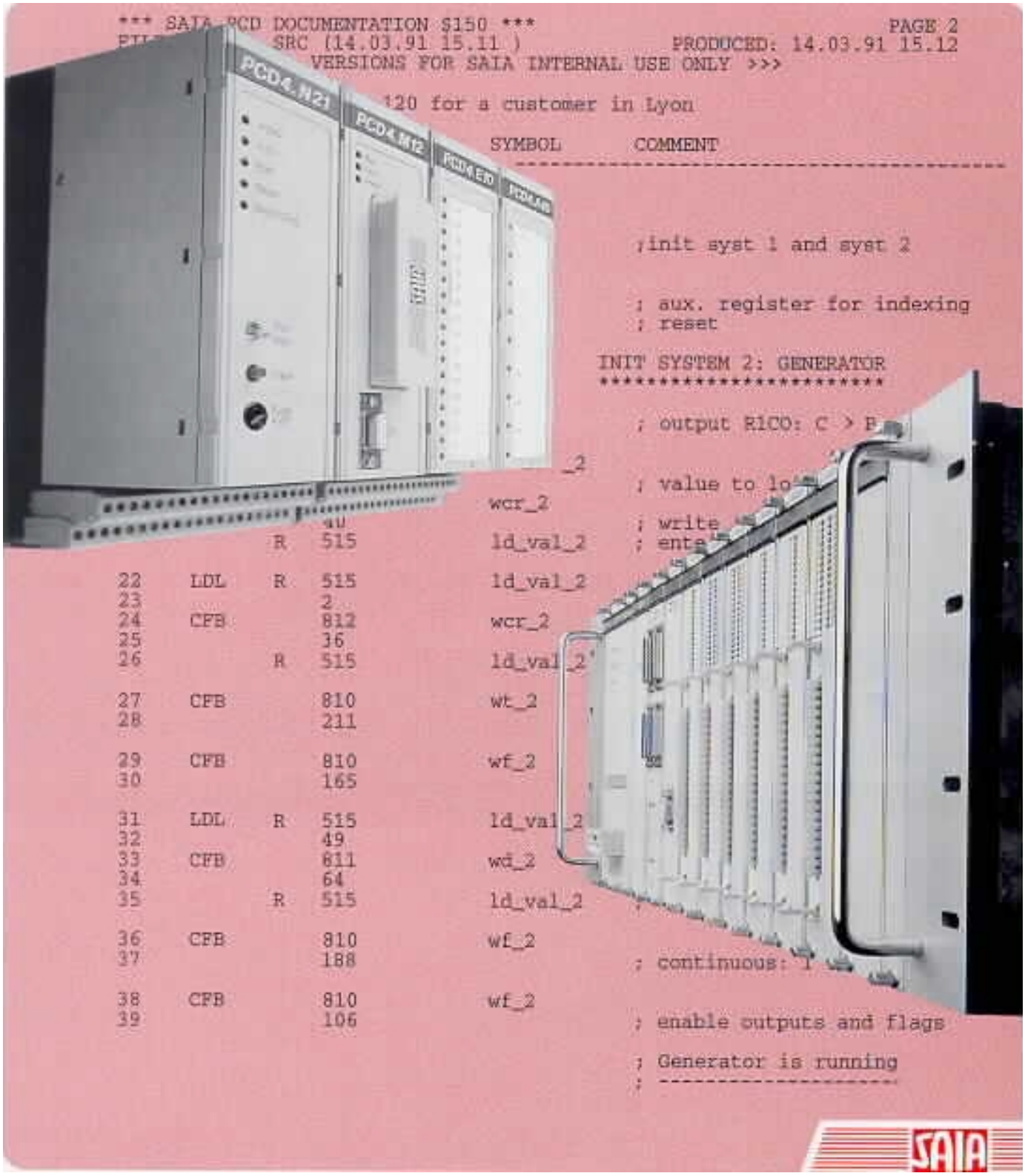
SAIA-Burgess Electronics

SWITCHES - MOTORS - CONTROLLERS

SAIA® PCD

Process Control Devices

Befehlssatz für die PCD-Familie



Deutsche Ausgabe 26/733 D4

Saia-Burgess Controls AG
Bahnhofstrasse 18
CH-3280 Murten (Schweiz)
http;//www.saia-burgess.com

Telefon 026 / 672 72 72
Telefax 026 / 672 74 99

SAIA-Burgess Gesellschaften

Schweiz
Saia-Burgess Controls AG
Bahnhofstrasse 18
CH-3280 Murten
☎ 026 672 72 72, Fax 026 672 74 99

Deutschland
SAIA-Burgess Electronics GmbH & Co. KG
Otto-Hahn-Strasse 31 - 33
D-63303 Dreieich
☎ 06103 89 060, Fax 06103 89 06 66

Österreich
SAIA-Burgess Electronics Ges.m.b.H.
Schallmooser Hauptstrasse 38
A-5020 Salzburg
☎ 0662 88 49 10, Fax 0662 88 49 10 11

Italien
SAIA-Burgess Electronics S.r.l.
Via Cadamosto 3
I-20094 Corsico MI
☎ 02 48 69 21, Fax 02 48 60 06 92

Frankreich
SAIA-Burgess Electronics Sàrl.
10, Bld. Louise Michel
F-92230 Gennevilliers
☎ 01 46 88 07 70, Fax 01 46 88 07 99

Niederlande
SAIA-Burgess Electronics B.V.
Hanzeweg 12c
NL-2803 MC Gouda
☎ 0182 54 31 54, Fax 0182 54 31 51

Belgien
SAIA-Burgess Electronics Belgium
Avenue Roi Albert 1er, 50
B-1780 Wemmel
☎ 02 456 06 20, Fax 02 460 50 44

Ungarn
SAIA-Burgess Electronics Automation Kft.
Liget utca 1.
H-2040 Budaörs
☎ 23 501 170, Fax 23 501 180

Vertretungen

Gross-britannien
Canham Controls Ltd.
25 Fenlake Business Centre, Fengate
Peterborough PE1 5BQ UK
☎ 01733 89 44 89, Fax 01733 89 44 88

Dänemark
Malthe Winje Automation AS
Håndværkerbyen 57 B
DK-2670 Greve
☎ 70 20 52 01, Fax 70 20 52 02

Norwegen
Malthe Winje Automasjon AS
Haukelivn 48
N-1415 Oppegård
☎ 66 99 61 00, Fax 66 99 61 01

Schweden
Malthe Winje Automation AB
Truckvägen 14A
S-194 52 Upplands Våsby
☎ 08 795 59 10, Fax 08 795 59 20

Suomi/ Finnland
ENERGEL OY
Atomitie 1
FIN-00370 Helsinki
☎ 09 586 2066, Fax 09 586 2046

Portugal
INFOCONTROL Electronica e Automatismo LDA.
Praceta Cesário Verde, No 10 s/cv, Massamá
P-2745 Queluz
☎ 21 430 08 24, Fax 21 430 08 04

Spanien
Tecnosistemas Medioambientales, S.L.
Poligono Industrial El Cabril, 9
E-28864 Ajalvir, Madrid
☎ 91 884 47 93, Fax 91 884 40 72

Tschechische Republik
ICS Industrie Control Service, s.r.o.
Modranská 43
CZ-14700 Praha 4
☎ 2 44 06 22 79, Fax 2 44 46 08 57

Polen
SABUR Ltd.
ul. Druzynowa 3A
PL-02-590 Warszawa
☎ 22 844 63 70, Fax 22 844 75 20

Argentinien
MURTEN S.r.l.
Av. del Libertador 184, 4° “A”
RA-1001 Buenos Aires
☎ 054 11 4312 0172, Fax 054 11 4312 0172

Kundendienst

USA
SAIA-Burgess Electronics Inc.
1335 Barclay Boulevard
Buffalo Grove, IL 60089, USA
☎ 847 215 96 00, Fax 847 215 96 06

Ausgabe: 12.01.2001

Änderungen vorbehalten

SAIA® Programmable Control Devices

Handbuch

Befehlssatz
für die PCD-Familie

Saia-Burgess Controls AG

Ausgabe 26/733 D4 - 02.98 Letzte Anpassung - 10.2002

Technische Änderungen vorbehalten

Anpassungen

Handbuch: SAIA® PCD Befehlssatz - Ausgabe D4

Datum	Abschnitt	Seite	Beschreibung
12.04.2000	6	6-5 .. 6-8	XOB: diverse Korr. und XOB 6 nachgetragen
12.04.2000	6	6-18 .. 6-19	SCOB: alt und neu
12.04.2000	8	8-18 .. 8-19	SASI: $ wird akzeptiert
12.04.2000	8	8-48	SOCL: Umschaltung RS 485 und RS 422
12.04.2000	12	12-14 .. 12-17	SYSRD: div. Korr., lesen der Datum-Uhr
12.04.2000	12	12-18 .. 12-22	SYSWR: div. Korr., schreiben der Datum-Uhr
06.10.2000	12	12-21	SYSWR : Code 6000 (Schreiben ins EEPROM)
08.03.2001	3	3-8	DEC : Diagramm DEC-Register korrigiert.
29.10.2002	8	8-3	MC5 ergänzt.

Inhalt

Seite

3. WORT-Befehle

Seite

4. GANZZAHL ARTIHMETIK (Integer Arithmetic)

5. FLIESSPUNKT-ARITHMETIK

Page

6. BLOCTEC-Instruktionen

7. GRAFTEC Instruktionen

Page

8. Befehle für die SERIELLE KOMMUNIKATION

Seite

9. LAN2-Befehle

10. Befehle zur PROGRAMMSTEUERUNG

11. DEFINITIONS-Befehle

12. SPEZIAL-Befehle

13. History-Liste

Befehlsliste (alphabetisch geordnet)

Befehl	Inx	Pm	Seite
ACC			2-11
ADD		=	4-3
ALGI	X	=	12-24
ALGO	X	=	12-25
AND	X	=	3-31
ANH	X	=	2-5
ANL	X	=	2-6
BITI	X	=	3-23
BITIR	X	=	3-24
BITO	X	=	3-25
BITOR	X	=	3-26
CCOB			6-20
CFB			6-16
CMP	X	=	4-8
COB			6-3
COM	X	=	2-16
COPY	X	=	3-15
CPB			6-12
CPBI			6-13
CSB			7-5
DEC	X	=	3-8
DEFTB			11-5
DEFTC			11-4
DEFTR			11-6
DEFVM			11-3
DEFWPH			11-8
DEFWPR			11-7
DEI		=	3-11
DIAG			12-13
DIGI	X	=	3-27
DIGIR	X	=	3-28
DIGO	X	=	3-29
DIGOR	X	=	3-30
DIV		=	4-6
DSP			3-6
DYN	X	=	2-12
ECOB			6-4
EFB			6-15
EPB			6-11
ESB			7-4
EST			7-9
ETR			7-11
EXOB			6-9
EXOR	X	=	3-33
FABS	X	=	5-16
FADD			5-5
FATAN			5-13
FB		=	6-14
FCMP	X		5-10
FCOS		=	5-12
FDIV		=	5-8
FEXP		=	5-14
FNL		=	5-15
FMUL		=	5-7

Befehl	Inx	Pm	Seite
FPI	X	=	5-4
FSIN	X	=	5-11
FSQR		=	5-9
FSUB		=	5-6
GET	X	=	3-16
HALT			10-6
IFP	X	=	5-3
INC	X	=	3-7
INI		=	3-10
IST			7-7
JPD			10-4
JPI			10-5
JR			10-3
LD	X		3-3
LDH	X	=	3-5
LDL	X	=	3-4
LOCK			10-7
LRXD	X	=	9-3
LRXS	X	=	9-5
LTXD	X	=	9-4
LTXS	X	=	9-6
MOV	X	=	3-14
MUL		=	4-5
NCOB			6-17
NOP			12-3
NOT	X	=	3-34
OR	X	=	3-32
ORH	X	=	2-7
ORL	X	=	2-9
OUT	X	=	2-13
OUTS	X	=	12-27
PB			6-10
PID			12-6
PUT	X		3-19
RCOB			6-21
RES	X	=	2-15
RESD	X		2-18
ROTD		=	3-38
ROTL	X	=	3-41
ROTR	X	=	3-42
ROTU		=	3-37
RSB			7-6
RSI		=	3-13
RTIME		=	12-4
SASI		=	8-8
SASII		=	8-20
SB			7-3
SCOB			6-18
SCON		=	8-49
SCONI		=	8-51
SEI		=	3-9
SET	X	=	2-14
SETD	X		2-17
SHID		=	3-36

Befehl	Inx	Pm	Seite
SHIL	X	=	3-39
SHIR	X	=	3-40
SHIU		=	3-35
SICL		=	8-46
SOCL		=	8-47
SQR		=	4-7
SRXD	X	=	8-21
SRXM		=	8-32
SRXMI		=	8-37
ST			7-8
STH	X	=	2-3
STHS	X	=	12-26
STI		=	3-12
STL	X	=	2-4
STXD	X	=	8-22
STXM		=	8-39
STXMI		=	8-44
STXT	X	=	8-23
SUB		=	4-4
SYSCMP		=	12-23
SYSRD		=	12-14
SYSWR		=	12-18
TEST			12-10
TR			7-10
TFR	X	=	3-21
TFRI	X	=	3-22a
UNLOCK			10-8
WTIME		=	12-5
XOB			6-5
XOR	X	=	2-10

Legende:

Kolonne "Inx" (Index) zeigt an. dass dieser Befehl indexiert werden kann.

Kolonne "Pm" (Parameter) zeigt an, dass dieser Befehl FB-Parameter enthalten kann.

Notizen

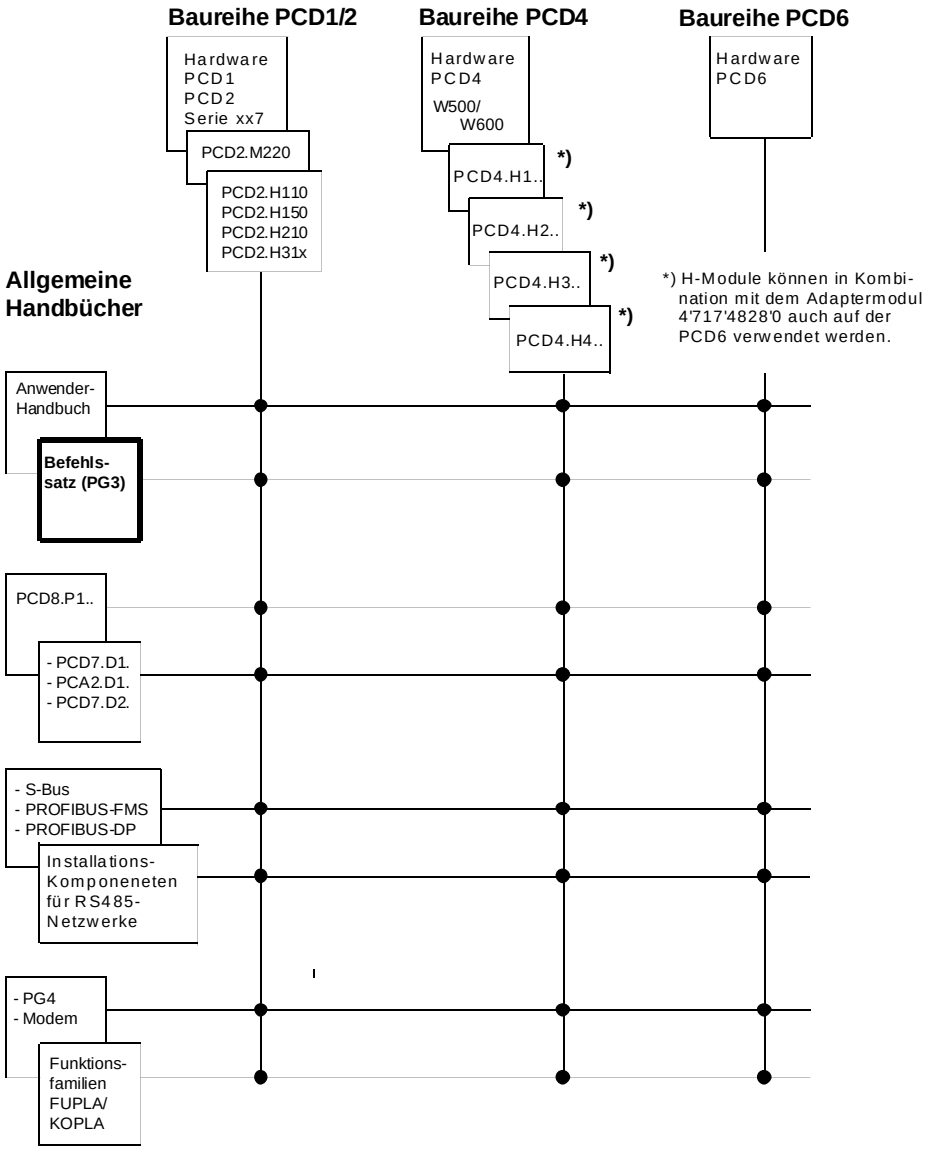
Wichtiger Hinweis:

Um den einwandfreien Betrieb von SAIA® PCD sicherstellen zu können, wurde eine Vielzahl detaillierter Handbücher geschaffen. Diese wenden sich an technisch qualifiziertes Personal, das nach Möglichkeit auch unsere Workshops erfolgreich absolviert hat.

Die vielfältigen Leistungen der SAIA® PCD treten nur dann optimal in Erscheinung, wenn alle in diesen Handbüchern aufgeführten Angaben und Richtlinien bezüglich Montage, Verkabelung, Programmierung und Inbetriebnahme genau befolgt werden.

Damit allerdings werden Sie zum grossen Kreis der begeisterten SAIA® PCD Anwendern gehören.

Übersicht



Zuverlässigkeit und Sicherheit elektronischer Steuerungen

Die Firma SAIA-Burgess Electronics AG konzipiert, entwickelt und stellt ihre Produkte mit aller Sorgfalt her:

- Neuster Stand der Technik
- Einhaltung der Normen
- Zertifiziert nach ISO 9001
- Internationale Approbationen: z.B. Germanischer Lloyd, UL, Det Norske Veritas, CE-Zeichen ...
- Auswahl qualitativ hochwertiger Bauelemente
- Kontrollen in verschiedenen Stufen der Fertigung
- In-Circuit-Tests
- Run-in (Wärmelauf bei 85°C während 48h)

Die daraus resultierende hochstehende Qualität zeigt trotz aller Sorgfalt Grenzen. So ist z.B. mit natürlichen Ausfällen von Bauelementen zu rechnen. Für diese gibt die Firma SAIA-Burgess Electronics AG Garantie gemäss den "Allgemeinen Lieferbedingungen".

Der Anlagebauer seinerseits muss auch seinen Teil für das zuverlässige Arbeiten einer Anlage beitragen. So ist er dafür verantwortlich, dass die Steuerung datenkonform eingesetzt wird und keine Überbeanspruchungen, z.B. auf Temperaturbereiche, Überspannungen und Störfelder oder mechanischen Beanspruchungen auftreten.

Darüber hinaus ist der Anlagebauer auch dafür verantwortlich, dass ein fehlerhaftes Produkt in keinem Fall zu Verletzungen oder gar zum Tod von Personen bzw. zur Beschädigung oder Zerstörung von Sachen führen kann. Die einschlägigen Sicherheitsvorschriften sind in jedem Fall einzuhalten. Gefährliche Fehler müssen durch zusätzliche Massnahmen erkannt und hinsichtlich ihrer Auswirkung blockiert werden. So sind z.B. für die Sicherheit wichtige Ausgänge auf Eingänge zurückzuführen und softwaremässig zu überwachen. Es sind die Diagnoseelemente der PCD wie Watch-Dog, Ausnahme-Organisations-Blocks (XOB) sowie Test- und Diagnose-Befehle konsequent anzuwenden.

Werden alle diese Punkte berücksichtigt, verfügen Sie mit der SAIA® PCD über eine moderne und sichere programmierbare Steuerung, die Ihre Anlage über viele Jahre zuverlässig steuern, regeln und überwachen wird.

1. Einführung

Im vorliegenden Handbuch wird der vielseitige und praxisorientierte Befehlssatz der SAIA PCD-Familie im Detail beschrieben. Die einzelnen Befehle sind der Übersichtlichkeit halber in Funktionsgruppen zusammengefasst und nicht alphabetisch geordnet.

Das Handbuch “Befehlssatz” ist die Ergänzung zum “Anwender-Handbuch”, in welchem die PCD-Programmierwerkzeuge (Utilities) und die Methoden zur erfolgreichen Strukturierung der Anwenderprogramme beschrieben sind.

Wichtiger Hinweis

Die Beschreibungen sind gültig für
- PCD2 ab Version V002 (und höher)
- PCD 4 ab Version V004 (und höher)
- PCD 6.M5.. ab Version V003 (und höher)
- PCD 6.M1../M2.. ab Version V008 (und höher)
- PCD-Utilities (Programmierwerkzeuge) ab Version V1.7

Bei älteren Versionen können kleine Unterschiede bestehen oder gewisse Funktionen sind noch nicht enthalten.

1.1 Der Aufbau dieses Handbuchs

Jedem PCD-Befehl ist eine oder mehrere Seiten gewidmet.

Titelzeile: Mnemocode und Kurzbeschreibung des Befehls.

Beschreibung: Detaillierte Beschreibung des Befehls.

Aufbau: Zeigt den Aufbau des Befehls und gibt den Typ und den Adressbereich des Operanden an, auf welchen dieser Befehl angewendet werden kann.

Ein [X] nach dem Mnemonic gibt an, dass dieser Befehl durch Anfügen eines ‘X’ indexiert adressiert werden kann (z. B. STHX, INCX).

Für die indexierte Adressierung ist der Operand oder sind die entsprechenden Operanden mit (i) markiert.

Beispiel: Zeigt eine Schreibweise des Befehls.

Flags: Gibt an, welche Status-Flags beeinflusst werden (ACCU, N, P, Z, E)

Siehe auch: Zeigt an, welche verwandten Befehle im gleichen Zusammenhang von Interesse sein können.

Anwendung: Praktisches Beispiel, das diese Instruktion enthält.

1.2 Typografische Übereinkunft

[ ] Ein in eckigen Klammern stehender Text ist optional.
Beispiel: [;Kommentar] heisst, dass ";Kommentar" nicht zwingend geschrieben werden muss.

[X] Ein [X] nach dem Mnemonic gibt an, dass diese Instruktion durch Hinzufügen eines 'X' indexiert adressiert werden kann (z.B. STHX, INCX).

(i) Für die indexierte Adressierung ist der Operand oder sind die entsprechenden Operanden mit (i) markiert.

..... Gibt an, dass die angefangene Reihe weiter geht.

LABEL: In den Beispielen werden Labels mit ':' abgeschlossen. Wird mit einem freien ASCII-Editor gearbeitet, ist ':' zu schreiben. Wird SEDIT verwendet, ist ':' nicht zu schreiben.

< > Zwischen Pfeilklammern stehender Text ist durch einen aktuellen Namen oder Wert zu ersetzen.
Beispiel: <baud_rate> wird zu 9600.

1.3 Einige Begriffserläuterungen

Elemente

Elemente sind adressierbare, im Rahmen der Bereiche frei zuweisbare Funktionsstellen in der PCD. 1-Bit-Elemente können auf ihren logischen Zustand abgefragt, verknüpft oder auch gesetzt und rückgesetzt werden. Mehr-Bit-Elemente können mit Werten geladen werden. Auch können die Werte verändert, verrechnet, gelesen und verknüpft werden.

1-Bit-Elemente sind Eingänge, Ausgänge und Flags. Eingänge können nur von ausserhalb der PCD beeinflusst werden.
Mehr-Bit-Elemente sind Register, Timer und Counter, Timer und Counter können, wie 1-Bit-Elemente auch auf ihren logischen Zustand abgefragt werden.

Mnemocode	Element-Typ	Adress-Bereich
I	Input (Eingang)	0 - 8191
O	Output (Ausgang)	0 - 8191
F	Flag (Merker)	0 - 8191
T	Timer (Zeitglied)	0 - 450
C	Counter (Zähler)	0 - 1599
R	Register	0 - 4095

Resourcen

Resourcen enthalten sämtliche Elemente sowie zusätzlich weitere adressierbare Funktionsteile wie COB, XOB, PB, FB, SB, TR, ST, Konstanten, Anwendertexte, Data-Blocks.
Die verwendeten Resourcen eines Projektes sind in der Datei ".RES" (Resource Table) in numerischer und grafischer Form dargestellt.

Resource	Beschreibung	Adress-Bereich	Bemerkung	Zuteilung
I	Input (Eingang)	0 - 8191		Pro System
O	Output (Ausgang)	0 - 8191		Pro System
F	Flag (Merker)	0 - 8191	nicht flüchtig *)	Pro System
T	Timer (Zeitglied)	0 - 450	flüchtig	Pro System
C	Counter (Zähler)	0 - 1599	nicht flüchtig	Pro System
R	Register	0 - 4095	nicht flüchtig	Pro System
K	K Constant (Konstante)	0 - 16'383	(Wert Bereich)	
COB	Cyclic Org. Block	0 - 15		Pro CPU
XOB	Exception Org. Block	0 - 31		Pro CPU
PB	Program Block	0 - 299		Pro CPU
FB	Function Block	0 - 999		Pro CPU
SB	Sequential Block	0 - 31		Pro CPU
IST	Initial Step	0 - 1999		Pro CPU
ST	Step	0 - 1999		Pro CPU
TR	Transition	0 - 1999		Pro CPU
TEXT	Text	0 - 3999		Pro CPU
DB	Data Block	0 - 3999		Pro CPU
-	Semaphore	0 - 99		Pro System

***)** im Anwenderprogramm mit Befehl DEFVM in nicht-flüchtig/flüchtig aufteilbar.

Gemeinsamer Adressbereich für I/O, T/C, IST/ST und TEXT/DB.

Medien

Medien sind der Sammelbegriff aller im Zusammenhang mit einer PCD und deren Anwenderprogrammen auftretenden Begriffe. Medien enthalten alle Elemente und Resourcen sowie im weiteren z.B. ACCU, Statusflags, Indexregister, Semaphore, dann auch Dateien, Programmstrukturen wie GRAFTEC und BLOCTEC, Parameter, Resultate, Assignierungen, Definitionen, Programm- und Textspeicher, Watch-Dog, Befehlssatz, Bedingungen, Interrupts, Makros usw.

Logischer Zustand von Elementen und ACCU

Eingänge, Ausgänge und Flags sind “H” (High, gesetzt) oder “L” (Low, zurückgesetzt).
Timer und Counter sind ebenfalls “H” oder “L”. “L” bedeutet, dass der Inhalt = 0 ist (Timer abgelaufen Couterinhalt = 0). Andernfalls ist der logische Zustand = H. Der ACCU ist auch “H” oder “L”.
Im Debugger wird “H” mit “1" und ”L" mit “0" angezeigt und muss auch so geschrieben werden.

Status

Der Zustand der Status-Flags wird als Status bezeichnet.

P-Flag gesetzt ⟶ Status “P” (Positiv)
N-Flag gesetzt ⟶ Status “N” (Negativ)
Z-Flag gesetzt ⟶ Status “Z” (Zero, Null)
E-Flag gesetzt ⟶ Status “E” (Error)

Bedingungen (Conditions)

Bedingungen z.B. zum Aufruf von Blocks oder zum Aktivieren von gewissen Befehlen können die folgenden sein:

Bedingung	**Beschreibung**
-	ohne Bedingung
H	wenn ACCU = H
L	wenn ACCU = L
P	wenn Positiv-Flag = H
N	wenn Negativ-Flag = H
Z	wenn Zero-Flag = H
E	wenn Error-Flag = H
(C)	Complement, nur bei Instruktion ACC verwendet

2 BIT-Befehle

Bit-Befehle beeinflussen den ACCU oder werden nur ausgeführt, wenn der ACCU = H ist.

STH	**ST**art **H**igh	Start einer logischen Verknüpfung mit, Abfrage eines 1-Bit Elementes
STL	**ST**art **L**ow	Start einer logischen Verknüpfung mit, invertierter Abfrage eines 1-Bit Elementes
ANH	**AN**d **H**igh	Logische 1-Bit UND-Verknüpfung
ANL	**AN**d **L**ow	Invertierte logische 1-Bit UND-Verknüpfung
ORH	**OR H**igh	Logische 1-Bit ODER-Verknüpfung
ORL	**OR L**ow	Invertierte logische 1-Bit ODER-Verknüpfung
XOR	E**X**clusive **OR**	Logische 1-Bit XOR-Verknüpfung
ACC	**ACC**u	Direktzugriff zum ACCU
DYN	**DYN**amic	Dynamische Abfrage, Flankendetektierung
OUT	set element from ACCU	Setze Ausgang/Flag mit ACCU-Inhalt
SET	**SET** element	Setze Ausgang/Flag speichernd
RES	**RES**et element	Rücksetze Ausgang/Flag speichernd
COM	**COM**plement element	Komplementiere Ausgang/Flag
SETD	**SET** element **D**elayed	Setze Ausgang/Flag zeitverzögert
RESD	**Res**et element **D**elayed	Rücksetze Ausgang/Flag zeitverzögert

Notizen

STH START EINER LOGISCHEN VERKNÜPFUNG MIT ABFRAGE EINES 1-BIT ELEMENTES (Start High)

BESCHREIBUNG: Start einer neuen Verknüpfung. (Vorgängiges logisches Resultat wird gelöscht). Abfrage des adressierten Elementes auf seinen logischen Zustand (H/L) und speichern dieses logischen Zustands im ACCU.

AUFBAU:

STH[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

BEISPIEL:

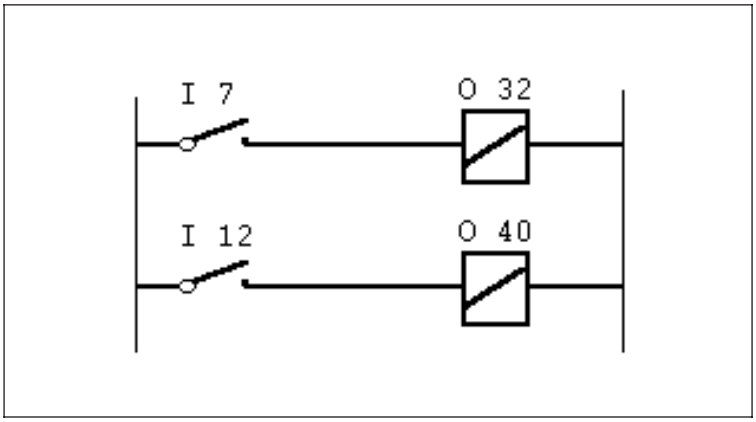
```
STH     I 7     ; ACCU logischer Zustand
                :    des Eingangs I 7
```

FLAGS: Der **ACCU** wird gemäss dem logischen Zustand des Elementes gesetzt.

ANMERKUNG: Wird “STH” auf einen Timer oder Counter angewendet, wird der ACCU = L, wenn der Zählerwert = 0 ist. Ist der Zählerwert 1 oder grösser, wird nach “STH” der ACCU = H.

SIEHE AUCH: STL.

ANWENDUNG:




```
COB     0
        0
STH     I 7     ; log. Zustand von    I   7 → ACCU
OUT     O 32    ; ACCU → Ausgang O   32

STH     I 12    ; log. Zustand von    I   7 → ACCU
OUT     O 40    ; ACCU → Ausgang O   32

ECOB
```

STL START EINER LOGISCHEN VERKNÜPFUNG MIT INVERTIERTER ABFRAGE EINES 1-BIT ELEMENTES (Start Low)

BESCHREIBUNG: Start einer neuen Verknüpfung. (Vorgängiges logisches Resultat wird gelöscht). Abfrage des adressierten Elementes auf seinen logischen Zustand (H/L) und speichern des INVERTIERTEN logischen Zustands im ACCU.

AUFBAU:

STL[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

BEISPIEL:

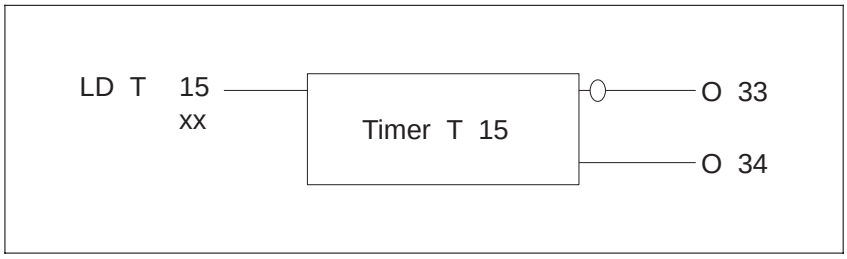
```
STL   I 7     ; ACCU = invertierter logischer
              ; Zustand des Eingangs I 7
```

FLAGS: Der **ACCU** wird gemäss dem invertierten logischen Zustand des Elementes gesetzt.

ANMERKUNG: Wird "STL" auf einen Timer oder Counter angewendet, wird der ACCU = H, wenn der Zählerwert = 0 ist. Ist der Zählerwert 1 oder grösser, wird nach "STL" der ACCU = L.

SIEHE AUCH: STH

ANWENDUNG:




```
STL   T   15   ;   ist Timer 15 = H   →   O  33 = L
OUT   O   33   ;   ist Timer 15 = L   →   O  33 = H

STH   T   15   ;   ist Timer 15 = H   →   O  34 = H
OUT   O   34   ;   ist Timer 15 = L   →   O  34 = L
```

ANH LOGISCHE 1-BIT UND-VERKNÜPFUNG (AND High)

BESCHREIBUNG: Das adressierte Element wird auf seinen logischen Zustand abgefragt (H/L). Das logische Resultat wird mit dem ACCU UND-verknüpft. Das neue Resultat wird wieder im ACCU abgelegt.

AUFBAU:

ANH[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

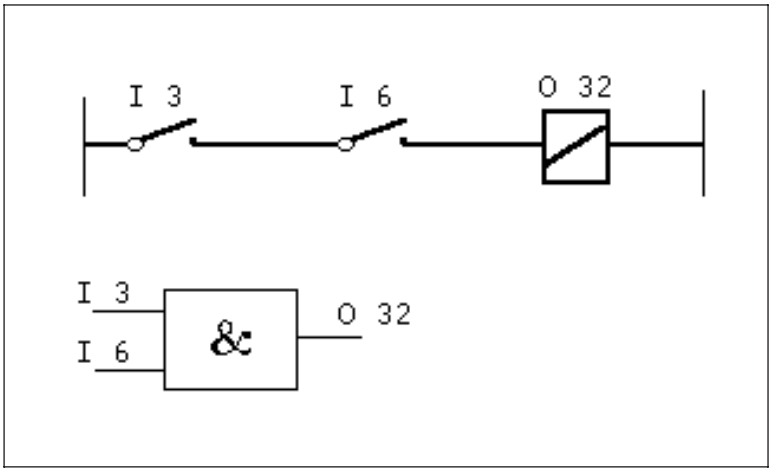
BEISPIEL:

```
ANH    I 3      ; der log. Zustand von I 3 wird
                  mit dem ACCU UND-verknüpft
ANHX   I 128    ; der log. Zustand des Flags 128
                ;  + dem momentanen Wert des Indexregisters
                ;    wird mit dem ACCU UND-verknüpft
```

FLAGS: Der **ACCU** enthält das aktuelle Verknüpfungsresultat.

SIEHE AUCH: ANL.

ANWENDUNG:




```
COB    0
       0

STH    I 3      ; Ist Eingang I 3 = H
ANH    I 6      ;      UND Eingang I 6 = H
OUT    O 32     ;      wird Ausgang O 32 gesetzt
                ;      andernfalls wird Ausgang
                ;      O 32 rückgesetzt
ECOB
```

ANL INVERTIERTE LOGISCHE 1-BIT UND-VERKNÜPFUNG (AND Low)

BESCHREIBUNG: Das adressierte Element wird auf seinen logischen Zustand abgefragt (H/L). Das INVERTIERTE logische Resultat wird mit dem ACCU UND-verknüpft. Das neue Resultat wird wieder im ACCU abgelegt.

AUFBAU:

ANL[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

BEISPIEL:

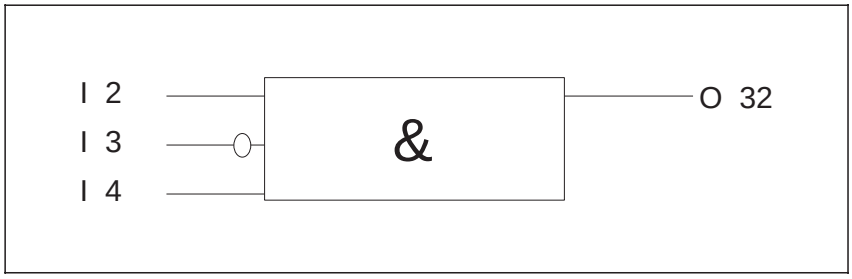
```
ANL    I 3      ; der log. Zustand von I 3 wird mit dem ACCU
                ;    invertiert UND-verknüpft

ANLX   F 128    ; der log. Zustand des Flags 128
                ;    + dem momentanen Wert des Indexregisters
                ;    wird mit dem ACCU invertiert UND-verknüpft
```

FLAGS: Der **ACCU** enthält das aktuelle Vernüpfungsresultat.

SIEHE AUCH: ANH.

ANWENDUNG:




```
COB    0
       0

STH    I 2      ; Ist Eingang I 3 = H
ANL    I 3      ;      UND Eingang I 3= L
ANH    I 4      ;      UND Eingang I 4 = H
OUT    O 32     ;      wird Ausgang O 32 gesetzt
                ;      andernfalls wird Ausgang
                ;      O 32 rückgesetzt
ECOB
```

ORH LOGISCHE 1-BIT ODER-Verknüpfung (OR High)

BESCHREIBUNG: Das adressierte Element wird auf seinen logischen Zustand abgefragt (H/L). Das logische Resultat wird mit dem ACCU ODER-verknüpft. Das neue Resultat wird wieder im ACCU abgelegt.
Die ODER-Instruktionen erzeugen Parallelverknüpfungen einzelner Elemente oder Teilverknüpfungen.
Eine Gesamtverknüpfung beginnt immer mit einem Startbefehl (STH, STL). Jede Parallelverzweigung beginnt mit einem ODER-Befehl (ORH, ORL). (Der Startbefehl für eine Parallelverknüpfung ist im ODER-Befehl enthalten).
Wird eine Parallelverknüpfung als erfolgreich erkannt (ACCU = H), werden die nachfolgenden Parallelverknüpfungen wohl abgearbeitet, der ACCU wird jedoch bis zum nächsten Startbefehl durch weitere UND- bzw. ODER-Befehle nicht mehr verändert.

AUFBAU:

ORH[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

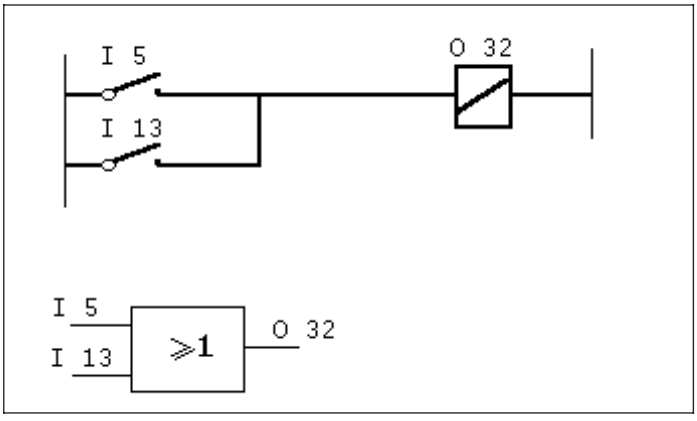
BEISPIEL:

```
STH    I 5     ; Ist Eingang I 5
ORH    I 13    ;   ODER ist Eingang I 13 = H
               ;   wird der ACCU = H,
               ;   andernfalls wird der ACCU = L
```

FLAGS: Der **ACCU** enthält das aktuelle Verknüpfungsresultat.

SIEHE AUCH: ORL.

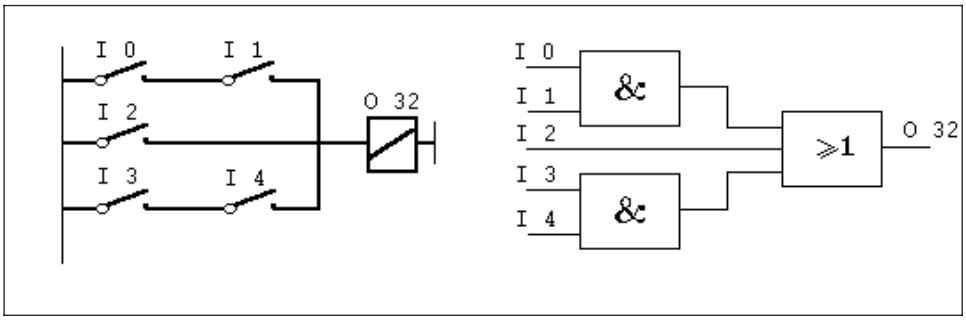
ANWENDUNG:




```
COB    0
       0

STH    I 5     ; Ist Eingang I 5 = H
ORH    I 13    ;   ODER ist I 13 = H
OUT    O 32    ;   wird Ausgang O 32 gesetzt,
               ;   andernfalls wird der
               ;   Ausgang O 32 rückgesetzt
ECOB
```

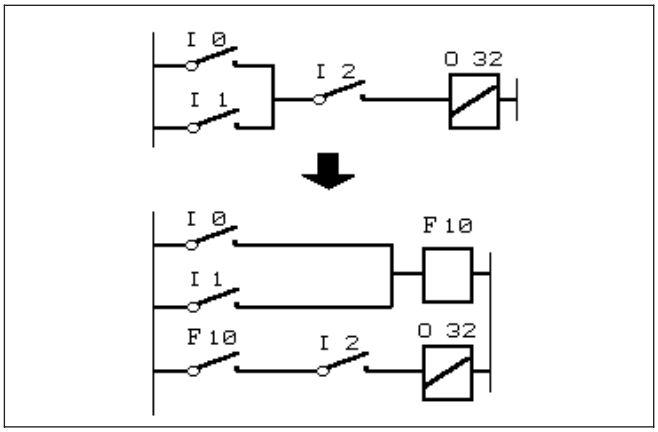
ORH




```
COB     0
        0

STH     I 0     ; Ist Eingang I 0 = H
ANH     I 1     ;   UND Eingang I 1 = H
ORH     I 2     ;   ODER ist Eingang I 2 = H
ORH     I 3     ;   ODER ist Eingang I 3 = H
ANH     I 4     ;   UND ist Eingang I 4 = H
OUT     O 32    ;   wird Ausgang O 32 gesetzt,
                ;     andernfalls wird
                ;     Ausgang O 32 rückgesetzt
ECOB
```

Dieses Beispiel zeigt, dass ODER Priorität vor UND hat.




```
COB     0
        0

STH     I 0     ; Ist Eingang I 0 = H
ORH     I 1     ;   ODER ist Eingang I 1 = H
OUT     F 10    ;   wird Flag F 10 gesetzt
                ;     andernfalls wird Flag 10 rückgesetzt

STH     F 10    ; Ist Flag F 10 = H
ANH     I 2     ;   UND Eingang I 2 = H
OUT     O 32    ;   wird Ausgang O 32 gesetzt
                ;     andernfalls wird
                ;     Ausgang O 32 rückgesetzt
ECOB
```

ORL INVERTIERTE LOGISCHE 1-BIT ODER-VERKNÜPFUNG (OR Low)

BESCHREIBUNG: Das adressiete Element wird auf seinen logischen Zustand abgefragt (H/L). Das logische Resultat wird mit dem ACCU invertiert ODER-verknüpft. Das neue Resultat wird wieder im ACCU abgelegt.

AUFBAU:

ORL[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

BEISPIEL:

```
STH   I 3    ; Ist Eingang I 3 = H
ORL   I 7    ;   ODER ist Eingang I 7 = L
             ;   wird der ACCU = H,
             ;   andernfalls wird der ACCU = L
```

FLAGS: Der **ACCU** enthält das aktuelle Verknüpfungsresultat.

SIEHE AUCH: ORH.

XOR LOGISCHE 1-BIT XOR-VERKNÜPFUNG (Exclusive OR)

BESCHREIBUNG: Mit der XOR-Instruktion können die logischen Zustände zweier Elemente verglichen werden. Sind beide logischen Zustände gleich, wird der ACCU = L, sind sie ungleich, wird der ACCU = H.

AUFBAU:

XOR[X]	**Element (i)**	**; I 0-8191, O 0-8191, F 0-8191** **; T 0-450, C 0-1599**

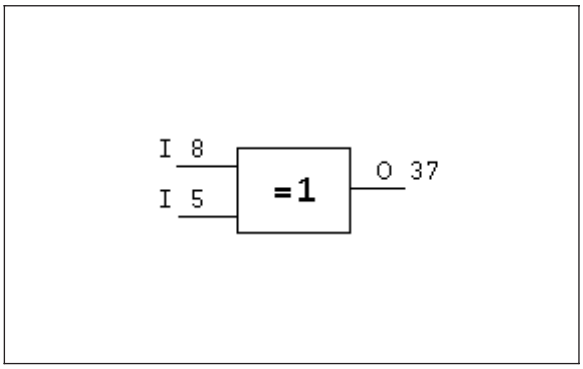
BEISPIEL:

```
STH    I 5     ; Sind beide Eingänge I 5
XOR    I 6     ;    und I 6 logisch gleich,
               ;    wird der ACCU = L,
               ;    andernfalls wird der ACCU = H
```

FLAGS: Der **ACCU** enthält das aktuelle Verknüpfungsresultat.

SIEHE AUCH: ORH, ORL, ANH, ANL

ANWENDUNG:




```
COB    0
       0

STH    I 8     ; Sind beide Eingänge
XOR    I 5     ;    I 8 und I 5 log. gleich
OUT    O 37    ; wird Ausgang O 37 = L (rückgesetzt)
               ;    andernfalls wird der
               ;    Ausgang = H (gesetzt)
ECOB
```

I 8	I 5	O 37
L	L	L
L	H	H
H	L	H
H	H	L

ACC DIREKTZUGRIFF ZUM ACCU (ACCU)

BESCHREIBUNG: Der ACCU wird auf H oder L gesetzt, komplementiert oder entsprechend dem angegebenen Status-Flag gesetzt.

C	Complement	ACCU wird komplementiert
H	High	ACCU wird “H” gesetzt
L	Low	ACCU wird “L” gesetzt
P	Positive	ACCU wird gemäss dem P-Flag gesetzt
N	Negative	ACCU wird gemäss dem N-Flag gesetzt
Z	Zero	ACCU wird gemäss dem Z-Flag gesetzt
E	Error	ACCU wird gemäss dem ERROR-Flag gesetzt

Der Operand kann nicht als Parameter übergeben werden.

AUFBAU:

```
ACC     Code     ; Code = C|H|L|P|N|Z|E
```

BEISPIEL:

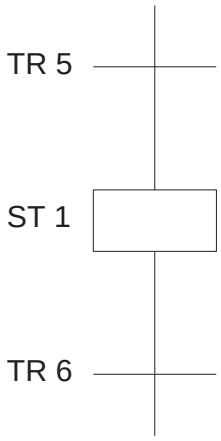
```
ACC   H    ; Setzt den ACCU auf H
ACC   E    ; Setzt den ACCU entsprechend
           ;    dem Status des Error-Flags
```

FLAGS: Der **ACCU** wird beeinflusst

ANMERKUNG: Die Parametrierung kann nicht angewendet werden

SIEHE AUCH: OUT, Condition Codes (Bedingungen)

ANWENDUNG:




```
CMP   R 99   ; Vergleiche R 99
             ‘x’ ;       mit ASCII x
ACC   Z      ; wenn gleich, wird
             ;       Z-Flag gesetzt und
             ;       der ACCU wird = H

STXT    1
        10

. . . .
```

DYN DYNAMISCHE ABFRAGE, FLANKENERKENNUNG (DYNAMIC)

BESCHREIBUNG: Nach einer erfolgreichen Abfrage eines Elementes oder einer Verknüpfung (ACCU = H) kann mit dem Befehl DYN eine dynamische Ansteuerung (Flankenerkennung) der nachfolgenden (accuabhängigen) Sequenz erreicht werden.
Anders gesagt: Wechselt der Inhalt des ACCU von “L” nach “H”, so bleibt nach DYN der Zustand “ACCU = H” nur während dieses Programmzyklus erhalten. Bei den folgenden Zyklen ist der ACCU an dieser Stelle “L”.
Jeder DYN-Befehl verlangt im Operand ein Flag zur Abspeicherung der erkannten Flanke.

AUFBAU:

DYN[X]	**Element (i)**	**; F 0-8191, O 0-8191**

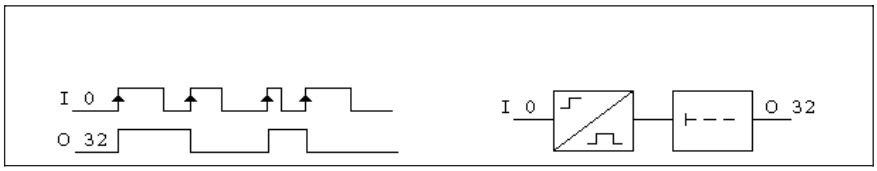
BEISPIEL:

```
DYN   F 100   ; im Merker F 100 ist die detektierte Flanke
              ; gespeichert
```

FLAGS: Der **ACCU** wird in den folgenden Programmzyklen nach der Erkennung der Flanke wieder = L.

SIEHE AUCH: STH, STL, ANH, ANL, ORH, ORL.

ANWENDUNG:



Lösung mit DYN-Instruktion

```
COB   0
      0
STH   I 0     ; Ist Eingang I 0 erstmals H (Flanke),
DYN   F 500   ;   wird F 500 gesetzt
COM   O 32    ;   und COM wird in diesem Zyklus
              ;   ausgeführt
ECOB
```

Lösung ohne DYN-Instruktion

```
COB   0
      0
STH   I 0     ; Ist Eingang I 0 erstmals H (Flanke)
ANL   F 500   ;   und ist F 500 = l (1. Zyklus),
SET   F 500   ;   wird F 500 = H.
COM   O 32    ; COM wird im 1. Zyklus nach der
              ;   Flanke ausgeführt.
STL   I 0     ; Wird Eingang I 0 = L,
RES   F 500   ;   wird F 500 rückgesetzt.

ECOB
```

OUT SETZE AUSGANG/FLAG MIT ACCU-INHALT (Set Element from Accumulator)

BESCHREIBUNG: Setzt den adressierten Ausgang oder das Flag gemäss dem logischen Zustand des ACCU.

AUFBAU:

OUT[X]	**Element (i)**	**; O 0-8191, F 0-8191**

BEISPIEL:

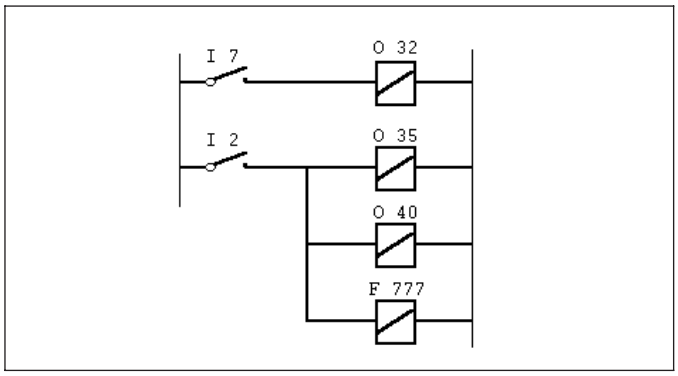
```
OUT    O 32    ; Überträgt den logischen
               ;    Zustand des ACCU (H,L)
               ;    auf den Ausgang O 32
```

FLAGS: Der **ACCU** wird nicht verändert.

ANMERKUNG: OUT wird vorwiegend in Umlaufprogrammen eingesetzt. In sequentiellen Programmen kommen SET und RES zur Anwendung.

SIEHE AUCH: (OUTS)

ANWENDUNG:




```
COB    0
       0

STH    I 7     ; Abfrage von Eingang I 7
OUT    O 32    ; Übertragen des logischen
               ;    Zustandes auf O 32
STH    I 2     ; Abfrage von Eingang I 2
OUT    O 35    ; Übertragen des logischen
OUT    O 40    ;    Zustandes auf O 35,
OUT    F 777   ;    O 40 und F 777

ECOB
```

SET

SET SETZE AUSGANG/FLAG SPEICHERND (Set Element)

BESCHREIBUNG: Der adressierte Ausgang oder das Flag wird gesetzt.
Das Setzen wird nur ausgeführt, wenn der ACCU = H ist.

Mit SET gesetzte Ausgänge oder Flags bleiben gesetzt bis diese mit RES wieder zurückgesetzt oder die Steuerung ausgeschaltet wird.

AUFBAU:

SET[X]	**Element (i) ; O 0-8191, F 0-8191**

BEISPIEL:

```
SET     O 32    ; Ist der ACCU = H, setze
                ;   Ausgang O 32
```

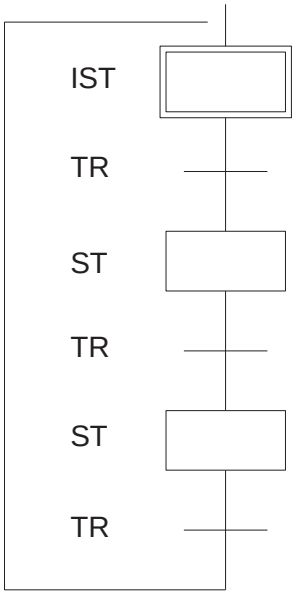
FLAGS: Der **ACCU** wird nicht verändert.

ANMERKUNG: Die Befehle SET und RES kommen hauptsächlich in sequentiellen Programmstrukturen (GRAFTEC, Flussdiagramm) zur Anwendung.

Im Umlaufprogrammen wird vorallem mit dem Befehl OUT gearbeitet. Selbstverständlich kann SET und RES auch in Umlaufprogrammen verwendet werden. Es ist dann auf die Prioritäten von RES und SET zu achten.

SIEHE AUCH: RES, (OUT)

ANWENDUNG: Nach dem Einschalten von Eingang I 7 sollen die Ausgänge 36 und 37 blinken.




```
STH   I     7
SET   O     36
SET   O     37
LDL   T     1,    5     ; 0,5 sek
STL   T     1
RES   O     36
RES   O     37
LDL   T     2,    10    ; 1 sek
STL   T     2
```

RES RÜCKSETZE AUSGANG/FLAG SPEICHERND (Reset Element)

BESCHREIBUNG: Der adressierte Ausgang oder das Flag wird rückgesetzt.
Das Rücksetzen wird nur ausgeführt, wenn der ACCU = H ist.

AUFBAU:

RES[X]	**Element (i)**	**; O 0-8191, F 0-8191**

BEISPIEL:

```
RES     O 32    ; Ist der ACCU = H, wird
                ;    Ausgang O 32 rückgesetzt
```

FLAGS: Der **ACCU** wird nicht verändert.

ANMERKUNG: Die Befehle SET und RES kommen hauptsächlich in sequentiellen Programmstrukturen (GRAFTEC, Flussdiagramm) zur Anwendung.

In Umlaufprogrammen wird vorallem mit dem Befehl OUT gearbeitet. Selbstverständlich kann SET und RES auch in Umlaufprogrammen verwendet werden. Es ist dann auf die Prioritäten von RES und SET zu achten.

SIEHE AUCH: SET, (OUT)

ANWENDUNG: siehe SET

COM KOMPLEMENTIERE AUSGANG/FLAG (Complement Element)

BESCHREIBUNG: Der adressierte Ausgang oder das Flag wird auf seinen logischen Zustand abgefragt und anschliessend auf den invertierten logischen Zustand gesetzt. **Der Befehl wird nur ausgeführt, wenn der ACCU = H ist.**

AUFBAU:

COM[X]	**Element (i)**	**; O 0-8191, F 0-8191**

BEISPIEL:

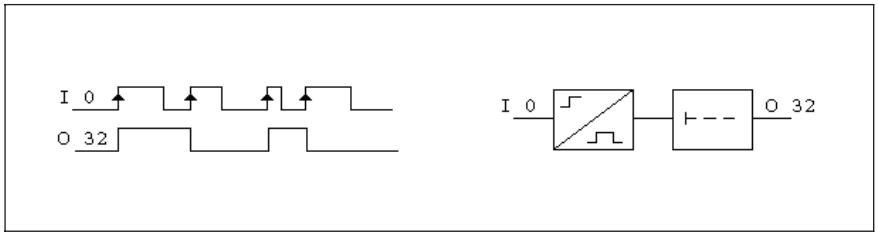
```
COM     O 32    ; ist der ACCU = H, wird der O 32 invertiert
```

FLAGS: Der **ACCU** wird nicht verändert.

ANMERKUNG: Dieser Befehl wird hauptsächlich zur Aktivierung des Watch-Dogs verwendet. In einem Umlaufprogramm ist “COM O 255" zu scheiben. Es ist darauf zu achten, dass der ACCU = H ist. (Befehl ”ACC H" oder COM direkt nach der Instruktion “COB” einfügen. Am Anfang jedes Blocks ist der ACCU = H).

SIEHE AUCH:

ANWENDUNG:




```
COB     0
        0

STH     I 0     ; Abfrage Eingang I 0
DYN     F 500   ;    Flankenerkennung
COM     O 32    ; wenn ACCU = H,
                ;    komplementiere O 32
ECOB
```

SETD SETZE AUSGANG/FLAG ZEITVERZÖGERT (Set Element Delayed)

BESCHREIBUNG: Der adressierte Ausgang oder das Flag werden erst nach der in der 2. Zeile des Befehls angegebenen Zeit gesetzt **und dies nur, wenn der ACCU = H ist.**

Die Zeitbasis für die Verzögerung ist 100 ms, falls nicht mit dem Befehl DEFTB ein anderer Wert definiert wurde.

Es können max. 16 verzögerte Funktionen dieser Art (SETD, RESD) gleichzeitig ausgeführt werden.

AUFBAU:

SETD[X]	**Element (i)**	**; O 0-8191, F 0-8191**
	Verzögerung	**; Anzahl Zeitbasis-Einheiten**

BEISPIEL:

```
SETD   O 32    ; ist der ACCU = H wird O 32
       100     ;    nach 100 Zeiteinheiten gesetzt
```

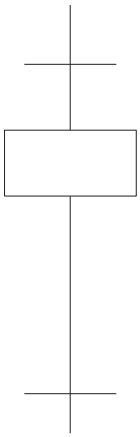
FLAGS: Das Error-Flag wird gesetzt, wenn versucht wird, mehr als 16 verzögerte Aktionen gleichzeitig auszuführen.

ANMERKUNG: Dieser Befehl ist vorwiegend für sequentielle Abläufe gedacht (GRAFTEC),wo sich eine vereinfachte Struktur ergeben kann, da das Ende der Verzögerung nicht abgewartet werden muss.

Wird der Befehl in einem Umlaufprogramm eingesetzt, ist dieser IMMER dynamisch anzusteuern (DYN), ansonst in jedem Programmzyklus automatisch ein nächstes der 16 Zeitglieder aktiviert wird.
Die Operanden können nicht als Parameter an FB übergeben werden.

SIEHE AUCH: RESD, DEFTB

ANWENDUNG:



```
STH . . .              ;  Weiterschaltbedingung

SET  O     35     ;  O  35 wird sofort gesetzt
SETD O     40     ;  O  40 wird erst nach
           1200   ;       120 sek gesetzt und zwar
                  ;       unabhängig davon, ob der
                  ;       Programmablauf weiter
                  ;       geht oder nicht

STH . . .              ; Weiterschaltbedingung
```

RESD RÜCKSETZE AUSGANG/FLAG ZEITVERZÖGERT (Reset Element Delayed)

BESCHREIBUNG: Der adressierte Ausgang oder das Flag wird erst nach der in der 2. Zeile des Befehls angegebenen Zeit rückgesetzt **und dies nur, wenn der ACCU = H ist.**

Die Zeitbasis für die Verzögerung ist 100 ms, falls nicht mit dem Befehl DEFTB ein anderer Wert definiert wurde.

Es können max. 16 verzögerte Funktionen dieser Art (SETD, RESD) gleichzeitig ausgeführt werden.

AUFBAU:

```
RESD[X]   Element (i)    ; O 0-8191, F 0-8191
          Verzögerung    ; Anzahl Zeitbasis-Einheiten
```

BEISPIEL:

```
RESD   O 32    ; ist der ACCU = H wird O 32
         100   ;    nach 100 Zeiteinheiten rückgesetzt
```

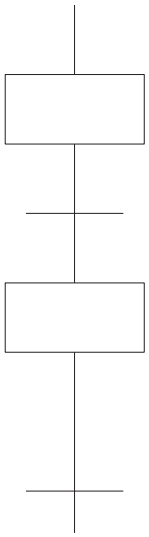
FLAGS: Das Error-Flag wird gesetzt, wenn versucht wird, mehr als 16 verzögerte Aktionen gleichzeitig auszuführen.

ANMERKUNG: Dieser Befehl ist vorwiegend für sequentielle Abläufe gedacht (GRAFTEC), wo sich eine vereinfachte Struktur ergeben kann, da das Ende der Verzögerung nicht abgewartet werden muss.

Wird der Befehl in einem Umlaufprogramm eingesetzt, ist dieser IMMER dynamisch anzusteuern (DYN), ansonst in jedem Programmzyklus automatisch ein nächstes der 16 Zeitglieder aktiviert wird.
Die Operanden können nicht als Parameter an FB übergeben werden.

SIEHE AUCH: SETD, DEFTB

ANWENDUNG:



```
. . .
SET   O     41    ;   O 41 wird sofort gesetzt

STH . . .         ;   Weiterschaltbedingung

SET   O     39    ;   O  39 wird sofort gesetzt
RESD  O     41    ;   O  41 wird erst nach
            100   ;        10 sek rückgesetzt und
                  ;        zwar unabhängig davon,
                  ;        ob der Programmablauf
                  ;        weiter geht oder nicht
STH . . .         ; Weiterschaltbedingung
```

3. WORT Befehle

Alle Wortbefehle arbeiten mit Registern, Timern, Countern oder dem Indexregister. Register enthalten binäre, dezimale, hexadezimale, BCD, ASCII, oder Fliesspunkt-Werte. Für das Fliesspunkt-Format müssen die entsprechenden Befehle verwendet werden. CPU-intern sind die Registerwerte immer binär abgelegt. Für Fliesspunkt liegt ein anderes Format vor. Die beiden Formate dürfen nicht gemischt werden.

LADEN VON REGISTERN

LD	**L**oa**D**	Lade 32-Bit Wert
LDL	**L**oa**D** **L**ow word	Lade die unteren 16 Bit
LDH	**L**oa**D** **H**igh word	Lade die höheren 16 Bit
DSP	Load **D**i**SP**lay Reg.	Lade das Display-Register

AUF-UND ABZÄHLEN

INC	**INC**rement	Erhöhe Register/Counter (+ 1)
DEC	**DEC**rement	Erniedrige Register/Counter (- 1)

INDEXREGISTER

SEI	**SE**t Indexregister	Setze das Indexregister
INI	**IN**crement **I**ndexregister	Erhöhe das Indexregister (+ 1)
DEI	**DE**crement **I**ndexregister	Erniedrige das Indexregister (-1)
STI	**ST**ore **I**ndexregister	Speichere das Indexregister
RSI	**R**e**S**tore **I**ndexregister	Hole das Indexregister zurück

VERSCHIEBEN VON DATEN

MOV	**MOV**E data	Verschiebe Daten	
COPY	**COPY** data	Kopiere Daten	¦ Interessant im
GET	**GET** data	Hole Daten	¦ Zusammenhang
PUT	**PUT** data	Bringe Daten	¦ mit der Indexierung
TFR	**T**rans**F**e**R** Data	Transfer Daten	
TFRI	**T**rans**F**e**R** Data **I**ndirect	Transfer Daten indirekt	

EIN- UND AUSGABE VON BINÄRWERTEN

BITI	**BIT** In	Einzel-Bit in Register, PCD-Format
BITIR	**BIT** In Reversed	Einzel-Bit in Register, PCA-Format
BITO	**BIT** Out	Register zu Einzel-Bit, PCD-Format
BITOR	**BIT** Out Reversed	Register zu Einzel-Bit, PCA-Format

EIN- UND AUSGABE VON BCD-WERTEN

DIGI	**DIG**it **I**n	Digit in Register, PCD-Format
DIGIR	**DIG**it **I**n **R**eversed	Digit in Register, PCA-Format
DIGO	**DIG**it **O**ut	Register zu Digit, PCD-Fon-nat
DIGOR	**DIG**it **O**ut **R**eversed	Register zu Digit, PCA-Format

(Fortsetzung nächste Seite)

LOGISCHE VFRKNÜPFUNGEN VON REGISTERN

AND	**AND** registers	UND-Verknüpfung von Registern
OR	**OR** registers	ODER-Verknüpfung von Registern
EXOR	**EXOR** registers	EXOR-Verknüpfung von Registern
NOT	Complement Register (NOT function)	Komplementierung von Registern

SCHIEBEN UND ROTIEREN

SHIU	**SHIft** registers **U**p	Schiebe Register aufwärts
SHID	**SHIft** registers **D**own	Schiebe Register abwärts
ROTU	**ROT**ate registers **U**p	Rotiere Register aufwärts
ROTD	**ROT**ate registers **D**own	Rotiere Register abwärts
SHIL	**SHI**ft register content **L**eft	Schiebe Registerinhalt nach rechts
SHIR	**SHI**ft reg. content **R**ight	Schiebe Registerinhalt nach links
ROTL	**ROT**ate reg. content **L**eft	Rotiere Registerinhalt nach rechts
ROTR	**ROT**ate reg. content **R**ight	Rotiere Registerinhalt nach links

LD LADE 32-BIT WERT (LOAD 32-Bit Value)

BESCHREIBUNG: Das adressierte Register, Timer oder Counter wird mit dem in der 2. Zeile notierten Wert geladen.

Für Timer und Counter gilt:

- Die Operation wird nur ausgeführt, **wenn der ACCU = H ist**.
- Es können nur positive dezimale, hexadezimale, ASCII- oder binäre Werte geladen werden. (keine negativen oder Fliesspunkt-Werte)
- Ist ein Timer geladen, läuft er sofort ab.
- Der logische Zustand eines Timers oder Counters ist = H, wenn der Inhalt 1 oder grösser ist. Ist der Inhalt = Null, ist der logische Zustand = L.

Für Register gilt:

- Die Operation wird immer, d.h. unabhängig vom Zustand des ACCU ausgeführt.
- Es können positive und negative Werte in allen Formaten geladen werden.

Binärwerte haben die Endung Q oder Y: 110011Y
Hexadezimalwerte enden mit H: 12ABCH
Fliesspunkt-Werte müssen einen Dezimalpunkt und/oder einen Exponenten enthalten: 1.25E-3.
ASCII-Werte sind zwischen einfache Anführungszeichen zu scheiben: ‘a’, ‘A’.
Obwohl LD nur 2-zeilig geschrieben wird, beansprucht dieser Befehel im Programm 3 Zeilen.

Die Operanden können nicht als Parameter an FB übergeben werden.

AUFBAU:

```
LD[X]     Element (i)   ; R 0-4095, T 0-450, C 0-1599.
          Wert          ; Dezimal:  -2.147.483.648  - +2.147.483.647
                        ; Hex: 0H - FFFFFFFFH
                        ; Binär: 0 - 1111...1111 (32 Bit)
                        ; Fliesspunkt: +5.42101E-20..+ 9.22337E+18
                                       -5.42101E-20. . -9.22337E+18
                        ; ASCII: ‘A’-’Z’, ‘0’-’9’, ‘!’, usw.
```

BEISPIEL:

```
LD    R  555     ; Lade Register 555
         12AB3H  ;    mit dem Hex-Wert 12AB3
```

FLAGS: Unverändert

ANMERKUNG: LD belegt 3 Programmzeilen

LD T/C : wird nur ausgeführt, wenn ACCU = H
LD R : wird immer ausgeführt

SIEHE AUCH: LDL, LDH, Konstanten.

LDL LADE DIE UNTEREN 16 BIT (Load Low Word)

BESCHREIBUNG: Die unteren 16 Bit des adressierten Registers, Timers oder Counters werden mit dem in der 2. Zeile notierten Wert geladen (0 - 65’535). Die höheren 16 Bit werden dabei immer 0 gesetzt.
Für Timer und Counter gilt: LDL wird nur ausgeführt, wenn der ACCU = H ist.
Fü Register gilt: LDL wird immer, d.h. unabhängig vom Zustand des ACCU ausgeführt.

LDL ermöglicht sowohl das direkte Laden der Elemente als auch das parametrierte Laden in einen FB.

Mit diesen beiden Befehlen können 32-Bit Werte parametriert in Register geladen werden. Es sind zuerst immer die unteren 16 Bit mit LDL und anschliessend die höheren 16 Bit mit LDH zu laden.

Werte können in den Formaten dezimal, hexadezimal, binär oder ASCII geladen werden. Das Fliesspunkt- Format kann nicht angewendet werden.

AUFBAU:

```
LDL[X]    Element (i)    ; R 0-4095, T 0-450, C 0-1599
          Wert           ; Dezimal:  0-65535
                         ; Hex.:     0H-FFFFH
                         ; Binär:    16 Bit
```

BEISPIEL:

```
LDL    R 555      ; Lade Register R 555
       7FE3H      ;  mit dem Hex-Wert 7FE3
```

FLAGS: Unverändert

ANMERKUNG: Da sehr oft mit Werten < 65535 gearbeitet wird, kann LDL als Standardbefehl zum Laden von Registern, Timern und Countern betrachtet werden.

SIEHE AUCH: LDH, LD, Konstanten.

LDH LADE DIE HÖHEREN 16 BIT (Load High Word)

BESCHREIBUNG: Die höheren 16 Bit des adressierten Registers werden mit dem in der 2. Zeile notierten Wert geladen. Die unteren 16 Bit werden dabei nicht beeinflusst.
LDH kann nur auf Register angewendet werden.
LDH ermöglicht sowohl das direkte Laden von Registern als auch das parametrierte Laden in einen FB.
Mit diesen beiden Befehlen können 32-Bit Werte parametriert in Register geladen werden. Es sind zuerst immer die unteren 16 Bit mit LDL und anschliessend die höheren 16 Bit mit LDH zu laden.
Werte können in den Formaten dezimal, hexadezimal, binär oder ASCII geladen werden. Das Fliesspunkt- Format kann nicht angewendet werden.

AUFBAU:

```
LDH[X]   Element (i)   ; R 0-4095
         Wert          ; Dezimal:  0-65'535
                       ; Hex.:     0H-FFFFH
                       ; Binär:    16 Bit
```

BEISPIEL:

```
LDH   R 100      ; Lade Register R1000
      0A7C3H ;   ; mit dem Hex-Wert A7C3
           ;     ; R 100 = A7C3xxxx Hex.
```

FLAGS: Unverändert

SIEHE AUCH: LDL, LD, Konstanten

ANWENDUNG: Es soll der Dezimalwert 12345678 parametriert in ein Register, das sich in einem FB befindet, geladen werden. Unter Zuhilfenahme der Assembler-Operationszeichen "&" (AND) und "/" (DIVIDE) können die unteren und die höheren 16 Bit direkt als Dezimalwert, ohne umständliche Rechnerei, eingegeben werden.

WICHTIG: LDL muss vor LDH stehen.

```
COB     0
        0

CFB     5              ; Aufruf FB 5
12345678 & 0FFFFH      ; Param. 1 (untere 16 Bit)
12345678 / 0FFFFH      ; Param. 2 (höhere 16 Bit)

ECOB
;-------------------------------------------------------
FB      5              ; Funktions Block 5
LDL     R  25          ; Lade die unteren 16 Bit
        =  1           ;  mit dem 1. Parameter
LDH     R  25          ; Lade die höheren 16 Bit
        =  2           ;  mit dem 2. Parameter
EFB
```

DSP LADE DAS DISPLAY-REGISTER (Load Display Register)

BESCHREIBUNG: Der logische Zustand eines Einganges, Ausganges oder Flags oder der Inhalt eines Registers, Timers oder Counters oder eine Konstante kann in das Display- Register geladen werden.
Dieser Wert kann am Servicegerät PCD8.P100 ohne Bedienung angezeigt werden.

Der Operand dieses Befehls kann nicht als Parameter an einen FB übergeben werden.

AUFBAU:

```
DSP     Element     ; I 0-8191, O 0-8191, F 0-8191,
                    ; T 0-450, C 0-1599, R 0-4095,
                    ; K 0-16383
```

BEISPIEL:

```
        DSP     R 120       ; Display Register = Inhalt von R 120

        DSP     K 12345     ; Display Register = Konstante 12345
```

FLAGS: Unverändert

SIEHE AUCH: -

ANWENDUNG: Siehe Anwendung bei INC-Befehl (nächste Seite)

Bei der PCD2 wird der DSP-Befehl im Zusammenhang mit dem PCD2.F510- bzw. PCD2.F530-Displaymodul verwendet.

Siehe “Handbuch der Baureihe PCD1/PCD2 Hardware”.

INC ERHÖHE REGISTER/COUNTER (+1) (Increment)

BESCHREIBUNG: Der Wert des adressierten Registers oder Counters wird um 1 erhöht.
Für Counter: der Befehl wird nur ausgeführt, **wenn der ACCU = H ist**.
Für Register: der Befehl wird unabhängig vom Zustand des ACCU immer ausgeführt.

AUFBAU:

INC[X]	Element (i)	; R 0-4095, C 0-1599

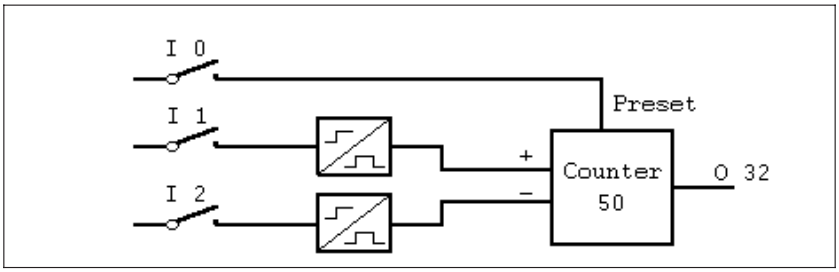
BEISPIEL:

```
INC     R 100   ; R 100 = R 100 + 1
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Wert im Register oder Counter gesetzt.
Im Falle eines Überlaufes (Overflow) im Register oder Counter wird das Error-Flag gesetzt.

SIEHE AUCH: DEC, ADD.

ANWENDUNG: Auf/Ab-Zähler mit Vorwahl und Display des Zählwertes




```
COB      0
         0

STH      I  0    ; Ist Eingang I 0 = H
LD       C 50    ;    Lade Counter C 50 mit 5
           5
                 ; sonst gehe weiter
STH      I  1    ; Ist Eingang I 1 = H
DYN      F  1    ;   (Flanke)
INC      C 50    ;    Counter C 50 +1
                 ; sonst gehe weiter
STH      I  2    ; Ist Eingang I 2 = H
DYN      F  2    ;   (Flanke)
DEC      C 50    ;    Counter C 50 -1
                 ;   sonst gehe weiter
STH      C 50    ; Ist Counterinhalt > 0
OUT      O 32    ;    Setze Ausgang 0 32
                 ; sonst rücksetze 0 32
DSP      C 50    ; Display Counter C 50
ECOB
```

DEC DEKREMENTIERE REGISTER ODER ZÄHLER

Beschreibung Der Wert des adressierten Registers oder Zählers wird um 1 dekrementiert.

Für Zähler: der Befehl wird nur ausgeführt, wenn der **ACCU = H** ist.

Für Register: der Befehl wird nur ausgeführt, unabhängig vom ACCU-Zustand.

Aufbau:

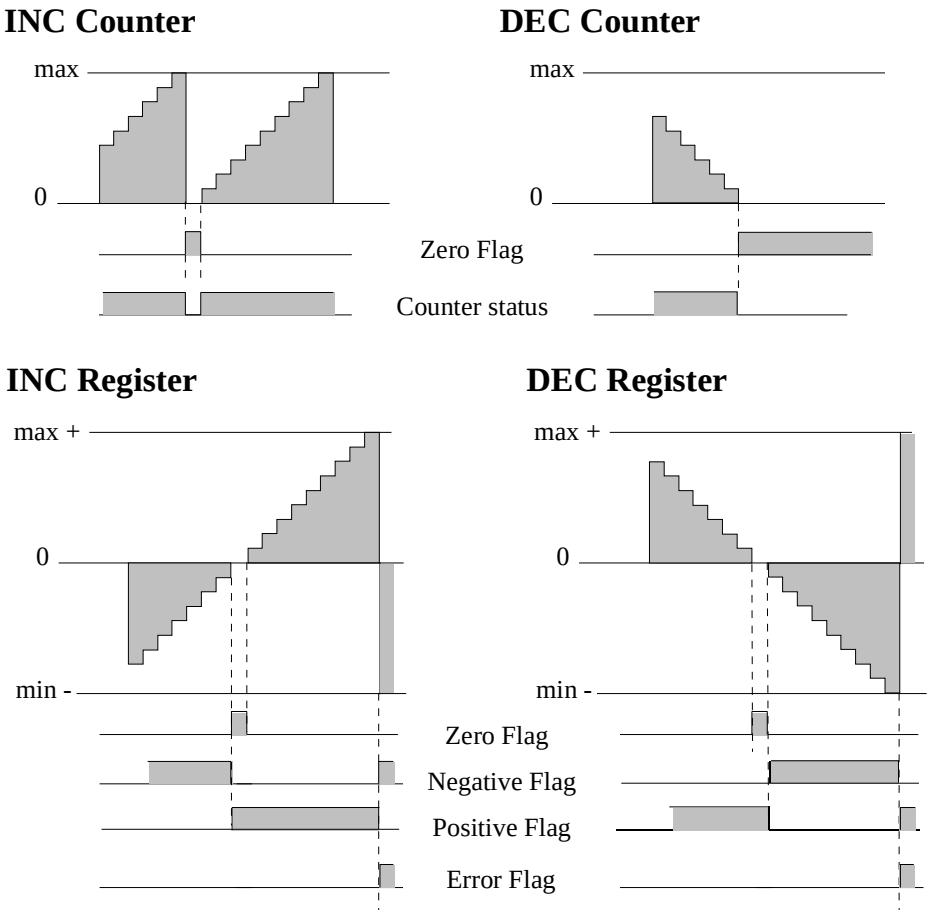
DEC[X] Element (i) ; R 0-4095, C 0-1599

Beispiel:
```
DEC    R 100        ; R 100 = R 100 - 1
```

Flags Das Z- (Zero), das P- und **das** N-Flag werden gemäss dem Wert im Register oder Zähler gesetzt. Das E-Flag (Error) wird bei einem Register- oder Zähler-Überlauf (Overflow).

Hinweis: Zähler können keine negativen Werte einnehmen.

Siehe auch: INC, SUB



INC-/ DEC-Verhalten von Zählern und Registern

SEI SETZE DAS INDEXREGISTER (Set Indexregister)

INDEXIERTE ADRESSIERUNG

Es kommt in der Praxis oft vor, dass ganze Reihen von Elementen in der gleichen Weise behandelt werden müssen, z.B. das Rücksetzen von Flags oder Ausgängen oder das Rücksetzen (mit Null laden) von Registern oder Countern. Mit Hilfe der Indexierung können solche Aufgaben elegant und mit wenig Programmschritten gelöst werden.
Jeder COB und alle XOB gemeinsam haben je ein Indexregister (IR).
Bei der indexierten Adressierung wird zu dem indexiert zu behandelnden Operanden der aktuelle Wert des IR addiert. Den zu indexierenden Elementen wird dem Mnemonic des betreffenden Befehls ein "X" nachgestellt, z.B. STHX, BITIX.
Das IR kann in gewissen Grenzen beliebig geladen, inkrementiert und dekrementiert sowie gerettet und wieder hergestellt werden.

BESCHREIBUNG: Das Indexregister (IR) des aktuellen COB wird mit der im Operand angegebenen Konstante (K 0-8191) oder mit dem Inhalt des angegebenen Registers geladen.

WICHTIG: Der Wertebereich des IR ist 0-8191 (13 Bit) (nur positive Werte).

Ist der Wert > 8191, wird das IR auf 8191 gesetzt und der XOB 12 aufgerufen.
Ist der Wert < 0, wird das IR auf 0 gesetzt und der XOB 12 aufgerufen.

AUFBAU:

SEI	**Konstante, Register ; K 0-8191, R 0-4095**

BEISPIEL:

```
SEI     K 32    ; Lädt IR mit dem Wert 32
SEI     R 32    ; Lädt IR mit dem Inhalt von Register R 32
                ;   (nur die unteren 13 Bit)
```

FLAGS: Unverändert.

ANMERKUNG: Wird mit der Indexierung gearbeitet, ist der XOB 12 zu programmieren.

SIEHE AUCH: INI, DEI, STI, RSI.

ANWENDUNG: Der logische Zustand eines Flags, dessen Adresse an 2 BCD-Schaltern eingegeben wird, ist am Ausgang O 32 sichtbar zu machen. (Die BCD-Schalter sind an die Eingänge I 24 - 31 verdrahtet).

```
COB     0
        0
DIGI    2       ; Lese 2 Digits
        I 24    ;   von Eingang I 24 (-31)
        R 500   ;   in Register R 500
SEI     R 500   ; Lade IR mit Wert von R 500
STHX    F 0     ; Ist Flag F 0 + Index = H
OUT     O 32    ;   wird Ausgang O 32 gesetzt sonst wird Ausgang
                ;   O 32 rückgesetzt
ECOB
```

INI ERHÖHE DAS INDEXREGISTER (+1) (Increment Indexregister)

BESCHREIBUNG: Das Indexregister (IR) wird mit dem Wert im Operand verglichen. (Konstante oder Wert des angegebenen Registers).
Ist der Stand des IR kleiner als dieser Wert, wird das IR inkrementiert (+1) und der ACCU wird oder bleibt = H.
Ist der Stand des IR gleich oder grösser als dieser Wert, wird das IR nicht inkrementiert und der ACCU wird oder bleibt = L.
Ist der Wert im Operand bzw. im adressierten Register >8191 oder <0, wird der XOB 12 aufgerufen.

AUFBAU:

INI	**Konstante, Register; K0-8191, R 0..4095**

BEISPIEL:

```
        INI     K 100   ; IR +1, solange IR < 100

        INI     R 333   ; IR +1, solange IR < als
                        ;    Wert  in Register R 333 ist
```

FLAGS: **ACCU** = H, wenn IR **kleiner** als Wert im Operand
ACCU = L, wenn IR **gleich oder grösser** als der Wert im Operand bzw. im adressierten Register.

ANMERKUNG: Wird mit der Indexierung gearbeitet, ist der XOB 12 zu programmieren.

SIEHE AUCH: DEI, SEI

ANWENDUNG: Nach dem Einschalten der PCD sollen die Register R 1500 - 1999 gelöscht, d.h. mit 0 geladen werden.

```
        XOB     16          ; Kaltstart-Routine
        .....
        SEI     K 0         ; Setze IR = 0

Reset:  LDX     R 1500      ; Lade Register 1500+IR
                  0         ;    mit 0
        INI     K 499       ; IR +1
        JR      H Reset     ;    wiederhole solange IR < 499
        (ACC    H)
        .....
        EXOB
```

DEI ERNIEDRIGE DAS INDEXREGISTER (-1) (Decrement Indexregister)

BESCHREIBUNG: Das Indexregister (IR) wird mit dem Wert im Operand verglichen. (Konstante oder Wert des angegebenen Registers).
Ist der Stand des IR grösser als dieser Wert, wird das IR dekrementiert (-1) und der ACCU wird oder bleibt = H.
Ist der Stand des IR gleich oder kleiner als dieser Wert, wird das IR nicht dekrementiert und der ACCU wird oder bleibt = L.
Ist der Wert im Operand bzw. im adressierten Register > 8191 oder < 0, wird der XOB 12 aufgerufen.

AUFBAU:

DEI	**Konstante, Register; K 0-8191, R 0..4095,**

BEISPIEL:

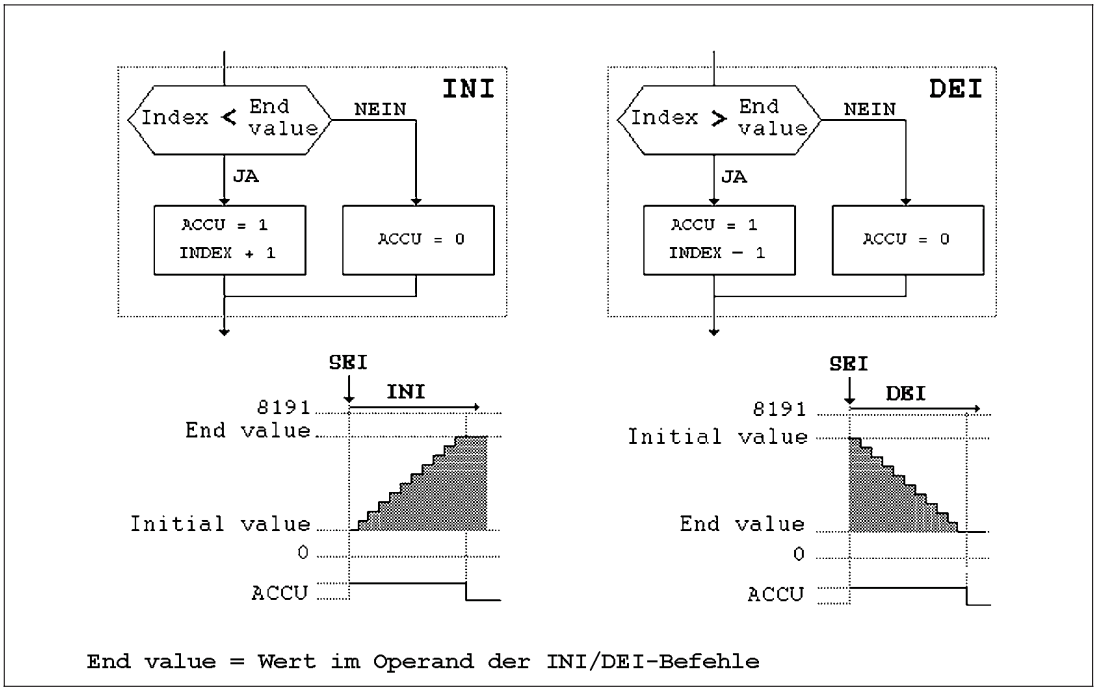
```
DEI   K 100   ; IR -1, solange IR > 100

DEI   R 444   ; IR -1, solange IR > als
              ;    Wert in Register R 444 ist
```

FLAGS: **ACCU** = H, wenn IR **grösser** als Wert im Operand.
ACCU = L, wenn IR **gleich oder kleiner** als der Wert im Operand oder im adressierten Register.

ANMERKUNG: Wird mit der Indexierung gearbeitet, ist der XOB 12 zu programmieren.

SIEHE AUCH: INI, SEI



Verhalten des Indexregisters und des ACCU bei INI/DEI

STI SPEICHERE (RETTE) DAS INDEXREGISTER (Store Indexregister)

BESCHREIBUNG: Der aktuelle Stand des Indexregisters (IR) wird in das im Operand angegebene PCD-Register geladen.
Mit dem Befehl RSI kann dieser Wert aus dem PCD-Register wieder ins IR zurückgeladen werden.

AUFBAU:

```
STI          Register          ; R 0-4095
```

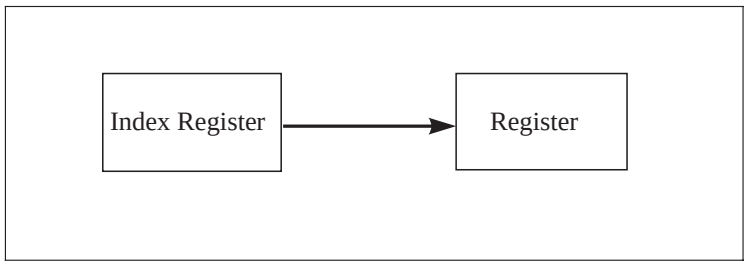
BEISPIEL:

```
STI     R 100   ; Lädt (rettet) das IR in
                ;    das PCD-Register R 100
```

FLAGS: Unverändert

ANMERKUNG: Wird mit der Indexierung gearbeitet, ist der XOB 12 zu programmieren.

SIEHE AUCH: RSI



RSI HOLE DAS INDEXREGISTER ZURÜCK (Restore Indexregister)

BESCHREIBUNG: Lädt das Indexregister (IR) mit dem Wert des im Operanden angegebenen PCD-Registers. Dieser Wert wurde i.a. vorher mit dem Befehl STI vom IR in dieses PCD-Register gerettet.
Der max. Wert im PCD-Register darf 8191 nicht übersteigen, da nur die letzten 13 Bit ins IR übertragen werden.
Wird versucht, einen Wert > 8191 oder < 0 zu laden, wird der XOB 12 aufgerufen.

AUFBAU:

RSI	**Register**	**; R 0-4095**

BEISPIEL:

```
        RSI     R 100   ; Lädt das IR mit dem Wert
                        ;    des PCD-Registers R 100
```

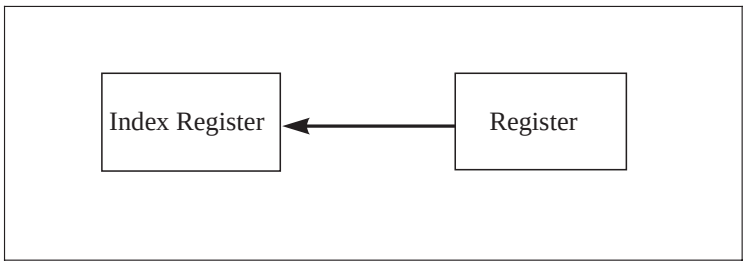
ist das gleiche wie

```
        SEI     R 100   ; (SEI aus Register wurde
                        ;    später eingeführt)
```

FLAGS: Unverändert

ANMERKUNG: Wird mit der Indexierung gearbeitet, ist der XOB 12 zu programmeren.

SIEHE AUCH: STI, SEI.

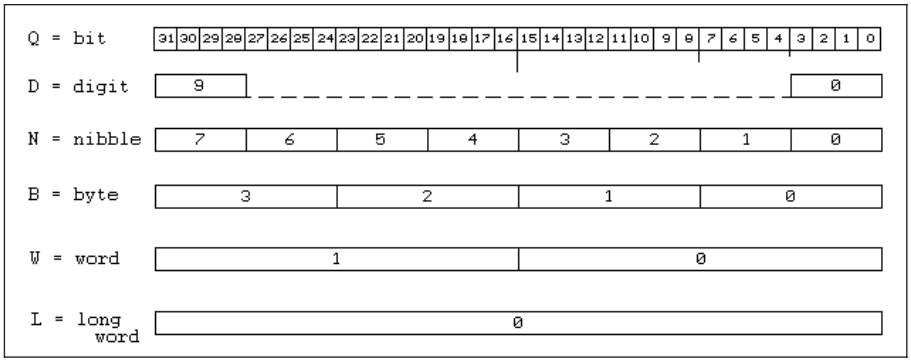


MOV VERSCHIEBE DATEN (Move Data)

BESCHREIBUNG: Verschiebt Daten von Registern, Timern und Countern in Register. Der Befehl besteht aus 4 Zeilen.
In der 1. Zeile wird das Quell-Element (R, T, C),
in der 2. Zeile das Datenformat und die Position der Daten im Quell-Element,
in der 3. Zeile das Ziel-Element (R) und
in der 4. Zeile das Datenformat und die Position der Daten im Ziel-Register angegeben.

Datenformat	Position
Q = Bit (Quantum)	0-31
D = Digit (4 Bit BCD)	0-9
N = Nibble (4 Bit Binär)	0-7
B = Byte (8 Bit)	0-3
W = Word (16 Bit)	0/1
L = Long word (32 Bit)	0

Das Datenformat in der 2. und in der 4. Zeile muss dasselbe, die Positionen können verschieden sein. Es können keine negativen Werte verschoben werden.



AUFBAU:

```
MOV[X]   Quell-Element(i)    ; R 0-4095, T 0-450, C 0-1599
         Format, Position    ; Q|D|N|B|W|L + Position
         Ziel-Element (i)    ; R 0-4095
         Format, Position    ; Q|D|N|B|W|L + Position
```

BEISPIEL: Verschiebe das höchste Nibble (N7) des Registers 100 zum niedrigsten Nibble (N0) des Registers 101.

```
;vor MOV:   R100:  1111 1010 1010 1010 1010 1010 1010 1010
            R101:  0001 0001 0001 0001 0001 0001 0001 0001

            MOV    R 100
                   N 7
                   R 101
                   N 0

;nach MOV:  R100:  1111 1010 1010 1010 1010 1010 1010 1010
            R101:  0001 0001 0001 0001 0001 0001 0001 1111
```

FLAGS: Unverändert

ANMERKUNG: Für das Kopieren eines GANZEN Registers, Timers oder Counters, wie dies in der Praxis auch meistens vorkommt, wird vorteilhafterweise der Befehl COPY verwendet. COPY hat eine einfachere Schreibweise, belegt nur 2 Programmzeilen und ist erst noch schneller.

SIEHE AUCH: COPY, GET, PUT, LD, LDL, LDH.

COPY KOPIERE DATEN (Copy Data)

BESCHREIBUNG: COPY, GET und PUT sind einander sehr ähnlich, was das Kopieren von Daten zwischen Elementen betrifft.
Die Befehle unterstützen den INDEXIERTEN Datentransfer zwischen Registern, Timern und Countern.
Es wird immer das GANZE Register, der GANZE Timer oder der GANZE Counter kopiert.
Für COPYX sind BEIDE Programmzeilen, Quelle und Ziel, indexiert.

AUFBAU:

COPY[X]	**Quell-Element(i)**	**; R 0-4095, T0-450, C0-1599**
	Ziel-Element(i)	**; R 0-4095, T0-450, C0-1599**

BEISPIEL:

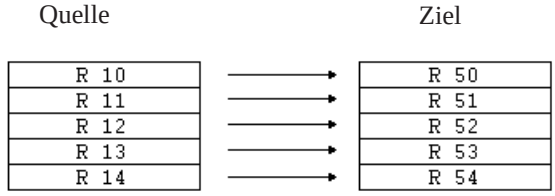
```
        COPYX   R 10   ; kopiert R 10 + Wert in IR
                R 50   ;     in R 50 + Wert in IR
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Wert im Register, Timer oder Counter gesetzt.

ANMERKUNG: Dieser Befehl wird, neben der universellen allg. Verwendung, vorallem für die indexierte Mathematik, die selbst nicht indexierbar ist, eingesetzt.

SIEHE AUCH: GET, PUT, MOV.

```
        SEI     K 0
loop:   COPYX   R 10
                R 50
        ......
        INI     K 4
        JR      H loop
```



GET HOLE DATEN (Get Data)

BESCHREIBUNG: COPY, GET und PUT sind einander sehr ähnlich, was das Kopieren von Daten zwischen Elementen betrifft.
Die Befehle unterstützen den INDEXIERTEN Datentransfer zwischen Registern, Timern und Countern.

Es wird immer das GANZE Register, der GANZE Timer oder der GANZE Counter kopiert.

Für **GETX** ist nur die ERSTE Programmzeile (Quelle) indexiert.
Zusätzlich zum erwähnten Datentransfer zwischen R, T, C kann der Befehl GET auch zum Kopieren von Datenblocks (DB) und Anwendertexten (X) zu Registern verwendet werden.

AUFBAU:

GET[X]	**Quell-Element (i)**	**; R, T, C, DB 0-3999, X 0-3999**
	Ziel-Element	**; R 0-4095,T0-450, C0-1599**

BEISPIEL:

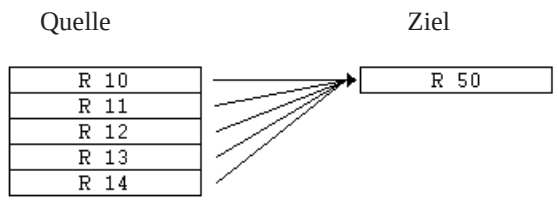
```
        GETX  R 10    ; kopiert R 10 + Wert in IR
              R 50    ; in Register R 50
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Wert im Register, Timer oder Counter gesetzt.

ANMERKUNG: Dieser Befehl wird für die indexierte Mathematik, die selbst nicht indexierbar ist, eingesetzt.

SIEHE AUCH: PUT, COPY, MOV.

```
        SEI   K 0
loop:   GETX  R 10
              R 50
        ......
        INI   K 4
        JR    H loop
```



Austausch von Information zwischen Datenblocks/Texten und Registern

Mit dem Befehl GET[X] kann auch Information aus Datenblocks (DB) und Anwendertexten (X) zu Registern übertragen werden.

Datenblocks sind PCD-Resourcen zum Speichern von Registern, Countern und Timern. Datenblocks belegen die gleichen Adressen wie die Texte und sind auch an der gleichen Stelle im Anwenderspeicher abgelegt wie Texte. Jedem Prozessor stehen folgende Adressen zur Verfügung:

TEXT	: 0 ... 3999	Ein- und dieselbe Adresse kann nur
DATENBLOCK	: 0 ... 3999	entweder als Text oder als Datenblock definiert werden.

DATA BLOCK und TEXT Formate

Formate der TEXTE und DATA-BLOCKS:

DB Nummer '[' [Länge] ']' [Wert [,Wert] ...] [,Kommentar]

TEXT Nummer '[' [Länge] ']' "Text Linie 1"[; Kommentar]
["Text Linie 2" [;Kommentar]]
["Text Linie n"[;Kommentar]]

Nummer, Länge:
- dezimal
- symbolisch

Wert:
- dezimal
- hexadezimal
- binär
- Fliess Punkt
- ASCII
- symbolisch

Texte bzw. Data-Blocks können folgende maximale Längen aufweisen:

TEXTE : je 3069 ASCII-Charakter (der erste Textcharakter darf nicht <254> oder <255> sein)

DATA-BLOCKS : je 383 Werte (ein Wert entspricht dem Inhalt eines Registers, Counters oder Timers).

Beispiele:

```
DB    100   [15]                    ; im DB 100 sind 15 Werte
                                    ;   "reserviert" (Werte = 0)

DB    101   [12]  10, 55, 777, 2.5e-6 ; 6 Werte werden geladen, die
                  2abh, 'z'         ;   restlichen Werte bis 12
                                    ;   sind = 0

DB    102   [ ] 1,2,3, 4, 5, 6, 7, 8  ; Die Länge des DB ergibt sich
                9, 10, 11, 12       ;   aus der Anzahl Werte. Die
                10101010011111y,    ;   Eingabe kann sich über mehrere
                0ffffh, 'abcd',     ;   Zeilen erstrecken. Jede Zeile
                toto, titi, -456,   ;   ist mit einem Komma zu
                45.6789, -45e12     ;   beenden. Nach Komma: Space
                                    ;   oder kein Space. Die Symbole
                                    ;   müssen definiert werden.

DB    elsa  [maria]                 ; DB- bzw. TEXT-Nummer und deren
                                    ;   Länge können symbolisch
                                    ;   geschrieben werden.

TEXT  120   "normaler Text <10><13>"

TEXT  121   [150]                   ; dieser Text besteht aus
                                    ;   150 Spaces (Leerschlägen)

TEXT  122 [5] "xyz"                 ; Text: xyz plus 2 Spaces
```

GET Data-Block/Text

Beispiel 1:

Data-Block gemäss Definition im Quellprogramm:

```
DB    100          [5] 0h,1h,2h,0a5a5a5a5h,720h
```

Befehl:

```
GET     DB  100     ; Kopiere Data-Block 100
        R  1000     ;    in Register 1000 und folgende
```

Resultat:

Register	Hex.-Wert
1000	00000000
1001	00000001
1002	00000002
1003	a5a5a5a5
1004	00000720

Beispiel 2:

Text gemäss Definition im Quellprogramm:

```
TEXT    100       "DIES IST EIN TEXT"
```

Befehl:

```
GET     X  100      ; Kopiere den Text 100
        R  1000     ;    in Register 1000 und folgende
```

Resultat:

(Die Register 1000 - 1004 waren vor der Ausführung des Befehls =0)

Register	ASCII-Wert	**Hex.-Wert**
1000	DIES	68736984
1001	IST	32738384
1002	EIN	32697378
1003	TEX	32846988
1004	T	84000000

↑
Ende der Übertragung

PUT BRINGE DATEN (Put Data)

BESCHREIBUNG: COPY, GET und PUT sind einander sehr ähnlich, was das Kopieren von Daten zwischen Elementen betrifft.
Die Befehle unterstützen den INDEXIERTEN Datentransfer zwischen Registern, Timern und Countern.
Es wird immer das GANZE Register, der GANZE Timer oder der GANZE Counter kopiert.

Für **PUTX** ist nur die ZWEITE Programmzeile (Ziel) indexiert.

Zusätzlich zum erwähnten Datentransfer zwischen R, T, C kann der Befehl GET auch zum Kopieren von Registern zu Data-Blocks (DB) und Anwendertexten (X) verwendet werden.

AUFBAU:

PUT[X]	**Quell-Element**	**; R 0-4095,T0-450, C0-1599**
	Ziel-Element (i)	**; R, T, C, DB 0-3999, X 0-3999**

BEISPIEL:

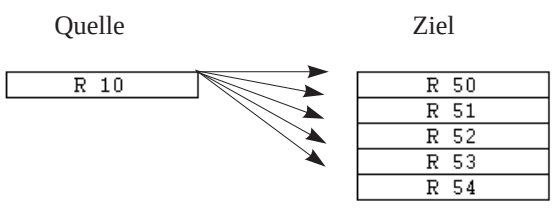
```
PUTX   R 10    ; kopiert Register R 10
       R 50    ;    in R 50 + Wert in IR
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Wert im Register, Timer oder Counter gesetzt.

ANMERKUNG: Dieser Befehl wird für die indexierte Mathematik, die selbst nicht indexierbar ist, eingesetzt.
Mit dem Befehl PUT[X] kann auch der Inhalt eines oder einer Reihe von Registern (auch Timer oder Counter) in Data-Blocks oder Texten übertragen werden.
Wird ein leeres Register (Wert = 0) in einen Text oder Data-Block, die ja beide im Textsegment liegen, geladen, würde dies automatisch das Textende signalisieren (ASCII NUL) und alle nachfolgende Information unterdrücken. Es wird deshalb steuerungsintern ein 00H zu einem 20H (32 dez.), d.h. zu einem Space gewandelt. Dies ist der Grund, dass in einem DB doppelt soviel Speicherplatz belegt wird, als die Information tatsächlich benötigt.

SIEHE AUCH: GET, COPY, MOV.

```
        SEI    K 0
loop:   PUTX   R 10
               R 50
        .....
        INI    K 4
        JR     H loop
```



PUT

PUT Data-Block/Text

WICHTIG: Mit dem Befehl PUT kann die Länge eines Data-Blocks oder Textes NICHT erweitert werden.
Der Transfer von Information mit PUT funktioniert nur, wenn als Datenspeicher ein nicht schreibgeschütztes RAM eingesetzt ist. PUT kann mit EPROM-Speichern nicht verwendet werden.

Beispiel 1:

Data-Block gemäss Definition im Quellprogramm:

```
DB   100        [5]    ; leer
```

Inhalt von 5 Registern:

Register	Dez.-Wert
1000	00000001
1001	00000002
1002	00000003
1003	01234567
1004	00000720

Befehl:

```
PUT     R  1000    ;Kopiere Register R 1000 und
        DB  100    ;   folgende in Data Block DB 100
```

Resultat: angezeigt im Debugger

```
DB  100 (0):   1          2          3          1234567          720
```

Beispiel 2:

Text gemäss Definition im Quellprogramm:

```
TEXT   100      [17]   ; Text mit 17 Spaces (Leerschlägen)
```

Inhalt von 5 Registern:

Register	ASCII-Wert	Hex.-Wert
1000	DIES	68736984
1001	IST	32738384
1002	EIN	32697378
1003	TEX	32846988
1004	T	84000000

Befehl:

```
PUT     R  1000    ; Kopiere Register R 1000 und
        X  100     ;   folgende in Text 100
```

Resultat: angezeigt im Debugger

```
TEXT 100:      "DIES IST EIN TEXT"
```

TFR TRANSFER DATEN *)

BESCHREIBUNG: Der Befehl ermöglicht den indexierten Datentransfer von einzelnen Werten eines Datenblocks in Register, Timer oder Counter und umgekehrt.

AUFBAU 1: Von einem Daten-Block einen einzelnen Wert (32 Bit) in ein Register, Timer oder Counter kopieren.

TFR[X]	**Quelle**	**; DB (oder X) 0-3999, 4000-7999**
	Position	**; R 0-4095, K 0-382, 0-16383**
	Ziel (i)	**; R 0-4095, T/C 0-1599**

Die Adressierung (Position) eines einzelnen Wertes in einem DB kann direkt mit einer Konstanten oder indirekt über ein Register erfolgen.

Daten-Block-Länge:
Standardspeicher DB 0-3999 → max. Länge/DB = 383 Werte
Erweiterungsspeicher DB 4000-7999 → max. Länge/DB = 16’384 Werte

Ein Wert (32 Bit) entspricht dem Inhalt eines Registers, Timers oder Counters.

BEISPIEL 1:

```
TFR   DB 4010   ; Kopiert aus Datenblock 4010
      K 13      ; den Wert an Position 13
      R 26      ; in Register 26
```

AUFBAU 2: Ein Register, Timer oder Counter in einen Daten-Block kopieren.

TFR[X]	**Quelle (i)**	**; R 0-4095, T/C 0-1599**
	Position	**; DB (oder X) 0-3999, 4000-7999**
	Position	**; R 0-4095, K 0-382, 0-16383**

BEISPIEL 2:

```
TFR   R 120     ; Kopiert Register 120
      DB 4025   ; in Daten-Block 4025
      K 6       ; an Position 6
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Wert im Register, Timer oder Counter gesetzt.

*) ab Utility-Version: → V1.7

ab Firmware-Version:

PCD6.M1../M2..	→ V007
PCD6.M540	→ V002
PCD4.M..	→ V003
PCD6.P800	→ V004
PCD2	→ alle

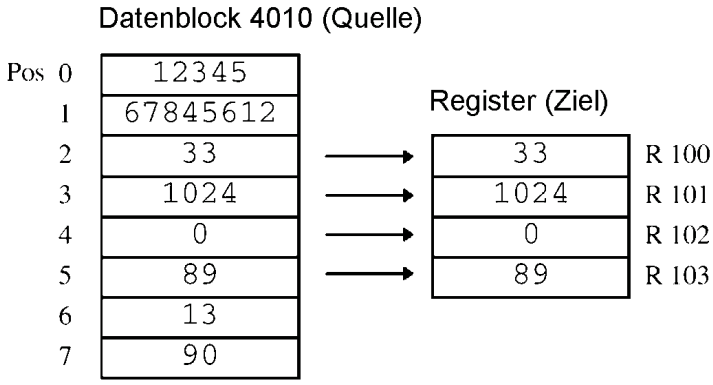
TFR

ANMERKUNG: Aus Speicher-Organisationsgründen erfolgt der Zugriff auf DBs von 4000-7999 wesentlich schneller als auf DBs von 0-3999. Es empfiehlt sich deshalb, den Befehl hauptsächlich auf DBs von 4000-7999 anzuwenden.

SIEHE AUCH: PUT, GET

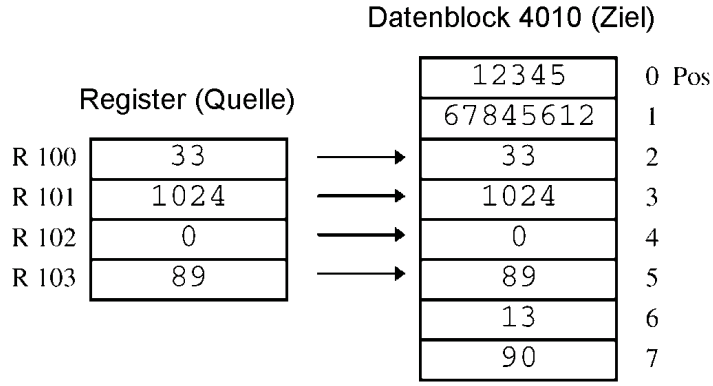
ANWENDUNG: Aus Datenblock 4010 werden die 4 Werte von Position 2-5 in Register 100-103 kopiert.

```
          ....
          LD     R   999
                     2
          SEI    K   0
LOOP:     TFRX   DB  4010
                 R   999
                 R   100
          INC    R   999
          INI    K   3
          JR     H   LOOP
          ...
```



Register 100-103 werden in Datenblock 4010 an Position 2-5 kopiert.

```
          ....
          LD     R   999
                     2
          SEI    K   0
LOOP:     TFRX   R   100
                 DB  4010
                 R   999
          INC    R   3
          JR     H   LOOP
          ...
```



TFRI Transfer Daten indirekt

Beschreibung: Dieser Befehl ermöglicht den indirekten Datentransfer von einzelnen Werten eines Datenblocks in Register, Timer oder Counter und umgekehrt.

Der Befehl kann nicht indexiert und auch nicht als Parameter an einen FB übergeben werden

Aufbau: **TRANSFER DATA BLOCK (oder Text) → R, T, C**
Von einem Daten-Block einen einzelenen Wert (32 Bit) in ein Register, Timer oder Counter kopieren.

```
TFRI    Quelle      ; DB (oder X) 'x'
        Position    ; R 0..4095, K 0..382, K 0..16383
        Ziel        ; R or T|C 'y'
```

Der 1. Operand zeigt an, ob die Quelle ein DB oder ein Text ist. Die Variable 'x' ist die Adresse eines Registers, welches die Adresse des DB oder des Texts enthält.
Der 2. Operand zeigt die Position des Wertes innerhalb des DB oder des Texts. Dieser Wert kann mittels einer Konstanten oder indirekt in einem Register definiert werden.
Der 3. Operand zeigt das Zielmedium (R oder T/C). Die Variable 'y' ist die Adresse eines Registers, welches die Adresse des Zielmediums enthält.

TRANSFER R, T, C → DATA BLOCK (oder TEXT)
Kopieren eines Registers, Timers oder Counters in einen DB (oder Text)

```
TFRI    Quelle      ; R or T|C 'x'
        Ziel        ; DB ( oder X) 'y'
        Position    ; R 0..4095, K 0..382, K 0..16383
```

Der 1. Operand zeigt den Typ der Quelle (R oder T/C). Die Variable 'x' ist die Adresse des Registers, welches die Quelldaten enthält.
Der 2. Operand spezifiziert den Typ des Zielmediums (DB oder Text). Die Variable 'y' ist die Nummer des Registers, welches die Ziel-Adresse enthält.
Der 3. Operand zeigt die Position innerhalb des DB (oder Texts) wohin der Wert zu transferieren ist. Dieser Wert kann mittels einer Konstanten oder indirekt in einem Register definiert werden.

Bemerkung: Die Länge eines DB ist vom PCD-Speicher abhängig

Speicher	Addresse	Max. Länge eines DB
Standard	DB 0..3999	383 Werte
Erweitert	DB 4000..7999	16383 Werte

Beispiele: Übertragen des Elementes auf Position 10 des Datenblock 4000 ins Register 4095.

```
LD      R  100      ; Initialisierung der DB-Adresse
           4000
LD      R  101      ; Initialisierung der Register-Adresse
           4095
TFRI    DB 100      ; Transfer des DBs
        K  10
        R  101
```

Übertragen des Counters 1000 auf die Position 50 des DB 4000

```
LD      R  100      ; Initialisierung der DB-Adresse
           4000
LD      R  101      ; Initialisierung der Position
           50
LD      R  102      ; Initialisierung der Counter-Adresse
           4095
TFRI    C  102      ; Transfer des Counters
        DB 100
        R  101
```

Flags: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt.

Siehe auch: TFR, PUT, GET

Bemerkung: Aus Speicher-Organisationsgründen erfolgt der Zugriff auf DBs von 4000..7999 wesentlich schneller als auf DBs von 0..3999. Es empfiehlt sich deshalb, den Befehl hauptsächlich auf DBs von 4000.7999 anzuwenden.

BITI EINZEL-BIT IN REGISTER, PCD-FORMAT (Bit In)

BESCHREIBUNG: Eine Reihe von Einzel-Bit ab Eingängen, Ausgängen oder Flags oder der Wert eines Timers oder Counters werden in ein Register übertragen.

Der Befehl besteht aus 3 Zeilen.

In der 1. Zeile wird die Anzahl der zu übertragenden Bit angegeben. (1 - 32).

In der 2. Zeile wird der Typ und die Adresse des Quell-Elementes definiert.

Bei 1-Bit Elementen wird die niedrigste Adresse der Element-Reihe angegeben. Dieses Bit wird im Zielregister zum LSB.

In der 3. Zeile wird das Ziel-Register bezeichnet.

Diese Adressierung ist umgekehrt zur PCA!

AUFBAU:

```
BITI[X]   Anzahl Bit          ; 1 - 32
          Quelle              ; I, O, F, T, C
          Ziel-Register (i)   ; Register R 0-4095
```

BEISPIEL:

```
        BITI    16      ; Anzahl Bit
                I 32    ;   ab Eingang I 32 - 47
                R 0     ;   in Register R 0
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt.

SIEHE AUCH: BITIR, DIGI, DIGIR.

ANWENDUNG: Wird Eingang 8 = H, soll der 8-Bit Binärwert, der an den Eingägen 0 - 7 anliegt, gelesen und in Register 500 abgelegt werden.

```
        COB     0
                0

        .....
        STH     I 8     ; Ist Eingang I 8 = H
        DYN     F 100
        JR      L NEXT

        BITI     8      ;   Lese 8 Bit
                 I 0    ;       ab Eingang I 0 (-7)
                 R 500          in Register R 500

NEXT:   ....

        ECOB
```

BITIR EINZEL-BIT IN REGISTER, PCA-FORMAT (Bit In Reversed)

BESCHREIBUNG: Eine Reihe von Einzel-Bit ab Eingängen, Ausgängen oder Flags oder der Wert eines Timers oder Counters werden in ein Register übertragen.

Der Befehl besteht aus 3 Zeilen.

In der 1. Zeile wird die Anzahl der zu übertragenden Bit angegeben. (1-32).

In der 2. Zeile wird der Typ und die Adresse des Quell-Elementes definiert. Bei 1-Bit Elementen wird die niedrigste Adresse der Element-Reihe angegeben. Dieses Bit wird im Zielregister zum MSB.

In der 3. Zeile wird das Ziel-Register bezeichnet.

Diese Adressierung entspricht derjenigen der PCA.

AUFBAU:

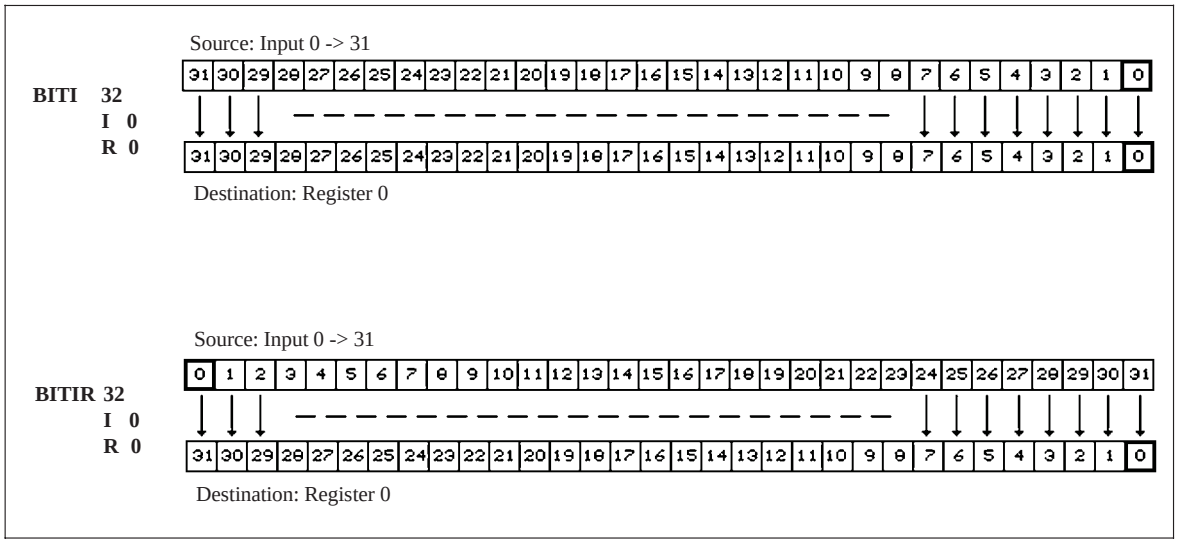
BITIR[X]	**Anzahl Bit**	**; 1 - 32**
	Quelle	**; I, O, F, T, C**
	Ziel-Register (i)	**; Register R 0-4095**

BEISPIEL:

```
BITIR   16      ; Anzahl Bit
        I 32    ;   ab Eingang I 47 - 32
        R 15    ;   in Register R 15
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt.

SIEHE AUCH: BITI, DIGI, DIGIR.



Vergleich BITI - BITIR

BITO REGISTER ZU EINZEL-BIT, PCD-FORMAT (Bit Out)

BESCHREIBUNG: Aus einem Register werden eine Reihe von Einzel-Bit auf Ausgänge oder Flags oder in Timer oder Counter übertragen.

Der Befehl besteht aus 3 Zeilen.

In der 1. Zeile wird die Anzahl der zu übertragenden Bit angegeben. (1 - 32).

In der 2. Zeile wird das Quell-Register, aus welchem die Information ausgegeben werden soll, angegeben.

In der 3. Zeile wird die niedrigste Adresse der Reihe von Ausgängen oder Flags bzw. die Adresse des Timers oder Counters angegeben, die das Ziel darstellen. Sind Ausgänge oder Flags das Ziel, so wird die niedrigste Adresse zum LSB.

Diese Adressierung ist umgekehrt zur PCA!

AUFBAU:

BITO[X]	**Anzahl Bit**	**; 1 - 32**
	Quell-Register (i)	**; Register 0-4095**
	Ziel	**; O, F, T, C**

BEISPIEL:

```
BITO     8     ; Anzahl Bit
         R 10  ;   ab Register R 10
         O 48  ;   zu Ausgängen O 48 - 55
```

FLAGS: Unverändert

SIEHE AUCH: BITOR, DIGO, DIGOR.

ANWENDUNG: Kopiere die logischen Zustände der Eingänge 0 - 15 zu den Ausgängen 32 - 47

```
COB    0
       0

BITI    16    ; Lese 16 Bit
        I 0   ;   ab Eingängen I 0 - 7
        R 0   ;   in Register R 0

BITO    16    ; Schreibe 16 Bit
        R 0   ;   aus Register R 0
        O 32  ;   auf die Ausgänge 32-47

ECOB          ;   (R 0 = Hilfsregister)
```

BITOR REGISTER ZU EINZEL-BIT, PCA-FORMAT (Bit Out Reversed)

BESCHREIBUNG: Aus einem Register werden eine Reihe von Einzel-Bit auf Ausgänge oder Flags oder in Timer oder Counter übertragen.

Der Befehl besteht aus 3 Zeilen.

In der 1. Zeile wird die Anzahl der zu übertragenden Bit angegeben. (1 - 32).

In der 2. Zeile wird das Quell-Register, aus welchem die Information ausgegeben werden soll, angegeben.

In der 3. Zeile wird die niedrigste Adresse der Reihe von Ausgängen oder Flags bzw. die Adresse des Timers oder Counters angegeben, die das Ziel darstellen. Sind Ausgänge oder Flags das Ziel, so wird die niedrigste Adresse zum MSB.

Diese Adressierung entspricht derjenigen der PCA:

AUFBAU:

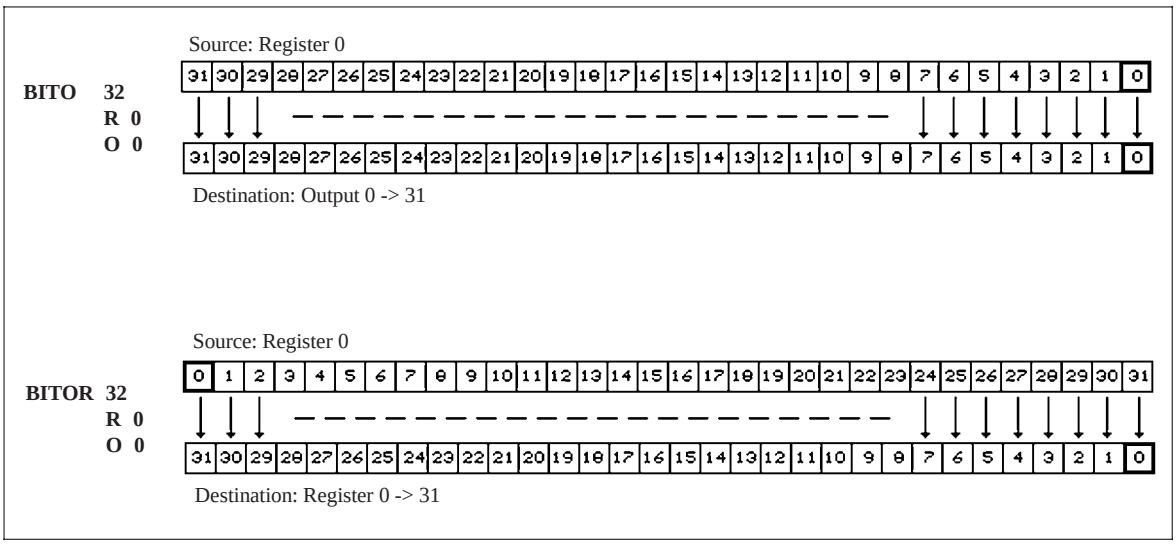
BITOR[X]	**Anzahl Bit**	**; 1 - 32**
	Quell-Register (i)	**; Register 0-4095**
	Ziel	**; O, F, T, C**

BEISPIEL:

```
BITOR    8      ; Anzahl Bit
         R 10   ;   ab Register R 10
         O 48   ;   zu Ausgängen O 55 - 48
```

FLAGS: Unverändert

SIEHE AUCH: BITO, DIGO, DIGOR.



Vergleich BITO - BITOR

DIGI DIGIT IN REGISTER, PCD-FORMAT (Digit In)

BESCHREIBUNG: Eine binär-codierte Dezimal-Information (BCD) wird ab Einzel-Bit Elementen, d.h. ab Eingängen, Ausgängen oder Flags in ein Register übertragen.
Ein BCD-Digit besteht aus 4 Bit (z.B. 4 Eingängen). Es können bis zu 9 ganze BCD-Digits in 1 Register geladen werden.
Der Befehl besteht aus 3 Zeilen.
In der 1. Zeile wird die Anzahl zu übertragender Digits angegeben. (1 - 10).
In der 2. Zeile wird der Typ und die Adresse der Quell-Elemente definiert: I,O,F.
In der 3. Zeile wird das Ziel-Register bezeichnet.
Die niedrigste Adresse der Elementen-Reihe wird zum LSB im Zielregister.
Diese Adressierung ist umgekehrt zur PCA!

AUFBAU:

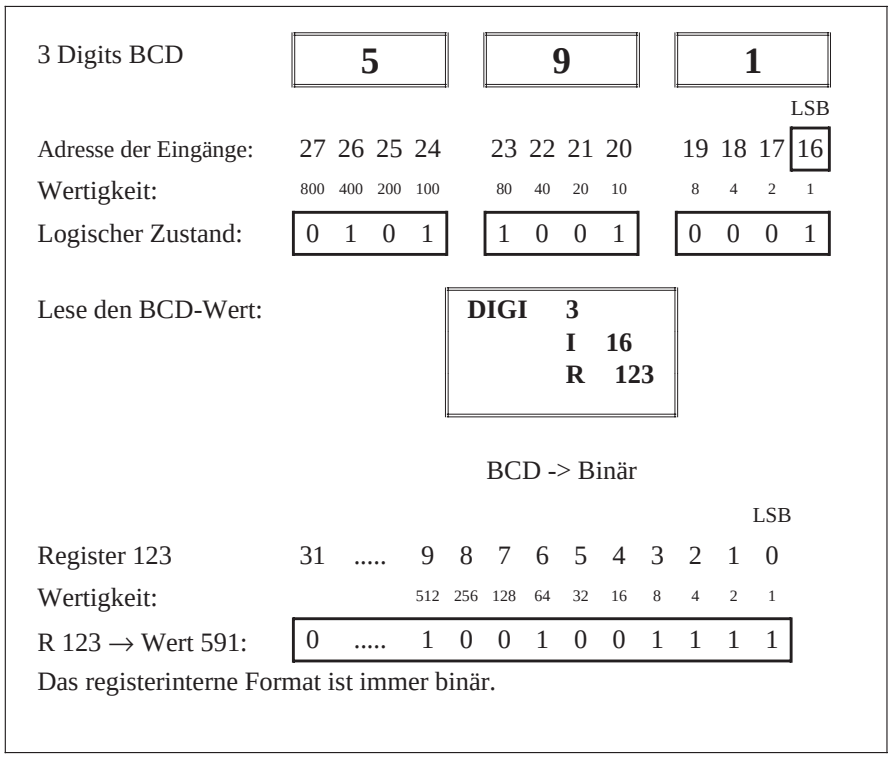
DIGI[X]	**Anzahl Digit**	**; 1 - 10**
	Quelle	**; I, O, F**
	Ziel-Register (i)	**; Register R 0-4095**

BEISPIEL:

```
DIGI    2       ; Anzahl Digits
        I 32    ;   ab I 39-36 und 35-32
        R 100   ;   in Register R 100
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt.

SIEHE AUCH: DIGIR, DIGO, DIGOR, BITI, BITIR.



Befehl DIGI

DIGIR DIGIT IN REGISTER, PCA-FORMAT (Digit In Reversed)

BESCHREIBUNG: Eine binär-codierte-dezimal-information (BCD) wird ab Einzel-Bit Elementen, d.h. ab Eingängen, Ausgängen oder Flags in ein Register übertragen.
Ein BCD-Digit besteht aus 4 Bit (z.B. 4 Eingängen). Es können bis zu 9 ganze BCD-Digits in 1 Register geladen werden.
Der Befehl besteht aus 3 Zeilen.
In der 1. Zeile wird die Anzahl zu übertragender Digits angegeben. (1 - 10).
In der 2. Zeile wird der Typ und die Adresse der Quell-Elemente definiert: I,O,F.
In der 3. Zeile wird das Ziel-Register bezeichnet.
Die niedrigste Adresse der Elementen-Reihe wird zum MSB im Zielregister.
Diese Adressierung entspricht derjenigen der PCA.

AUFBAU:

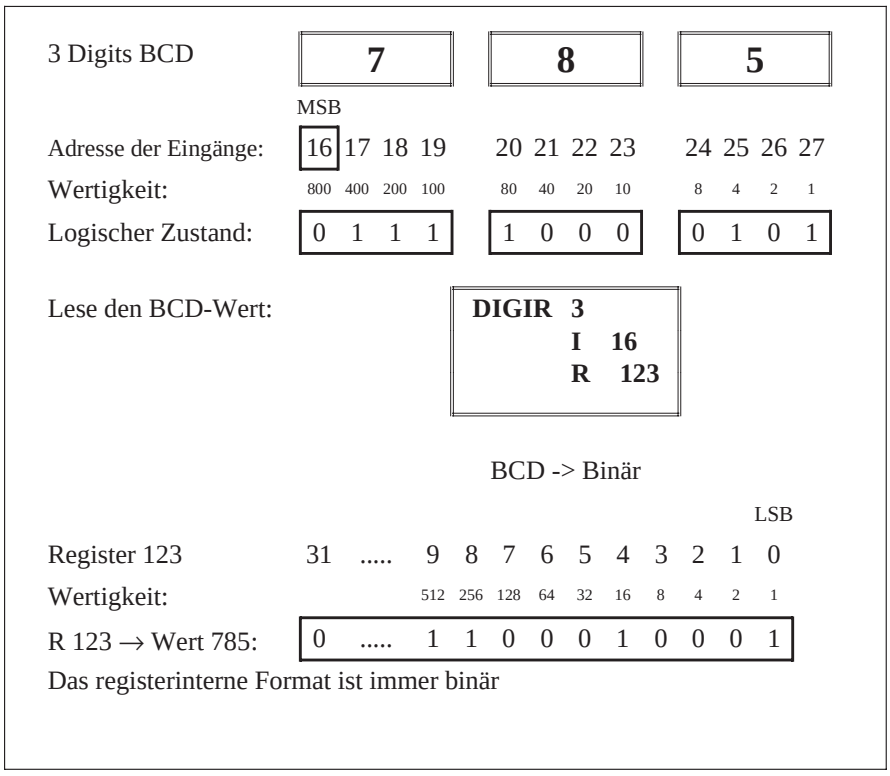
```
DIGIR[X]   Anzahl Digits      ; 1 - 10
           Quelle             ; I, O, F
           Ziel-Register (i)  ; Register R 0-4095
```

BEISPIEL:

```
DIGIR   2        ; Anzahl Digits
        I 32     ;   ab I 32-35 und 36-39
        R 100    ;   in Register R 100
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt.

SIEHE AUCH: DIGI, DIGO, DIGOR, BITI, BITIR.



Befehl DIGIR

DIGO REGISTER ZU DIGIT, PCD-FORMAT (Digit Out)

BESCHREIBUNG: Aus einem Register werden eine Reihe von Einzel-Bit im BCD-Format auf Ausgänge oder Flags übertragen.
Ein BCD-Digit besteht aus 4 Bit (z.B. 4 Ausgängen).
Der Befehl besteht aus 3 Zeilen.
In der 1. Zeile wird die Anzahl der zu übertragenden Digits angegeben. (1-10).
In der 2. Zeile wird das Quell-Register, aus welchem die Information ausgegeben werden soll. angegeben.
In der 3. Zeile wird die niedrigste Adresse der Reihe von Ausgängen oder Flags angegeben, die das Ziel bilden.
Die niedrigste Adresse der Elementen-Reihe wird zum LS-Digit und zum LS-Bit innerhalb des Digits.
Diese Adressierung ist umgekehrt zur PCA!

AUFBAU:

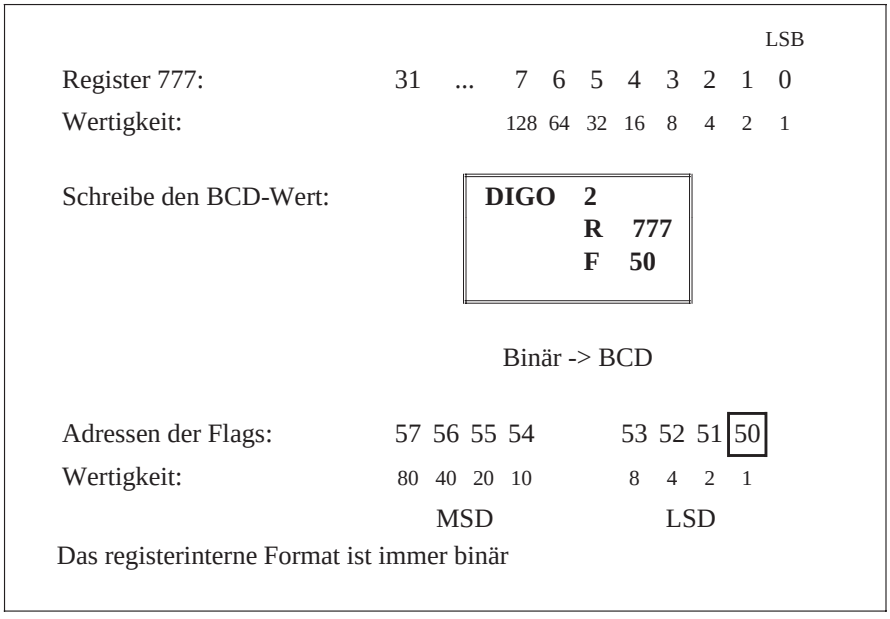
DIGO[X]	**Anzahl Digits**	**; 1 - 10**
	Quell-Register (i)	**; Register 0-4095**
	Ziel	**; O 0-8191, F 0-8191**

BEISPIEL:

```
DIGO    2       ; Anzahl Digits
        R 123   ;   ab Register R 123
        O 40    ;   zu O 47-44 und 43-40
```

FLAGS: Das Error-Flag wird gesetzt, wenn die angegebene Anzahl Digits ungültig ist.

SIEHE AUCH: DIGOR, DIGI, DIGIR, BITO, BITOR.



Befehl DIGO

DIGOR REGISTER ZU DIGIT, PCA-FORMAT (Digit Out Reversed)

BESCHREIBUNG: Aus einem Register werden eine Reihe von Einzel-Bit im BCD-Format auf Ausgänge oder Flags übertragen.
Ein BCD-Digit besteht aus 4 Bit (z.B. 4 Ausgängen).
Der Befehl besteht aus 3 Zeilen.
In der 1. Zeile wird die Anzahl der zu übertragenden Digits angegeben. (1-10).
In der 2. Zeile wird das Quell-Register, aus welchem die Information ausgegeben werden soll. angegeben.
In der 3. Zeile wird die niedrigste Adresse der Reihe von Ausgängen oder Flags angegeben, die das Ziel bilden.
Die niedrigste Adresse der Elementen-Reihe wird zum MS-Digit und zum MS-Bit innerhalb des Digits.
Diese Adressierung entspricht derjenigen der PCA.

AUFBAU:

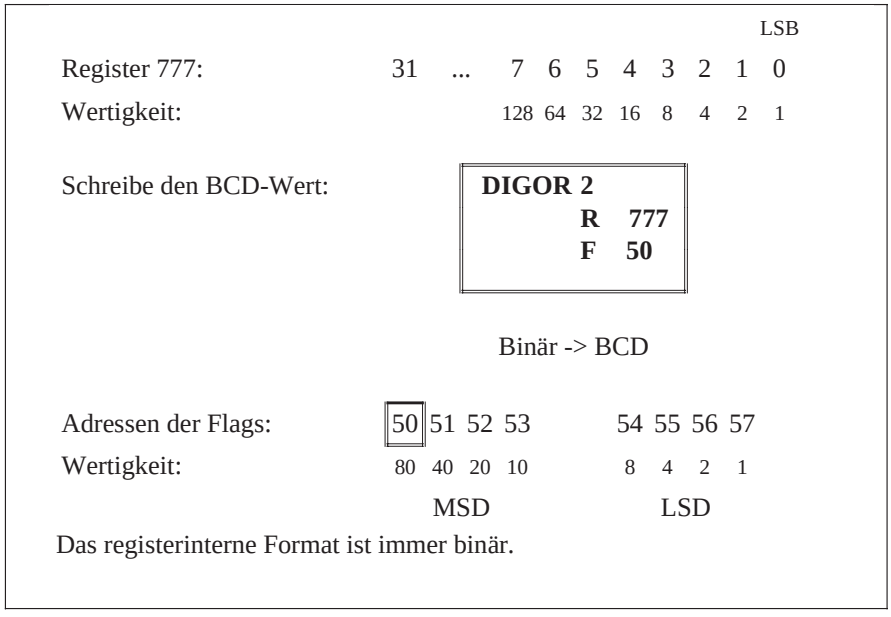
```
DIGOR[X]  Anzahl Digits      ; 1 - 10
          Quell-Register (i) ; Register R 0-4095
          Ziel               ; O 0-8191, F 0-8191
```

BEISPIEL:

```
DIGOR    2      ; Anzahl Digits
         R 123  ;   ab Register R 123
         O 40   ;   zu O 40-43 und 44-47
```

FLAGS: Das Error-Flag wird gesetzt, wenn die angegebene Anzahl Digits ungültig ist.

SIEHE AUCH: DIGO, DIGI, DIGIR, BITO, BITOR.



Befehl DIGOR

AND UND-VERKNÜPFUNG VON REGISTERN (AND Registers)

BESCHREIBUNG: Der Inhalt des Registers 1 (1. Zeile) wird mit dem Inhalt des Registers 2 (2. Zeile) UND-verknüpft und das Resultat im Register 3 (3. Zeile) abgelegt.

AUFBAU:

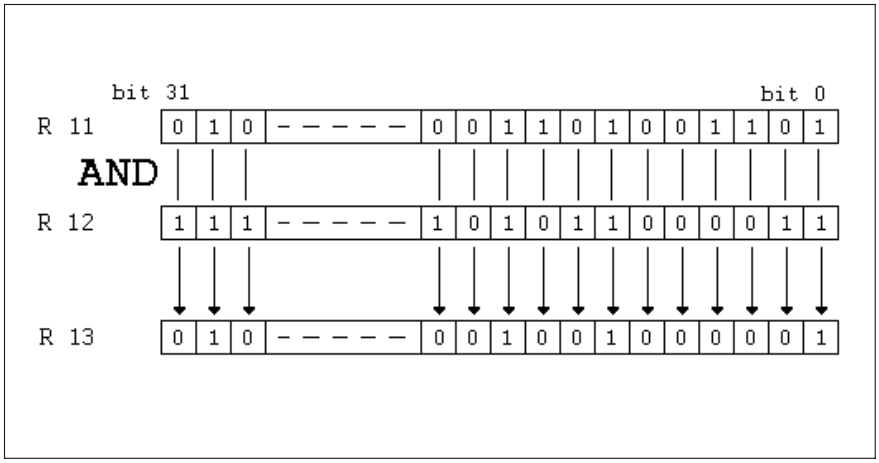
```
AND [X]   Wert 1    (i)   ; Register R 0-4095
          Wert 2          ; Register R 0-4095
          Resultat  (i)   ; Register R 0-4095
```

BEISPIEL:

```
        AND   R 11    ; Der Inhalt des R 11 wird
              R 12    ;    mit dem Inhalt des R 12
              R 13    ;    UND verknüpft und das
                      ;    Resultat in R 13 abgelegt
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem dezimalen Resultat gesetzt. Das ERROR-Flag wird immer zurückgesetzt.

SIEHE AUCH: OR, NOT, EXOR.



OR ODER-VERKNÜPFUNG VON REGISTERN (OR Registers)

BESCHREIBUNG: Der Inhalt des Registers 1 (1. Zeile) wird mit dem Inhalt des Registers 2 (2. Zeile) ODER-verknüpft und das Resultat im Register 3 (3. Zeile) abgelegt.

AUFBAU:

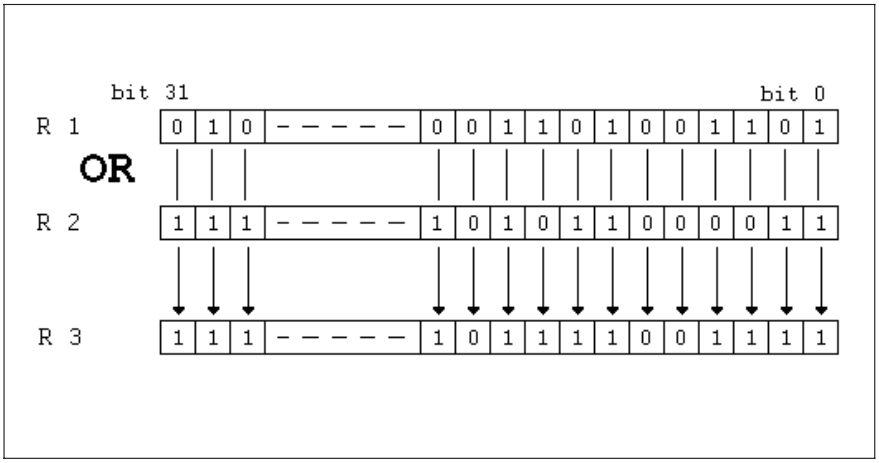
OR[X]	**Wert 1**	**(i)**	**; Register R 0-4095**
	Wert 2		**; Register R 0-4095**
	Resultat	**(i)**	**; Register R 0-4095**

BEISPIEL:

```
OR      R 1     ; Der Inhalt des R 1 wird
        R 2     ;    mit dem Inhalt des R 2
        R 3     ;    ODER verknüpft und das
                ;    Resultat in R 3 abgelegt
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem dezimalen Resultat gesetzt. Das ERROR-Flag wird immer zurückgesetzt.

SIEHE AUCH: EXOR.



EXOR EXOR-VERKNÜPFUNG VON REGISTERN (EXOR Registers)

BESCHREIBUNG: Der Inhalt des Registers 1 (1. Zeile) wird mit dem Inhalt des Registers 2 (2. Zeile) EXOR-verknüpft und das Resultat im Register 3 (3. Zeile) abgelegt.

AUFBAU:

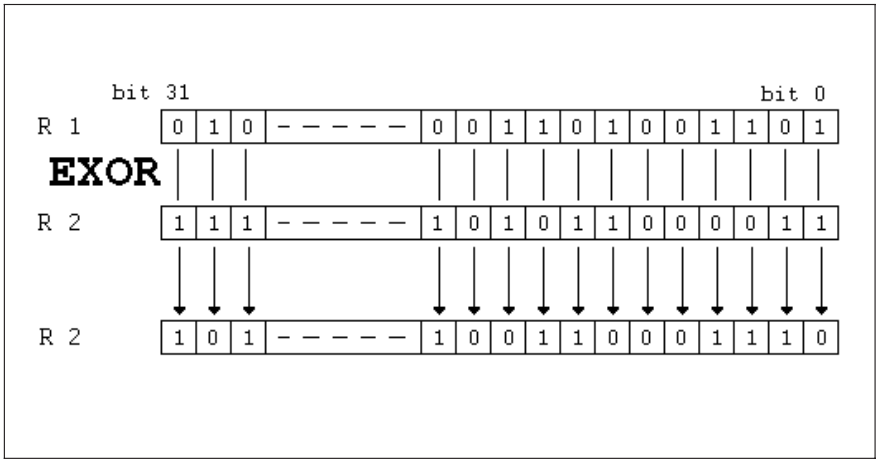
EXOR[X]	**Wert 1**	**(i)**	**; Register R 0-4095**
	Wert 2		**; Register R 0-4095**
	Resultat	**(i)**	**; Register R 0-4095**

BEISPIEL:

```
EXOR   R 20   ; Der Inhalt des R 20 wird
       R 21   ;    mit dem Inhalt des R 21
       R 50   ;    EXOR verknüpft und das
              ;    Resultat in R 50 abgelegt
```

FLAGS: Das Z-, das P- un das N-Flag werden gemäss dem dezimalen Resultat gesetzt. Das ERROR-Flag wird immer zurückgesetzt.

SIEHE AUCH: OR.



NOT KOMPLEMENTIERUNG EINES REGISTERS (Complement Registers)

BESCHREIBUNG: Der Inhalt des Registers 1 (1. Zeile) wird komplementiert (Einer-Komplement) und das Resultat im Register der 2. Zeile abgelegt.

AUFBAU:

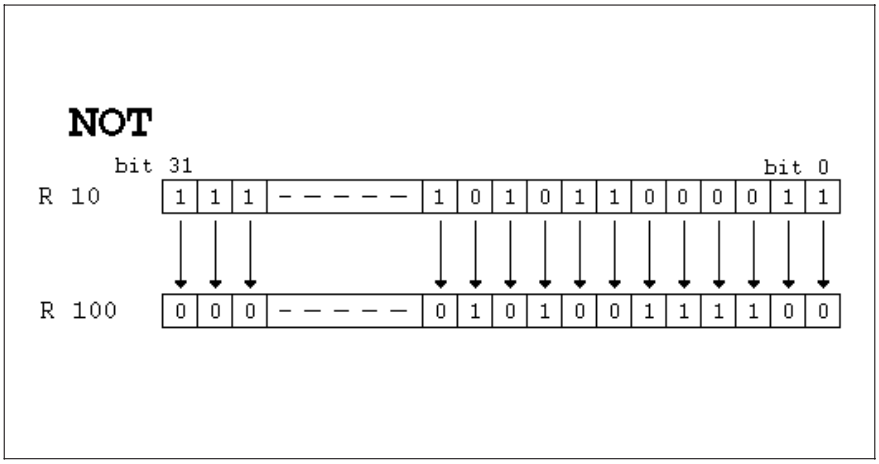
NOT[X]	**Wert**	**(i)**	**; Register R 0-4095**
	Resultat	**(i)**	**; Register R 0-4095**

BEISPIEL:

```
NOT     R 10    ; Der Inhalt des R 10 wird
        R 100   ;    komplementiert und das
                ;    Resultat in R 100 abgelegt
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem dezimalen Resultat gesetzt. Das ERROR-Flag wird immer zurückgesetzt.

SIEHE AUCH:



SHIU SCHIEBE REGISTER AUFWÄRTS (Shift Registers Up)

BESCHREIBUNG: Der Inhalt des definierten Blockes wird um 1 Adresse nach oben geschoben. Das Register in der 1. und das Register in der 2. Zeile bezeichnen den Anfang und das Ende des zu schiebenden Blocks.

Nach der Ausführung des Befehls enthält das Register des unteren Blockendes = 0. Der Inhalt des Registers des oberen Blockendes wird noch 1 Register weiter geschoben, also ins Register der 2. Zeile +1.

Ist die Adresse des Registers des oberen Blockendes kleiner als die Adresse des Registers des unteren Blockendes, wird die Operation mit vertauschten Adressen durchgeführt.

AUFBAU:

SHIU	**Anfangs-Register**	**; R 0-4094**
	End-Register	**; R 0-4094**

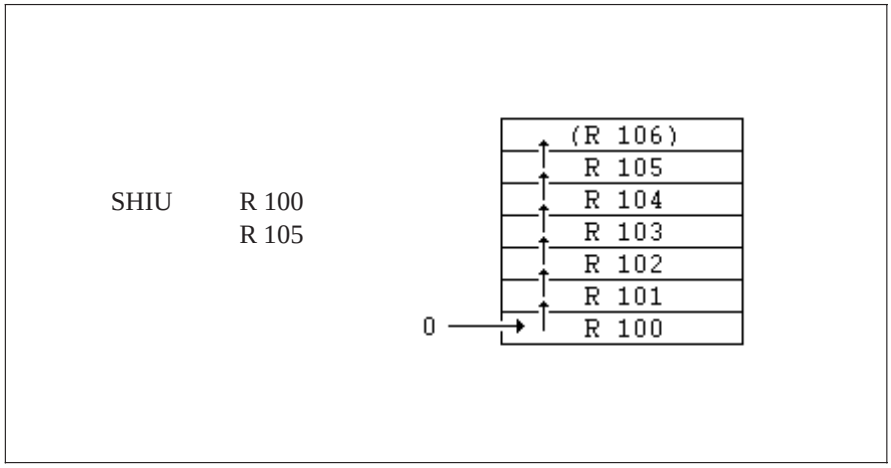
BEISPIEL:

```
SHIU    R 100   ; Verschiebt R 100 - R105
        R 105   ;    um 1 Adresse nach oben
```

FLAGS: Unverändert

ANMERKUNG: Bei diesem Befehl wird 1 Register mehr benützt, als definiert wird.

SIEHE AUCH: SHID, ROTU, ROTD.



SHID SCHIEBE REGISTER ABWÄRTS (Shift Registers Down)

BESCHREIBUNG: Der Inhalt des definierten Blockes wird um 1 Adresse nach unten geschoben. Das Register in der 1. und das Register in der 2. Zeile bezeichnen den Anfang und das Ende des zu schiebenden Blocks.

Nach der Ausführung des Befehls enthält das Register des oberen Blockendes = 0. Der Inhalt des Registers des unteren Blockendes wird noch 1 Register weiter geschoben, also ins Register der 2. Zeile -1.

Ist die Adresse des Registers des oberen Blockendes kleiner als die Adresse des Registers des unteren Blockendes, wird die Operation mit vertauschten Adressen durchgeführt.

AUFBAU:

SHID	**Anfangs-Register**	**; R 0-4095**
	End-Register	**; R 0-4095**

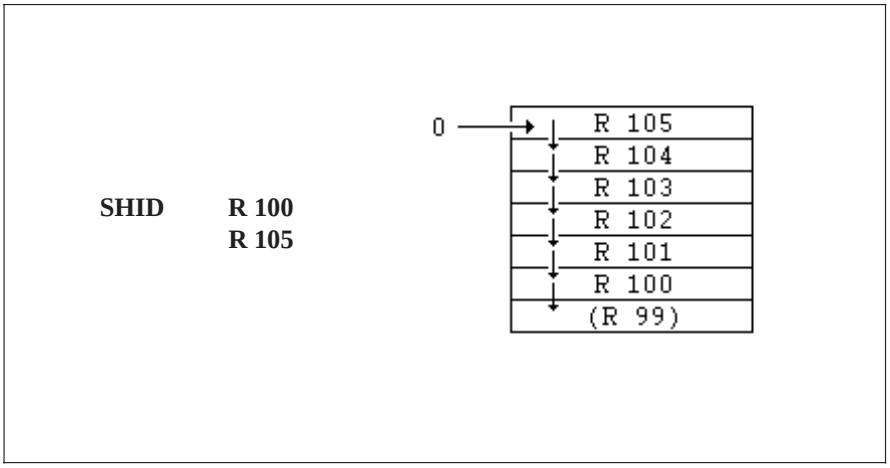
BEISPIEL:

```
SHID    R 100   ; Verschiebt R 100 - R 105
        R 105   ;    um 1 Adresse nach unten
```

FLAGS: Unverändert

ANMERKUNG: Bei diesem Befehl wird 1 Register mehr benützt, als definiert wird.

SIEHE AUCH: SHIU, ROTU, ROTD.



ROTU ROTIERE REGISTER AUFWÄRTS (Rotate Registers Up)

BESCHREIBUNG: Der Inhalt des definierten Blockes wird um 1 Adresse nach oben rotiert. Das Register in der 1. und das Register in der 2. Zeile bezeichnen den Anfang und das Ende des zu rotierenden Blocks.

Nach der Ausführung des Befehls enthält das Register des unteren Blockendes den Inhalt des Registers des oberen Blockendes.

Ist die Adresse des Registers des oberen Blockendes kleiner als die Adresse des Registers des unteren Blockendes, wird die Operation mit vertauschten Adressen durchgeführt.

AUFBAU:

ROTU	**Anfangs-Register**	**; R 0-4095**
	End-Register	**; R 0-4095**

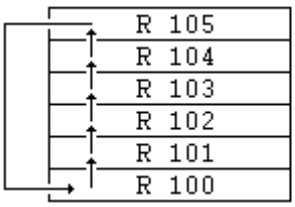
BEISPIEL:

```
ROTU   R 100   ; Rotiert R 100 - R 105
       R 105   ;    um 1 Adresse nach oben
```

FLAGS: Unverändert

SIEHE AUCH: ROTD, SHIU, SHID.

```
ROTU    R 100
        R 105
```



ROTD ROTIERE REGISTER ABWÄRTS (Rotate Registers Down)

BESCHREIBUNG: Der Inhalt des definierten Blockes wird um 1 Adresse nach unten rotiert. Das Register in der 1. und das Register in der 2. Zeile bezeichnen den Anfang und das Ende des zu rotierenden Blocks.

Nach der Ausführung des Befehls enthält das Register des oberen Blockendes den Inhalt des Registers des unteren Blockendes.

Ist die Adresse des Registers des oberen Blockendes kleiner als die Adresse des Registers des unteren Blockendes, wird die Operation mit vertauschten Adressen durchgeführt.

AUFBAU:

ROTD	**Anfangs-Register**	**; R 0-4095**
	End-Register	**; R 0-4095**

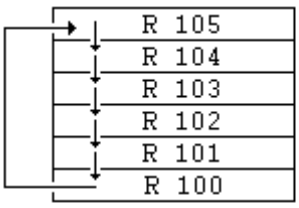
BEISPIEL:

```
ROTD   R 100   ; Rotiert R 100 - R 105
       R 105   ;    um 1 Adresse nach unten
```

FLAGS: Unverändert

SIEHE AUCH: ROTU, SHIU, SHID.

ROTD R 100
R 105



SHIL SCHIEBE REGISTERINHALT NACH LINKS (Shift Register Left)

BESCHREIBUNG: Der Inhalt des adressierten Registers wird um die in der 2. Zeile des Befehls angegebene Anzahl Bit (1 - 31) nach links verschoben.

Ins Bit Nr. 0 (und folgende n-1 Bit) wird der logische Zustand des ACCU (H, L) kopiert und das letzte aus dem Register geschobene bit (MSB) wird in den ACCU geladen.

AUFBAU:

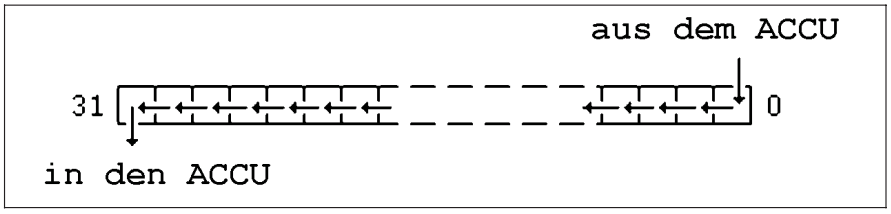
SHIL[X]	**Register (i)**	**; R 0-4095**
	n Bit	**; Anzahl Bit 1-31**

BEISPIEL:

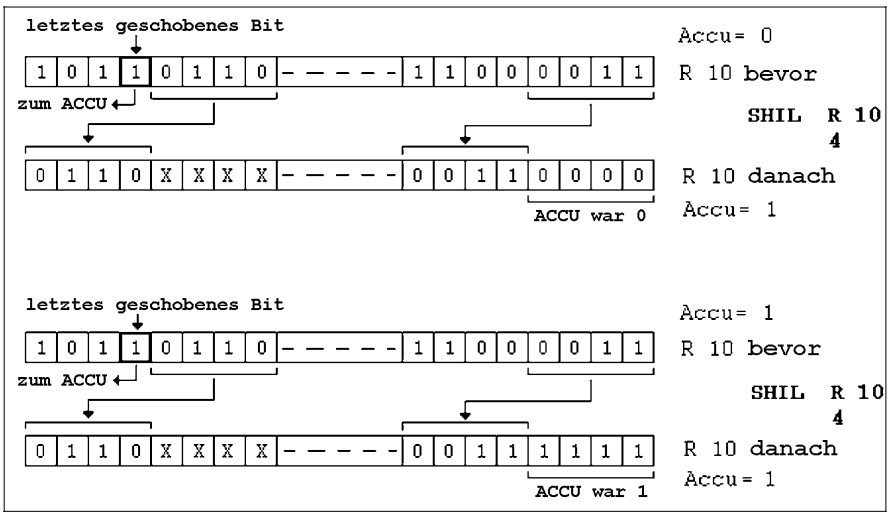
```
SHIL    R 10    ; Inhalt von R 10 wird um
          4     ;   4 Bit nach links geschoben
                ;   (Multiplikation mit 16, wenn ACCU = 0 war)
```

FLAGS: Der ACCU wird gemäss dem letzten aus dem Register geschobenen Bit gesetzt.

SIEHE AUCH: SHIR, ROTL, ROTR.



SHIL um 1 Bit



SHIL um 4 Bit

SHIR SCHIEBE REGISTERINHALT NACH RECHTS (Shift Register Right)

BESCHREIBUNG: Der Inhalt des adressierten Registers wird um die in der 2. Zeile des Befehls angegebene Anzahl Bit (1 - 31) nach rechts verschoben.

Ins Bit Nr. 31 (und weitere n-1 Bit) wird der logische Zustand des ACCU (H, L) kopiert und das letzte aus dem Register geschobene Bit (LSB) wird in den ACCU geladen.

AUFBAU:

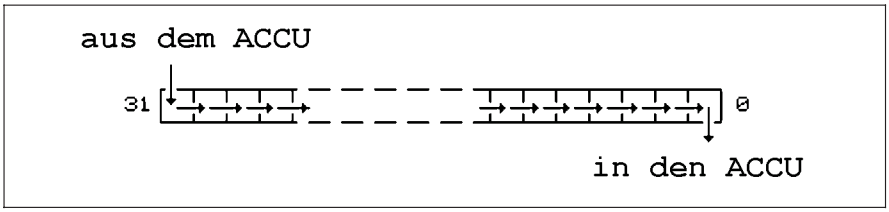
SHIR[X]	**Register (i)**	**; R 0-4095**
	n Bit	**; Anzahl Bit 1-31**

BEISPIEL:

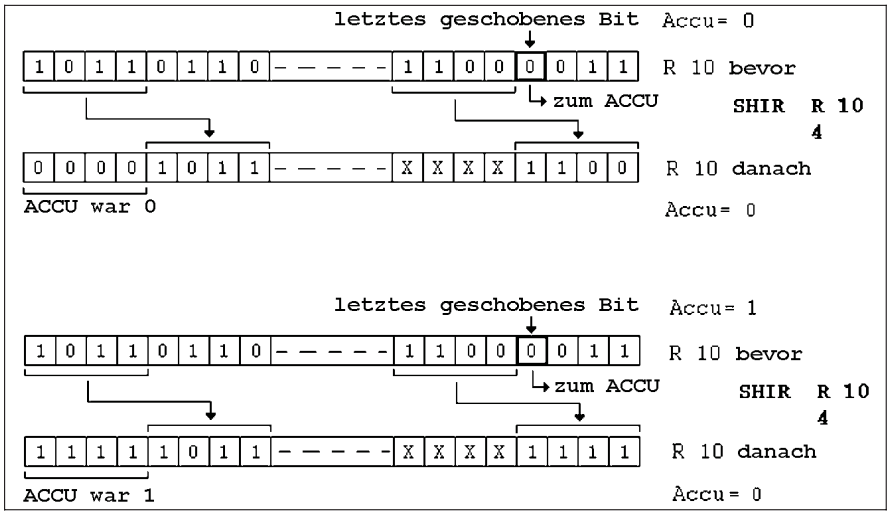
```
SHIR    R 10    ; Inhalt von R 10 wird um
          4     ;   4 Bit nach rechts geschoben
                ;   (Division durch 16, wenn ACCU = 0 war)
```

FLAGS: Der ACCU wird gemäss dem letzten aus dem Register geschobenen Bit gesetzt.

SIEHE AUCH: SHIL, ROTL, ROTR.



SHIR um 1 Bit



SHIR um 4 Bit

ROTL ROTIERE REGISTERINHALT NACH LINKS (Rotate Register Left)

BESCHREIBUNG: Der Inhalt des adressierten Registers wird um die in der 2. Zeile des Befehls angegebene Anzahl Bit (1 - 31) nach links rotiert.

Der ACCU wird mit dem letzten rotierten Bit geladen.

AUFBAU:

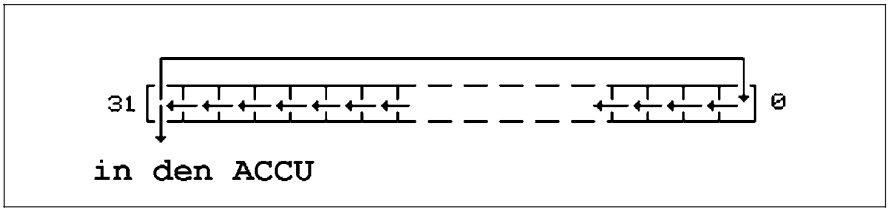
```
ROTL[X]   Register (i)   ; R 0-4095
          n Bit          ; Anzahl Bit 1-31
```

BEISPIEL:

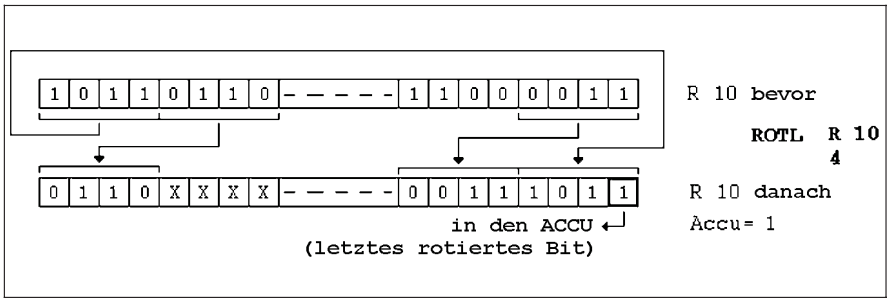
```
ROTL   R 10   ; Inhalt von R 10 wird um
         4    ;   4 Bit nach links rotiert
```

FLAGS: Der ACCU wird gemäss dem letzten rotierten Bit gesetzt.

SIEHE AUCH: ROTR, SHIL, SHIR.



ROTL um 1 Bit



ROTL um 4 Bit

ROTR ROTIERE REGISTERINHALT NACH RECHTS (Rotate Register Right)

BESCHREIBUNG: Der Inhalt des adressierten Registers wird um die in der 2. Zeile des Befehls angegebene Anzahl Bit (1 - 31) nach rechts rotiert.

Der ACCU wird mit dem letzten rotierten Bit geladen.

AUFBAU:

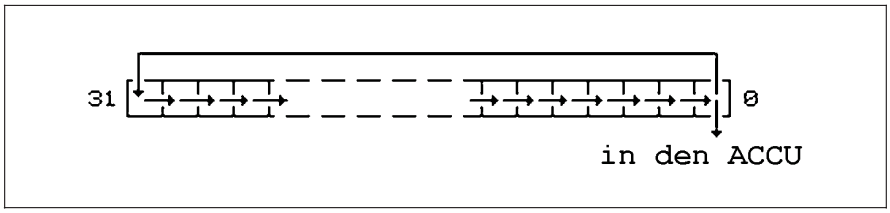
ROTR[X]	**Register (i)**	**; R 0-4095**
	n Bit	**; Anzahl Bit 1 - 31**

BEISPIEL:

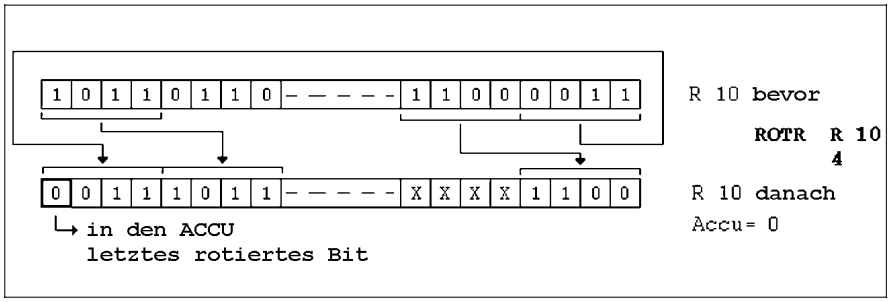
```
ROTR    R 10    ; Inhalt von R 10 wird um
          4     ;    4 Bit nach rechts rotiert
```

FLAGS: Der ACCU wird gemäss dem letzten rotierten Bit gesetzt.

SIEHE AUCH: ROTR, SHIL, SHIR.



ROTR um 1 Bit



ROTR um 4 Bit

4 GANZZAHL ARITHMETIK (Integer Arithmetic)

Die Befehle zur Ganzzahl-Arithmetik arbeiten ausschliesslich mit Registern und Konstanten und nicht mit Countern, Timern oder Einzel-Bit.

ADD	**ADD** Registers	Addiere Register
SUB	**SUB**tract Registers	Subtrahiere Register
MUL	**MUL**tiply Registers	Multipliziere Register
DIV	**DIV**ide Registers	Dividiere Register
SQR	**SQ**uare **R**oot	Quadratwurzel
CMP	**C**o**MP**are Registers	Vergleiche Register

Für Fliesspunkt-Werte sind die Instruktionen für Fliesspunkt-Format zu verwenden.

Fliesspunkt- und Ganzzahl-Arithmetik darf NICHT gemischt werden.

Dem Ganzzahl-Format liegt folgendes 32-Bit Binär-Format zugrunde:

s x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x

31 0

darin sind: x Binär-Stellen (0/1)
s Vorzeichen

Bit 0 bis 30 ist der Ganzzahl-Wert im Binär-Format.
Bit 31 ist das Vorzeichen (0 für einen positiven, 1 für einen negativen Wert).

Der Bereich erstreckt sich auf:

Dezimal:	2’147’483’647	bis	-2’147’483’648
Hexadezimal:	7FFF’FFFF	bis	8000’0000

Ganzzahl-Format

Notizen

ADD ADDIERE REGISTER (ADD Registers)

BESCHREIBUNG: Addition im Ganzzahl-Format.

Addiert den Inhalt des Registers oder die Konstante der 1. Zeile mit dem Inhalt des Registers oder der Konstante der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
ADD     Wert 1      ; R 0-4095, K 0-16'383
        Wert 2      ; R 0-4095, K 0-16'383
        Resultat    ; R 0-4095
```

BEISPIEL:

```
ADD   R 20    ; Addiert zu R 20
      K 123   ;    den Wert 123
      R 21    ;    Resultat in R 21
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: FADD (Addition im Fliesspunkt-Format).

ANWENDUNG: Lese zwei 2-stellige BCD-Werte ab den Eingängen I 16-23 bzw. I 24-31, addiere beide Werte und lege das Resultat ins Register 0.

```
COB   0
      0

DIGI    2     ; Lese 2 Digit
        I 16  ;    ab I 16 (-23)
        R 100 ;    in Register R 100

DIGI    2     ; Lese 2 Digit
        I 24  ;    ab I 24 (-31)
        R 101 ;    in Register R 101

ADD   R 100   ; Addiere R 100
      R 101   ;    mit R 101
      R 0     ;    Resultat in R 0

ECOB
```

SUB SUBTRAHIERE REGISTER (Subtract Registers)

BESCHREIBUNG: Subtraktion im Ganzzahl-Format.

Subtrahiert den Inhalt des Registers oder die Konstante der 2. Zeile vom Inhalt des Registers oder der Konstanten der 1. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
SUB     Wert 1      ; R 0-4095, K 0-16'383
        Wert 2      ; R 0-4095, K 0-16'383
        Resultat    ; R 0-4095
```

BEISPIEL:

```
SUB   R 1    ; Register R 1 minus
      R 2    ;    Register R 2
      R 3    ;    = Resultat in R 3
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem, Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: FSUB (Subtraktion im Fliesspunkt-Format)

ANWENDUNG: Lese zwei 2-stellige BCD-Werte ab den Eingängen I 16-23 bzw. I 24-31, subtrahiere Wert 2 von Wert 1 und lege das Resultat ins Register 12.

```
COB    0
       0

DIGI   2     ; Lese 2 Digit
       I 16  ;    ab I 16 (-23)
       R 10  ;    in Register R 10

DIGI   2     ; Lese 2 Digit
       I 24  ;    ab I 24 (-31)
       R 11  ;    in Register R 11

SUB    R 10  ; Register R 10 minus
       R 11  ;    Register R 11
       R 12  ;    = Resultat in R 12

ECOB
```

MUL MULTIPLIZIERE REGISTER (Multiply Registers)

BESCHREIBUNG: Multiplikation im Ganzzahl-Format.

Multipliziert den Inhalt des Registers oder die Konstante der 1. Zeile mit dem Inhalt des Registers oder der Konstante der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
MUL     Wert 1      ; R 0-4095, K 0-16'383
        Wert 2      ; R 0-4095, K 0-16'383
        Resultat    ; R 0-4095
```

BEISPIEL:

```
MUL   R 0    ; Multipliziert
      K 10   ;   Register R 0 mit
      R 0    ;   der Konstanten 10
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: FMUL (Multiplikation im Fliesspunkt-Format)

ANWENDUNG: Lese zwei 2-stellige BCD-Werte ab den Eingängen I 16-23 bzw. I 24-31, multipliziere die beiden Werte und lege das Resultat ins Register 4000.

```
COB    0
       0

DIGI   2     ; Lese 2 Digit
       I 16  ;   ab I 16 (-23)
       R 50  ;   in Register R 50

DIGI   2     ; Lese 2 Digit
       I 24  ;   ab I 24 (-31)
       R 55  ;   in Register R 55

MUL    R 50  ; Multipliziere R 50
       R 55  ;   mit R 55
       R 4000;   Resultat in R 4000

ECOB
```

DIV DIVIDIERE REGISTER (Divide Registers)

BESCHREIBUNG: Division im Ganzzahl-Format.

Dividiert den Inhalt des Registers oder die Konstante der 1. Zeile mit dem Inhalt des Registers oder der Konstante der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.
Der Divisionsrest wird im Register, das in der 4. Zeile definiert ist, abgelegt. Diese 4. Zeile muss bei der Ganzzahl-Division immer angegeben werden.

AUFBAU:

```
DIV     Wert 1      ; R 0-4095, K 0-16'383
        Wert 2      ; R 0-4095, K 0-16'383
        Resultat    ; R 0-4095
        Rest        ; R 0-4095
```

BEISPIEL:

```
DIV   R 20    ; Dividiert R 20
      K 1000  ;   durch 1000
      R 21    ;   Resultat in R 21
      R 1     ;   Rest in R 1
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.
Bei einer Division durch Null wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: FDIV (Division im Fliesspunkt-Format).

ANWENDUNG: Lese zwei 2-stellige BCD-Werte ab den Eingängen I 16-23 bzw. I 24-31, dividere Wert 1 durch Wert 2 und lege das Resultat in Register 100 und den Rest in Register 101.

```
COB    0
       0

DIGI   2     ; Lese 2 Digit
       I 16  ;   ab I 16 (-23)
       R 1   ;   in Register R 1

DIGI   2     ; Lese 2 Digit
       I 24  ;   ab I 24 (-31)
       R 2   ;   in Register R 2

DIV    R 1   ; Dividiere R 1
       R 2   ;   durch R 2
       R 100 ;   Resultat in R 100
       R 101 ;   Rest in R 101

ECOB
```

SQR QUADRAT-WURZEL (Square Root)

BESCHREIBUNG: Quadrat-Wurzel aus einer ganzen Zahl.
Es wird die Quadrat-Wurzel aus dem Inhalt des Registers der 1. Zeile gezogen und das Resultat ins Register der 2. Zeile abgelegt.
Ein eventueller Rest geht verloren.

AUFBAU:

SQR	**Wert**	**; R 0-4095**
	Resultat	**; R 0-4095**

BEISPIEL:

```
SQR     R 0     ; Die Quadrat-Wurzel von R 0
        R 100   ;    wird in R 100 abgelegt
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.
Wird versucht, die Quadrat-Wurzel aus einer negativen Zahl zu ziehen, wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: FSQR (Quadrat-Wurzel im Fliesspunkt-Format).

ANWENDUNG: Ziehe die Quadrat-Wurzel aus einem 4-stelligen BCD-Wert der an den Eingängen I 16-31 anliegt und bringe das Resultat ins Register R 101.
Im Falle eines ERRORs ist Ausgang 47 zu setzen.

```
COB     0
        0

DIGI    4       ; Lese 4 Digit
        I 16    ;    ab I 16 (-31)
        R 100   ;    in Register R 100

SQR     R 100   ; Quadrat-Wurzel aus R 100
        R 101   ;    Resultat in R 101

CPB     E 99    ; Bei ERROR, rufe PB 99 auf

ECOB
;-------------------------------------------
PB      99
SET     O 47    ; Alarm, da Wurzel aus
EPB             ;    negativer Zahl
```

CMP VERGLEICHE REGISTER (Compare Registers)

BESCHREIBUNG: Vergleicht den Inhalt des Registers oder der Konstanten der 1. Zeile mit dem Inhalt des Registers oder der Konstanten der 2. Zeile und setzt die entsprechenden Status-Flags.
Eigentlich wird die 2. Zeile von der 1. Zeile subtrahiert, das Resultat wird aber nirgends abgelegt, und die beiden Werte bleiben unverändert.

AUFBAU:

```
CMP[X]   Wert 1   (i)   ; R 0-4095, K 0-16'383
         Wert 2         ; R 0-4095, K 0-16'383
```

BEISPIEL:

```
CMP   R 0    ; Vergleiche R 0 mit R 1 und
      R 1    ;   setze die Status-Flags
             ;   entsprechend dem Ergebnis
```

FLAGS: Die Status-Flags werden gemäss der nachstehenden Tabelle gesetzt:

	Z-	P-	N-Flag
Wert 1 = Wert 2	H	H	L
Wert 1 > Wert 2	L	H	L
Wert 1 < Wert 2	L	L	H

ANMERKUNG: Es ist zu beachten, dass bei Gleichheit sowohl das Z- als auch das P-Flag gesetzt wird. Das P-Flag ist immer das Komplement des N-Flags.

SIEHE AUCH: AND, OR, EXOR, FCMP.

ANWENDUNG: Ein auf der seriellen Schnittstelle 1 angekommener ASCII-Charakter soll geprüft werden. Handelt es sich um ein "Y", soll der PB 10 aufgerufen werden.

```
. . .
SRXD    1      ; Verschiebe den ASCII-Charakter
        R 999  ;   aus UART Nr. 1 in R 999

CMP     R 999  ; Vergleiche R 999 mit ASCII-
        'Y'    ;   Char. Y. Wenn gleich = Z-Flag
CPB     Z 10   ;   Rufe PB 10 wenn Z-Flag gesetzt
. . .
```

5 FLIESSPUNKT-ARITHMETIK

Fliesspunkt Werte können nur in Registern abgelegt werden. Mit dem Befehl LD (Lade 32 Bit-Wert) können Fliesspunkt-Werte direkt in Register geladen werden. Der Wert muss einen Dezimalpunkt und/oder einen Exponenten haben, um als Fliesspunkt-Wert erkannt zu werden, z,B. 1.2, 1e3, -4.656E-2.

Das Fliesspunkt-Format kennt keine Konstanten.

Die Bereiche der Fliesspunkt-Werte sind:

+5.42101E-20 . . . +9.22337E+18
- 5.42101E-20 . . . -9.22337E+18

Nur die 5 höchstwertigen Digits sind genau!

IFP	**I**nteger to **FP**	Ganzzahl- zu FP-Format
FPI	**FP** to **I**nteger	FP- zu Ganzzahlformat
FADD	**F**p **ADD**ition	FP Addition
FSUB	**Fp SUB**traction	FP Subtraktion
FMUL	**F**p **MUL**tiplication	FP Multiplikation
FDIV	**F**p **DIV**ision	FP Division
FSQR	**F**p **SQ**uare **R**oot	FP Quadrat Wurzel
FCMP	**F**p **C**o**MP**are	FP Vergleich
FSIN	**F**p **SIN**e function	FP Sinus Funktion
FCOS	**F**p **COS**in function	FP Cosinus Funktion
FATAN	**F**p **A**rc **TAN**gent function	FP Arcus Tangens Funktion
FEXP	**F**p **EXP**onentioal function	FP Exponential Funktion
FLN	**F**p **L**ogarithm function	FP natürlicher Logarithmus
FABS	**F**p **ABS**olute value	FP Absolutwert

Fp = Floating point **FP** = Fliesspunkt-Format

WICHTIG: Fliesspunkt-Werte werden in einem eigenen Format in Register abgelegt und können dezimal nicht interpretiert werden.
Ganzzahl- und Fliesspunkt-Werte dürfen in mathematischen Operationen nicht gemischt werden. Ganzzahl-Werte müssen zuerst mit dem Befehl IFP in Fliesspunkt-Werte gewandelt werden. Andererseits können Fliesspunkt-Werte mit dem Befehl FPI in Ganzzahl-Werte gewandelt werden.

Dem Fliesspunkt-Format liegt folgendes 32-Bit Format zugrunde:

m m m m m m m m m m m m m m m m m m m m m m m m s e e e e e e e
31 0

darin sind: m 24 Bit Mantisse
s Vorzeichen des Wertes (0 = pos, 1 = neg)
e 7 Bit Exponent

Fliesspunkt-Format

Notizen

IFP GANZZAHL- ZU FLIESSPUNKT-FORMAT (Integer to Floating Point)

BESCHREIBUNG: Wandelt den Ganzzahl-Wert im adressierten Register ins Fliesspunkt-Format.

Der Exponent, der in der 2. Zeile angegeben werden muss, gibt die Zehnerpotenz an, in welcher der Registerinhalt im FP-Format stehen soll. (Multiplikator).

AUFBAU:

IFP[X]	**Register (i)**	**; R 0-4095**
	Zehner-Potenz	**; -20 bis +18**

BEISPIEL:

```
IFP     R 0     ; Fliesspunkt-Wert von
          3     ;    R 0 * 10^3
```

FLAGS: Wird eine nicht mögliche Wandlung versucht, wird das ERROR-Flag gesetzt.

SIEHE AUCH: FPI.

ANWENDUNG:

R500 vor der Wandlung	**Befehl**		**Bedeutung**	R 500 nach der Wandlung
	IFP	R 500 0	R 500 * 10^0	1.23E+2
123	IFP	R 500 -2	R 500 * 10^{-2}	1.23E+0
	IFP	R 500 3	R 500 * 10^3	1.23E+5

FPI FLIESSPUNKT- ZU GANZZAHL-FORMAT (Floating Point to Integer)

BESCHREIBUNG: Wandelt den Fliesspunkt-Wert im adressierten Register ins Ganzzahl-Format.

Der Exponent, der in der 2. Zeile angegeben werden muss, gibt die Zehnerpotenz an, in welcher der Registerinhalt im Ganzzahl-Format stehen soll. (Multiplikator).

AUFBAU:

FPI[X]	**Register (i)**	**; R 0-4095**
	Zehner-Potenz	**; -20 bis +18**

BEISPIEL:

```
FPI     R 1     ; Ganzzahl-Wert von
          -3    ;    R 1 * 10^-3
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: IFP.

ANWENDUNG:

R500 vor der Wandlung	**Befehl**		**Bedeutung**	R 500 nach der Wandlung
	FPI	R 500 0	R 500 * 10^0	123
123.456	FPI	R 500 -2	R 500 * 10^{-2}	1
	FPI	R 500 3	R 500 * 10^3	123456

FADD FP ADDITION (Floating Point Addition)

BESCHREIBUNG: Addition im Fliesspunkt-Format.

Addiert den Inhalt des Registers der 1. Zeile mit dem Inhalt des Registers der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
FADD    Register 1    ; R 0-4095
        Register 2    ; R 0-4095
        Resultat      ; R 0-4095
```

BEISPIEL:

```
FADD   R 100   ; Addiere R 100
       R 101   ;    mit R 101
       R 500   ;    Resultat in R 500
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: ADD.

FSUB FP SUBTRAKTION (Floating Point Subtraction)

BESCHREIBUNG: Subtraktion im Fliesspunkt-Format.

Subtrahiert den Inhalt des Registers der 2. Zeile vom Inhalt des Registers der 1. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
FSUB    Register 1    ; R 0-4095
        Register 2    ; R 0-4095
        Resultat      ; R 0-4095
```

BEISPIEL:

```
        FSUB    R 0     ; Subtrahiere R 1
                R 1     ;    von R 0
                R 0     ;    Resultat in R 0
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: SUB.

FMUL FP MULTIPLIKATION (Floating Point Multiplication)

BESCHREIBUNG: Multiplikation im Fliesspunkt-Format.

Multipliziert den Inhalt des Registers der 1. Zeile mit dem Inhalt des Registers der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.

AUFBAU:

```
FMUL    Register 1    ; R 0-4095
        Register 2    ; R 0-4095
        Resultat      ; R 0-4095
```

BEISPIEL:

```
        FMUL   R 20     ; Multipliziere R 20
               R 30     ;    mit R 30
               R 3999   ;    Resultat in R 3999
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einem Überlauf des Resultat-Registers wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: MUL.

FDIV FP DIVISION (Floating Point Divide)

BESCHREIBUNG: Division im Fliesspunkt-Format.

Dividiert den Inhalt des Registers der 1. Zeile mit dem Inhalt des Registers der 2. Zeile und legt das Resultat im Register der 3. Zeile ab.

Bei der Division im Fliesspunkt-Format gibt es keinen Rest.

AUFBAU:

```
FDIV    Register 1    ; R 0-4095
        Register 2    ; R 0-4095
        Resultat      ; R 0-4095
```

BEISPIEL:

```
        FDIV    R 1     ; Dividiere R 1
                R 2     ;    durch R 2
                R 3     ;    Rusultat in R 3
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Bei einer Division durch Null wird das ERROR-Flag gesetzt und die Operation nicht ausgeführt.

SIEHE AUCH: DIV.

FSQR FP QUADRAT-WURZEL (Floating Point Square Root)

BESCHREIBUNG: Quadrat-Wurzel im Fliesspunkt-Format.

Es wird die Quadrat-Wurzel aus dem Inhalt des Registers der 1. Zeile gezogen und das Resultat ins Register der 2. Zeile abgelegt.

Wird versucht, die Quadrat-Wurzel aus einer negativen Zahl zu ziehen, wird das ERROR-Flag gesetzt und die Wurzel aus dem Absolutwert gezogen.

AUFBAU:

```
FSQR    Wert        ; R 0-4095
        Resultat    ; R 0-4095
```

BEISPIEL:

```
        FSQR    R 0     ; Die Quadrat-Wurzel von R 0
                R 0     ;    wird in R 0 abgelegt
```

FLAGS: Das Z- und das P-Flag werden gemäss dem Resultat gesetzt.

Wird die Quadrat-Wurzel aus einer negativen Zahl gezogen wird das ERROR-Flag gesetzt und die Wurzel aus dem Absolutwert gezogen.

SIEHE AUCH: SQR.

FCMP FP VERGLEICH (Floating Point Compare)

BESCHREIBUNG: Vergleich von Registern im Fliesspunkt-Format.

Vergleicht den Inhalt des Registers der 1. Zeile mit dem Inhalt des Registers der 2. Zeile und setzt die Status-Flags entsprechend dem Resultat.

Eigentlich wird die 2. Zeile von der 1. Zeile subtrahiert, das Resultat wird aber nirgends abgelegt, und die beiden Werte bleiben unverändert.

Achtung: Das Vergleichen von Fliesspunkt-Werten auf Gleichheit (Z) ist zu unterlassen!

AUFBAU:

```
FCMP[X]   Register 1   (i)    ; R 0-4095
          Register 2           ; R 0-4095
```

BEISPIEL:

```
FCMP   R 0     ; Vergleiche R 0 mit R 1
       R 1     ;   Setze die Status-Flags
               ;   entsprechend dem Ergebnis
```

FLAGS: Die Status-Flags werden gemäss der nachstehenden Tabelle gesetzt:

	Z-	P-	N-Flag
Wert 1 = Wert 2	H	H	L
Wert 1 > Wert 2	L	H	L
Wert 1 < Wert 2	L	L	H

Das ERROR-Flag wird immer zurückgesetzt.

ANMERKUNG Es ist zu beachten, dass bei Gleichheit sowohl das Z- als auch das P-Flag gesetzt wird. Das P-Flag existiert im Prinzip gar nicht, es ist immer das Komplement des N-Flag.

Achtung: Das Vergleichen von Fliesspunkt-Werten auf Gleichheit (=) ist zu unterlassen!

SIEHE AUCH: CMP.

FSIN FP SINUS-FUNKTION (Sine Function)

BESCHREIBUNG: Sinus-Funktion im Fliesspunkt-Format.

Es wird die Sinus-Funktion aus dem Inhalt des Registers der 1. Zeile (im Bogenmass) gezogen und das Resultat ins Register der 2. Zeile abgelegt.

Der Wert im Register der 1. Zeile muss im Bogenmass stehen und muss innerhalb von $\pm 10^6$ liegen.

AUFBAU:

FSIN[X]	**Wert**	**(i)**	**; R 0-4095**
	Resultat	**(i)**	**; R 0-4095**

BEISPIEL:

```
FSIN    R 55    ; Sinus vom Inhalt von R 55,
        R 88    ;    Resultat in R 88
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

SIEHE AUCH FCOS, FATAN.

FCOS FP COSINUS-FUNKTION (Floating Point Cosine Function)

BESCHREIBUNG: Sinus-Funktion im Fliesspunkt-Format.

Es wird die Cosinus-Funktion aus dem Inhalt des Registers der 1. Zeile (im Bogenmass) gezogen und das Resultat ins Register der 2. Zeile abgelegt.

Der Wert im Register der 1. Zeile muss im Bogenmass stehen und muss innerhalb von $\pm 10^6$ liegen.

AUFBAU:

FCOS[X]	**Wert**	**(i)**	**; R 0-4095**
	Resultat	**(i)**	**; R 0-4095**

BEISPIEL:

```
FCOS   R 5     ; Cosinus vom Inhalt von R 5,
       R 8     ;    Resultat in R 8
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

SIEHE AUCH: FSIN, FATAN.

FATAN FP ARCUS TANGENS FUNKTION (Floating Point Arc Tangent)

BESCHREIBUNG: Arcus Tangens Funktion im Fliesspunkt-Format.

Es wird die Arcus Tangens Funktion aus dem Inhalt des Registers der 1. Zeile (im Bogenmass) gezogen und das Resultat ins Register der 2. Zeile abgelegt.

Im Register der 1. Zeile wird der Tangens im FP-Format angegeben.

Das Resultat im Register der 2. Zeile ist der Winkel im Bogenmass innerhalb -Π/2 bis +Π/2

AUFBAU:

FATAN[X]	**Wert**	**(i)**	**; R 0-4095 (Tangens)**
	Resultat	**(i)**	**; R 0-4095 (Bogenmass)**

BEISPIEL:

```
FATAN   R 1   ; Arcus Tangens von R 1,
        R 2   ;   Resultat in R 2
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

SIEHE AUCH: FSIN, FCOS.

FEXP FP EXPONENTIAL-FUNKTION (Floating Point Exponetial Function)

BESCHREIBUNG: Exponential-Funktion im Fliesspunkt-Format.

Auf den Inhalt des Registers der 1. Zeile wird die Exponentialfunktion e^x ausgeübt und das Resultat ins Register der 2. Zeile abgelegt.

AUFBAU:

FEXP[X]	**Wert**	**(i)**	**; R 0-4095**
	Resultat	**(i)**	**; R 0-4095**

BEISPIEL:

```
FEXP   R 0     ;
       R 44    ; R 44 = e^R 33
```

FLAGS: Das Z-Flag wird gemäss dem Resultat gesetzt. Das N-Flag ist immer zurückgesetzt (P-Flag gesetzt).

Bei einem Überlauf wird das ERROR-Flag gesetzt.

SIEHE AUCH: FPI, IFP.

FLN FP NATÜRLICHER LOGARITHMUS (Floating Point Logarithm Function)

BESCHREIBUNG: Natürlicher Logarithmus im Fliesspunkt-Format.

Es wird der natürliche Logarithmus aus dem Inhalt des Registers der 1. Zeile gebildet und das Resultat ins Register der 2. Zeile abgelegt.

AUFBAU:

FLN[X]	**Wert**	**(i)**	**; R 0-4095**
	Resultat	**(i)**	**;R 0-4095**

BEISPIEL:

```
FLN     R 5     ;
        R 6     ; R 6 = In R 5
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

Wird versucht, den natürlichen Logarithmus aus einer negativen Zahl zu bilden, wird das ERROR-Flag gesetzt und der Logarithmus aus dem Absolutwert errechnet.

SIEHE AUCH: FCOS, FATAN.

FABS FP ABSOLUTWERT (Floating Point Absolute Value)

BESCHREIBUNG: Absolutwert im Fliesspunkt-Format.

Es wird der Absolutwert (immer positiver Wert) aus dem Inhalt des Registers der 1. Zeile gebildet und das Resultat ins Register der 2. Zeile abgelegt.

AUFBAU:

FABS[X]	**Wert**	**(i)**	**; R 0-4095**
	Resultat	**(i)**	**; R 0-4095**

BEISPIEL

```
FABS   R 100   ; wenn Wert in R 100 = -7.5
       R 99    ;      Resultat in R 99 = 7.5
```

FLAGS: Das Z-, das P- und das N-Flag werden gemäss dem Resultat gesetzt.

SIEHE AUCH: -

6. BLOCTEC-Instruktionen

BLOCTEC ist eine Methode zur Programmstrukturierung, d.h. zum Unterteilen von Anwenderprogrammen in einzelne, eigenständige Programmteile, sog. Blocks.

Der **zyklische Organisations-Block** (COB) ist das Grundstrukturelement eines Anwenderprogramms.

Mindestens 1 COB muss vorhanden sein, damit ein Anwenderprogramm lauffähig ist.

Aus einem COB werden im Normalfall kleinere Programmeinheiten wie **Programm-Blocks** (PB) und **Funktions-Blocks** (FB) aufgerufen.

PB und FB können wiederum PB und/oder FB aufrufen. Dies bis zu einer Verschachtelungstiefe von 7 Ebenen. (8 mit dem COB).

Die strukturierte Programmierung ist ausführlich im PCD Anwender-Handbuch beschrieben.

Die Operanden aller Befehle dieses Kapitels können nicht als Parameter an FB übergeben werden.

COB	Cyclic Organisation Block	Zyklischer Organisations-Block
ECOB	End of Cyclic Org'n Block	Ende eines COB
XOB	Exception Org'n Block	Ausnahme Organisations-Block
EXOB	End of Except. Org'n Block	Ende XOB
PB	Program Block	Programm-Block
EPB	End of Program Block	Ende eines Programm-Blocks
CPB	Call Program Block	Aufruf eines Program-Blocks
CPBI	Call Program Block Indirect	Indirekter Aufruf eines Program-Blocks
FB	Function Block	Funktions-Block
EFB	End of Function Block	Ende eines funktions-blocks
CFB	Call Function Block	Aufruf eines funktions-Blocks
NCOB	Next Cyclic Org'n Block	Wechsel zum nächsten COB
SCOB	Stop Cyclic Org'n Block	Neustart eines COB
CCOB	Continue Cyclic Org'n Block	Stoppe eienen COB
RCOB	Restart Cyclic Org'n Block	Weiterarbeiten im COB

Die folgenden BLOCTEC-Befehle sind mit Vorsicht anzuwenden:

RCOB, NCOB, SCOB, CCOB sowie die COB Überwachungszeit

Die Verwendung dieser Befehle in einem GRAFTEC-Programm sollte vermieden werden, da dies zu ernsthaften Problemen führen kann.

Werden diese Befehle nicht mit der notwendigen Sorgfalt eingesetzt, werden im günstigsten Fall die Anwenderprogramme langsamer oder im schlimmsten Fall kann eine Desynchronisation eines GRAFTEC-Progamms auftreten, was zu einem Absturz führen kann.

Notizen

COB Zyklischer Organisations-Block
(Cyclic Organisation Block)

Beschreibung: Kennzeichnet den Beginn eines zyklischen Organisations-Blocks mit seiner Nummer in der 1. und der Überwachungszeit-Konstanten (in l0 ms Einheiten) in der 2. Zeile des Befehls.

Dauert die Abarbeitungszeit des COB länger als die in der 2. Zeile spezifizierte Zeit, wird der XOB 11 aufgerufen oder, wenn der XOB 11 nicht programmiert ist, zum nächsten COB, d.h. zum COB mit der nächst höheren Nummer gewechselt.

Wird in der 2. Zeile der Wert 0 (Null) eingegeben, wird ohne COB-Überwachungzeit gearbeitet.

Aufbau:

```
COB     Nummer              ; COB-Nummer 0 - 15
        Überwachungszeit    ; Wert * 10 ms
```

Der Wertebereich der Überwachungszeit ist 2 - 2'147'483'648
2 bedeutet 10 .. 20 ms
1 wäre 0 .. 10 ms, was nicht zu empfehlen ist

Beispiel:

```
COB     0       ; Start des COB 0
        0       ;    ohne Überwachungszeit
        ....
        ....
        ....
ECOB            ; Ende des COB 0
```

Flags: Bei jedem Block-Anfang wird der ACCU = H gesetzt

Siehe auch: ECOB, Anwender-Handbuch

ECOB Ende eines COB

(End of Cyclic Organisation Block)

Beschreibung: Beendet den laufenden COB.

Jeder COB muss mit dem Befehl ECOB abgeschlossen werden.

Sind im gleichen Anwenderprogramm weitere COB vorhanden, wird mit dem COB mit der nächst höheren Nummer weitergefahren. Ist kein weiterer COB vorhanden, geht die Programmabarbeitung am Anfang dieses einzelnen COB weiter.

Aufbau:

ECOB		**; kein Operand**

Beispiel:

```
COB     0       ; Beginn des COB 0
        0
        ...     ; Programmcode im COB
        ...
        ...
ECOB            ; Ende des COB 0
```

Flags: Unverändert

Siehe auch: COB, Anwender-Handbuch

XOB Ausnahme Organisations-Block
(Exception Organisation Block)

Beschreibung: Markiert den Beginn eines Ausnahme Organisations-Blocks

Aufbau:

XOB	**Nummer**	**; XOB number 0 - 30**

Beispiel:

```
XOB     16      ; Kalt-Start XOB
        ...     ;  beliebiger Programmcode
EXOB            ; Ende des XOB 16
```

Flags: Der ACCU wird zu Beginn des XOB = H gesetzt

Siehe auch: EXOB, Anwender-Handbuch

XOB werden vom System automatisch aufgerufen, wenn ein Fehler in der Hardware oder im Anwenderprogramm auftritt. XOB können vom Anwender **nicht** aufgerufen werden. (Ausnahme: XOB 17, 18, 19). XOBs enthalten Anwenderprogrammcode zur Behandlung der aufgetretenen Fehler. Ist der XOB nicht programmiert wird die ERROR-Lampe aktiviert. Nach Beendigung des XOB geht das Anwenderprogramm an der Stelle weiter, wo dieses beim Aufruf des XOB verlassen wurde.

Jeder XOB wird bei einem spezifischen Ereignis aufgerufen:

XOB	Beschreibung	Priorität
0	Speisungsfehler im Hauptgerät	4
8	Ungültiger Operationscode	4
7	System überlastet	3
11	Überschreitung COB-Überwachungszeit	3
14	Zyklischer XOB	3
15	Zyklischer XOB	3
17	S-Bus XOB Interrupt-Aufruf	3
18	S-Bus XOB Interrupt-Aufruf	3
19	S-Bus XOB Interrupt-Aufruf	3
20	Interrupt-Eingang INB1	3
25	Interrupt-Eingang INB2	3
1	Speisungsfehler in Erweiterungsgehäuse	2
2	Speisungsfehler der Batterie	2
4	Paritätsfehler auf dem I/0-Bus (nur PCD6)	1
5	Adressierungsfehler eines E/A-Moduls	1
6	Externer Fehler (nicht verwendet)	1
9	Zuviele aktive GRAFTEC-Zweige	1
10	Mehr als 7 PB/FB-Ebenen	1
12	Inhalt des Indexregisters zu gross	1
13	Error-Flag gesetzt	1
16	Kalt-Start Routine	1
30	RIO-Verbindung Master <-> Slaves	1

Wird während der Abarbeitung eines XOB ein anderer XOB der gleichen oder einer niedrigeren Priorität aufgerufen, wird dieser ignoriert. Wird also während der Abarbeitung des XOB 16 ein Fehler festgestellt, wird der XOB 13 **nicht** aufgerufen.

XOBs des Levels 4 :

Level 4 hat die höchste Priorität. Nur die XOB 0 und 8 können die andern XOB in deren Abarbeitung unterbrechen.

XOB 0 Speisungsfehler im Hauptgerät

Zwischen dem Aufruf des XOB 0 und dem definitiven Spannungsausfall können bis zu 10 ms verstreichen. Diese Zeit kann zum Speichern wichtiger Prozessdaten genutzt werden.
Ist der XOB 0 programmiert, wird beim Aufruf "XOB 0 START EXEC" und beim Erreichen des Endes des XOB 0 "XOB 0 EXECUTED" in die History-Liste geschrieben. Dies zeigt an, dass der XOB 0 komplett ausgeführt wurde, bevor die Speisung ganz weg war.
Ist der XOB 0 nicht programmiert, wird bei der Feststellung eines unzulässigen Abfalls der Speisespannung sofort ein "Restart Cold" durch geführt. Ist der XOB 0 programmiert, wird nach Beendigung der Abarbeitung des XOB 0 auch ein "Restart Cold" durchgeführt, falls die Speisung noch da ist.

XOB 8 Ungültiger Operationscode

Dieser XOB wird aufgerufen, wenn die Firmware einen ungültigen Befehl im Anwenderprogramm feststellt. (z.B. fehlerhafte Programmänderung im Debugger)

XOBs des Levels 3:

Wird ein XOB eines Levels 2 oder 3 während der Abarbeitung eines XOB einer niedrigeren Priorität aufgerufen, wird die Behandlung dieser XOB unmittelbar nach Beendigung der Abarbeitung des aktuellen XOB vorgenommen.
Den XOB 20/25/11 wurde eine höhere Priorität zugedacht, was bedeutet, dass bei einem Aufruf während der Abarbeitung eines XOB gleicher oder tieferer Priorität, die Abarbeitung gleich nach dem Ende des laufenden XOB erfolgt.

XOB 7 System überlastet

Die Stapelung für die 3 XOB-Levels ist überlastet.

XOB 11 Überschreitung der COB-Überwachungszeit

Wird im Befehl COB in der 2. Zeile eine Überwachungszeit (in 1/100 sek) angegeben und dauert die Abarbeitungszeit des COB länger als diese definierte Zeit, wird der XOB 11 aufgerufen. Die COB Abarbeitungszeit dauert vom Befehl COB bis zum ECOB.

XOB 14 Zyklischer XOB
XOB 15 Zyklischer XOB

XOB 14 und 15 werden periodisch in Intervallen von 5 bis 1000s aufgerufen. Die Aufruffrequenz kann mittels des Befehls SYSWR bestimmt werden.

XOB 17 S-Bus XOB Interrupt-Aufruf
XOB 18 S-Bus XOB Interrupt-Aufruf
XOB 19 S-Bus XOB Interrupt-Aufruf

Diese 3 XOB können als Interrupt-Routinen verwendet werden. Der Aufruf erfolgt via das S-Bus-Netzwerk. Der Aufruf kann auch mit dem Befehl SYSWR erfolgen.

XOB 20 Interrupt-Eingang INB1
XOB 25 Interrupt-Eingang INB2
XOB 20 (bzw. XOB 25) wird mit einer ansteigenden Flanke am Interrupt-Eingang INB1 (bzw. INB2) aufgerufen. (Siehe Handbuch der PCD1/2)

XOB des Levels 2:

XOB 1 Speisungsfehler im Erweiterungsgehäuse
Die Spannungsüberwachung eines Erweitungsgehäuses (PCD2 oder PCD6) ermittelte einen zu grossen Spannungsabfall. Es werden in diesem Fall alle digitalen Ausgänge innerhalb von 2 ms = L geschaltet und der XOB 1 aufgerufen. Werden die Ausgänge dieses "toten" Gehäuses im Anwenderprogramm weiter behandelt (ein-, ausschalten, abfragen) werden auch die XOB 4 und/oder XOB5 aufgerufen. Der XOB 1 wird nur einmal und zwar 250 ms nach dem Auftreten des Fehlers aufgerufen.

XOB 2 Speisungsfehler der Batterie
Die Spannung der Batterie ist zu tief, die Batterie ist defekt oder nicht vorhanden. Daten in den nicht-flüchtigen Flags, den Registern und das Anwenderprogramm auf RAM sind möglicherweise verändert.
Im Fehlerfall wird der XOB 2 von der CPU 0 alle 250 ms aufgerufen.

XOB des Levels 1:

Werden XOBs des Levels 1 während der Abarbeitung eines andern XOB aufgerufen, sind diese Aufrufe verloren.

XOB 4 Paritätsfehler auf dem I/O-Bus
Ein XOB 4 kann nur durch eine PCD6 mit Erweiterungsgehäuse aufgerufen werden. Die Überwachungsschaltung des Adressbusses hat einen Paritäts-Fehler entdeckt. Ein solcher Fehler kann durch ein defektes Erweiterungskabel, ein defektes Erweiterungsgehäuse, ein defektes Erweiterungsmodul oder auch nur von einer falschen Erweiterungsadresse herrühren.

XOB 5 Adressierungsfehler eines E/A-Moduls
Die digitalen Eingangs- und Ausgangsmodule der PCD senden der aufrufenden CPU ein Bestätigungssignal zurück. Fehlt dieses Signal, wird der XOB 5 aufgerufen. Normalerweise erfolgt der Aufruf dieses XOB, wenn das adressierte Modul nicht bestückt ist. Es kann jedoch auch von einer fehlerhaften Adressdecodierung herrühren.
Bei einem PCD4-Modul mit nur 8 Elementen wird der XOB 5 nicht aufgerufen, wenn eines der nicht vorhandenen Elemente angesteuert wird, da diese Adressen auch decodiert und das jeweilige Signal zurückgeschickt wird.

Bei der PCD1 und 2 ist dieser Mechanismus nicht vorhanden.

XOB 6 Externer Fehler
Nicht verwendet (Ursprünglich für intelligente Module der PCD6 vorgesehen)

XOB 9 Zuviele aktive GRAFTEC-Zweige
Sind in einem Sequential Block (SB) mehr als 32 GRAFTEC-Zweige gleichzeitig aktiv, wird der XOB 9 aufgerufen.

XOB 10 Mehr als 7 PB/FB-Ebenen
PB und FB können bis zu einer Tiefe von 7 Ebenen verschachtelt werden. Ein weiterer Aufruf (8. Ebene) aktiviert den XOB 10. Der letzte Aufruf (8. Ebene) wird nicht ausgeführt.

XOB 12 Inhalt des Indexregisters zu gross
Enthält ein Anwenderprogramm ein indexiertes Element welches bei der Behandlung ausserhalb dessen Adressbereich fällt (0 bis 8191), wird der XOB 12 aufgerufen.

XOB 13 Error Flag gesetzt
Der XOB 13 wird jedesmal aufgerufen, wenn das Error-Flag gesetzt wird und zwar unabhängig davon, ob es sich um einen Rechenfehler, einen Fehler im Datentransfer oder einen Fehler in der seriellen Kommunikation handelt.

XOB 16 Kalt-Start Routine
Der XOB 16 ist der Kalt-Start Anwenderprogrammteil. Der XOB 16 wird jedesmal nach dem Einschalten der PCD oder nach einem "Restart Cold" einmal durchlaufen. Im XOB 16 werden üblicherweise Initialisierungen vorgenommen bevor das eigentliche Anwenderprogramm abgearbeitet wird. Ereignet sich während der Ausführung des XOB 16 ein Fehler, wird der XOB 13 nicht aufgerufen.

XOB 30 RIO-Verbindung Master ↔ Slaves (in Vorbereitung)
Nach jedem Telegramm vom Master zu einem Slave wird die Verbindung getestet. Fällt die Antwort negativ aus, ruft der Master den XOB 30 auf. Dies ist hauptsächlich dann der Fall, wenn eine Station online vom Netz getrennt oder ausgeschaltet wird.

EXOB Ende des XOB

(End of Exception Organisation Block)

Beschreibung: Steht am Ende des aktuellen XOB.

Nach EXOB geht das Programm bei der Adresse, wo der XOB aufgerufen wurde +1 weiter, wenn der Fehler, welcher für den XOB-Aufruf massgebend war, dies noch erlaubt.

Aufbau:

EXOB	**; kein Operand**

Beispiel:

```
XOB     13      ; Error-Flag gesetzt
        ...     ;       beliebiger Programmcode
        ...
        ...
EXOB            ; Ende des XOB 13
```

Flags: Unverändert

Siehe auch: XOB, Anwender-Handbuch

PB Programm-Block
(Program Block)

Beschreibung: Kennzeichnet den Beginn eines Programm-Blocks (PB) mit der entprechenden Nummer.

Ein PB ist ein Unterprogramm (Subroutine), das ohne Parameter, bedingt oder unbedingt, aus einem andern Block (COB, PB, FB, XOB, SB) aufgerufen werden kann.

Aufbau:

PB	**Nummer**	**; PB-Nummer 0-299**

Beispiel:

```
PB      26      ; Beginn des PB 26
        ...     ;    Programmcode
        ...
        ...
EPB             ; Ende des PB 26
```

Flags: Bei jedem Block-Anfang wird der ACCU = H gesetzt

Siehe auch: EPB, CPB, FB, Anwender-Handbuch

EPB Ende eines Programm-Blocks
(End of Program Block)

Beschreibung: Ende eines Programm-Bocks (PB).

Der Programmablauf geht an der Aufrufstelle des PB +1 weiter.

Aufbau:

EPB	**; kein Operand**

Beispiel:

```
PB      26      ; Beginn des PB 26
        ...     ;    Programmcode
        ...
        ...
EPB             ; Ende des PB 26
```

Flags: Im aufrufenden Block ist der Zustand des ACCU wieder gleich wie vor dem Aufruf des PB. Die Status-Flags können verändert sein.

Siehe auch: PB, CPB, Anwender-Handbuch

CPB Aufruf eines Programm-Blocks
(Call Program Block)

Beschreibung: Bedingter oder unbedingter Aufruf eines Programm-Blocks. Ist die Bedingung nicht erfüllt, wird der Programm-Block nicht aufgerufen und das Programm fährt mit dem nächsten Befehl fort.

Die Aufrufbedingungen können die folgenden sein:

Bedingung	Programm-Block wird aufgerufen::
-	ohne Bedingung (immer)
H	wenn ACCU = H (1)
L	wenn ACCU = L (0)
P	wenn Positiv-Flag = H (Negativ-Flag = L)
N	wenn Negativ-Flag = H
Z	wenn Zero-Flag = H
E	wenn Error-Flag = H

Aufbau:

```
CPB [Bedingung] Nummer     ; PB-Nummer 0-299
                           ; Bedingung: H|L|P|N|Z|E
```

Beispiel: CPB 10 ; unbedingter Aufruf des PB 10

Flags: Im aufgerufenen PB wird der ACCU = H gesetzt.
Bei der Rückkehr des Programms in den aufrufenden Block, wird der Zustand des ACCU wieder wie vor dem Aufruf hergestellt.

Siehe auch: PB, EPB, CFB, Anwender-Handbuch

Anwendung: IF .. THEN .. ELSE Struktur (WENN .. DANN .. SONST)

```
COB       0
          0
...
STH       I 15    ; WENN Eingang 15 = H
CPB       H 20    ;    DANN rufe PB 20 auf
CPB       L 25    ;    SONST rufe PB 25 auf
...
ECOB
-----------------------------------
PB        20
.....
EPB

PB        25
.....
EPB
```

CPBI Indirekter Aufruf eines Programm-Blocks *)

(Call Program Block Indirect)

Beschreibung: Bedingter oder unbedingter Aufruf eines Programm-Blocks, dessen Nummer im angegebenen Register liegt.
Da dieser Befehl einen Conditions-Code enthalten kann, wird der Mediacode "R" nicht geschrieben. Enthält das Register eine ungültige PB-Nummer (>299) oder existiert der PB nicht, wird das Error-Flag gesetzt und der XOB 13 aufgerufen (falls vorhanden).

Die Aufrufbedingungen können die folgenden sein:

Bedingung	**Programm-Block wird aufgerufen**:
-	ohne Bedingung (immer)
H	wenn ACCU = H (1)
L	wenn ACCU = L (0)
P	wenn Positiv-Flag = H (Negativ-Flag = L)
N	wenn Negativ-Flag = H
Z	wenn Zero-Flag = H
E	wenn Error-Flag = H

Aufbau:

```
CPBI [Bed]      Register    ; Nummer des Registers, welches die
                            ; PB-Nummer enthält
                            ; Bed. = H|L|P|N|Z|E
```

Beispiel:

```
CPBI    L 10    ; Ist der ACCU = L, wird der PB, dessen Nummer
                ; im Register 10 enthalten liegt, aufgerufen
```

Flags: Im aufgerufenen PB wird der ACCU = H gesetzt.
Bei der Rückkehr des Programms in den aufrufenden Block, wird der Zustand des ACCU wieder wie vor dem Aufruf hergestellt.

Siehe auch: PB, EPB, CFB, Anwender-Handbuch

*) ab Utility Version: V1.7

ab Firmware Version:

PCD6.M1/M2	V007
PCD6.M540	V002
PCD6.M3	alle
PCD4.M..	V003
PCD6.P800	V004
PCD1/2	alle

FB Funktions-Block
(Function Block)

Beschreibung: Kennzeichnet den Beginn eines Funktions-Blocks (FB) mit der entprechenden Nummer.
Ein FB ist ein Unterprogramm (Subroutine), welches mit oder ohne Parameter, bedingt oder unbedingt, aus einem andern Block (COB, PB, FB, XOB, SB) aufgerufen werden kann.
0 - 128 Parameter können beim Aufruf eines FB übergeben werden.

Aufbau:

FB	**Nummer**	**; FB-Nummer 0-999**

Beispiel:

```
FB      260     ; Beginn des FB 260
        ...
STH     = 1     ; parametrierter Befehl
        ...
EFB             ; Ende des FB 260
```

Flags: Bei jedem Blockanfang wird der ACCU = H gesetzt

Siehe auch: EFB, CFB, Anwender-Handbuch

Anwendung: Paramterierte Berechnung der Formel: Z = X * (X+Y)

```
FB        25)
ADD     = 1     ; X + Y = Z
        = 2
        = 3
MUL     = 3     ; Z * X = Z
        = 1
        = 3
EFB
-------------------------------------
COB       7
          0
...
STH     I 1     ; Wird Eingang 1 = H
DYN     F 1
CFB     H 25    ;   dann R107 = R100 * (R100+330)
        R 100   ; Parameter 1  (X)
        K 330   ; Parameter 2  (Y)
        R 107   ; Parameter 3  (Z)

STH     I 2     ; Wird Eingang 2 = H
DYN     F 2
CFB     H 25    ;   dann R107 = R200 * (R200+R201)
        R 200   ; Parameter 1  (X)
        R 201   ; Parameter 2  (Y)
        R 107   ; Parameter 3  (Z)
...
ECOB
```

EFB Ende eines Funktions-Blocks
(End of Function Block)

Beschreibung: Ende eines Funktions-Bocks (FB).

Der Programmablauf geht an der Aufrufstelle des FB +1 weiter.

Azfbau:

EFB	**; kein Operand**

Beispiel:

```
FB      0       ; Beginn des FB 0
        ...     ;    Porgramcode
        ...
        ...
EFB             ; Ende des FB 0
```

Flags: Im aufrufenden Block ist der Zustand des ACCU gleich wie vor dem Aufruf. Die Status-Flags können verändert sein.

Siehe auch: FB, CFB, Anwender-Handbuch

CFB Aufruf eines Funktions-Blocks
(Call Function Block)

Beschreibung: Bedingter oder unbedingter Aufruf eines Funktions-Blocks. Ist die Bedingung nicht erfüllt, wird der Funktions-Block nicht aufgerufen und das Programm fährt mit dem nächsten Befehl fort. Aufrufbedingungen siehe CPB, Seite 6.9.
Anschliessend an den Befehl kann eine Parameterliste angegeben werden. Es können max. 128 Parameter pro FB-Aufruf mitgegeben werden. Die Numerierung der Parameter ergibt sich aus der Reihenfolge der Auflistung. Die Numerierung der Parameter ist NICHT sichtbar und muss bei Bedarf als Kommentar angegeben werden. Pro Programmzeile kann nur 1 Parameter definiert werden. Die in der Parameterliste definierten Resourcen werden beim Aufruf des FB automatisch eingesetzt. Die Reihenfogle in der Parameterliste entspricht daher den Nummern (= 1, =2, ...) im FB.
Als Parameter können die folgenden Resourcen angegeben werden:

Type	Recource	Adressbereich
I	Input (Eingang)	0..8191
O	Output (Ausgang)	0..8191
F	Flag (Merker)	0..8191
C	Counter (Zähler)	0..1599
T	Timer Zeitglied)	0..450
R	Register	0..4095
K	K constant (Konstante)	0..16383
X	teXt	0..7999
DB	Data-Block	0..7999
S	Semaphore	0..99
W	Word (16 Bit für LDL, LDH)	0..65535
M	MOV (Datenformat)	Q 0..31 D 0..9 N 0..7 B 0..3 W 0..1 L 0

Aufbau:

```
CFB [Bedingung] Nummer      ; FB-Number 0-999
                            ; Bedingung: H|L|P|N|Z|E
        [param 1]           ; freiwillige
        [param 2]           ;  Parameterliste
        [param n]
```

Beispiel:

```
CFB     H 10  ; Aufruf von FB 10, wenn ACCU = H
        I 32  ;   Parameter 1
        R 10  ;   Parameter 2
```

Flags: Im aufgerufenen PB wird der ACCU = H gesetzt.
Bei der Rückkehr des Programms in den aufrufenden Block, wird der Zustand des ACCU wieder wie vor dem Aufruf hergestellt.

Siehe auch: FB, CPB, Anwender-Handbuch

NCOB Wechsle zum nächsten COB

(Next Cyclic Organisation Block)

Beschreibung: Bedingter oder unbedingter Wechsel zum nächsten COB. Ist die Bedingung nicht erfüllt, wird nicht zum nächsten COB gewechselt und das Programm fährt mit dem nächsten Befehl im ursprünglichen COB fort.

Mit diesem Befehl können die COB als Parallel-Programme, wie bei der PCA, verwendet werden. Bei jeder Wartebedingung ist ein NCOB einzufügen (siehe Anmerkung). Bei der nächsten Abarbeitung des ursprünglichen COB wird an der NCOB-Stelle + 1 weitergefahren. Der ACCU-Zustand von vor dem NCOB-Befehl bleibt erhalten.

Aufbau:

NCOB	**[Bedingung]**	**; Bedingung: H|L|P|N|Z|E**

Beispiel:

```
STH     I 15    ; Warte, solange I 15 = L
NCOB    L
JR      L -2
```

Flags: Der ACCU-Zustand von vor dem NCOB-Befehl bleibt erhalten.

Siehe auch: Anwender-Handbuch

Anmerkung:

In gut strukturierten Programmen kommt dieser Befehl **nicht** zur Anwendung. Sequentielle Prozess-Abläufe sind in GRAFTEC und **nicht** in Flussdiagramm mit Warteschleifen zu programmieren

SCOB Stoppe einen COB (alt)

(Stop Cyclic Organisation Block (old))

Beschreibung: Stoppt bedingt oder unbedingt den bezeichneten COB und schaltet die Programmabarbeitung zum nächstfolgenden COB um. Es kann auch der eigene COB gestoppt werden.

Ist die Bedingung nicht erfüllt, wird kein COB gestoppt.

Das Weiterlaufen eines gestoppten COB kann mit dem Befehl CCOB (Continue COB) aus einem andern COB veranlasst werden.

Aufbau:

```
SCOB  [Bedingungc]  COB-Nr.     ; COB-Nummer 0-15
                                ; Bedingung: H|L|P|N|Z|E
```

Beispiel:

```
SCOB      L   10    ; Stoppe COB 10 wenn ACCU = L (0)
```

Flags: Unverändert

See Also CCOB, Anwender-Handbuch

Tabelle: Firmware-Versionen

PCD-Typ	SCOB "alt": FW ≤ V...	SCOB "neu". FW ≥ V...
PCD1.M1xx	-	V001
PCD2.M110/M120	V003	V004
PCD2.M150	-	V0A0
PCD4.Mxx0	V005	-
PCD4.Mxx5	V00B	V00C
PCD4.M445	V001	V00C
PCD6.M540	V004	-
PCD6.M1/M2	V00A	-
PCD6.M3	-	V001

Anmerkung:

Dieser Befehl sollte nur in Ausnahmefällen verwendet werden. In gut strukturierten Programmen kann auf diesen Befehl immer verzichtet werden.

SCOB Stoppe einen COB (neu)

(Stop Cyclic Organisation Block (new))

Beschreibung: Stoppt bedingt oder unbedingt den bezeichneten COB.
Wird SCOB aus einem aktiven COB veranlasst, geht die Abarbeitung zum nächsten COB über, ansonst arbeitet das Anwenderprogramm bei der nächsten Programmzeile weiter.

Ist die Bedingung nicht erfüllt, wird kein COB gestoppt.

Das Weiterlaufen eines gestoppten COB kann mit dem Befehl CCOB (Continue COB) aus einem andern COB veranlasst werden.

Aufbau:

```
SCOB [Bedingung]  COB-Nr.      ; COB-Nummer 0 - 15
                               ; Bedingung: H|L|P|N|Z|E
```

Beispiel:

```
SCOB    L   10    ; Stoppe COB 10 wenn ACCU = L (0)
```

Flags: Unverändert

Siehe auch: CCOB, Anwender-Handbuch

Anmerkung:

Dieser Befehl sollte nur in Ausnahmefällen verwendet werden. In gut strukturierten Programmen kann auf diesen Befehl immer verzichtet werden.

CCOB Weiterarbeiten im COB

(Continue Cyclic Organisation Block)

Beschreibung: Lässt einen mit SCOB gestoppten COB bedingt oder undebingt weiterlaufen.

Ist die Bedingung nicht erfüllt, wird der bezeichnete COB nicht beeinflusst, und das Programm fährt mit dem nächsten Befehl fort.

Das Weiterlaufen eines gestoppten COB geschieht nicht unmittelbar, sondern erst dann, wenn dieser COB für die Abarbeitung wieder an der Reihe ist.

Aufbau:

```
CCOB [Bedingung] COB-Nr.      ; COB-Nummer 0 - 15
                              ; Bedingung: H|L|P|N|Z|
```

Beispiel:

```
CCOB    L 10    ; der gestoppte COB 10 läuft weiter, wenn ACCU = L (0)

CCOB    0       ; der gestoppte COB 0 läuft unbedingt weiter
```

Flags: Unverändert

Siehe auch: SCOB, Anwender-Handbuch

Anmerkung:

Die Befehle SCOB und CCOB sollten nur in Ausnahmefällen verwendet werden. In gut strukturierten Programmen kann auf diese Befehle immer verzichtet werden.

RCOB Neustart eines COB

(Restart Cyclic Organisation Block)

Beschreibung: Mit diesem Befehl kann jeder COB durch jeden anderen COB oder XOB, an definierter Stelle, bedingt oder unbedingt, neu gestartet werden. Ist die Bedingung nicht erfüllt, wird der Befehl nicht ausgeführt und das Programm fährt mit dem nächsten Befehl fort.

In der 1. Zeile wird im Operand die Nummer des neu zu startenden COB angegeben.

In der 2. Zeile wird die Anzahl Programmschitte vom Anfang des neu zu startenden COB angegeben. Die Angabe ist also **keine** absolute Adresse.

Die angegebene Anzahl Programmschritte darf nicht grösser als die Gesamtanzahl der Programmschritte des betreffenden COB sein.

Die Neustartstelle kann auch in einer labelähnlichen Form angegeben werden (siehe Anwender-Handbuch).

Aufbau:

```
RCOB [Bedingung] COB-Nr.          ; COB-Nummer 0 - 15
     Anzahl Programmzeilen        ; Anzahl Programmzeilen von Anfang des
                                  ;   COB (0-65535)
                                  ;   Bedingung: H|L|P|N|Z|E
```

Beispiel:

```
RCOB    5       ; unbedingter Neustart des COB 5
        10      ;   bei der 10. Programmzeile des COB 5
```

Flags: Unverändert

Siehe auch: Anwender-Handbuch

Anmerkung:

Dieser Befehl sollte nur in Ausnahmefällen verwendet werden. In gut strukturierten Programmen kann auf diesen Befehl immer verzichtet werden.

Notizen:

7 GRAFTEC-Befehle

Das SAIA-GRAFTEC ist ein graphisches Strukturierungs-Werkzeug zur Beschreibung und Dokumentierung von schrittweise (sequentiell) ablaufenden Prozessen.

Jedes GRAFTEC-Programm ist immer in einen SEQUENTIAL-Block (SB) eingebaut. Dieser SB wird normalerweise aus einem COB aufgerufen.

Ein GRAFTEC besteht aus STEPs und TRANSITIONEN, die sich ständig abwechseln.

SETPs enthalten Aktionen wie SET, RES, STXT usw.

TRANSITIONEN enthalten Weiterschaltbedingungen zum nächsten STEP wie z.B. STH, AHL usw.

Ein STEP wird nur ausgeführt, wenn die vorangehende TRANSITION erfüllt war, d.h. dass am Ende der TRANSITION der ACCU = H war.

Zum erfolgreichen Arbeiten mit GRAFTEC wird der SAIA GRAFTEC EDITOR verwendet. Mit diesem Editor wird die GRAFTEC-Struktur direkt am Bildschirm entworfen und danach automatisch in das Format einer Quelldatei gewandelt. Das fertige Programm kann “online” in der GRAFTEC-Struktur am Bildschirm verfolgt werden, was eine enorme Erleichterung bei der Inbetriebnahme und bei der Fehlersuche darstellt.

Mehr Informationen zum SAIA-GRAFTEC sind dem Anwender-Handbuch, Abschnitt “Die strukturierte Programmierung” zu entnehmen.

Die Operanden aller Befehle dieses Kapitels können nicht als Parameter an FB übergeben werden.

SB	**S**equential **B**lock	Sequential-Block
ESB	**E**nd of **S**equential **B**lock	Ende eines SB
CSB	**C**all **S**equenital **B**lock	Aufruf eines SB
RSB	**R**estart **S**equential **B**lock	Neustart eines SB
IST	**I**nitial **ST**ep	Initial-Step
ST	**ST**ep	Step
EST	**E**nd of **ST**ep	Ende eines Steps
TR	**TR**ansition	Transition
ETR	**E**nd of **TR**ansition	Ende einer Transition

Notizen

SB SEQUENTIAL-BLOCK (Sequential Block)

BESCHREIBUNG: Beginn eines Sequential-Blocks (SB).

Jeder SB enthält ein in sich geschlossenes GRAFTEC-Programm.

Ein SB enthält nur GRAFTEC-Befehle: IST, ST, TR, EST, ETR und ESB.

AUFBAU:

SB	**Nummer**	**; SB-Nummer 0-31**

BEISPIEL:

```
SB      10      ; Beginn des SB 10
...
ESB             ; Ende des SB 10
```

FLAGS: Unberührt, nicht von Bedeutung

SIEHE AUCH: ESB, CSB, RSB, IST, ST, TR, Anwender-Handbuch

ESB ENDE EINES SEQUENTIAL-BLOCKS (End of Sequential Block)

BESCHREIBUNG: Ende eines Sequential-Blocks (SB).

AUFBAU:

```
ESB                              ; Kein Operand
```

BEISPIEL:

```
SB      10      ; Beginn des SB 10
...
ESB             ; Ende des SB 10
```

FLAGS: Nach der Rückkehr in den aufrufenden Block (im allg. COB), wird der ACCU wieder in den Zustand von vor dem Aufruf des SB gebracht.

SIEHE AUCH: SB, ST, TR, Anwender-Handbuch

CSB AUFRUF EINES SEQUENTIAL-BLOCKS (Call Sequential Block)

BESCHREIBUNG: Bedingter oder unbedingter Aufruf eines Sequential-Blocks (SB). Ist die Bedingung nicht erfüllt, wird der SB nicht aufgerufen und das Programm fährt mit dem nächsten Befehl fort.

Ein SB wird normalerweise aus einem COB aufgerufen. Der Aufruf kann aber auch aus einem XOB, PB oder FB, jedoch NICHT aus einem andern SB erfolgen.

Die Aufrufbedingungen können die foglenden sein:

Bedingung	SB wird aufgerufen:
-	ohne Bedingung
H	wenn ACCU = H
L	wenn ACCU = L
P	wenn Positiv-Flag = H
N	wenn Negativ-Flag = H
Z	wenn Zero-Flag = H
E	wenn Error-Flag = H

AUFBAU:

```
CSB   [Bedingung]   Nummer    ; SB-Nummer 0-31
                              ;   Bedingung:
                              ;   H, L, P, N, Z, E
```

BEISPIEL:

```
CSB     L 10     ; Aufruf des SB 10, wenn ACCU = L
```

FLAGS: Im aufgerufenen SB bzw. in der Transition, die im SB nach dem Aufruf abgearbeitet wird, wird der ACCU = H gesetzt.

Bei der Rückkehr des Programms in den aufrufenden Block, wird der Zustand des ACCU wieder wie vor dem Aufruf hergestellt.

SIEHE AUCH: SB, CPB, Anwender-Handbuch

RSB NEUSTART EINES SEQUENTIAL-BLOCKS (Restart Sequential Block)

BESCHREIBUNG: Bedingter oder unbedingter Neustart eines Sequential-Blocks (SB). Ist die Bedingung nicht erfüllt, wird der SB nicht neu gestartet, und das Programm fährt mit dem nächsten Befehl fort.

In der 1. Zeile des Befehls wird die Bedingung und die Nummer des SB, der neu gestartet werden soll, angegeben.

In der 2. Zeile des Befehls wird die STEP-Nummer angegeben, wo das GRAFTEC neu beginnen soll.

Soll das GRAFTEC in einer Parallelverzweigung neu gestartet werden, so sind im Befehl auf mehreren Zeilen die STEP-Nummern jedes Parallelpfades aufzuführen.

Die Aufrufbedingungen können die folgenden sein:

Bedingung	RSB wird aufgerufen:
-	ohne Bedingung
H	wenn ACCU = H
L	wenn ACCU = L
P	wenn Positiv-Flag = H
N	wenn Negativ-Flag = H
Z	wenn Zero-Flag = H
E	wenn Error-Flag = H

AUFBAU:

```
RSB  [Bedingung]  Nummer     ; SB-Nummer 0-31
                             ;  Bedingung:
                             ;  H, L, P, N, Z, E
                  ST Nr.     ;  ST Nr. 0-1999
                  [ST Nr.]   ;  (ST Nr. 0-1999)
                  [. . .  ]  ;  (ST Nr. 0-1999)
                  [ST Nr.]   ;  (ST Nr. 0-1999)
```

BEISPIEL:

```
RSB    H 10    ; Neustart des SB 10 bei ST 350
         350   ;  wenn ACCU = H
```

FLAGS: Nach dem Neustart wird im SB bzw. im STEP der ACCU = H gesetzt.

SIEHE AUCH: SB, CSB, ST, Anwender-Handbuch

IST INITIAL STEP (Initial Step)

BESCHREIBUNG: Kennzeichnet den ersten auszuführenden STEP beim ersten Aufruf eines Sequential-Blocks (SB).

In jedem SB ist mindestens 1 IST erforderlich.

Alle andern Merkmale eines IST sind die gleichen wie bei den STEPs. (Siehe ST auf der nächsten Seite).

AUFBAU:

```
IST     Nummer    ; ST-Nummer 0-1999
        Liste     ;    GRAFTEC-Strukturparameter *)
                  ;    - der IST kommt aus TR . . .
                  ;    - der IST geht zu TR . . .
```

***)** die GRAFTEC-Strukturparameter werden vom GRAFTEC-Editor automatisch erzeugt.

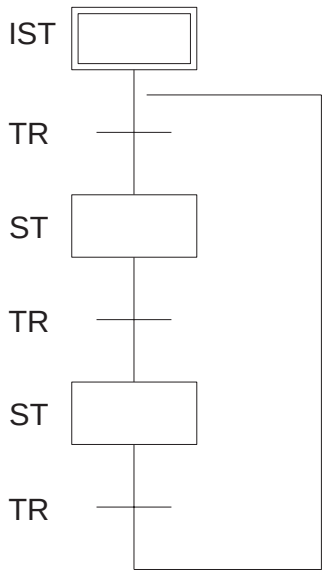
BEISPIEL:

```
IST      1     ; Initial-Step Nr. 1
     I  900    ;      kommt von TR 900
     O  2      ;      geht zu TR 2
```

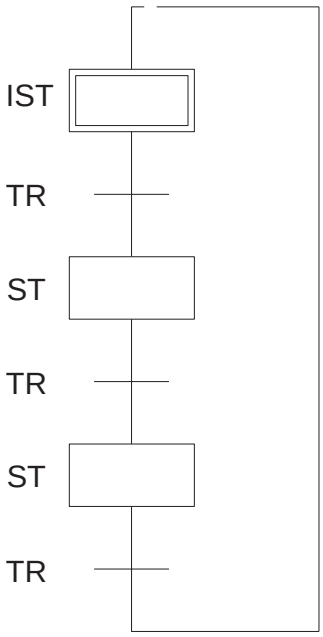
FLAGS: Am Anfang des IST wird der ACCU = H gesetzt.

SIEHE AUCH: EST, SB, ST. Anwender-Handbuch

ANWENDUNG: 2 Beispiele für die Anordnung eines IST.



Eigentliche Funktion des IST. Der IST wird nur beim ersten Aufruf des SB abgearbeitet.



Mögliche Stuktur. Der IST wird in jedem SB-Zyklus abgearbeitet.

ST STEP (Step)

BESCHREIBUNG: Kennzeichnet den Beginn eines STEP.

Ein STEP wird im SAIA-GRAFTEC 1-mal abgearbeitet. Danach wird der Programmablauf bei der nächsten TRANSITION fortgesetzt.

Ein STEP enthält grundsätzlich Aktionen wie SET, RES, LD, ADD, FMUL, STXT usw. aber normalerweise KEINE Abfragebefehle wie STH, ANL usw.

Aus einem STEP können PB und FB aufgerufen werden.

In einem STEP dürfen KEINE Warteschleifen programmiert werden.

Der Befehl STEP ist mehrzeilig. In der 1. Zeile steht die STEP-Nummer. In den nachfolgenden Zeilen stehen das “Woher” und das “Wohin” (Incomings und Outgoings), die sich immer auf TRANSITIONEN beziehen. Diese GRAFTEC-Strukturparameter werden vom GRAFTEC-Editor automatisch erzeugt. In den folgenden Zeilen steht der eigentliche Programm-Code. In der letzten Zeile steht EST (Ende STEP).

Ein STEP kann nur innerhalb eines SB programmiert werden.

AUFBAU:

```
ST      Nummer      ; ST-Nummer 0-1999
        Liste       ;    GRAFTEC-Strukturparameter *)
                    ;    - der ST kommt aus TR . . .
                    ;    - der ST geht zu TR . . .
```

***)** Die GRAFTEC-Strukturparameter werden vom GRAFTEC-Editor automatisch erzeugt.

Die Strukturparameter-Liste ist in ihrer Länge variabel, da aus mehreren TR in einen ST und aus einem ST in mehrere TR verzweigt werden kann.

BEISPIEL:

```
ST      10      ; Step Nr. 10
        I 9     ;    kommt von TR 9
        O 2     ;    geht zu TR 2
        O 5     ;    und zu TR 5
. . .           ;    Programm-Code

EST             ;    Ende des Steps 10
```

FLAGS: Am Anfang des ST wird der ACCU = H gesetzt.

SIEHE AUCH: EST, IST, TR, SB, Anwender-Handbuch.

EST ENDE EINES STEP (End of Step)

BESCHREIBUNG: Kennzeichnet das Ende eines STEP oder INITIALSTEP

AUFBAU:

```
EST                              ; Kein Operand *)
```

*) Der GRAFTEC-Editor setzt die Nummer des beendeten ST oder IST als Kommentar ein

BEISPIEL:

```
ST      10    ; Step Nr. 10
  I     9     ;    kommt von TR 9
  O     2     ;    geht zu TR 2
  O     5     ;    und zu TR 5

...           ; Programm-Code

EST           ; Ende des Steps 10
```

FLAGS: -

SIEHE AUCH: ST, TR, SB, Anwender-Handbuch

TR TRANSITION (Transition)

BESCHREIBUNG: Kennzeichnet den Beginn einer TRANSITION (TR).

Eine TR enthält die Weiterschaltbedingung. Diese kann aus einer einzigen Abfrage z.B. STH I 5, einer ganzen Verknüpfung mit ANH, ORL usw. oder einem ganzen Programmteil bestehen. Massgebend, ob eine TR erfüllt ist und der Programmablauf zum nächsten STEP weiter geht, ist der Zustand des ACCU am Ende der TR. Ist der ACCU = H, ist die TR erfüllt und das Programm geht zum nächsten STEP. Ist der ACCU = L wird der SB zum aufrufenden Block hin (normalerweise ein COB) verlassen. Beim nächsten Aufruf wird wieder die GANZE vorher nicht erfüllte TR abgearbeitet.

Aus einer TR können PB und FB aufgerufen werden.

In einer TR dürfen KEINE Warteschleifen programmiert werden.

Der Befehl TR ist mehrzeilig. In der 1. Zeile steht die TR-Nummer. In den nachfolgenden Zeilen stehen das “Woher” und das “Wohin” (Incomings und Outgoings), die sich immer auf STEPs beziehen. Diese GRAFTEC-Strukturparameter werden vom GRAFTEC-Editor automatisch erzeugt. In den folgenden Zeilen steht der eingentliche Programm-Code. In der letzten Zeile steht ETR (Ende TR).

Eine TR kann nur innerhalb eines SB programmiert werden.

AUFBAU:

```
TR      Nummer      ; TR-Nummer 0-1999
        Liste       ;   GRAFTEC-Strukturparameter *)
                    ;   - aus welchem ST kommt die TR
                    ;   - zu welchem ST geht die TR
```

***)** Die GRAFTEC-Strukturparameter werden vom GRAFTEC-Editor automatisch erzeugt.

Die Strukturparameter-Liste ist in ihrer Länge variabel, da aus mehreren ST in eine TR und aus einer TR in mehrere ST verzweigt werden kann.

BEISPIEL:

```
TR          25  ; TR Nr. 25
        I   19  ;   kommt von ST 19
        O   20  ;   geht zu ST 20

...             ; Programm-Code

ETR             ; Ende der TR 25
```

FLAGS: Am Anfang der TR wird der ACCU = H gesetzt.

SIEHE AUCH: ETR, ST, SB, Anwender-Handbuch.

ETR ENDE EINER TRANSITION (End of Transition)

BESCHREIBUNG: Kennzeichnet das Ende einer TRANSITION (TR)

AUFBAU:

ETR	**; kein Operand *)**

***)** Die GRAFTEC-Editor setzt die Nummer der beendeten TR als Kommentar ein.

BEISPIEL:

```
TR      25   ; TR Nr. 25
     I  19   ;    kommt von ST 19
     O  20   ;    geht zu ST 20

     ...     ; Programm-Code

ETR          ; Ende der TR 25
```

FLAGS: -

SIEHE AUCH: TR, ST, SB, Anwender-Handbuch

Notizen

8. Befehle für die SERIELLE KOMMUNIKATION

Diese Befehle können nur mit CPUs mit seriellen Schnittst. verwendet werden.

Bevor eine serielle Kommunikation ausgeführt werden kann, muss mit dem Befehl SASI die entsprechende Schnittstelle assigniert werden. Dies konfiguriert den Operationsmodus und die Übertragungsgeschwindigkeit. Jeder Kanal kann in einem andern Modus und mit einer andern Baudrate betrieben werden. Jede Schnittstelle hat ihre eigenen Empfangs- und Sendebuffer.

Modus	**Funktion**	Befehle
C	Senden/Empfangen von ASCII-Charaktern	STXD, SRXD
	Senden von Texten	STXT
MD/SD	Senden/Empfangen von Medien	STXM, SRXM
MM4	Senden/Empfangen von Registern über ein LAC-Netzwerk	STXM, SRXM
SBUS	Senden/Empfangen von Medien über ein SAIA BUS-Netzwerk	STXM(I), SRXM(I)
PROFIBUS	Senden/Empfangen von Medien über ein PROFIBUS-Netzwerk	STXM(I), SRXM(I) SCON(I),
OFF	De-Assignieren einer seriellen Schnittstelle	-

Die Steuer- und Meldeleitungen (CTS, RTS, DSR, DTR und DCD) können auf deren logischen Zustand abgefragt oder gesetzt werden (SICL, SOCL). SCON ermöglicht das Öffnen und Schliessen eines virtuellen PROFIBUS-Kanals. Auch eine Kommunikation via das LAN1 ist mit Hilfe von SCON möglich.

SASI Assignierung einer seriellen Schnittstelle
SASII Indirekte Assignierung einer seriellen Schnittst.

SRXD Empfang eines Charakters
STXD Senden eines Charakters
STXT Senden von Anwender-Text

SRXM Empfang von Medien
SRXMI Indirekter Empfang von Medien
STXM Senden von Medien
STXMI Indirektes Senden von Medien

SICL Prüfen von Steuersignalen
SOCL Setzen von Steuersignalen

SCON Öffnen und Schliessen eines Kanals für LAN 1 und PROFIBUS
SCONI Indirektes Öffnen und Schliessen eines Kanals für LAN 1 und PROFIBUS

MODE C

Charakter- oder Text-Modus:

- Ausgabe von Charaktern aus einem Register oder Textausgabe
- Empfang und Senden von Einzelcharaktern via Register
- Verwendung für den Dialog mit einem Terminal oder einem Drucker

MC0 **Mode C ohne automatisches Handshaking**

Der Anwender muss im Bedarfsfall die Steuersignale mit den Befehlen SICL und SOCL selbst steuern

MC1 **Mode C mit RTS und CTS Handshaking**

Das RTS-Signal wird von der PCD automatisch gesteuert.
Das CTS-Signal beeinflusst die Übertragung in der PCD

RTS	Low (L)	Der Empfangsbuffer enthält mehr als 450 Char.
	High (H)	Der Empfangsbuffer enthält weniger als 450 Char.
CTS	Low (L)	Übertragung gestoppt
	High (H)	Übertragung freigegeben

MC2 **Mode C mit Xon/Xoff Protokoll**

Dieser Modus ist ähnlich dem RTS/CTS-Handshaking. Dieser Modus wird verwendet, wenn keine Steuerleitungen zur Verfügung stehen.
Zwei Spezialzeichen 'Xon' (Ctrl/Q) und 'Xoff' (Ctrl/S) werden zur Steuerung der Übertragung des Partners gesendet.

Empfänger sendet, wenn		
	Xoff	Der Empfangsbuffer mehr als 450 Char. enthält
	Xon	Der Empfangsbuffer weniger als 450 Char. enthält
Sender empfängt, wenn		
	Xoff	Übertragung gestoppt ist
	Xon	Übertragung freigegeben ist

MC3 **Mode C mit Echo**

Dieser Modus kann bei der Kommunikation mit einem Terminal angewendet werden. Alle von der Tastatur empfangenen Charakter werden zum Bildschirm zurückgesandt.

MC4/MC5 Mode C für RS485-Interface

Die MC4-/MC5-Modi setzen den RS485-Treiber/Empfänger nur dann in den Sendemodus, wenn Information (Charakter/Text) gesendet wird. Ansonsten ist das Interface auf Empfang.

Die Flags XBSY und TBSY werden jedoch zu unterschiedlichen Zeitpunkten zurückgesetzt:

Bei MC4:

- Das Zurücksetzen in den Empfangsmodus erfolgt zwischen 0 und 1 ms nach dem Ende des letzten gesendeten Zeichens
- Die Flags XBSY/TBSY werden zwischen 0 und 1 ms nach dem Ende des letzten gesendeten Zeichens auf 0 zurückgesetzt.

Bei MC5:

- Das Zurücksetzen in den Empfangsmodus erfolgt 1 Zeiteinheit/Bit (Zeit die für die Übertragung eines Bits benötigt wird) nach dem **Ende** des letzten gesendeten Zeichens.
- Die Flags XBSY/TBSY werden zwischen 0 und 1 ms nach dem **Anfang** des letzten gesendeten Zeichens auf 0 zurückgesetzt.

MODE D

In Übereinstimmung mit ISO 1745, IBM BSC und DIN 66019.

PCD-spezifische Daten können zwischen PCD-Systemen oder zwischen einer PCD und einem andern intelligenten System (z.B. IBM PC oder so) direkt oder via einem SAIA® LAN1 ausgetauscht werden.

PCD-Daten sind die Status von Ein-/Ausgängen, Flags oder der Inhalt von Registern, Timern oder Counern

<mode>	Beschreibung
MD0	Modus D master
SD0	Modus D slave

Der Unterschied zwischen MD0 und SD0 ist der, dass bei einem Übertragungskonflikt die Master-Station den Vorrang hat.

Bei der Kommunikation mit einem Personal-Computer muss die PCD im SD0-Modus arbeiten.

Eine Beschreibung des kompletten Protokolls kann den "Functional specification for the SAIA® P8 Protocol" entnommen werden.

SAIA LAN 1

Der D-Modus kann mit dem LAN 1 realisiert werden. Für diesen Fall wird die Verbindung mit dem Befehl SCON eröffnet bzw. geschlossen.

Der Status der Verbindung wird durch das LAN 1 automatisch in ein Register, welches im SASI-Befehl definiert wurde, eingeschrieben.

Mehr Information kann dem SAIA® LAN1-Handbuch entnommen werden.

MODE MM4

Dieser Modus wird zusammen mit dem LAC- und dem LAC2- Netzwerk der Firma COMPEX angewendet, kann jedoch auch zum direkten Datenaustausch zwischen 2 PCD (ohne LAC-Netzwerk) verwendet werden.

Der Datentransfer beschränkt sich beim MM4-Modus auf den Austausch von Registerwerten zu 32 Bit = 4 ASCII-Charakter. Pro Telegramm können max. 64 Register über alle Schnittstellen der PCD übertragen werden: 20mA-Stromschleife, RS232/422/485.

Das MM4-Protokoll beinhaltet eine Überwachung des Datenverkehrs nach CRC-16 und hat ausserdem eine ausführliche Fehlerdiagnose.

Der MM4-Modus verwendet die Befehle SASI, SRXM, STXM sowie SICL und SOCL.

Für weitergehende Informationen ist die "Description of the LAC MM4 Protocol" zu konsultieren.

MODE SBUS

S-Bus ist primär die Bezeichnung für ein effizientes Kommunikationsprotokoll für die SAIA® PCD Steuergeräte-Generation. Der S-Bus kann einerseits in der Punkt/Punkt-Kommunikation als auch in einem lokalen Master/Slave-Netzwerk eingesetzt werden.

Im Punkt/Punkt-Verkehr können alle seriellen PCD-Schnittstellen verwendet werden.

Als physikalisches Übertragungsmedium für das Netzwerk dient die busfähige Schnittstelle RS485 mit einem 2-adrigen, verdrillten und abgeschirmten Kabel. Mit dem S-Bus können bis zu 255 PCD-Systeme, aufgeteilt in bis zu 8 Segmente, bestehend aus je max. 32 Stationen, einfach und kostengünstig miteinander vernetzt werden.

Der S-Bus zeichnet sich besonders durch die folgenden Eigenschaften aus:

- Einfache Handhabung (Installation, Inbetriebnahme und Anwenderprogr.)
- Günstiger Preis, da das S-Bus Protokoll in jedem Prozessor enthalten ist.
- Hohe Übertragungssicherheit dank der CRC-16 Fehlererkennung.
- Hohe Datenübertragungsrate dank Verwendung des effizienten binären Protokolls mit einer Übertragungsgeschwindigkeit bis zu 38.4 kBd.
- Daten-Fern-Übertragung und -Diagnose via Modem wird unterstützt.
- Für Prozessleitsysteme z. B. von WIZCON, Genesis, InTouch oder FactoryLink und Fix D-Macs sind Treiber vorhanden.
- Mit der Anwendungsstufe 2 hat auch die Programmiereinheit Zugriff auf jede Slave-Station im lokalen Netz. Dadurch können die Funktionen des Programmiergerätes (z.B. der Debugger) für alle angeschlossenen Slave-Stationen von einem zentralen Ort aus über das ganze Netzwerk genutzt werden.
- Multi-Master Möglichkeiten durch Einsatz von S-Bus Gateway

<mode>	**Beschreibung**
SM2	S-Bus Master, mit Data Modus
SM1	S-Bus Master, mit Parity Bit Control
SM0	S-Bus Master, ohne Parity, mit Break Character
SS2	S-Bus Slave, mit Data Modus
SS1	S-Bus Slave, mit Parity Bit Control
SS0	S-Bus Slave, ohne Parity, mit Break Character
GS2	S-Bus Gateway Slave, mit Data Modus
GS1	S-Bus Gateway Slave, mit Parity Bit Control
GS0	S-Bus Gateway Slave, ohne Parity, mit Break Char.
GM	S-Bus Gateway Master
OFF	De-Initialisierung der seriellen Schnittstelle

Für mehr Informationen ist das "SAIA® S-Bus Handbuch" zu konsultieren (Bestellnummer 26/739).

PROFIBUS

PROFIBUS ist der erfolgreiche, offene Feldbusstandard für ein breites Anwendungsspektrum.

PROFIBUS ermöglicht den Datenaustausch zwischen Geräten unterschiedlicher Hersteller ohne spezielle Schnittstellenanpassungen.

PROFIBUS ist standardisiert als stabile Deutsche Norm DIN 19 245 und als europäische Norm pr EN 50170.

PROFIBUS ist firmenneutral und Geräte werden von einer Vielzahl qualifizierter Hersteller angeboten.

PROFIBUS besteht aus den 3 folgenden anwendungsspezifischen Varianten:

- PROFIBUS-DP Dezentrale Peripherie
- PROFIBUS-FMS Fieldbus Message Specification
- PROFIBUS-PA Prozess Automation

SAIA® PCD arbeitet mit PROFIBUS-FMS und -DP (Master/Slave)

PROFIBUS-FMS ist die universelle Lösung für Kommunikationsaufgaben in der Feld- und Zellenebene der industriellen Kommunikationshirarchie.

Die Verwendung des PROFIBUS mit der SAIA® PCD erfordert einen dafür vorgesehenen Prozessor PCD4.M445 oder ein PCD7.F700 Modul.

PROFIBUS-DP ist punkto Geschwindigkeit optimiert und erfüllt die Forderungen für eine Kommunikation zwischen Automatisierungssystemen und dezentralen Peripheriegeräten. Die Verwendung eines PROFIBUS-DP Netzwerks (Master oder Slave) mit SAIA® PCD erfordert ein Modul PCD7.F750 bzw. ..F770.

Die Definition und Konfiguration (Busparameter, Kommunkationsbeziehungsliste und Objektverzeichnis) eines PROFIBUS-Netzwerks kann, je nach Grösse des Projekts, recht umfangreich sein. Diese Aufgabe wird dem Anwender durch die Verwendung des unter WINDOWS laufenden PROFIBUS-Konfigurators erheblich erleichtert. Der Konfigurator erstellt eine Datei mit den Definitions-Texten für alle PROFIBUS-Kanäle einer Station. Diese Textdatei wird im SASI-Befehl des PROFIBUS-Kanals eingesetzt.

Für mehr Informationen sind nachfolgend erwähnten Handbücher zu konsultieren:

"SAIA® PROFIBUS-FMS" (Bestellnummer 26/742 D)
"SAIA® PROFIBUS-DP" (Bestellnummer 26/765 D)

SASI ASSIGNIERUNG SERIELLE SCHNITTSTELLE

(Assign Serial Interface)

Beschreibung: Initialisiert eine serielle Schnittstelle.
Der 1. Operand ist die Nummer der Schnittstelle
Der Operand der 2. Zeile bezeichnet einen Text, welcher die verschiedenen Definitionen der Schnittstelle enthält (siehe folgende Seiten).
Die Initialisierung ist für jede einzelne Schnittstelle durchzuführen. Die SASI-Befehle stehen normalerweise im XOB 16 und werden somit nur beim Einschalten der Steuerung ausgeführt. Für die Assignierung des PROFIBUS ist das "SAIA® PROFIBUS"-Handbuch (26/742) zu konsultieren.

Aufbau:

SASI	**Kanal**	**; Nummer der Schnittst. 0-3, [PROFIBUS: 10-99]**
	Text	**; Text-Nummer 0-3999**

Beispiel:

```
SASI    0       ; Assigniert Kanal 0
        100     ;  Informationen im Text 100
```

Flags: Das ERROR Flag wird gesetzt wenn der SASI-Befehl fehlt oder fehlerhaft ist

Siehe auch: SASI-Texte

Anmerkung: Für PROFIBUS ist die Kanalnummer (10..99) der virtuelle Kanal, auch CREF (Communication Relationship) genannt. Die notwendige Information zur Assignierung steht auch in einem Text. Dieser Text wird vom PROFIBUS-Konfigurator automatisch erzeugt.

Anwendung: Es ist die Schnittstelle Nr. 1 für die Textausgabe mit einer Übertragungsgeschwindigkeit von 4800 Bit/sek, mit 7 Daten-Bit pro Charakter, gerader Priorität und 1 Stop-Bit zu assignieren.

Der SASI-Befehl ist im XOB 16 zu programmieren

```
        XOB     16      ; Kaltstart-Routine
        SASI    1       ; Assignierung Kanal 1
                10      ;     Parameter in Text 10
        EXOB

TEXT 10 "UART:4800,7,E,1;"
        "MODE:MC0;"
        "DIAG:F1000,R4000;"
```

Flags 1000..1007 sind die Diagnose-Flags, R 4000 enthält die Detaildiagnose

Der SASI-Text kann auch 1-zeilig geschrieben werden:

```
TEXT 10 "UART:4800,7,E,1;MODE:MC0;DIAG:F1000,R4000;"
```

SASI-Texte

Zur Initialisierung der Schnittstellen sind die Definitions-Texte in einem bestimmten Format zu schreiben.

FORMAT:

TEXT xxxx	**"<uart_def>;"**
	"<mode_def>;"
	"<diag_def>;"
	["<rx_buf>;"]
	["<tx_buf>;"]

Darin bedeuten:

xxxx	Text-Nummer 0 .. 3999
<uart_def>	Definiert die Baudrate, Charakter-Länge, Parität, Anzahl Stop-Bit und Timeout
<mode_def>	Übertragungs-Modus
<diag_def>	Diagnose-Flags und Diagnose-Register

Die nächsten 2 Parameter sind optional und nur im Modus 'C' anwendbar:

<rx_buf>	Füllhöhe des Empfangs-Buffers (Default = 1)
<tx_buf>	Füllhöhe des Sende-Buffers

Die SASI-Texte können auch in einer einzigen Zeile geschrieben werden.

MODE OFF

Dieser Modus weicht von der Schreibweise der Standard-SASI-Texte ab und kommt dann zur Anwendung, wenn eine bereits assignierte Schnittstelle neu assiginiert werden soll.

Um Kollisionen zwischen den verschiedenen Schnittstellen zu verhindern, wurde ein Sempaphore-Mechanismus implementiert. Wird eine Schnittstelle assigniert, wird ein Semaphore-Flag gesetzt. Wird versucht, dieselbe Schnittstelle nochmals zu assignieren, wird das ERROR-Flag aktiviert, der XOB 13 aufgerufen und der Befehl abgebrochen. Eine Schnittstelle kann nur nach einer vorgängigen De-Assignierung neu assigniert werden. Diese De-Assignierung wird mit einem SASI-Befehl mit dem folgenden Text durchgeführt:

TEXT xxxx "MODE:OFF"

Wird versucht, eine bereits de-assignierte Schnittestlle noch einmal zu de-assignieren, wird ebenfalls das ERROR-Flag gesetzt und der XOB 13 aufgerufen.

<uart_def>

Enthält die Baudrate, Charakter-Länge, Parität, Anzahl Stop-Bit, Timeout.:

Format:
"UART:<baud-rate>,<char_len>,<parity>,<stop_bit>[,<timeout>];

<baud_rate>	<char_len>	<parity>	<stop_bit>	<time_out>	or by default
110	7	E (even)	1	10..15000 ms	15 s
150	8	O (odd)	2		9 s
300		L (low)			5 s
600		H (high)			3 s
1.200		N (none)			2 s
2.400					1 s
4.800					0,5 s
9.600					0,25 s
19.200					0,2 s
38.400					0,1 s

even = gerade, odd = ungerade none = keine Parität

<time_out>-Werte werden in ms angegeben und gelten nicht für den Modus 'C'

In den anderen Modi gibt das Timeout die Zeitspanne an, welche verstreicht, bis ein Telegramm wiederholt wird, wenn keine positive Rückantwort gekommen ist. Kommt auch nach dem 3. Versuch keine gültige Rückantwort, wird ein Diagnose-Flag und das Bit 'keine Antwort' im Diagnose-Register aktiviert.

Mode S-BUS:
Der S-Bus-Modus verwendet immer 11 Bit pro Charakter (10 Bit im 'Data'-Mode). Es gibt deshalb keine Angaben für <char_len>, <parity>, <stop_bit>.

"UART:<Baudrate>[,<Timeout>][,<TS-Delay>][,<TN-Delay>][,<Break-Length>];"

<baud_rate>: 110 .. 38'400
<time_out>: 10 .. 15000 ms
<TS delay>: 10 .. 15000 ms
<Break-Length>: 4 .. 15 char

TimeOut, TS-Delay, TN-Delay und Break-Length sind optional und werden normalerweise nur in Spezialfällen verwendet.

Für mehr Inform. ist das "SAIA® S-Bus-Handbuch" (26/739) zu konsultieren

Beispiel für <uart_def>-Text:

```
"UART:9600,7,E,1;"
```

9600 Bauds, 7 Daten-Bit, Even Parity, 1 Stopbit und Standard-Timeout

Bemerkung: Alle Charakter müssen in Grossbuchstaben notiert werden. Steht der SASI-Text zwischen '$sasi'- und '$endsasi'-Direktiven prüft der Assembler die korrekte Syntax und wandelt alle Charakter in Grossbuchstaben.

<mode_def>

Definiert den Modus der seriellen Schnittstelle

Format: **"MODE: >[,<mode_opt>];"**

<mode_opt> ist eine optionale Angabe von Parametern und wird nicht bei allen Modi verwendet.

<mode>	**<mode_opt>**	**Beschreibung**
MC0	-	Mode C ohne automatisches Handshaking
MC1	-	Mode C mit RTS und CTS Handshaking
MC2	-	Mode C mit Xon/Xoff-Protokoll
MC3	-	Mode C mit Echo
MC4	-	Mode C für RS485 Interface
MD0	-	Mode D Master
	R xxxx (1)	via LAN1; Register = SCON-Status
SD0	-	Mode D Slave
	R xxxx (1)	via LAN1; Register = SCON-Status
SM2	R xxxx (2)	Mode SBUS Master (client) Data-Mode Partner -
SM1	R xxxx (2)	Mode SBUS Master (client) Parity-Mode station
SM0	R xxxx (2)	Break-Mode (Remote)
SS2	-	Mode SBUS Slave (server) Data-Mode
SS1	-	Mode SBUS Slave (server) Parity-Mode
SS0	-	Break-Mode
GM	-	Mode SBUS Gateway Master
GS2	-	Mode SBUS Gateway Slave Data-Mode
GS1	-	Mode SBUS Gateway Slave Parity-Mode
GS0	-	Break-Mode
MM4	(3)	Mode MM4
OFF	-	De-Assignierung von Schnittstellen

(1) D Mode mit LAN1
Wird über LAN1 kommuniziert, muss ein Zusatz-Register definiert werden, in welchem der Kommunikations-Status abgelegt wird. (Siehe Befehl SCON)

(2) SBUS Client
Die Adresse der Partner-Station liegt in einem Register.

(3) MM4 <mode_opt> besteht aus:
<BCS_opt>,<trpartner>,<trinfo>,<repartner>,<reinfo>,<rechar>

<mode_opt>	**Wert**	**Beschreibung**
<BCS_opt>	0 oder 1	Block Check Sum (0: no BCS, 1: CRC-16
<trpartner>	R xxxx	Transmission partner station number
<trinfo>	R xxxx	Remote ACK information
<repartner>	R xxxx	Reception partner station number
<reinfo>	R xxxx	Receive information
<rechar>	R xxxx	Number of received characters

<diag_def>

Definiert die Elemente für die Kommunikations-Diagnose.
Format: **"DIAG:<dia_elem>,<dia_reg>;"**

	Type	**Beschreibung**
<dia_elem>	O xxxx F xxxx	Basisadresse von 8 aufeinanderfolgenden Flags (oder Ausgängen)
<dia_reg>	R xxxx	Register für die Fein-Diagnose

xxxx sind Element-Adressen

Die 8 Flags liefern Informationen zum Status der seriellen Schnittstelle. Im Fehlerfall (eines der Flags bleibt = H) kann im Diagnose-Register mehr über die mögliche Ursache des Fehlers erfahren werden.

DIAGNOSE FLAGS

Das bei DIAG angegebene Flag (oder Ausgang) ist die Basisadresse von 8 aufeinanderfolgenden Flags, deren Bedeutung in der nachfolgenden Zusammenstellung gezeigt ist:

Adresse	**Name**	**Beschreibung**
xxxx	**RBSY**	Receiver busy
xxxx+1	**RFUL**	Receive buffer full
xxxx+2	**RDIA**	Receiver diagnostic
xxxx+3	**TBSY**	Transmitter busy
xxxx+4	**TFUL**	Transmit buffer full
xxxx+5	**TDIA**	Transmitter diagnostic
xxxx+6	**XBSY**	Text Busy
xxxx+7	**NEXE**	Not executed

RBSY **Receiver Busy** (Empfänger belegt)

Mode	
C	RBSY ist = H, wenn mindestens 1 Charakter im Empfangsbuffer liegt. Nachdem alle angekommenen Charakter mit dem Befehl SRXD aus dem Buffer herausgelesen wurden, ist RSBY = L.
D **MM4**	RBSY ist = H, solange der Empfänger beschäftigt ist.
SBUS	RBSY ist = H, wenn eine Slave-Station ein Telegramm erhält. Das Flag wird = L, wenn die Antwort gesendet wurde.
PROFIBUS	Nicht verwendet.

RFUL **Receive Buffer Full** (Füllhöhe des Empfangsbuffers erreicht)

Mode	
C	RFUL ist = H, wenn die im SASI-Befehl definierte (optionale) Füllhöhe 'rx_buf' erreicht oder überschritten ist. Wird die Füllhöhe kleiner als definiert, wird RFUL = L. Der Empfangsbuffer hat immer, d.h. unabhängig von der definierten Füllhöhe, eine Kapazität von 512 Charaktern.
D **MM4**	RFUL = H, wenn ein gültiges Datenpaket empfangen wurde.
SBUS	RFUL = H, wenn von der Master-Station Elemente in der Slave-Station verändert wurden.
PROFIBUS	RFUL = H, wenn ein Schreibtelegramm empfangen wurde

RDIA **Receiver Diagnostic** (Empfänger Diagnose)

Mode	
C **D** **SBUS** **MM4** **PROFIBUS**	RDIA wird = H, wenn während dem Empfang von Charaktern irgendwelche Fehler festgestellt werden. Eine Fein-Diagnose kann dem mit 'dia_reg' bezeichneten Register entnommen werden. RDIA wird zurückstellt, wenn alle Bit der Empfänger-Diagnose (0..15) = 0 sind.

TBSY **Transmitter Busy** (Sendebuffer belegt)

Mode	
C	TBSY = H, währenddem die PCD über die serielle Schnittstelle Daten sendet. TBSY wird wieder = L, wenn alle zu sendenden Daten gesendet sind.
D **SBUS** **MM4** **PROFIBUS**	TBSY = H, währenddem die PCD Daten sendet. TBSY wird wieder = L, wenn das Telegramm quittiert wurde oder wenn die Anzahl Sendeversuche erreicht ist.

TFUL **Transmit Buffer Full** (Füllhöhe des Sendebuffers erreicht)

Mode	
C	TFUL = H, wenn die im SASI-Befehl definierte (optionale) Füllhöhe 'tx_buf' erreicht oder überschritten ist. Wird die definierte Füllhöhe wieder unterschritten, wird TFUL = L. Der Sendebuffer hat immer, d.h. unabhängig von der definierten Füllhöhe, eine Kapazität von 512 Charaktern.
D	Nicht verwendet
SBUS	
PROFIBUS	
MM4	TFUL = H, wenn die Quittung des gesendeten Telegramms eingetroffen ist.

TDIA **Transmitter Diagnostic** (Sender Diagnose)

Mode	
C **D** **SBUS** **MM4** **PROFIBUS**	TDIA wird = H, wenn während dem Senden von Charaktern irgendwelche Fehler festgestellt werden. Eine Fein-Diagnose kann dem mit 'dia-reg' bezeichneten Register entnommen werden.

XBSY **Text busy** (Übertragung aus Textspeicher)

Mode	
C	XBSY ist = H, wenn die PCD Text über die serielle Schnittstelle sendet. Ist der Text gesendet, wird XBSY automatisch wieder = L. **Achtung:** das XBSY wird = L beim **Beginn** des letzten Textcharakters.
D	XBSY = H, solange eine Verbindung über das SAIA-LAN1 geöffnet ist.
SBUS	XBSY = L, wenn der Anwender die Erlaubnis zur Durchführung eines SASI OFF-Befehls hat.
MM4	XBSY = H, während einer Aktivität (ausgeführter STXM-Befehl) auf dem LAC-Netzwerk.
PROFIBUS	XBSY = H, wenn der virtuelle Kanal geöffnet ist

NEXE **Not executed** (Nicht ausgeführt)

Mode	
C	
D **SBUS** **MM4**	Kann ein Auftrag nicht ausgeführt werden, wird das NEXE-Flag = H. Weitere Informationen über den aktuellen Fehler liefert das Diagnose-Register, das im SASI-Befehl unter 'dia-reg' definiert wurde.
PROFIBUS	NEXE = H, wenn nach 3 Versuchen ein Befehl (STXM, RSXM) nicht ausgeführt werden konnte. Das Flag wird nach dem nächsten Befehl wieder zurückgesetzt.

DIAGNOSE-REGISTER DER KOMMUNIKATION **<dia reg>**

Die Adresse des Detail-Diagnose-Registers wird im SASI-Befehl angegeben. Normalerweise sind alle 32 Bit dieses Registers = L. Ist ein Bit = H, liegt ein Fehler vor. Die folgende Tabelle gibt Aufschluss über die Fehlerursache im betreffenden Modus. Das Diagnose-Register wird **nicht** automatisch rückgesetzt. Der Anwender muss dies im Programm vorsehen.

Bit	Bedeutung	Ursache	Modus				
			C	D	S-BUS	MM4	P-BUS
0	Overrun error	Überlauf des internen Empfangsbuffers)	●	●	●	●	
1	Parity error	Paritätsfehler	●	●		●	
2	Framing error	Wahrscheinlich falsche Baudrate	●	●	●	●	
3	Break	Datenleitung unterbrochen	●	●	●	●	
4	BCC error	Fehler im Telegramm BCC oder CRC-16		●	●	●	
5	S-Bus PGU status	S-Bus PGU mit Public-Line Modem			●		
6	End of transmit	Übertragung beendet		●			
		SASI-OFF			●		
7	Overflow error	Überlauf des Empfangsbuffers	●	●		●	
8	Length error	Unkorrekte Telegrammlänge			●		
9	Format error	Unkorrektes Telegrammformat		●		●	●
10	Address error	Adresse der Rückantwort ungültig			●	●	
11	Status error	PCD in falschem Status			●		
12	Range error	Ungültiger Elementbereich		●	●		●
13	Value error	Ungültiger Datenwert			●		
14	Missing media err	Medienadresse fehlt oder falsch			●		
15	Program error	Lesen des leeren Empfangsbuffers	●				
		LAN 1 nicht assigniert oder falsche Nr.		●	●		
16	Retry count	Anzahl Telegrammwiederholungen			●		
17							
18	Transmission off	Sendung unterbrochen (CTS = L / XOFF)	●				
19							
20	NAK response	Negative Rückantwort (NAK)		●	●	●	●
21	No response	Keine Antwort nach Timeout		●	●	●	●
22	Multiple NAK	NAK nach mehreren Versuchen		●	●	●	
23	TX buffer full	Sendebuffer voll	●				
	TS Delay	Kein CTS nach TS-Delay			●		
24	Enquiry error	Keine Antwort zu ENQ nach mehreren V.		●			
25	Format error	Ungültiger SASI-Text	●				
		Ungültiger Befehl		●			●
26	Partner error	Schwierigkeiten mit Partner-Station				●	
27	Network error	Schwierigkeiten mit Netzwerk				●	
28	Range error	Ungültige Element-Adresse		●	●	●	●
29							
30	Receive error	Fehler beim Empfang		●			●
31	Program error	Unerlaubter Sendeversuch		●	●	●	●

<rx_buf>

<rx-buf> Empfangs-Buffer (nur im Modus "C" verwendet)

Mit dieser Angabe wird die obere Füllgrenze des Empfangs-Buffers bestimmt.

Format: **"RBUF:<rbuf_len>;"**

	Wert	**Beschreibung**
<rbuf_len>	1.. 511	Länge des Eingangs-Buffers

Der Empfangsbuffer hat immer eine Kapazität von 512 x 8 Bit = 512 Charakter.

Die Angabe 1 - 511 gibt die Füllhöhe an, bei wievielen Charaktern im Buffer das RFUL-Flag gesetzt wird.

Wird keine Angabe gemacht, beträgt die Füllhöhe = 1, d.h. dass nach einem einzigen empfangenen Charakter das RFUL-Flag bereits gesetzt wird.

<tx_buf>

<tx-buf> Sende-Buffer (nur im Modus "C" verwendet)

Es wird mit dieser Angabe die obere Füllgrenze des Sende-Buffers bestimmt

Format: **"TBUF:<tbuf_len>;"**

	Wert	**Beschreibung**
<tbuf_len>	1.. 511	Länge des Sende-Buffers

Der Sende-Buffer hat immer eine Kapazität von 512 x 8 Bit = 512 Charakter.

Die Angabe 1 - 511 gibt die Füllhöhe an, bei wievielen Charaktern im Buffer das TFUL-Flag gesetzt wird.

Wird keine Angabe gemacht, beträgt die Füllhöhe = 1.

Beispiele für SASI-Texte

Mode MC0 Modus MC0 mit 9600 Baud, 7 Data-Bit, gerade Parität, 1 Stop-Bit, Diagnose-Flags F 1000- 1007, Diagnose-Register R 4000.

```
TEXT 10     "UART:9600,7,E,1;MODE:MC0;DIAG:F1000,R4000;"
```

Mode MC2 Modus MC2 mit 4800 Baud, 8 Data-Bit, keine Parität, 1 Stop-Bit, Diagnose-Flags 0 32-39, Diagnose-Register R 100, Empfangsbuffer-Länge = 25 Charakter.

```
TEXT 20     "UART:4800,8,N,1;MODE:MC2;DIAG:O32,R100;"
            "RBUF:25;"
```

Mode SD0 Modus SD0 (Slave) mit 19200 Baud, 8 Data-Bit, gerade Parität, 1 Stop-Bit, Diagnose-Flags F 1000- 1007, Diagnose-Register R 999.

```
TEXT 30     "UART:19200,8,E,1;MODE:SD0;DIAG:F1000,R999;"
```

Mode MD0 Modus MD0 (Master) mit 9600 Baud, 8 Data-Bit, keine Parität, 1 Stop-Bit, Diagnose-Flags F1000-1007, Diagnose-Register R 4000. Zusätzlich wird das LAN1 verwendet und ein Timeout von 3 sek. soll angegeben werden.

```
TEXT 40     "UART:9600,8,N,1,3000;"
            "MODE:MD0,R1;"
            "DIAG:F1000,R4000;"
```

In Register R 1 wird der LAN1-Status gespeichert.

Mode MM4 Modus MM4 mit 9600 Baud, 8 Data-Bit, keine Parität, 1 Stop-Bit, Timeout = 300ms. BCS sei ausgeschaltet, die 5 Register werden definiert und die Diagnose-Flags sind F 1000-1007, die Detail-Diagnose liege in Register 1000.

```
TEXT 50     "UART:9600,8,N,1,300;"
            "MODE:MM4,0,R100,R101,R102,R103,R104;"
            "DIAG:F1000,R1000;"
```

Mode SBUS Parity-Modus, Master-Station mit 9600 Baud, Partner-Stationsnummer in R 555,
Diagnose-Flags F 8000-8007, Diagnose-Register R 4005.

```
TEXT 60     "UART:9600;MODE:SM1,R555;DIAG:F8000,R4005;"
```

Mode SBUS Paritiy-Modus, Slave-Station mit 9600 Baud. Diagnose-Flags F 8000-8007. Diagnose-Register R 4005.

```
TEXT 60     "UART:9600;MODE:SS1;DIAG:F8000,R4005;"
```

Mode SBUS Data-Modus, Master-Station mit 9600 Baud. Register R 55 wird für die Nummer der Partner-Station verwendet. Diagnose-Flags F 8000-8007. Diagnose-Register R 4005.

```
TEXT 60     "UART:9600;MODE:SM2,R55;DIAG:F8000,R4005;"
```

```
Mode        SASI    10          ; Kanal 10
PROFIBUS            T_As_10     ; SASI-Text (wird vom Konfigurator erzeugt)
```

Die Verwendung von Symbolen in SASI-Texten

In SASI-Texten können auch Symbole verwendet werden.

Der Wert und optional auch der Typ des Symbols wird in den Text eingesetzt. Das Symbol wird ausserhalb des ASCII-Textsegments zwischen doppelte Anführungszeichen geschrieben und muss von diesem oder von anderen Symbolen durch ein Komma getrennt werden. Nach dem Symbol, können wahlweise eine optionale Feldbreite und ein Vorzeichen definiert werden.

Format:

symbol [. [[-] [0] width] [t | T]]	
symbol	Symbolname. Es kann (innerhalb der zulässigen Syntax) ein beliebiger Ausdruck gewählt werden, z.B.: MotorOn + 100. Symbole mit Fliesspunktwerten sind nicht erlaubt.
.	Der Punkt nach dem Symbol zeigt an, daß eine Feldbreite und/oder ein Vorzeichen vorgesehen ist.
Width	Die Feldbreite: Anzahl Zeichen oder Leerschäge. Beginnt der Ausdruck mit 0, werden führende Nullen eingesetzt.
t | T	Angabe, ob Elementtyp in Gross- oder Kleinbuchstaben geschrieben wird: 't '→ o, f, r, ...; 'T', → O, F, R,...

Beispiel:

```
BAUD      EQU        9600
D_FLAGS   EQU      F 500
D_REG     EQU      R 4095

          XOB        16
          SASI       1
                     3999

TEXT 3999"UART:",BAUD,",7,E,1;MODE:MC0;"
                  "DIAG:",D_FLAGS.T,",",D_REG.T,";"

          EXOB
```

Dies ergibt den folgenden SASI-Text:

```
          "UART:9600,7,E,1;MODE:MC0;"
          "DIAG:F500,R4095;"
```

Neu: SASI mit '$' (SASI-Text akzeptiert $) siehe nächste Seite
SASI mit $ kann ab den folgenden Firmware-Versionen verwendet werden:

PCD1: V070	PCD4.Mxx5: V0E0	PCD6.M3: V030
PCD2: V080	PCD4.M445: V0E0	

$SASI, $ENDSASI

Um die doch recht komplexen SASI-Texte einfacher zu handhaben, wurden die Assemblerdirektiven $SASI und $ENDSASI eingeführt.

Sind die SASI-Texte zwischen diese Direktiven geschrieben, wird die Syntax (korrekte Schreibweise) dieser SASI-Texte beim Assemblieren überprüft. Bei fehlerhaftem Text wird eine Meldung ausgegeben.

Wurden die SASI-Texte ganz oder teilweise mit Kleinbuchstaben editiert, werden automatisch die obligatorischen Grossbuchstaben erzeugt.

Format:

$SASI
<SASI-Texte>
...
$ENDSASI

Wird $SASI und $ENDSASI nicht geschrieben, besteht die Gefahr, dass die SASI-Befehle nicht richtig ausgeführt werden. Normalerweise wird im Fehlerfall das ERROR-Flag gesetzt.

Beispiel:

```
            XOB    16

            SASI   0     ; Kanal Nr. 0
                   100   ;  Definitionen in Text 100
            SASI   2     ; Kanal Nr. 2
                   101   ;  Definitionen in Text 101

            EXOB

$SASI

   TEXT 100 "uart:9600,7,e,1;mode:mc0;diag:f0,r0;"
   TEXT 101 "UART:4800,8,n,1;MODE:mc2;DIAG:f10,r1;"

$ENDSASI
```

Neu: SASI mit '$' (SASI-Text akzeptiert $)

z.B..: `"UART:$Ra,$Rb,$Rc,$Rd;MODE:$Re,$Rf;DIAG:F$Rg,R$Rh;"`

Ra	Baudrate	110 .. 38400 (numerisch)
Rb	Bits	7, 8 (numerisch)
Rc	Parity	E, O, N (ASCII codiert)
Rd	Stop	1 or 2 (numerisch)
Re	Mode	'MC0', 'SM2' etc. (ASCII codiert)
Rf	Station	Register mit S-Bus Station (numerisch)
Rg	Diagnostic flags	Register mit Basisasdreese der Diagnose-Flags (0 .. 8191 numerisch)
Rh	Diagnostic register	Register mit der Adresse des Diagnose-Registers (0 .. 4095 numerisch)

Passende Firmware: siehe vorangehende Seite (unten).

SASII INDIREKTE ASSIGNIERUNG

(Assign Serial Interface Indirect)

Beschreibung: Initialisiert eine Schnittstelle oder einen PROFIBUS-Kanal im indirekten Modus. Diese Anweisung arbeitet wie der SASI-Befehl. Der Unterschied liegt darin, dass die Assignierung indirekt erfolgt, d.h. dass die Kanalnummer und die Nummer des Definitionstexts in einem Register enthalten sind.

Aufbau:

SASII	**Kanal**	**; Nummer des Kanals oder eines Registers**
	Text	**; Register**

Kanal:
Nummer des zu intitialisierenden Kanals
Dieser Parameter kann direkt oder indirekt angegeben werden:

0..3	Nummer des Kanals
10..99	Nummer des PROFIBUS-Kanals
R 0..4095	Register mit der Kanalnummer (0..3, 10..99)

Text:
Dieser Parameter ist eine Registernummer (R 0..4095)
Dieses Register enthält die Adresse des Textes mit den Definitions-Parametern. Gültige Adressen für einen Text:

0..3999	im Standard-Speicher
4000..7999	im erweiterten Speicher

Beispiel:

```
SASII    0    ; Initialisiert Kanal Nr. 0
         R 1  ; Definitionstext im Register R 1
```

Flags: Das ERROR-Flag wird gesetzt wenn der SASI-Befehl fehlt oder fehlerhaft ist.

Siehe auch: SASI-Texte

SRXD EMPFANG EINES CHARAKTERS

(Serial Receive Character (Mode C))

Beschreibung: Lädt den zuerst empfangenen Charakter aus dem Empfangsbuffer in das PCD-Register, das in der 2. Zeile des Befehls angegeben ist.

Im Register werden nur die 8 niederwertigsten Bit beeinflusst. Die andern werden alle = 0 gesetzt.

Der Befehl darf nur ausgeführt werden, wenn mindestens 1 Charakter im Empfangsbuffer liegt (RBSY = H). Andernfalls wird das ERROR-Flag gesetzt.

Es können bis 512 Charakter in jedem Empfangsbuffer gespeichert sein. Nach jedem SRXD wird der nächste Charakter aus dem Empfangsbuffer ausgelesen.

Überläuft der Empfangsbuffer (mehr als 512 Charakter), wird das RDIA-Flag gesetzt und im Diagnose-Register das Bit Nr. 7: Overflow Error = 1 (H) gesetzt. (Empfangsbuffer Überlauf).

Aufbau:

```
SRXD[X]   Kanal          ; Kanal-Nummer 0-3
          Register (i)   ; PCD-Register R 0-4095
```

Beispiel:

```
SRXD       3    ; Liest den Charakter im Empfangsbuffer des Kanals 3
         R 100  ;   und lädt den Charakter ins Register R 100
```

Flags: Das ERROR-Flag wird gesetzt, wenn der Befehl trotz leerem Empfangsbuffer ausgeführt wird oder wenn die Assignierung nicht oder falsch gemacht wurde.

Siehe auch: STXD, SRXM, Kommunikations-Befehle, Diagnose-Flags

Anwendung: Typische Anwendung in einem nach BLOCTEC strukturierten Anwenderprogramm::

```
STH    F RBSY              ; Ist mindestens 1 Charakter da,
CFB    H READ_CHAR         ;   rufe den FB auf
....

FB       READ_CHAR         ; FB 'Lese Charakter'
[STH   H RDIA]             ; Fehler vorhanden?
[CFB   H RCV_ERROR]        ;   wenn ja, Fehler behandeln
SRXD     0                 ; Lese den Charakter bei Kanal 0
         R 999             ;   und bringe diesen in R 999
....
EFB
```

In einfacheren Anwendungen kann die Behandlung der Diagnose entfallen. (Im Beispiel in Klammern).

STXD SENDEN EINES CHARAKTERS

(Serial Transmit Character (Mode C))

Beschreibung: Der zu sendende Charakter liegt auf den niedrigsten 8 Bit des Registers, dessen Adresse im Operand der 2. Zeile des Befehls steht. Nach der Ausführung des Befehls, wird der Charakter sofort gesendet.

Der Sendebuffer jedes Kanals hat eine Kapazität von 512 Charaktern. Ist der Sendebuffer leer, ist das TBSY-Flag = L. Stehen Charakter zum Senden an, ist das TBSY-Flag = H.

Ist nach der Ausführung von STXD das TDIA-Flag = H, liegt ein Fehler vor. Die Erklärung dazu gibt das Diagnose-Register, welches bei der Assignierung definiert wurde.

Aufbau:

STXD[X]	**Kanal**	**; Kanal-Nummer 0-3**
	Register (i)	**; PCD-Register R 0-4095**

Beispiel:

```
STXD      1     ; Sendet via Kanal 1 den
          R 100 ;   Register 100 (Bit 7 - 0)
```

Flags: Das ERROR-Flag wird gesetzt, wenn die Assignierung nicht oder nicht richtig durchgeführt wurde.

Siehe auch: SRXD, STXT, Kommunikations-Befehle, Diagnose-Flags

Anwendung: Typische Anwendung in einem nach BLOCTEC strukturieren Anwender-programm:

```
....
STL     F TFUL              ; Ist genügend Platz im Sendebuffer,
CFB     H SEND_CHAR         ;   dann rufe den FB auf
....

FB        SEND_CHAR         ; FB zum Senden eines Charakters
STXD      0                 ; Sende den Charakter aus
        R 900               ;   R 900 via Kanal 0
[STH    H TDIA]             ; Fehler vorhanden?
[CFB    H SND_ERROR]        ;   wenn ja, behandeln
....
EFB
```

In einfacheren Anwendungen kann die Behandlung der Diagnose entfallen. (Im Beispiel in Klammern).

STXT SENDEN VON ANWENDER-TEXT

(Serial Transmit Text (Mode C))

Beschreibung: Sendet den in der 2. Zeile des Befehls angegebenen Anwender-Text über die in der 1. Zeile des Befehls angegebene Schnittstelle. Während der Textausgabe ist das XBSY-Flag = H.

Das Ausgeben eines langen Text kann mehrere Sekunden dauern. Das Anwenderprogramm läuft in dieser Zeit normal weiter und wird durch die Textausgabe nicht beeinflusst. STXT delegiert das Senden eines Textes an die Schnittstelle selbst.

Das NEXE-Flag wird gesetzt, wenn der Text falsche Zeichen enthält.

Aufbau:

```
STXT[X]   Kanal          ; Kanal Nummer 0-3
          Text Nr. (i)   ; Text-Nr. 0 - 3999
```

Beispiel:

```
STXT    0       ; Sendet via Kanal 0
        123     ;  den Text 123
```

Flag: Das ERROR-Flag wird gesetzt, wenn der Text nicht existiert oder die Schnittstelle nicht oder nicht richtig assigniert wurde.

Siehe auch: Texte, STXD, SRXD, Kommunikations-Befehle, Diagnose-Flags.

Anwendung: Wird Eingang 1 = H, soll folgender Text über die Schnittstelle 1 ausgegeben werden: "Es lebe die PCD!"

```
        XOB     16
        SASI    1       ; Initialisierung Kanal 1
                99      ;  mit Parametern in Text 99
        EXOB
$SASI
        TEXT 99 "UART:9600,7,E,1;MODE:MC0;DIAG:F1000,R1000;"
$ENDSASI

        COB     0
                0
        STH   I 1       ; Wird Eingang I 1 = H
        DYN   F 0       ;  (Flanke)
        ANL   F 1006    ;  und Schnittstelle frei
        JR    L END     ;  (F 6 = XBSY)
        STXT    1       ;  dann sende via Kanal 1
                10      ;  den Text 10
END:    ECOB

TEXT 10 "Es lebe die PCD!<10><13>"
```


Texte

Texte, welche aus dem Anwenderprogramm mit dem Befehl STXT über eine serielle Schnittstelle an ein Terminal oder einen Drucker ausgegeben werden sollen oder Texte, welche Parameter für die LAN-Kommunikation enthalten, können im Programm an beliebiger Stelle geschrieben werden, d.h. direkt an die Stelle, wo der Text abgerufen wird oder in einer Tabelle im, vor oder nach dem eigentlichen Anwenderprogramm oder in einer eigenen Text-Datei.

Die folgenden Regeln sind zu beachten:

- Ein Text wird definiert mit:
  TEXT n "Dies ist ein Text"
  wobei 'n' ist die Text-Nummer (0...3999 bzw ...7999) ist.
- Texte können mehrere Zeilen lang sein. Jede Zeile beginnt und endet mit einem doppelten Anführungszeichen: "....."
- Steuerzeichen (dezimal ASCII 1-31) werden als dezimale ASCII-Nummer zwischen Pfeilklammern geschrieben, z.B. CR = <13>, LF = <10>, ESC = <27> oder BELL = <7> Alle andere Zeichen (ASCII 32-255) werden im Klartext geschrieben, können aber auch mit der ASCII-Nr. <xxx> notiert werden, z.B. dann, wenn das Zeichen nicht auf der Tastatur sein sollte.
- Das Text-Ende ist nicht speziell zu markieren d.h. ein ASCII NUL wird vom System automatisch eingesetzt.
- Dieses ASCII NUL darf in Texten **nicht** verwendet werden. *)

Text-Beispiele:
Die beiden nachfolgend gezeigten Texte ergeben das gleiche Resultat:

```
TEXT 10    "SAIA PCD, die leistungsfähige SPS"
TEXT 11    "SAIA PCD,"
           "die leistungsfähige SPS"
```

Soll nach einem Text ein "Line Feed" und ein "Carriage Return" folgen, müssen diese Steuerzeichen in Pfeilklammern stehen:

```
TEXT 12    "SAIA PCD, die leistungsfähige SPS<10><13>"
```

Der folgende Text, mit " **SAIA PCD"** in Fett-Druck
Die SAIA PCD kennt keine Grenzen
ist z.B. wie folgt zu schreiben (druckerspezifisch):

```
TEXT 13    "Die<27> E SAIA PCD<27>F kennt keine Grenzen"
```

wobei <27>E Fettdruck einschaltet und <27>F den Drucker wieder auf normal zurückschaltet.

*) Soll in einem Text trotzdem ein ASCII NUL ausgegeben werden, z.B. zur Steuerung des Bildschirms, so muss das Zeichen mit dem Befehl STXD als Einzelcharakter oder mit der Sequenz "$Axxxx" gesendet werden (siehe nächste Seite).

Texte und Variablen (Sonder-Texte)

Texte können Funktions-Charakter zur Ausgabe von PCD-Daten enthalten, um z.B. die Uhrzeit, das Datum oder den Inhalt eines Registers auf ein Druckerprotokoll oder auf den Bildschirm auszugeben.
Zwei Spezial-Charakter leiten die Sonder-Texte ein: $ und @.

$ = Direkte Adressierung,
d.h. die Elementadressen werden absolut angegeben

$H	Zeit (Stunden, Minuten, Sekunden): hh:mm:ss		
$HH	Zeit (nur Stunden):	hh	
$HM	Zeit (nur Minuten):	mm	
$HS	Zeit (nur Sekunden):	ss	
$D	Datum (Jahr, Monat, Tag):	yy-mm-dd	
$d	Datum (Tag, Monat, Jahr):	dd.mm.yy	
$DD	Datum (nur Tag):	dd	
$DM	Datum (nur Monat):	mm	
$DY	Datum (nur Jahr):	yy	
$W	Woche (Wochennummer, Wochentag):	ww-dd	
$WN	Woche (nur Wochennummer):	ww	
$WD	Woche (nur Wochentag):	dd	
$innnn	Logischer Zustand von 1 Eingang (0/1)		nnnn: Element-Adresse (immer 4 Ziffern)
$onnnn	Logischer Zustand von 1 Ausgang (0/1)		
$fnnnn	Logischer Zustand von 1 Flag (0/1)		
$Innnn	Logischer Zustand von 8 Eingängen (nnnn bis nnnn+7)		nnnn: tiefste Element-Adresse (immer 4 Ziffern)
$Onnnn	Logischer Zustand von 8 Ausgängen (nnnn bis nnnn+7)		
$Fnnnn	Logischer Zustand von 8 Flags (nnnn bis nnnn+7)		
$Cnnnn	Counter-Inhalt		nnnn: Element-Adresse (immer 4 Ziffern)
$Rnnnn	Register-Inhalt		
$Tnnnn	Timer-Inhalt		
$Lnnnn	Einfügen eines Untertextes, max. 3 Ebenen (include)		nnnn: Text-Nummer (immer 4 Ziffern)
$xnn	'x' wird nn mal repetiert Der zu repetierende Charakter darf kein Datentyp sein (kein H|h|D|d|W|i|o|f|I|O|F|C|R|T|L|x$|@|%)		nn: immer 2 Ziffern
$Annnn	Inhalt eines Registers im ASCII-Format.		nnnn: Register-Nummer (immer 4 Ziffern)

Beispiel für $Annnn:

"$A0999"	wobei R 999 = 00000000 hex	'NUL'
"$A0999"	wobei R 999 = 00000061 hex	'a'
"$A0999"	wobei R 999 = 00006162 hex	'ab'
"$A0999"	wobei R 999 = 00616263 hex	'abc'
"$A0999"	wobei R 999 = 61626364 hex	'abcd'

Die führenden Nullen werden nicht ausgegeben. Nur wenn das niederwertigste Byte = Null ist, wird ein ASCH NUL ausgegeben.

@ = Indirekte Adressierung, d.h. die Elementadressen werden aus einem spezifizierten Register entnommen.

@innnn	Logischer Zustand von 1 Eingang (0/1)	nnnn: Register-Nummer immer 4 Ziffern
@onnnn	Logischer Zustand von 1 Ausgang (0/1)	
@fnnnn	Logischer Zustand von 1 Flag (0/1)	
@Innnn	Logischer Zustand von 8 Eingängen (0/1) (Adresse bis Adresse +7)	
@Onnnn	Logischer Zustand von 8 Ausgängen (0/1) (Adresse bis Adresse +7)	
@Fnnnn	Logischer Zustand von 8 Flags (0/1) (Adresse bis Adresse +7)	
@Cnnnn	Counter-Inhalt	
@Rnnnn	Register-Inhalt	
@Lnnnn	Einfügen eines Untertextes, max. 3 Ebenen	
@xnnnn	'x' wird so viele Male repetiert, wie der aktuelle Registerinhalt lautet. Der zu repetierende Charakter darf kein Datentyp sein (kein H|h|D|d|W|i|o|f|I|O|F|C|R|T|L|x$|@|%)	

Anmerkung: Soll in einem Text ein "$" oder ein "@" angegeben werden, ist zu schreiben "$$"oder "@@".

Beispiel 1: TEXT 10 "Datum: $D Zeit: $H<CR><LF>"
"$-34<CR><LF>"
"Eingänge 0..7: $I0000 <CR><LF>"
"Register 100: $R0100 <CR><LF>"
"$+20 <CR><LF>"

Angenommen dieser Text werde von einer PCD am 27. Februar 1992 um 20.13 Uhr ausgegeben. Die Eingänge 0 und 1 sind = H und im Register 100 steht 12345, dann wird folgendes ausgegeben:

```
Datum: 92-02-27   Zeit: 20:13:00
---------------------------------
Input 0..7: 11000000
Register 100: 12345
++++++++++++++++++++
```

Beispiel 2: Praktisches Beispiel für "$A...."
Die Cursor-Positionierung bei einem Bildschirmterminal soll für die X- und die Y-Position aus 2 Registern erfolgen:

X-Position aus Register R 1 (1..80)
Y-Position aus Register R 2 (1..25)

Die "Escape"-Sequenz für die Cursorpositionierung ist
<27><17><Wert für X><Wert für Y>
Es kann programmiert werden: "...<27><17>$A0001$A0002..."

Soll z.B. eine feste Position von X = 40 und Y = 12 ausgegeben werden, so kann die ganze Sequenz von 4 Charaktern in ein einziges Register geladen und mit $A.... ausgegeben werden. Es ist zu beachten, dass alle Werte im Hex-Format stehen müssen:
ESC = 1B hex; 17 = 11 hex; 40 = 28 hex (X-Wert); 12 = 0C hex (Y-Wert)
Laden des Registers R 1000:

```
LD R 1000
   1B11280CH
```

Textausgabe für Cursorpositionierung: ". . . $A1000 . . ."

AUSGABE-FORMATE FÜR REGISTER UND COUNTER

Das Ausgabe-Format von Registern und Countern kann im Text definiert werden. Es werden die Anzahl Zeichen und die Stelle des Dezimalpunkts angegeben. Die Format-Definition wird in der Form "$%xxxx" gemacht, wobei xxxx das Format angibt. Ab einer Format-Definition gilt diese für alle nachfolgenden Register- und Counterwert-Ausgaben bis entweder das Format rückgängig gemacht oder ein anderes Format definiert wird.

In den Format-Definitionen steht 'd' oder 'D' für dezimal, (auch 'x' oder 'X' für hexadezimal und 'b' oder 'B' für binär sind zulässig). Andere Formate werden nicht unterstützt.

Ist der Wert grösser als das definierte Format, wird das Standardformat ausgegeben. (ohne Dezimalpunkt).

AUSGABEFORMAT DEFINITIONEN:

Die Möglichkeiten der Formatwahl wird an einigen Beispielen erläutert. Es werden für diese Beispiele die Register 10, 11 und 12 mit einem festen Wert geladen und danach in den verschiedenen Formaten ausgegeben.

R 10 = 123456 R 11 = -7890 R 12 = 5

OHNE FORMATIERUNG (Standard Format):

TEXT 0 "REGISTER 10: $R0010 <10><13>"
"REGISTER 11: $R0011 <10><13>"
"REGISTER 12: $R0012"

Ausgabe:

```
REGISTER 10: 123456
REGISTER 11: -7890
REGISTER 12: 5
```

Kommentar: Normalformat, ohne Leerstellen.

DEFINITION DER FELDLÄNGE:

Mit einer Sequenz <$%xxd>, welche im Text vor dem <$Rnnnn> steht, kann die Feldlänge (xx = 1-99) der Ausgabe definiert werden. <d> steht für "dezimal" und Leerstellen vor der Zahl. <D> steht auch für "dezimal" jedoch für Nullen vor der Zahl.

TEXT 1 "$%08d"
"REGISTER 10: $R0010 <10><13>"
"REGISTER 11: $R0011 <10><13>"
"REGISTER 12: $R0012"

Ausgabe:

```
REGISTER 10:• • 1 2 3 4 5 6
REGISTER 11:• • • - 7 8 9 0
REGISTER 12:• • • • • • • 5    • = Leerstelle
```

Kommentar: Hat die auszugebende Zahl mehr Stellen als definiert wurde, kommt das Normalformat zur Anwendung.

Rev. 01 - 10.99

```
TEXT 1      "$%08D"
            "REGISTER 10: $R0010 <10><13>"
            "REGISTER 11: $R0011 <10><13>"
            "REGISTER 12: $R0012"
Ausgabe:
            REGISTER 10: 0 0 1 2 3 4 5 6
            REGISTER 11: - 0 0 0 7 8 9 0
            REGISTER 12: 0 0 0 0 0 0 0 5
```

DEFINITION DER FELDLÄNGE UND DER STELLE DES DEZIMALPUNKTES:
Mit einer Sequenz <$%xx.yd> wird mit <xx> die totale Feldlänge der Ausgabe, mit <y> (0 - 9) die Anzahl Stellen nach dem Dezimalpunkt definiert.

```
TEXT 2      "$%07.3d"
            "REGISTER 10: $R0010 <10><13>"
            "REGISTER 11: $R0011"<10><13>"
            "REGISTER 12: $R0012"
Ausgabe:
            REGISTER 10: • 1 2 3 . 4 5 6
            REGISTER 11: • • - 7 . 8 9 0
            REGISTER 12: • • • 0 . 0 0 5   • = Leerstelle
```

DEFINITION DES DEZIMALPUNKTES ALLEIN:
Mit der Sequenz <$%00.yd> wird nur die Stelle des Dezimalpunktes definiert. Am Anfang der Zahl sind keine Leerstellen.

```
TEXT 3      "$%00.5d"
            "REGISTER 10: $R0010 <10><13>"
            "REGISTER 11: $R0011"<10><13>"
            "REGISTER 12: $R0012"
Ausgabe:
            REGISTER 10:   1 . 2 3 4 5 6
            REGISTER 11: - 0 . 0 0 7 8 9
            REGISTER 12:   0 . 0 0 0 0 5
```

RÜCKKEHR ZUM NORMALFORMAT:
Mit der Sequenz <$%00d> kann in einem Text nach einer Formatänderung wieder in das Normalformat zurückgekehrt werden.

Das in einem Text gewählte Format ist auch in den nachfolgenden Textsubroutinen wirksam. Wird jedoch das Format in einer Textsubroutine geändert, so wird bei der Rückkehr in die dem Haupttext näheren Ebene das ursprüngliche Format dieser Ebene wieder wirksam.

Sind in einem Programm mehrere Ausgabeformate vorgesehen, so können bis 10 verschiedene Formate pro Schnittstelle vordefiniert und dann an den entsprechenden Stellen auf einfachere Weise aufgerufen werden.

Definieren und anwenden von Formatvorlagen
Die Sequenz für das Vordefinieren eines Formates ist <$sn>, diejenige für das Aufrufen <$n>. Darin bedeutet <s> "store" (speichern) und <n> eine Ziffer (0 - 9), welche die Vorwahl definiert. Diese Vorwahlen werden am einfachsten im XOB 16 vorgenommen.

Beispiel:

```
          XOB       16
          ....
;
TEXT 991 "$%05.1d$s1"       ; Format 1 Definition (nnn.n)
TEXT 992 "$%04.2d$s2"       ; Format 2 Definition (n.nn)
TEXT 993 "$%08.3d$s3"       ; Format 3 Definition (nnnn.nnn)
;
          SEI     K 0       ; Aktivierung der Format-Definition
DEF:      STH       XBSY
          JR      H DEF
          STXTX     0
                    991
          INI     K 2
          JR      H DEF
          ....
          EXOB
;
          COB       0
                    0
          ....
          STXT      1
                    10
;
TEXT 10 "Pumpe Liter     Preis/l    Total <10><13>"
        "  1   $1$R0010  $2$R0011   $3$R0012 <10><13>"
        "  2   $1$R0013  $2$R0014   $3$R0015 <10><13>"
;
          ....
          ECOB
```

Dies ergibt das Gleiche wie die folgende Text-Formatierung:

```
"1 $%05.1d$R0010 $%04.2d$R0011 $%8.3d$R0012"
"2 $%05.1d$R0013 $%04.2d$R0014 $%8.3d$R0015"
```

Ausgabe:

```
          Pumpe  Liter    Preis/l    Total
            1     13.8     0.86      11.868
            2    158.2     0.95     150.290
```

TEXT-UNTERPROGRAMME:

Mit "$Lnnnn" wird in einen Text ein "Untertext" eingebettet. Ein solcher Untertext wird wie ein normaler Text verarbeitet.

Untertexte können in bis zu 3 Ebenen verschachtelt werden. (4 Ebenen mit dem Haupttext). Am Ende eines Untertextes geht der Text selbständig in die obere Ebene zurück. Das Ende des Untertextes muss nicht markiert werden.

Beispiel:

```
         COB       0
                   0
         ....
         STXT      1    ; Ausgabe Text 10
                   10
TEXT 10  "$L0100 Motor zu schnell<10><13>"
         ....
         STXT      1    ; Ausgabe Text 11
                   11
TEXT 11  "$L0100 Oeldruck zu niedrig<10><13>"
         ...
         ECOB
TEXT 100 "ALARM beim Dieselaggregat:"
```

Ausgabe:

ALARM beim Dieselaggregat: Motor zu schnell
ALARM beim Dieselaggregat: Oeldruck zu niedrig

Die Verwendung von SYMBOLEN in Texten

Ab der Utility-Version Vl.3 können Symbole auch in Texten verwendet werden. Die Symbole müssen aber in der gleichen Datei definiert werden. Ab der Version V1.5 können Symbole in Texten auch in einer andern Datei definiert sein, müssen aber dort, wie normale Symbole als "PUBL" und in der Datei, wo die Symbole verwendet werden, als "EXTN" deklariert werden.
Das Symbol ist ausserhalb des PCD-Textes, zwischen Kommas zu schreiben. Nach dem Symbol folgt, durch einen Punkt getrennt, das Format im Sondertext (t oder T). (siehe auch Beispiele).

Format:

Symbol [. [[-] [0] Länge] [t | T]]	
Symbol	Name des Symbols. Jedes Symbol, auch mit mathematischem Ausdruck, ist erlaubt, z.B.: Motor Ein+100, ... (FP-Format geht nicht)
. (Punkt)	Ein Punkt zeigt an, dass eine Format-Definition folgt
Länge	Anzahl Ziffern der Elementnummer. Beginnt die 'Länge' mit 0, werden führende Nullen geschrieben.
t | T	Angabe, ob Elementtyp in Gross- oder Kleinbuchstaben geschrieben wird: t --> i, o, f-, T --> I, 0, F

Beispiele:

```
Flag      EQU     F 123
Output    EQU     O 32
Reg       EQU     R 999

TEXT 0    "$",Flag.04T          ; ergibt "$F0123"
TEXT 1    "",Flag               ; ergibt"123" ("" ist notwendig)
TEXT 2    "DIAG:",Ausgang.T,",",Reg.T
                                ; ergibt "DIAG:O32,R999"
TEXT 3    "55:",Flag.T,"-",Flag+7,":",Ausgang.T,"-",Ausgang+7
                  ; ergibt den LAN2-Text "55:F123-130:O32-39"
TEXT 4    "FLAG Nummer: *",Flag.-8,"*"; -x:Anzahl Stellen
                                ; ergibt "FLAG Number: *123     *"
```

Symbole in SASI-Texten

```
D_FLAGS   EQU     F 500
D_REG     EQU     R 4095

          XOB       16
          SASI      1
                    3999
TEXT 3999"UART:9600,7,E,1;MODE:MC0;"
                  "DIAG:",D_FLAG.T,",",D_REG.T,";"
```

Dies ergibt den SASI-Text: ..."DIAG:F500,R4095;"

Anmerkung: Symbole und Texte müssen in der gleichen Datei definiert sein.

SRXM EMPFANG VON MEDIEN IM MODUS D

(Serial Receive Media (Mode D))

Beschreibung: Liest von der Partner-Station die Quellelemente und lädt diese in der eigenen PCD auf die Zielelemente. Es können I, 0, F auf 0, F und T, C, R auf T, C, R übertragen werden.
In der 1. Zeile wird die Kanal-Nummer, in der 2. Zeile die Anzahl Elemente, welche zu lesen sind, angegeben. In der 3. Zeile wird die niedrigste Adresse der zu lesenden Elemente in der Partnerstation und in der 4. Zeile die niedrigste Elementadresse in der eigenen Station angegeben.
Während der Telegrammübertragung ist das TBSY-Flag = H.

Aufbau:

```
SRXM[X]   Kanal             ; Kanal-Nummer 0 - 3
          Anzahl Elemente   ; Anzahl Elemente 1 - 16
          Quelle    (i)     ; I, O, F, R, T, C
          Ziel      (i)     ; O, F, R, T, C
```

Beispiel:

```
SRXM      0      ; Überträgt via Kanal 0
          16     ;  16 Elemente
        R 100    ;  ab Partner-Station R 100 - R 115
        R 0      ;  zu eigener Station R 0 - R 15
```

Flags: Das ERROR-Flag wird gesetzt, wenn trotz laufender Kommunikation ein SRXM versucht wird oder der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: STXM, Kommunikations-Handbuch, Diagnose-Flags

Anwendung: Die Eingänge I 0-15 der Partner-Station sollen auf die Ausgänge 32-47 der eigenen PCD abgebildet werden. (Die beiden Stationen sind über die seriellen Schnittstellen 2 bzw. 1 verbunden).

Eigene PCD:

```
     XOB       16
     SASI      2
               0

TEXT 0   "UART:9600,7,E,1;MODE:MD0;"
         "DIAG:F1000,R1000;"

     EXOB

     COB       0
               0
     STH     F 1003   ; TBSY
     JR      H next
     SRXM      2      ; Kanal 1
               16     ; 16 Elemente
             I 0      ; ab I 0 (bis 15) Partner-Station
             O 32     ; zu O 32 (bis 47) Eigene-St.
next:          ECOB
```

Partner-PCD:
(Es muss nur die Schnittstelle assigniert werden)

```
     XOB       16
     SASI      1
               0

TEXT 0
"UART:9600,7,E,1;MODE:SD0;"
"DIAG:F1000,R1000;"

     EXOB
```

SRXM EMPFANG VON REGISTERN IM MODUS MM4
(Serial Receive Media (Mode MM4))

Beschreibung: Es können nur Register übertragen werden. Jedes Register enthält 4 ASCII-Charakter. Der Daten-Transfer erfolgt über das LAC/LAC2 Netzwerk mittels des MM4-Protokolls. Eine direkte Punkt zu Punkt-Übertragung kann auch realisiert werden.
In der 1. Zeile wird die Kanal-Nummer angegeben. In die 2. Zeile wird 0 (Null) geschrieben (nicht verwendet). In der 3. Zeile wird die Adresse eines Registers angegeben. In dieses Register wird eingeschrieben, wieviele ASCII-Charakter empfangen wurden.
Der Empfangsbuffer wird in aufeinanderfolgende PCD-Register geladen. Die niedrigste Adresse dieser Register wird in der 4. Zeile des Befehls SRXM angegeben.
Die Charakter werden wie folgt in die Register eingeschrieben:

Reg 1:	11111111	22222222	33333333	44444444	Charakter 1 - 4
Reg 2:	55555555	66666666	77777777	88888888	Charakter 5 - 8

...
Ist die Anzahl der empfangenen Charakter nicht ein Vielfaches von 4, wird der Rest des Registers Null gesetzt.
Die Adresse des Partners, welcher das Telegramm gesendet hat, steht Register <repartner>, welches im SASI-Text definiert wurde.

Aufbau:

```
SRXM    Kanal       ; Kanal-Nummer 0-3
        0           ; nicht verwendet
        Reg. 1      ; Anzahl empfangener Char. R 0-4095
        Reg. 2      ; Basisadresse der Empfangsreg. R 0-4095
```

wobei:

Reg 1 Nach dem Empfang des Telegramms steht in diesem Register oder Counter die Anzahl empfangener Charakter.

Reg 2 Niedrigste Registeradresse einer Reihe von Regstern, in welche die empfangenen ASCII-Charakter abgelegt werden.

Beispiel:

```
SRXM    1       ; Nummer der Schnittstelle
        0       ;  nicht verwendet
      R 100     ;  Anzahl empfangener Charakter
      R 20      ;  Niedrigste Reg.adresse der empfangenen Information
```

Flags: Das ERROR-Flag wird gesetzt, wenn trotz laufender Kommunikation ein SRXM versucht wird oder die Schnittstelle nicht richtig oder gar nicht assigniert ist.

Für mehr Information ist die Beschreibung des "LAC-MM4-Protokolls" zu konsultieren

SRXM EMPFANG VON MEDIEN IM MODUS S-BUS

(Serial Receive Media (Mode SBUS))

Beschreibung: Liest von einer Slave-Station die Quellelemente und lädt diese in der eigenen PCD (Master) auf die Zielelemente. Es können I, 0, F auf 0, F und T, C, R auf T, C, R übertragen werden. Auch die PCD-interne Uhr, der Status der einzelnen CPU und das Display-Register können übertragen werden. Die Adresse der Slave-Station wird dem Register entnommen, das im SASI-Befehl als <dest-reg> angegeben wurde.
In der 1. Zeile wird die Kanal-Nummer, in der 2. Zeile die Anzahl Elemente, welche zu übertragen sind, angegeben. In der 3. Zeile wird die niedrigste Adresse der zu lesenden Elemente in der Slave-Station und in der 4. Zeile die niedrigste Elementadresse in der eigenen Station angegeben.
Während der Telegrammübertragung ist das TBSY-Flag = H.

SRXM kann nur von der Master-Station ausgeführt werden

Aufbau:

```
SRXM[X]   Kanal              ; Kanal-Nummer 0-3
          Anzahl Elemente    ; Anzahl Elemente                1)
          Quelle    (i)      ; Basisadr. der Quell-Elemente   2)
          Ziel      (i)      ; Basisadr. der Ziel-Elemente    3)
```

1) Die max. Anzahl zu lesender Elemente, ist vom Typ der El. abhängig:
   - 1 - 31 — Für Register, Timer, Counter.
   - 1 - 128 — Für Eingänge, Ausgänge, Flags.
   - 0 — Spezialcode für Uhr oder Status der Slave-PCD

2) Quelle
   - I/0/F 0-8191, R 0-4095, T/C 0-1599 — Basisadresse der Elemente in der Slave-PCD
   - K 1000 — Spezialcode um die Uhr zu lesen
   - K 2000 — Spezialcode um das Display-Register zu lesen
   - K 0-7 — Spezialcode um den CPU-Status zu empfangen
   - 0-6 — CPU-Nummern der Slave-PCD
   - 7 — Status der eigenen CPU

3) Ziel
   - I/0/F 0-8191, R 0-4095 — Basisadresse der Elemente in der Master-PCD

Beispiel:

```
LD      R 100  ; <destreg> gemäss SASI-Befehl
          10   ;  Nummer der Slave-Station
SRXM      0    ; Übertragen via Kanal 0
          20   ;  20 Elemente
        R 100  :  aus Slave-Station Nr. 10, R100 - 119
        R 0    ;  zu eigener Station R0 - R19
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn SRXM während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: STXM, Kommunikations-Befehle, Diagnose-Flags

Für mehr Information ist das "S-Bus Handbuch" zu konsultieren

Die folgende Tabelle zeigt, welche Elemente von einer Quell-Station aus auf welche Elemente in einer Partner-Station kopiert werden können.

		Master-PCD (Ziel)					
		O	**F**	**R**	**C**	**T**	**DB**
Slave-PCD (Quelle)	**I**	•	•				
	O	•	•				
	F	•	•				
	R			•	•	•	•
	C			•	•	•	•
	T			•	•	•	•
	K			•			
	DB			•	•	•	

Spezial-Funktionen

Code	**Funktions-Beschreibung**	**Resultat-Beispiele**
K 0 ..7	**Lesen des CPU-Status**: 0..6: CPU-Nr. der Slave-PCD 7: eigener CPU-Status	R, C, H, S, D
K 1000	**Lesen der Uhr**	gleiches Format wie RTIME-Bef.
K 2000	**Lesen des Display-Registers**	
K 3000	**Lesen der Grösse eines Data-Blocks**	
K 5000	**Lesen des PCD-Typs** in ASCII	" D1" , " D2", " D4", …
K 5010	in Dezimal	1, 2, 4, …
K 5100	**Lesen des Modultyps** in ASCII	" M1_", " M11", " M12", " M14",..
K 5110	in dezimal	10, 11, 12, 14, 24, …
K 5200	**Lesen Firmwarevers.** in ASCII	" $4C", " 004", " X41", …
K 5210	in dez	5, .. oder -1 dez für alle '$', 'X', 'β'
K 5300	**Lesen der CPU-Nr.** in ASCII	" 0", " 1"
K 5310	in dezimal	0 .. 6
K 6000	**Lesen der S-Bus Stations-Nummer** in BROADCAST Dies funktioniert nur in einer Punkt-zu-Punkt-Kommunikation	

Übertragung von Daten-Blocks (DB)

Wird mit Daten-Blocks gearbeitet, weicht das Format von SRXM etwas vom Standardformat ab. Zur Adressierung eines Elementes in einem DB muss immer die Nummer des DB **und** die Position des Elementes im DB angegeben werden.

SRXM **Kanal**
Anzahl + Position
Quelle
Ziel

wobei Anzahl + Position → Register-Nummer.
Dieses Register enthält die Anzahl zu übertragender Elemente (1 .. 32) und die Position innerhalb des DB. "Anzahl" liegt im oberen Datenwort, "Position" im unteren Datenwort des Registers.

SRXM EMPFANG VON MEDIEN IM MODUS PROFIBUS

(Serial Receive Media (PROFIBUS))

Beschreibung: Liest Daten (Objekte) von der Partner-Station (remote) und kopiert diese in die eigene (lokale) Station.
Der 1. Operand ist die Kanalnummer.
Der 2. Operand ist der Subindex des Quell- und Zielobjekts.
Der 3. Operand ist der Quell-Objektindex der Partner-Station.
Der 4. Operand der Ziel-Objektindex der eigenen Station.

Aufbau:

```
SRXM    Kanal     ; Kanal-Nummer 10-99
        Anzahl    ; Sub-Index 0-255
        Quelle    ; Quell-Objektindex K 0-16383
        Ziel      ; Ziel-Objektindex K 100-499
```

Wobei::

Kanal	10..99	für PCD4.M445
	10..19	für PCD2 mit PCD7.F700
Anzahl	0..255	Subindex (0 = kein Subindex)
Quelle		Quell-Objektindex (Remote-Station)
	K 0..16383	
Ziel		Ziel-Objektindex (eigene Station)
	K 100..499	PCD4.M445
	K 100..199	PCD2 mit PCD7.F700

Beispiel:

```
SRXM    13      ; Liest via Kanal 13
        0       ;  ohne Subindex
      K 22      :  kopiert Objekt 22 von der Remote-Station
      K 150     ;  zum Objekt 150 der eigenen Station
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn SRXM während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: STXM

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

SRXMI INDIREKTER EMPFANG IM MODUS S-BUS
(Serial Receive Media Indirect (Mode SBUS))

Beschreibung: Dieser Befehl arbeitet gleich wie der SRXM-Befehl. Der Unterschied liegt im indirekten Modus. Indirekt heisst, dass die Anzahl Medien für Quelle und Ziel mittels eines Registerinhalts angegeben werden. SRXMI kann nur für die Übertragung von Medien verwendet werden. Das Übertragen der Uhr, des Display-Register usw. ist nicht möglich.

Aufbau:

```
SRXMI   Kanal      ; Kanal
        Anzahl     ; Anzahl oder Anzahl + Position
        Quelle     ; Quellen-Typ und Registernummer
        Ziel       ; Ziel-Typ und Registernummer
```

Kanal:
Parameter für die Kanalnummer (Bereich 0 .. 3)

Anzahl oder Anzahl + Position:
Dieser Parameter ist eine Registernummer. Dieses Register enthält die "Anzahl" der Standard-Medien und die "Position" im Daten-Block. Für DB: "Anzahl" liegt im oberen Datenwort, "Position" im unteren Datenwort des Registers. In diesem Fall kann die Initialisierung dieses Registers einfach mit den LDL- und LDH-Befehlen ausgeführt werden.

Quellen-Typ und Registernummer:
Ziel-Typ und Registernummer:
Diese Parameter spezifizieren die Quelle und das Ziel einer Datenübertragung. Jeder dieser Parameter besteht aus einem Charakter mit dem Media-Typ (I/O/F/R/T/C/DB) und einer Registernummer 0 - 4095). Die Regeln der Quelle-Ziel-Datenüberragung der Befehle SRXM/STXM sind einzuhalten.

SRXMI kann weder indexiert noch parameteriert werden

Beispiel:

```
SRXMI     3     ; Kanal 3
        R 100   ;   Anzahl zu übertragende Elemente in R 100
        O 101   ;   Quelle: Ausgänge mit der Basisadresse in R 101
        F 102   ;   Ziel: Flags mit der Basisadresse in R 102
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn SRXMI während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: SRXM, Kommunikations-Befehle, Diagnose

Für weitergehende Informationen ist das "S-Bus Handbuch" (Best. Nr. 26/739) zu konsultieren.

SRXMI INDIREKTER EMPFANG IM MODUS PROFIBUS

(Serial Receive Media Indirect (PROFIBUS))

Beschreibung: Liest Daten (Objekte) im indirekten Modus von der Remote-Station und kopiert diese in die lokale PCD. Der Kanal kann sowohl direkt als auch indirekt adressiert werden.

Aufbau:

```
SRXM    Kanal     ; Kanalnummer 10-99, R 0-4095
        Anzahl    ; Subindex R 0-4095
        Quelle    ; Quell-Objektindex K [R 0-4095]
        Ziel      ; Ziel-Objektindex K [R 0-4095]
```

Wobei::

Kanal:	10..99	für PCD4.M445	
	10..19	für PCD2 mit PCD7.F700	
	R 0..4095	für indirekte Adressierung	
Anzahl:	R 0..4095	Register, welches den Subindex enthält (0 = ohne Subindex)	
Quelle:	Register mit Quell-Objektindex (Remote-Station)		
	K [R 0..4095]	0..16383	
Ziel:	Register mit Ziel-Objektindex		
	K [R 0..4095]	100..499	PCD4.M445
		100..199	PCD2 mit PCD7.F700

SRXMI kann weder indexiert noch parameteriert werden

Beispiel:

```
SRXMI   R 50   ; Kanalnummer in R 50
        R 51   ;  Subindex in R 51
        K 52   :  Quell-Objektindex in R 52
        K 53   ;  Ziel-Objektindex in R 53
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn SRXMI während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: SRXM

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

STXM SENDEN VON MEDIEN IM MODUS D

(Serial Transmit Media (Mode D))

Beschreibung: Sendet zur Partner-Station die Quellelemente der eigenen PCD. Es können I, O, F auf O, F und T, C, R auf T, C, R geladen werden.
In der 1. Zeile wird die Kanal-Nummer, in der 2. Zeile die Anzahl Elemente, welche zu senden sind, angegeben. In der 3. Zeile wird die niedrigste Adresse der zu sendenden Elemente aus der eigenen Station und in der 4. Zeile die niedrigste Elementadresse in der Partner-Station angegeben.
Während der Telegrammübertragung ist das TBSY-Flag = H.

Aufbau:

```
STXM[X]   Kanal              ; Kanal-Nr. 0-3
          Anzahl Elemente    ; Anzahl Elemente 1 - 16
          Quelle    (i)      ; I, O, F, R, T, C
          Ziel      (i)      ; O, F, R, T, C
```

Beispiel:

```
STXM       0      ; Überträgt via Kanal 0
           8      ;  8 Elemente
         R 50     ;  ab eigener Station R 50 - 57
         R 10     ;  zur Partner-Station R 10 - 17
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXM während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: SRXM, Kommunikations-Befehle, Diagnose-Flags

Anwendung: Die Eingänge I 0-15 der eigenen Station sollen auf die Ausgänge 32-47 der Partner-PCD abgebildet werden.
In der Partner-PCD ist als Minimum die Assignierung durchzuführen.

Eigene PCD:

```
          XOB       16
          SASI      1      ; Schnittstelle Nr. 1
                    15     ; Assignierungs-Text Nr. 15
$SASI
TEXT 15   "UART:9600,8,E,1;MODE:MD0;DIAG:F1000,R1000;"
$ENDSASI
          EXOB

          COB       0
                    0
          STH     F 1003 ; TBSY
          JR      H NEXT
          STXM      1    ; Kanal 1
                    16   ;  16 Elemente
                  I 0    ;  ab Eingang 0 (-15)
                  O 32   ;  zu Ausgang 32 (-47)
NEXT:     ECOB
```

STXM SENDEN VON REGISTERN IM MODUS MM4

(Serial Transmit Media (Mode MM4))

Beschreibung: Es können nur Register übertragen werden. Jedes Register enthält 4 ASCII-Charakter.
Der Daten-Transfer erfolgt über das LAC/LAC2 Netzwerk mittels des MM4-Protokolls. Eine direkte Punkt zu Punkt-Übertragung kann auch realisiert werden. Es wird die in PCD-Registern bereitgestellte Information (4 ASCII-Charakter/Register) gesendet.
In der 1. Zeile wird die Kanal-Nummer angegeben.
In der 2. Zeile wird die Sende-Funktion definiert.
In der 3. Zeile des Befehls wird die Anzahl zu sendender Charakter angegeben und in der 4. Zeile des Befehls wird die Adresse des niedrigsten Registers für die Ziel-PCD mitgeteilt.

Die Charakter werden wie folgt in die Register eingeschrieben:

Reg 1:	11111111	22222222	33333333	44444444	Charakter 1 - 4
Reg 2:	55555555	66666666	77777777	88888888	Charakter 5 - 8

...

Die Adresse der Partner-Station steht in einem Register, welches bei der Assignierung definiert wurde.

Nach der Ausführung von STXM wird das XBSY-Flag = H. Nach erfolgreicher Operation (Acknowledge des Partners) wird das XBSY-Flag = L.

Aufbau:

```
STXM    Kanal       ; Kanal-Nummer 0 - 3
        Fkt         ; Auszuführende Funktion 0-4
        R/C 1       ; Anzahl Charakter in R 0-4095
        R   2       ; Tiefste Register-Adresse der Zielstation
```

wobei:

Fkt auszuführende Funktion
- 0 /2 Übertragen von Daten
- 4 Senden an alle Stationen im Netz

Beispiel:

```
STXM    1       ; Nummer der Schnittstelle
        0       ;   senden von Daten
      C 100     ;   Anzahl zu übertragende Charakter
      R 20      ;   Niedrigste Registeradresse der empfangenen
                ;   Information in der Partner-Station
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXM während einer laufenden Kommunikation ausgeführt wird.

Für mehr Information ist die Beschreibung des "LAC-MM4-Protokolls" zu konsultieren

STXM SENDEN VON MEDIEN IM MODUS S-BUS

(Serial Transmit Media (Mode SBUS))

Beschreibung: Überträgt Elemente von der eigenen (Master-) Station zu Elementen in der Partner-Station. Die Adresse der Partner-Station liegt in einem Register, welches im SASI-Text definiert wurde. Es können I, O, F auf O, F oder R, T, C auf R, T, C übertragen werden.
In der 1. Zeile wird die Kanalnummer, in der 2. Zeile die Anzahl Elemente, welche zu übertragen sind, angegeben. In der 3. Zeile wird die niedrigste Adresse der Quell-Elemente der eigenen PCD und in der 4. Zeile die niedrigste Adresse der Ziel-Elemente der Partner-Station angegeben.
Während der Ausführung des STXM-Befehls ist das TBSY-Flag = H. Nach erfolgreicher Beendigung der Operation wird das TBSY-Flag wieder = L.

STXM kann nur in der eigenen (Master) PCD ausgeführt werden.

Aufbau:

```
STXM[X]   Kanal              ; Kanal-Nummer 0-3
          Anzahl Elemente    ; Anzahl zu übertragende Elemente
          Quelle    (i)      ; Basisadresse der Quell-Elemente
          Ziel      (i)      ; Basisadresse der Ziel-Elemente
```

Wobei:

Kanal	0..3	zu verwendende Schnittstelle	
Anzahl Elem.	1..32	Anzahl zu sendende R, T, C	
	1..128	Anzahl zu sendende I, O, F	
	0	Spezial-Funktionscode	
Quelle	I/O/F	0..8191	
	R	0..4095	Basisadresse der Elemente
	T/C	0..1599	in der Master-PCD
	DB	0..7999	
	K	4000	Spezial-Funktion
Ziel	I/O/F	0..8191	
	R	0..4095	Basisadresse der Elemente
	T/C	0..1599	in der Slave-PCD
	DB	0..7999	
	K	1000	Schreiben der Uhr in der Slave-PCD
	K	17, 18, 19	Spezial-Funktionen

Beispiel:

```
LD      R 100  ; Im SASI-Text definiertes Register
          22   ;   für die Adresse der Partner-Station
STXM      0    ; Übertragung via Kanal 0
        100    ;   100 Elemente
      F 100    ;   Flags  F 100 - 199
      O 32     ;   zu Ausgängen O 32 - 131 der Station 22
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXM während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: SRXM, Kommunikations-Befehle, Diagnose-Flags.

Die folgende Tabelle zeigt, welche Elemente von einer Quell-Station aus auf welche Elemente in einer Partner-Station kopiert werden können.

		Slave-PCD (Ziel)						
		O	**F**	**R**	**C**	**T**	**DB**	**Clock**
Master-PCD (Quelle)	**I**	•	•					
	O	•	•					
	F	•	•					
	R			•	•	•	•	•
	C			•	•	•	•	
	T			•	•	•	•	
	DB			•	•	•		

Beim Schreiben der Datumuhr werden 2 Register gesendet. Das Format entspricht demjenigen des Befehls WTIME.

Spezial-Funktion

Es kann in der Slave-Station mittels des Befehls STXM ein XOB aufgerufen werden. Es sind dazu die folgenden Angaben zu machen:

```
STXM      0..3          ; Kanal-Nummer
          0             ; (muss 0 sein)
        K 4000          ; Code zum Aufruf eines XOB
        K 17|18|19      ; aufzurufender XOB
```

Diese Möglichkeit kann auch als "Broadcast"-Befehl, z.B. zum Synchronisieren von Ereignissen herangezogen werden.

Übertragung von Daten-Blocks (DB)

Wird mit Daten-Blocks gearbeitet, weicht das Format von STXM etwas vom Standardformat ab. Zur Adressierung eines Elementes in einem DB muss immer die Nummer des DB **und** die Position des Elementes im DB angegeben werden.

STXM **Kanal**
Anzahl + Position
Quelle
Ziel

wobei Anzahl + Position → Register-Nummer.

Dieses Register enthält die Anzahl zu übertragender Elemente (1 .. 32) und die Position innerhalb des DB. "Anzahl" liegt im oberen Datenwort, "Position" im unteren Datenwort des Registers.

Für mehr Information ist das "S-Bus Handbuch" zu konsultieren

STXM SENDEN VON MEDIEN IM MODUS PROFIBUS

(Serial Transmit Media (PROFIBUS))

Beschreibung: Sendet Daten (Objekte) von der eigenen (lokalen) Station zur Partner-Station (remote).
Der 1. Operand ist die Kanalnummer.
Der 2. Operand ist der Subindex des Quell- und Zielobjekts.
Der 3. Operand ist der Quell-Objektindex der eigenen Station.
Der 4. Operand der Ziel-Objektindex der Partner-Station.

Aufbau:

```
STXM    Kanal      ; Kanal-Nummer 10-99
        Anzahl     ; Subindex 0-255
        Quelle     ; Quell-Objektindex K 100-499
        Ziel       ; Ziel-Objektindex K 0-16383
```

Wobei:

Kanal:	10..99	für PCD4.M445
	10..19	für PCD2 mit PCD7.F700
Anzahl:	0..255	Subindex
		(0 = kein Subindex)
Quelle:		Quell-Objektindex (eigene Station)
	K 100..499	PCD4.M445
	K 100..199	PCD2 mit PCD7.F700
Ziel:		Ziel-Objektindex (Partner-Station)
	K 0..16383	

Beispiel:

```
STXM    11      ; Sendet via Kanal 11
        3       ;  Subindex (Element) 3
      K 122     :  sendet Objekt 122 der eigenen Station
      K 150     ;  zum Objekt 150 der Partner-Station
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXM während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: SRXM

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

STXMI INDIREKTES SENDEN IM MODUS S-BUS

(Serial Transmit Media Indirect (Mode SBUS))

Beschreibung: Dieser Befehl arbeitet gleich wie der STXM-Befehl. Der Unterschied liegt im indirekten Modus. Indirekt heisst, dass die Anzahl Medien für Quelle und Ziel mittels eines Registerinhalts angegeben werden. STXMI kann nur für die Übertragung von Medien verwendet werden. Das Übertragen der Uhr, des Display-Register usw. ist nicht möglich.

Aufbau:

STXMI	**Kanal**	**; Kanal**
	Anzahl	**; Anzahl oder Anzahl + Position**
	Quelle	**; Quellen-Typ und Registernummer**
	Ziel	**; Ziel-Typ und Registernummer**

Kanal:

Parameter für die Kanalnummer (Bereich 0 .. 3)

Anzahl oder Anzahl + Position:

Dieser Parameter ist eine Registernummer. Dieses Register enthält die "Anzahl" der Standard-Medien und die "Position" im Daten-Block. Für DB: "Anzahl" liegt im oberen Datenwort, "Position" im unteren Datenwort des Registers. In diesem Fall kann die Initialisierung dieses Registers einfach mit den LDL- und LDH-Befehlen ausgeführt werden.

Quellen-Typ und Registernummer

Ziel-Typ und Registernummer

Diese Parameter spezifizieren die Quelle und das Ziel einer Datenübertragung. Jeder dieser Parameter besteht aus einem Charakter mit dem Media-Typ (I/O/F/R/T/C/DB) und einer Registernummer (0 - 4095). Die Regeln der Quelle-Ziel-Datenüberragung der Befehle SRXM/STXM sind einzuhalten.

STXMI kann weder indexiert noch parameteriert werden

Beispiel:

```
STXMI     1
        R  100
        DB 101
        R  102
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXMI während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: STXM, Kommunikations-Befehle, Diagnose-Flags

Für weitergehende Informationen ist das "S-Bus Handbuch" (Best. Nr. 26/739) zu konsultieren.

STXMI INDIREKTES SENDEN IM MODUS PROFIBUS

(Serial Transmit Media Indirect (PROFIBUS))

Beschreibung: Sendet Daten (Objekte) im indirekten Modus von der eigenen Station und kopiert diese in die Partner-PCD. Der Kanal kann sowohl direkt als auch indirekt adressiert werden.

Aufbau:

```
STXMI   Kanal     ; Kanalnummer 10-99, R 0-4095
        Anzahl    ; Subindex R 0-4095
        Quelle    ; Quell-Objektindex K [R 0-4095]
        Ziel      ; Ziel-Objektindex K [R 0-4095]
```

Wobei::

Kanal:	10..99	für PCD4.M445
	10..19	für PCD2 mit PCD7.F700
	R 0..4095	für indirekte Adressierung
Anzahl:	R 0..4095	Register, welches den Subindex enthält (0 = ohne Subindex)
Quelle:	Register mit Quell-Objektindex (Eigene Station)	
	K [R 0..4095]	100..499 PCD4.M445
		100..199 PCD2 mit PCD7.F700
Ziel:	Register mit Ziel-Objektindex (remote)	
	K [R 0..4095]	0..16383

STXMI kann weder indexiert noch parameteriert werden

Beispiel:

```
STXMI   R 100  ; Kanal-Nummer in R 100
        R 110  ;  Subindex in R 110
        K 200  :  Quell-Objektindex in R 200
        K 300  ;  Ziel-Objektindex in R 300
```

Flags: Das Error-Flag wird gesetzt, wenn der Kanal nicht richtig assigniert wurde oder wenn STXMI während einer laufenden Kommunikation ausgeführt wird.

Siehe auch: STXM

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

SICL PRÜFEN VON STEUERSIGNALEN

(Serial Input Control Line)

Beschreibung: Prüft ein Meldesignal der Schnittstelle und lädt dessen Zustand in den ACCU.

Im Operand der 1. Zeile wird die Kanal-Nummer angegeben.

Im Operand der 2. Zeile wird angegeben, welches Signal gelesen werden soll:

0 = CTS	Clear To Send	Sendebereitschaft
1 = DSR	Data Set Ready	Betriebsbereitschaft
2 = DCD	Data Carrier Detect	Empfangssignalpegel

Für Schnittstelle 0 (PGU) der PCD1, PCD2, PCD4, PCD6.M3 und PCD6.M540 kann der Befehl SICL immer ausgeführt werden und zwar unabhängig davon, wie die Schnittstelle konfguriert ist.
Für alle andern Schnittstellen der PCD1, PCD2, PCD4, PCD6.M3 oder PCD6.M540 kann SICL nur ausgeführt werden, wenn die Schnittstelle als S-Bus-PGU assigniert wurde.
Generell ist SICL nur nach der Ausführung von SASI erlaubt.

Aufbau:

```
SICL    Kanal     ; Kanal-Nummer 0-3
        Signal    ; Signal-Nummer CTS|DSR|DCD  (0|1|2)
```

Beispiel:

```
SICL      0    ; Kanal 0
          1    ;   wenn DSR = H
CPB     H 25   ;   dann rufe  PB 25
```

Flags: Der ACCU wird gleich dem logischen Zustand des bezeichneten Steuersignals gesetzt.

Das ERROR-Flag wird gesetzt, wenn der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: SOCL, Kommunikations-Befehle

SOCL SETZEN VON STEUERSIGNALEN
(Serial Output Control Line)

Beschreibung: Setzt ein Steuersignal der bezeichneten Schnittstelle mit dem aktuellen logischen Zustand des ACCU (H, L)
Im Operand der 1. Zeile wird die Kanal-Nummer angegeben.
Im Operand der 2. Zeile wird angegeben, welches Signal gesetzt werden soll

0 = RTS	Request To Send	Sendeteil einschalten
	RS485: Accu high = senden, Accu low = empfangen	
1 = DTR	Data Terminal Ready	Endgerät betriebsbereit
2 =	Special functions	Spezial-Funktionen

Für Schnittstelle 0 (PGU) der PCD1, PCD2, PCD4, PCD6.M3 und PCD6.M540 kann der Befehl SOCL immer ausgeführt werden und zwar unabhängig davon, wie die Schnittstelle konfiguriert ist.
Für alle andern Schnittstellen der PCD1, PCD2, PCD4, PCD6.M3 und PCD6.M540 kann SOCL nur ausgeführt werden, wenn die Schnittstelle als S-Bus-PGU assigniert wurde.
Generell ist SOCL nur nach der Ausführung von SASI erlaubt.

Aufbau:

```
SOCL    Kanal     ; Kanal-Nummer 0-3
        Signal    ; Signal-Nummer RTS|DTR (0|1|2)
```

Beispiel:

```
ACCU  L
SOCL    0       ; Kanal 0
        1       ;   DTR wird = L gesetzt (wie ACCU)
```

Flags: Das ERROR-Flag wird gesetzt, wenn der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: SICL, Kommunikations-Befehle

Spezial-Funktionen:

Schnittstelle 0 der PCD2

Ein SASI für SM/SS im Anwenderprogramm konfiguriert die Schnittstelle für RS485. Möchte der Anwender die Schnittstelle als RS232 verwenden, ist nach SASI die folgende Sequenz zu programmieren:

```
ACC     L
SOCL      0
          2
```

Umschalten von RS485 zu RS422

Die serielle Schnittstelle RS422/RS485 der PCD4.C130, PCD4.C340, PCD7.F110/F150, PCD7.F520/530 schalten automatisch zu RS485 wenn gewisse Modi assigniert sind.

Mode	**Typ**
MC0 .. MC3, MD0 / SD0	RS422
MC4, S-Bus	RS485

Es kann in gewissen Fällen notwendig werden, die PCD dazu zu bringen, S-Bus mit RS422 zu verwenden. In diesem Fall sind die folgenden Befehle nach SASI auszuführen:

```
ACC         L
SOCL          Port_nb
              2
```

Es ist auch möglich den RS485-Modus mit MC0..MC3 oder MD0/SD0 zu erzwingen. Es ist zu programmieren:

```
ACC         H
SOCL          Port_nb
              2
```

Umschalten vom Empfangs- zu Sendemodus in RS485

RS485 in Sendemodus schalten:

Es sind die folgenden Befehle nach SASI auszuführen:

```
ACC         H
SOCL          Port_nb
              0
```

RS485 in Empfangsmodus schalten:

```
ACC         L
SOCL          Port_nb
              0
```

SCON VERBINDUNG VIA LAN1 ÖFFNEN/SCHLIESSEN

(Serial Connect to LAN 1)

Beschreibung: Öffnet oder schliesst eine virtuelle Verbindung über das SAIA-LAN1 zu einer andern Station.

Im Operand der 1. Zeile wird die Nummer der Schnittstelle angegeben
Im Operand der 2. Zeile steht die Stationsnummer der Partner-Station
Im Operand der 3. Zeile wird angegeben, ob die Verbindung geöffnet (1) oder geschlossen (0) werden soll.

Der Status der Verbindung steht im Register <mode_opt>, welches bei der Assignierung angegeben wurde. (z.B. "MODE:SD0,R4000;")

Das Register <mode_opt> kann folgende Information enthalten:

Wert	Bedeutung
0	Disconnected (Verbindung geschlossen)
1	Connected (Verbindung offen)
2	Queued (in Warteschlange)
3	Destination busy (Partner-Station belegt)
4	Destination unknown (Partner-Station unbekannt)
6	Remote PLC not connected (Partner-Station ausgeschaltet oder vom Netz getrennt)
10..2500	Wurde die Verbindung durch eine Partner-Station erstellt, steht die Nummer der aufrufenden Station, multipliziert mit 10, im Register <mode_opt>.

Aufbau:

```
SCON    Kanal      ; Kanal-Nummer 0 - 3
        Station    ; LAN 1 Stations-Nummer 1 - 250
        Status     ; Verbindungs-Status
                   ; (0 = geschlossen, 1 = offen)
```

Beispiel:

```
SCON    0       ; Kanal 0
        100     ;   LAN1-Station Nr. 100
        1       ;   Verbindung erstellen (öffnen)
```

Flags: Das ERROR-Flag wird gesetzt, wenn der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: SASI, Kommunikations-Befehle

Für weitergehende Informationen ist das SAIA® LAN1-Handbuch zu konsultieren

SCON VERBINDUNG VIA PROFIBUS ÖFFNEN

(Open Communication Channel (PROFIBUS))

Beschreibung: Öffnet oder schliesst einen virtuellen PROFIBUS-Kanal
Bevor Informationen zwischen Stationen ausgetauscht werden können, muss der virtuelle Kommunikations-Kanal mit SCON geöffnet werden.
Der Operand der 1. Zeile ist die Kanalnummer
Der Operand der 2. Zeile ist immer = 0 (nicht verwendet)
Der Operand der 3. Zeile ist Verbindungs-Status (0 = geschlossen, 1 = offen)

Aufbau:

```
SCON    Kanal     ; PROFIBUS Kanal-Nummer 10-99
        0         ;
        Status    ; Verbindungs-Status
                  ; (0 = schliessen, 1 = öffnen)
```

Beispiel:

```
SCON    10      ; Kanal 10
        0       ;
        1       ;  öffnen
```

Flags Das Error-Flag wird gesetzt, wenn der Kanal nicht existiert

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

SCONI INDIREKTE VERBINDUNG VIA LAN1

(Serial Connect to LAN 1 Indirect)

Beschreibung: Öffnet oder schliesst eine virtuelle Verbindung über das SAIA-LAN1 zu einer andern Station im indirekten Modus.

Im Operand der 1. Zeile wird die Nummer der Schnittstelle angegeben
Im Operand der 2. Zeile steht die Adresse eines Registers, welches die Stationsnummer der Partner-Station enthält
Im Operand der 3. Zeile steht die Adresse eines Registers, welches den Verbindungsstatus enthält (0 = schliessen, 1 = öffnen)

SCONI kann weder indexiert noch parameteriert werden

Der Status der Verbindung steht im Register <mode_opt>, welches bei der Assignierung angegeben wurde. (z.B. "MODE:SD0,R4000;")

Das Register <mode_opt> kann folgende Information enthalten:

Wert	Bedeutung
0	Disconnected (Verbindung geschlossen)
1	Connected (Verbindung offen)
2	Queued (in Warteschlange)
3	Destination busy (Partner-Station belegt)
4	Destination unknown (Partner-Station unbekannt)
6	Remote PLC not connected (Partner-Station ausgeschaltet oder vom Netz getrennt)
10..2500	Wurde die Verbindung durch eine Partner-Station erstellt, steht die Nummer der aufrufenden Station, multipliziert mit 10, im Register <mode_opt>.

Aufbau:

```
SCONI    Kanal      ; Kanal-Nummer 0 - 3
         Station    ; LAN 1 Stations-Nummer R 0 - 4095
         Status     ; Verbindungs-Status R 0 - 4095
```

Beispiel:

```
SCONI     0     ; Kanal 0
        R 100   ;   LAN 1 Stations-Nummer in R 100
        R 101   ;   Verbindungs-Status in R 101
```

Flags: Das ERROR-Flag wird gesetzt, wenn der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: SCON, SASI, Kommunikations-Befehle

Für weitergehende Informationen ist das SAIA® LAN1-Handbuch zu konsultieren

SCONI INDIREKTE VERBINDUNG VIA PROFIBUS

(Open Communication Channel Indirect (PROFIBUS))

Beschreibung: Öffnet oder schliesst einen virtuellen PROFIBUS-Kanal im indirekten Modus.
Bevor Informationen zwischen Stationen ausgetauscht werden können, muss der virtuelle Kommunikations-Kanal mit SCON geöffnet werden.
Der Operand der 1. Zeile ist die Kanalnummer oder ein Register, welches die Kanalnummer enthält.
Der Operand der 2. Zeile ist ein Register, dessen Inhalt immer 0 sein muss.
Der Operand der 3. Zeile ist ein Register, welches den Verbindungsstatus enthält (0 = geschlossen, 1 = offen)

SCONI kann weder indexiert noch parameteriert werden

Aufbau:

```
SCONI   Kanal     ; PROFIBUS Kanal-Nr. 10-99 oder R 0- 4095
        Station   ; R 0- 4095 (immer 0)
        Status    ; Verbindungs-Status R 0- 4095
```

Beispiel:

```
SCONI   R 10    ; Öffnet den PROFIBUS-Kanal, definiert in R 10
        R 11    ;  Stationsnummer in R 11 (muss immer = 0 sein)
        R 12    ;  Verbindungs-Status in R 12
```

Flags: Das ERROR-Flag wird gesetzt, wenn der Kanal nicht richtig oder gar nicht assigniert ist.

Siehe auch: SCON

Für mehr Information ist das "PROFIBUS-Handbuch" zu konsultieren

9 LAN2-Befehle

Das SAIA LAN2 ist ein geschlossenes Bus-Netzwerk (Local Area Network), das nach dem “Token Passing” -Prinzip arbeitet. Es können bis zu 255 Stationen in 8 Segmenten zu je 32 Stationen bedient werden.

Das LAN2 ist ein Master-Master-System d.h., jede Station kann Master sein.

Jede Station muss mit einem eigenen LAN2-Modul bestückt sein. (PCD6.T1.. oder PCD4.M340).

Mit dem LAN2 können PCD-Elemente wie Eingänge, Ausgänge, Flags, Register, Timer, Counter sowie der Status einer Station (Connect, Disconnect, usw.) übertragen werden.

Es stehen 4 Befehle zur Verfügung. 2 zum Senden und Empfangen von Elementen und 2 zum Senden und Empfangen des Status. Die Information, was wohin zu übertragen ist, steht in Anwendertexten.

Für ein erfolgreiches Arbeiten mit dem SAIA-LAN2 ist eine saubere Installation der Kabel, der Abschluss-, der “Pull-Up” - und “Pull-Down” -Widerstände eine wesentliche Voraussetzung. Es sind unbedingt die Hinweise im SAIA-LAN2 Handbuch und in den Hardware-Handbüchern zu befolgen.

WICHTIG:
Die nachfolgenden Beschreibungen gelten für die LAN2-Firmware ab V004.

LAN2-Befehle:

LRXD	Receive Data via LAN2	PCD-Daten empfangen
LTXD	Transmit Data via LAN2	PCD-Daten senden
LRXS	Receive Status via LAN2	Status empfangen
LTXS	Transmit Status via LAN2	Status senden

Notizen

LRXD PCD-DATEN VIA LAN2 EMPFANGEN (Receive Data Via LAN2)

BESCHREIBUNG: Empfängt PCD-Daten via das LAN2 von einer Partner-Station.

Im Operand der 1. Zeile des Befehls wird die Priorität der Behandlung des Befehls angegeben. (0 = hohe, 1 = niedrige Priorität, Normalfall).

Im Operand der 2. Zeile wird eine Text-Nummer angegeben. In diesem Text steht der Auftrag, ab WELCHER Partner-Station WAS zu lesen und WOHIN die Information in der eigenen Station abzulegen ist.

In Operand der 3. Zeile wird die niedrigste Adresse von 10 aufeinanderfolgenden Diagnose-Flags angegeben. Die angegebene Flag-Adresse ist zugleich das EXEC-Flag. Dieses ist während der Übertragung eines Telegramms = L und wird am Ende der Übertragung = H.

AUFBAU:

LRXD[X]	**Priorität**	**; 0 = hoch, 1 = tief**
	Text-Nr. (i)	**; Text-Nr. 0-3999**
	Diagnose	**; F, O 0-8181**

BEISPIEL:

```
        LRXD    1       ; normale Priorität (tief)
                900     ;   Auftrag in Text 900
                F 200   ;   Diagnose in F 200-209
```

FLAGS: Unverändert

SIEHE AUCH: LTXD, LRXS, LTXS, LAN2 Diagnose-Flags, LAN2-Texte, SAIA-LAN2 Handbuch.

ANWENDUNG: Nach dem Einschalten des Eingangs I 1 an der eigenen Station, sind die Eingänge I 0 - 7 von der Partner-Station Nr. 3 zu lesen und auf die Ausgänge O 32-39 der eigenen Station zu übertragen.

```
                COB     5
                        0
                STH     I 1     ; wird Eingang I 1 = H
                DYN     F 1     ;   (Flanke)
                ORL     F 100   ;   oder EXEC-Flag = L
                CPB     H 50    ; dann rufe PB 50 auf
                ECOB
                ;-------------------------------------------

                PB      50
                LRXD    1       ; Priorität tief
                        15      ;   Text-Nr.
                      F 100     ;   EXEC-Flag + Diagnose
$LAN
        TEXT    15      "3:I0-7:O32-39"; Auftrag
$ENDLAN
                EPB
```

LTXD PCD-DATEN VIA LAN2 SENDEN (Transmit Data Via LAN2)

BESCHREIBUNG: Sendet PCD-Daten via das LAN2 zu einer Partner-Station.

Im Operand der 1. Zeile des Befehls wird die Priorität der Behandlung des Befehls angegeben. (0 = hohe, 1 = niedrige Priorität, Normalfall).

Im Operand der 2. Zeile wird eine Text-Nummer angegeben. In diesem Text steht der Auftrag, zu WELCHER Partner-Station WAS zu senden und WO-HIN die Information in der Partner-Station abzulegen ist.

Im Operand der 3. Zeile wird die niedrigste Adresse von 10 aufeinanderfolgenden Diagnose-Flags angegeben. Die angegebene Flag-Adresse ist zugleich das EXEC-Flag. Dieses ist während der Übertragung eines Telegramms = L und wird am Ende der Übertragung = H.

AUFBAU:

LTXD[X]	**Priorität**	**; 0 = hoch, 1 = tief**
	Text-Nr. (i)	**; Text-Nr. 0-3999**
	Diagnose	**; F, O 0-8182**

BEISPIEL:

```
        LTXD     1      ; normale Priorität (tief)
                 900    ;   Auftrag in Text 900
               F 200    ;   Diagnose in F 200-209
```

FLAGS: Unverändert

SIEHE AUCH: LRXD, LRXS, LTXS, LAN2 Diagnose-Flags, LAN2-Texte, SAIA-LAN2 Handbuch.

ANWENDUNG: Nach dem Einschalten des Eingangs I 8 an der eigenen Station, sind die Eingänge I 0 - 7 der eigenen Station zu den Ausgängen O 40 - 47 der Partner-Station Nr. 3 zu senden.

```
                COB      5
                         0
                STH      I  8     ; wird Eingang I 8 = H
                DYN      F  8     ;   (Flanke)
                ORL      F  110   ;   oder EXEC-Flag = L
                CPB      H  51    ; dann rufe PB 51 auf
                ECOB
                ;-------------------------------------------------

                PB         51
                LTXD       1      ; Priorität tief
                           16     ;   Text-Nr.
                         F 110    ;   EXEC-Flag + Diagnose
$LAN
        TEXT    16       "3:I0-7:O40-47"   ; Auftrag
$ENDLAN
                EPB
```

LRXS STATUS VIA LAN2 EMPFANGEN (Receive Status Via LAN2)

BESCHREIBUNG: Liest den Status einer Partner-Station oder die Statistik der Übertragungen der eigenen Station.

Die Status können sein:

HALT	HALT der CPU
RUN	RUN der CPU
DIS	DISCONNECT: Ausgeschaltet oder vom LAN2 getrennt
CON	CONNECT: Ans LAN2 angegschaltet

Im Operand der 1. Zeile wird eine Text-Nummer angegeben. In diesem Text steht der Auftrag, von welcher Station der Status zu lesen ist. Das Resultat steht in den eigenen Status-Flags.

In Operand der 2. Zeile wird die niedrigste Adresse von 10 aufeinanderfolgenden Diagnose-Flags angegeben. Die angegebene Flag-Adresse ist zugleich das EXEC-Flag. Dieses ist während der Übertragung = L und wird am Ende der Übertragung = H.

LRXS hat immer hohe Priorität.

AUFBAU:

```
LRXS[X]   Text  (i)     ;Text-Nr. 0-3999
          Diagnose      ; F, O 0-8182
```

BEISPIEL:

```
        LRXS     100   ; Auftrag in Text 100
                 O 32   ;   Diagnose in 0 32-41

TEXT 100         "020"  ; Lese Status von Station 20
```

FLAGS: Unverändert

SIEHE AUCH: LTXD, LRXD, LTXS, LAN2 Diagnose-Flags, LAN2-Texte, SAIA-LAN2 Handbuch.

LTXS STATUS VIA LAN2 SENDEN (Transmit Status Via LAN2)

BESCHREIBUNG: Ändert den Status einer Partner-Station.

Die änderbaren Status können sein:

HALT	HALT der CPU (aller CPU) einer Station
RUN	RUN der CPU (aller CPU) einer Station
DIS	DISCONNECT: Vom LAN2 trennen
CON	CONNECT: Ans LAN2 anschalten
TOUT:nnn	Setze den Timeout nnn ist der Wert in 1/10sek. (10-250).

Im Operand der 1. Zeile wird eine Text-Nummer angegeben. In diesem Text steht der Auftrag, an welcher Station der Status zu ändern ist.

Im Operand der 2. Zeile wird die niedrigste Adresse von 10 aufeinanderfolgenden Diagnose-Flags angegeben. Die angegebene Flag-Adresse ist zugleich das EXEC-Flag. Dieses ist während der Übertragung = L und wird am Ende der Übertragung = H.

LTXS hat immer hohe Priorität.

AUFBAU:

```
LTXS[X]   Text  (i)      ;  Text-Nr. 0-3999
          Diagnose       ;  F, O 0-8182
```

BEISPIEL:

```
          LTXS   1000  ; Auftrag in Text 1000
                 F 100 ;   Diagnose in F 100-109

TEXT 1000 "035:DIS"     ; Disconnect Station 35
```

FLAGS: Unverändert.

SIEHE AUCH: LRXD, LTXD, LRXS, LAN2 Diagnose-Flags, LAN2-Texte, SAIA-LAN2 Handbuch.

LAN2 DIAGNOSE-FLAGS

In jedem LAN2-Befehl wird die niegrigste Adresse von 10 aufeinanderfolgenden Diagnose-Flags (oder Ausgängen) angegeben.

Flag		Status “H”	Status “L”
0	EXECuted	Auftrag ausgeführt	Auftrag in Arbeit
+1	FAILure	Fehler	i. o.
+2	Status der eigenen PCD	Ungültiger LAN-Text	Gültiger Text
+3		LAN2-Prozessor abgeschaltet	LAN2-Prozessor angeschaltet
+4		Time out (Sendefehler)	i.o.
+5	Status der Partner-PCD	LAN2-Prozessor abgeschaltet	LAN2-Prozessor abgeschaltet
+6		-	-
+7		Elemente schreibschützt	i.o.
+8		CPU 0 in HALT	CPU 0 in RUN
+9	Watch Dog *)	Neu konfiguriert	i.o.

*) ab CPU Firmware Version: V002 für PCD4
V006 für PCD6

EXEC-Flag: Ausführungs-Flag

Das erste Diagnose-Flag ist das EXEC-Flag. Dieses ist während dem Senden oder Empfangen eines Telegramms = L.

Das Anwenderprogramm muss so gestaltet werden, dass der Ausführungs-Befehl bis zum Ende der Telegramm-Übertragung dauernd repetiert wird. Das Ende des Telegramms wird durch das Abfragen des EXEC-Flags festgestellt.
Ob für alle LAN2-Befehle jeweils die gleichen oder für jeden Befehl andere Diagnose-Flags definiert werden, hängt von der Struktur des Programms ab. (Siehe Kommunikations- Handbuch).
Beim Einschalten der Steuerung sind im XOB 16 alle EXEC-Flags “H” zu setzen.

PRIORITY: Priorität “0" = hohe, ”1" = niedrige Priorität.
Der Transfer von Daten geschieht in sogenannten “Frames”. (Frame = Rahmen). 1 Frame ist ein Datenpaket von 256 Bit. Dies entspricht 256 Einzel-Bit (Eingänge, Ausgänge, Flags) bzw. 8 Register (Timer oder Counter) zu je 32 bit.
Ist ein Telegramm länger als 1 Frame, wird dieses in eben diese Datenpakete aufgeteilt, und bei jedem Vorbeikommen des “Token” (Token = aktiver Punkt im LAN2-Transfer) wird 1 Frame übertragen. Ein solches Telegramm hat immer niedrige Priorität (auch wenn hohe Priorität angegeben ist).
Kurze Telegramme, d.h. Telegramme deren Inhalt innerhalb 1 Frames liegen, können mit hoher (0) oder niedriger (1) Priorität übertragen werden. Hohe Priorität heisst, dass ein solches Telegramm bei einer laufenden Übertragung eines langen Telegramms zwischen zwei Frames des langen Telegramms eingeschoben und nicht erst am Ende des langen Telegramms behandelt wird, wie dies bei niedriger Priorität der Fall ist.
Status-Telgramme haben immer hohe Priorität (0).

Die Bedeutung und die Verwendung der andern Diagnose-Flags ist dem SAIA-LAN2 Handbuch zu entnehmen.

LAN2-Texte für die Datenübertragung (LRXD und LTXD)

Der Auftrag WAS von WO nach WOHIN zu übertragen ist, steht in einen LAN2-Text. LAN2-Texte sind Anwendertexte in einem LAN2-spezifischen Format.
FORMAT:

Stations-Nummer: Nummer der Partner Station (0-254).
Quell-Bereich: Element-Typ und Adressbereich der Quelle (beim Senden die Elemente der eigenen, beim Empfangen die Elemente der Partner-Station).
Ziel-Bereich: Element-Typ und Adressbereich des Ziels (beim Senden die Elemente der Partner-, beim Empfangen die Elemente der eigenen Station).

Die Adress-Bereiche der Elemente können aus **max. 8 Einzel-Adressen** vom gleichen Typ oder einer **zusammhängenden Reihe** von Elementen bestehen, z.B.:

- "R15,25,35,45,55,65" für Einzel-Elemente
- "F1000-2000" für eine Reihe

Der Quell-Bereich und der Ziel-Bereich müssen übereinstimmen.

Welche Elemente zu welchen übertragen werden können, ist der folgenden Tabelle zu entnehmen.

Quelle	Ziel					Adress-
	O	F	T	C	R	Bereich
I	•	•				0..8191
O	•	•				0..8191
F	•	•				0..8191
T			•	•		0..450
C			•	•		0..1599
R					•	0..4095

Beispiele für Empfangs-Texte (LRXD): (direkte Adressierung)

TEXT 100 "015:O64-127:F1000-1063"

Liest in Station 15 die log. Zustände der Ausgänge 64-127 und bringt diese auf die Flags 1000-1063 der eigenen Station.

TEXT 101 "008:T11,13,25:C55,125,1231"

Liest in Station 8 die Inhalte der Timer 11, 13 und 25 und lädt diese in die Counter 55, 125 und 1231 der eigenen Station.

Beispiel für Sende-Text (LTXD): (direkte Adressierung)

TEXT 75 "000:R11-22:R33-44"

Überträgt den Inhalt der Register R 11-22 der eigenen Station in die Register 33-44 der Partner-Station Nr. 0.

Indirekte Adressierung

Sind in einem Projekt sehr viele LAN2-Transfers vorgesehen, müssen auch ebenso viele Einzel-Texte bereitgestellt werden. Die Anzahl Texte sind auf 4000 beschränkt. Mit der indirekten Adressierung können Texte eingespart werden, dies allerdings auf Kosten der Übertragungszeiten, bzw. auf Kosten der Aufbereitungszeit der Telegramme in der PCD.

Die indirekte Adressierung verwendet das Zeichen “@”. Kommen in einem LAN2-Text ein oder mehrere “@” vor, gelten die Regeln der indirekten Adressierung, die von der direkten, bezüglich der Adressierung von Element-Reihen, etwas abweicht. Die direkte und die indirekte Adressierung können also gemischt werden.

Indirekte Adressierung von Einzelementen:

FORMAT:

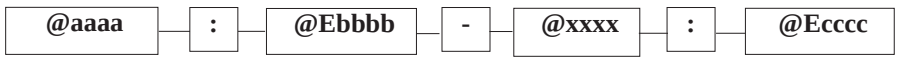
aaaa, bbbb, cccc sind Register-Adressen (R 0-4095) E bezeichnet den Element-Typ (I, O, F, R, T, C).
aaaa : Register enthält die Adresse der Partner-Station
bbbb : Register enthält die Adresse des Quell-Elements
cccc : Register enthält die Adresse des Ziel-Elements

Beispiel: “@100:@I400:@F600"

In Register 100 steht die Adresse der Partner-Station.
In Register 400 steht die Adresse der Quell-Eingangs und
in Register 600 steht die Adresse des Ziel-Flags.

Indirekte Adressierung von Reihen-Elementen:

FORMAT:



aaaa, bbbb, cccc, xxxx sind Register-Adressen (R 0-4095) E bezeichnet den Element-Typ (I, O, F, R, T, C).
aaaa : Register enthält die Adresse der Partner-Station
bbbb : Register enthält die niedrigste Adresse der Quell-Elements
xxxx : Register enthält die ANZAHL zu übertragender Elemente
cccc : Register enthält die niedrigste Adresse der Ziel-Elements

Beispiel: “@200:@C100-@101:@C500"

In Register 200 steht die Adresse der Partner-Station. In Register 100 steht die niedrigste Adresse der Quell-Counter, in Register 101 steht die ANZAHL der zu übertragenden Counter und in Register 500 steht die niedrigste Adresse der Ziel-Counter. Die Bereichs-Endadresse wird von der PCD selbst ausgerechnet.

Achtung: Es ist daruf zu achten, dass die Anzahl zu übertragender Elemente die Bereiche nicht überschreiten. Der Assembler kann diese Werte nicht überprüfen, da das Laden der jeweiligen Register im Anwenderprogramm geschieht.

Ein Überlaufen der Bereiche kann fatale Folgen haben!

Beispiele für gemischte Schreibweise:

“@5:R100-50:@R99"

In Register 5 steht die Adresse der Partner-Station. Es werden 50 Register ab Register-Adresse 100 übertragen. Die niedrigste Adresse des Ziel-Registers liegt in Register 99.

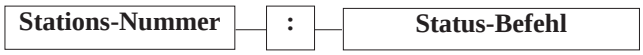
“25:I0-@55:F1000"

Die Partner-Station hat die Nummer 25. Der niedrigste Quell-Eingang hat die Adresse 0. Die Anzahl zu übertragender Eingänge steht in Register 55. Die Eingänge werden in Flags ab der Flag-Adresse 1000 übertragen.

Es ist zu beachten, dass bei gemischter Adressierung die Schreibweise der indirekten Adressierung gilt!

LAN2-Text zum Übertragen des Status (LTXS)

Der Status einer Partner-Station kann beeinflusst werden.
FORMAT:



Stations-Nummer	
0-254	Der Status wird zur spezifizierten Station übertragen
255	Der Status wird an ALLE Stationen im Netzwerk übertragen, ausser der eigenen.
Befehl	
HALT	Bringt alle CPU der adressierten Station in HALT. Das Anwenderprogramm wird gestoppt. Diagnose-Flag 8 wird gesetzt.
RUN	Bringt alle CPU der adressierten Station in RUN.
CON	Schaltet die adressierte LAN-Station ans Netz. Diagnose-Flag 3 oder 5 wird “L”.
DIS	Schaltet die adressierte LAN-Station vom Nezt ab. In diesem Fall werden keine LAN-Befehle mehr abgearbeitet und auch keine Daten mehr transferiert. Diagnose-Flag 3 oder 5 wird “L”.
TOUT:NNN	Setzt den Timeout der adressierten Station. “nnn” ist die Anzahl Stationen im Netzwerk (2-255).

Beispiel für einen Status-Text (LTXS):

```
TEXT    213      “018:HALT”
```

Der Status der Station 18 wird HALT.

LAN2-Text zum Lesen des Status (LRXS)

Der Status einer Partner-Station kann abgefragt werden.

FORMAT:

Der Status der adressierten Partner-Station wird in die im Befehl bezeichneten Status-Flags 3, 5 oder 8 geladen.

Beispiel für einen Status-Text (LRXS):

```
TEXT 137        “015"
```

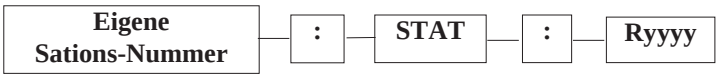
Der Status der Station 15 wird in die Status-Flags gelesen:

“Disconnected”	EXEC-Flag + 5
“Halted”	EXEC-Flag + 8

Das “Failure” -Flag (EXEC-Flag +1) wird nicht verändert, da es sich nicht um Fehler handelt.

Überwachung der LAN2-Übertragungen (LRXS)
(Line Traffic Control)

Mit Hilfe eines Spezial-Texts kann der LAN2-Transfer überwacht werden.
FORMAT:



Mit diesem LAN2-Text wird eine Statistik erstellt. Das bezeichnete Register yyyy ist die niedrigste Adresse von 4 aufeinanderfolgenden Registern.

Diese Statistik gibt Aufschluss über die Güte des Netzwerks.

Die Werte in den Registern bedeuten:

Register	Bedeutung
xxxx	Anzahl empfangene Frames
xxxx + 1	Anzahl gesendete Frames
xxxx + 2	Anzahl Wiederholungen (=Anzahl Timeouts * 3)
xxxx + 3	Anzahl zurückgewiesener Telegramme (unvollständige, beschädigte Telegramme)

Die Werte werden im 16-Bit-Format abgelegt (0-65’535).

Beispiel :

```
TEXT    123     “100:STAT:R20"
```

Es wird eine Statistik der eigenen Station Nr. 100 erstellt. Die Werte sind in den Registern R 20-23 abgelegt.

Schreibweise der LAN2-Texte in der Quelldatei (Source-File):

Werden LAN2-Texte zwischen die beiden Direktiven $LAN und $ENDLAN geschrieben, wird die Syntax (richtige Schreibweise) dieser Texte beim Assemblieren geprüft. Wurden LAN2-Texte z.B. mit Kleinbuchstaben geschrieben, werden automatisch die obligatorischen Grossbuchstaben erzeugt.

Beispiele:

```
$LAN

TEXT 0        “2:I0-255:F1000-1255"
TEXT 1        “5:r501-510:r101-110"
TEXT 2        “7:f0-31:o96-127"
TEXT 15       “2:con”
TEXT 37       “15"
TEXT 101      “8:t11,13,25:c55,125,1231"
TEXT 75       “0:R11-22:R33-44"
TEXT 200      “@100:@i400:@f600"
TEXT 1111     “@200:@C100-@101:@C500"
TEXT 99       “@5:R100-50:@R99"
TEXT 3999     “25:I0-@55:F1000"

$ENDLAN

TEXT 224      “Dies ist ein ASCII-Text<10><13>”
```

Symbole in LAN2-Texten

Es können in den LAN2-Texten Symbole verwendet werden:

```
QUELLE      EQU   R 55
ZIEL        EQU   R 66

$LAN

TEXT  25    “8:” , QUELLE.T , “:” , ZIEL.T

$ENDLAN
```

Dies ergibt nach dem Assemblieren: “008:R55:R66"

Siehe auch “Symbole in Texten”, in Abschnitt 8.

Notizen

10 Befehle zur PROGRAMMSTEUERUNG

Mit diesen Befehlen kann der Programmablauf beeinflusst werden.

Sprungbefehle sollten nur in überschaubaren, gut getesteten Programmteilen (in Programm-Blocks PB oder Funktions-Blocks FB) angewendet werden und NICHT zur Steuerung des Hauptprogramms.

Programmsprünge sind gefährliche Fehlerquellen, die Programme blockieren und zum Absturz bringen können.

Gut strukturierte Programme haben KEINE Programmsprünge !

BLOCTEC und GRAFTEC erlauben Strukturen ohne Programmsprünge.

Die Operanden aller Befehle dieses Kapitels können nicht als Parameter an FB übergeben werden.

Programmsteuer-Befehle:

JR	**J**ump **R**elative	Relativer Programm-Sprung
JPD	**J**um**P** **D**irect	Direkter Programm-Sprung
JPI	**J**um**P** **I**ndirect	Indirekter Programm-Sprung
HALT	**HALT**s a CPU	HALT einer CPU
LOCK	**LOCK** Semaphore	Setzen eines Spezial-Flags
UNLOCK	**UNLOCK** Semaphore	Rücksetzen eines Spezial-Flags

Notizen

JR RELATIVER PROGRAMM-SPRUNG (Jump Relative)

BESCHREIBUNG: Bewirkt einen bedingten oder unbedingten Sprung vorwärts oder rückwärts. Die im Operand angegebene Anzahl Programmzeilen bezieht sich auf die Stelle im Programm, wo dieser Befehl geschrieben steht. Die Anzahl zu überspringenden Zeilen sind also relativ zur Adresse des Sprung-Befehls.
Es können max. -4096 Schritte rückwärts oder (+)4096 Schritte vorwärts gesprungen werden.
Ist die Bedingung nicht erfüllt, wird der Sprung nicht ausgeführt und das Programm fährt mit dem nächsten Befehl fort.

Die Sprungbedingungen können die folgenden sein:

-	Unbedingter Sprung (ohne Bedingung)
H	Sprung wenn ACCU = H
L	Sprung wenn ACCU = L
P	Sprung wenn Positiv-Flag = H
N	Sprung wenn Negativ-Flag = H
Z	Sprung wenn Zero-Flag = H
E	Sprung wenn Error-Flag = H

Es darf nur innerhalb eines Blocks gesprungen werden. (COB, XOB, PB, FB, ST oder TR).
Wenn aus irgendwelchen Gründen auf Sprünge nicht verzichtet werden kann, sind die Sprungadressen als Labels zu programmieren. Der Assembler rechnet dann die Anzahl Zeilen selbst richtig aus.
Die Anwendung von Labels ist im Anwender-Handbuch beschrieben.

AUFBAU:

```
JR [Bedingung] Ziel      ; Relative Anzahl Zeilen
                         ;    -4096. . .[+]4096
                         ;    oder Label
                         ;    Bedingung:
                         ;    H, L, P, N, Z, E
```

BEISPIEL:

```
        JR    H   -2      ; springe 2 Zeilen zurück,
                          ;    wenn ACCU = H
        JR    Z   NEXT    ; springe zum Label NEXT,
                          ;    wenn Z-Flag gesetzt
```

FLAGS: Unverändert

SIEHE AUCH: JPD, JPI, Anwender-Handbuch

ANMERKUNG: Gut strukturierte Programme haben keine Jumps!

```
        STH   I  15
        ANL      XBSY
        DYN   F  15
        JR    L  NEXT
        STXT     1
                 57
        ...
NEXT:   ...
```

———

```
w       STH   I  15
        ANL      XBSY
        DYN   F  15
        CPB   H  25
        ...
        ...
        PB       25
        STXT     1
                 57
        EPB
```

JPD DIREKTER PROGRAMM-SPRUNG (Jump Direct)

BESCHREIBUNG: Bewirkt einen bedingten oder unbedingten Sprung vorwärts. Die im Operand angegebene Anzahl Programmschritte beziehen sich auf den Beginn des laufenden Blocks (COB, XOB, PB, FB, ST oder TR).

Die Anzahl Schritte ist immer positiv und muss kleiner als die Gesamtzahl der Programmschritte des laufenden Blocks sein.

Ist die Bedingung nicht erfüllt, wird der Sprung nicht ausgeführt und das Programm fährt mit dem nächsten Befehl fort.

Die Sprungbedingungen können die folgenden sein:

-	Unbedingter Sprung (ohne Bedingung)
H	Sprung wenn ACCU = H
L	Sprung wenn ACCU = L
P	Sprung wenn Positiv-Flag = H
N	Sprung wenn Negativ-Flag = H
Z	Sprung wenn Zero-Flag = H
E	Sprung wenn Error-Flag = H

AUFBAU:

```
JPD  [Bedingung] Ziel     ; Anzahl Schritte vom
                          ;    Blockanfang (0-8191)
                          ; Bedingung:
                          ;    H, L, P, N, Z, E
```

BEISPIEL:

```
JPD   L 10    ; springt auf die 10. Zeile
              ;    des Blocks wenn ACCU = L
```

FLAGS: Unverändert.

SIEHE AUCH: JR, JPI.

JPI INDIREKTER PROGRAMM-SPRUNG (Jump Indirect)

BESCHREIBUNG: Dieser Befehl ist ähnlich dem vorangehenden JPD-Befehl. Der Unterschied liegt darin, dass die Anzahl Programmschritte ab dem Beginn des laufenden Blocks nicht direkt, sondern aus dem Register, das im Operand angegeben ist, geholt werden. Die max. Anzahl Schritte im Register ist auf 255 beschränkt. (Nur die letzten 8 Bit sind signifikant).

Ist die Bedingung nicht erfüllt, wird der Sprung nicht ausgeführt und das Programm fährt mit dem nächsten Befehl fort.

Die Sprungbedingungen können die folgenden sein:

-	Unbedingter Sprung (ohne Bedingung)
H	Sprung wenn ACCU = H
L	Sprung wenn ACCU = L
P	Sprung wenn Positiv-Flag = H
N	Sprung wenn Negativ-Flag = H
Z	Sprung wenn Zero-Flag = H
E	Sprung wenn Error-Flag = H

AUFBAU:

```
JPI  [Bedingung] Register ; Register, das die Anzahl
                          ;   Schritte ab dem
                          ;   Blockanfang enthält
                          ;   (1 - 255)
                          ;   Bedingung:
                          ;   H, L, P, N, Z, E
```

Da eine Bedingung angegeben werden kann, entfällt der Mediacode “R” für Register.

BEISPIEL:

```
JPI   H 300   ; springt auf die x-te Zeile
              ;   vom Blockanfang, wobei
              ;   x im Reg. R 300 steht
```

FLAGS: Unverändert.

ANMERKUNG: Es muss darauf geachtet werden, dass die Anzahl Programmschritte im Register nicht grösser als die Blocklänge sein darf. Da der Registerinhalt während dem Programmablauf eingeschrieben wird, kann der Assembler die Anzahl Schritte NICHT prüfen.

Es handelt sich hier um einen “gefährlichen” Befehl, der nur von geübten Programmierern angewendet werden sollte. Andererseits ergeben sich interssante Aspekte, um z.B. Sprungtabellen zu erstellen.

SIEHE AUCH: JR, JPI, LD.

HALT HALT EINER CPU (Halts the CPU)

BESCHREIBUNG: Erzwingt bedingt oder unbedingt den HALT einer CPU.
HALT ist nicht gleich Stop. Nach einem HALT geht nichts mehr und die CPU kann nur mit einem “Restart Cold” oder einem Aus- und Wiedereinschalten der Speisung wieder in RUN gebracht werden. Die Kommunikation zum Programmiergerät bleibt erhalten.

Ist die Bedingung nicht erfüllt, wird der HALT nicht ausgeführt und das Programm fährt mit dem nächsten Befehl fort.

Die HALT-Bedingungen können die folgenden sein:

-	Unbedingter HALT (ohne Bedingung)
H	HALT wenn ACCU = H
L	HALT wenn ACCU = L
P	HALT wenn Positiv-Flag = H
N	HALT wenn Negativ-Flag = H
Z	HALT wenn Zero-Flag = H
E	HALT wenn Error-Flag = H

AUFBAU:

```
HALT      [Bedingung]    ; Bedingung:
                                H, L, P, N ,Z, E
```

BEISPIEL:

```
HALT    E       ; HALT, wenn ERROR-Flag gesetzt
```

FLAGS: Unverändert (kommt auch nicht mehr drauf an)

ANMERKUNG: Der Befehl HALT wird für erste RUN-Tests ohne angeschlossenen Prozess verwendet. Es kann z.B. sofort auf den Aufruf eines XOB reagiert und die Fehlerursache geklärt werden.

In der Praxis ist der HALT-Befehl denkbar für im Programm erfassbare Notfälle, damit der Prozess angehalten und Schlimmeres verhindert werden kann.

In einem solchen Fall (STOP oder HALT) ist mit der Brücke “Reset Output” das sofortige Rücksetzen aller Ausgänge zu veranlassen. Über den Watch Dog-Kontakt kann auch die gesamte Speisung der Steuerung weggenommen werden.

SIEHE AUCH: Anwender-Handbuch, Abschnitt 3.5: Die XOB, DIAG.

ANWENDUNG: Falls das ERROR-Flag gesetzt wird, ist die PCD in HALT zu bringen und die Diagnose zu aktivieren.

```
XOB     13
DIAG    R  1000
HALT
EXOB
```

LOCK SETZEN EINES SPEZIAL-FLAGS (Semaphore-Variable) (Lock Semaphore)

BESCHREIBUNG: LOCK und UNLOCK dienen zur Verriegelung von Programmteilen bei Zugriffen von 2 oder mehreren CPU.

Es stehen dem Anwender im ganzen 100 Semaphore-Variablen (SV) zur Verfügung

Der LOCK-Befehl prüft die SV: ist diese “aus” (Ruhezustand), wird der ACCU = H und die SV gesetzt, ist diese bereits “ein” (eine andere CPU hat die SV gesetzt), wird der ACCU = L.

UNLOCK setzt die SV zurück. Dem LOCK-Befehl muss unbedingt und möglichst rasch ein UNLOCK folgen, da sonst Programmteile blockiert oder zumindest nicht mehr abgearbeitet werden.

AUFBAU:

LOCK	**Semaphore**	**; Semaphore 0-99**

BEISPIEL:

```
LOCK    1       ; Ist die SV = L (keine andere CPU hat die SV
CFB     H 100   ;   gesetzt) wird der FB 100 aufgerufen.
                ;   Ist die SV = H kann der FB nicht aufgerufen
                ;   werden, da eine andere CPU gerade z.B. Daten
                ;   verändert, die im FB 100 verarbeitet werden
                ;   sollen.
```

FLAGS: Der **ACCU** wird beeinflusst

ANMERKUNG: Zugriffskonflikte zwischen verschiedenen CPU können nur mit LOCK und UNLOCK verhindert werden. Versuche mit normalen PCD-Flags führen nicht zum gewünschten Ziel!

SIEHE AUCH: UNLOCK.

ANWENDUNG: siehe Beispiel bei UNLOCK (nächste Seite)

UNLOCK

UNLOCK RÜCKSETZEN EINES SPEZIAL-FLAGS (Semaphore-Variable) (Unlock semaphore)

BESCHREIBUNG: Siehe LOCK

AUFBAU:

UNLOCK Semaphore ; Semaphore 0-99

BEISPIEL:

```
UNLOCK    1  ; Semaphore wird rückgesetzt
```

FLAGS: Unverändert

SIEHE AUCH: LOCK.

ANWENDUNG: Eine PCD ist mit 2 CPU bestückt. In CPU 0 wird der Inhalt von 2 Registern addiert. In der CPU 1 werden BCD-Werte in diese beiden Register eingelesen.

Es muss verhindert werden, dass die beiden Register mathematisch behandelt werden. während gerade neue Werte geladen werden und umgekehrt.

CPU 0

```
....
LOCK        1
CFB     H   10
....
;————————————

FB          10
ADD     R   88
        R   89
        R   90
UNLOCK      1
EFB
```

CPU 1

```
....
LOCK        1
CFB     H   100
....
;————————————

PB          25
DIGI        2
        I   16
        R   88
DIGI        2
        I   24
        R   89
UNLOCK      1
```

Mit dieser Programmiertechnik ist gewährleistet, dass der FB 10 nur dann aufgerufen werden kann, wenn PB 25 NICHT in Abarbeitung ist und umgekehrt.

11. DEFINITIONS-Befehle

Diese Befehle werden nur einmal, unmittelbar nach dem Einschalten der Steuerung ausgeführt. Wird versucht, einen dieser Befehle während dem Programmablauf zu wiederholen, wird der Befehl ignoriert.

Normalerweise stehen diese Befehle im XOB 16 (Kaltstart-Routine).

Die Operanden aller Befehle dieses Kapitels können nicht als Parameter an FB übergeben werden.

DEFVM	**DEF**ine **V**olatile **M**emory	Definiert flüchtige Flags
DEFTC	**DEF**ine **T**imers/**C**ounters	Definiert Timer/Counter
DEFTB	**DEF**ine **T**ime**B**ase	Definiert Zeitbasis für Timer
DEFTR	**DEF**ine **T**imer **R**esolution	Definiert Auflösung der Timer

DEFWPR	**DEF**ine **W**rite **P**rotected Area (**R**un)
DEFWPH	**DEF**ine **W**rite **P**rotected Area (**H**alt)

Definieren schreibgeschützter Bereiche (LAN2)

Notizen

DEFVM DEFINIERT FLÜCHTIGE FLAGS

(Define Volatile Memory)

Beschreibung: Definiert die flüchtigen Flags (Volatile Flags).

Alle Flags in der PCD sind von Haus aus nicht-flüchtig, d.h. alle Flags behalten nach dem Aus- und Wiedereinschalten der Steuerung ihren logischen Zustand.

Mit dem Befehl DEFVM wird im Operand angegeben, welche Flags beim Einschalten der PCD rückgestellt werden und welche nicht. Die Zahl im Operand bedeutet, dass die Flags von Adresse 0 bis zu dieser Adresse flüchtig sind.

Dieser Befehl kann nur in der CPU 0 ausgeführt werden.

Aufbau:

DEFVM	**Flag**	**; Flag 0-8191**

Beispiel:

```
DEFVM   200     ; Flags   0 - 199 sind flüchtig,
                ; Flags 200 - 8191 sind nicht-flüchtig
```

Flags: Unverändert

Siehe auch: DEFTC, DEFTB, DEFWPR, DEFWPH

Anwendung: Flags 0 - 399 sollen als flüchtig definiert werden.

```
XOB         16      ; Kalt-Start XOB

DEFVM       400     ; Flags 0 - 399 sind flüchtig
....
EXOB
```

DEFTC DEFINIERT TIMER/COUNTER

(Define Timers/Counters)

Beschreibung: Definiert das Feld der Timer und Counter.

Von Haus aus sind die T/C-Adressen 0 - 31 Timer, und 32 - 1599 Counter.

Mit DEFTC kann diese Aufteilung geändert werden. Die Zahl im Operand bedeutet, dass ab dieser T/C-Adresse Counter liegen.

Es sollen nicht mehr Timer definiert, als wirklich gebraucht werden. Das Maximum von 450 Timern sollte nicht überschritten werden, da noch mehr Timer die Zykluszeit der PCD verlängern.

Dieser Befehl kann nur in der CPU 0 ausgeführt werden.

Aufbau:

DEFTC	**T/C-Adresse**	**; Untere Grenze für Counter (0-450)**

Beispiel:

```
DEFTC    64      ; Timer 0 - 63, Counter 64 - 1599
```

Flags Unverändert

Siehe auch: DEFTB, DEFTR, DEFVM, DEFWPR, DEFWPH

Anwendung: Es sollen 100 Timer breitgestellt werden

```
XOB         16    ; Kalt-Start XOB

DEFTC       100   ; 0..99 sind Timer
                  ; 100..1599 sind Counter
....
EXOB
```

DEFTB DEFINIERT ZEITBASIS FÜR TIMER

(Define Timebase)

Beschreibung: Definiert die Zeitbasis für das Dekrementieren der Timer.

Von Haus aus ist die Zeitbasis 100 ms (l/l0 sek).
Mit DEFTB kann die Zeitbasis zwischen l0 ms und l0 sek verändert werden. Für l0 ms ist im Operand = 1, für 1 sek = 100 und für l0 sek = 1000 zu schreiben. Zwischenwerte sind erlaubt.

Die Zeitbasis sollte nur im wirklichen Bedarfsfall kleiner als die voreingestellten 100 ms gemacht werden, da sonst die Zykluszeit der PCD erhöht wird.

Mit DEFTB wird die Zeitbasis für die Timer **aller** CPU definiert. Dieser Befehl kann nur in CPU 0 ausgeführt werden.

Wird DEFTB in anderen CPU ausgeführt, wird die Zeitbasis für die zeitverzögerten Befehle SETD und RESD der entsprechenden CPU verändert.

Aufbau:

DEFTB	**Zeitbasis**	**; in 1/100 sek (1- 1000)**

Beispiel:

```
DEFTB     100   ; Zeitbasis = 1 sec (100 * 10 ms)
```

Flags: Unverändert

Siehe auch: DEFVM, DEFTC, DEFWPR, DEFWPH,SETD, RESD

Anwendung: Die Zeitbasis soll 1 Sekunde betragen.

```
XOB       16    ; Kalt-Start XOB

DEFTB     100   ; Die Zeitbasis ist 100 * 10 ms = 1000 ms = 1 sek
....
EXOB
```

DEFTR DEFINIERE TIMER AUFLÖSUNG

(Define Timer Resolution)

Beschreibung: Definiert die Auflösung der Dekrementierung der Timer in **Millisekunden**. Wird z.B. "DEFTR 100" angegeben, werden alle Timer, welche nicht = 0 sind alle 100 ms um 100 dekrementiert. "DEFTR 1000" dekrementiert alle Timer alle 1000 ms um 1000. Sind DEFTR und DEFTB im gleichen Programm verwendet, wird die Meldung "DOUBLE TIME BASE" in die History Datei eingetragen und die CPU in "HALT" gebracht.

Der Vorteil des Befehls DEFTR gegenüber DEFTB liegt darin, dass die definierten Werte bei der Anwendung der Timer unabhängig von der eingestellten Zeitbasis oder der Auflösung sind und immer als ein Mehrfaches von 10 ms eingegeben werden. Damit der Befehl DEFTR die Wirkung auf die Timer ausüben kann, muss dieser in der CPU 0 programmiert werden. DEFTR erlaubt eine feinste Auflösung von 10 ms, was bedeutet, dass der angegebene Wert eventuell gerundet wird.

Beispiel: DEFTR 25: Es wird eine Zeitbasis von 20 ms eingestellt (25 wird zu 20 abgerundet). Der DEFTR Befehl, wie auch DEFTB beeinflussen die Befehle SETD, RESD and OUTD. Ist DEFTR im Anwenderprogramm verwendet, wird die Zeitbasis für diese Befehle auf **10 ms** fixiert, unabhängig des spezifizierten Wertes in DEFTR.

Aufbau:

DEFTR	**Auflösung**	**; Auflösung ≥ 10 ms**

Beispiel:

```
DEFTR   100     ; Auflösung  = 100 ms
```

Flags: Unverändert

Siehe auch: DEFTB

DEFWPR DEFINIERT SCHREIBGESCHÜTZTEN BEREICH IN RUN

(DEFINE WRITE PROTECTED AREA - RUN (LAN2))

Beschreibung: Definiert den Bereich eines Elementtyps (0, F, T/C, R), welcher durch eine andere PCD via das LAN2 während des Status RUN nicht verändert werden kann.

Wird der Befehl nicht ausgeführt, sind keine Elemente schreibgeschützt.

Der im Operand angegeben Wert ist die obere Grenze des bezeichneten Elementtyps. Die untere Grenze ist immer die Elementadresse 0. Pro Befehl kann jeweils nur ein Elementtyp schreibgeschützt werden. Sollen alle Elemente geschützt werden, ist der Befehl 4 mal auszuführen.

Dieser Befehl kann nur von CPU 0 ausgeführt werden.

Aufbau:

DEFWPR Adresse ; O 0 - 8191, F 0 - 8191, T 0 - 450, C 0- -1599, R 0 - 4095

Beipsiel:

```
DEFWPR  F 999 ; Flags 0 - 999 sind in RUN schreibgeschützt
              ;   und können via eine andere LAN-Station
              ;   nicht überschrieben werden.
```

Flags: Unverändert

Siehe auch: DEFWPH, DEFTC, DEFTB, DEFVM, LAN2

Anwendung: In einer mit LAN2 venetzten Anlage sollen die 1000 unteren Flags und die 500 unteren Register gegen Überschreiben via LAN2 in RUN geschützt sein.

```
XOB       16    ; Kalt-Start XOB

DEFWPR  F 999   ; geschützte Flags 0..999
DEFWPR  R 499   ; geschützte Register 0..499
....
EXOB
```

DEFWPH DEFINIERT SCHREIBGESCHÜTZTEN BEREICH IN HALT

(DEFINE WRITE PROTECTED AREA - HALT (LAN2))

Beschreibung: Definiert den Bereich eines Elementtyps (0, F, T/C, R), welcher durch eine andere PCD via das LAN2 während des Status HALT nicht verändert werden kann.

Wird der Befehl nicht ausgeführt, sind keine Elemente schreibgeschützt.

Der im Operand angegebene Wert ist die obere Grenze des bezeichneten Elementtyps. Die untere Grenze ist immer die Elementadresse 0. Pro Befehl kann jeweils nur ein Elementtyp schreibgeschützt werden. Sollen alle Elemente geschützt werden, ist der Befehl 4 mal auszuführen.

Dieser Befehl kann nur von CPU 0 ausgeführt werden.

Aufbau:

DEFWPH Adresse ; O 0 - 8191, F 0 - 8191, T 0 - 450, C 0 - 1599, R 0 - 4095

Beispiel:

```
DEFWPH  C 79    ; Timer und Counter 0 - 79 sind schreibgeschützt
                  (im Halt-Status)
```

Flags: Unverändert.

Siehe auch: DEFWPR, DEFTC, DEFTB, DEFVM, LAN2

Anwendung: In einer mit LAN2 venetzten Anlage sollen die 1000 unteren Flags und die 500 unteren Register gegen Überschreiben via LAN2 sowohl in RUN als auch in HALT geschützt sein.

```
XOB         16    ; Kalt-Start XOB

; Definition des Schreibschutzes in RUN
DEFWPR  F 999   ; geschützte Flags 0..999
DEFWPR  R 499   ; geschützte Register 0..499

; Definition des Schreibschutzes in HALT
DEFWPH  F 999   ; geschützte Flags 0..999
DEFWPH  R 499   ; geschützte Register 0..499
....
EXOB
```

12. SPEZIAL-Befehle

NOP	No operation	Keine Operation
RTIME	Read time	Lesen aus der PCD-internen Uhr
WTIME	Write time	Schreiben in die PCD-interne Uhr
PID	P.I.D. control	PID-Regelung
TEST	Test hardware	Hardware-Test
DIAG	Read XOB diagnostic	XOB Detail-Diagnose
SYSRD	System Read	Lesen von Systemdaten
SYSWR	System Write	Schreiben von Systemdaten
SYSCMP	System Compare	Vergleich von Systemdaten

Die folgenden Befehle sind nicht mehr zu verwenden. Die Befehle sind nur noch aus Kompatibilitätsgründen erwähnt.

ALGI	Analogue input	Analogwerte einlesen
ALGO	Analogue output	Analogwerte ausgeben

Diese beiden Befehle werden nur im Zusammenhang mit den alten Analogmodulen PCA2.W1x verwendet. Zum Lesen und Schreiben der Analogmodule der PCD1/2, PCD4 und PCD6 sind die entsprechenden Handbücher zu konsultieren.

STHS	Start high slow	Verlangsamte Abfrage
OUTS	Out slow	Verlangsamtes Setzen

Diese Befehle werden nur für den Zugriff zu den alten Analogmodulen PCA2.W2xx/W3xx verwendet.

Notizen

NOP KEINE OPERATION
(No Operation)

Beschreibung: Leerzeile, Befehl ohne Funktion.

NOP wird zum Überschreiben von nicht mehr gültigen Befehlszeilen verwendet. Auch um in einem Anwender-Programm freien Raum für eventuelle Korrekturen bzw. zusätzlichen Code zu schaffen, kann NOP eingesetzt werden.

Für die Programmierung ganz kurzer Pausen im µs-Bereich, können einige NOP eingesetzt werden, wobei zu beachten ist, dass die Länge eines NOP vom CPU-Typ abhängig ist.

Aufbau:

NOP	**; ohne Operand**

Beispiel:
```
NOP         ; Leerzeile
```

Flags: Unverändert.

RTIME LESEN AUS DER PCD-INTERNEN UHR

(Read Time)

Beschreibung: Liest die Daten der PCD-internen Uhr und überträgt diese in 2 aufeinanderfolgende Register. Im Operand des Befehls wird das untere der beiden Register angegeben. Nach der Ausführung von RTIME ist der Inhalt der beiden Register der folgende:

Digit Nr.	9	8	7	6	5	4	3	2	1	0
Register	0	0	0	0	Hour		Minute		Second	
Reg. + 1	0	Week		Wday	Year		Month		Day	

Week	Wochen-Nummer	1..53
Wday	Tages-Nummer	1..7 (Montag = 1, Sonntag = 7)
Year	Jahr	0..99
Month	Monat	1..12
Day	Datum	1..28/29/30/31
Hour	Stunde	0..23
Min	Minute	0..59
Second	Sekunde	0..59

In den Registern sind die Daten im Binärformat abgelegt und **nicht** in BCD. Mit den Befehlen MOV und DIGO können aber die einzelnen Werte im BCD-Format herausgelesen werden.

Aufbau:

RTIME	**Register**	**; Register-Nummer R 0-4095**

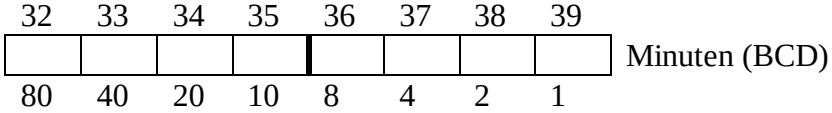
Beispiel:

```
RTIME   R 10   ; Die Datumuhr wird in die Register 10 und 11 kopiert
```

Flags: Unverändert

Siehe auch: WTIME, MOV, DIGO

Anwendung: Nach dem Einschalten von Eingang 3, sind die Minuten im BCD-Format an den Ausgängen 32-39 anzuzeigen.




```
COB       0
          0
STH     I 3
DYN     F 3
CPB     H 25
...
ECOB
```

```
PB        25
RTIME   R 20
MOV     R 20
        D 2
        R 99
        D 0
MOV     R 20
        D 3
        R 99
        D 1
DIGOR     2
        R 99
        O 32
EPB
```

WTIME SCHREIBEN IN DIE PCD-INTERNE UHR
(Write Time)

Beschreibung: Überträgt den Inhalt aus 2 aufeinanderfolgenden Registern in die PCD-inteme Uhr. Im Operand des Befehls wird das untere der beiden Register angegeben. Das Format der Daten in den Registern ist das gleiche wie für RTIME.

Digit Nr.	9	8	7	6	5	4	3	2	1	0
Register	0	0	0	0	Hour		Minute		Second	
Reg. + 1	0	Week		Wday	Year		Month		Day	

Zum Laden der Register mit BCD-Werten sind die Befehle MOV und DIGI zu verwenden.

Aufbau:

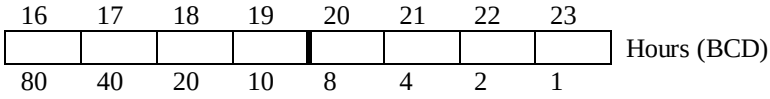
WTIME	**Register**	**; Register R 0-4094**

Beispiel: `WTIME R 500` ; Lädt die Uhr mit dem Inhalt von R 500 und R 501

Flags: Wird versucht, unmögliche Werte zu laden, wird das ERROR-Flag gesetzt und der Befehl wird nicht ausgeführt.

Siehe auch: RTIME, MOV, DIGI

Anwendung: Nach dem Einschalten von Eingang 4 sind in der Uhr die Stunden neu zu laden. Der neue Stundenwert liegt im BCD-Format an den Eingängen 16-23.




```
COB       0
          0
STH     I 4
DYN     F 4
CPB     H 26
...
ECOB
```

```
PB        26
RTIME   R 200
DIGIR     2
        I 16
        R 199
MOV     R 199
        D 0
        R 200
        D 4
MOV     R 199
        D 1
        R 200
        D 5
WTIME   R 200
EPB
```

PID PID-REGELUNG
(PID Control Algorithm)

Beschreibung: Ruft einen Algorithmus zur PID-Regelung auf. Die erforderlichen Parameter werden dabei aus dem im Operand des Befehls angegebenen Register und den nachfolgenden 12 Registern genommen.

Register	Bedeutung	Symbol		
+0	Neue Stellgrösse	Y_n	*	Wert 0 .. 'm' Bit
+1	Alte Stellgrösse	Y_{n-1}	*	Wert 0 .. 'm' Bit
+2	Neue Regelgrösse	X_n	w	Wert 0 .. 'm' Bit
+3	Alte Regelgrösse	X_{n-1}	*	Wert 0 .. 'm' Bit
+4	Aktueller Sollwert	W_n	w	Wert 0 .. 'm' Bit
+5	Alter Sollwert	W_{n-1}	*	Wert 0 .. 'm' Bit
+6	Proportional-Faktor	F_p	w	* 256
+7	Integral-Faktor	F_i	w	* 256
+8	Differential-Faktor	F_d	w	* 256
+9	Totzone	D_r	w	Wert 0 .. 'm' Bit
+10	Kaltstart-Stellgrösse	Y_s	w	Start-Wert für Yn Wert 0 .. 'm' Bit
+11	Auflösung	m	w	m = 8, 12, 16 Bit
+12	Rechenspeicher	Z_s	*	(Systemprogramm)

* Werte werden vom Systempr. der PCD automatisch behandelt
w Werte müssen durch Anwenderpr. in Register geschrieben werden

Aufbau:

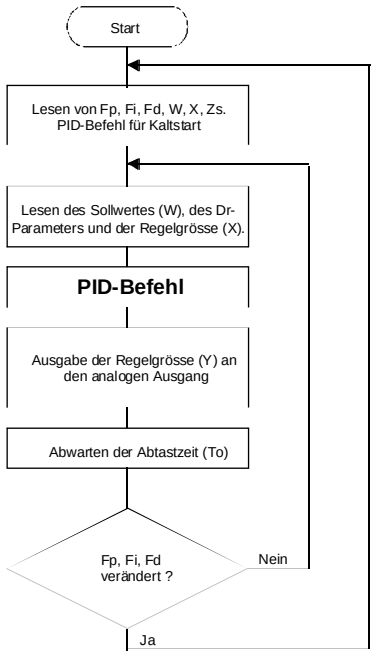
```
PID     Register  ; Das angegebene Register (R 0-4083) ist das
                  ;  unterste eines Blocks von 13 Registern
```

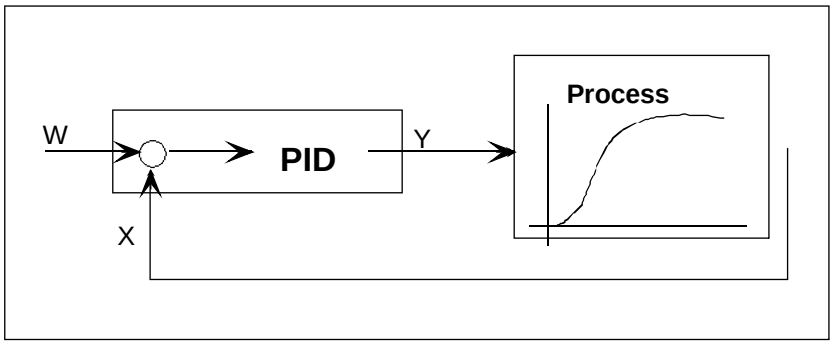
Beispiel:

```
PID     R 1000; Es werden die Register 1000 - 1012  verwendet
```

Flags: Das ERROR-Flag wird gesetzt, wenn ein Parameterbereich überschritten wird

Anwendung: Ein typischer PID-Regelkreis zeigt folgendes Bild:





Neue Stellgrösse Y_n:

Dies ist die aktuelle Stellgrösse, wie diese vom Systemprogramm gemäss der nachfolgenden Gleichung errechnet wird wobei $Z_s = Z_s + (W_n - X_n)$:

$$Y_n = \frac{F_p}{256} * \{ \ (W_n - X_n) + \frac{F_i}{256} * Z_s + \frac{F_d}{256} * [(W_n - W_{n-1}) - (X_n - X_{n-1})] \ \}$$

Ergibt der Rechenvorgang eine zu grosse Stellgrösse (> m Bit), so wird diese auf ihren Maximalwert (m Bit) begrenzt oder bei negativem Resultat auf 0 gesetzt.

Alte Stellgrösse Y_{n-1}:

Dies ist die alte, während dem vorangegangenen Durchlauf errechnete Stellgrösse.

Neue Regelgrösse X_n:

Die Regelgrösse Xn muss vom Anwenderprogramm in das entsprechende Register (R+2) geschrieben werden. Xn wird vom Systemprogramm zur Berechnung der Stellgrösse verwendet.

Alte Regelgrösse X_{n-1}:

Dies ist die alte, für den vorangegangenen Rechenvorgang verwendete Regelgrösse.

Aktueller Sollwert W_n:

Der Sollwert muss vom Anwenderprogramm in das entsprechende Register (R+4) geschrieben werden.

Alter Sollwert W_{n-1}:

Dies ist der alte, während dem vorangegangenen Durchlauf verwendete Sollwert. Damit lassen sich Sollwertsprünge feststellen und automatisch auswerten.

Proportional-Faktor F_P:

Dieser Faktor bestimmt die Proportional-Charakteristik (Verstärkung) der Regelung. Der Wert wird im Anwenderprogramm ins Register R+6 eingegeben. Bei der Berechnung werden nur die unteren 16 Bit verwendet (0-65'535).
Der Proportionalfaktor Fp ist folgendermassen definiert:

$$F_p = \frac{1}{X_p} * 256$$ mit X_p: Proportionalband

Bem.: Um ein Proportionalband von 5% einzugeben ist der Faktor F_P
(1 / 0.05) * 256 = 5120 einzugeben.

Die Ausführung eines Kaltstarts muss nach der Modifikation von F_p oder F_i erfolgen

Integral-Faktor F_i:

Dieser Faktor bestimmt die Integral-Charakteristik der Regelung. Der Wert wird im Anwenderprogramm in Register R+7 eingegeben. Bei der Berechnung werden nur die unteren 16 Bit verwendet (0-65'535).
Der Integral-Faktor ist wie folgt definiert:

$$F_i = \frac{T_0}{T_i} * 256$$ mit T_0: Abtastzeit des PID-Befehls
T_i : Integralzeit

Die Ausführung eines Kaltstarts muss nach der Modifikation von F_p oder F_i erfolgen

Differential-Faktor F_d:

Dieser Faktor bestimmt die Differential-Charakteristik der Regelung. Der Wert wird im Anwenderprogramm ins Register R+8 eingegeben. Bei der Berechnung werden nur die unteren 16 Bit verwendet.(0..65535)
Der Differetialfaktor Fd ist folgendermassen definiert:

$$F_d = \frac{T_d}{T_0} * 256$$ mit T_0: Abtastzeit des PID-Befehls
T_d : Differentialzeit

Totzone D_r:

Die Totzone gibt den Bereich an, in welchem Stellgrössenschwankungen auftreten können, ohne dass die Stellgrösse im Stellgrössenregister ändert.

Kaltstart-Stellgrösse Y_S:

Dies ist die Stellgrösse, welche vom Systemprogramm als Startwert Y_n verwendet wird. Sobald das Anwenderprogramm einen andern Wert als 0 in das Kaltstartregister schreibt, wird eine Kaltstartberechnung durchgeführt.

$$
\begin{array}{lcl}
Y_n & = & Y_s \\
Y_{n-1} & = & Y_s \\
Z_s & = & [(Y_s * 256/F_p) - (W_n - X_n)] * 256/F_i \\
W_{n-1} & = & W_n \\
X_{n-1} & = & X_n
\end{array}
$$

Der Wert von Y_s wird vom Systemprogramm nach einmaligem Gebrauch auf 0 zurückgesetzt und nicht mehr verwendet.

Für eine Kaltstart mit einem Ausgangswert von 0, muss das Register Y_s auf -1 gesetzt werden

Ist $F_i = 0$, kann der Wert Y_n nicht mit einem Kaltstart initialisiert werden. Ein Kaltstart wird jedoch für die Initialisierung des Arbeitsregisters empfohlen. In diesem Fall wird der Wert Y_s ignoriert, das Z_s-Register wird auf 0 gesetzt und Y_n nimmt den Wert des Proportionalteils des Algorithmus an.

Eine typische Anwendung für einen Kaltstart ist die Umschaltung von Handregelung auf automatische Regelung. Um einen möglichst gleichmässigen Übergang zu erreichen, kann Y_s gleich der momentanen Regelgrösse Y_n gesetzt werden.

Auflösung m:

Die max. Werte von X, W, Yn und Ys sind durch die Aufflösung gegeben.
Ist m= 8, wird mit 8 Bit gerechnet (0..255)
Ist m= 12, wird mit 12 Bit gerechnet (0..4095)
Ist m= 16, wird mit 16 Bit gerechnet (0..65'535)
Die Auflösung ist meistens durch das verwendete Analogmodul bestimmt. Ist die Auflösung für den Eingang und den Ausgang ungleich, muss der Y_n-Wert nach der Ausführung des PID-Befehls angepasst werden.

Abtastzeit:

Die Abtastzeit T_o muss ausserhalb des PID-Befehls mittels eines Timers bestimmt werden. Als praktischer Wert wird 1/10 der Zeitkonstante des Prozesses gewählt. 80 ms sind der kleinste praktisch erreichbare Wert für T_o.

Rechenkapazität:

Das Arbeitsregister Z_s hat die Kapazität von 2^{31}.
Werden 16-Bit Werte verwendet (m = 16), kann ein Überlauf erfolgen. In diesem Fall läuft die PID-Regelung nicht sauber. Um diesem Problem auszuweichen, muss der Faktor $F0 \geq 2$ sein. (Mit m = 8 oder 12 taucht dieses Problem nicht auf).

TEST TEST HARDWARE

Beschreibung: Bedingte oder unbedingte Funktionskontrolle von einem oder mehrerer Medien.

Wird bei einem Medium ein Fehler festgestellt, wird der Test abgebrochen und der ACCU = L gesetzt.

Ein fehlerfreier Test setzt den ACCU = H.

Die Tests werden nach der folgenden Tabelle gewählt.

Wert	Bit Nummer	Beschreibung des Tests
	11	
400	10	Verlust des Anwenderspeichers
200	9	Erweiterter Speicher verändert
100	8	RAM-Speicher des Systems
	7	
40	6	System EPROM
20	5	Serielle Schnittstellen
10	4	Datumuhr
	3	
4	2	Programm- und Textspeicher (Checksum)
2	1	Programm- und Textspeicher (RAM)
1	0	Allgemeines RAM (F, T, C, R, Mailbox)

Jedes gesetzte Bit entspricht dem entsprechenden Test.
Test 0 und 4 werden nur korrekt ausgeführt, wenn die zu testende CPU allein im RUN-Betrieb ist.

Anmerkung: Der Befehl TEST hat, je nach zu testenden Medien, eine relativ lange Abarbeitungszeit. Dieser Befehl sollte deshalb, nur beim Einschalten der Steuerung, vorzugsweise im XOB 16 durchgeführt werden.

Der Operand kann nicht als Parameter an FB übergeben werden.

Aufbau:

```
TEST    [CC] Code    ; CC = Condition Code (H|L|P|N|Z|E)
                     ;  Code gemäss Tabelle
```

Beispiel:

```
TEST     50   ; Unbedingter Test von: System EPROM (40)
              ;  + Datumuhr (10)

TEST     L 4  ; Test wenn ACCU = L: Progr.- und Textspeicher (EPROM)
```

Flags: ACCU = H, wenn alle Tests in Ordnung.
Bei einem unmöglichen Code wird das ERROR-Flag gesetzt.

Test des allgemeinen RAMs (Wert = 1)

Testet das RAM, welches die F/R/T/C enthält.
In einem Multiprozessorsystem darf nur eine einzige CPU in RUN sein.

ACCU	Error Flag	Beschreibung
0	1	Eine andere CPU war in RUN
0	0	Fehler im RAM festgestellt
1	x	Allgemeines RAM ist OK

Test des Programm- und Textspeichers (RAM) (Wert = 2)

Testet das RAM, welches Anwenderprogramm und Anwendertexte enthält. Ist der Speicher kein RAM oder ist das RAM schreibgeschützt, wird die Anwenderprogramm/Text-Checksum gebildet.

ACCU	Error Flag	Beschreibung
0	1	Programm-Header ist ungültig
0	0	RAM ist defekt
1	0	RAM ist OK

Test des Programm- und Textspeichers (Checksum) (Wert = 4)

Berechnet die Summe des ganzen Programm- + Textbereichs und vergleicht diese mit der Checksum im Header

ACCU	Error Flag	Beschreibung
0	1	Programm-Header ist ungültig
0	0	Checksum ist nicht OK
1	0	Checksum ist OK

Test der Datumuhr (Wert = 10)

Überprüft die Existenz der Datumuhr und testet das korrekte Inkrementieren.

ACCU	Error Flag	Beschreibung
0	1	Datumuhr ist nicht vorhanden
0	0	Datumuhr ist defekt
1	0	Datumuhr ist OK

Test der seriellen Schnittstellen (Wert = 20)

Test der seriellen Schnittstellen durch Initialisierung im Test-Modus "Loop-back". Es wird ein Test-Telegramm gesandt und danach der korrekte Empfang überprüft. Ist eine Schnittstelle bereits initialisiert, kann der Test nicht durchgeführt werden.

ACCU	Error Flag	Beschreibung
0	1	Eine der Schnittstellen assigniert
0	0	Eine Schnittstelle ist nicht OK
1	0	Alle Schnittstellen sind OK

Test des System-EPROMs (Wert = 40)

Die EPROMs mit der Firmware werden geprüft.

ACCU	Error Flag	Beschreibung
0	0	Checksum ist ungültig
1	0	Checksum ist OK

Test des System-RAM-Speichers (Wert = 100)

Die RAM-Speicher werden geprüft.

ACCU	Error Flag	Beschreibung
0	0	Checksum ist ungültig
1	0	Checksum ist OK

Test des erweiterten Speichers (Wert = 200)

Der erweiterte Speicher wird beim Einschalten der PCD geprüft. Sollte der Speicher verändert sein, wird ein internes Flag gesetzt.

ACCU	Error Flag	Beschreibung
0	0	Erweiterter Speicher ist verändert
1	0	Keine Veränderung

Test des Verlustes des Anwenderspeichers (Wert = 400)

Wird eine Charaktersequenz in der Mailbox der CPU in der Einschaltphase als verändert vorgefunden, kann angenommen werden, dass der Anwenderspeicher, während die PCD nicht an Spannung lag, wegen einer zu tiefen Batteriespannung verändert wurde.

ACCU	Error Flag	Beschreibung
0	0	Anwenderspeicher ist verändert
1	0	Keine Veränderung

DIAG XOB DETAIL-DIAGNOSE

(Read XOB Diagnostic)

Beschreibung: Überträgt Informationen zum aktuellen bzw. letzten XOB-Aufruf in einen Registerblock von 12 Registern. Im Operand wird die niedrigste Register-Adresse des Registerblocks angegeben. DIAG sollte in allen XOB einprogrammiert werden.

Bedeutung der einzelnen Register:

Register.		
0	XOB-Nummer	Nr. des zuletzt aufgerufenen XOB
+1	Programmzeile (Pz)	Pz., wo der XOB aufgerufen wurde
+2	Index-Register (IR)	Stand des IR beim Aufruf des XOB
+3	COB-Programmzeile	Programmzeile des Aufrufs
+4	Ebene 1	Programmzeile des Aufrufs
+5	Ebene 2	Programmzeile des Aufrufs
+6	Ebene 3	Programmzeile des Aufrufs
+7	Ebene 4	Programmzeile des Aufrufs
+8	Ebene 5	Programmzeile des Aufrufs
+9	Ebene 6	Programmzeile des Aufrufs
+10	Ebene 7	Programmzeile des Aufrufs
+11	nicht verwendet	

Die wichtigsten Informationen stehen in den Registern "0" und "1" und ev. noch im Register "2". Die Informationen in den folgenden Registern kann bei tiefverschachtelten PB-/FB-Strukturen von Interesse sein.

Der Operand kann nicht als Parameter an einen FB übergeben werden.

Aufbau:

DIAG	**Register**	**; R 0-4084, niedrigste Adresse von 12 Registern**

Beispiel: `DIAG R 1000`; XOB-Diagnose in den Registern 1000-1011

Flags: Unverändert

Siehe auch: Anwender-Handbuch, Abschnitt 3.5

Anwendung: Wird das ERROR-Flag gesetzt, soll die Uhrzeit, das Datum und die Programmzeile, wo das ERROR-Flag gesetzt wurde, via eine serielle Schnittstelle auf einen Drucker ausgegeben werden.

```
          XOB       13
          DIAG    R 1000
          STXT      1
                    100
TEXT 100
          "$D $ H :<CR><LF>"
          "Error-Flag bei Adresse $R1001 gesetzt"
          EXOB
```

SYSRD LESEN VON SYSTEMDATEN
(System Read)

Beschreibung: Liest Systemdaten wie:

Typ der PCD, Typ der CPU, Firmware-Version, Name des Anwenderprogramms, S-Bus-Parameter, usw.

Aufbau:

```
SYSRD    Funktion     ; Funktionscode, K-Code, R 0..4095
         Resultat     ; Resultat der Lesung, R 0..4095
```

Funktion

K x oder R x: Konstante oder Register welche(s) den Funktionscode enthalten. Der Befehl kann entweder direkt (durch Verwendung einer Konstanten) oder indirekt (durch Verwendung eines Registers) eingesetzt werden.

Resultat

R 0..4095 Register oder Folge von Registern, welche das Resultat enthalten.

Beispiel:

```
SYSRD   K 5000; Liest den Typ der PCD in ASCII
        R 20    ;  und legt das Resultat in R 20
```

Flags: Bei einem falschen Funktionscode wird das ERROR-Flag gesetzt.

Siehe auch: SYSWR

Function codes

Code	Beschreibung der Funktion	Resultat		
	Lesen Anwender EEPROM	Wert im EEPROM		
2000	Register 0 } PCD1			
2001	Register 1			
2002	Register 2			
2003	Register 3 } andere PCD			
2004	Register 4			
2005	Register 5			
¦	Register nn			
2049	Register 49			
5000	**Lesen PCD-Typ** in ASCII	ASCII	Dezima	Typ
5010	in Dezimal	" D1"	1	PCD1
		" D2"	2	PCD2
		" D4"	4	PCD4
		" D6"	6	PCD6
5100	**Lesen CPU-Typ** in ASCII	ASCII	Dezimal	Typ
5110	in Dezimal	" M1_"	10	PCD1.M1
		" M1_"	10	PCD2.M12
		" M15"	15	PCD2.M15
		" M11"	11	PCD4.M11
		" M12"	12	PCD4.M12
		" M14"	14	PCD4.M14
		" M24"	24	PCD4.M24
		" M34"	34	PCD4.M34
		" M44"	44	PCD4.M44
		" M1_"	10	PCD6.M1
		" M2_"	20	PCD6.M2
		" M3_"	30	PCD6.M3
		" M54"	54	PCD6.M5
5200	**Lesen der Firmware-Version** in ASCII	Beispiele von Resultaten: " $4C", " 004", " X41"		
5210	in Dezimal	Bsp.: 5 dez. für Version 005 -1 dez für alle '$', 'X', 'β'		
5400	**Lesen Name des Anwenderprogr.** in ASCII Der Name des Anwenderprogramms enthält immer 8 ASCII Character	**R x** enthält die 4 ersten Charakter des Namens in ASCII **R x+1** enthält die 4 nächsten Ch. des Namens in ASCII		
6000	**Lesen S-Bus Stations-Nummer**	Beispiel eines Resultats: 2 Stations-Nummer = 2 -1 Station nicht konfiguriert		
6010	**Lesen S-Bus PGU TN delay**			
6020	**Lesen S-Bus PGU TS delay**	Beispiel eines Resultats: 10 Delay in ms -1 S-Bus nicht konfiguriert		
6030	**Lesen S-Bus PGU timeout**			

Code	Beschreibung der Funktion	Resultat	
6040	**Lesen S-Bus PGU baudrate**	Beispiel eines Resultats: 9600 bps -1 S-Bus nicht konfiguriert	
6050	**Lesen des S-Bus PGU-Modus**	Status	Dez
		BREAK ohne Modems	0
		PARITY ohne Modems	1
		Data-Mode ohne Modem	2
		BREAK mit Modems	10
		PARITY mit Modems	11
		Data-Mode mit Modem	12
		S-Bus nicht konfiguriert	-1
6060	**Lesen S-Bus PGU-Port-Nummer**	Beispiel eines Resultats:: 1 S-Bus PGU-Port konfiguriert auf Port 1 -1 S-Bus nicht konfiguriert	
6070	**Lesen des S-Bus Level**	Status	Dez
		S-Bus Level 1 (reduziert)	1
		S-Bus Level 2 (voll)	2
		S-Bus nicht konfiguriert	-1
6080	**Lesen der CPU-Nr. an PGU** (S-Bus oder P8-Protokoll)	CPU 0	0
		CPU 1	1
6100	**Lesen des Modem-Status Bytes** Liest den aktuellen Status der Modem-Verbindung. Diese Information sagt dem Anwender, in welcher Phase der Initialisierung sich das Modem befindet. Mögliche Resultate: 2 PCD wartet auf Modem-Verbindung 6 .. 39 PCD initialisiert das Modem. 40 Neuassignierung der Schnittstelle im Modus SS1/SS0. 45..49 Modem-Verbindung verloren. Dies ist die Phase unmittelbar vor der Neuinitialiserung des Modems 50 Alles ist OK, die PCD ist online im Modus SS0/1.		
6500 6510 6520	**Lesen des Modem-Typs** **Lesen des Modem 'Reset Strings'** **Lesen des Modem 'Initialisierungs-Strings'**. Liest den spezifizierten Modem-String in den Registerblock, welcher mit der Basisadresse 'R x' beginnt.		
7000	**Lesen des System-Counters**	0.. 2.147.483.647	
	Ein interner System-Zähler wird alle Millisekunden inkrementiert. Dieser Zähler wird beim Einschalten der PCD auf 0 zurückgesetzt; ein "Restart Cold" hat auf diesen Zähler keinen Einfluss. Die Kapazität des Zählers ist genau: **24 Tage 20 Stunden 31 Minuten 23 Sekunden 647 ms** Ein Beispiel wird beim Befehl SYSCMP gezeigt.		

Lesen der Datum-Uhr

Es kann, abhängig vom Funktionscode, jeder Wert einzeln gelesen werden. Der gelesene Wert liegt im Dezimalformat vor. Die Funktionscodes liegen zwischen 7050 und 7090. Mit dem Funktionscode 7090 können die seit dem 01/01/1970 (00:00:00), gemäss der koordinierten Universalzeit, abgelaufenen Sekunden gelesen werden.

Funktionscode-Tabelle:

Funktionscode:	**Für:**
7050	Sekunden
7051	Minuten
7052	Stunden
7053	Minuten und Sekunden
7054	Stunden und Minuten
7055	Stunden, Minuten und Sekunden
7060	Datum
7061	Monat
7062	Jahr (< 100)
7063	Monat und Datum
7064	Jahr und Monat
7065	Jahr, Monat und Datum
7070	Wochentag
7071	Woche
7072	Woche und Wochentag
7081	Zeit und Datum (in 2 Register)
7090	Abgelaufene Sekunden seit 1970

Beispiele:

```
1)  SYSRD    7055     ; lesen der Stunden, Minuten und Sekunden
          R  0

    Resultat:  R 0:  120203


2)  SYSRD    7081     ; lesen der Zeit und des Datums
          R  0

    Resultat:  R 0:  120203    R 1: 991130
```

Anmerkung:

Wird ein Funktionscode zwischen 7050 und 7090 gewählt, welcher nicht in der Tabelle erwähnt ist, wird der XOB 13 aufgerufen und das Error-Flag gesetzt.

SYSWR SCHREIBEN VON SYSTEMDATEN

(System Write)

Beschreibung: Dieser Befehl ist das Komplement zum Befehl SYSRD. Es können damit Systeminformationen und Initialisierungsfunktionen via das Anwenderprogramm eingegeben werden.

Aufbau:

```
SYSWR    Funktion    ; Funktionscode, K-Code, R 0..4095
         Wert        ; Einzuschreibender Wert
```

Funktion

K x oder R x:	Konstante oder Register welche(s) den Funktionscode enthalten. Der Befehl kann entweder direkt (durch Verwendung einer Konstanten) oder indirekt (durch Verwendung eines Registers) eingesetzt werden.

Wert

K y	Einzuschreibender Wert
R 0..4095	Register, welches den einzuschreibenden Wert enthält

Beispiel:

```
SYSWD    K 4014; Initialisiert den XOB 14 mit einer Wiederholrate
         K 10    ;  von 10 ms
```

Flags: Bei einem falschen Funktionscode wird das ERROR-Flag gesetzt.

Siehe auch: SYSRD

Code	Beschreibung der Funktion
1000	**System-Watchdog (nur PCD1 und PCD2)** Erlaubte Werte für **K y** oder **R y**: 0 Desaktiviert den Watchdog 1 Aktiviert den Watchdog und erzwingt einen 'Restart Cold' wenn der Watchdog nicht innerhalb von 200 ms aufgefrischt wird. 2 Aktiviert den Watchdog und ruft den XOB 0 auf, bevor ein 'Restart Cold' innerhalb der 200 ms ausgeführt wird. Der Aufruf des XOB 0 wird bei einem Watchdog-Aufruf oder beim Aufruf durch einen Speisungsfehler unterschiedlich in die Historyliste eingetragen Watchdog: "XOB0 WDOG START" Speisung: "XOB 0 START EXEC".
2000 2001 2002 2003 2004 2005 ¦ ¦ ¦ ¦ 2049	**Schreiben ins EEPROM (nicht bei allen PCD)** Die PCD ist mit einem EEPROM bestückt, welches max. 49 Anwender-Register enthält. Diese können wie folgt beschrieben werden: Funktionscode (2000 - 2049) bezeichnet die Register 0 .. 49 im EEPROM. Erlaubte Werte für **K y** oder **R y**: PCD1; max. 5 Register (0 .. 5) Andere PCD: max. 49 Register (0 .. 49) R y: enthält den Wert, welcher ins EEPROM geschrieben werden soll. Achtung: Das EEPROM-Register lässt sich max. 100'000 mal überschreiben. Der SYSWR-Befehl darf deshalb nie in Programmschlaufen enthalten sein. Beim Schreiben muss beachtet werden, dass der Befehl SYSWR etwa 20 ms dauert. Der Befehl sollte nur für Intialisierungen beim Einschalten verwendet werden (XOB 16).
4000	**Definieren der XOB-Warteschlange** Die XOB 14/15/17/18/19/20/25 arbeiten mit dem Mechanismus der Warteschlange. Ist ein XOB aktiv, wird ein während dieser Zeit eintreffender neuer XOB in die Warteschlange gelegt, welche 127 Einträge pro XOB verkraften kann. Wird diese Limite überschritten, wird der XOB 7 aufgerufen und die Warteschlange gelöscht. Die Meldung 'SYSTEM OVERLOAD' wird in die Historyliste eingetragen. Eine Limite von 127 Einträgen ist möglicherweise für gewisse Anwendungen zu hoch und kann deshalb angepasst werden. Die Anpassung erfolgt gemeinsam für alle XOB, welche diesen Mechanismus verwenden. Erlaubte Werte für 'R y' oder 'K y': 0 .. 127

Code	Beschreibung der Funktion
4005 4013	**Aktivieren/Deaktivieren XOB 5 / 13** In bestimmten Fällen kann die sofortige Ausführung der XOB 5 bzw. 13 den Programmablauf stören, weshalb diese XOB deaktiviert werden können. Werden diese XOB während diese deaktiviert sind ein- oder mehrmals aufgerufen, werden diese nach dem erneuten Aktivieren 1x aufgerufen. Funktionscode: 4005 XOB 5 4013 XOB 13 Erlaubte Werte für 'R y' oder 'K y': 0 Deaktiviert den XOB 1 Aktiviert den XOB 2 Löscht das ERROR-Flag im aktuellen COB und im aktiven XOB (nur für K 4013)
4014 4015	**Installation des XOB 14 / 15** Konfigurierung von periodisch aufgerufenen XOB mit einer in 'K y' oder 'R y' definierten Wiederholungsrate. Es können 2 solche XOB mit einem Wiederholungsintervall von 5 ms bis 1000 sek definiert werden. Die Werte für 'K y' oder 'R y' werden in ms angegeben. Null bedeutet, dass der XOB deaktiviert ist. Dieser Befehl kann zu jeder beliebiger Zeit ausgeführt werden. Ist ein XOB in Ausführung während ein anderer XOB aufgerufen wird, wird dieser bis zum Moment, wo kein XOB mehr aktiv ist, aufgehoben und dann abgearbeitet. Die XOB werden nur behandelt, wenn die CPU in RUN oder CONDITIONAL RUN ist. Funktionscode 4014 Konfigurierung XOB 14 4015 Konfigurierung XOB 15 Erlaubte Werte für 'R y' oder 'K y': 5 .. 1'000'000
4017 4018 4019	**Ausführung der XOB 17 / 18 / 19** Aufruf der in 'R x' oder 'K x' definierten XOBs in der mit 'K y' oder 'R y' spezifizierten CPU Die XOBs 17/18/19 sind Anwender-XOBs, welche via den S-Bus oder das Anwenderprogrmm aufgerufen werden können. Die XOB werden nur behandelt, wenn die CPU in RUN oder CONDITIONAL RUN ist. Funktionscode: 4017 Aufruf XOB 17 4018 Aufruf XOB 18 4019 Aufruf XOB 19 Erlaubte Werte für 'R y' oder 'K y': 0 .. 6 CPU in welcher der XOB aufgerufen werden soll 7 Aufruf des XOB in der eigenen CPU 8 Aufruf des XOB in allen CPUs.

Code	Beschreibung der Funktion
6000	**Schreiben der S-Bus Stationsnummer** Ändert die S-Bus Stationsnummer zum Wert in 'K y' oder 'R y' (im System-RAM und im EEPROM). Dieser Befehl kann in Anwenderprogrammen in RAM (Schreibschutz), in EPROM und in Flash EPROM verwendet werden. Erlaubte Werte für 'K y' oder 'R y': 0 .. 254 **Schreiben ins EEPROM (nicht bei allen PCD)** Achtung: Das EEPROM-Register lässt sich max. 100'000 mal überschreiben. Der SYSWR-Befehl darf deshalb nie in Programmschlaufen enthalten sein. Beim Schreiben muss beachtet werden, dass der Befehl SYSWR etwa 20 ms dauert.
7000	**Konvertierung FFP-IEEE** Konvertiert zwischen FFP- (Fast Floating Point Format) und IEEE-Format für Fliesspunkt-Werte. Der FFP-Standard wird bei allen Fliesspunkt-Befehlen in der SAIA® PCD angewendet. Wurde ein Wert ins IEEE-Format gewandelt, kann dieser nicht mehr für Fliesspunkt-Operationen in der PCD verwendet werden. Funktionscode 7000 FFP- zu IEEE-Format 7001 IEEE- zu FFP-Format Erlaubte Werte für 'R y': 'R y' enthält den zu konvertierenden Wert. Das Resultat wird ins gleiche Register eingetragen.

Schreiben der Datum-Uhr

Es kann, abhängig vom Funktionscode, jeder Wert einzeln geschrieben werden. Der Wert ist im 2-stelligen Dezimalformat einzugeben, z.B. ist für 12h, 2 min und 3 sek 120203 zu schreiben. Die Funktionscodes liegen zwischen 7050 und 7081.

Funktionscode Tabelle:

Funktionscode:	Für:
7050	Sekunden
7051	Minuten
7052	Stunden
7053	Minuten und Sekunden
7054	Stunden und Minuten
7055	Stunden, Minuten und Sekunden
7060	Datum
7061	Monat
7062	Jahr (< 100)
7063	Monat und Datum
7064	Jahr und Monat
7065	Jahr, Monat und Datum
7081	Zeit und Datum (in 2 Register)

Beispiele:

```
1) LD     R  0
             120203

    SYSWR    7055     ; Schreiben von Stunden, Min. und Sek.
          R  0


2) LD     R  0
             120203
   LD     R  1
             991130

   SYSWR     7081     ; Schreiben von Zeit und Datum
          R  0
```

Anmerkung:

Wird ein Funktionscode zwischen 7050 und 7081 gewählt, welcher nicht in der Tabelle erwähnt ist, wird der XOB 13 aufgerufen und das Error-Flag gesetzt.

SYSCMP VERGLEICH VON SYSTEMDATEN
(System Compare)

Beschreibung: Mit der Kombination des vorliegenden Befehls SYSCMP und dem Befehl SYSRD K 7000 kann ein Timer mit einer Auflösung von 1 ms realisiert werden. Auch kann die Zeit zwischen zwei Ereignissen auf 1 ms genau gemessen werden.

SYSCMP wandelt jedes beliebige Register in einen Pseudo-Timer. Die Aufgabe des Befehls ist der Vergleich der Summe des ersten und des zweiten Operands des Systemcounters und das Setzen des ACCUs entsprechend dem Resultat.

Ist das Resultat der Addition grösser als der Systemcounter, wird der ACCU = H (1) gesetzt. Ist das Resultat kleiner oder gleich, wird der ACCU = L (0).

Aufbau:

```
SYSCMP   R x              ; R 0..4095
         K y oder R y     ; K 0..16383 oder R 0..4095
```

Beispiel:

```
SYSCMP   R 100    ; Vergleicht den Inhalt des Reg. 100 + der Konstanten
         K 1500;  1500 mit dem Systemcounter und setzt den ACCU
                  ;  entsprechend

SYSCMP   R 100    ; Vergleicht die Inhalte der Reg. R100+R101 mit dem
         R 101    ;  Systemcounter und setzt den ACCU entsprechend
```

Flags: Der ACCU wird gemäss dem Resultat gesetzt

Siehe auch: SYSRD

Anwendung: Programmierung eines hochauflösenden Timers (1 ms) mit SYSRD und SYSCMP

```
         COB      0
                  0
         …
         LD       R 100      ; Laden der Zeit in ms (1500)
                  K 1500     ;  in R 100

         SYSRD    K 7000     ; Lesen des Systemcounters in R 101
                  R 101

wait:    SYSCMP   R 101      ; Vergleich des Systemcounters
                  R 100      ;  mit R100 + R101
         JR       H wait     ; Ist ACCU = H (1), warten
         …
         ECOB
```

ALGI ANALOG-WERTE EINLESEN
(Analogue Input)

Beschreibung: Es wird der Analogwert auf dem Analog-Eingangsmodul **PCA2.W1..** A/D gewandelt (12 Bit) und in das in der 2. Zeile des Befehls angegebene Register übertragen.

In der 1. Zeile des Befehls wird der Analog-Kanal (0-7) **und** die Basisadresse des Moduls angegeben.

Wird der Operand der 1. Zeile als FB-Parameter übergeben, müssen die Kanalnummer und die Basisadresse auf der gleichen Zeile angegeben werden.

Aufbau:

```
ALGI[X]   Kanal   Basisadresse   ; Kanalnummer 0 - 7,
          Register (i)          ; Basisadresse 0 - 8176
                                ; Register R 0-4095
```

Beispiel:

```
ALGI    2   64  ; Liest den Analogwert von Kanal 2 des Moduls mit
        R   10  ;  der Basisadress 54 und überträgt den Wert in R 10
```

Flags: Das Z-, das P- und das N-Flag werden gemäss dem gelesenen Wert gesetzt. Das ERROR-Flag wird immer zurückgesetzt.

Siehe auch: ALGO, Hardware-Handbuch der PCA2

Anmerkung: Dieser Befehl kann nicht für die Analog-Module der PCD-Reihe verwendet werden.

ALGO ANALOG-WERTE AUSLESEN

(Analogue Output)

Beschreibung: Es wird eine 12 Bit Binärzahl aus dem in der 1. Zeile des Befehls bezeichneten Registers auf dem Analog-Ausgangsmodul PCA2.W1.. D/A gewandelt und ausgegeben.

In der 2. Zeile wird der Analog-Ausgangskanal (0-3) **und** die Basisadresse des Moduls angegeben.

Wird der Operand der 2. Zeile als FB-Parameter übergeben, müssen die Kanalnummer und die Basisadresse auf der gleichen Zeile angegeben werden.

Aufbau:

```
ALGO[X]   Register  (i)          ; Register R 0 - 4095
          Kanal     Basisadresse ; Kanalnummer 0 - 3
                                 ; Basisadresse 0 - 8176
```

Beispiel:

```
ALGO    R   100; Gibt den Wert im R 100 als Analogwert auf Kanal 3
        3   128;  des Moduls mit der Basisadresse 128 aus
```

Flags: Unverändert

Siehe auch: ALGI, Hardware-Handbuch der PCA2

Anmerkung: Dieser Befehl kann nicht für die Analog-Module der PCD-Reihe verwendet werden.

STHS VERLANGSAMTE ABFRAGE

(Start High Slow)

Beschreibung: Abfrage des adressierten Elementes auf seinen logischen Zustand (H/L) und speichern dieses logischen Zustandes in den ACCU, genau wie bei STH.

Die Abfragezeit ist jedoch länger und somit für langsamere (ältere) Module, wie die PCA2.W2../W3.. geeignet.

Aufbau:

STHS[X]	**Element (i)**	**; I 0 - 8191, O 0 - 8191, F 0 - 8191**

Beispiel:

```
STHS    I 25
```

Flags: Der ACCU wird gemäss dem logischen Zustand des Elementes gesetzt

Siehe auch: OUTS, STH, Bit-Befehele

OUTS VERLANGSAMTES SETZEN
(Out Slow)

Beschreibung: Das adressierte Element wird mit dem logischen Zustand des ACCU gesetzt (H/L), genau wie bei OUT. Die Dauer des Setzens oder Rücksetzens ist aber länger als bei OUT.

Der Befehl wird für die älteren PCA2.W2../W3.. verwendet.

Aufbau:

OUTS[X]	**Element (i)**	**; I 0 - 8191, O 0 - 8191, F 0 - 8191**

Beispiel:
```
OUTS    O 32
```

Flags: Der ACCU wird nicht verändert

Siehe auch: OUT, OUTD, STHS

Anwendung: Es soll der Analog-Eingangswert von Kanal 0 eines PCA2.W2..-Moduls mit der Basisadresse 96 gelesen und ins Register R 100 abgelegt werden.

```
COB       0
          0
...
ACC     H           ; zur Sicherheit: ACCU = H
OUTS    O 96        ; Wahl des Ausgangskanals
...                 ; Warte ≥ 100 ms *)
CPB       RD_VAL    ; Call RD_VAL Program Block
....
ECOB

PB        RD_VAL
BITIR     8         ; Liest 8 Bit
        I 104       ;    ab Adressen 104 .. 111
        R 100       ;    in Register 100
EPB
```

*) Die 100 ms Wartezeit kann mit einem Timer, welcher geladen und abgewartet wird oder auch mit einer Anzahl aufeinanderfolgenden NOPs realisiert werden. (Die Anzahl NOPs ist vom CPU-Typ abhängig)

(Diese Wartezeit hat nichts mit dem Befehl OUTS zu tun).

Notizen

Notizen

13. History-Liste

Es folgt eine detaillierte Aufstellung aller Fehler, welche in die History-Liste ("HISTORY LIST") oder ins Halt-Register ("HALT REASON REGISTER") eingeschrieben werden können. Die Liste kann im Debugger mit dem Befehl "Display History" angezeigt werden.

Fehler, welche den Aufruf eines XOB (wenn programmiert) zur Folge haben

Meldung	HALT	XOB	System*)	Bedeutung
XOB START EXEC	N	0	Alle	XOB 0 wurde aufgerufen
XOB 0 EXECUTED	N	0	Alle	XOB 0 wurde ganz abgearbeitet
XOB 0 WDOG START	N	0	1	System-Watchdog wurde aufgerufen
EXTERN PWR FAIL	N	1	2, 5, 6	Speisungsfehler in der Erweiterung
PARITY FAILURE	N	4	5, 6	Bus-Fehler in der PCD6
SYSTEM OVERLOAD	N	7	Alle	Warteschlange der Level-3 XOB überlaufen
ILLEGAL OPCODE	N	8	Alle	Nach einem ungültigen Befehl wird der XOB 8 wird aufgerufen und danach ein "Restart Cold" ausgeführt
>32 ST/TR ACTIVE	N	9	Alle	Zu viele aktive GRAFTEC-Zweige
>7 CALL LEVELS	N	10	Alle	PB/FB-Verschachtelung zu tief

Die nachfolgenden Fehler haben einen festen Platz und einen Ereigniszähler in der History-Liste

Meldung		HALT	XOB	System	Bedeutung
BATT FAIL	000	N	2	2, 4, 5, 6	Batterie ist entladen
IO QUIT FAIL	000	N	5	4, 5, 6	Nicht bestückter E/A-Platz wurde angesprochen
IR OVERFLOW	000	N	12	2, 4, 5, 6	Index-Register >8191 inkrementiert
ERROR FLAG	000	N	13	2, 4, 5, 6	Error-Flag wurde gesetzt

*) System: 1: PCD1
2: PCD2
4: PCD4
5: PCD6.M540
6: PCD6.M1.., M2..**, M3..**

Fehler beim Start der PCD (System Startup Errors)

Alle nachfolgend aufgeführten Fehler werden beim Start der PCD erfasst

Meldung	HALT	System	Bedeutung
RTC FAILURE	Y	Alle	Datumuhr erkannt aber funktioniert nicht richtig
DUART HW ERROR	Y	Alle	Einer der DUART ist defekt
CHECKSUM FAIL	Y	Alle	Checksum des EPROM des Anwenderprogr. ist falsch
BAD TXT/DB TABLE	Y	Alle	Problem beim 'make text table' beim Start
TXT/DB HW ERROR	Y	Alle	Problem beim 'make text table' beim Start
BAD MEM EXT INIT	Y	Alle	Problem im erweiterten Speicher beim 'make text table' beim Start
USR MEM HW ERROR	Y	6	Fehler beim Anwender Programm-Test beim Start
CPU SYNCH ERROR	Y	6	Timeout der 2. CPU
CPU FIRMWARE MIX	Y	4, 6	Verschiedene Firmware-Versionen in den CPU bei Multiprossessor-Anwendung

System-Fehler

Diese Fehler werden ins "HALT REASON"-Register eingeschrieben, welches nach einem HALT im Debugger eingesehen werden kann. Diese Fehler können beim Start der PCD oder während RUN auftreten.

Meldung	HALT	System	Bedeutung
BUS QUIT FAILURE	Y	Alle	Die Firmware versuchte eine nicht existiernede Adresse anzusprechen
68K INVALID OPC	Y	Alle	Eine ungültiger 68000 Befehl wurde ausgeführt
68K ADDR ERROR	Y	Alle	Aufruf einer ungeraden Adresse
ZERO DIVIDE	Y	Alle	Interne Firmware-Fehler
68K CHK INSTR	Y	Alle	…
68K TRAPV INSTR	Y	Alle	…
PRIVILEGE VIOL	Y	Alle	…
TRACE	Y	Alle	…
ILLEGAL AUTO VEC	Y	Alle	…
INTERRUPT ERROR	Y	Alle	…
RESERVE INT	Y	Alle	…

Fehlermeldungen zur Programmierung oder zur Konfigurierung

Die folgenden Fehler werden beim Einschalten der PCD entdeckt

Meldung	HALT	System	Bedeutung
EVERYTHING IS OK	N	Alle	Alles OK beim Einschalten
MODIFIED PROGRAM	N	Alle	Das Anwenderprogramm wurde im Debugger modifiziert. Nur angezeigt, wenn Programm schreibgeschützt ist.
CPU NUMBER > 6	Y	6	Am DIL-Schalter gewählte CPU-Nummer ist falsch
CPU 0 START FAIL	Y	4, 5, 6	Keine CPU 0 vorhanden
INIT-FAILURE	Y	Alle	Mehr als GRAFTEC 32 Initial-Steps sind programmiert
HEADER FAIL	Y	Alle	Programm-Header verändert
NO PROGRAM	Y	Alle	Kein Anwenderprogramm da
MEM-EXT CORRUPT	Y	Alle	Speichererweiterung im RAM verändert
INVALID OPCODE	Y	Alle	Ungültiger IL-Befehl in die CPU geladen
MEDIA CORRUPTION	Y/N	Alle	Batteriefehler
DOUBLE TIME BASE	Y	Alle	DEFTB- und DEFTR -Befehl im gleichen Programm
BAD MODEM STRING	Y	1	Modem-String im EEPROM zu lang

Die folgenden Fehler werden während RUN erfasst. Die Befehle, welche einen HALT verursachen, werden auch ins HALT-Register eingeschrieben.

Meldung	HALT	System	Bedeutung
BLOC NONEXISTENT	Y	Alle	Aufruf nicht vorhandener PB, FB, SB, ST, TR
HALTED BY LAN-2	Y	4, 6	Der LAN-2 Coprozessor hat die PCD in HALT gebracht
LAN-2 WATCHDOG	N	4, 6	Der LAN-2 FW-Watchdog hat angesprochen
HALT INSTRUCTION	Y	Alle	Ein programmierter HALT-Befehl wurde ausgeführt
MANUAL HALT	Y	4, 5, 6	CPU wurde durch den HALT-Schalter blockiert
HALTED BY CPU 0	Y	4, 6	Coprozessor(en) durch die CPU 0 angehalten
SBUS-PGU ERROR	Y	Alle	Ungültige S-Bus-Assignierung auf einem Port

Notizen

Absender:

Firma
Abteilung
Name
Adresse

Tel.

Datum

An:

Saia-Burgess Controls AG
Bahnhofstrasse 18
CH-3280 Murten (Schweiz)
http://www.saia-burgess.com

Befehlssatz für die PCD-Familie

Falls Sie Vorschläge zu SAIA® PCD zu machen oder Fehler in diesem Handbuch gefunden haben, sind wir Ihnen für einen kurzen Bericht dankbar.

Ihre Vorschläge: