INDEL BETRIEBSSYSTEM

I S M - 6.0

Version: PC

Referenz Manual

Inhalt

Einleitung

Allgemeines

Geschichte: Das INDEL-Betriebssystem ISM wurde 1980 geschaffen, um komplexe Maschinen, Anlagen und Prozesssteuerungen zu programmieren. Durch laufende Anpassung an die Bedürfnisse der Zeit ist es ein leistungsfähiges und einfach zu handhabendes Multitasks-Betriebssystem.
Es stellt dem Anwender 32 Tasks mit einer anwendungsbezogenen Programmiersprache zur Verfügung. Es wurden praxisnahe Befehle implementiert, um auch Programmierlaien, Maschineningenieuren und Betriebselektrikern die Möglichkeit zum Lesen und Ändern der Abläufe zu geben. Das System eignet sich hervorragend zum Programmieren von Abläufen, weniger zum sturen Verarbeiten von Verknüpfungen (SPS).

System: Das System selbst ist vollständig in Assembler für eine CPU der Familie NS32000 von National geschrieben. Der Anwender wird normalerweise nicht damit konfrontiert, ausser er will eigene zeitkritische Funktionen oder Interrupts selbst implementieren. Häufig werden solche kundenspezifische Funktionen wie Regelungen, Auswertungen usw. von INDEL AG realisiert und implementiert. Diese Funktion steht dem Kunden dann als REX-Aufruf oder neue Instruktion zur Verfügung.

ISM-Tasks: Die 32 Tasks werden quasiparallel abgearbeitet. Es wird immer ein Befehl pro Task verarbeitet und dann zum nächsten Task gewechselt. Pro Durchgang wird auch einmal das Assembler-Modul "USER-CPY" abgearbeitet, in dem anlagespezifische Funktionen wie z.B. eine elektronische Königswelle implementiert werden können.

Register: Jeder Task hat 128 eigene 16-Bit Register (R00..R7F), die aber auch als 32-Bit (R01,R00) oder 64-Bit (R03,R02,R01,R00) Register genutzt werden können. Davon sind 16 Register (R70..R7F) für System-Funktionen reserviert und fest belegt.

Befehle: Die Tasks werden in einer eigenen, Assembler ähnlichen Sprache programmiert. Die Befehle verfügen über symmetrische Adressierungsarten, welches bedeutet, dass es z.B. nur einen Move-Befehl gibt, der von überall nach überall Daten verschieben kann.

Timer: Das System stellt jedem Task zwei 16-Bit Timer innerhalb seiner Task-Register zur Verfügung: Ein 10ms-Timer und ein Sekunden-Timer.

Flags: Alle Tasks haben 256 gemeinsame Flags. Damit können Abläufe koordiniert und gesteuert werden. Die Flags 0..127 werden beim Einschalten immer gelöscht, die Flags 128..255 bleiben auch bei Spannungsausfall erhalten, sofern ein CRAM mit Batterie eingesetzt wird.

I/sec Die Systemleistung ISEC gibt an, wieviele Instruktionen pro Sekunde von jedem Task abgearbeitet werden (I/sec wird z.B. im INDEL-Debugger ID angezeigt). Daraus kann auch die maximale Reaktionszeit ermittelt werden.
Es ist klar, je weniger Task gestartet werden, um so schneller werden diese verarbeitet. Es lohnt sich daher, sich gegenseitig ausschliessende Abläufe in einem Task zu verarbeiten.

Beispiel

Vorgabe

Hardware: Zum Austesten des folgenden Beispiels benötigen Sie einen PC-Master (egal welche Version) mit einer EXT-IO Karte.

Eingänge:	0	RESET	TASK-0
	1	START	TASK-0
	2	STOP	TASK-0
	4+5	Lauflicht-Modus 0 ... 3	TASK-2
Ausgänge:	0	ALARM	TASK-0
	1	READY	TASK-0
	2	RUN	TASK-0
	4..7	Blinklicht	TASK-1
	8..15	Lauflicht	TASK-2

TASK-0: Initialisiert alles, verarbeitet die RESET, START und STOP-Tasten, startet und killt die Tasks 1+2 und steuert die ALARM, READY und RUN-Lampen an.

TASK-1: Blinkt einfach mit den Ausgängen 4 ... 7 hin und her.

TASK-2: Lässt das Lauflicht je nach Modus (Inp-4 und 5) wie folgt laufen:

00	links
01	rechts
10	addiert 1
11	subtrahiert 1

EQUAL: Da kein Linker für die Task-Programierung existiert, werden alle gemeinsamen Zuweisungen vorzugsweise in ein File (EQUAL) geschrieben. Der Assembler "MSI/O" erzeugt daraus neben dem Listing (EQUAL.LS) auch ein Symbol-File (EQUAL.SY), das beim Assemblieren der einzelnen Tasks dazu geladen werden kann. (Das EQUAL-File könnte auch als Include-File in jedem Task eingebunden werden.)

Files: Die folgenden Source-Files finden Sie auch in Ihrem PCMASTER-Verzeichnis unter PCMASTER\BEISPIEL\ISM\.

EQUAL

```
.TITLE  EQUAL-File für DEMO-Tasks

;*************************************************************************
;*                                                                       *
;*                    Gemeinsame Zuweisungen                             *
;*                      für die DEMO-Tasks                               *
;*                                                                       *
;*=======================================================================*
;*      Assemblieren:   MSI/O EQUAL     ; Erzeugt EQUAL.LS und .SL       *
;*************************************************************************
; Rev.  1.0  920515-FB  Grundversion                          INDEL AG

;************** Task Start-Adressen ***************************************
                .LOC    08000           ; Programm-Bereich Start
TASK_0:         .BLKW   0200            ; TASK-0, 0200 WORD Länge
TASK_1:         .BLKW   0100            ; TASK-0, 0100 WORD Länge
TASK_2:         .BLKW   0100            ; TASK-0, 0100 WORD Länge

;************** Hardware Adressen *****************************************
HW_DPR:         .EQU    0160000         ; DUAL-PORT RAM
HW_ADC:         .EQU    0080    ; {DPR} ; ADC-Karte  0
HW_POS:         .EQU    0280    ; {DPR} ; POSI-Karte 0
HW_PCR:         .EQU    0400    ; {DPR} ; Freies Übergabe-RAM an den PC/AT

;************** Gemeinsame Pointers ***************************************
DPR:            .EQU    1               ; Poi-1 zeigt auf DUAL-PORT RAM
PCR:            .EQU    2               ; Poi-2 zeigt auf PC-RAM
ADC:            .EQU    3               ; Poi-3 zeigt auf ADC-Karte 0
POS:            .EQU    4               ; Poi-4 zeigt auf POS-Karte 0

;************** PC-RAM Belegung *******************************************
                .LOC    0       ; {PCR} ;
STAT:           .BLKW   1               ; STATUS an PC/AT
  S_ALARM        =      1               ; 1 = ALARM
  S_READY        =      2               ; 2 = ALARM
  S_RUN          =      3               ; 3 = ALARM

WERT_1:         .BLKW   1               ; 16-BIT Übergabe
WERT_2:         .BLKD   1               ; 32-BIT Übergabe

;************** Eingänge **************************************************
I_RESET:        .EQU    0               ; RESET-Taste
I_START:        .EQU    1               ; START-Taste
I_STOP:         .EQU    2               ; STOP -Taste

I_MODE:         .EQU    4       ;+5     ; Lauflicht Mode 0..3
```

```
;************** Ausgänge ****************************************************
O_ALARM:        .EQU    0               ; ALARM-Lampe
O_READY:        .EQU    1               ; READY-Lampe
O_RUN:          .EQU    2               ; RUN  -Lampe

O_BLK0:         .EQU    4               ; Blinklicht 0
O_BLK1:         .EQU    5               ; Blinklicht 1
O_BLK2:         .EQU    6               ; Blinklicht 2
O_BLK3:         .EQU    7               ; Blinklicht 3

O_LAUF:         .EQU    8       ;..15   ; Lauflicht  0..7

;************** Flags *******************************************************
F_RUN:          .EQU    0               ; RUN-Flag

;************** EQUAL ENDE **************************************************
```

TASK0

```
.TITLE          **- Demo Task 0 -**
.SUBTITLE       Reset,Start,Stop

;*****************************************************************************
;*                                                                           *
;*                           Demo-Task 0                                     *
;*                        Reset, Start, Stop                                 *
;*                                                                           *
;*---------------------------------------------------------------------------*
;*      Assemblieren:   MSI TASK0 EQUAL         ; Erzeugt TASK0.LS und .HX   *
;*****************************************************************************
; Rev.  1.0  920515-FB  Grundversion                                INDEL AG

;-------------- Lokale Zuweisungen ---------------------------------------------
        .LOC    TASK_0                  ; TASK Start Adresse
TNr_1:  .EQU    R10                     ; Task-Nummer von Task-1
TNr_2:  .EQU    R12                     ; Task-Nummer von Task-2

;************** Grund-Initialisation ***************************************
;-------------- Lade gemeinsame Pointers für alle Tasks ------------------------
INIT:   MOVD    HW_DPR,2*DPR{0}         ; DUALPORT RAM
        ADDR    HW_ADC{DPR},2*ADC{0}    ; ADC-BASE
        ADDR    HW_POS{DPR},2*POS{0}    ; POS-BASE
        ADDR    HW_PCR{DPR},2*PCR{0}    ; PC/AT RAM BASE

;-------------- Lösche das DUALPORT RAM —---------------------------------------
        MOV     0,0400{PCR}             ; Lösche erste PC-RAM Zelle
        DUMP    0400{PCR},07FC,1{PCR}   ; Lösche ganzes PC-RAM

;************** Warte auf RESET ********************************************
W_RESET:SBIT    O_ALARM,OB              ; ALARM-Lampe ein
        MOV     S_ALARM,STAT{PCR}       ; STATUS = ALARM an PC/AT
        THT0    I_RESET,IB              ; Warte bis RESET-Taste gedrückt
        CBIT    O_ALARM,OB              ; ALARM-Lampe aus
        THT1    I_RESET,IB              ; Warte bis RESET-Taste losgelas

;-------------- Starte Task 1 und 2 --------------------------------------------
EXQ_1:  EXQ     TASK_1,TNr_1,EXQ_1      ; Starte TASK 1
EXQ_2:  EXQ     TASK_2,TNr_2,EXQ_2      ; Starte TASK 2

;************** Warte auf START ********************************************
READY:  SBIT    O_READY,OB              ; READY-Lampe ein
        MOV     S_READY,STAT{PCR}       ; STATUS = READY an PC/AT
W_START:TBR1    I_RESET,IB,T_RESET      ; RESET-Taste betätigt ?
        TBR0    I_START,IB,W_START      ; START-Taste betätigt ?
```

```
;-------------- START-Taste betätigt ----------------------------------------
T_START:CBIT    O_READY,OB                      ; READY-Lampe aus
        SBIT    O_RUN,OB                        ; START-Lampe ein
        THT1    I_START,IB                      ; Warte bis START-Taste losgelas

        SBIT    F_RUN,FB                        ; RUN-FLAG ein
        MOV     S_RUN,STAT{PCR}                 ; STATUS = RUN an PC/AT

;************** Blinke mit RUN-Lampe bis STOP *********************************
BLINK:  IBIT    O_RUN,OB                        ; Blinke mit RUN-Lampe
        MOV     50,TIM                          ; Lade TIM mit 500ms

W_STOP: TBR1    I_STOP,IB,T_STOP                ; STOP-Taste  betätigt ?
        TBR1    I_RESET,IB,T_RESET              ; RESET-Taste betätigt ?
        CBR     TIM,<>,0,W_STOP                 ; TIMER = 0
        BRA     BLINK

;———————— STOP-Taste betätigt ————————————————————————————————————————
T_STOP: CBIT    O_RUN,OB                        ; RUN-Lampe aus
        CBIT    F_RUN,FB                        ; RUN-FLAG aus
        BRA     READY                           ; Wir sind wieder Ready

;************** RESET-Taste betätigt ******************************************
T_RESET:CBIT    F_RUN,FB                        ; RUN-FLAG aus
        CBIT    O_RUN,OB                        ; RUN-Lampe aus
        CBIT    O_READY,OB                      ; READY-Lampe aus
        SBIT    O_ALARM,OB                      ; ALARM-Lampe ein
        MOV     S_ALARM,STAT{PCR}               ; STATUS = ALARM an PC/AT
        JOAB    TNr_1                           ; Task-1 Abort
        JOAB    TNr_2                           ; Task-2 Abort

        THT1    I_RESET,IB                      ; RESET-Taste noch betätigt ?
        DELAY   100                             ; Warte 1 Sekunde!
        CBIT    O_ALARM,OB                      ; ALARM-Lampe ein
        BRA     EXQ_1                           ; Restart Tasks

;************** TASK-0 ENDE ***************************************************
```

TASK1

```
.TITLE          **- Demo Task 1 -**
.SUBTITLE       Blinken

;*****************************************************************************
;*                                                                           *
;*                        Demo-Task 1                                        *
;*                     Blinke mit Out BLK0..3                                *
;*                                                                           *
;*****************************************************************************
;*      Assemblieren:   MSI TASK1 EQUAL         ; Erzeugt TASK1.LS und .HX   *
;*****************************************************************************
; Rev.  1.0  920515-FB  Grundversion                                INDEL AG

;———————— Lokale Zuweisungen ———————————————————————————————————————————————
        .LOC    TASK_1                          ; TASK Start Adresse

;************** Grund-Initialisation ***************************************
INIT:   MOV     ABORT,ABA                       ; Springe auf ABORT wenn JOAB
        SBIT    O_BLK0,OB                       ; BLK-Lampe 0 ein
        SBIT    O_BLK2,OB                       ; BLK-Lampe 2 ein

;************** RUN / STOP *************************************************
W_RUN:   THT0    F_RUN,FB                       ; Warte bis RUN
        IBIT    O_BLK0,OB                       ; Alle BLK-Ausgänge invertieren
        IBIT    O_BLK1,OB
        IBIT    O_BLK2,OB
        IBIT    O_BLK3,OB
        DELAY   20                              ; 20ms WARTEN
        BRA     W_RUN

;************** Task killen ************************************************
ABORT:  SBR     O_BLK0,OB,4,0                   ; Blinklicht aus
        JSKI                                    ; KILLE diesen Task

;************** TASK-1 ENDE ************************************************
```

TASK2

```
.TITLE          **- Demo Task 2 -**
.SUBTITLE       Lauflicht
;*****************************************************************************
;*                                                                           *
;*                        Demo-Task 2                                        *
;*                    Lauflicht mit Out LAUF0..7                             *
;*                                                                           *
;*****************************************************************************
;
;*      Assemblieren:   MSI TASK2 EQUAL         ; Erzeugt TASK2.LS und .HX   *
;*****************************************************************************
; Rev.  1.0  920515-FB  Grundversion                                INDEL AG

;------- Lokale Zuweisungen ----------------------------------------------------
        .LOC    TASK_2                          ; TASK Start Adresse
MODE:   .EQU    R10                             ; MODE 0..3
LICHT:  .EQU    R12                             ; Lauflicht Register

;************** Grund-Initialisation ***************************************
INIT:   MOV     ABORT,ABA                       ; Springe auf ABORT wenn JOAB
        MOV     0101,LICHT                      ; 2*8-Bit in 16-Bit Register
        SBR     O_LAUF,OB,8,LICHT               ; Setze 8 Ausgänge ab O_LAUF

;************** RUN / STOP *************************************************
W_RUN:  THT0    F_RUN,FB                        ; Warte bis RUN
;------- Mode 0..3 auswerten ---------------------------------------------------
RUN:    LBR     I_MODE,IB,2,MODE                ; Lese 2 INP ab I_MODE
        JSM     MODE@EXQ_TAB                    ; MODE 0..3 ausführen
        SBR     O_LAUF,OB,8,LICHT               ; Setze 8 Ausgänge ab O_LAUF
        DELAY   3                               ; 30ms warten!
        BRA     W_RUN                           ; noch RUN ?

EXQ_TAB:.WORD   LINKS                           ;0; Schiebe 1 links
        .WORD   RECHTS                          ;1; Schiebe 1 rechts
        .WORD   PLUS                            ;2; + 1
        .WORD   MINUS                           ;3; - 1

;============ Lauflicht Funktionen ===================================
LINKS:  ROT     1,LICHT                         ; Schiebe 1 links
        RTM     0
RECHTS: ROT     -1,LICHT                        ; Schiebe 1 rechts
        RTM     0
PLUS:   ADD     0101,LICHT                      ; + 1
        RTM     0
MINUS:  SUB     0101,LICHT                      ; - 1
        RTM     0

;************** Task killen ************************************************
ABORT:  SBR     O_LAUF,OB,8,0                   ; Lauflicht aus
        JSKI                                    ; KILLE diesen Task
;************** TASK-2 ENDE ************************************************
```

Inbetriebnahme

MSI: Um die Files zu assemblieren, geben Sie folgendes ein:

```
MSI/O   EQUAL           ; erzeugt EQUAL.LS, .HX und .SY
MSI/O   TASK0   EQUAL   ; erzeugt TAKS0.LS, .HX und .SY
MSI/O   TASK1   EQUAL   ; erzeugt TAKS1.LS, .HX und .SY
MSI/O   TASK2   EQUAL   ; erzeugt TAKS2.LS, .HX und .SY
```

CONFIG: Der PC-Master braucht ein Konfigurations-File, damit er sich auf die angeschlossenen Peripherie-Karten einstellen kann (siehe PC-Master Dokumentation). Ein solches File wird erstellt, wenn Sie CONFIG eingeben, die Anzahl der IO-Karten z.B. auf 2 setzen und unter DEMO.PCM abspeichern.

INDEL.INI: Um die Tasks zu testen, muss erst das Projekt-File INDEL.INI erstellt oder angepasst werden (siehe auch Kap. TOOLS). Es wird von TRANS, SHOW und ID benötigt und sieht etwa wie folgt aus:

```
[Target]
System=PCMaster

[PCMaster]
Address=CB00
ConfigFile=DEMO.PCM
WarmBoot=no

[Trans]
Systemsoftware=..\..\PCM.HEX
DownLoad=yes
AutoStart=yes
FloatingPointUnit=no

[ProjectFiles]
TASK0=0
TASK1=0
TASK2=0
```

TRANS: Jetzt können das Betriebssystem und die Tasks mit TRANS in den PC-Master geladen werden. Da im File INDEL.INI AutoStart=yes steht, wird der Task-0 (meist auch als Monitortask bezeichnet) automatisch gestartet. Die andern Tasks werden von diesem erst gestartet, wenn die RESET-Taste (INP-0) betätigt wird (siehe Listing TASK0).

SHOW, ID:	Mit dem SHOW-Programm können Sie die Funktion der IO-Karte testen und überwachen. Der INDEL-Debugger ID dient der Inbetriebnahme und Fehlersuche der Tasks auf Source-Level.

TOOLS

INDEL.INI

FILE.INI — Alle Hilfsprogramme der INDEL AG beziehen ihre Konfigurationsdaten aus einer zentralen '.INI' - Datei, deren Name beim Aufruf des Programmes als Parameter übergeben werden kann.

z.B. TRANS MyIni.ini

INDEL.INI — Wird kein Name als Parameter angegeben, suchen alle Hilfsprogramme nach der Konfigurationsdatei INDEL.INI im aktuellen Verzeichnis.

Der Aufbau einer solchen Datei lehnt sich an die von Windows bekannten '.INI'-Datei Strukturen an. Einer Überschrift (Applicationname) folgen sogenannte Schlüsselworte (Keynames), welche die einzelnen Konfigurationspunkte beschreiben :

```
[Application1]
Keyname1=...
Keyname2=...

[Application2]
Keyname1=...
...
```

Es folgt eine Beschreibung der einzelnen Einträge :

[Target]

System= Definiert das zu behandelnde Zielsystem.
- PCMASTER - das Zielsystem ist ein PC-Master
- IPS-32 - das Zielsystem ist ein Indel 19"-Rack

Default : PCMASTER

[PCMaster]

Address= Angabe der Adresse, auf der sich der PC-Master befindet (Drehschalterwerte), z.B. CA00.
Default : D000

ConfigFile= Name und Pfad der DualportRAM-Konfigurationsdatei, die mit Hilfe von CONFIG.EXE erstellt wurde, z.B. c:\Project\test.pcm.
Default : CONFIG.PCM

WarmBoot=
- NO - das Zielsystem wird in jedem Fall zuerst initialisiert und anschliessend mit Software befruchtet
- YES - das Zielsystem wird nur dann initialisiert und mit Software befruchtet, wenn es nicht bereits läuft oder den Geist aufgegeben hat

Default : NO

EnableTime= NO - Evtl. gebrauchte Time-Befehle liefern ein falsches Ergebnis
YES - Im PCMaster stehen PC-Zeit und -Datum über die Standard-Time-Befehle zur Verfügung.

Hinweis : Diese Option bezieht sich immer auf alle im PC installierten PCMaster. Der TSR-Treiber holt sich die jeweiligen Adressen von SET PCMASTER = ... Eintrag in Autoexec.bat.

FloatingPointValues=
NO - die Werte im DPR werden im üblichen Festkommaformat dargestellt

YES - die Werte werden im Fliesskommaformat (floating point) dargestellt (diese Option steht nur in Verbindung mit einem INFO-Master zur Verfügung)

[IPS-32]

Baudrate= Baudrate für den Datentransfer PC → IPS-32 Rack

2400	2400 Baud	(Modem)
9600	9600 Baud	(Modem)
19200	19200 Baud	HST-Modem
38400	38400 Baud	Direkte Verbindung PC/AT → IPS-32 Rack

DataBits= Anzahl Data-Bits pro BYTE.

7	7-Bit
8	8-Bit

Stop-Bits=

1	1 Stop-Bit
2	2 Stop-Bit

Parity=

no	no Parity
even	even Parity
odd	odd Parity

Retries= Anzahl Versuche bei Übertragungsfehler bis Bildschirm-Meldung.

5	5 Versuche

Timeout= Wartezeit in ms bis Retry. Normalerweise wird dieser Eintrag nicht benötigt, da die optimale Timeoutzeit auf Grund der aktuellen Baudrate berechnet wird.

SlaveNumber= Slave-Nummer von IPS-32 Rack

1	Slave-Nummer 1

Port=

PC/AT	Schnittstellen Nummer
COM1	erste Schnittstelle
COM2	zweite Schnittstelle

[Trans]

SystemSoftware= Name und Pfad der Systemsoftware, z.B. c:\pcmaster\pcm.hex
Default : PCM.HEX

SystemOffset= Hier kann ein Downloadoffset angegeben werden (nur bei Target=IPS-32). Der Offset wird als Wortadresse in Hex angegeben.
Default : 0

SystemDownload= NO - die Systemsoftware wird nicht ins Zielsystem geladen
YES - die Systemsoftware wird ins Zielsystem geladen
Default : YES

SystemVerify= NO - es findet kein Vergleich zwischen Source und Zielcode statt
YES - Source und Zielcode werden miteinander verglichen und evtl. Fehler angezeigt.
Default : NO

SystemAutostart= NO - das Betriebssystem wird gestartet und sogleich auf HALT gesetzt (für eingefleischte Indel Freaks entspricht dies dem 'Init-Halt' mit dem Utility)
YES - das Betriebssystem wird normal gestartet
Default : YES

DownLoad= NO - die evtl. unter [ProjectFiles] angegebenen Dateien werden nicht automatisch ins Zielsystem geladen
YES - die evtl. unter [ProjectFiles] angegebenen Dateien werden ins Zielsystem geladen
Default : NO

Verify= NO - es findet kein Vergleich zwischen Source und Zielcode statt
YES - Source und Zielcode werden miteinander verglichen und evtl. Fehler angezeigt.
Default : NO

Autostart= NO - der Monitortask wird gestartet und sogleich auf HALT gesetzt
YES - der Monitortask wird gestartet
Default : NO

FloatingPointUnit= NO - das Zielsystem besitzt keine Floatingpointunit
YES - das Zielsystem besitzt eine Floatingpointunit
Default : NO

[Show]

ScreenMode=
- 2 - 25 Zeilen, Schwarzweissmodus auf Farbadapter
- 3 - 25 Zeilen, Farbmodus
- 7 - 25 Zeilen, Monochromer Modus
- 258 - 43/50 Zeilen, Schwarzweissmodus auf Farbadapter
- 259 - 43/50 Zeilen, Farbmodus
- 263 - 43/50 Zeilen, Monochromer Modus

[Debug]

ScreenMode= siehe [Show]

RefreshRate= Display-Refresh-Rate in sec
Default : 1

TabSize= Tabulatorzeichen (09) werden in Dateien zu TABSIZE Leerzeichen erweitert.
Default : 8

maxInputs= Hiermit kann die maximale Anzahl Eingänge eingetragen werden, die ein 'Inputs'-Fenster verwalten soll. Die Zahl wird vom Debugger auf ein Vielfaches von 16 gerundet und kann 4096 nicht überschreiten.
Default: 256

maxOutputs= Wie 'maxInputs' aber für die Ausgänge.

maxFlags= Wie 'maxInputs' aber für die Flags.

AutoTaskWndClose=
- YES - das Fenster eines nicht mehr existierenden Tasks wird automatisch gelöscht
- NO - kein automatisches Löschen
- Default : YES

WatchCaseSensitiv=
- YES - bei den Watches wird auf Gross-/Kleinschreibung geachtet
- NO - keine Gross/Kleinunterscheidung
- Default : NO

SourceFileTrace=
- YES - es wird immer automatisch das aktuelle Listing angezeigt (z.B. bei einem SingleStep in ein anderes File)
- NO - ein Listingwechsel muss von Hand durchgeführt werden (mit 'View' → 'Task source')

Verify=	YES	- Bei Runterladen von Projektdateien findet ein Vergleich zwischen Source- und Zielcode statt. Eventuelle Differenzen werden angezeigt.
	NO	- Bei Runterladen von Projektdateien findet kein Vergleich zwischen Source- und Zielcode statt.
MemoryFilename=		- Ab ID Rev. 1.32 ist es möglich, einen MemoryDump direkt in ein File zu schreiben. MemoryDump öffnen (→ ALT-F10 → W). Der Default-Filename kann hier angegeben werden.
NumberOfValues=		- Ab ID Rev. 1.32 ist es möglich, einen MemoryDump direkt in ein File zu schreiben. MemoryDump öffnen (→ ALT-F10 → W). Die Default-Anzahl kann hier angegeben werden.

+

[ProjectFiles]

FILE1= Hier werden die Projektdateien eingetragen, die von TRANS.EXE ins Zielsystem geladen werden, bzw. die dem Debugger bekannt sein sollten.

Hinter '=' kann ein Downloadoffset (in hex) angegeben werden, da mit dem ISM-Compiler MSI.EXE nur Compilate im Adressbereich von 0..FFFF erzeugt werden können.

0	- Das File wird in den Bereich 00'0000 ... 00'FFFF geladen.
10000	- Das File wird in den Bereich 01'0000 ... 01'FFFF geladen.
Default :	0

MSI

MSI [/O] [/S] [/F] [/L] [/I] Sourcefile [Symbolfile]

MSI.EXE Der Assembler für das ISM-5.0 Betriebssystem kennt folgende Switches:

/O Erzeugt ein Symbolfile NAME.SY
Alle Zuweisungen des ersten Files können damit beim Assemblieren der weiteren Files weiter verwendet werden.
Dieses File wird vom INDEL-DEBUGGER "ID" benötigt, um Watches zu setzten.

/S Erzeugt eine sortierte Liste aller Symbole im Listing-File NAME.LS.

/F Ab der Rev. ISM-5.0 können die Bitbefehle mit den Adressierungsarten
Immediate,IB , Immediate,OB , Immediate,FB
schneller ausgeführt werden, wenn beim Assemblieren /F angegeben wird. Die Befehle werden dann der neuen 17'er Befehlsgruppe zugeordnet, bei der nur die oben genannten Addressierungsarten möglich sind, die dafür aber sehr schnell ausgeführt werden können.

/L Das Debugflag /L zeigt das Listing aller Passes auf dem Bildschirm an und dient nur der Fehlersuche bei unerklärlichen Passerrors.

/I Passes und Includefiles werden beim Assemblieren angezeigt.

FILES:

NAME	SOURCE-File
NAME.LS	LISTING-File
NAME.SY	SYMBOL-File
NAME.HX	CODE-File

Beispiel: Die Maschine hat ein gemeinsames EQUAL-File, ein gemeinsames Textfile DTEXT und drei Tasks 0..2:

```
MSI /O      EQUAL
```

Erzeugt EQUAL.LS und EQUAL.SY . Das File EQUAL.HX wird nicht gebraucht, wenn es keine Tabellen enthält, die Code erzeugen.

```
MSI /O      DTEXT       EQUAL
```

Übernimmt die Zuweisungen vom Symbolfile EQUAL.SY, assembliert den Text in DTEXT und erzeugt nebem dem .LS und .HX File auch das neue Symbolfile DTEXT.SY, welches alle Zuweisungen von EQUAL und die Startadressen der Texte enthält. Die drei Task-Files können jetzt alle Zuweisungen von EQUAL und alle Texte von DTEXT verwenden.

```
MSI         TASK0       DTEXT
MSI         TASK1       DTEXT
```

MSI TASK2 DTEXT

TRANS

TRANS [IniFile.INI]

TRANS.EXE Dieses Programm erlaubt das Laden der Betriebssoftware und der ISM-5.0 Tasks in das Zielsystem PC-Master oder IPS-32 Rack.

INDEL.INI Das Trans-Programm benötigt ein .INI File, in dem alle Angaben über das Target-System und die Projekt-Files stehen. Wird kein spezielles IniFile.INI angegeben, sucht TRANS automatisch nach INDEL.INI im lokalen Direktory.

Keynames: TRANS sucht nach den folgenden Keynames in INDEL.INI:

[Target]
[PCMaster] oder [IPS-32]
[Trans]
[ProjectFiles]

Eine genaue Beschreibung der Einträge finden Sie unter INDEL.INI am Anfang von diesem Kapitel.

FILES:

ConfigFile.PCM Wird mit dem PC-Master gearbeitet, so braucht TRANS die mit CONFIG erstellte Dualport-Ram-Konfigurationsdatei ConfigFile.PCM. TRANS findet dieses File über einen Eintrag in [PCMaster].

System.HEX TRANS findet das zu ladende Betriebssystem System.HEX über einen Eintrag in [TRANS].

Tasks.HX Sofern vorhanden werden die zu ladenden ISM-5.0 Task-Programme Tasks.HX unter [ProjectFiles] eingetragen.

ID

ID [IniFile.INI]

ID.EXE Der INDEL-DEBUGGER ID erlaubt das Debuggen der ISM-5.0 Tasks in den Zielsystemen PC-Master oder IPS-32 Rack. Es kann mit mehreren Tasks ohne gegenseitige Beeinflussung gleichzeitig gearbeitet werden.

INDEL.INI Das ID-Programm benötigt ein .INI File, in dem alle Angaben über das Target-System und die Projekt-Files stehen. Wird kein spezielles IniFile.INI angegeben, sucht ID automatisch nach INDEL.INI im lokalen Direktory.

Keynames: ID sucht nach den folgenden Keynames in INDEL.INI:

[Target]
[PCMaster] oder [IPS-32]
[Debug]
[ProjectFiles]

Eine genaue Beschreibung der Einträge finden Sie unter INDEL.INI am Anfang von diesem Kapitel.

FILES:

Tasks.HX Beim Starten sucht ID alle Task-Files Tasks.HX, die unter [ProjectFiles] eingetragen wurden und erstellt ein .MAP File, in dem alle Start- und Endadressen dieser Programme eingetragen werden. Damit kann der ID jedem Task jederzeit das entsprechende Listing zuordnen und anzeigen.

Tasks.LS Die Tasks.LS Files werden für das Source-Level Debugging benötigt.

Tasks.SY Wird ein Watch-Fenster geöffnet, so benötigt der Debugger das entsprechende Symbolfile.

CONFIG

CONFIG [ConfigFile.PCM]

CONFIG.EXE Mit dem CONFIG-Programm wird die Dualport-RAM Konfigurationsdatei erstellt. Damit kennt der PC-Master (PC/AT) oder Master-32 (IPS-32) alle angeschlossenen Peripheriekarten und deren Betriebsmodi.

PC-Master

ConfigFile.PCM Diese Datei wird von TRANS beim Starten ins PC-Master Dualport-RAM geschrieben.

IPS-32

MASx.INC Das Betriebssystem für das IPS-32 Rack benötigt zum Betreiben jeder MASTER-32 Karte ebenfalls eine Dualport-RAM Konfigurationsdatei mit den Namen MAS1.INC bis MAS3.INC. Diese Dateien im .BYTE-Format werden am Ende vom File IOMAS32.32K angehängt.

CONVERT.EXE Das Programm CONVERT wandelt eine ConfigFile.PCM Datei in eine .BYT Datei ConfigFile.INC:

CONVERT ConfigFile

RAM-AUFTEILUNG

PC-MASTER RAM

WORD-ADR	
00'0000.. 00'7FFF	System-Programm
00'8000.. 01'BFFF	User-CRAM wenn 1MB Ram-Chips bestückt
00'8000.. 07'BFFF	User-CRAM wenn 4MB Ram-Chips bestückt
07'C000.. 07'FFFF	System-Ram und Task-Register
16'0000.. 16'03FF	Dualport Ram zu PC/AT FirmWare
16'0400.. 16'07FE	Dualport Ram zu PC/AT User Bereich

INFO-MASTER RAM

WORD ADR	
00'0000.. 00'7FFF	System-Programm
00'8000.. 01'BFFF	User-CRAM wenn 1MB Ram-Chips bestückt
00'8000.. 07'BFFF	User-CRAM wenn 4MB Ram-Chips bestückt
07'C000.. 07'FFFF	System-Ram und Task-Register
16'0000.. 16'03FF	Dualport Ram zu PC/AT FirmWare
16'0400.. 16'07FE	Dualport Ram zu PC/AT User Bereich

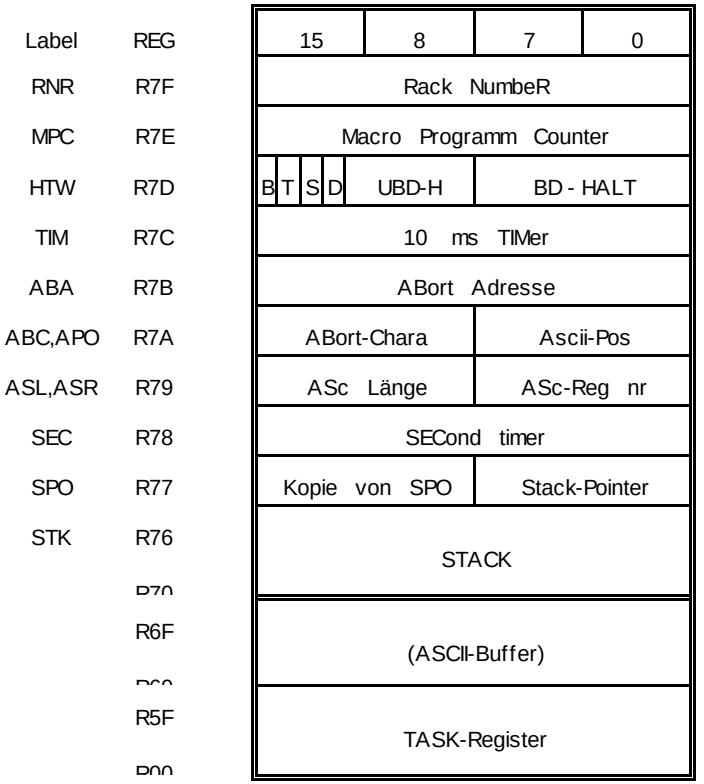
INFO-Interface

40'07FE	INFO-Mask
40'07FF	INFO-Status
40'0800	Card-IRQ-Vector
40'0801.. 40'08FF	Job Tabelle
40'0900.. 40'09FF	Rec Adresse
40'0A00.. 40'0AFF	Rec Data Bit 0..15
40'0B00.. 40'0BFF	Rec Data Bit 16..32
40'0C00.. 40'0CFF	Spez Trans Adresse
40'0D00.. 40'0DFF	Trans Adresse
40'0E00.. 40'0EFF	Trans Data Bit 0..15
40'0F00..	Trans Data Bit 16..32

40'0FFF

REGISTER

Task Register



R70..R7F: Die Register R70..R7F sind SYSTEM-REGISTER und fest zugeordnet. Man kann sie wie jedes andere Register ansprechen (zB. R7E) oder mit ihrem Namen (zB. MPC).

R60..R6F: Die Register R60..R6F werden bei Video- und ASCII-Befehlen als ASCII-Buffer belegt (Standard-Belegung nach SETD). Werden keine solche Operationen durchgeführt, können diese Register ganz normal belegt werden.

R00..R5F: Die Register R00..R5F sind die Task-Arbeitsregister.

Task-Kontroll Register

RNR,MPC: Die beiden Register RNR,MPC bilden zusammen den 32-Bit Macro-Programm-Counter.

HTW: Das Haltwort HTW enthält 8 bedingte und 7 unbedingte HALT-Bits.

BD	B0..B7	Bedingt HALT, nur wenn B15=0 ist
UBD	B8..B11	Unbedingt HALT
D	B12	Belegt von DEBUG
S	B13	Belegt von S-I/O
T	B14	Belegt von Timer (DELAY)
B	B15	Haltsperr-Bit für B0..B7

TIM: Das Timer-Register wird vom System alle 10 msec um 1 dekrementiert, bis es 0000 ist. Es kann mit beliebigen Befehlen angesprochen werden, wird aber auch vom DELAY-Befehl benutzt.

SEC: Das SEC-Register wird vom System jede Sekunde um 1 dekrementiert, bis es 0000 ist.

ABA: Im Register ABA wird eine Adresse gespeichert, auf die der Johann bei einem Abort springt. Ist sie 0000 wird der Task bei einem Abort gekillt und alle gebrauchten Devices (VIDEO,SIO) freigegeben. Ist sie nicht 0000 springt der Johann auf (ABA). Die Rack-Nummer RNR kann dabei nicht verlassen werden! SPO und HTW werden dabei zurückgesetzt, die Devices bleiben reserviert.

ABORT: * Die Adresse vom abgebrochenen Befehl kann mit RTM 255 auf das Stack geholt werden, um dann mit RTM 0 auf den Befehl zurück zu springen (Retry).

SPO: Der Stackpointer SPO (Lower-Byte) zeigt die STACK-Tiefe an. Der Stack ist leer, wenn er 00. Bei 0FF (-1) ist der erste Platz belegt und so weiter. Er wird von JSR, BSR automatisch bedient und ist für den Anwender nur in Sonderfällen interessant.

```
Kill Stack:        MOV        0,SPO
```

Bei jedem ABORT wird eine Kopie vom Lower- to Higher-Byte gemacht und das niedere Byte = 00 gesetzt. Damit ist der Stack grundsätzlich gelöscht, kann aber mit MHLB SPO,SPO wieder rekonstruiert werden (Abort in einer Subroutine, in die man zurück möchte).

STK: Die Register R76..R70 bilden den eigentlichen STACK. R76 ist der erste, R75 der zweite Stackplatz u.s.w.

VORSICHT: Die Stacktiefe ist nicht begrenzt !

* Ab System Rev. 5.11

ASCII-Kontroll Register

ABC: Dies ist das höhere Byte in R7A und muss mit speziellem MOVE geladen werden (zB. MLHB "A",ABC).
Wird dieser Charakter bei TIP oder TOP auf dem Keyboard gedrückt, führt dies zu einem Abort (Der Task springt auf die ABORT-Adresse).
ABC wird mit INID oder SETD auf 01B (ESC) gesetzt.

APO: Dies ist das niedere Byte in R7A und muss mit speziellem MOVE geladen werden (zB. MLLB 0,APO).
Dieses Register wird automatisch von TIP, RTIP, ABR und ACMP bedient und ist für den Anwender nur in Sonderfällen interessant.
APO wird mit INID oder SETD auf 00 gesetzt.
Bei Abort (z. B. bei SETD, Floppy-Befehlen usw.) wird in APO eine Fehlernummer übergeben, aus der die genaue Abort-Ursache hervorgeht.

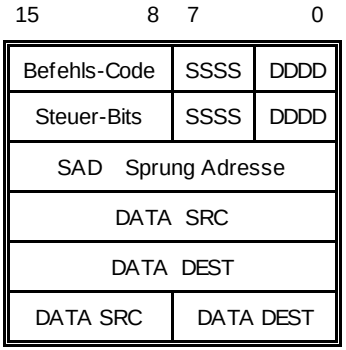
ASL: Dies ist das höhere Byte in R79 und muss mit speziellem MOVE geladen werden (zB. MLHB 01F,ASL).
ASL ist die maximale Anzahl Zeichen, die mit TIP eingelesen werden. Man kann damit die Grösse des Eingabe-Fensters in einer Bildschirm-Maske begrenzen oder den ASCII-Buffer länger oder kürzer machen (01F Zeichen == 010 REG). ASL wirkt auch wenn beim TIP der Text nicht im ASCII-Buffer gespeichert wird, sondern an einem beliebigen RAM-Buffer.
ASL wird mit INID oder SETD auf 01F gesetzt.

ASR: Dies ist das niedere Byte in R79 und muss mit speziellem MOVE geladen werden (zB. MLLB 060,ASR).
Im ASR steht das Register, bei dem der, mit ASC adressierte, ASCII-Buffer beginnt. Man kann damit den ASCII-Buffer in einen beliebigen Register-Bereich legen.
ASR wird mit INID oder SETD auf 060 gesetzt.

ASC: Ist eine eigene Adressierungsart! (Erzeugt keine Reg Nr.)
Mit ASC wird der ASCII-Buffer adressiert. Der Anfang des ASCII- Buffers steht in ASR, die Länge in ASL.

ADRESSIERUNGSARTEN

Befehlsaufbau



Befehls-Kopf

Bei mehr als 2 Argumenten

Bei bedingten Sprüngen

B :B,Rxx,(Rxx),[Rxx]

BBBB = Befehls-Code
SSSS = Adressierungsart SRC
DDDD = Adressierungsart DEST
SAD = Sprung Adresse (LABEL oder Adresse)
Die Steuerbits werden bei Text-Befehlen gebraucht (z.B. CR/LF)
oder als Befehlscode-Erweiterung (z.B. Floppy-Befehle)

Beispiele:

```
20C3    MOV     033,R44
3344

2008    MOV     03333,@4444
3333
4444

60C3    CBR     033,=,R44,SAD     ; SAD = 05555
5555
3344

9033    TOP     R5E,R5F,ASC' ,CRLF
02A0
5E5F

9F01    ARC     04F0C,100:200,R33,(R44),REL+C
0834
0500
4F0C
0064
00C8
3344
```

Adressierungsarten

SRC	DEST	DATEN		ADRESSIERUNGS ART
0	0	WWWW WWWW	WWWW WWWW	WORD :W
1	1	LLLL LLLL	LLLL LLLL	D_WORD :D:F
		HHHH HHHH	HHHH HHHH	INT / FLOAT
2	2		QQQQ QQQQ	BYTE :B
3	3		0RRR RRRR	REG R00..R7F
3	3		1xxx xxxx	NOT USED
4	4		0RRR RRRR	(REG)
4	4		1RRR RRRR	[REG]
5	5	0000 0000 :	0RRR RRRR	OFF(REG)
5	5	0000 0000 :	1RRR RRRR	OFF[REG]
6	6	0000 0000 :	0BBB BBBB	OREG(BREG)
6	6	0000 0000 :	1BBB BBBB	OREG[BREG]
6	6	1NNN NNNN :	0RRR RRRR	(REG)N
6	6	1NNN NNNN :	1RRR RRRR	[REG]N
7	7	0000 0000 :	0RRR RRRR	REG@ADRE
7	7	0000 0000 :	1RRR RRRR	REG@@ADRE
8	8	AAAA AAAA		@ADRE
9	9	0000 0000		OFF{POI}
9	9	1000 0000		OFF@{POI}
A	A	keine		ASC
B	B	MANTISSE		DOUBLE
		MANTISSE		-PRECISION
		MANTISSE		-FLOATING
		S: EXPONT :MANT		-POINT
C		QQQQ QQQQ		BYTE
D		0RRR RRRR		REG
D		1		NOT USED
E		0RRR RRRR		(REG)
E		1RRR RRRR		[REG]
	C	keine		IB
	D	keine		OB
	E	keine		FB
F	F			NOT USED

xxx

Immediate

2/C	xx[Exx][:B]
0	xxxx[Exx][:W]
1	xxxxxxxx[Exx][:D]

Erklärung: Es kann im Befehl einfach eine Zahl angegeben werden.
Sofern das Format mit :B, :W oder :D nicht erzwungen wird, legt der MSI-Assembler die Werte im Befehl wie folgt ab:

BYTE:	0	...	127	000000000	...	00000007F
	-128	...	-1	0FFFFFF80	...	0FFFFFFFF
WORD:	128	...	65535	000000080	...	00000FFFF
	-32768	...	-129	0FFFF8000	...	0FFFFFF7F
DOUBLE-WORD:	-2147483648	...	4294967294	000010000	...	0FFFFFFFF
	65536	...	-32769	080000000	...	0FFFF7FFF

Das System expandiert BYTE und WORD-Angaben im Befehl immer mit Vorzeichen auf DOUBLE und führt erst dann die Operation aus!

Vorsicht: Der Assembler wechselt bei Werten > 07F automatisch von BYTE auf WORD, jedoch nicht bei Werten > 08000 von WORD auf DOUBLE! (Am meisten werden BYTE und WORD Operationen durchgeführt!) Bei DOUBLE-Instruktionen kann das zu Fehler führen:

Beispiele:	MOV	0A0,R10	; R10	=		00A0	richtig!
	MOV	0A000,R10	; R10	=		A000	richtig!
	MOVD	0A000,R10	; R11,10	=	FFFF'	A000	falsch??
	MZWD	0A000,R10	; R11,10	=	0000'	A00	richtig!
	MOVD	0A000:D,R10	; R11,10	=	0000'	A000	richtig!
	MXWD	0A000,R10	; R11,10	=	FFFF'	A000	gewollt!
	MXWD	0A000:D,R10	; R11,10	=	FFFF'	A000	gewollt!
	MOV	01E4,R10	; R10	=	2710 =		10000

xxx.xx

FLOATING POINT Immediate

1	xxx.xx[Exx][:F]
B	xxx.xx[Exx][:L]

Erklärung: Wird eine Zahl mit Dezimalpunkt geschrieben, so setzt der Assembler automatisch eine Floating Point Zahl ein (sofern im Befehl zugelassen!).

SINGE-PREC:	-3.4028235E-38	...	3.4028235E38
DOUBLE-PREC:	-2.225073858507201E-308	...	2.225073858507201E308

Beachte: Der Befehl selbst bestimmt, ob SINGLE oder DOUBLE PRECISION Zahlen eingesetzt w erden müssen. Die Angaben :F und :L haben keinen Einfluss und können w eggelassen w erden!

Vorsicht: Der MSI-Assembler für PC/AT kann nur Exponenten bis E38 verarbeiten!

Beispiele:

```
MOVF    1.2E3,R00       ; SINGLE PRECISION
MOVL    1.2E3,R00       ; DOUBLE PRECISION

.FLOAT  -1.2E-3         ; SINGLE PRECISION
.LONG   1.2E3           ; DOUBLE PRECISION
```

@ADR

	Adresse
8	@ADR

Erklärung: Zeigt auf eine Adresse im lokalen (64K) RACK-Bereich.
ADR = 0000...0FFFF

Beachte: Diese Adressierungsart wird vorteilhaft nur innerhalb eines Listings verwendet (@LABEL). Adressen ausserhalb des lokalen 64K-Bereiches sind mit den Adressierungsarten REGISTER-INDEXED und POINTER-INDEXED anzusprechen!

Beispiel:

```
        TOP     DEV,POS,@TEXT
        ...
TEXT:   .TXT    "INDEL AG"
```

REG@ADR

Adresse mit Register-Offset

7 REG@ADR

Erklärung: Innerhalb einer Tabelle auf ADR wird auf den Wert gezeigt, der in REG steht.

Beachte: Die Tabelle muss unmittelbar in der Nähe des Befehls sein!
‼ ADRE muss im Bereich ±127. von MPC sein ‼

Beispiel: R11 = 0003

```
        MOV     R11@ATAB,R66        ; R66 = 03333
        ...
ATAB:   .WORD   0000,01111,02222,03333,04444,...
```

REG@@ADR

Indirekt (Adresse mit Register-Offset)

7 REG@@ADR

Erklärung: Innerhalb einer Tabelle auf ADR wird auf eine Adresse gezeigt, die in REG steht. Diese Adresse wird vom Befehl angesprochen.

Beachte: Die Tabelle muss unmittelbar in der Nähe des Befehls sein!
‼ ADRE muss im Bereich ±127. von MPC sein ‼

OFFSET-REG: Das Offset-Register enthält immer einen 16-Bit Offset mit Vorzeichen (-32768 ... 0 ... +32767).

Beispiel: R11 = 0002

```
        MOV     R11@@ATAB,R66       ; R66 = 01234
        ...

ATAB:   .WORD   01000,02000,ADRE,03000,...
        ...

ADRE:   .WORD   01234
```

OFF{POI}

Pointer indexed

9 OFF{POI}

Erklärung: Alle Task haben 12 Pointer gemeinsam (Pointer 0..11) und jeder Task hat 4 eigene, lokale Pointer (Pointer 12..15). Ein solcher Pointer enthält immer eine 32-Bit (Basis-)Adresse. Relativ zu diesen Pointern kann nun mit festen Offsets ein Datenelement angesprochen w erden.

Beachte: Der Offset ist immer positiv und muss im Bereich 000 ... 07FF sein.

Pointer Laden: Damit die Pointer selbst geladen w erden können, zeigt der Pointer-0 nach dem Hochstarten immer auf die gemeinsame Pointer-Tabelle und nach dem Starten eines Tasks (EXQ..) zeigt der Pointer-12 auf sich selbst. Dadurch ist es möglich, erst die andern Pointer zu laden und bei Bedarf auch den Pointer-0 bzw Pointer-12 neu zu belegen.

Beispiel: Lade den Pointer 4 mit der Basis 01' A000 und schreibe dann auf den 16' ten Platz dieses Daten-Bereiches den Wert 01234:
(Die Adresse von Pointer-4 = 8{0} , da Double-Word Einträge!)

```
MOVD    01A000,2*4{0}   ; Pointer-4       = 01' A000
MOV     01234,16{4}     ; ADR 01' A010    = 01234
```

OFF@{POI}

Indirekt (Pointer indexed)

9 OFF@{POI}

Erklärung: Alle Task haben 12 Pointer gemeinsam (Pointer 0..11) und jeder Task hat 4 eigene, lokale Pointer (Pointer 12..15). Ein solcher Pointer enthält immer eine 32-Bit (Basis-)Adresse. Relativ zu diesen Pointern kann nun mit festen Offsets auf eine Adresse gezeigt werden, über die ein Datenelement angesprochen wird.

Die WORD-Adresse auf OFF@{POI} bezieht sich auf das Rack, in dem sich die Adress-Tabelle befindet!

Beachte: Der Offset ist immer positiv und muss im Bereich 000 ... 07FF sein.

Pointer Laden: Siehe OFF{POI}

Diese Adressierung dient zum Beispiel der indirekten Textausgabe über Text-Tabellen. Der Text kann sich dabei in einem beliebigen (64k)RACK-Bereich befinden. Durch Umladen des Text-Pointers kann die ganze Maschine auch auf eine andere Landessprache umgestellt werden.

Beispiel:

```
TEXT    = 5                     ; POINTER-Belegung
ADDR    @TTAB,2*TEXT{0}         ; Anwahl der TEXT-Tabelle
...

TOP     DEV,POS,0@{TEXT},PCR    ; Anzeige = INDEL AG
TOP     DEV,POS,3@{TEXT}        ; Anzeige = CH-8308 ILLNAU
...

TTAB:   .WORD   TXT0,TXT1,TXT2,TXT3     ; TEXT-Tabelle

TXT0:   .TXT    'INDEL AG'
TXT1:   .TXT    'Industrielle Elektronik'
TXT2:   .TXT    ' Länggstrasse 17'
TXT3:   .TXT    ' CH-8308 ILLNAU'
```

REG

Register

3/D REG

Erklärung: Jeder Task hat 128 Register (R00..R7F), die damit angesprochen werden. Die Register R70..R7F können auch mit deren Namen angesprochen werden (siehe auch SYSTEM-REGISTER).

Beachte: Bei DOUBLE-WORD Zugriffen werden immer zwei, bei LONG- FLOATING immer vier Register hintereinander angesprochen!

```
Beispiel:       MOV     ABORT,ABA       ; LADE ABA MIT DER ADR ABORT
                MOV     1300,TIM        ; LADE DEN TIMER MIT 1.3 SEC
          ABORT: MOVD   012345678,R10   ; R11 = 01234 , R10 = 05678
```

OFF[REG]

Register indexed (mit Offset)

4/E	(REG)
5	OFF(REG)
4/E	[REG]
5	OFF[REG]

Erklärung: Das Register (Rxx) enthält eine Adresse, die (mit Offset) angesprochen wird.

(REG) Mit runden Klammern (Rxx) enthält das Register eine 16-Bit Adresse im gleichen (64k)RACK-Bereich wie der Befehl.

[REG] Mit eckigen Klammern [Rxx] enthält das Register eine 32-Bit Adresse.

OFFSET: Vor der Klammer kann ein Offset von maximal -128 ... +127 zu dieser Adresse angegeben werden.

Beispiel:

```
        MOV     TAB,R11     ; R11       = TAB-ADR
        ...

        MOV     3(R11),R66  ; R66       = 3333
        MOV     (R11),R66   ; R66       = 1234
        ...

        ADDR    ASC,R10     ; R11,R10   = ADR VON ASCII-Puffer
        MLLB    3[R10],R00  ; R00       = 7' TER CHARA IN ASC
        ...

TAB:    .WORD   01234,01111,02222,03333,04444...
```

[REG]N

Register indexed mit Auto-Increment/Decrement

6 (REG)N

6 [REG]N

Erklärung: Das Register enthält eine Adresse, zu der N bei Decrement vor, bei Increment nach der Operation automatisch addiert wird.
‼ POST-INCREMENT / PRE-DECREMENT ‼

(REG) Mit runden Klammern (Rxx) enthält das Register eine 16-Bit Adresse im gleichen (64k)RACK-Bereich wie der Befehl.

[REG] Mit eckigen Klammern [Rxx] enthält das Register eine 32-Bit Adresse.

N: N muss im Bereich von -64 ... +63 liegen.

```
Beispiel:   ADDR   @TAB,R10        ; R11,R10 = TAB-ADR
            MOV    (R10)+5,R66     ; R66 = 01234 , R10 = TAB+5
            MOV    (R10)-3,R66     ; R66 = 02222 , R10 = TAB+2

      TAB:  .WORD  01234,01111,02222,03333,04444,05555
```

REG[REG]

Register indexed mit Register Offset

6 REG(REG)

6 REG[REG]

Erklärung: Die Zieladresse bildet sich durch addieren der Basisadresse in (Rxx) und dem Offset in Ryy.

(REG) Mit runden Klammern (Rxx) enthält das Register eine 16-Bit Adresse im gleichen (64k)RACK-Bereich w ie der Befehl.

[REG] Mit eckigen Klammern [Rxx] enthält das Register eine 32- BitAdresse.

OFFSET-REG: Das Offset-Register enthält immer einen 16-Bit Offset mit Vorzeichen (-32768 ... 0 ... +32767).

Beispiele:

```
        MOV     TAB,R10         ; R10 = TAB-ADR
        MOV     2,R00           ; R00 = OFFSET

        MOV     R00(R10),R66    ; R66 = 02222

        ADDR    ASC,R10         ; R11,R10 = ASCII-PUFFER ADR
        MOV     3,R00           ; R00 = (WORD)OFFSET
        MHLB    R00[R10],R66    ; R66 = 6' TER CHARA IN ASC

TAB:    .WORD   01234,01111,02222,03333,04444,05555
```

ASC

ASCII-Puffer

A ASC

Erklärung: ASC zeigt auf den ASCII-Puffer, definiert in den Registern ASR(ASCII-Register Nummer) und ASL (ASCII-Puffer Länge).

Beachte: Nach INID oder SETD bilden die Register R60..R6F den ASCII-Puffer!

Diese Adressierungsart erzeugt keine SRC/DEST-Daten im Befehl!

Beispiel:

```
MLHB    10,ASL          ; MAX 10-ZEICHEN EINGABE
TIP     DEV,POS,ASC     ; TEXT-INPUT IN DEN ASCII-PUFFER
TIME    ATIM,ASC        ; UHRZEIT IN ASCII
TOP     DEV,POS,ASC     ; ANZEIGEN DES ASCII-PUFFERS
```

IB

INPUT-Base

C IB

Erklärung: IB zeigt auf die erste Eingangs-Karte.

Beachte: Darf nur als zweiter Parameter (DEST) angegeben werden !

Diese Adressierungsart erzeugt keine SRC/DEST-Daten im Befehl!

Beispiel:

```
THT0    15,IB          ; WARTE BIS I-15 = 1 WIRD
TBR1    128,IB,ERROR   ; ERROR WENN I-128 = 1 IST
```

OB

OUTPUT-Base

D OB

Erklärung: OB zeigt auf die erste Ausgangs-Karte (oder OUT-COPY).

Beachte: Darf nur als zweiter Parameter (DEST) angegeben werden !

Diese Adressierungsart erzeugt keine SRC/DEST-Daten im Befehl!

Beispiel:

```
SBIT    010,OB          ; SETZE DEN AUSGANG 16

MOTOR   = 35            ; AUSGANG MOTOR EIN
TBR0    MOTOR,OB,MOT_AUS ; TESTE OB MOTOR = AUS
```

FB

FLAG-Base

E FB

Erklärung: FB zeigt auf das erste FLAG-Wort.

Beachte: Darf nur als zweiter Parameter (DEST) angegeben werden !

Diese Adressierungsart erzeugt keine SRC/DEST-Daten im Befehl!

Beispiel:

```
THT0    13,FB    ; WARTE BIS F-13 = 1
CBIT    14,FB    ; SETZT F-14 = 0
```

Globale Adressen - Befehle

GGA

Get Global Address

B7_ 00_ GGA SRC, DEST:D

Erklärung: Suche das Label mit dem Namen in SRC in der globalen Variablentabelle und schreibe die Wort-Adresse (des Labels) nach DEST.

Existieren in verschiedenen Modulen Label mit dem selben Namen, kann auch noch der Modulname als Suchkriterium angegeben werden.
Damit ein Label in die globale Variablentabelle aufgenommen wird, muss es exportiert werden.

ERRORS: Der Task springt bei folgenden Errors auf seine ABORT-Adresse:
(Die Error-Nummer steht im 'APO')

041 Das Label wurde nicht gefunden
042 Das Label hat eine ungerade Byte-Adresse

Beispiel 1: Schreibe die Adresse des ISEC-Zählers nach R20/21.

```
        GGA     @TX.ISEC, R20

TX_ISEC:.TXT 'V_SYISEC'
```

Beispiel 2: Schreibe die Adresse der System-Busy-Tabelle nach R0/R1.

```
        GGA     @TX_BUSY, R0

TX_BUSY:.TXT 'SYSTEM.V_BUSY'
```

GGP

Get Global Pointer

B7_ 02_ GGP SRC, DEST:D

Erklärung: Suche das Label mit dem Namen in SRC in der globalen Variablentabelle, interpretiere das Doppelwort der Adresse des Labels als Byte-Pointer, wandle diesen in einen Word-Pointer und schreibe das Ergebnis nach DEST.

Existieren in verschiedenen Modulen Label mit dem selben Namen, kann auch noch der Modulname als Suchkriterium angegeben werden.
Damit ein Label in die globale Variablentabelle aufgenommen wird, muss es exportiert werden.

ERRORS: Der Task springt bei folgenden Errors auf seine ABORT-Adresse:
(Die Error-Nummer steht im 'APO')

041 Das Label wurde nicht gefunden
042 Der Byte-Pointer ist ungerade

Beispiel: Schreibe den Pointer auf den zentralen 1ms Timer nach R0/R1.

```
GGP     @TX_1MS, R0

TX_1MS: .TXT 'P_TIM1MS'
```

P_TIM1MS ist im Modul INIT z. Bsp. folgendermassen definiert:

```
P_TIM1MS:       .DOUBLE X'1603EA*2
```

➔ R0/R1 = 01603EA

GGD

Get Global Descriptor

B7_ 01_ GGD SRC, DEST:D

Erklärung: Suche das Label mit dem Namen in SRC in der globalen Variablentabelle und schreibe den Pointer auf dessen Deskriptor nach DEST.

Existieren in verschiedenen Modulen Label mit dem selben Namen, kann auch noch der Modulname als Suchkriterium angegeben w erden.
Damit ein Label in die globale Variablentabelle aufgenommen w ird, muss es exportiert w erden.

ERRORS: Der Task springt bei folgenden Errors auf seine ABORT-Adresse:
(Die Error-Nummer steht im 'APO')

041 Das Label w urde nicht gefunden

Beispiel : Benutze die Library-Funktion " F_EXQTSK" um einen Johann auf Adresse 045A000 zu starten.

```
GGD     @TX_EXQ, R10
RCXP    045A000, 0, R10
```

TX_EXQ: .TXT. 'F_EXQTSK'

TASK-KONTROLL-Befehle

EXQ

EXeQute

0Cxx SAD EXQ SRC,DEST,SAD

Erklärung: Starte das Programm bei SRC auf dem ersten freien Task und schreibe die Nummer von diesem Task nach DEST. Alle Register im neuen Task w erden gelöscht!

Ist kein Task mehr frei, springe nach SAD.

Beispiel 1: Starte den ersten freien Task mit der Start-Adresse ADRE.
Rechne die neue Task-Nummer nach REG 00:

```
EXQ     ADRE,R00,SAD
```

Beispiel 2: Starte einen Task auf der Doublew ord-Adresse 045' A000:

```
        MOVD    045A000,R10         ; R10 = TASK START-ADRESSE
        BSR     EXQD                ; STARTE DEN TASK
        ...

        Subroutine für den Befehl EXQD:
        R10     = ADR:D
        R00..03 Used
EXQD:   EXQ     HALT,R00,ERROR      ; R00 = TASK-PROG NUMMER
        GPNR    R01                 ; R01 = EIGENE PROG-NUMMER
        SUB     R01,R00             ; R00 = PNR-DIFFERENZ
        MUL     080,R00             ; R00 = REGISTER-SPACE
        ADDR    MPC,R02             ; R02 = ADR VOM EIGENE MPC
        MOVD    R10,R00[R02]        ; STARTE DEN TASK AUF 45' A000
        RTM     0
HALT:   BRA     HALT                ; TASK BLEIBT STEHEN
```

GPNR

Get Program NumbeR

0Bx0 GPNR DEST

Erklärung: Schreibe die eigene Task-Nummer nach DEST.

Beispiel: Rechne die eigene Task-Nummer nach REG 00:

```
GPNR    R00
```

JSKI

Johann Self KIll

00x0 JSKI

Erklärung: Lösche den eigenen Task und gib alle reservierten Devices frei.

Beispiel: Lösche den eigenen Task:

JSKI

JOKI

JOhann KIll

0Fx0 JOKI SRC

Erklärung: Lösche den Task mit der Task-Nummer in SRC und gebe alle von ihm reservierten Devices frei.

Beispiel: Lösche den Task Nr. 5:

```
JOKI  5
```

JSAB

Johann Self ABort

0Dxx JSAB

Erklärung: Setze den eigenen Task auf seine Abort-Adresse ABA. Rette den aktuellen Stackpointer (ins R77-HIGH-Byte) und setze ihn (R77- LOW-Byte) auf 00.

Ist ABA = 0000 dann lösche den Task und gebe alle von ihm reservierten Devices frei.

Beispiel: Springe auf ABA; (Kill Stack):

JSAB

JOAB

JOhann ABort

0Exx JOAB SRC

Erklärung: Setze den Task mit der Task-Nummer in SRC auf seine Abort-Adresse ABA. Rette dessen aktuellen Stackpointer (ins R77-HIGH-Byte) und setzt ihn (R77-LOW-Byte) auf 00.

Ist ABA = 0000 dann lösche den Task und gebe alle von ihm reservierten Devices frei.

Beispiel: Abort den Task mit der Nummer in R00:

```
JOAB R00
```

DELAY

DELAY

A5x0 DELAY SRC

Erklärung: Setze den 10ms Timer ' TIM' mit dem Wert in SRC und setze das Delay-Haltbit T im Haltwort HTW. Der Timer-Interrupt löscht dieses Haltbit wenn der TIM = 0000 wird.

Beachte: Da der Task während dem Delay auf HALT steht, wird das System in der Zeit um einen Task entlastet. Die Systemleistung kann somit durch geistreiche Anwendung dieses Befehls ganz erheblich gesteigert werden!

Kritische Befehle sind zum Beispiel: GTOP, TIP, HTOP, TIME

Beispiel 1: Setze den Ausgang 15 für 1-Sekunde auf eins:

```
SBIT    15,OB
DELAY   100
CBIT    15,OB
```

Beispiel 2: Zeit-Grossanzeige (nur RACK-Version):

```
LOOP:   TIME   ATIM,ASC      ; Get Time (ASCII)
GTOP    DEV,POS,ASC          ; Grossanzeige
DELAY   100                  ; System-Entlastung 1-SEC
BRA     LOOP
```

SPRUNG-Befehle

BRA

BRanch Always

F___ SAD BRA SAD

Erklärung: Springe auf die Adresse SAD. Im Befehl wird nur das Displacement
SAD - momentane Adresse abgelegt.
Mit SAD kann nur ein LABEL angegeben werden!
Displacement max. ± 07FF (1-WORD Befehl)

Beispiel: Springe auf LABEL:

LABEL: BRA LABEL

BSR

Branch to Sub-Routine

E___ SAD BSR SAD

Erklärung: Rette den aktuellen MPC im Stack und springe auf die Adresse SAD. Im Befehl wird nur das Displacement SAD - momentane Adresse abgelegt. Mit SAD kann nur ein LABEL angegeben werden!
Displacement max. ± 07FF (1-WORD Befehl)

Beispiel: Rufe ein Unterprogramm mit dem Namen SUBROUT auf:

```
BSR     SUBROUT
```

JMP_

JuMP

01x0	JMP	SRC
02x0	JMPD	SRC:D

Erklärung: Springe auf die Adresse SRC.
Hier kann mit SRC jede Adressierungsart verwendet werden!

Beispiel 1: Springe auf die Adresse 0A000 im aktuellen (64k) RACK-Bereich:

```
        JMP     0A000           ; RNR unverändert !
```

Beispiel 2: Springe ins Rack-3 auf die Adresse 04000:

```
IADR:   DOUBLE 034000
        JMPD    @IADR           ; RNR = 3 , MPC = 4000
```

JSM

Jump to Subroutine

03x0 JSM SRC

Erklärung: Rette den aktuellen MPC im Stack und springe auf die Adresse SRC im aktuellen (64k) RACK-Bereich. Hier kann mit SRC jede Adressierungsart verwendet werden!

Beispiel 1: Springe in die Subroutine auf ADRE:

```
JSM ADRE
```

Beispiel 2: Springe auf Adresse die in 011(R22) steht:

```
JSM 011(R22)
```

JAT

Jump indirect Address-Table

06__ JAT AT

Erklärung: Springe auf die Adresse, die unter AT in der Adresstabelle steht.
AT max. 0 ... 0FF (1-WORD Befehl)
Die Adresse von ATAB wird im INIT mit dem Pointer (HWMCB) auf die Macro Base-Page festgelegt.

Beispiel: MPC = (33(ATAB))

JAT 033

JST

Jump to Subroutine indirect address-Table

07__ JST AT

Erklärung: Rette den aktuellen MPC im Stack und springe auf die Adresse, die unter AT in der Adresstabelle steht.
AT max. 0 ... 0FF (1-WORD Befehl)
Die Adresse von ATAB wird im INIT mit dem Pointer (HWMCB) auf die Macro Base-Page festgelegt.

Beispiel: MPC = (33(ATAB))

JST 033

RTM

Return To Mainprogram

04__ RTM N

Erklärung: Rücksprung ins Hauptprogramm am Ende einer Subroutine. Dabei w erden N Worte vom Hauptprogramm übersprungen.
N max. ± 07F

Beispiel: Rücksprung ins Hauptprogramm und überspringe die nächsten fünf Worte:

RTM 5

JEX

Jump EXternal

0800 MSAD JEX MSAD

Erklärung: Springe in ein MIKRO-Programm mit der Adresse MSAD.
Wenn MSAD < 02000 ist, dann CXP MSAD(JEX-MODULE)
Wenn MSAD >= 02000 ist, dann JSR MSAD

CPU-Register: Die NS32016-Register werden wie folgt geladen:
R7 = Adresse von REG 00 vom aufrufendem Task
R6 = Adresse von JEX BEFEHL (Byte Adresse)
R5 = Adresse von NEXT BEFEHL (Wort Adresse)
Es dürfen alle CPU-Register verändert werden!

Das JEX-Modul wird im INIT bestimmt (z.B. HWJMD = MOD-5).

Beispiel 1: Call MICRO-ROUTINE auf Adresse 0100:B vom REX-Modul:

```
JEX     0100        ; PC = 0100          (REX-MODUL)
```

Beispiel 2: Call LOCAL-MICRO-ROUTINE, die auf Adresse MICRO steht:

```
JEX     MICRO       ; PC = 2*MICRO       (BYTE-ADR)
...
MICRO:  .BYTE  012,00       ; RET 0              NS32000-Micro
```

NSB.EXE: Das Programm NSB übersetzt ein NS32000'er Assembler-Programm (NAME.LST) in ein .BYTE-File (NAME.BYT), welches mit .INCLUDE eingebunden werden kann.

REX

load Registers and jump EXternal

09xx MSAD REX SRC:D,DEST:D,MSAD

Erklärung: Springe in ein MIKRO-Programm mit der Adresse MSAD und übergib die Parameter in SRC und DEST.
Wenn MSAD < 02000 ist, dann CXP MSAD(REX-MODULE)
Wenn MSAD >= 02000 ist, dann JSR MSAD

CPU-Register: Die NS32016-Register sind wie folgt geladen:
R7 = Adresse von REG 00 vom aufrufendem Task
R6 = Adresse von REX BEFEHL (Byte Adresse)
R5 = Adresse von NEXT BEFEHL (Wort Adresse)
R4 = Adresse von SRC
R3 = Adresse von DEST
R2 = Adresse von DEST
R1 = Inhalt von SRC:D
R0 = Inhalt von DEST:D
Es dürfen alle CPU-Register verändert werden!

Das REX-Modul wird im INIT bestimmt (z.B. HWJMD = MOD-5).

Beispiel 1: Call REX-MODUL PC(MOD)=0A und übergebe den Inhalt von REG00 und die Konstante 045 ans Micro Programm:

```
REX     R00,045,0A
```

Beispiel 2: Berechne die WORD-Adresse von der OUT-BASE ins REG 01,00 (Da FB,IB,OB als SRC nicht zugelassen ist, geht der ADDR-Befehl nicht!):

```
REX     R00,OB,D_ADR            ; geht auch für FB,IB...
```

Mikro-Programm: Adr von DEST nach SRC !

```
D_ADR:  .BYT    0CE,00F,013,0,03E,012,0,0
        EXTSD   R2,0(R4),1,31   ; R2/2 → [R4]
        RET     0               ; zurück ins Makro
```

NSB.EXE:	Das Programm NSB übersetzt ein NS32000'er Assembler-Programm (NAME.LST) in ein .BYTE-File (NAME.BYT), welches mit .INCLUDE eingebunden werden kann.

CXP

Call eXternal Procedure

```
B7_03_      CXP     DESC:D
```

Erklärung: Springe in die Mikro-Procedure mit dem Deskriptor DESC.
DESC muss zuerst mit GGD geladen werden.

CPU-Register: Die NS32016-Register werden wie folgt geladen:
R7 = Adresse von REG 00 vom aufrufendem Task
R6 = Adresse von JEX BEFEHL (Byte Adresse)
R5 = Adresse von NEXT BEFEHL (Wort Adresse)
Es dürfen alle CPU-Register verändert werden!

Beispiel : Call MICRO-ROUTINE "MEIN_PROC", die in irgendeinem Modul definiert ist.

```
        GGD     @TX_MEIN, R10
        CXP     R10

TX_MEIN:.TXT 'MEIN_PROC'
```

RCXP

load Registers and Call eXternal Procedure

B7_04_ RCXP SRC:D,DEST:D,DESC:D

Erklärung: Springe in die MIKRO-Procedure mit dem Descriptor DESC und übergib die Parameter SRC und DEST.
DESC muss zuerst mit GGD geladen w erden

CPU-Register: Die NS32016-Register sind w ie folgt geladen:
R7 = Adresse von REG 00 vom aufrufendem Task
R6 = Adresse von REX BEFEHL (Byte Adresse)
R5 = Adresse von NEXT BEFEHL (Wort Adresse)
R4 = Adresse von SRC
R3 = Adresse von DEST
R2 = Adresse von DEST
R1 = Inhalt von SRC:D
R0 = Inhalt von DEST:D
Es dürfen alle CPU-Register verändert w erden!

Beispiel : Benutze die Library-Funktion " F_EXQTSK" um einen Johann auf Adresse 045A000 zu starten.

```
GGD     @TX_EXQ, R10
RCXP    045A000, 0, R10

TX_EXQ: .TXT. 'F_EXQTSK'
```

BIT-Befehle

TBR0

Test and BRanch if bit = 0

10xx SAD TBR0 OFF,BASE,SAD

Erklärung: Teste das Bit (Offset,Base) und springe auf SAD, wenn das Bit = 0 ist.

TBSR0: * Springt der Befehl auf SAD, so kann die Rücksprung-Adresse mit RTM 255 auf das Stack geholt werden und anschliessend mit RTM 0 (unter den TBR-Befehl) zurück gesprungen werden (entspricht einem TBSR-Befehl).

Beispiel 1: Springe auf LABEL, wenn der Eingang 35 = 0 ist:

```
TBR0    35,IB,LABEL
```

Beispiel 2: Springe auf LABEL, wenn das Flag mit der Nummer in R00 nicht gesetzt ist:

```
TBR0    R00,FB,LABEL
```

* Ab System Rev. 5.11

TBR1

Test and BRanch if bit = 1

11xx SAD TBR1 OFF,BASE,SAD

Erklärung: Teste das Bit (Offset,Base) und springe auf SAD, wenn das Bit = 1 ist.

TBSR1: * Springt der Befehl auf SAD, so kann die Rücksprung-Adresse mit RTM 255 auf das Stack geholt werden und anschliessend mit RTM 0 (unter den TBR-Befehl) zurück gesprungen werden (entspricht einem TBSR-Befehl).

Beispiel 1: Springe auf LABEL, wenn der Eingang 15 = 1 ist:

```
TBR1    15,IB,LABEL
```

Beispiel 2: Springe auf LABEL, wenn in R10 eine negative Zahl ist (Bit 15 = Signum der Zahl = 1 wenn negativ):

```
TBR1    15,R10,LABEL
```

* Ab System Rev. 5.11

THT0

Test and HalT if bit = 0

15xx THT0 OFF,BASE

Erklärung: Halt, wenn das Bit (Offset,Base) = 0 ist.

Beispiel 1: Warte bis das FLAG 5 = 1 wird:

```
THT0    5,FB
```

Beispiel 2: Warte bis der Eingang 35 gesetzt wird:

```
THT0    35,IB
```

THT1

Test and HalT if bit = 1

16xx THT1 OFF,BASE

Erklärung: Halt, wenn das Bit (Offset,Base) = 1 ist.

Beispiel 1: Warte bis das FLAG mit der Nummer in R00 gelöscht wird:

```
THT1    R00,FB
```

Beispiel 2: Warte bis der Eingang 5 nicht mehr gesetzt ist:

```
THT1    5,IB
```

THTT0

Test and HalT if bit = 0 and branch if Timeout

46xx THTT0 OFF, BASE, TIME, ERRORNR, SAD

Erklärung: Halt (wenn das Bit (Offset,Base) = 0 ist) solange, bis entweder das Bit (Offset, Base) = 1 wird oder die Zeit (TIME) abgelaufen ist. Ist die Zeit TIME abgelaufen, so springe auf SAD und schreibe ERRORNR nach R70.

RETRY: * Springt der Befehl auf SAD, so kann die Adresse vom THTT-Befehl selbst mit RTM 255 auf das Stack geholt werden und (z. B. nach einer Fehlermeldung) mit RTM 0 auf den Befehl zurück gesprungen werden (Retry).

Beispiel : Warte max. 1 sec. bis der Eingang 5 = 1 ist. Bei Timeout springe nach LABEL und schreibe 7 ins R70.

```
THTT0   5, IB, 1000, 7, LABEL
```

* Ab System Rev. 5.11

THTT1

Test and HalT if bit = 1 and branch if Timeout

47xx THTT1 OFF, BASE, TIME, ERRORNR, SAD

Erklärung: Halt (wenn das Bit (Offset,Base) = 1 ist) solange, bis entweder das Bit (Offset, Base) = 0 wird oder die Zeit (TIME) abgelaufen ist. Ist die Zeit TIME abgelaufen, so springe auf SAD und schreibe ERRORNR nach R70.

RETRY: * Springt der Befehl auf SAD, so kann die Adresse vom THTT-Befehl selbst mit RTM 255 auf das Stack geholt werden und (z. B. nach einer Fehlermeldung) mit RTM 0 auf den Befehl zurück gesprungen werden (Retry).

Beispiel : Warte max. 1 sec. bis der Eingang mit der Nummer in R10 = 0 ist. Bei Timeout springe nach LABEL und schreibe 8 ins R70.

```
THTT1   R10, IB, 1000, 8, LABEL
```

* Ab System Rev. 5.11

SBIT

Set BIT

12xx SBIT OFF,BASE

Erklärung: Setze das Bit (Offset,Base) = 1.

Bemerkung: Dieser READ-MODIFY-WRITE Befehl wird im Interlocked-mode ausgeführt und kann daher auch im Multiprozessor-Betrieb nicht von einer andern CPU unterbrochen werden. Deshalb wird er auch für die Kommunikation mehrerer CPUs auf dem BUS mittels FLAGs verwendet.
(Nur SBIT- und CBIT-Befehl!)

Beispiel 1: Setze den Ausgang 45 auf 1:

```
SBIT    45,OB
```

Beispiel 2: Setze das Bit mit der Nummer in R00 im Register R10:

```
SBIT    R00,R10
```

Beispiel 3: Setze das Flag 128:

```
SBIT    128,FB
```

CBIT

Clear BIT

13xx CBIT OFF,BASE

Erklärung: Lösche das Bit (Offset,Base) = 0.

Bemerkung: Dieser READ-MODIFY-WRITE Befehl wird im Interlocked-mode ausgeführt und kann daher auch im Multiprozessor-Betrieb nicht von einer andern CPU unterbrochen werden. Deshalb wird er auch für die Kommunikation mehrerer CPUs auf dem BUS mittels FLAGs verwendet.
(Nur SBIT- und CBIT-Befehl!)

Beispiel 1: Lösche Flag 5:

```
CBIT    5,FB
```

Beispiel 2: Lösche Bit 15 in REG 33:

```
CBIT    15,R33
```

IBIT

Invert BIT

14xx IBIT OFF,BASE

Erklärung: Invertiere das Bit (Offset,Base); 1 → 0 ; 0 → 1 .

Beispiel: Blinke mit dem Ausgang 155 im Sekunden-Takt:

```
LOOP:   IBIT    155,OB      ; WECHSELN
        DELAY   100         ; 1 SEC
        BRA     LOOP
```

MBIT

Move BIT

18xx 00xx MBIT OFF,BASE,OFF2,BASE2

Erklärung: Kopiere das Bit (OFF,BASE) nach Bit (OFF2,BASE2).

Beispiel: Kopiere das Eingangs-Bit 045 auf den Ausgang 5:

```
MBIT  045,IB,5,OB
```

MINB

Move INvert Bit

19xx 00xx MINB OFF1,BASE1,OFF2,BASE2

Erklärung: Kopiere das invertierte Bit von (OFF1,BASE1) nach Bit (OFF2,BASE2).

Beispiel: Kopiere das invertierte Bit 1 aus R22 nach FLAG 5:

```
MINB 1,R22,5,FB
```

FFSB

Find First Set Bit

1Bxx 00xx FFSB OFF,BASE,N,DEST

Erklärung: Teste N Bits ab Bit (OFF,BASE) auf ‘ 1’ und übergebe die Nummer des ersten gesetzten Bits nach DEST. Ist keines dieser Bits gesetzt, setze DEST = 0FFFF.

ACHTUNG: Obwohl N mit 1 ... 32 angegeben werden kann, werden je nach Start-Bit nur bis zu 25 Bits verarbeitet. Die CPU holt zuerst die zu verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins interne Register. Das heisst, der Bit-Range kann sich nur innerhalb von 4 Bytes bewegen. Daher ist N wie folgt beschränkt:

OFF = 00,08,10,18...	N max = 32
OFF = 01,09,11,19...	N max = 31
OFF = 02,0A,12,1A...	N max = 30
OFF = 03,0B,13,1B...	N max = 29
OFF = 04,0C,14,1C...	N max = 28
OFF = 05,0D,15,1D...	N max = 27
OFF = 06,0E,16,1E...	N max = 26
OFF = 07,0F,17,1F...	N max = 25

Beispiel 1: Suche die Bit-Nummer des ersten gesetzten Bits in REG 00 und übergebe es dem REG 22:

```
FFSB    0,R00,16,R22
```

Beispiel 2: Suche das erste gesetzte FLAG im Bereich FL-45 ... 54 und schreibe die Bit-Nummer in R10: Ist z.B. das FL-50 das erste gesetzt Flag, so wird R10 = 5!

```
FFSB    45,FB,10,R22
```

SBR_

Set Bit Range

1Dxx 00xx SBR OFF,BASE,N,SRC

SBRD OFF,BASE,N,SRC:D

Erklärung: Kopiere N Bits aus SRC nach Bit (OFF,BASE) und folgende.
SBR N = 1 ... 16
SBRD N = 1 ... 32

ACHTUNG: Obwohl N mit 1 ... 32 angebeben werden kann, werden je nach Start-Bit nur bis zu 25 Bits verarbeitet. Die CPU holt zuerst die zu verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins interne Register. Das heisst, der Bit-Range kann sich nur innerhalb von 4 Bytes bewegen. Daher ist N wie folgt beschränkt:

OFF = 00,08,10,18...	N max = 32
OFF = 01,09,11,19...	N max = 31
OFF = 02,0A,12,1A...	N max = 30
OFF = 03,0B,13,1B...	N max = 29
OFF = 04,0C,14,1C...	N max = 28
OFF = 05,0D,15,1D...	N max = 27
OFF = 06,0E,16,1E...	N max = 26
OFF = 07,0F,17,1F...	N max = 25

Beachte: Bei diesem Befehl ist die Reihenfolge DEST,N,SRC !

Beispiel: Kopiere 24 Bits aus REG 01,00 auf die Ausgänge ab Ausgangs-Bit 045:

SBRD 045,OB,24,R00

LBR_

Load Bit Range

1Fxx 00xx LBR OFF,BASE,N,DEST

LBRD OFF,BASE,N,DEST:D

Erklärung: Kopiere N Bits ab Bit (OFF,BASE) rechtsbündig nach DEST und fülle die restlichen Bits in DEST mit '0'.
LBR N = 1 ... 16
LBRD N = 1 ... 32

ACHTUNG: Obwohl N mit 1 ... 32 angebeben werden kann, werden je nach Start-Bit nur bis zu 25 Bits verarbeitet. Die CPU holt zuerst die zu verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins interne Register. Das heisst, der Bit-Range kann sich nur innerhalb von 4 Bytes bewegen. Daher ist N wie folgt beschränkt:

OFF = 00,08,10,18...	N max = 32
OFF = 01,09,11,19...	N max = 31
OFF = 02,0A,12,1A...	N max = 30
OFF = 03,0B,13,1B...	N max = 29
OFF = 04,0C,14,1C...	N max = 28
OFF = 05,0D,15,1D...	N max = 27
OFF = 06,0E,16,1E...	N max = 26
OFF = 07,0F,17,1F...	N max = 25

Beispiel: Lade 30 Eingänge ab Eingangs-Bit 045 nach REG 01,00:

LBRD 045,IB,24,R00

MOVE-Befehle

MOV_

MOVe

20xx	MOV	SRC,DEST
30xx	MOVD	SRC:D,DEST:D
CAxx	MOVF	SRC:F,DEST:F
DAxx	MOVL	SRC:L,DEST:L

Erklärung: Kopiere den Inhalt von SRC nach DEST.

VORSICHT: MOVF und MOVL gehen auf TRAP-3 wenn die Zahlen keine Floating- Point Zahlen sind!

Beispiel: Lade R00 mit dem Inhalt von der Adresse ‘ ADRE ‘ :

```
MOV     @ADRE,R00
```

XCH_

eXCHange

21xx	XCH	SRC,DEST
31xx	XCHD	SRC:D,DEST:D

Erklärung: Tausche den Inhalt von SRC und DEST.

Beispiel: Tausche den Inhalt von R00 und R10:

```
XCH     R00,R10
```

MZ__

Move Zero extended

22xx	MZBW	SRC:B,DEST:W
32xx	MZBD	SRC:B,DEST:D
34xx	MZWD	SRC:W,DEST:D

Erklärung: Kopiere den Inhalt von SRC nach DEST und fülle die weiteren Bits ins DEST mit 0.

Beispiel 1: Kopiere das erste Zeichen vom ASCII-Puffer nach R10 und lösche das obere Byte in R10 und das ganze R11:

```
MZBD    ASC,R10
```

Beispiel 2:

```
MZWD    08000,R10       ; R11 = 0000 , R10 = 8000
```

MX__

Move signum eXtended

23xx	MXBW	SRC:B,DEST:W
33xx	MXBD	SRC:B,DEST:D
35xx	MXWD	SRC:W,DEST:D

Erklärung: Kopiere den Inhalt von SRC nach DEST und fülle die weiteren Bits ins DEST mit dem Vorzeichen von SRC.
SRC = positiv : fülle mit 0
SRC = negativ : fülle mit 1

Beispiele: R22 = 0087 !

```
MXBW   R22,R22  ; R22      = FF87
MXBD   R22,R22  ; R23,22   = FFFF,FF87
MXWD   1234,R55           ; R56,55  = 0000,1234
```

MB__

Move Byte

27xx	MLLB	SRC:LB,DEST:LB
24xx	MLHB	SRC:LB,DEST:HB
25xx	MHLB	SRC:HB,DEST:LB
26xx	MHHB	SRC:HB,DEST:HB

Erklärung: Kopiere ein Byte von SRC nach DEST. Das andere Byte in DEST bleibt unverändert, wenn DEST im CRAM-Bereich ist.
L = Lower Byte H = Higher Byte

Beispiel 1: Limitiere den ASCII-Puffer auf 10 Zeichen:

```
MLHB    10,ASL
```

Beispiel 2: Überschreibe den zweiten Buchstaben im ASCII-Puffer mit 'A':

```
MLHB    "A",ASC
```

DUMP

Dump

0Axx DUMP SRC,N,DEST

Erklärung: Kopiere N 16-Bit Worte von SRC nach DEST. (kopiert aufsteigend!)

Beispiel 1: Kopiere 01000..013FF nach 02000..023FF:

```
DUMP    @01000,0400,@02000
```

Beispiel 2: Lösche den Speicher 0A000..0BFFF:

```
MOV     0,@0A000
DUMP    @0A000,01FFF,@0A001
```

LOGIK-Befehle

AND_

AND

28xx AND SRC,DEST

28xx ANDD SRC:D,DEST:D

Erklärung: Lösche alle Bits in DEST, die in SRC gelöscht sind.

SRC	&	DEST	=	DEST
0	&	0	=	0
0	&	1	=	0
1	&	0	=	0
1	&	1	=	1

Beispiel: Maskiere R00 mit 0FF00:

```
AND     0FF00,R00
```

OR_

OR

29xx OR SRC,DEST

39xx ORD SRC:D,DEST:D

Erklärung: Setzte alle Bits in DEST, die in SRC gesetzt sind.

SRC	#	DEST	=	DEST
0	#	0	=	0
0	#	1	=	1
1	#	0	=	1
1	#	1	=	1

Beispiel: Setze alle Bits im R00, die in ADRE gesetzt sind:

```
OR          @ADRE,R00
```

XOR_

eXclusive OR

2Axx XOR SRC,DEST

3Axx XORD SRC:D,DEST:D

Erklärung: Invertiere alle Bits in DEST, die in SRC gesetzt sind.

SRC	$	DEST	=	DEST
0	$	0	=	0
0	$	1	=	1
1	$	0	=	1
1	$	1	=	0

Beispiel: Invertiere BIT 4 und 2 in R33:

```
XOR 014,R33
```

COM_

COMplement

2Bxx	COM	SRC,DEST
3Bxx	COMD	SRC:D,DEST:D

Erklärung: Kopiere das invertierte SRC nach DEST.

Beispiel: Invertiere alle Bits im R22:

```
COM     R22,R22
```

LSH_

Logic SHift

2Dxx LSH N,DEST

3Dxx LSHD N,DEST:D

Erklärung: Schiebe DEST N mal links (N=pos) oder rechts (N=neg) und fülle die neuen Bits mit '0'.

Shift left: N = 1 ... 31
Shift right: N = -1 ... -31

Beispiel: Schiebe R00 5mal links:

```
                                ; R00     =  0001
LSH    5,R00                    ; Schiebe links
                                ; R00     =  0020
```

ASH_

Arithmetic SHift

2Cxx	ASH	N,DEST
3Cxx	ASHD	N,DEST:D

Erklärung: Schiebe DEST N mal links (N=pos) und fülle die neuen Bits mit 0 oder schiebe DEST N mal rechts (N=neg) und erweitere mit dem Vorzeichen von DEST.

Shift left: N = 1 ... 31
Shift right: N = -1 ... -31

Beispiel: Teile R00 durch vier. Vorzeichen bleibt erhalten:

```
                        ; R00 = 0FF00    (-256)
ASH     -2,R00          ; Schiebe rechts, behalte Vorzeichen
                        ; R00 = 0FFC0    (-64)
```

ROT_

ROTate

2Exx	ROT	N,DEST
3Exx	ROTD	N,DEST:D

Erklärung: Rotiere DEST N mal links (N=pos) oder rechts (N=neg).

Rotate left: N = 1 ... 31
Rotate right: N = -1 ... -31

Beispiel: Rotiere R22 5mal links:

```
                    ; R22  = 1000
ROT   5,R22         ; Rotiere links
                    ; R22  = 0002
```

ARITHMETIK-Befehle

ADD_

ADDition

40xx	ADD	SRC,DEST
50xx	ADDD	SRC:D,DEST:D
C0xx	ADDF	SRC:F,DEST:F
D0xx	ADDL	SRC:L,DEST:L

Erklärung: Addiere SRC und DEST nach DEST.

Beispiel 1: Addiere 1 zu R00:

```
ADD     1,R00           ; R00 = R00+1
```

Beispiel 2: Addiere Pi zu R23,22:

```
ADDF    3.141592654,R22 ; R23,22 + Pi       Floating Point
```

SUB_

SUBtraktion

41xx	SUB	SRC,DEST
51xx	SUBD	SRC:D,DEST:D
C1xx	SUBF	SRC:F,DEST:F
D1xx	SUBL	SRC:L,DEST:L

Erklärung: Subtrahiere SRC von DEST nach DEST.

Beispiel: Subtrahiere 1 von R23,22

```
SUBD    1,R22       ;R23,22 = R23,22-1
```

MUL_

MULtiplikation

42xx	MUL	SRC,DEST
52xx	MULD	SRC:D,DEST:D
C2xx	MULF	SRC:F,DEST:F
D2xx	MULL	SRC:L,DEST:L

Erklärung: Multipliziere SRC mit DEST nach DEST.

Beispiel: Multipliziere R23,22 mit 3.3:

```
MULF    3.3,R22          ; R23,22 = R23,22 * 3.3
```

DIV_

DIVision

43xx	DIV	SRC,DEST
53xx	DIVD	SRC:D,DEST:D
C3xx	DIVF	SRC:F,DEST:F
D3xx	DIVL	SRC:L,DEST:L

Erklärung: Dividiere DEST durch SRC nach DEST.

```
+   10  /  +   3      =      +      3
- 10  / +   3  =   -      4 *           * Rundet anders als QUO!
+   10  /  - 3  =      -      4 *
- 10  / - 3  = +   3
```

Beispiel: Dividiere R25,24,23,22 mit 3.3:

```
DIVL    3.3,R22         ; R25,24,23,22 = R25,24,23,22 / 3.3
```

QUO_

QUOtient

48xx QUO SRC,DEST

58xx QUOD SRC:D,DEST:D

Erklärung: Berechne den Quotienten von DEST/SRC nach DEST.

```
+  10 QUO  +  3   = + 3
-  10 QUO  +  3   = -  3 *          * Rundet anders als DIV!
+  10 QUO  -  3   = -  3 *
-  10 QUO  -  3   = + 3
```

Beispiel: Berechne den Quotienten von R22 / 033:

```
QUO   033,R22
```

MOD_

MODulus

49xx	MOD	SRC,DEST
59xx	MODD	SRC:D,DEST:D

Erklärung: Berechne den Rest von DEST/SRC nach DEST.

+ 10	MOD + 3 =	+	1	
- 10	MOD + 3 =	+	2 *	* Rundet anders als REM!
+ 10	MOD - 3 =	-	2 *	
- 10	MOD - 3 =	-	1	

Beispiel: Rechne R22 MOD R00 nach R22:

```
MOD     R00,R22
```

REM_

REMainder

4Axx	REM	SRC,DEST
5Axx	REMD	SRC:D,DEST:D

Erklärung: Berechne den Rest von DEST/SRC nach DEST.

```
+  10 REM+3  = +1
- 10 REM  +3  = - 1 *      * Rundet anders als MOD !
+  10 REM- 3  = +1 *
- 10 REM  - 3  = - 1
```

Beispiel: Berechne den Rest der DIV R22 / 3 nach R22:

```
REM    3,R22
```

SQR_

SQuare Root

C6xx	SQRF	SRC:F,DEST:F
D6xx	SQRL	SRC:L,DEST:L

Erklärung: Berechne die Quadrat-Wurzel von SRC nach DEST.

Beispiel: Rechne die Wurzel von 2 nach R23,22,21,20: (Long Floating)

```
SQRL    2.0,R20
```

ABS_

ABSolute

4Bxx	ABS	SRC,DEST
5Bxx	ABSD	SRC:D,DEST:D
C5xx	ABSF	SRC:F,DEST:F
D5xx	ABSL	SRC:L,DEST:L

Erklärung: Berechne den absoluten Wert von SRC nach DEST.
neg → pos ; pos bleibt positiv !

Beispiel: Rechne den absoluten Wert von R00 nach R22:

```
ABS     R00,R22
```

NEG_

NEGate

4Cxx	NEG	SRC,DEST
5Cxx	NEGD	SRC:D,DEST:D
C4xx	NEGF	SRC:F,DEST:F
D4xx	NEGL	SRC:L,DEST:L

Erklärung: Berechne den negativen Wert von SRC nach DEST.
neg → pos ; pos → neg

Beispiel: Negiere den Wert in R25,24,23,22:

```
NEGL    R22,R22
```

CONVERT-Befehle

MOV__

Floating to Integer

CExx	MOVFW SRC:F,DEST:W
CFxx	MOVFD SRC:F,DEST:D
DExx	MOVLW SRC:L,DEST:W
DFxx	MOVLD SRC:L,DEST:D

Erklärung: Wandle die Floating Point Zahl in SRC in eine Integer-Zahl nach DEST.

Beispiel: Konvertiere die Floating Point Zahl R25,24,23,22 in eine Integer-Zahl R45,44:

```
MOVLD R22,R44
```

MOV__

Integer to Floating

CCxx	MOVWF SRC:W,DEST:F
CDxx	MOVDF SRC:D,DEST:F
DCxx	MOVWL SRC:W,DEST:L
DDxx	MOVDL SRC:D,DEST:L

Erklärung: Wandle die Integer-Zahl in SRC in eine Floating Point Zahl nach DEST.

Beispiel: Konvertiere die Integer-Zahl 123 in eine Floating Point Zahl nach R25,24,23,22:

```
MOVWL  123,R22          ; R22:L = 123.0
```

HDCV_

Hex Decimal ConVert

4Exx	HDCV	SRC,DEST
5Exx	HDCVD	SRC:D,DEST:D

Erklärung: Wandle die HEX-Zahl in SRC in eine Dezimal-Zahl (BCD-Zahl) nach DEST.

Beispiel: Wandle den HEX-Wert in R22 in den Dezimal-Wert:

```
HDCV    R22,R22
```

DHCV_

Decimal Hex ConVert

4Fxx	DHCV	SRC,DEST
5Fxx	DHCVD	SRC:D,DEST:D

Erklärung: Wandle die Dezimal-Zahl (BCD-Zahl) in SRC in eine HEX-Zahl nach DEST.

Beispiel: Wandle den Dezimal-Wert in R22 in den HEX-WERT:

```
DHCV    R22,R22
```

ADDR

ADDRess calculation

5Dxx ADDR SRC,DEST:D

Erklärung: Berechne die Adresse von SRC nach DEST (Double-Word Address).

Beispiel 1: Rechne die Adresse von REG 00 nach REG 00/R01:

```
ADDR    R00,R00
```

Beispiel 2: Berechne die Adresse vom ASCII-PUFFER nach R01,00:

```
ADDR    ASC,R00
```

VERGLEICHS-Befehle

CBR_

Compare and BRanch absolute

6_xx SAD	CBR	SRC:W,COND,DEST:W,SAD
7_xx SAD	CBRD	SRC:D,COND,DEST:D,SAD

Erklärung: Vergleiche SRC mit DEST und springe nach SAD wenn die Bedingung erfüllt ist. Das Vorzeichen wird nicht getestet (08000>07FFF!)

BEF	COND	Funktion	
0	=	BR IF EQUAL	
1	<>,><	BR IF NOT EQUAL	
2	<	BR IF LESS THAN	
3	<=,=<	BR IF LESS THAN OR EQUAL	
4	>	BR IF GREATER	
5	>=,=>	BR IF GREATER OR EQUAL	
C	&Z	BR IF AND = 0	DEST unverändert
D	&N	BR IF AND >< 0	DEST unverändert
E	+Z	BR IF ADD = 0	DEST=DEST+SRC !!
F	+N	BR IF ADD >< 0	DEST=DEST+SRC !!

Beispiel 1: Springe nach SAD, wenn R10,11 = R22,23 ist:

```
CBRD    R10,=,R22,SAD
```

Beispiel 2: Durchlaufe einen LOOP 125 mal:

```
        MOV     125,R00             ; INIT LOOP-Counter
LOOP:   ...                         ; LOOP-Befehle
        CBR     -1,+N,R00,LOOP      ; LOOP-Counter
```

Beispiel 3: Suche das Text-Ende im ASCII-Puffer:

```
        ADDR    ASC,R0              ; Adresse vom ASCII-PUFFER
LOOP:   CBR     000FF,&Z,[R0],EOTL  ; Test Lower-Byte
        CBR     0FF00,&N,[R0]+1,LOOP ; Test Higher-Byte, Adresse+1
EOTH:                               ; Text-Ende im High-Byte -1[R4]
EOTL:                               ; Text-Ende im Low-Byte 0[R4]
```

CBRS_

Compare and BRanch Signed

6_xx SAD CBRS SRC:W,COND,DEST:W,SAD
7_xx SAD CBRSD SRC:D,COND,DEST:D,SAD

Erklärung: Vergleiche SRC mit DEST und springe nach SAD, wenn die Bedingung erfüllt ist. Das Vorzeichen wird getestet (08000<07FFF!)

BEF	COND	Funktion
6	<	BR IF LESS THAN
7	<=,=<	BR IF LESS THAN OR EQUAL
8	>	BR IF GREATER
9	>=,=>	BR IF GREATER OR EQUAL

Beachte: Die Vergleiche = und <> dürfen auch beim CBRS und CBRSD angegeben werden. Sie werden jedoch automatisch in einen normalen CBR oder CBRD gewandelt.

Beispiel 1: Springe nach SAD, wenn R22 positiv ist:

```
CBRS    R22,>=,0,SAD            ; Test SIGNED
```

Beispiel 2: Springe nach SAD, wenn R22,23 negativ ist:

```
CBRSD   R22,<,0,SAD             ; Test SIGNED
```

CBR_

Compare and BRanch floating

```
A_xx SAD    CBRF    SRC:F,COND,DEST:F,SAD
B_xx SAD    CBRL    SRC:L,COND,DEST:L,SAD
```

Erklärung: Vergleiche SRC mit DEST und springe nach SAD, wenn die Bedingung erfüllt ist.

BEF	COND	Funktion
A	=	BR IF EQUAL
B	<>,><	BR IF NOT EQUAL
C	<	BR IF LESS THAN
D	<=,=<	BR IF LESS THAN OR EQUAL
E	>	BR IF GREATER
F	>=,=>	BR IF GREATER OR EQUAL

Beispiel 1: Springe nach SAD, wenn R22,23 >= 123.456 E15 ist:

```
CBRF    R22,>=,123.456E15,SAD          ; Floating-Point
```

Beispiel 2: Springe nach SAD, wenn R10..13 < PHI ist:

```
CBRL    R22,>=,3.141592654,SAD         ; Long-Floating
```

TIME- Befehle

TIME

get/set TIME

B4x_ TIME ART,ADRE

Voraussetzung:
- PCMASTER Firmware Rev.1.56 oder höher
- TRANS.EXE 1.7 oder höher
- INI-File-Eintrag unter Rubrik [PCMaster] EnableTime= YES

Erklärung: Setze oder lese die Uhr-Zeit, Datum, Wochentag oder Tagesnummer.

Code	ART	Übergabe	Beispiel	
0	ADAT	ASC DATE	"DD.MM.YY"	26.04.90
1	ATIM	ASC TIME	"HH:MM:SS"	11:51:33
2	ADOW	ASC DAY OF WEEK	"DW"	DO
3	ADNR	ASC DAY NR.	"DNR"	116
B	ATOT	ASC TIME TOTAL	"DD.MM.YY__HH.MM.SS__DW__DNR"	
4	BDAT	BIN DATE	00YY' MMDD	00900426
5	BTIM	BIN TIME	HHMM' SSZZ	11513300
*6	BDOW	BIN LANGUAGE & DAY OF WEEK	0L0D	0004
7	BDNR	BIN DAY NR.	0YYY' YDNR	01990116

Language: Der Wochentag kann in mehreren Sprachen angezeigt werden:
L: 0 = German , 1 = English , 2 = Italian , 3 = French

Wenn die Anlage mehrsprachig ausgelegt ist, kann LANGUAGE als (Batterie-gespeichertes) Sprachwahl-Bit für die ganze Anlage dienen!

Day of week: D: 1 = Montag , 2 = Dienstag ... 7 = Sonntag

Beachte: Die Länge des ASCII-Puffers ' ASL' wird nicht getestet! Der Jahreswechsel und das Schaltjahr wird auch über das Jahr 2000 automatisch und richtig verarbeitet.

Beispiel: Zeige dauernd die aktuelle Zeit an. Da die Zeit nur mit Sekunden-Auflösung angezeigt wird, kann das System wesentlich entlastet werden, wenn ein DELAY-Befehl eingefügt wird:

```
LOOP:   TIME    ATOT,ASC        ; DATUM , ZEIT , DOW , DNR
        TOP     DEV,POS,ASC     ; ANZEIGE AUF BILDSCHIRM
        DELAY   100             ; SYSTEM-ENTLASTUNG 1-SEC
        BRA     LOOP
```

PC-SCHNITTSTELLEN - Befehle

PCCOM

Allgemeines: Diese Befehle dienen dazu, vom Makro des PCMasters aus auf die seriellen Schnittstellen des PCs zuzugreifen. Es wurde darauf geachtet, dass die Ansteuerung einer seriellen PC-Schnittstelle (COMx) einigermassen kompatibel zur Ansteuerung einer INDEL 2K-SIO geschehen kann.

Vorausetzungen: Um vom PCMaster aus eine PC-Schnittstelle ansteuern zu können, müssen im PC zuerst 2 Treiber in der richtigen Reihenfolge geladen werden.

1. COMDRIVE.COM Rev. 2.03 oder höher

   Treiber der seriellen PC-Schnittstellen.

2. PCMIRQ.EXE Rev. 1.00 oder höher

   Treiber des PC-Master Interrupts.

Die Tasks, welche die PCCOM-Befehle verwenden, müssen diese zu Beginn mit '.INLCUDE PCCOM.INC' dem Makroassembler bekannt machen. Das Modul PCCOM.OBJ muss im Betriebssystem auf die Nummer gelinkt sein, die im PCCOM.INC definiert ist oder umgekehrt.

Dem Treiber PCMIRQ können mittels INDEL.INI noch 3 Parameter übergeben werden :

[PCMaster]	
IRQNumber=	Nummer des PCMaster-Interrupts (über Jumper einstellbar) Default : 11
[PCMIRQHandler]	
COMInputBufferSize=	Grösse des Input-Puffers in Byte Default : 256
COMOutputBufferSize=	Grösse des Output-Puffers in Byte Default : 256

Achtung: Der Überlauf des Input- oder Output-Puffers wird nicht überwacht, d.h. es liegt in Ihren Händen, die Grössen den Anforderungen entsprechend zu wählen.

Device Nummer: Die Devicenummern werden von 0 an gezählt, d.h. die PC-Schnittstelle COM1 hat die PCCOM-Devicenummer 0, COM2 die Nummer 1 usw. (zur Zeit werden allerdings nur 2 Schnittstellen unterstützt).

Baud Rate: Die Baud Rate wird wie folgt angegeben:

BAUD:

7				4	3	0
odd	PEn	2SB	8DB	res	BAUD-RATE	

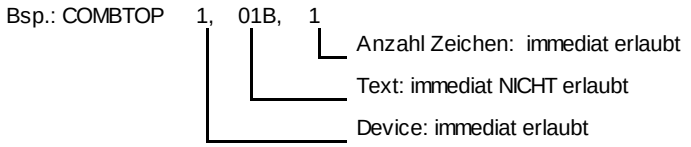
B0..3	Baud-Rate	B0..3	Baud-Rate
0	300	4	4800
1	600	5	9600
2	1200	6	19200
3	2400	7	38400

BIT	MODE	0	1
7	PARITY	EVEN	ODD
6	PARITY	DIS	EN
5	STOP BITS	1	2
4	DATA BITS	7	8
3	reserved	-	-

Errors : Übertragungsfehler jeglicher Art (z.B. Parity Error usw.) werden noch nicht erkannt, d.h. der entsprechende Task wird bei einem solchen Fehler noch nicht automatisch auf Abort gesetzt.

Steuer-Leitungen: Die Steuerleitungen DTR,RTS,CTS,DSR können mit speziellen Befehlen verändert bzw. abgefragt werden.

Hinweise:
- Bei allen PCCOM-Befehlen werden vom System die Makro-Register R70 und R71 benötigt, d.h. die Stacktiefe verringert sich um zwei Plätze.
- Da die Abarbeitung der PCCOM - Befehle völlig asynchron zum System geschieht, dürfen für gewisse Parameter keine immediate Werte verwendet werden !!!



COMSETD

COM SET Device

COMSETD BAUD/DEV

BAUD/DEV: immediat erlaubt

Erklärung : Initialisiere die PC-SIO Nr. DEV mit der Baudrate BAUD. Ab sofort hat nur die initialisierende Task Zugriff auf diese SIO.

Hinweis : Die Devicenummern werden hier von 0.. angegeben, d.h. die erste PC-SIO im System hat die Device-Nummer 0, die zweite die Nummer 1 usw.

Beispiel : Initialisiere die PC-SIO COM2 mit der Baudrate 9600,n,8,1.

COMSETD 01501

COMRESD

COM RESet Device

COMRESD DEV

DEV: immediat erlaubt

Erklärung : Gib die PC-SIO Nr. DEV wieder frei.

Hinweis : Die Devicenummern werden hier von 0.. angegeben, d.h. die erste PC-SIO im System hat die Device-Nummer 0, die zweite die Nummer 1 usw.

Beispiel : Erlöse die PC-SIO COM1 von den Schandtaten des Johanns.

COMRESD 0

COMTOP

COM Text OutPut

COMTOP DEV,TADR

DEV: immediat erlaubt
TADR: Immediat nicht erlaubt

Erklärung : Ausgabe vom Text-String TADR auf DEV.

Beispiel: Schreibe den Text ‘ Beim Zeus, wo sind die Weiber ?’ auf die PC-SIO COM2

```
COMTOP 1,@ZEUS

ZEUS:  .TXT  ‘ Beim Zeus, wo sind die Weiber ?’
```

COMBTOP

COM Block Text OutPut

COMBTOP DEV,TBLK,N

DEV: immediat erlaubt
TBLK: immediat nicht erlaubt
N: immediat erlaubt

Erklärung : Ausgabe von N Zeichen aus dem Textblock TBLK auf DEV.

Beispiel : Sende diese 5 Byte Steuersequenz an den Tintenstrahldrucker:

```
COMBTOP 1,@TBLK,5

TBLK:  .BYTE  01B,'T',00,035,'Q'
```

COMTIP

COM Text InPut

COMTIP DEV,TADR

DEV: immediat erlaubt

Erklärung : Lese Zeichen von DEV und schreibe sie nach TADR bis CR kommt oder bis ASL - Zeichen gelesen wurden. Beim TIP-Ende wird APO=0.

Beispiel : Lese die ankommenden Zeichen von COM1 in den ASC-Puffer :

COMTIP 0,ASC

COMJTIP

COM Jump Text InPut

COMJTIP DEV,TADR,SAD

DEV: immediat erlaubt
SAD: immediat erlaubt

Erklärung : Lese Zeichen von DEV genau wie bei TIP, aber springe solange auf SAD, bis CR empfangen wird oder ASL-Zeichen gelesen wurden. Um dem System zu zeigen, dass es sich um einen neuen JTIP handelt, muss aus technischen Gründen zuerst R70=0 gesetzt werden.

Beispiel : Warte, bis ein beliebiges Zeichen empfangen wird :

```
        MOV     0,ASC
        MOV     0,R70           ; neuer JTIP
WAIT:   CBR     0,<>,ASC,CONT   ; etwas getippt ?
        JTIP    1,ASC,WAIT      ; COM2 - Abfrage
CONT:
```

COMSST

COM Set line STatus

COMSST DEV,STAT

DEV: immediat erlaubt
STAT: immediat nicht erlaubt

Erklärung : Setze den Pegel der Steuerleitungen RTS und DTR.

Bit 0 → DTR, Bit 1 → RTS

Beispiel : Setze DTR=1 und RTS=0 von COM2 :

```
MOV     1,R0
COMSST  1,R0
```

COMGST

COM Get line STatus

COMGST DEV,STAT

DEV: immediat erlaubt

Erklärung : Lies den Pegel der Steuerleitungen CTS und DSR.

Bit 0 → DSR, Bit 1 → CTS

Beispiel : Lies DSR und CTS von COM1 in Register R10 :

COMSST 0,R10

Info_Master-Slave-Protokoll

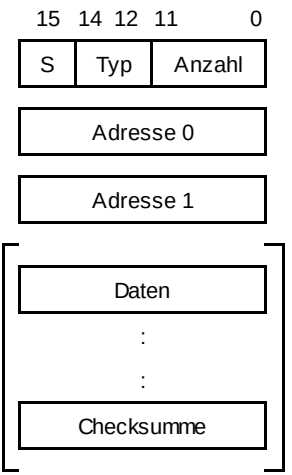
16-Bit Protokoll

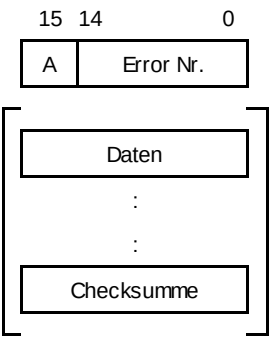
Voraussetzung: Um diese Funktionen benutzen zu können, benötigen Sie die Masterkarten mit der Softw are Rev. 2.7 oder höher, für den Info PC-Master das Modul info_com.32k und für das Rack das Modul ips_com.32k.

Beschreibung: Es w urden die neuen Funktionen F_GETB8, F_PUTB8, F_GETB16, F_PUTB16,F_GETB32 und F_PUTB32 implementiert um in Zukunft mit den verschiedenen Prozessorensystemen (Big- und Little-Endian) einen definierten Datenaustausch zu garantieren. Diese Funktionen sollen nur für Debugzw ecke oder für Parameterdefinitionen verw endet w erden.

Aufbau des Protokolls:

Master **Slave**





S: Bestimmt die Put, Get-Art.
0 = Normales Put, Get.
1 = Spezielles Put, Get.

Bei S = 1 wird die Adresse als Kommando oder als Parameter verwendet. Eine mögliche Anwendung sehen Sie im Zusammenhang mit den SIMOVERT-Funktionen.

Typ: Als Datentyp sind folgende Werte zugelassen
0 = put 8-Bit integer block
1 = put 16-Bit integer block
2 = put 32-Bit integer block

4 = get 8-Bit integer block
5 = get 16-Bit integer block
6 = get 32-Bit integer block

Anzahl: Die Anzahl zu empfangenden oder zu sendenden Daten des Typs Byte, Word oder DWords. 0 entspricht 2^12 = 4096.

Adresse 0: Low-Word der Speicheradresse oder ein Word Parameter je nach Zustand von S.

Adresse 1: Hi-Word der Speicheradresse oder ein Word Parameter je nach Zustand von S.

Daten: Byte, Word oder DWord je nach dem angegebenen Datentyps.

A: Antwortstatusbit der Slavekarte.
A = 0 bedeutet: NACK, Checksumme war falsch.
A = 1 bedeutet: ACK, Checksumme war in Ordnung.

Error Nr: Nummer des Fehlers. Null bedeutet kein Fehler.

Checksumme: Die Checksumme ist ein WORD gross. Sie wird aus dem Komplement der Wordsumme der übertragenen Worte gebildet. Das heisst Checksumme + Wordsumme = -1 (0FFFF).

Aufbau des Befehlsblockes

Adresse der Karte	(word)
Anzahl Datenelemente	(word)
Quelladresse	(dword)
Endadresse	(dword)

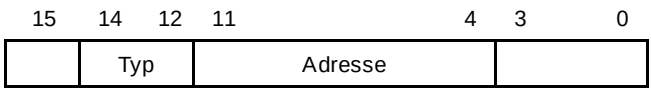
Beschreibung: Dieser Befehlblock wird für die Blockfunktionenen F_GETB8, F_PUTB8,F_GETB16, F_PUTB16,F_GETB32 und F_PUTB32 benutzt, welche einen Speicherbereich von oder zur Karte senden. Im Zusammenhang mit dem Spezialblockkennzeichen bekommen die Quell- und Endadresse eine andere Bedeutung zugewiesen. Ein Beispiel sehen Sie bei den SIMOVERT Master Drive-Funktionen.

Adresse der Karte:

Bit 14-12 Kartentyp:

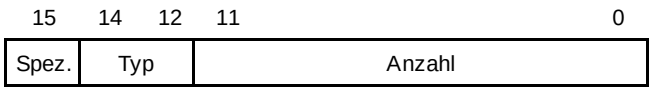
0 = res	
1 = Analog Inp	(ADC, PT100, FAD ..)
2 = IO	(16-Bit IO, Ventil-IO ..)
3 = Posi,DAC	(4K-Pos, DAC)
6 = Spezial Karten	(DEnd, UltraSchall..)

Bit 11-4 Adresse:
0-255, Wahl der Kartennummer, Achsen oder Ausgängen



Anzahl Datenelemente:

Bit 15 Kennzeichen für Spezialblock 1, sonst 0
Bit 11-0 Anzahl zu schreibende Bytes / Words / DWords
Bereich von 0-4095. (0 = 4096)

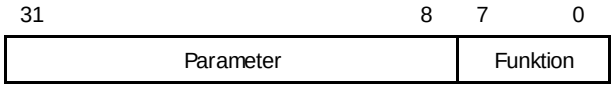


Quelladresse: Byteadresse des Puffers der zu übertragenen Bytes / Words oder DWords. (bei F_PUTB ist diese Adresse im Speicher, bei F_GETB auf der Karte)

Endadresse: Destinationsbyteadresse an der die Daten geschrieben werden. (bei F_PUTB ist diese Adresse auf der Karte, bei F_GETB im Speicher)

Besonderes: Beim Kennzeichen des Spezialblockes bekommen die Adressen eine andere Bedeutung. Bei der Funktion F_PUTB ist es die Endadresse und bei F_GETB ist es die Quelladresse. Die andere Bedeutung ist folgendermassen definiert.

Bit 31 - 8 Wird als zusätzlicher Parameterwert verwendet.
Bit 7 - 0 Bestimmen die Funktionsserviceroutine auf der Karte
0 - 127 sind für INDEL Funktionen reserviert
128 - 255 sind frei für den Anwender



Fehlercode im APO Register

APO = 1 : Leitungsunterbruch zwischen den Karten. (Link Down)
APO = 2 : Karte antwortet nicht. Vermutlich nicht angeschlossen.
APO = 3 : Checksummenfehler. Die Übertragung verlief nicht fehlerfrei.
APO = 4 : Timeout error.
APO = 5 : Fehlernummer der Antwort war <> 0.

Beschreibung: Das APO Register wird nur bei Sprung auf die ABORT-Adresse des Tasks mit der Fehlernummer besetzt; sonst bleibt es unverändert.

F_RESCOM

Reservieren eines Kanales

Info-Master: RCXP KomDes,MasterNr,'F_RESCOM'
InfoPC-Master: RCXP KomDes,0,'F_RESCOM'

Beschreibung: Reserviert einen Kommunikationskanal zum angegebenen Master. Ein Kommunikationskanal wird benötigt, um den Datenaustausch ohne Störungen durch die anderen Task sicherzustellen. Der reservierte Kanal muss wieder freigegeben werden, da nur eine beschränkte Anzahl von Deskriptoren zur Verfügung stehen. Nur beim InfoPC-Master kann der Kommunikationsdeskriptor als Pointer auf die Datenstruktur der Kommunikation benutzt werden. So kann zum Beispiel die genaue Fehlernummer, welche von der Karte zurückgeliefert wurde, herausgefunden werden.

Übergabe-
parameter: Info-Master: Masternummer, InfoPC-Master:

Rückgabe: Kommunikationsdeskriptor

Besonderes: Beim InfoPC-Master spielt die Masternummer keine Rolle. Bei Fehler: Sprung auf Abort.

Beispiel: Reservieren und wieder freigeben eines Kommunikationskanals zur Masterkarte 2.

```
P_COMCH     = R22                ; Kommunikationsdeskriptor (dword)
  :
  :
GGD    @T_RSCOM,R20              ; get descriptor of F_RESCOM
RCXP   P_COMCH,2,R20             ; reservieren eines Kanales zum
                                 ; Master 2
  :
  :
GGD    @T_FRCOM,R20              ; get descriptor of F_FRECOM
RCXP   P_COMCH,0,R20             ; Kanal wieder freigeben, wichtig !
```

```
T_RSCOM:  .TXT  'F_RESCOM'
T_FRCOM:  .TXT  'F_FRECOM'
```

F_FRECOM

Freigeben eines Kanales

RCXP KomDes,0,'F_FRECOM'

Beschreibung: Gibt den reservierten Kommunikationskanal zum Master w ieder frei

Übergabe-
parameter: Kommunikationsdeskriptor

Rückgabe: -

Besonderes: Bei Fehler: Sprung auf Abort

Beispiel: Freigeben des vorher reservierten Kommunikationskanals.
Deskriptor in P_COMCH.

```
P_COMCH  = R22                  ; Kommunikationsdeskriptor (dw ord)
     :
     :
GGD      @T_FRCOM,R20           ; get descriptor of F_FRECOM
RCXP     P_COMCH,0,R20          ; Kanal freigeben
```

```
T_FRCOM: .TXT     'F_FRECOM'
```

F_PUTBxx

8/16/32-Bit Block schreiben

Byte -Block: RCXP KomDes,Befehlsblock,'F_PUTB8'
Word -Block: RCXP KomDes,Befehlsblock,'F_PUTB16'
DWord-Block: RCXP KomDes,Befehlsblock,'F_PUTB32'

Beschreibung: Schreibt n-Bytes / -Words / -DWords von der Quelladresse im Speicher zur Endadresse in der gewählten Karte. Beim Schreiben eines Spezial-Blockes wird die Endadresse als Parameter verwendet. Ein Spezialblock-Beispiel sehen Sie im Kapitel der SIMOVERT Master Drive-Funktionen.

Übergabe-parameter: Kommunikationsdeskriptor, Befehlsblock

Rückgabe: -

Besonderes: Diese Funktionen sollen nur für Debug-Zwecke verwendet werden. Ein unkontrolliertes Schreiben kann zu einem 'Absturz' der Karte führen und sollte deshalb nur mit ausreichenden Kenntnissen angewendet werden.
Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register (1-5).

Beispiel: Setze den Hertz-Zähler des Siemens-Reglers, mit der Achsennummer 1, auf 0. Der Word-Zählerwert befindet sich auf der Adresse 07FE00040.

```
                                        ; Der Kanal zum Master wurde schon
                                        ; reserviert.
                                        ; Der Deskriptor ist in P_COMCH.

CrdAdr    = R24                         ; Adresse der Karte      (word)
Anzword   = R25                         ; Anzahl der Worte       (word)
SRCADR    = R26                         ; Quelladresse           (dword)
ENDADR    = R28                         ; Endadresse             (dword)
  :
  :
MOV       03810,CrdAdr                  ; Siemensregler, Achse 1
MOV       01,Anzword                    ; schreibe 1 word
ADDR      R0,SRCADR                     ; Puffer auf Reg R0
ASHD      1,SRCADR                      ; auf Byteadresse anpassen
MOVD      07FE00040,ENDADR              ; Adresse des Hertz-Zählers
MOV       0,R0                          ; Setzen des Zählers auf 0

GGD       @T_PUTB16,R20                 ; get descriptor of F_PUTB16
RCXP      P_COMCH,CrdAdr,R20            ; Block schreiben
```

```
T_PUTB16:       .TXT    'F_PUTB16'
```

F_GETBxx

8/16/32-Bit-Block lesen

```
RCXP   KomDes,Befehlsblock,'F_GETB8'
RCXP   KomDes,Befehlsblock,'F_GETB16'
RCXP   KomDes,Befehlsblock,'F_GETB32'
```

Beschreibung: Liest n-Bytes / -Words / -DWords von der Quelladresse in der Karte zur Endadresse im Speicher. Beim Lesen eines Spezial-Blockes wird die Quelladresse als Parameter verwendet.

Übergabe-parameter: Kommunikationsdeskriptor, Befehlsblock

Rückgabe: [Puffer]

Besonderes: Bei Fehler: Sprung auf Abort

Beispiel: Lese den Hertz-Zählerwert des Siemens-Reglers mit der Achsennummer 3. Der Word-Zählerwert befindet sich auf der Adresse 07FE00040.

```
                                    ; Der Kanal zum Master wurde schon
                                    ; reserviert.
                                    ; Der Deskriptor ist in P_COMCH.

CrdAdr    = R24                     ; Adresse der Karte     (word)
Anzword   = R25                     ; Anzahl der Worte      (word)
SRCADR    = R26                     ; Quelladresse          (dword)
ENDADR    = R28                     ; Endadresse            (dword)
  :
  :
MOV       03830,CrdAdr              ; Siemensregler, Achse 3
MOV       01,Anzword                ; lese 1 word
ADDR      R0,ENDADR                 ; Puffer auf Reg R0
ASHD      1,ENDADR                  ; auf Byteadresse anpassen
MOVD      07FE00040,SRCADR          ; Adresse des Hertz-Zählers

GGD       @T_GETB16,R20             ; get descriptor of F_GETB16
RCXP      P_COMCH,CrdAdr,R20        ; Zählerstand lesen


T_GETB16: .TXT      'F_GETB16'
```

SIMOVERT Master Drive-Funktionen

Einleitung

Beschreibung: Dieses Kapitel beschreibt eine Anwendung der INFO_SMA, INFO_SMD und INFO_SMT Karten zur Ansteuerung des SIMOVERT Master-Drive. Der Vorteil dieses Siemens-Reglers ist die Möglichkeit der Ansteuerung von verschiedenen Motortypen, ohne jedoch Änderungen an der Hardware vornehmen zu müssen. Aber um dies zu erreichen benötigt der SIMOVERT Master-Drive die spezifischen Parameterwerte für den Motor, wie z.B. Anzahl Pol-Paare, Ankerspannung und die Art des Inkrement-Gebers. Um solche spezifischen Parameterwerte ändern zu können, definierte Siemens spezielle Funktionen. Diese und weitere werden durch die Indel-SIMOVERT-Funktionen nachgebildet.

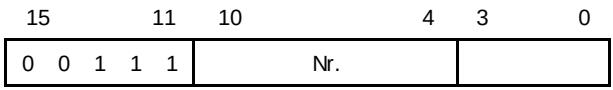
Aufbau des Befehlsblockes

Adresse der Karte	(word)
Anzahl Datenelemente	(word)
Quelladresse	(dword)
Endadresse	(dword)

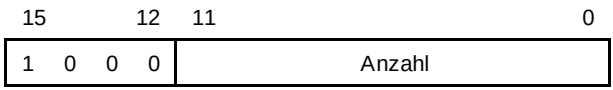
Beschreibung: Dieser Befehlsblock wird für die Blockfunktionenen F_GETB8, F_PUTB8,F_GETB16, F_PUTB16,F_GETB32 und F_PUTB32 benutzt. Im Zusammenhang mit dem Spezialblockkennzeichen bekommen die Quell- und Endadresse eine andere Bedeutung zugewiesen. Ein Beispiel sind die Indel-SIMOVERT-Funktionen, die auf den Funktionen F_PUTB16 und F_GETB16, mit Verwendung des Spezialblockes, basieren.

Besonderes: Bei Verwendung der Funktionen F_GETBxx wird die Quelladresse für die Parameter benutzt und auf die Endadresse werden die empfangenen Daten geschrieben. Bei F_PUTBxx wird die Endadresse für die Parameterübergabe benutzt.

Adresse der Karte:	Bit 15-11	Kartentyp INFO_SMx	00111b
	Bit 10-4	Achsennummer	0-127

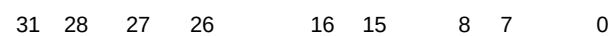


Anzahl Datenelemente:	Bit 15	Kennzeichen für Spezialblock 1	
	Bit 11-0	Anzahl zu schreibende Worte, 0 = 4096 Worte	



Quelladresse: Byteadresse des Puffers der zu schreibenden Worte

Endadresse:	Bit 31 - 28	Befehl für SIMOVERT-Regler	0 - 15
	Bit 27	Toggle Bit	0
	Bit 26-16	Parameternummer	0 - 2047
	Bit 15-8	Parameterindex	0 - 255
	Bit 7-0	Funktionswahl	10hex = Siemensspezifikation



Befehl	Tog	Para-Nr.	Index	10hex

Beispiel: Ändern der Sprachwahl beim Siemensregler mit der Achsennummer 1.

```
                                        ; Der Kanal zum Master wurde schon
                                        ; reserviert.
                                        ; Der Deskriptor ist P_COMCH.

CrdAdr   = R24                          ; Adresse der Karte          (word)
Anzword          = R25                          ; Anzahl der Worte (word)
SRCADR = R26                            ; Quelladresse       (dword)
ParSpez  = R28                          ; Parameterspez.     (dword)
   :
   :
MOV      03810,CrdAdr                   ; Siemensregler, Achse 1
MOV      08001,Anzword                  ; Spezialblock, 1 word
ADDR     R0,SRCADR                      ; Puffer auf Reg R0
ASHD     1,SRCADR                       ; auf Byteadresse anpassen
MOVD     020330010,ParSpez              ; Change PARA VALUE word
                                        ; Parameter 33h = 50 = Sprache
                                        ; Kein Index
MOV      0,R0                           ; Wahl der Sprache Deutsch
GGD      @T_PUTB16,R20                  ; get descriptor of F_PUTB16
RCXP     P_COMCH,CrdAdr,R20             ; Block schreiben


T_PUTB16:       .TXT    'F_PUTB16'
```

Übergabeparameterblock

	Wertebereich	
Siemens Befehl	(word)	1 - 15
Achsen Nummer	(word)	0 - 127
Parameter Nummer	(word)	0 - 4095
Parameter Index	(word)	0 - 255
Puffer Adresse	(dword)	word Adresse
Anzahl Bytes	(word)	0 - 4095
Parameter chara.	(word)	0 - 65535

Beschreibung: Je nach der gewählten Funktion wird nur ein Teil dieses Parameterblockes verwendet. Dieser Block dient als Basis für die folgenden beschriebenen Indel-SIMOVERT-Funktionen.

Beachte: Die Reihenfolge und deren Abstände Word/DWord der Parameter müssen immer eingehalten werden. Die Parameternummer beinhaltet auch das Toggle Bit (Bit 11). Aus diesem Grund ist der Bereich von 0-4095

Besonderes: Wenn diese Funktionen verwendet werden, so muss das Modul Info_SMD.32k mit eingelinkt werden. Für die genaueren Parameterbedeutungen und Funktionsbeschreibungen sehen Sie bitte in den Unterlagen von Siemens nach.

F_RWPARV
F_RDPARV

Lesen des Parameterwertes ohne Index
(read PARA VALUE)

Lesen eines 'word' :	RCXP	KomDes,AchsenNr,'F_RWPARV'
Lesen eines 'dword' :	RCXP	KomDes,AchsenNr,'F_RDPARV'

Beschreibung: Der Wert des Parameters wird eingelesen. Es wird keine Datentypenanpassung vorgenommen. Das heisst zum Beispiel ein Byte-Wert wird nicht richtig auf ein WORD angepasst. Das Hi-Byte ist nicht definiert und kann einen beliebigen Wert annehmen.

Übergabe-parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Pufferadresse

Rückgabe: [Buffer] word oder dword

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register. Je nach dem Einrichtemodus des Reglers können nicht alle Parameter gelesen werden.

Simovert-Funktion: 1 Read PARA VALUE

Beispiel: Lesen der Drehzahl, Parameter 2 ohne Index, von der Reglerachse 2.

```
                Befehl      = R30                   ; (word)  Befehl Block lesen/
                                                    ; schreiben
                AchsenNr    = R31                   ; (word)  Reglerachse
                Param       = R32                   ; (word)  Parameternummer
                Index       = R33                   ; (word)  Parameterindex
                Puffer      = R34                   ; (dword) Pufferadresse der Werte
                Anzahl      = R36                   ; (word)  Anzahl zu holende/
                                                    ; sendende Bytes
                Pchar       = R38                   ; (word)  Parameter- characteristics
                P_COMCH     = R26                   ; (dword) Kommunikations- deskript
                  :
                  :
                GGD         @T_RSCOM,R20            ; get descriptor of F_RESCOM
                MOV         0,Master                ; Master 0  wählen
                RCXP        P_COMCH,Master,R20      ; reserve channel

                MOV         2,AchsenNr              ; Wahl der Achse 2
                MOV         2,Param                 ; Parameter für Drehzahl
                ADDR        R0,Buffer               ; Puffer auf R0 setzen

                GGD         @T_RDPARV,R20           ; get descriptor of F_RDPARV
                RCXP        P_COMCH,AchsenNr,R20    ; read dword parameter value
                ZTOPD       DEV,POS,R0,061          ; Ausgeben der Drehzahl


T_RSCOM:        .TXT        'F_RESCOM'
T_RDPARV:       .TXT        'F_RDPARV'
```

F_WWPARV
F_WDPARV

Schreiben des Parameterwertes ohne Index
(write PARA VALUE)

Schreiben eines 'word': RCXP KomDes,AchsenNr,'F_WWPARV'
Schreiben eines 'dword': RCXP KomDes,AchsenNr,'F_WDPARV'

Beschreibung: Der Wert des Parameters wird geschrieben. Die Werte werden jedoch nicht im EEprom gesichert.

Übergabe-
parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Pufferadresse

Rückgabe: -

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 2 Change PARA VALUE word, 3 Change PARA VALUE dword

Beispiel: Hintergrundleuchten des Bedienfeldes, Parameter 54 ohne Index, von der Reglerachse 1 aktivieren.

```
                                        ; Registerdefinition und Reservieren
                                        ; eines Masters
                                        ; siehe im Beispiel 'Lesen des
                                        ; Parameterwertes ohne Index'

        MOV     1,AchsenNr              ; Wahl der Achse 1 (2. Achse)
        MOV     54,Param                ; Parameter OP-Hinterleucht.
        ADDR    R0,Buffer               ; Puffer auf R0 setzen
        MOV     0,R0                    ; Hinterleuchtung immer Aktiv

        GGD     @T_WWPARV,R20           ; get descriptor of F_WWPARV
        RCXP    P_COMCH,AchsenNr,R20    ; write word parameter value

T_WWPARV:   .TXT    'F_WWPARV'
```

F_RWPARA
F_RDPARA

Lesen des Parameterwertes mit Index
(read PARA VALUE ARRAY)

Lesen eines 'word': RCXP KomDes,AchsenNr,'F_RWPARA'
Lesen eines 'dword': RCXP KomDes,AchsenNr,'F_RDPARA'

Beschreibung: Der Indexwert des Parameters wird eingelesen. Es wird keine Datentypenanpassung vorgenommen.

Übergabe-parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Pufferadresse

Rückgabe: [Buffer] word oder dword

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 6 Read PARA VALUE ARRAY

Beispiel: Lesen Motor-Pol-Paarzahl, Parameter 109 Index 1, der Reglerachse 1

```
                                    ; Registerdefinition und Reservieren
                                    ; eines Masters
                                    ; siehe im Beispiel 'Lesen des
                                    ; Parameterwertes ohne Index'

        MOV     1,AchsenNr          ; Wahl der Achse 1
        MOV     109,Param           ; Parameter für Mot.Pol-Paarzahl
        MOV     1,Index             ; 1. Wert Lesen
        ADDR    R4,Buffer                   ; Puffer auf R4 setzen

        GGD     @T_RWPARA,R20       ; get descriptor of F_RWPARA
        RCXP    P_COMCH,AchsenNr,R20 ; read word parameter value array
        ZTOP    DEV,POS,R4,040      ; Ausgeben der Pol-Paarzahl

T_RWPARA:   .TXT    'F_RWPARA'
```

F_WWPARA
F_WDPARA

Schreiben des Parameterwertes mit Index
(write PARA VALUE ARRAY)

Schreiben eines 'word': RCXP KomDes,AchsenNr,'F_WWPARA'
Schreiben eines 'dword': RCXP KomDes,AchsenNr,'F_WDPARA'

Beschreibung: Der Indexwert des Parameters wird geschrieben. Die Werte werden nicht im EEprom gesichert.

Übergabe-parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Pufferadresse

Rückgabe: -

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 7 Change PARA VALUE ARRAY word, 8 Change PARA VALUE ARRAY dword

Beispiel: TRC-Abtastzeit des Kanals 3 auf die 50. fache Grundabtastzeit setzen.
Reglerachse 1, Parameter 739, Index 3 für Kanal 3.

```
                                    ; Registerdefinition und Reservieren
                                    ; eines Masters
                                    ; siehe im Beispiel 'Lesen des
                                    ; Parameterwertes ohne Index'

        MOV   1,AchsenNr            ; Wahl der Achse 1
        MOV   739,Param             ; Parameter für TRC-Abtastzeit
        MOV   3,Index               ; Abtastzeit für Kanal 3
        MOV   50,R0                 ; 50. fache Grundabtastzeit
        ADDR  R0,Buffer                     ; Puffer auf R0 setzen

        GGD   @T_WWPARA,R20         ; get descriptor of F_WWPARA
        RCXP  P_COMCH,AchsenNr,R20  ; write word parameter value array

T_WWPARA: .TXT  'F_WWPARA'
```

F_RCPARA
F_WCPARA

Lesen / Schreiben des Parameterwertes
(read/write characteristics value)

Lesen eines Wertes:	RCXP	KomDes,AchsenNr,'F_RCPARA'
Schreiben eines Wertes:	RCXP	KomDes,AchsenNr,'F_WCPARA'

Beschreibung: Diese beiden Funktionen vereinfachen die Handhabung der vorher beschriebenen Funktionen, indem sie die Parameterwerte entsprechend ihres Datentypes auf ein dword konvertieren und die richtige Funktion (mit und ohne Index) anhand der Parametercharakteristik aufrufen.

Übergabe-parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Pufferadresse, Parametercharakteristik

Rückgabe: read [Buffer] dword, write -

Besonderes: Parametercharakteristik muss vor dem Funktionsaufruf definiert sein. Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Beispiel:

```
                                        ; Registerdefinition und Reservieren
                                        ; eines Masters
                                        ; siehe im Beispiel 'Lesen des
                                        ; Parameterwertes ohne Index'

        MOV     0,AchsenNr              ; Wahl der Achse 0
        MOV     15,Param                ; Parameter für Drehmoment
        MOV     1,Index                 ; Parameter characteristic
        ADDR    R0,Buffer                       ; Puffer auf R0 setzen

        GGD     @T_RPARAD,R20           ; get descriptor of F_RPARAD
        RCXP    P_COMCH,AchsenNr,R20    ; read parameter descriptor

        MOV     R0,Pchar                ; Parametercharakteristik übernehmen

        GGD     @T_RCPARA,R20           ; get descriptor of F_RCPARA
        RCXP    P_COMCH,AchsenNr,R20    ; read parameter Value

        ZTOPD   DEV,POS,R0,091          ; Drehmoment ausgeben


T_RPARAD:   .TXT    'F_RPARAD'
T_WWPARA:   .TXT    'F_RCPARA'
```

F_RPARAD
F_WPARAD

Lesen/Schreiben der Parameterbeschreibungen
(read/write parameter descriptor)

Deskriptor lesen:	RCXP KomDes,AchsenNr,'F_RPARAD'
Deskriptor schreiben:	RCXP KomDes,AchsenNr,'F_WPARAD'

Beschreibung: Je nach dem Wert des Indexes können die verschiedenen Informationen über einen Parameter abgerufen werden, wie zum Beispiel der Skalierungsfaktor oder die Anzahl der Indexelemente.

Übergabe-parameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Pufferadresse, Anzahl Bytes zu Lesen / Schreiben

Rückgabe: read [Buffer], write -

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 4 Read PARA DESCR element, 5 Change PARA DESCR element

Beispiel

```
                                        ; Registerdefinition und Reservieren
                                        ; eines Masters
                                        ; siehe im Beispiel 'Lesen des
                                        ; Parameterwertes ohne Index'

        MOV     1,AchsenNr              ; Wahl der Achse 1
        MOV     952,Param               ; Parameter für Anzahl Störfälle
        MOV     1,Index                 ; Parameter characteristic
        ADDR    Pchar,Buffer            ; Puffer auf Pchar setzen
        MOV     2,Anzahl                ; Lesen 1 word = 2 Bytes

        GGD     @T_RPARAD,R20           ; get descriptor of F_RPARAD
        RCXP    P_COMCH,AchsenNr,R20    ; read parameter descriptor

T_RPARAD: .TXT  'F_RPARAD'
```

F_RUCOM
F_WUCOM

Lesen / Schreiben vom Anwender gewählter Kommandos
(read/write user command)

Command lesen: CXP KomDes,Befehl,'F_RUCOM'
Command schreiben: RCXP KomDes,Befehl,'F_WUCOM'

Beschreibung: Mit diesen beiden Funktionen können alle Siemensbefehle aufgerufen werden. Sie vereinfachen die Funktionen F_PUTB16 und F_GETB16.

Übergabe-parameter: Kommunikationsdeskriptor, Befehl, Achsennummer, Parameternummer, Parameterindex, Pufferadresse, Anzahl Bytes zu Lesen / Schreiben.

Rückgabe: read [Buffer], write -

Besonderes: Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 0-15 sind möglich. Die Funktionen 11-14 speichern die Werte im EEPROM ab. Die Funktionen 2,3,5,7 und 8 ändern die Werte nur temporär ab. Das bedeutet nach Ausschalten des Reglers sind die Daten verloren.

Beispiel: Ersatz für read parameter descriptor:

```
                                        ; Registerdefinition und Reservieren
                                        ; eines Masters
                                        ; siehe im Beispiel 'Lesen des
                                        ; Parameterwertes ohne Index'

        MOV     4,Befehl                ; read PARA DESCRIPTOR
        MOV     1,AchsenNr              ; Wahl der Achse 1
        MOV     952,Param               ; Parameter für Anzahl Störfälle
        MOV     1,Index                 ; Index für Parameter characteristic
        ADDR    Pchar,Buffer            ; Puffer auf Pchar setzen
        MOV     2,Anzahl                ; Lesen 1 word = 2 Bytes

        GGD     @T_RUCOM,R20            ; get descriptor of F_RUCOM
        RCXP    P_COMCH,Befehl,R20      ; read user selected command

T_RUCOM: .TXT   'F_RUCOM'
```

F_RTEXT
F_WTEXT

Lesen / Schreiben von Text
(read or write text refer to index word)

Text lesen: RCXP KomDes,AchsenNr,'F_RTEXT'
Text schreiben: RCXP KomDes,AchsenNr,'F_WTEXT'

Beschreibung: Mit diesen Funktionen können zusätzliche Informationen über einen Parameterwert in Erfahrung gebracht werden.

Übergabeparameter: Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Pufferadresse, Anzahl Bytes zu Lesen / Schreiben

Rückgabe: read [Buffer], write -

Besonderes: Beim Lesen wird der Buffer mit einem 0 Byte abgeschlossen. Bei Fehler: Sprung auf Abort. Abbruchgrund im APO Register.

Simovert-Funktion: 15 Read or change text

Beispiel:

```
                                        ; Registerdefinition und Reservieren
                                        ; eines Masters
                                        ; siehe im Beispiel 'Lesen des
                                        ; Parameterwertes ohne Index'

        MOV     1,AchsenNr              ; Wahl der Achse 1
        MOV     947,Param               ; Parameter für Störtextliste
        MOV     35,Index                ; Index für 'Ext. Fehler1'
        ADDR    R0,Buffer               ; Puffer auf R0 setzen
        MOV     16,Anzahl               ; Lesen 16 Bytes

        GGD     @T_RTEXT,R20            ; get descriptor of F_RTEXT
        RCXP    P_COMCH,AchsenNr,R20    ; read Text refer to index word

        TOP     DEV,POS,0[Buffer]       ; Text ausgeben

T_RTEXT:.TXT    'F_RTEXT'
```

INFO_SIO - Befehle

INFO_SIO

Allgemeines: Von der Standard-Firmware werden 2 INFO_SIO - Karten unterstützt, d.h. 4 SIO-Kanäle. Diese Kanäle werden über die Device - Nummern 0..3 angesprochen.

Vorausetzungen: INFO-PC Master Firmware ab Rev. 2.88.

Achtung: Die INFO_SIO - Befehle wurden mit Hilfe von .NEWINST des Makro Assemblers implementiert. Die Befehle sind im Modul SIO_NSIO implementiert, d.h. dem Equal MOD_NSIO in der Datei SIO_NSIO.INC muss die Modul-Nummer von SIO_NSIO zugewiesen werden (siehe Datei INFO.IND).

DataFrame: Das Uebertragungsformat wird in der üblichen INDEL - Form angegeben. Eine Ausnahme bildet die Baudrate: sie ist bis 115200 Baud frei wählbar, somit können auch Geräte mit exotischen Baudraten angesprochen werden (siehe Befehl SIOSETD).

15	14	13	12	11	10	9	8 7 0
odd	PEn	2SB	8DB	xon	RS	res	Device

Bit	Mode	0	1	
10	RS422 driver	EN	EN	beim Senden
11	XON/XOFF	DIS	EN	
12	DATA BITS	7	8	
13	STOP BITS	1	2	
14	PARITY	DIS	EN	
15	PARITY	EVEN	ODD	

Errors: Der Johann springt bei folgenden Errors auf seine Abort-Adresse (die Error - Nummer steht im APO - Register):

INFO_SIO Kommunikations - Fehler
001 LinkDown (kein geschlossener INFO-Link vorhanden)
002 INFO_SIO - Karte meldet sich nicht
003 Checksummen - Fehler (der INFO-Link ist schlecht, die Kommunikation zur INFO_SIO - Karte wird stark gestört -> Error-Counter überprüfen)
004 Timeout (bei der Kommunikation mit der INFO_SIO-Karte ist ein Timeout aufgetreten)
005 interner Kommunikationsfehler (call INDEL)

INFO_SIO Protokoll - Fehler

010	Put/Get ohne vorher ein Protokoll gewählt zu haben (sollte nur bei Eigenprotokollimplementationen vorkommen)
020	Abort Character empfangen (nur bei SIOTIP)
021	Framing Error
022	Parity Error
023	Overrun Error
024	Input-Buffer overflow
025	DSR bei SIOSETD = 0
026	Timeout bei SIOTOP, SIOBTOP, SIOTIP oder SIOBTIP

Debug-Hinweis: Die INFO_SIO - Befehle wurden z.T. mit Makro implementiert, d.h. ein SingleStep auf einem INFO_SIO - Befehl führt Sie und Ihren Johann irgendwo in die tiefen Systemniederungen. Wir empfehlen deshalb, nicht mit SingleStep die INFO_SIO - Befehle zu debuggen, sondern unmittelbar nach dem Befehl einen Breakpoint zu setzen und mit F9 (go) den Befehl abzuarbeiten.

Sonderzeichen: Eine Text-Eingabe wird dann abgeschlossen, wenn entweder die gewünschte Anzahl Zeichen (SIOBTIP) oder der gewünschte Endstring (SIOTIP) eingetroffen ist. Wurde bei SIOTIP kein Endstring definiert, so verhält sich dieser kompatibel zum TIP auf die 2K-SIO, d.h. der LF (0A) und der 00 - Character werden ignoriert und die Eingabe abgeschlossen, sobald ein CR (0D) eingetroffen ist. Wird bei SIOTIP der Abort-Character 'ABC' empfangen, so wird der Eingangspuffer gelöscht und der Johann springt auf Abort.

Puffer: Die INFO_SIO hat pro Kanal 2KByte Eingangs und 2KB Ausgangspuffer. Folgende max. Blocklängen sind zulässig :

SIOTOP	512 Bytes
SIOBTOP	512 Bytes
SIOTIP	255 Bytes (da ASL nur ein Byte gross ist)
SIOBTIP	512 Bytes

OUT: Der Ausgangspuffer nimmt immer Daten von SIOTOP oder SIOBTOP auf, solange er nicht voll ist, auch wenn der Ausgabekanal mit CTS oder XOFF gebremst wird. Er kann nur mit SIOSETD gelöscht werden.

INP : Der Eingangspuffer ist immer empfangsbereit, solange er nicht voll ist. Der TIP nimmt immer nur Daten aus dem Eingangspuffer. Dieser kann mit SIOSETD gelöscht werden. Er wird ebenfalls bei FRAMING, PARITY und OVERRUN -Error gelöscht.

DTR: Der DTR-Ausgang dient zum Bremsen des Eingabe-Gerätes. Da der Eingangs-Puffer immer offen ist, bremst der DTR (-15V) nur, wenn dieser Puffer bis auf 256 Zeichen voll ist. Wird dieser Wert wieder unterschritten, so wird auch der DTR wieder aktiv (+15V).

RTS: Der RTS-Ausgang wird immer aktiv (+15V), wenn sich Daten im Ausgangs-Puffer befinden.

CTS: Ueber den CTS-Eingang kann die Ausgabe gebremst werden. Sobald er inaktiv (-15V) ist, wird die Ausgabe gestoppt (angefangenes Zeichen wird noch gesendet). Wird dies nicht gebraucht: CTS -> +5..15V.

DSR: Ist bei SIOSETD der DSR-Eingang inaktiv (-15V), so springt der Johann auf seine ABORT-Adresse. Dies dient der Erkennung, ob überhaupt ein Ausgabe-Gerät angeschlossen und zum Empfang der Daten bereit ist (Papier-Ende). Danach funktioniert der DSR wie die CTS-Leitung.
Wird dies nicht gebraucht: DSR -> +5..15V.

DCD: Dieser Eingang dient bei Modem-Betrieb als Data Carrier Detect. Wenn der DCD inaktiv (-15V) ist, wird der Eingangs-Kanal abgeschaltet und es können keine falschen oder undefinierten Zeichen empfangen werden. Ist er aktiv (+15V), so wird der Eingang ganz normal durchgeschaltet.
Wird dies nicht gebraucht: DCD -> +5..15V.

XON/XOFF: Beim XON/OFF-Betrieb werden neben den Steuerleitungen auch noch XON (011) und XOFF (014) verarbeitet. Ist XON/XOFF ausgeschaltet, werden diese Zeichen wie andere behandelt. Ist XON/XOFF eingeschaltet, merkt der Benutzer davon überhaupt nichts und die empfangenen XON/XOFF kommen nicht in den INP-Puffer.

- XOFF wird gesendet, wenn der INP-Puffer bis auf 256 Zeichen voll wird. Wird XOFF empfangen, so wird die Ausgabe sofort gestoppt (angefangenes Zeichen wird noch gesendet).
- XON wird gesendet, wenn vorher XOFF gesendet wurde und der INP-Puffer wieder Platz hat; ebenfalls nach Power-On beim ersten SIOSETD (nur wenn XON/XOFF gewählt!). Wird XON empfangen und es stehen noch Daten im OUT-Puffer, wird das Senden fortgesetzt.

Beachte: Der XON/XOFF - Betrieb wird in der aktuellen INFO_SIO Revision noch nicht unterstützt.

Beachte: Die Steuerleitungen CTS,DSR,DCD werden auch bei 20mA, RS422 und XON/XOFF-Betrieb verarbeitet.
Werden sie nicht benötigt: alle auf +5..15V !

SIOSETD

SIO SET Device

SIOSETD DEV, BAUD:D

Erklärung : Mit dem SIOSETD - Befehl wird der SIO-Kanal DEV für diesen Johann reserviert. Im High-Byte von DEV können Sie das DataFrame definieren (siehe Einleitung), im Low-Byte wird die Kanalnummer angegeben (0..3).
Mit BAUD können Sie eine beliebige Baudrate wählen (bis 115200 Baud), auch exotische wie z.B. 1326 Baud sind erlaubt. Beachten Sie, dass die Baudrate als Doppelwort angegeben werden muss.

Ist der DSR - Eingang beim Ausführen von SIOSETD inaktiv (-15V), so springt der Johann auf Abort (APO = 025).

Mit SIOSETD werden Eingangs- und Ausgangspuffer gelöscht.

Beispiel: Reserviere den SIO-Kanal 0 auf Karte 1 (->Kanal 2). Initialisiere das Uebertragungsformat auf 19200 Baud, E,8,1.

SIOSETD 0C002,19200:D

SIORESD

SIO RESet Device

SIORESD DEV

Erklärung : Gibt einen, mit SIOSETD reservierten, SIO-Kanal wieder frei.

Beispiel : Gib Kanal 2 wieder frei

SIORESD 2

SIOTOP

SIO Text OutPut

SIOTOP DEV,TADR,EADR,TIMOUT:D

Erklärung : Ausgabe von Text-String TADR (mit 00 char abgeschlossen) + EADR (mit 00 char abgeschlossen) auf DEV.

Mit EADR kann eine beliebige Abschlusszeichenfolge definiert werden, z.B. CR,LF ...

SIOTOP schreibt die Zeichenfolge in den Ausgabepuffer und kehrt sofort zurück. Ist der Puffer jedoch voll und wird nicht abgebaut (weil z.B. CTS oder DSR inaktiv sind -> es kann nicht gesendet werden), so kann mit TIMOUT eine maximale Wartezeit in ms angegeben werden. TIMOUT = 0 bedeutet -> keine Zeitüberwachung.

Es wird empfohlen immer eine Timeoutzeit anzugeben, um unnötiges Blockieren des Tasks zu verhindern.

Hinweis: max. Länge des Ausgabestrings (TADR+EADR) = 512 Zeichen

Beachte: Die Timeoutzeit muss als Doppelwort angegeben werden. Die Timeoutüberwachung wird erst ab INFO_SIO Rev. 1.10 und INFO-PCMaster Firmware Rev. 2.91 unterstützt.

Beispiel : Schreibe den Text 'Oh, du meine SIO' auf Kanal 1 und schliesse ihn mit CRLF ab. Beachten Sie, dass 0A0D als Doppelwort definiert wird; damit wird automatisch ein 00 char erzeugt.

```
SIOTOP   1,@TEXT,0A0D:D,10:D

TEXT:    .TXT      'Oh, du meine SIO'
```

SIOBTOP

SIO Block Text OutPut

SIOBTOP DEV,TBLK,N,TIMOUT:D

Erklärung : Ausgabe von N Zeichen aus dem Text-Block TBLK auf den Kanal DEV. Mit diesem Befehl können alle Zeichen von 00..FF ohne Einschränkungen ausgegeben w erden.

SIOBTOP schreibt die Zeichenfolge in den Ausgabepuffer und kehrt sofort zurück. Ist der Puffer jedoch voll und w ird nicht abgebaut (w eil z.B. CTS oder DSR inaktiv sind -> es kann nicht gesendet w erden), so kann mit TIMOUT eine maximale Wartezeit in ms angegeben w erden. TIMOUT = 0 bedeutet -> keine Zeitüberw achung.

Es w ird empfohlen immer eine Timeoutzeit anzugeben, um unnötiges Blockieren des Tasks zu verhindern.

Hinw eis: Die Anzahl Zeichen N ist auf max. N=512 beschränkt.

Beachte: Die Timeoutzeit muss als Doppelw ort angegeben w erden.Die Timeoutüberw achung w ird erst ab INFO_SIO Rev. 1.10 und INFO-PCMaster Firmw are Rev. 2.91 unterstützt.

Beispiel : Sende diese 5-Byte Steuersequenz an den Drucker, der an Kanal 1 hängt.

```
TBLK:    .BYTE    01B,'T',000,035,'q'

SIOBTOP 1,@TBLK,5,10:D
```

SIOTIP

SIO Text InPut

SIOTIP DEV,TADR,EADR,TIMOUT:D

Erklärung : Lese Zeichen von DEV nach TADR, bis der TIP-Ende-Erkennungs-String eintrifft oder bis TIMOUT ms abgelaufen sind.

Mit EADR kann ein beliebiger Ende-Erkennungs-String definiert werden, z.B. CR,LF .... Ohne Ende-String-Angabe verhält sich der Befehl wie der TIP auf die 2K-SIO (0A und 00 char werden ignoriert, 0D gilt als Ende-Erkennung).

Beachte: Die Timeoutzeit muss als Doppelwort angegeben werden.

Beispiel : Lese solange Zeichen von Kanal 0 in den ASC-Puffer, bis CRLF eintrifft oder 2 sec abgelaufen sind.Beachten Sie, dass 0A0D als Doppelwort definiert wird; damit wird automatisch ein 00 char erzeugt.

SIOTIP 0,ASC,0A0D:D,2000:D

SIOSTAT

SIO STATus

SIOSTAT DEV,STRUCT

Erklärung : Lese den SIO-Kanalstatus nach STRUCT

STRUCT : offset 0 Modem Status Register (MSR)
offset 1 Aktuelle Anzahl Zeichen im Input-Puffer
offset 2 Aktuelle Anzahl Zeichen im Output-Puffer

MSR MSR - Definition von 16550 UART.
Bit 4 CTS
Bit 5 DSR
Bit 7 DCD

Beispiel : Lese Status von Kanal 2. Schreibe MSR nach R0, Input-Puffer nach R1 und Output-Puffer nach R2.

```
SIOSTAT 2,R0
```

SIOBTIP

SIO Block Text InPut

SIOBTIP DEV,TBLK,N,TIMOUT:D

Erklärung : Lese Zeichen von DEV, bis entweder N Zeichen gelesen wurden oder bis TIMOUT ms abgelaufen sind. Mit diesem Befehl können alle Zeichen von 00..FF ohne Einschränkungen eingelesen werden.

Beachte: Die Timeoutzeit muss als Doppelwort angegeben werden.

Beispiel : Lese 6 Zeichen von Kanal 3 nach R00..R02, warte max. 500ms.

SIOBTIP 3,R0,6,500:D

PSEUDO-Befehle

PSEUDO BEFEHLE

```
TITEL:                  .TITLE      "**- PSEUDO BEFEHLE -**"
                        .SUBTITLE   "- Allgemein -"

LISTING:                .LINE 85                    ; 85 Zeilen / Seite
                        .NOLIST                     ; Listing aus
                        .LIST                       ; Listing ein
                        .EJECT                      ; Neue Seite

FILE:                   .INCLUDE    NEXTFIL         ; File zuladen

ADRESSE:                .LOC        01000           ; Programm-Start

ZUWEISUNGEN:  WT1:      .EQU    012345678           ; Mit .EQU oder =
              WT2       =       -WT1                ; Wird in DW abgelegt
              FW1       =       123.456             ; Wird in LONG abgelegt
              FW2:      .EQU    -123.456
              PI        =       3.1415926536
              RL1:      .EQU    033(R77)            ; R77 - relativ
              RL2       =       044(R22)            ; R22 - relativ
              BUFFER    =       ASC
              NAME      =       R11                 ; NAME = R11

ZAHLEN:       DZ1:      .EQU    999                 ; Dezimal
              DZ3:      .EQU    1E3                 ; Exponent
              HX1:      .EQU    0ABCD               ; Hex
              FL1:      .EQU    123.456E15          ; Floating Point
              FL2:      .EQU    -123.456E-15
              FL3:      .EQU    2.0                 ; Dez-Punkt = Floating!

KOORDINATEN:  YYXX:     .EQU    1234|5678           ; DW aus zwei Dez-Zahlen
                                                    ; (| = ALT124 )

KONSTANTEN:   LABEL:    .BYTE       1,2,' A' ,4,5
                        .WORD       01234,05678,START
                        .DOUBLE     012345678,087654321
                        .FLOAT      1.2,PI,3.4E5
                        .LONG       6.7,PI,-8.9E-10

BLÖCKE:                 .BLKB       AnzahlBytes     ; Byteblock
                        .BLKW       AnzahlWorte     ; Wortblock
                        .BLKD       AnzahlDoppelWorte ; Doppelw ortblock

TEXT:                   .TXT        '^
              <000009>  Text mit    CR/LF
              <000009>  und ohne    CR/LF^
              <000009>  New-Line'
                        .TXT        ' <09>Sonderzeichen<0D><0A>'
```

INDEX

ASCII-SET

Spezial-Zeichen

Dez	Hex	Label	Definition
0	00	NUL	Null
1	01	SOH	Start of heading
2	02	STX	Start of text
3	03	ETX	End of text
4	04	EOT	End of transmission
5	05	ENQ	Enquiry
6	06	ACK	Acknowledge
7	07	BEL	Rings the bell
8	08	BS	Backspace
9	09	HT	Horizontal tab
10	0A	LF	Line feed
11	0B	VF	Vertical tab
12	0C	FF	Form feed
13	0D	CR	Carriage return
14	0E	SO	Shift out
15	0F	SI	Shift in
16	10	DLE	Data link escape
17	11	DC1	Device control 1
18	12	DC2	Device control 2
19	13	DC3	Device control 3
20	14	DC4	Device control 4
21	15	NAK	Not acknowledge
22	16	SYN	Synchronus idle
23	17	ETB	End of trans block
24	18	CAN	Cancel
25	19	EM	End of medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	File separator
29	1D	GS	Group separator
30	1E	RS	Record separator
31	1F	US	Unit separator

FCV-Charakter

Dez	Hex	0 00	16 10	32 20	48 30	64 40	80 50	96 60	112 70	128 80	144 90	160 A0	176 B0	192 C0	208 D0	224 E0	240 F0
0	0				0	@	P	`	p	Ç	É	á					
1	1			!	1	A	Q	a	q	ü	æ	í					
2	2			"	2	B	R	b	r	é	Æ	ó					
3	3			#	3	C	S	c	s	â	ô	ú					
4	4			$	4	D	T	d	t	ä	ö	ñ					
5	5			%	5	E	U	e	u	à	ò	Ñ					
6	6			&	6	F	V	f	v	å	û	ª					
7	7				7	G	W	g	w	ç	ù	º					
8	8			(	8	H	X	h	x	ê	ÿ	¿					
9	9			)	9	I	Y	i	y	ë	Ö						
10	A			*	:	J	Z	j	z	è	Ü						
11	B			+	;	K	[	k	{	ï		½					
12	C			,	<	L	\	l	|	î	£	¼					
13	D			-	=	M	]	m	}	ì		¡					
14	E			.	>	N	^	n	~	Ä		«					
15	F			/	?	O	_	o	_	Å		»					