 bis E 116 übereinstimmt. Damit werden also die niederwertigsten Bits gesetzt, und die restlichen gelöscht. Die Folge davon ist, dass die Eingänge E 101 bis E 108 gesetzt (also aktiv) sind und die Eingänge E 109 bis E 116 gelöscht sind.

2)

Komfortable Maskierung der Ein- und Ausgänge

Gute Möglichkeiten ergeben sich mit diesen Registern auch vorallem in der Kombination mit den Registerbefehlen WUND, WODER und WXODER.

```
REG 62784
=
REG 62784
WUND
b00000000101010101010101010
```

Diese Zuweisung hat zur Folge, dass von den Ausgängen A 101 bis A 116, alle ungeraden (A 101, A 103, A 105 etc.) ausgeblendet, respektive gelöscht werden. Die übrigen Ausgänge behalten ihren alten Zustand bei.

4.1.2 Merker

Merker haben den Wert 1 oder 0

Merker sind eigentlich Ein-Bit-Register. Sie können also die Werte 1 oder 0 speichern. Der Anwender kann die Merker zum Kennzeichnen von Zuständen verwenden. So kann über die Merker eine sehr einfach programmierbare zeitliche Abstimmung von verschiedenen Tasks erreicht werden. Auch bei den Merkern wird zwischen Spezialmerkern und den "normalen" Merkern unterschieden. Die Spezialmerker sind vom Betriebssystem benutzt und speichern jeweils einen Zustand, zum Beispiel einen Tastendruck von der Eingabetastatur oder Fehlermeldungen etc.

Alle Merker können mit den Merkerbefehlen verändert, also gesetzt, gelöscht oder auch nur abgefragt werden. Diese Merkerbefehle sind im *Kapitel 3.6.5 Merker und Merker -Befehle* genau erklärt.

Spezialmerker

Beim Umgang mit Spezial-merkern Vorsicht walten lassen!

Die Spezialmerker werden vom Betriebssystem verwendet um bestimmte Zustände anzuzeigen oder Funktionen zu steuern.
Die Spezialmerkernummern sind den entsprechenden Steuerungshandbüchern zu entnehmen. Hier wird ein allgemeiner Überblick über die Register und Merker der PROZESS-SPS gegeben.

5. Echtzeituhr

5.1 Überblick, Funktion

Auf verschiedenen PROZESS-SPS ist eine Echtzeituhr (Real Time Clock) integriert. Die Echtzeituhr ist unabhängig von dem RAM-Speicher batteriegepuffert.

Die Echtzeituhr wird hier exemplarisch anhand der Register-nummern der DELTA dargestellt

Die hier exemplarisch verwendeten Registernummern beziehen sich auf die DELTA.

Es gibt zwei Registersätze von jeweils 8 Registern. Registersatz 1 (62920 bis 62927) ist schreib- und lesbar. Schreibzugriffe mit diesen Registernummern schreiben direkt an das Echtzeituhrmodul (stellen der Uhrzeit), Lesezugriffe lesen direkt aus dem Uhrzeitmodul.

Außerdem gibt es den Registersatz 2 (62912 bis 62919). Dieser zweite Registersatz hat folgenden Sinn: Wenn per Programm auf eine bestimmte Uhrzeit gewartet wird, dann muss verhindert werden, dass während der Vergleichsoperation sich die Operanden (Uhrzeit...) ändern. Deshalb werden bei jedem Lesezugriff auf Registersatz 1 alle Echtzeitdaten in die Register des Satzes 2 kopiert. Dort stehen sie unverändert zu Verfügung, bis wieder auf ein Register des Registersatzes 1 lesend zugegriffen wird (siehe Beispielprogramm).

Zum Stellen der Uhr werden die Werte in den Registersatz 2 eingetragen und dann durch Beschreiben eines der Register von Satz 1 komplett an die Echtzeituhr übertragen.

5.2 Registerbeschreibung

Registersatz1 schreiben/lesen direkt	**Registersatz 2** lesen/schreiben Puffer	**Daten**	**Bereich**
62920	62912	Sekunden	0-59
62921	62913	Minuten	0-59
62922	62914	Stunden	0-23
62923	62915	12/24h Format	0,128
62924	62916	Tag der Woche	1-7
62925	62917	Tag (Datum)	1-31
62926	62918	Monat	1-12
62927	62919	Jahr	0-99

Folgende Spezialfunktion ist im Register 62924 versteckt: Der Inhalt dieses Registers ist "Tag der Woche". 1=Sonntag, 2=Montag, 3=Dienstag usw.

Um die Uhrzeit in der üblichen Darstellung anzuzeigen bzw. drucken zu können, wurde der Wertebereich des Spezialregisters **61454** erweitert. Wenn dieses Register den Wert 2 hat, dann wird die Vorzeichenstelle bei einem ANZEIGE_REG Befehl unterdrückt (siehe Beispielprogramm).

5.3 Beispielprogramm Echtzeituhr

Die Echtzeituhr wird hier exemplarisch anhand der Register-nummern der DELTA dargestellt

Das folgende Beispielprogramm zeigt die aktuellen Daten der Echtzeituhr auf der Anzeige an.
Folgender Trick wurde angewandt, um bei der Minuten- und Sekundendarstellung führende Nullen zu erhalten:
Bei rechtbündiger Zahlendarstellung mit dem Register 61453 kann bestimmt werden, wieviele Stellen angezeigt werden. Werden weniger Stellen zugelassen als signifikante Stellen in der Zahl vorhanden sind, so werden **führende** Stellen weggelassen.
Im Programm wird das genutzt, indem der Wert 100 zu Sekunden und Minuten addiert wird und diese führende 1 dann aber nicht dargestellt wird.

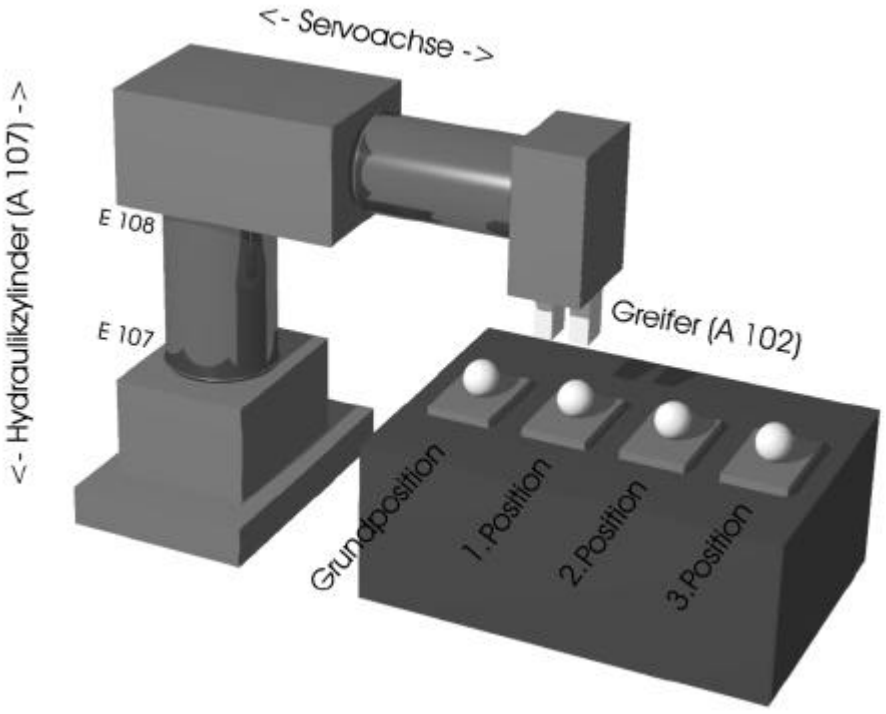
```
   0: TASK 0──────────────────────────────────────
   1:       ;
   2:       LADE_REGISTER [61454 mit 2]              ;kein
Vorzeichen
   3:       LADE_REGISTER [61453 mit 6]              ;2-stellige
Zahlen
   4:       ANZEIGE_TEXT [#0, cp=1, "_Aktuelle Uhrzeit ist:"]
   5:       ;
   6: MARKE 100
   7:       UNTERPROGRAMM 900
   8:       WARTEZEIT 5
   9:       SPRUNG 100
  10:       ;
  11: MARKE 900                                      ;-> ANZEIGEN
  12:       ANZEIGE_REG [#0, cp=25, Reg=62925]       ;Uhrzeit
speichern
  13:       ANZEIGE_TEXT [#0, cp=27, ".  .19  ,  :  :"]
  14:       ANZEIGE_REG [#0, cp=28, Reg=62918]       ;Monat
  15:       ANZEIGE_REG [#0, cp=33, Reg=62919]       ;Jahr
  16:       ;
  17:       ;------- Uhrzeit anzeigen -------
  18:       ;
  19:       ANZEIGE_REG [#0, cp=36, Reg=62914]       ;Stunde
  20:       REG 900                                  ;TRICK, damit
  21:       =                                        ;Zehnerstelle
  22:       REG 62913                                ;dargestellt
wird,
  23:       +                                        ;auch wenn
sie den
  24:       100                                      ;Wert 0 hat.
  25:       ANZEIGE_REG [#0, cp=39, Reg=900]         ;Minute
  26:       REG 900                                  ;TRICK, damit
  27:       =                                        ;Zehnerstelle
  28:       REG 62912                                ;dargestellt
wird,
```

```
  29:       +                                          ;auch wenn
sie den
  30:       100                                        ;Wert 0 hat.
  31:       ANZEIGE_REG [#0, cp=42, Reg=900]           ;Sekunde
  32:       RÜCKSPRUNG
```

6. Demonstrationsbeispiel: Handling-System

6.1 Problemstellung

Als Demonstrationsbeispiel ist nachfolgend das Steuerungsprogramm für eine zweiachsige Maschine gemäß folgender Abbildung dargestellt:



Die vertikale Achse wird durch Setzen des Ausganges 107 nach unten, und durch Rücksetzen wieder nach oben bewegt (Hydraulikzylinder). Die Eingänge 108 und 107 werden beim Erreichen der Grund- (E 108) bzw. der Arbeitsposition (E 107) des vertikalen Zylinders aktiv.
Bei dem horizontalen Zylinder handelt es sich um eine Servo-NC-Achse. Der Greifer wird mittels des Ausganges 2 geöffnet bzw. geschlossen.

Es sollen Teile von der Grundposition aus, der Reihe nach, zu drei verschiedenen Ablagepositionen gebracht werden. Diese Ablagepositionen sind im Teach-In-Mode vom Anwender frei programmierbar; es wird im Handbetrieb zu der gewünschten Position gefahren und diese per Tastendruck am Displaymodul gespeichert.
Neben dem Automatikbetrieb sollen alle Bewegungsabläufe auch von Hand aus durchgeführt werden können.
Außerdem wird der Prozess durch interaktive Ein-/ und Ausgabe auf dem Bediengerät unterstützt.

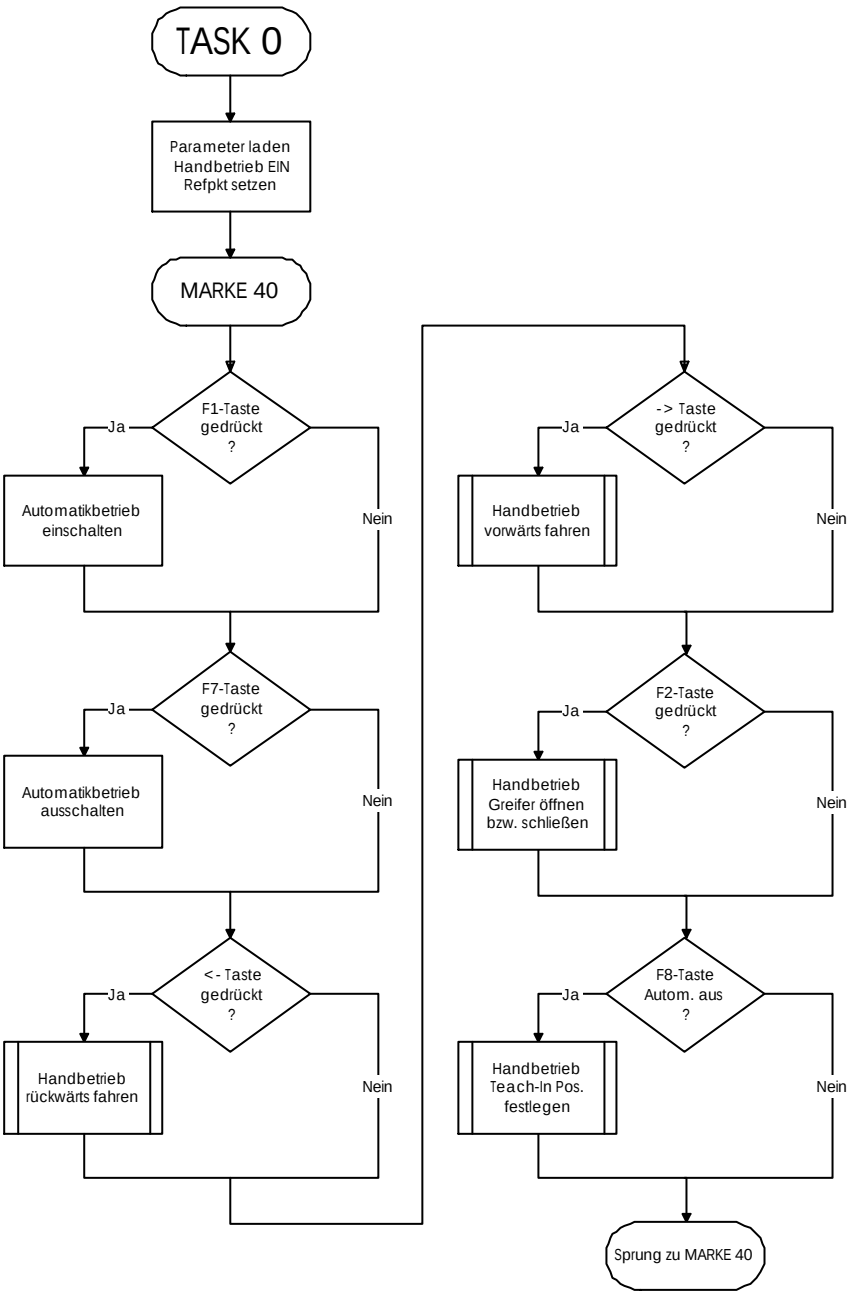
Folgende Tasten am Bediengerät (LCD9, LCD 10) finden zur Steuerung des Prozesses Verwendung.

Taste:	**Funktion:**
F1	Automatik EIN, Handbetrieb AUS
F2	Greifer AUF/ZU
F7	Handbetrieb EIN, Automatik AUS
F8	Teach-In; Abspeichern der Grund- und der drei Ablagepositionen
<-	Handfahrt "Rückwärts"
->	Handfahrt "Vorwärts"

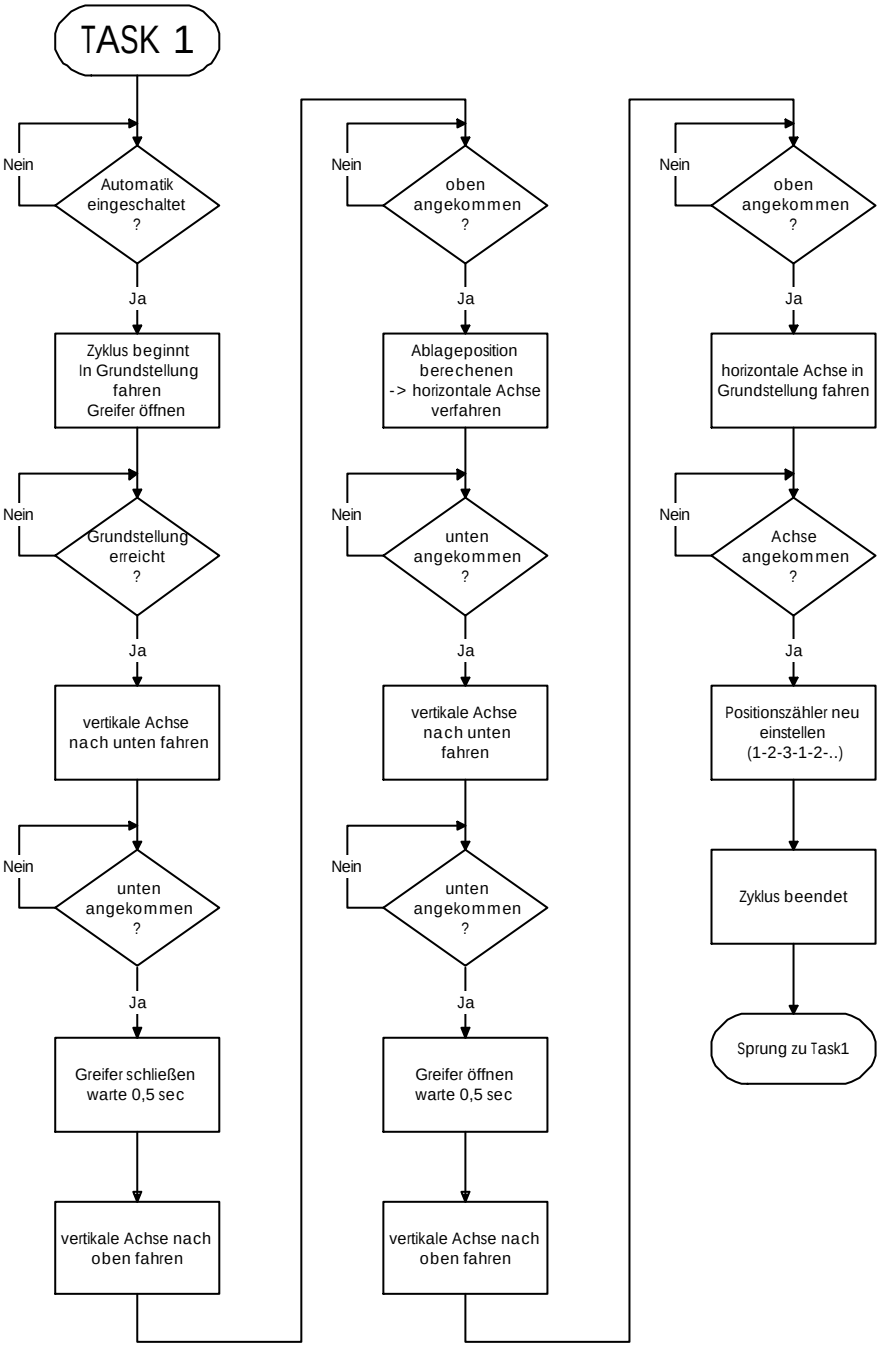
Das Programm ist in drei Haupttask gegliedert. Die folgenden Seiten geben einen umfassenden Überblick über den Aufbau der drei Task, über das Programmlisting und das Symbollisting. Die ausführlichen Kommentare erklären den Programmaufbau.

6.2 Flussdiagramme der drei Task

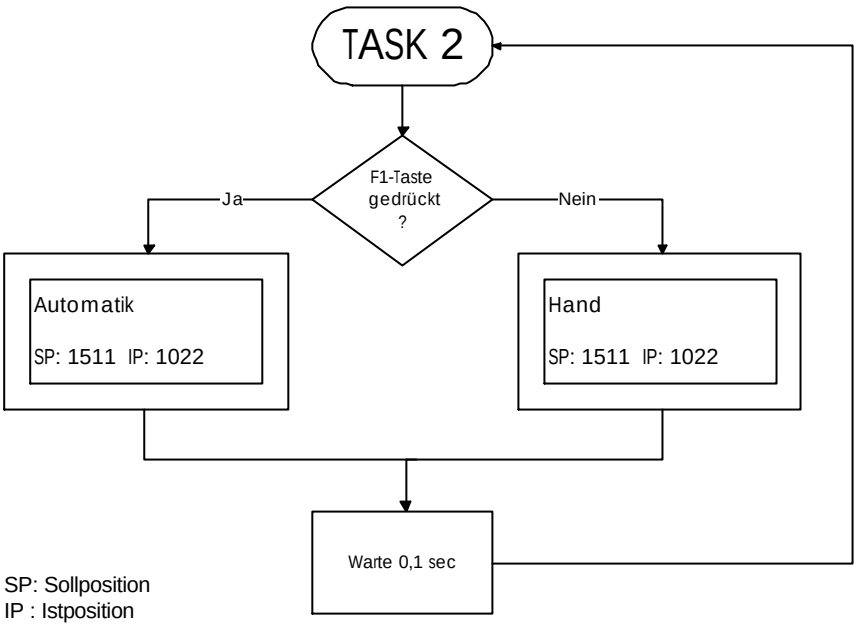
6.2.1 TASK 0 - Steuertask



6.2.2 TASK 1 - Automatiktask



6.2.3 TASK 2 - Anzeigetask



6.3 Programmlisting

```
NANO-B  -  Programmlisting  von  "DEMOPROG"  V1,  28.04.1996     8:21
Seite 1


JETTER PROZESS-SPS NANO-B

Kunde/Projekt: JETTER
Ort          : Ludwigsburg
Datum        : 28.04.1996   8:21
Version      : 1


 0: TASK tInitialisierung
 1:       ; ************************************
 2:       ;           TASK Initalisierung
 3:       ;
 4:       ; Initialisiert Steuerung, führt Refe-
 5:       ; renzfahrt aus und fragt die Funktions-
 6:       ; tasten ab.
 7:       ; ************************************
 8:       ;
 9:     DANN
10:       WARTEZEIT 2                          ;Warte 2/10 sec!
11:       KOPIERE [n=4, von rStartrampe nach rStart_Offset]
12:       ; -------------------------
13:       ; Festregister für Start- und Stoprampe,
```

```
14:       ; Zielfensterbereich und Offset werden
15:       ; beschrieben.
16:       ; -------------------------
17:         -MERKER mAutomatik                      ;Maschine im
Handbetrieb
18:        -MERKER mZyklusimGange                ;Kein Zyklus im
Gange
19:         -MERKER AutoLED                            ;Auto-LED
ausschalten!
20:         MERKER mHandLED                            ;Hand-LED
einschalten!
21:              LADE_REGISTER [rKommandoreg mit 3]      ;Setze
Referenzpunkt!
22:       A aRelais                            ;Schalte Relais ein!
23:                                                    ;Freigabe
Servoverstärker
24:         LADE_REGISTER [rZyklenzähler mit 1]   ;Zyklenzähler=1
setzen!
25:       ;
26:       ; *                 *                  *
27:       ; Zyklenzähler wird nach dem Einschalten
28:       ; "1" gesetzt, damit dann beim Start der
29:       ; Automatik die Maschine das erste Teil
30:       ; auf der 1. Ablageposition ablegt!
31:       ; *                 *                  *
32:       ;
33:       ; -------------------------
34:       ; -------------------------
35:       ;     Schleife Funktionstastenabfrage
36:       ; -------------------------
37:       ;
38: MARKE sFktabfrage
39:       ;
40:       ; -------------------------
41:       ; Abrage der Display-Tasten F1 und F7.
42:       ; (Automatik-/Handbetrieb umschalten)
43:       ; -------------------------
44:       ;
45:   FALLS
46:       MERKER mTaste_F1                    ;F1-Taste gedrückt?
47:     DANN
48:         MERKER mAutomatik                          ;Automatik
einschalten
49:       MERKER mAutoLED                     ;Auto-LED anzünden
50:       -MERKER mHandLED                    ;Hand-LED auslöschen
51:   FALLS
52:       MERKER mTaste_F7                    ;F7-Taste gedrückt?
53:     DANN
54:         -MERKER mAutomatik                         ;Automatik
ausschalten
55:       -MERKER mAutoLED                    ;Auto-LED auslöschen
56:       MERKER mHandLED                     ;Hand-LED anzünden
57:       ;
58:       ; -------------------------
59:       ; Abfrage Pfeiltasten für NC-Achsenbewe-
60:       ; gungen im Handbetrieb
 61:       ; ----------------------
 62:       ;
 63:   FALLS
 64:       ; ** Folgende drei Bedingungen sind    **
 65:       ; ** UND-verknüpft                     **
 66:       ;
 67:         MERKER mTasteZurück                    ;<=  - Taste
gedrückt?
```

```
 68:          -MERKER mAutomatik                        ;Automatik
ausgeschaltet?
 69:        -MERKER mZyklusimGange                ;Automatikzyklus
beendet?
 70:     DANN
 71:          UNTERPROGRAMM sHandzurück               ;* Handfahrt
rückwärts *
 72:   FALLS
 73:        ; ** Folgende drei Bedingungen sind    **
 74:        ; ** UND-verknüpft                     **
 75:        ;
 76:          MERKER mTasteVor                        ;=> - Taste
gedrückt
 77:          -MERKER mAutomatik                       ;Automatik
ausgeschaltet?
 78:        -MERKER mZyklusimGange                ;Automatikzyklus
beendet?
 79:     DANN
 80:          UNTERPROGRAMM sHandvorwärts             ;* Handfahrt
vorwärts *
 81:        ;
 82:        ; ------------------------
 83:        ; Abfrage der Taste F2 (Greifer öffnen
 84:        ; und schliessen
 85:        ; ------------------------
 86:        ;
 87:   FALLS
 88:        ; ** Folgende drei Bedingungen sind    **
 89:        ; ** UND-verknüpft                     **
 90:        ;
 91:        MERKER mTaste_F2                    ;F2-Taste gedrückt?
 92:           -MERKER mAutomatik                        ;Automatik
ausgeschaltet?
 93:        -MERKER mZyklusimGange                ;Automatikzyklus
beendet?
 94:     DANN
 95:        UNTERPROGRAMM sGreifer               ;* Greifer AUF bzw.
ZU *
 96:        ;
 97:        ; ------------------------
 98:        ;    Abfrage der Taste F8 (Teach-In)
 99:        ; ------------------------
100:        ;
101:   FALLS
102:        ; ** Folgende drei Bedingungen sind    **
103:        ; ** UND-verknüpft                     **
104:        ;
105:        MERKER mTaste_F8                    ;Taste_F8 gedrückt?
106:           -MERKER mAutomatik                        ;Automatik
ausgeschaltet?
107:        -MERKER mZyklusimGange                ;Automatikzyklus
beendet?
108:     DANN
109:          UNTERPROGRAMM sTeach_In                 ;* Teach-In
Ablagepos. *
110:        ;
111:      DANN                                    ;Funktionstasten
erneut
112:          SPRUNG sFktabfrage                       ;abfragen
(Schleifenende)
113:        ;
114:        ; *** Ende der Funktionstasteabfrage! ***
115:        ; ***       ( Schleifenende )         ***
116:        ; ------------------------
117:        ; ------------------------
118:        ;
```

```
119:       ;
120: TASK tAutomatikzyklus ------------------------
121:       ; ************************************
122:       ;         TASK Automatikzyklus
123:       ;
124:       ; Führt den Automatikzyklus aus, legt die
125:       ; gewünschten Teile abwechslungsweise auf
126:       ; den Ablagepositionen 1-3 ab.
127:       ; ************************************
128:       ;
129:   SOBALD
130:         MERKER mAutomatik                          ;Automatik
eingeschaltet?
131:       ; *                 *                   *
132:       ; Der Einschaltvorgang Automatik EIN/AUS
133:       ; ist im Task "Initialisierung" (0) pro-
134:       ; grammiert.
135:       ; *                 *                   *
136:     DANN
137:       ;
138:       ; ----------------------
139:       ;     Fahren in die Grundposition
140:       ; ----------------------
141:       ;
142:        MERKER mZyklusimGange                  ;Automatikzyklus
beginnt!
143:       ; *                 *                   *
144:       ; Dieser Merker wird am Schluss dieses
145:       ; Tasks am Ende des Automatikzyklus rück-
146:       ; gesetzt.
147:       ; Dadurch wird erreicht, dass, falls der
148:       ; Bediener während eines Automatikzyklus
149:       ; die Automatik auschaltet, dieser Zyklus
150:       ; zuerst noch beendet wird, bevor man
151:       ; Handfunktionen ausüben kann.
152:       ; (Dieser Merker wird im Task "Initiali-
153:       ; sierung" während der Funktionstastenab-
154:       ; frage mehrere Male abgefragt!)
155:       ; *                 *                   *
156:        -A aFahrevertikal                     ;Fahre vertikaler
Zylin-
157:        -A aGreiferAufZu                     ;der nach oben und
öffne
158:       LEER                                  ;den Greifer
159:       LEER                                  ;(Grundstellung!)
160:   SOBALD
161:        E eGrundposition                        ;Vert. Zylinder
oben
162:     DANN
163:       ; *                 *                   *
164:       ; Fahre die Achse mit Automatikgeschwin-
165:       ; digkeit in die Grundposition!
166:       ; *                 *                   *
167:       POS [Achse=21, Pos=R(rGrundpos), v=R(rGeschwAutomatik)]
168:    SOBALD                                     ;Horizontale Achse
in der
169:         HALTACHSE  Achse=21                          ;Grundpos.
angekommen?
170:     DANN
171:       ;
172:       ; ----------------------
173:       ;  Teil bei der Grundposition aufgreifen
174:       ; ----------------------
175:       ;
176:       ; *                 *                   *
```

```
177:       ; Fahre den vertikalen Zylinder nach
178:       ; unten!
179:       ; *                 *                 *
180:        A aFahrevertikal                        ;Zylinder nach
unten!
181:   SOBALD
182:        E eArbeitsposition                     ;Vert. Zylinder
unten?
183:     DANN
184:          A aGreiferAufZu                              ;Greifer
schliessen!
185:       WARTEZEIT 5                           ;Warte 0.5 Sekunden
186:        -A aFahrevertikal                     ;Fahre vertikale
Achse
187:       LEER                                  ;nach oben!
188:   SOBALD
189:        E eGrundposition                      ;Vert. Zylinder
oben?
190:     DANN
191:       ;
192:       ; ------------------------
193:       ; Je nach momentanen Wert des Registers
194:       ; "Zyklenzähler" zur entsprechenden Ab-
195:       ; lageposition fahren
196:       ; ------------------------
197:       ;
198:       ; *                 *                 *
199:       ; Anhand der Ablagepositionsnummer (In-
200:       ; halt des Registers "Zyklenzähler" wird
201:       ; die Registernummer berechnet, in dem
202:       ; der Sollwert dieser Position abgespei-
203:       ; chert ist.
204:       ; *                 *                 *
205:       ;
206:        REG rPositionsreg                     ;Berechnung der
Register-
207:       =                                      ;nummer mit der
gespei-
208:         REG  rZyklenzähler                           ;cherten
Sollposition der
209:         +                                            ;entspr.
Ablageposition!
210:       zRegoffset_1
211:       ;
212:       ; *                 *                 *
213:       ; Fahre die horizontalen Achse zur Abla-
214:       ; position mit der Nummer, die im Regis-
215:       ; ter "Zyklenzähler" steht.
216:       ; *                 *                 *
217:       ;
218:                      POS   [Achse=21,   Pos=RR(rPositionsreg),
v=R(rGeschwAutomatik)]
219:   SOBALD
220:        HALTACHSE Achse=21                     ;Achse bei der
Ablagepo-
221:       LEER                                  ;sition angekommen?
222:     DANN
223:       ;
224:       ; ------------------------
225:       ; Teil bei der momentan aktuellen Ablage-
226:       ; postition ablegen.
227:       ; ------------------------
228:       ;
229:       ; *                 *                 *
230:       ; Fahre den vertikalen Zylinder nach
231:       ; unten!
```

```
232:       ; *                  *                   *
233:         A aFahrevertikal                        ;Zylinder nach
unten!
234:   SOBALD
235:        E eArbeitsposition                      ;Vert. Zylinder
unten?
236:     DANN
237:       -A aGreiferAufZu                       ;Greifer öffnen!
238:        WARTEZEIT 5                                  ;Warte 0.5
Sekunden!
239:         -A aFahrevertikal                          ;Fahre den
vertikalen
240:        LEER                                     ;Zylinder nach
oben!
241:   SOBALD
242:        E eGrundposition                        ;Vert. Zylinder
oben?
243:     DANN
244:       ;
245:       ; ----------------------
246:       ;    In die Grundposition zurückfahren
247:       ; ----------------------
248:       ;
249:       ; *                   *                  *
250:       ; Fahre mit der horizontalen Achse in die
251:       ; Grundposition zurück
252:       ; *                   *                  *
253:       ;
254:       POS [Achse=21, Pos=R(rGrundpos), v=R(rGeschwAutomatik)]
255:   SOBALD
256:        HALTACHSE Achse=21                      ;Achse bei der
Grundpo-
257:       LEER                                   ;sition angekommen?
258:     DANN
259:       ;
260:       ; ----------------------
261:       ;      Nächster Zyklus vorbereiten
262:       ; ----------------------
263:       ;
264:       ; *                   *                  *
265:       ; Die Abfolge der drei Ablagepositionen,
266:       ; welche beim Automatikablauf eingehal-
267:       ; ten wird, lautet: 1-2-3-1-2-3-1-....
268:       ; Die nachfolgenden Befehle sorgen da-
269:       ; für, dass der Wert des Registers "Zyk-
270:       ; lenzähler" diese Abfolge aufweist.
271:       ; *                   *                  *
272:       ;
273:   FALLS
274:       REG rZyklenzähler                       ;Es war noch nicht
die
275:       <                                       ;dritte und letzte
Ab-
276:        3                                        ;lageposition an
der
277:       LEER                                   ;Reihe?
278:     DANN
279:        REGINC rZyklenzähler                    ;Zyklenzähler um 1
er-
280:       LEER                                   ;höhen!
281:     SONST
282:         LADE_REGISTER [rZyklenzähler mit 1]   ;Wieder von vorn
begin-
283:       LEER                                   ;nen!
284:     DANN
```

```
285:       ;
286:       ; -----------------------
287:       ;          Z y k l u s e n d e
288:       ; -----------------------
289:       -MERKER mZyklusimGange
290:       ; *                 *                  *
291:       ; Über die Bedeutung dieses Merkers in-
292:       ; formieren Sie sich bitte am Anfang
293:       ; dieses Tasts (Task "Automatikzyklus")
294:       ; *                 *                  *
295:       SPRUNG tAutomatikzyklus              ;Beginne Zyklus von
vorn
296:       ;
297:       ;
298: TASK tAnzeige -----------------------------
299:       ; ************************************
300:       ;             TASK Anzeige
301:       ;
302:       ; Zeigt am Display an, ob Automatik-
303:       ; oer Handbetrieb im Gange ist und gibt
304:       ; zusätzlich die Soll- und Istposition
305:       ; an.
306:       ; ************************************
307:       ;
308:   FALLS
309:          MERKER mAutomatik                         ;Automatik
angewählt?
310:     DANN
311:       ;
312:       ; -----------------------
313:       ; Falls Automatik angewählt ist, wird
314:       ; am Display oben links "AUTOMATIK" aus-
315:       ; ausgegeben.
316:       ; $ = Rest der ersten Zeile löschen!
317:       ; -----------------------
318:       ;
319:       ANZEIGE_TEXT [#0, cp=1, "AUTOMATIK$ "]
320:     SONST
321:       ;
322:       ; -----------------------
323:       ; Falls Handbetrieb angewählt ist, wird
324:       ; am Display oben links "HAND" ausgege-
325:       ; ben.
326:       ; $ = Rest der ersten Zeile löschen!
327:       ; -----------------------
328:       ;
329:       ANZEIGE_TEXT [#0, cp=1, "HAND$ "]
330:     DANN
331:       ;
332:       ; -----------------------
333:       ; In beiden Fällen wird auf der zweiten
334:       ; Zeile des Displays die jeweilige Soll-
335:       ; und Istposition ausgegeben.
336:       ; -----------------------
337:       ;
338:       ANZEIGE_TEXT [#0, cp=25, "SP: "]
339:       ANZEIGE_REG [#0, cp=28, Reg=rSollposition]
340:       ANZEIGE_TEXT [#0, cp=37, "IP: "]
341:       ANZEIGE_REG [#0, cp=41, Reg=rIstposition]
342:       ;
343:       ; -----------------------
344:       ; Im weiteren ist hier noch eine Warte-
345:       ; zeit von 0.1 Sekunden eingefügt. Ohne
346:       ; diese Wartezeit würde dieses Task, wel-
347:       ; ches ja immerzu das Display beschreibt,
348:       ; zuviel CPU-Time (Rechnerzeit) benötigen
```

```
349:       ; und so gegenüber den anderen beiden
350:       ; Tasks eine zu grosse Priorität erhal-
351:       ; ten.
352:       ; ----------------------
353:       ;
354:       WARTEZEIT 1
355:       SPRUNG tAnzeige
356:       ;
357:       ;
358:       ; ************************************
359:       ;      U N T E R P R O G R A M M E
360:       ; ************************************
361:       ;
362: MARKE sHandzurück
363:       ; ----------------------
364:       ;         UNTERPROGRAMM Handzurück
365:       ;
366:       ; Fährt im Handbetrieb mit der horizon-
367:       ; talen Achse zurück, bis die Taste <=
368:       ; wieder losgelassen wird.
369:       ; ----------------------
370:       ;
371:     DANN
372:       ;
373:       ; *                 *                 *
374:       ; Fahre mit Handgeschwindigkeit rück-
375:       ; wärts!
376:       ; *                 *                 *
377:       ;
378:       POS [Achse=21, Pos=zRückwärts, v=R(rGeschwHand)]
379:   SOBALD
380:          -MERKER  TasteZurück                        ;Taste <=
losgelassen?
381:     DANN
382:       HALTACHSE Achse=21                  ;Achse anhalten!
383:       RÜCKSPRUNG
384:       ;
385:       ;
386: MARKE sHandvorwärts
387:       ; ----------------------
388:       ;        UNTERPROGRAMM Handvorwärts
389:       ;
390:       ; Fährt im Handbetrieb mit der horizon-
391:       ; talen Achse vorwärts, bis die Taste =>
392:       ; wieder losgelassen wird.
393:       ; ----------------------
394:       ;
395:     DANN
396:       ;
397:       ; *                 *                 *
398:       ; Fahre mit Handgeschwindigkeit vor-
399:       ; wärts!
400:       ; *                 *                 *
401:       ;
402:       POS [Achse=21, Pos=zVorwärts, v=R(rGeschwHand)]
403:   SOBALD
404:          -MERKER mTasteVor                          ;Taste <=
losgelassen?
405:     DANN
406:       HALTACHSE Achse=21                  ;Achse anhalten!
407:       RÜCKSPRUNG
408:       ;
409:       ;
410: MARKE sGreifer
411:       ; ----------------------
```

```
412:       ;        UNTERPROGRAMM Greifer
413:       ;
414:       ; Falls der Greifer offen ist, schliesst
415:       ; ihn dieses Unterprogramm und umgekehrt.
416:       ; ----------------------
417:       ;
418:       ; *                *                 *
419:       ; Da bei dieser Handroutine am Display
420:       ; ausgegeben wird, ob der Greifer ge-
421:       ; schlossen oder geöffnet wird, muss in
422:       ; beiden Fällen das Task "Anzeige" un-
423:       ; terbrochen werden, da sonst das Dis-
424:       ; play stets mit dem im Task "Anzeige"
425:       ; ausgegebenen Text überschrieben würde.
426:       ; *                *                 *
427:       ;
428:   FALLS
429:         A aGreiferAufZu                            ;Greifer
geschlossen?
430:     DANN
431:          TASKBREAK #tAnzeige                       ;Anzeige
unterbrechen
432:       ANZEIGE_TEXT [#0, cp=1, "_ Greifer öffnen"]
433:       -A aGreiferAufZu                   ;Greifer öffnen
434:     SONST
435:          TASKBREAK #tAnzeige                       ;Anzeige
unterbrechen
436:       ANZEIGE_TEXT [#0, cp=1, "_ Greifer schliessen"]
437:       A aGreiferAufZu                    ;Greifer schliessen
438:   SOBALD
439:          -MERKER mTaste_F2                        ;Taste F2
losgelassen?
440:     DANN
441:       ANZEIGE_TEXT [#0, cp=1, "_ "]
442:          TASKCONTINUE #tAnzeige                    ;Anzeige
aktivieren!
443:       RÜCKSPRUNG
444:       ;
445:       ;
446: MARKE sTeach_In
447:       ; ----------------------
448:       ;        UNTERPROGRAMM Teach_In
449:       ;
450:       ; Mit dieser Routine können durch den Be-
451:       ; diener die drei Ablagepositionen sowie
452:       ; die Grundposition festgelegt werden.
453:       ; ----------------------
454:       ;
455:       ; *                *                 *
456:       ; Der Bediener legt die Grundposition so-
457:       ; wie die drei Ablagepositionen fest, in-
458:       ; dem er von Hand an die gewünschte Posi-
459:       ; tion fährt und dann über das Display
460:       ; eingibt, um welche Position es sich da-
461:       ; bei handelt.
462:       ; Da bei diesem Unterprogramm ebenfalls
463:       ; die Kommunikation über das Display er-
464:       ; forderlich ist, muss auch hier während
465:       ; dieser Zeit das Task "Anzeige" unter-
466:       ; brochen werden.
467:       ; *                *                 *
468:       ;
469:     DANN
470:          TASKBREAK #tAnzeige                       ;Anzeige
unterbrechen!
471:       ;
```

```
472:       ; *                  *                  *
473:       ; Bediener wird aufgefordert, die Posi-
474:       ; tionsnummer festzulegen und zwar nach
475:       ; folgendem Schlüssel:
476:       ; 1 = Grundposition
477:       ; 2 = 1. Ablageposition
478:       ; 3 = 2. Ablageposition
479:       ; 4 = 3. Ablageposition
480:       ; *                  *                  *
481:       ;
482:       ANZEIGE_TEXT [#0, cp=1, "Eingabe Positionsnr.(1-4"]
483:       ANZEIGE_TEXT [#0, cp=25, "1=Grundst.)$ "]
484:       BEDIENEREINGABE [#0, cp=40, Reg=rArbeitsregister]
485:       ;
486:       ; *                  *                  *
487:       ; Gültigkeit der durch den Bediener ein-
488:       ; gegebenen Positionsnummer wird über-
489:       ; prüft (liegt sie zwischen 1 und 4?)
490:       ; *                  *                  *
491:       ;
492:    FALLS
493:       GRENZEN [Reg=rArbeitsregister, unten=1, oben=4]
494:      DANN
495:       ;
496:       ; *                  *                  *
497:       ; Nun wird anhand der eingegebenen Posi-
498:       ; tionsnummer die Registernummer berech-
499:       ; net, in deren Register die aktuelle
500:       ; Istposition abgespeichert werden soll.
501:       ; *                  *                  *
502:       ;
503:        REG rArbeitsregister                   ;Berechnung der
Regis-
504:       =                                    ;ternummer
505:       REG rArbeitsregister
506:       +
507:       zRegOffset_2
508:       ;
509:       ; *                  *                  *
510:       ; Dann wird die Istposition in das vor-
511:       ; hin berechnete Register abgespeichert.
512:       ; Dann wird dem Bediener per Display an-
513:       ; gezeigt, dass die Eingabe korrekt war.
514:       ; Nach einer 0.5 Sekunden langen Pause
515:       ; wird das Task "Anzeige" wieder akti-
516:       ; viert und ins Programm zurückgesprun-
517:       ; gen.
518:       ; *                  *                  *
519:       ;
520:       LADE_REGISTER [R(rArbeitsregister) mit R(rIstposition)]
521:       ANZEIGE_TEXT [#0, cp=1, "_ok! "]
522:       WARTEZEIT 5                            ;Warte 0.5 Sekunde
523:       TASKCONTINUE #tAnzeige                 ;Anzeige aktivieren
524:       RÜCKSPRUNG
525:      SONST
526:       ;
527:       ; *                   *                 *
528:       ;        F e h l e r m e l d u n g !
529:       ;
530:       ; Am Display wird die fehlerhafte Eingabe
531:       ; 1 Sekunde lang signalisiert, dann wird
532:       ; der Bediener erneut abgefragt.
533:       ; *                   *                 *
534:       ;
535:       ANZEIGE_TEXT [#0, cp=1, "_Unzulässige Pos.-Nr.,"]
```

```
536:       ANZEIGE_TEXT [#0, cp=25, "bitte wiederholen !"]
537:       WARTEZEIT 10                          ;Warte 1 Sekunde
538:     DANN
539:       SPRUNG sTeach-In                      ;Erneute Abfrage
540:       ;
541:       ;
542:       ; ************************************
543:       ;   E N D E   D E S   P R O G R A M M S
544:       ; ************************************
545:       ; ************************************
Programmende
```

Folgende Register sind vor dem Start des Programmes im Inbetriebnahme-Bildschirm zu initialisieren:

- Register 100 (Startrampe) mit 10
- Register 101 (Stoprampe) mit 10
- Register 102 (Zielfensterbereich) mit 0
- Register 103 (Digitaler Offset) mit 32
- Register 110 (Geschwindigkeit Automatikbetr.) mit 10000
- Register 111 (Geschwindigkeit im Handbetrieb) mit 1000

6.4 Symbollisting

Das entsprechende Symbollisting sieht folgendermaßen aus:

```
NANO-B  Symbollisting von "DEMOPROG" V1, 28.04.1996   8:21    Seite 1


JETTER PROZESS-SPS NANO-B

Kunde/Projekt: JETTER
Ort          : Ludwigsburg
Datum        : 28.04.1996   8:21
Version      : 1


*****************************************************************************
                    S Y M B O L E D I T O R
*****************************************************************************

----------------------------------------
                                 T A S K S
----------------------------------------

tInitialisierung       0      Initialisiert Steuerung, führt Referenz-
                              fahrt aus und fragt die Funktionstasten ab.
tAutomatikzyklus       1      Führt den Automatikzyklus aus, legt die
                              gewünschten Teile abwechslungsweise auf den
                              Ablagepositionen 1-3 ab.
tAnzeige               2      Zeigt am Display an, ob Automatik- oder
                              Handbetrieb im Gange ist und gibt zudem die
                              Soll- und die Istposition an.

----------------------------------------
----------------------------------------
                      U N T E R P R O G R A M M E
----------------------------------------

sHandzurück          200      Fährt im Handbetrieb mit der horizontalen Achse
                              zurück, bis die Taste <= losgelassen wird.
sHandvorwärts        201      Fährt im Handbetrieb mit der horizontalen Achse
                              vorwärts, bis die Taste => losgelassen wird.
sGreifer             202      Falls der Greifer offen ist, schliesst ihn dieses
                              Unterprogramm und umgekehrt.
sTeach_In            203      Mit dieser Routine können durch den Bediener die
                              drei Ablagepositionen sowie die Grundposition
                              festgelegt werden.

----------------------------------------
----------------------------------------
                         S P R U N G M A R K E N
----------------------------------------

sFktabfrage           40      Auf diese Marke wird jeweils am Schluss des Tasks
                              "Initialisierung" zurückgesprungen, um die Funk-
                              tionstasten erneut abzufragen.

----------------------------------------


----------------------------------------
                            E I N G Ä N G E
----------------------------------------

eGrundposition        108     Ist aktiv, falls sich die vertikale Achse ganz
                              oben befindet (Grundposition)
eArbeitsposition      107     Ist aktiv, falls sich die vertikale Achse ganz
```

```
                              unten befindet (Arbeitsposition!)
---------------------------------------
---------------------------------------
                            A U S G Ä N G E
---------------------------------------

aFahrevertikal       107      Durch Setzen dieses Ausganges fährt der Hydrau-
                              likzylinder nach unten, durch Rücksetzen bewegt
                              er sich wieder nach oben
aGreiferAufZu        102      Wird dieser Ausgang gesetzt, schliesst der Grei-
                              fer, durch Rücksetzen öffnet er sich wieder
aRelais              101      Durch Setzen dieses Ausganges wird beim Demo-
                              koffer das Relais eingeschalten.

---------------------------------------
---------------------------------------
                            R E G I S T E R
---------------------------------------

rStartrampe          100      Speicherplatz für die Startrampe
rStoprampe           101      Speicherplatz für die Stoprampe
rZielfenster         102      Speicherplatz für das Zielfenster
rDigitalOffset       103      Speicherplatz für den digitalen Offset
rGeschwAutomatik     110      Enthält den Wert der Geschwindigkeit im Automatik-
                              Betrieb
rGeschwHand          111      Enthält den Wert der Geschwindigkeit im Handbe-
                              trieb
rGrundpos            120      Positionswert der Grundstellung (Teach-In!)
rArbeitspos_1        121      Positionswert der 1. Ablagestelle (Teach-In!)
rArbeitspos_2        122      Positionswert der 2. Ablagestelle (Teach-In!)
rArbeitspos_3        123      Positionswert der 3. Ablagestelle (Teach-In!)
rZyklenzähler        130      Zyklenzähler (Ablagestelle 1-2-3-1-2-3-1.....)
rPositionsreg        131      Hier wird vor der Positionierung die Registernum-
                              mer, der der jeweilige Postionswert abgespeichert
                              ist, bestimmt.
rArbeitsregister     200      In dieses Register gibt der Bediener beim Teach-
                              In die Positionsnummer (1-4, 1=Grundpos.)

---------------------------------------
---------------------------------------
                        F E S T R E G I S T E R
---------------------------------------

rKommandoreg        12101     Kommandoregister (mit dem Wert 3 wird im Pro-
                              gramm der Referenzpunkt gesetzt.
rSollposition       12102     Enthält den Wert der jeweiligen Sollposition.
                             (wird im Task "Anzeige" benötigt)
rStart_Offset       12105     Diese Registernummer wird dazu verwendet um die
                              in Register gespeicherten Werte der Start- und
                              Stoprampe, des Zielfensters und des digitalen
                              Offsets in die entsprechende Festregister zu
                              laden.
rIstposition        12109     Enthält den Wert der jeweiligen Istpostion
                              (wird im Unterprogramm Teach_In benötigt).

---------------------------------------
---------------------------------------
                             M E R K E R
---------------------------------------

mAutomatik            1       Gesetzt, falls Automatik angewählt wird.
mZyklusimGange        2       Gesetzt, solange ein Automatikzyklus noch im
                              Gange ist.

---------------------------------------
          M E R K E R  F Ü R  D I E  T A S T E N A B F R A G E
                   A U F  D E M  D I S P L A Y
---------------------------------------


mTaste_F1           2201      Merker zur Abfrage der Taste F1 (Automatik ein)
mTaste_F2           2202      Merker zur Abfrage der Taste F2 (Greifer auf/zu)
mTaste_F7           2207      Merker zur Abfrage der Taste F7 (Automatik aus)
mTaste_F8           2208      Merker zur Abfrage der Taste F8 (Teach-In)
mTasteZurück        2114      Merker zur Abfrage der Taste "Zurück"
mTasteVor           2113      Merker zur Abfrage der Taste "Vorwärts"
mAutoLED            2024      Merker zur Betätigung der LED von Taste F1
```

```
mHandLED             2023     Merker zur Betätigung der LED von Taste F7

---------------------------------------
                              Z A H L E N
---------------------------------------

zRegoffset_1         120      Differenzwert, um vom Wert des Zykluszählers
                              (Register 130) zur Registernummer zu gelangen,
                              in welchem der Wert der entsprechenden Ablage-
                              postition abgespeichert ist.
zRegoffset_2         119      Differenzwert, um vom Wert der durch den Bedie-
                              ner eingegebenen Positionsnummer zur Register-
                              nummer zu gelangen, in welchem der Wert der im
                              Handbetrieb eingestellten Position abgespeichert
                              wird.
zRückwärts       -500000      Sollposition, die verwendet wird, um im Hand-
                              betrieb mit der Taste <= rückwärts zu fahren.
                              (siehe Unterprogramm "Handzurück")
zVorwärts         500000      Sollposition, die verwendet wird, um im Hand-
                              betrieb mit der Taste => vorwärts zu fahren.
                              (siehe Unterprogramm "Handvorwärts).

 ---------------------------------------
```

Stichwortverzeichnis