

# INDEL BETRIEBSSYSTEM

ISM - 6.0

Version: IPS-32

## Referenz Manual



## Inhaltsverzeichnis

<b>Einleitung</b>	<b>11</b>
Allgemeines .....	12
<b>Beispiel</b>	<b>15</b>
Vorgabe .....	16
EQUAL .....	17
TASK0 .....	19
TASK1 .....	22
TASK2 .....	23
<b>TOOLS</b>	<b>25</b>
INDEL.INI .....	26
MSI .....	31
TRANS .....	32
CONFIG .....	34
<b>RAM-AUFTEILUNG</b>	<b>35</b>
GCPU-15 .....	36
Gx_CPU-15 und Gx_CPU-25 .....	37
Fx_CPU-25 .....	38
<b>REGISTER</b>	<b>39</b>
Task Register .....	40
Task-Kontroll Register .....	41
ASCII-Kontroll Register .....	42
<b>ADRESSIERUNGSARTEN</b>	<b>43</b>
Befehlsaufbau .....	44
Adressierungsarten .....	45
Immediate .....	46
FLOATING POINT Immediate .....	47
Adresse .....	48
Adresse mit Register-Offset .....	49
Indirekt (Adresse mit Register-Offset) .....	50
Pointer indexed .....	51
Indirekt (Pointer indexed) .....	52
Register .....	53
Register indexed (mit Offset) .....	54
Register indexed mit Auto-Increment/Decrement .....	55

Register indexed mit Register Offset .....	56
ASCII-Puffer .....	57
INPUT-Base .....	58
OUTPUT-Base .....	59
FLAG-Base .....	60
<b>Globale Adressen - Befehle</b>	<b>61</b>
Get Global Address .....	62
Get Global Pointer .....	63
Get Global Deskriptor .....	64
<b>TASK-KONTROLL-Befehle</b>	<b>65</b>
EXeQute .....	66
Get Program NumbeR .....	67
Johann Self Kill .....	68
JOhann Kill .....	69
Johann Self ABort .....	70
JOhann ABort .....	71
DELAY .....	72
<b>SPRUNG-Befehle</b>	<b>73</b>
BRanch Always .....	74
BRanch to Sub-Routine .....	75
JuMP .....	76
Jump to Subroutine .....	77
Jump indirect Address-Table .....	78
Jump to Subroutine indirect Address-Table .....	79
Return To Mainprogram .....	80
Jump EXternal .....	81
load Registers and jump EXternal .....	82
Call External Procedure .....	83
load Registers and Call External Procedure .....	84
<b>BIT-Befehle</b>	<b>85</b>
Test and BRanch if bit = 0 .....	86
Test and BRanch if bit = 1 .....	87
Test and HaIT if bit = 0 .....	88
Test and HaIT if bit = 1 .....	89
Test and HaIT if bit = 0 and branch if Timeout .....	90
Test and HaIT if bit = 1 and branch if Timeout .....	91
Set BIT .....	92
Clear BIT .....	93
Invert BIT .....	94
Move BIT .....	95
Move INvert Bit .....	96
Find First Set Bit .....	97

Set Bit Range .....	98
Load Bit Range .....	99
<b>MOVE-Befehle</b>	<b>101</b>
MOVE .....	102
eXCHange .....	103
Move Zero extended .....	104
Move signum eXtended .....	105
Move Byte .....	106
Dump .....	107
<b>LOGIK-Befehle</b>	<b>109</b>
AND .....	110
OR .....	111
eXclusive OR .....	112
COMplement .....	113
Logic SHift .....	114
Arithmetic SHift .....	115
ROTate .....	116
<b>ARITHMETIK-Befehle</b>	<b>117</b>
ADDition .....	118
SUBtraction .....	119
MULTiplikation .....	120
DIVision .....	121
Quotient .....	122
MODulus .....	123
REMAinder .....	124
SQUare Root .....	125
ABSolute .....	126
NEGate .....	127
<b>CONVERT-Befehle</b>	<b>129</b>
Floating to Integer .....	130
Integer to Floating .....	131
Hex Decimal ConVert .....	132
Decimal Hex ConVert .....	133
ADDRes calculation .....	134
<b>VERGLEICHS-Befehle</b>	<b>135</b>
Compare and BRanch absolute .....	136
Compare and BRanch signed .....	137

Compare and BRanch floating .....	138
-----------------------------------	-----

## **TEXT IN/OUT-Befehle** **139**

VIDEO FCV .....	140
SPLIT-SCREEN .....	141
WINDOW .....	142
S-I/O 32 .....	143
CENTRONICS .....	145
2 Kanal S-I/O .....	146
SET Device .....	150
INIt Device .....	151
RESet Device .....	152
CLear Device .....	153
Clear TIP .....	154
Carriage Return / Line Feed .....	155
PUT Device to screen-save n .....	156
GET Device from screen-save n .....	157
Text OutPut .....	158
Gross-Text OutPut .....	159
! Nur VIDEO ! .....	159
Balken TOP .....	160
Block Text OutPut .....	161
Multi Text OutPut .....	162
Horizontal Text OutPut .....	163
Vertikal Text OutPut .....	166
Zahlen Text OutPut .....	167
Hex-Zahlen Text OutPut .....	169
Text InPut .....	170
Jump Text InPut .....	171
Block Text InPut .....	172

## **GRAPHIK-Befehle** **173**

GRAPHIK FGV .....	174
ACRtC-MODI .....	175
ORiGin .....	176
DOT .....	177
LINE .....	178
PolyLINE .....	179
Continuous (poly)LINE .....	180
ReCTangle .....	181
Filled ReCTangle .....	182
Continuous Filled ReCTangle .....	183
CIrcLe .....	184
ARC (Kreissegment) .....	185
Continuous ARC .....	186
ELliPSe .....	187
Continuous Ellips ARC .....	188
PaTterN .....	189

Continuous PaTterN .....	190
PAINT .....	191
Graphic CoPY .....	192
WRite Drawing Parameter Register .....	193
WRite PaTern Register .....	194

**ASCII-Befehle 195**

Ascii → BinaRy .....	196
heX-Ascii → BinaRy .....	198
Ascii CoMPare .....	199

**TIME-Befehle 201**

get/set TIME .....	202
--------------------	-----

**FLOPPY-Befehle 203**

FLOPPY-FORMATE .....	204
DIRECTORY .....	205
PFAD .....	206
Disk ERRORS .....	207
DELETE file .....	208
RENAME file .....	209
DISK space .....	210
FILE parameter .....	211
Get ATtRibute .....	212
Set ATtRibute .....	213
READ file .....	214
WRITE file .....	215
APPEND file .....	216
ReaD BloCk .....	217
WRite BLoCk .....	218
COPY file .....	219
Disk COPY .....	220
CHange DIRectory .....	221
MaKe DIRectory .....	222
ReMove DIRectory .....	223
DIRectory .....	224
PATH .....	225
FORMAT disk .....	226

**MASTER/SLAVE- Protokoll 227**

MASTER/SLAVE .....	228
MASTER .....	229
SET Slave .....	230
SET Master .....	231
PUT data .....	232
GET data .....	233

PROTOKOLL .....	234
PUT - Beispiel .....	236
GET - Beispiel .....	237
<b>3964R-Protokoll</b>	<b>239</b>
3964R .....	240
SET3964R Master .....	242
SET 3964R Slave .....	243
AusgabeDaten 3964R .....	244
EingabeDaten 3964R .....	245
<b>Info_Master-Slave-Protokoll</b>	<b>247</b>
16-Bit Protokoll .....	248
Aufbau des Befehlsblockes .....	250
Fehlercode im APO Register .....	252
Reservieren eines Kanales .....	253
Freigeben eines Kanales .....	254
8/16/32-Bit Block schreiben .....	255
<b>SIMOVERT Master Drive-Funktionen</b>	<b>257</b>
Einleitung .....	258
Aufbau des Befehlsblockes .....	259
Uebergabeparameterblock .....	261
Lesen des Parameterwertes ohne Index .....	262
Schreiben des Parameterwertes ohne Index .....	264
Lesen des Parameterwertes mit Index .....	265
Schreiben des Parameterwertes mit Index .....	266
Lesen / Schreiben des Parameterwertes .....	267
Lesen/Schreiben der Parameterbeschreibungen .....	268
Lesen / Schreiben Anwender gewählte Kommandos .....	269
Lesen / Schreiben von Text .....	270
<b>PSEUDO-Befehle</b>	<b>271</b>
<b>INDEX</b>	<b>273</b>
<b>ASCII-SET</b>	<b>A-1</b>
Spezial-Zeichen .....	A-2
FCV - Charakter .....	A-3







## **Einleitung**

## Allgemeines

- Geschichte:** Das INDEL-Betriebssystem ISM wurde vor ca. 10 Jahren geschaffen, um komplexe Maschinen, Anlagen und Prozesssteuerungen zu programmieren. Durch die laufende Anpassung an die Bedürfnisse der Zeit, ist es nach wie vor ein leistungsfähiges und einfach zu handhabendes Multitasks-Betriebssystem. Es stellt dem Anwender 32 Tasks mit einer Anwendungs bezogenen 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 Verküpfungen (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.





## **Beispiel**

## Vorgabe

**Hardware:** Zum Austesten des folgenden Beispiels benötigen Sie einen INDEL 19" Rack mit CPU mit installiertem Betriebssystem (Rev. 5.0 oder höher) und min. 16 Inp/Out (egal ob intern oder EXT-IO). Zum Laden der Software und zum Debuggen benötigen Sie weiter eine 2K-SIO (87066C, Software Rev. 2.6 oder höher, Drehschalter X=4, Y=8) auf Steckplatz Nr. 27 mit Verbindung zum PC/AT (normale RS232-Verbindung ohne Steuerleitungen).

<b>Eingänge:</b>	0	RESET	TASK-0
	1	START	TASK-0
	2	STOP	TASK-0
	4+5	Lauflicht-Modus 0..3	TASK-2

<b>Ausgänge:</b>	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-Programmierung 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 IPS-32 Verzeichnis unter IPS-32\BEISPIEL\ISM\.



## EQUAL

```
.TITLE  EQUAL-File für DEMO-Tasks

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

;***** Task Start-Adressen *****
TASK_0:  .BLKW  0200      ; Programm-Bereich Start
TASK_1:  .BLKW  0100      ; TASK-1, 0100 WORD Länge
TASK_2:  .BLKW  0100      ; TASK-2, 0100 WORD Länge

;***** Hardware Adressen *****
HW_VWL:  .EQU  0440000    ; Vorwahlen RAM
VW_LNG:  .EQU  0400       ; 400 WORD Vorwahlen
HW_STA:  .EQU  045A000    ; Maschinen Status

;***** Gemeinsame Pointers *****
VWL:     .EQU  1          ; Poi-1 zeigt auf Vorwahlen
STA:     .EQU  2          ; Poi-2 zeigt auf Maschinen-Status

;***** Vorwahl-RAM Belegung *****
.VWL     .LOC  0          ; {VWL} ;
VWL_INI: .BLKW  1          ; Vorwahl-Init Erkennung
DEL1:    .BLKW  1          ; DELAY 1      RUN-Blinken
DEL2:    .BLKW  1          ; DELAY 2      Blinken Delay
DEL3:    .BLKW  1          ; DELAY 3      Lauflicht Delay

;***** STATUS Belegung *****
.VWL     .LOC  0          ; {STA} ;
STAT:    .BLKW  1          ; STATUS an PC/AT
S_ALARM  = 1              ; 1 = ALARM
S_READY  = 2              ; 2 = READY
S_RUN    = 3              ; 3 = RUN

WERT_1:  .BLKW  1          ; 16-BIT Übergabe
WERT_2:  .BLKW  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/O TASK0 EQUAL           ; Ereugt TASK0.LS, .HX und .SY
;*
;*****
; 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_VWL,2*VWL{0}                ; Vorwahlen RAM
        MOVD     HW_STA,2*STA{0}               ; Maschinen-Status

;----- Init Vorwahlen-RAM -----
--
; Wenn die INIT-Erkennung 01957 defekt ist, werden
; Grundvorgaben in das Vorwahl-RAM geschrieben!
        CBR      VWL_INI{VWL},=,01957,W_RESET  ; If Vorwahl-Init then
        MOV      0,0{VWL}                      ; Lösche erste VWL Zelle
        DUMP     0{VWL},VW_LNG-1,1{VWL}        ; Lösche ganzes VWL-RAM

        MOV      50,DEL1{VWL}                  ; DELAY-1 = 500ms RUN-

Blinken
        MOV      20,DEL2{VWL}                  ; DELAY-2 = 200ms Blinken
        MOV      3,DEL3{VWL}                  ; DELAY-3 = 30ms Lauflicht
        MOV      01957,VWL_INI{VWL}           ; Setze Init-Erkennung

;***** Warte auf RESET
*****
W_RESET:SBIT   O_ALARM,OB                      ; ALARM-Lampe ein
        MOV      S_ALARM,STAT{STA}           ; 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{STA}         ; STATUS = READY an PC/AT
W_START:TBR1   I_RESET,IB,T_RESET        ; RESET-Taste betätigt ?
        TBR0   I_START,IB,W_START        ; START-Taste nicht 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{STA}           ; STATUS = RUN an PC/AT

;***** Blinke mit RUN-Lampe bis STOP
*****
BLINK:  IBIT    O_RUN,OB                  ; Blinke mit RUN-Lampe
        MOV    DEL1{VWL},TIM             ; Lade TIM mit DELAY-1

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{STA}           ; 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/O TASK1 EQUAL           ; Erzeugt TASK1.LS,.HX und .SY
*
;*****
; 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        Laufflicht
;*****
;*
*
;*              Demo-Task 2
*
;*              Laufflicht mit Out LAUF0..7
*
;*
*
;*-----
-*
;*      Assemblieren:   MSI/O TASK2 EQUAL           ; Ereugt TASK2.LS, .HX und .SY
*
;*****
; 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                                   ; Laufflicht 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   DEL3{VWL}                              ; Warten DELAY-3
        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

;===== Laufflicht Funktionen
=====
LINKS:  ROT      1,LICHT                               ; Schiebe 1 links
        RTM      0

```





# 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 (Dreh-  
 schalterwerte), 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 Tack  
 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 Intel 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-Refreshrate 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 sollen.

Hinter '=' kann ein Downloadoffset (in hex) angegeben werden, da ja bekanntlich mit dem ISM-Compiler MSI.EXE nur Compile 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 beim Assemblieren der weiteren Files weiter verwendet werden. Dieses File wird vom INDEL-DEBUGGER "ID" benötigt, um Watches zu setzen.	
/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 Adressierungsarten möglich sind, dafür aber sehr schnell ausgeführt werden.	
/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 EUQAL.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 neben 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 ver-
wenden.
```

```
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 Directory.

**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].





## 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 Betriebsmodus.

**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

## RAM-AUFTEILUNG

## GCPU-15

### WORD-ADR

00'0000.. 01'FFFF	EPROM/CRAM-0 oder Steckplatz 2+3,4	wenn Schalter auf <b>INT</b> wenn Schalter auf <b>EXT</b>
02'0000.. 03'0000.. 03'FFFF	Steckplatz 5 Steckplatz 6..27	
04'xxxx.. 3C'xxxx	Software-Waitstates max 15 x 66ns	
40'0000.. 41'FFFF	EPROM/CRAM-0 Steckplatz 2+3	wenn Schalter auf <b>EXT</b> wenn Schalter auf <b>INT</b>
42'0000.. 43'FFFF	EPROM/CRAM-1	
44'0000.. 45'FFFF	CRAM-2	
7F'FF00..	ICU Interrupt Controll Unit	

### INT/EXT

Der Schalter INT/EXT schaltet den internen Bereich 00'0000...01'FFFF auf den externen Bereich 40'0000...41'FFFF und umgekehrt um.

		INT	EXT
CRAM/EPROM	=	00'0000...01'FFFF	40'0000...41'FFFF
Steckplatz 2+3	=	40'0000...40'FFFF	00'0000...00'FFFF
Steckplatz 4	=	41'0000...41'FFFF	01'0000...01'FFFF

### DIS/EN

Der Schalter DIS/EN verhindert das schreiben in die CRAM-0 und CRAM-1, wenn er auf DIS steht (EPROM-Simulation).

## Gx\_CPU-15 und Gx\_CPU-25

### WORD-ADR

00'0000.. 01'FFFF	EPROM-Stecker 0'0000..1'FFFF	<b>(2x4-MBit, 70ns Chip's)</b> oder Steckplatz 2+3,4 wenn <b>kein EPROM-Stecker</b> vorhanden
02'0000.. 03'0000.. 03'FFFF	Steckplatz 5 Steckplatz 6..27	
04'xxxx.. 3C'xxxx	Software-Waitstates max 15 x 66ns bei Gx_CPU-15, 15 x 80ns bei Gx_CPU25	
40'0000.. 41'FFFF	Steckplatz 2+3	
42'0000.. 43'FFFF	EPROM-Stecker 2'0000..3'FFFF	<b>(kein Jumper)</b> oder CRAM-1 wenn <b>Jumper "Mode auf GND" bestückt</b>
44'0000.. 45'FFFF	CRAM-2	(2x1-MBit, 70ns Chip's)
48'0000.. 4A'0000.. 4C'0000.. 4D'FFFF	CRAM-0 CRAM-1 CRAM-2	(2x1-MBit, 70ns Chip's) zweitbelegung (2x1-MBit, 70ns Chip's) zweitbelegung (2x1-MBit, 70ns Chip's)
60'0000.. 67'FFFF	EPROM 0'0000..7'FFFF,	das ganze <b>1-MByte</b> zusammenhängend ! <b>(2x4-MBit, 70ns Chip's)</b>
68'0000.. 6F'FFFF	CRAM-0 <b>(2x4-MBit, 70ns Chip's)</b>	zweitbelegung
7F'FF00..	ICU Interrupt Controll Unit	

### DEBUG-Stecker

00'0000.. 01'FFFF	CRAM NotSystem NotSystem	wenn Schalter auf <b>CR</b> wenn Schalter auf <b>EP</b> und <b>kein EPROM-Stecker</b> wenn Schalter auf <b>EP</b> und Schalter auf <b>PRG</b>
42'0000.. 43'FFFF	ebenso,	wenn Jumper "Mode auf GND" <b>nicht</b> bestückt
50'0000.. 57'FFFF	immer EPROM-Simulations CRAM (egal auf was das System gerade läuft)	
58'0000.. 5F'FFFF	EPROM brennen	(Schalter auf <b>PRG</b> )
70'0000.. 77'FFFF	immer NotSystem EPROM (egal auf was das System gerade läuft)	

## Fx\_CPU-25

### WORD-ADR

00'0000.. 07'FFFF	EPROM-Stecker <b>(2x4-MBit, 70ns Chip's)</b>			
30'0000.. 31'0000.. 32'0000.. 33'0000.. 33'FFFF	Steckplatz 2+3 Steckplatz 4 Steckplatz 5 Steckplatz 6..27			
34'xxxx.. 3C'xxxx	Software-Waitstates max 3 x 320ns			
40'0000.. 41'FFFF	CRAM-0 * Soft Write-Protect (2x1-MBit, 70ns Chip's)	WR H-Byte WR L-Byte	@71'xxxx @7x'xxxx	<b>AUF ZU</b>
42'0000.. 43'FFFF	CRAM-1 (2x1-MBit, 70ns Chip's)			
44'0000.. 45'FFFF	CRAM-2 (2x1-MBit, 70ns Chip's)			
58'0000.. 5F'FFFF	CRAM-0 *            zweitbelegung <b>(2x4-MBit, 70ns Chip's)</b>	WR H-Byte WR L-Byte	@71'xxxx @7x'xxxx	<b>AUF ZU</b>
60'0000.. 67'FFFF	CRAM1            zweitbelegung <b>(2x4-MBit, 70ns Chip's)</b>			
68'0000.. 6F'FFFF	CRAM-2            zweitbelegung <b>(2x4-MBit, 70ns Chip's)</b>			
7F'FF00..	ICU Interrupt Controll Unit			

### DEBUG-Stecker

00'0000.. 07'FFFF	CRAM            wenn Schalter auf <b>CR</b> NotSystem       wenn Schalter auf <b>EP</b> und <b>kein EPROM-Stecker</b> NotSystem       wenn Schalter auf <b>EP</b> und Schalter auf <b>PRG</b>
08'0000.. 0F'FFFF	EPROM brennen (Schalter auf <b>PRG</b> )
10'0000.. 17'FFFF	immer EPROM-Simulations CRAM (egal auf was das System gerade läuft)
70'0000.. 77'FFFF	immer NotSystem EPROM (egal auf was das System gerade läuft)

---

00'0000            Steckplatz 2..27 wenn **kein DEBUG-Print** und **kein EPROM-Stecker**  
03'FFFF            vorhanden sind!

---

# REGISTER

## Task Register

Label	REG	15	8	7	0						
RNR	R7F	Rack Number									
MPC	R7E	Macro Programm Counter									
HTW	R7D	B	T	S	D	U	B	D	-	H	BD - HALT
TIM	R7C	10 ms TIMer									
ABA	R7B	ABort Adresse									
ABC, APO	R7A	ABort-Chara			Ascii-Pos						
ASL, ASR	R79	ASc Länge			ASc-Reg nr.						
SEC	R78	SECond timer									
SPO	R77	Kopie von SPO			Stack-Pointer						
STK	R76	STACK									
	R70										
	R6F										
	R60	(ASCII-Buffer)									
	R5F	TASK-Register									
	R00										

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     | Haltsperre-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 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. Mann kann damit die Grösse des Eingabe-Fensters in einer Bildschirm-Maske begrenzen oder den ASCII-Buffer länger und

## **ADRESSIERUNGSARTEN**

## Befehlsaufbau

15		8	7		0	
Befehls Code		SSSS		DDDD		Befehls Kopf
Steuer-Bits		SSSS		DDDD		Bei mehr als 2 Argumenten
SAD Sprung Adresse						Bei bedingten Sprüngen
DATA SRC						
DATA DEST						
DATA SRC			DATA DEST			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	WWW WWW	WWW WWW	WORD :W
1	1	LLLL LLLL	LLLL LLLL	D_WORD :D:F
		HHH HHH	HHH HHH	INT / FLOAT
2	2		QQQ QQQ	BYTE :B
3	3		ORR RRR	REG R0..R7F
3	3		1-- —	NOT USED
4	4		ORR RRR	(REG)
4	4		1RR RRR	[REG]
5	5	OOO OOO : 0RR RRR		OFF (REG)
5	5	OOO OOO : 1RR RRR		OFF [REG]
6	6	OOO OOO : 0BB BBB		OREG (BREG)
6	6	OOO OOO : 1BB BBB		OREG [BREG]
6	6	1NN NNN : 0RR RRR		(REG)N
6	6	1NN NNN : 1RR RRR		[REG]N
7	7	OOO OOO : 0RR RRR		REG@ADRE
7	7	OOO OOO : 1RR RRR		REG@ADRE
8	8	AAA AAA AAA AAA		@ADRE
9	9	OOO OOO OOO:PPP		OFF{POI}
9	9	1OO OOO OOO:PPP		OFF@{POI}
A	A		keine	ASC
B	B		MANTISSE	DOUBLE
			MANTISSE	-PRECISION
			MANTISSE	-FLOATING
		S:	EXPONENT :MANT	-POINT
C		QQQ QQQ		BYTE
D		ORR RRR		REG
D		1		NOT USED
E		ORR RRR		(REG)
E		1RR RRR		[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	00000000...00000007F
	-128 ... -1	0FFFFFFF80...0FFFFFFF
WORD:	128 ... 65535	00000080...00000FFF
	-32768 ... -129	0FFFF8000...0FFFFFFF7F
DOUBLE-WORD:	65536 ... 4294967294	000010000...0FFFFFFF
	-2147483648 ... -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	0A00,R10	; R10	=	A000 richtig!
	MOVD	0A000,R10	; R11,10	=	FFFF'A000 falsch??
	MZWD	0A000,R10	; R11,10	=	0000'A000 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	1E4,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 werden müssen. Die Angaben :F und :L haben keinen Einfluss und können weggelassen werden!

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 werden.

**Beachte:** Der Offset ist immer positiv und muss im Bereich 000.07FF sein.

**Pointer Laden:** Damit die Pointer selbst geladen werden 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-8332 Russikon
...			
TTAB:	.WORD	TXT0,TXT1,TXT2,TXT3	; TEXT-Tabelle
TXT0:	.TXT	'INDEL AG'	
TXT1:	.TXT	'Industrielle Elektronik'	
TXT2:	.TXT	'Tüfiwis 26'	
TXT3:	.TXT	'CH-8332 Russikon'	

## 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-BitAdresse im gleichen (64k)RACK-Bereich wie 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\*TR 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                            ; WARTEN 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 Variablen-tabelle 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 Variablen-tabelle 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 der 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 Variablen-tabelle, interpretiere das Doppelwort an 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 Variablen-tabelle 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 Deskriptor

B7\_ 01\_      GGD      SRC, DEST:D

Erklärung:      Suche das Label mit dem Namen in SRC in der globalen Variablen-tabelle 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 werden.

Damit ein Label in die globale Variablen-tabelle 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

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



## **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 werden 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 Doubleword-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):



## **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  
werden 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-Module:

JEX      0100            ; PC = 0100(REX-MODULE)

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
```

## 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

## 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 werden

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!

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

GGD            @TX\_EXQ, R10  
 RCXP        045A000, 0, R10

## **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

## 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 1:        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

## 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 Bit verarbeitet. Die CPU holt erst die zu verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins interne Register. Das heisst, der Bit-Range sich kann 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 gesetzte 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 angeben werden kann, werden je nach Start-  
 Bit nur bis zu 25 Bit verarbeitet. Die CPU holt erst die zu  
 verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins  
 interne Register. Das heisst, der Bit-Range sich kann 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 angegeben werden kann, werden je nach Start-Bit nur bis zu 25 Bit verarbeitet. Die CPU holt erst die zu verarbeitenden Bits mit einem Double-Word Transfer vom Memory ins interne Register. Das heisst, der Bit-Range sich kann 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



## 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: Der MOVF und MOVL geht auf TRAP-3 wenn die Zahl keine Floating-Point Zahl ist !

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

```
MOV    @ADRE,R00
```

## XCH\_

### eXCHange

2 1xx	XCH	SRC,DEST
3 1xx	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-Buffer 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-Buffer auf 10 Zeichen:

MLHB 10,ASL

Beispiel 2: Ueberschreibe den zweiten Buchstaben im ASCII-Buffer 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:

```
LSH        5,R00                    ; R00 =        0001
                                     ; 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:

```

ASH       -2,R00                   ; R00 =       0FF00 (-256)
                                  ; 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:

```

ROT        5,R22                    ; R22 =        1000
                                     ; 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\_**

## SUBtraction

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	DIVSRC,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  
+ 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.

 $+ 10 \text{ MOD } +3 = +1$  $- 10 \text{ MOD } +3 = +2$  $+ 10 \text{ MOD } -3 = -2$  $- 10 \text{ MOD } -3 = -1$ 

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-BUFFER 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 = 0DEST unverändert
D	&N	BR IF AND >< 0DEST unverändert
E	+Z	BR IF ADD = 0DEST=DEST+SRC !!
F	+N	BR IF ADD >< 0DEST=DEST+SRC !!

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

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

Beispiel 2: Durchlaufe einen LOOP 125 mal:

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

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

```
LOOP:     ADDR     ASC,R0          ; Adresse vom ASCII-BUFFER
          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 und 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:

## **TEXT IN/OUT-Befehle**

## VIDEO FCV

Device Nummer: 00..03

Farb Attribut: Das Farb-Attribut wird mit der Device-Nummer im High-Byte angegeben. Jeder TIP und TOP kann eine neue Farbe bekommen. Wird keine Farbe angegeben (00), so wird auch nichts angezeigt (schwarz auf schwarz)!

	15	12 11	8 7	0
DEV:	I B G R	I B G R	DEVICE	
	Hintergrund	Vordergrund	Device	

Position: Das Positions-WORD sieht so aus: YYXX  
 Dabei ist: XX = X - Position 00..04F  
 YY = Y - Position 00..018

0000	004F
Bildschirm	
1800	184F

Wird XX>04F so wird automatisch auf eine neue Zeile geschrieben und XX=00 gesetzt.

Wird YY>018 so wird automatisch eine Zeile hoch gerollt und YY=18 gesetzt.

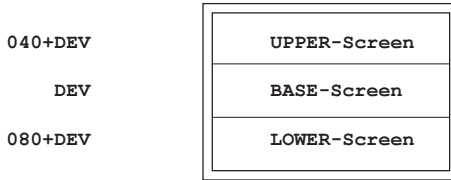
Extends: Folgende Extends können hinter die TIP und TOP Befehle geschrieben werden: (keine Angabe == OP)

Ext	Funktion	Positions-Update	Code
OP	Old Position	YY=OLD XX=OLD	00
NP	New Position	YY=NEW XX=NEW	01
CRLF	Return Linefeed	YY=NEW+1 XX=00	02
PCR	Linefeed	YY=NEW+1 XX=OLD	03
SP	Space	YY=NEW XX=NEW+1	04

## SPLIT-SCREEN

Device Nummer: 040..043 UPPER-Screen  
 000..003 BASE -Screen  
 080..083 LOWER-Screen

Split-Screen: Der Bildschirm von Device 0..3 kann in drei Bereiche aufgeteilt werden, die jedoch alle dem selben Task zugeordnet werden:



Aufteilung: Die Aufteilung des Schirmes erfolgt mit einem SETD oder INID- Befehl, indem dabei die Device Nummer + 040 angegeben wird. Dann lesen diese beiden Befehle ein zusätzlichs Wort, in welchem die Länge vom UPPER- und LOWER-Screen wie folgt angegeben werden:

	31	24 23	16 15	8 7	0
SET Split Screen:	LOWER-Length	UPPER-Length	COLOR	040 + DEV	
	Bildschirm Aufteilung		Farbe und Device		

Die Länge des BASE-Screens ergibt sich, da der Bildschirm insgesamt 25 Zeilen hat.

Beachte: Der UPPER- und LOWER-Screen können von 0..24 Zeilen lang sein. Der BASE-Screen kann von 1..25 Zeilen lang sein! (Hardware)  
 Nicht sichtbare Screens haben immer maximale Länge und können im Hintergrund beschrieben werden!

Ein/Ausgabe: Die Screens werden für TIP und TOP mit den Bits U für UPPER-, L für LOWER- und keines für den BASE-Screen in DEV angesprochen:

	15	12 11	8 7 6	0
DEV:	I B G R	I B G R L U	DEVICE	
	Hintergrund	Vordergrund	Device	

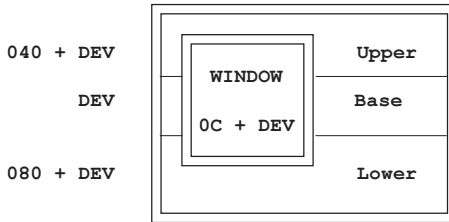
Position: Jeder Screen beginnt oben links mit 0000!  
 Wird YY>SCREEN-LENGHT so wird automatisch eine Zeile hoch gerollt und YY=SCREEN-LENGHT gesetzt. Die drei Bereiche rollen also unabhängig voneinander!

## WINDOW

Device Nummer: 0C..0F WINDOW

Soft-Window: Für Hilfe-Texte und ähnliches genügt meist ein Software-Window, wie es mit den Befehlen PUTD, GETD und HTOP Beispiel-5 beschrieben wird!

Hard-Window: Für Zusatzprogramme wie DEBUG, ERROR-Meldungen usw verfügt die FCV-Karte über ein Hardware-Window, das als eigenständiges Device (0C..0F) von einem beliebigen Task über dem Grund-Device (0..3) geöffnet werden kann.



Aufteilung: Die Definition des Windows erfolgt mit einem SETD oder INID- Befehl, indem die Window-Device Nummer (0C..0F) + 040 angegeben wird. Dann lesen diese beiden Befehle zusätzlich zwei Worte, in welchen die Position und Grösse des Windows wie folgt angegeben werden können:

	47	40	39	32	31	24	23	16	15	8	7	0
SET Window:	Y-POS		Y-Lenght		X-POS		X-WIDTH		COLOR		04C + DEV	

(Ein SETD/INID auf 0C..0F belegt einfach den ganzen Bildschirm)

Beachte: Das notwendige Dreifach-WORD kann nicht immediate angegeben werden und muss daher in Registern oder über Adresse angegebene werden.

Ein/Ausgabe: Die Ausgabe aufs Window erfolgt ganz normal auf die Device-Nummer 0C..0F (+040 ist nur für SETD,INID). Der TIP im Window bekommt einen eigenen Cursor und hat Vorrang, ebenso die Behandlung des ESC-Charakters. Eingaben auf dem Grund-Device sind erst wieder nach Schliessung des Windows möglich!

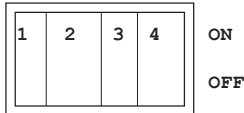
Position: Wird `XX>WINDOW-WIDTH` so wird automatisch auf eine neue Zeile geschrieben und `XX=00` gesetzt. Wird `YY>WINDOW-LENGHT` so wird automatisch eine Zeile hoch gerollt und `YY=WINDOW-LENGHT` gesetzt. Das Window rollt also unabhängig vom Grund-Device!

## S-I/O 32

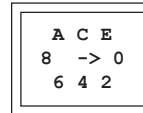
Device: 04...07

Baud Rate: Die Baud Rate wird bei der SIO-32 mit den Schaltern auf der Karte eingestellt:

Mode - Schalter



Baud-Rate



SWITCH	MODE	OFF	ON
1	PARITY	EVEN	ODD
2	PARITY	EN	DIS
3	STOP BITS	2	1
4	DATA BITS	8	7

Baud	Baud-Rate	Baud	Baud-Rate
0	50	8	2000
1	75	9	2400
2	110	A	3600
3	150	B	4800
4	300	C	9600
5	600	D	19200
6	1200	E	38400
7	1800	F	56000

Position: Das Positions-WORD sieht so aus: YYXX  
 Dabei ist: XX = X - Position  
 YY = Anzahl Linefeeds

XX wird vom Zeilen-Anfang gezählt.  
 Ist XX > akt Position, werden vor der Textausgabe bis XX Spaces ausgegeben.  
 Ist XX ≤ akt Position, so hat die Angabe keine Funktion.

YY bestimmt die Anzahl Leerzeilen, die vor der Text-Ausgabe gemacht werden.

Normalerweise wird mit POS=0000 gearbeitet, was einer kontinuierlichen Ausgabe entspricht.

Extends: Folgende Extends können hinter die TIP und TOP Befehle geschrieben

werden:

Ext	Funktion	
<del>CRLF</del>	<del>Return Linefeed</del>	<del>nach Text-Ausgabe</del>
SP	Spacenach	Text-Ausgabe
E	Echo (nur TIP)	für Terminal-Eingabe

**Sonderzeichen:** Eine Text-Eingabe wird immer mit CR (0D) abgeschlossen. Wenn ECHO aktiv ist, wird mit ECHO CR/LF (0D,0A) als Echo ausgegeben. Das letzte Zeichen wird bei der Eingabe immer ignoriert! Der BACK-SPACE-Character (08) wird bei der Eingabe immer ignoriert! Wird der Abort-Character empfangen, wird die Task auf seine ABORT-Adresse. Alle anderen Zeichen werden normal verarbeitet.

**DTR,RTS:** Beide Ausgänge werden identisch bedient und dienen zur Steuerung des Eingabegerätes. Sie werden nur aktiv (+15V) wenn die Task gestartet wurde.

**CTS:** Über den CTS-Eingang kann die Ausgabe gestoppt werden. Wenn er inaktiv (-15V) ist, wird die Ausgabe gestoppt (das nächste Zeichen wird noch gesendet). Wird dies nicht gebraucht, CTS -> +5..15V.

**DSR:** Ist bei INID,SETD der DSR-Eingang inaktiv (-15V), sendet die Task auf seine ABORT-Adresse. Dies dient der Erkennung, ob eine SIO-32 vorhanden ist, das Ausgabe-Gerät angeschlossen ist und zum Empfang der Daten bereit ist (Papier-Ende). Dient zur Steuerung der DSR wie die CTS-Leitung. Wird dies nicht gebraucht, DSR -> +5..15V.



## CENTRONICS

**Device:** 4...7 Hardware-kompatibel mit der SIO-32. Die Centronics-Karte kann anstelle einer SIO-32 eingesteckt werden (nur Textausgabe).

**Speed:** Auf der Karte ist ein Jumper, der die Uebertragungs-Geschwindigkeit auf 1000-Zeichen pro Sekunde beschränkt. Damit bleibt der CPU genügend Rechenzeit für die Steuerung der Maschine.

**Position:** Das Positions-WORD sieht so aus: YYXX  
 Dabei ist: XX = X - Position  
 YY = Anzahl Linefeeds

XX wird vom Zeilen-Anfang gezählt.  
 Ist  $XX > \text{akt Position}$ , werden vor der Textausgabe bis XX Spaces ausgegeben.  
 Ist  $XX \leq \text{akt Position}$ , so hat die Angabe keine Funktion.

YY bestimmt die Anzahl Leerzeilen, die vor der Textausgabe gemacht werden.

Normalerweise wird mit POS=0000 gearbeitet, was einer kontinuierlichen Ausgabe entspricht.

**Extends:** Folgende Extends können hinter die TIP und TOP Befehle geschrieben werden:

Ext	Funktion	Code
CRLF	Return Linefeed	nach Text-Ausgabe 02
SP	Space	nach Text-Ausgabe 04

**PAPER END:** Die PE-Leitung entspricht dem DSR bei der SIO-32. Ist bei INID,SETD die PE-Leitung inaktiv, so springt der Task auf seine ABORT-Adresse. Dies dient der Erkennung ob eine CENTRONICS-Karte vorhanden ist, das Ausgabe-Gerät angeschlossen und zum Empfang der Daten bereit ist (Papier-Ende). Danach funktioniert sie wie der CTS bei der SIO-32.

## 2 Kanal S-I/O

Device: 08..0B

Baud Rate: Die Baud Rate wird bei der 2k-SIO mit der Device Nummer im High-Byte angegeben. Ist das Format Byte=00 so gilt das letzte gewählte Format weiter (nach INIT 9600,N,8,1). Das Uebertragungsformat wird nur bei SETD und INID verarbeitet und hat bei den restlichen Befehlen keine Funktion mehr!

15	12	11	10	8	7	0
odd PEn 2SB 8DB				XON BAUD-RATE		07 ... 0B
Uebertragungs Format					Device	

BIT	MODE	0	1
15	PARITY	EVEN	ODD
14	PARITY	DIS	EN
13	STOP BITS	1	2
12	DATA BITS	7	8
11	XON-XOFF	Nein	JA

B8-10	Baud-Rate	B8-10	Baud-Rate
0	300	4	4800
1	600	5	9600
2	1200	6	19200
3	2400	7	38400

Position: Das Positions-WORD sieht so aus: YYXX  
 Dabei ist: XX = X - Position  
 YY = Anzahl Linefeeds

XX wird vom Zeilen-Anfang gezählt.  
 Ist XX>akt Position, werden vor der Textausgabe bis XX Spaces ausgegeben.  
 Ist XX%akt Position, so hat die Angabe keine Funktion.

YY bestimmt die Anzahl Leerzeilen, die vor der Textausgabe gemacht werden.

Normalerweise wird mit POS=0000 gearbeitet, was einer kontinuierlichen Ausgabe entspricht.

Extends: Folgende Extends können hinter die TIP und TOP Befehle geschrieben

werden:

Ext	Funktion		Code
CRLF	Return Linefeed	nach Text-Ausgabe	02
SP	Space	nach Text-Ausgabe	04
E	Echo	für Terminal-Eingabe	20

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

020	[ABC] empfangen
021	Framing Error
022	Parity Error
023	Overrun Error
024	INP-Puffer OVERFLOW
025	DSR bei SETD,INID = 0

Sonderzeichen: Eine Text-Eingabe wird immer mit CR (0D) abgeschlossen. Beim TIP mit ECHO wird CR/LF (0D,0A) als Echo ausgegeben. Der LF (0A) und das 00 wird bei der Eingabe ignoriert! Wird der Abort-Character 'ABC' empfangen, so wird der Eingabe- Puffer gelöscht und der Task springt auf seine ABORT-Adresse.  
XON (011) und XOFF (013) werden bei eingeschaltetem XON/XOFF-Betrieb abgefangen.  
Alle andern Zeichen werden normal verarbeitet.

Puffers: Die 2k-SIO hat pro Kanal:  
11k-Byte Ausgangs-Puffer  
4k-Byte Eingangs-Puffer

Die Uebergabe vom NS32016 an den HD64180 geht über ein 1K-FIFO- RAM. Daher ist die  
AUSGABE auf 768-Zeichen pro TOP-String und die  
EINGABE auf 256-Zeichen pro TIP-String begrenzt!

OUT Der Ausgangs-Puffer nimmt immer Daten von TOP oder TIP- ECHO auf, solange er nicht bis auf 1k-Byte voll ist, auch wenn der Ausgabe-Kanal mit CTS oder XOFF gebremst wird. Er kann nur mit CLRD gelöscht werden !

- Puffers: INP** Der Eingangs-Puffer ist immer Empfangsbereit, solange er nicht voll ist und auch wenn kein TIP gestartet wurde. Ein TIP nimmt immer nur Daten aus dem Eingangs-Puffer. Dieser Puffer wird mit INID, SETD, CLRD gelöscht. Er wird ebenfalls bei FRAMING, PARITY und OVERRUN-ERROR oder 'ABC' 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:** Im Gegensatz zur SIO-32 wird hier der RTS-Ausgang richtig als Request To Send angesteuert! Er 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 INID,SETD der DSR-Eingang inaktiv (-15V), so springt der Task auf seine ABORT-Adresse. Dies dient der Erkennung ob überhaupt eine 2k-SIO vorhanden ist, das 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 ist neu und dient bei Modem-Betrieb als Data Carrier Detect. Wenn der DCD inaktiv (-15V) ist, wird der Eingangs-Kanal abgeschaltet und es können keine falsche oder undefinierte Zeichen empfangen werden. Ist er aktiv (+15V), so wird der Eingang ganz normal durch geschaltet.  
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 SETD,INID (nur wenn XON/XOFF gewählt!). Wird XON empfangen und es stehen noch Daten im OUT-Puffer, wird das Senden fortgesetzt.
- Beachte: Die Steuerleitungen CTS,DSR,DCD werden auch bei 20mA, RS422 und XON/XOFF-Betrieb verarbeitet.  
Werden sie nicht benötigt, alle auf +5..15V !

## SETD

SET Device

81x0            SETD     DEV

Erklärung:        Mit dem SETD wird das DEV für diesen Task reserviert.  
 Hat bereits ein anderer Task das DEV für sich reserviert, wird  
 solange gewartet, bis das DEV wieder frei ist.

(ABC) = 01B ; (APO) = 000 ; (ASL) = 01F ; (ASR) = 060

SPLIT-SCREEN:    DEV+040,    zusätzlich 1 WORD    definiere Bildschirm Aufteilung  
 WINDOW:         DEV+04C,    zusätzlich 2 WORDs    definiere Window Position+Grösse  
 2K-SIO:           Setze das Uebertragungs-Format, lösche INP-Puffer !

Beispiel:         Reserviere Video 0 (alle 25-Zeilen = Base-Screen):

SETD     0

Beispiel 1:        Warte bis das Video 3 frei ist und reserviere es wie folgt:

Upper-Screen:     1    Zeile  
 Base-Screen:      22    Zeilen  
 Lower-Screen:     2    Zeilen

SETD     02010043

Beispiel 2:        Setze die 2K-SIO Kan-0 = 9600,n,8,1:

SETD     01508

Beispiel 3:        Reserviere das Window-Device von Bildschirm 02: (Full-Screen)

SETD     0E

Beispiel 4:        Oeffne ein Window auf DEV-00, 5-Zeilen Abstand oben, 8-Zeichen  
 vom linken Rand, 10-Zeilen hoch und 40-Zeichen breit:

WIND:            .WORD    004C,0828,050A    ; Window-Definition  
 SETD            @WIND

## INID

INIt Device

80x0            INID        DEV

Erklärung:        Mit dem INID wird das DEV sofort für diesen Task reserviert. Hat bereits ein anderer Task das DEV für sich reserviert, wird dieser Task auf seine ABORT-Adresse gesetzt und das DEV normal reserviert.(Sollte nur sehr bewusst eingesetzt werden!)

(ABC) = 01B ; (APO) = 000 ; (ASL) = 01F ; (ASR) = 060

SPLIT-SCREEN:    DEV+040,        zusätzlich 1 WORD    definiere Bildschirm Aufteilung  
 WINDOW:            DEV+04C,        zusätzlich 2 WORDs    definiere Window  
                           Position+Grösse

2k-SIO:            Setze das Uebertragungs-Format, lösche INP-Puffer !

Beispiel:            Reserviere Video 3 mit Gewalt (alle 25-Zeilen = Base-Screen):

INID            3

## RESD

RESet Device

83x0            RESD     DEV

Erklärung:        Gibt ein mit SETD,INID reserviertes DEV wieder frei für andere Tasks  
(noch laufende JTIPs werden gelöscht).

Beispiel:         Gib Video 3 frei:

RESD     3



## CLRD

CLear Device

82x0            CLRD     DEV

Erklärung:        Warte, wenn noch ein TIP auf dem DEV läuft (—> CTIP ).

VIDEO:            Lösche den Bildschirm.

GRAPHIK:         Werden L und U in DEV gesetzt, wird der Graphik-Schirm gelöscht!

SIO-32:            Sende CANCEL

2K-SIO:            Lösche den Eingangs- und Ausgangs-Puffer.

Beachte:          Mit den L und U-Bits kann gewählt werden, welcher Charakter-Schirmbereich gelöscht werden soll.  
Werden beide Bits gleichzeitig gesetzt (xxCx), so wird der Graphik-Bildschirm gelöscht. Dies muss immer am schluss eines Graphik-Programmes gemacht werden, da der normale CLRD am Anfang des nächsten Programmes die Graphik nicht löscht!

Beispiel 1:        Lösche den (BASE-)Screen von Video 3 mit Blau:

CLRD     04003

Beispiel 2:        Lösche den LOWER-SCHREEN von Video-0 mit grün:

CLRD     02080

Beispiel 3:        Lösche das WINDOW-Device:

CLRD     0C

Beispiel 4:        Lösche den Graphik-Schirm (Grund-Device-0)

CLRD     00C0

## CTIP

Clear TIP

84x0            CTIP        DEV

Erklärung:        Wenn noch ein JTIP auf dem DEV läuft, wird er abgebrochen.

Beispiel:            Lösche den Text vom Device:

CTIP        DEV

## CRLF

Carriage Return / Line Feed

85xx            CRLF        DEV,POS

VIDEO:            XX = 0 ; YY + 1 ; wenn YY>SCREEN-Lenght wird hoch gerollt.  
SIO:                Send CR/LF (0D,0A)

Beispiel:            New-Line auf DEV, update POS:

CRLF        DEV,POS

## PUTD

PUT Device to screen-save n

9Bxx            PUTD        DEV,N

Erklärung:        Rette den aktuellen Charakter-Screen in einen der 4 Bild- Speicher. Damit können Soft-Windows eröffnet werden (bestehender Text wird erst gespeichert und dann überschrieben), die mit dem GETD-Befehl wieder geschlossen werden (der überschriebene Text wird wieder zurückgeholt).  
N = 1..4

Beachte:          Jeder Screen (UPPER LOWER und BASE) und das Window-Device haben jeweils 4 eigene Bild-Speicher!

Beim Rollen eines Screens werden dessen Bild-Speicher überschrieben!

Es wird jeweils nur die aktuelle Screen-Länge abgelegt und auch der Bild-Speicher entsprechend verwaltet. Bei kürzeren Screens können also auch mehr Bilder abgelegt werden. Der Speicher pro Screen inkl. sichtbarem Bereich = 5 Bildschirme à 25 Zeilen!

Beispiel:          Rette den Base-Screen in den Speicher 3:

PUTD            0,3

## GETD

GET Device from screen-save n

9Cxx            GETD        DEV,N

**Erklärung:**        Hole den geretteten Charakter-Screen aus einem der 4 Bild-Speicher. Damit werden Soft-Windows geschlossen (der überschriebene Text wird zurückgeholt), die mit dem PUTD-Befehl eröffnet wurden. Auch ist es möglich, mehrere Bildschirm-Masken sehr schnell abzurufen.

**Beachte:**           Jeder Screen (UPPER LOWER und BASE) und das Window-Device haben jeweils 4 eigene Bild-Speicher!

Beim Rollen eines Screens werden dessen Bild-Speicher überschrieben!

Es wird jeweils nur die aktuelle Screen-Länge abgelegt und auch der Bild-Speicher entsprechend verwaltet. Bei kürzeren Screens können also auch mehr Bilder abgelegt werden. Der Speicher pro Screen inkl. sichtbarem Bereich = 5 Bildschirme à 25 Zeilen!

**Beispiel:**           Erstelle wieder den Original-Text auf dem LOWER-Screen aus dessen Bild-Speicher 3

GETD            080,3

## TOP

### Text OutPut

90xx \_\_x0      TOP      DEV,POS,TADR,EXT

Erklärung:      Ausgabe vom Text-String TADR auf DEV an der Position POS. Update POS entsprechend EXT.

Beispiel 1:      Schreibe den Textstring aufs VIDEO 3 in die Zeile 5, Spalte 10:

TOP      3,050A,@TAD1

Beispiel 2:      Schreibe den Textstring mit der Adresse in REG 22 auf das Device in DEV an die Position in POS! Update POS mit neuer Position:

TOP      DEV,POS,0(R22),NP

Beispiel 3:      TOP den ASCII-Puffer. Update POS mit CRLF:

TOP      DEV,POS,ASC,CRLF

Beispiel 3:      TOP 'A=5'. Behalte alte Position:

TOP      DEV,POS,"A=5"

## GTOP

Gross-Text OutPut  
! Nur VIDEO !

92xx 00x0      GTOP      DEV,POS,TADR

Erklärung:      Grosstext-Ausgabe vom Text-String TADR auf DEV an der Position POS.

Beachte:        GTOP ist nur auf VIDEO möglich!  
POS wird nie nachgeführt (keine Extends)!  
GTOP rollt das Bild nicht automatisch hoch !

Beispiel:        Schreib diesen Textstring aufs VIDEO 3 in die Zeile 5, Spalte 8.

TAD2:            .TXT            'GROSS!'  
...

GTOP            3,0508,@TAD2

1234567 . 1234567 . 1234567 . 1234567 . 1234567 . 1234567 .

```

XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX    XX    1
XX        XX  XX XX  XX XX        XX        XX        2
XX  XXX  XXXXXXXX XX    XX  XXXXXXXX XXXXXXXX    XX    3
XX  XX XX  XX XX  XX        XX        XX        4
XXXXXXXX XX    XX  XXXXXXXX XXXXXXXX XXXXXXXX    XX    5

```

YÇØ□

Grösse der Zeichen :    7 x 5 Zeichen mit 1 Space Abstand.  
Maximal 4 Zeilen mit je 10 Zeichen.

## BTOP

Balken TOP

9Axx \_\_xx BTOP DEV,POS,MAX,N,EXT

**Erklärung:** Zeichne einen MAX-Zeilen hohen BALKEN mit N ausgefüllten Linien. Der Balken ist immer 2 Zeichen breit und die POS = POS+2. Nicht gefüllte Linien werden mit Hintergrund-Farbe gelöscht. Update POS entsprechend EXT.

**Beachte:** Der BALKEN-TOP ist nur auf dem VIDEO möglich!  
1 ZEILE = 10 Linien  
Max Balkenhöhe = 25 Zeilen = 250 Linien (Full-Screen)

**Beispiel:** Zeichne einen 10-Zeilen hohen Balken (100%) der R00% ausgefüllt ist: (POS-Update für nächsten Balken !)

BTOP DEV,POS,10,R0,SP



## BTOP

Block Text OutPut

9Axx \_\_xx      BTOP      DEV,POS,TBLK,N,EXT

Erklärung:      Ausgabe von N-Zeichen aus dem Text-Block TBLK an die 2K-SIO.  
Mit diesem Befehl können alle Zeichen von 00..FF ohne  
Einschränkungen ausgegeben werden.

Beachte:      Der BLOCK-TOP ist nur auf der 2K-SIO möglich!  
N max = 768 Zeichen.

Beispiel 1:      Sende diese 5-Byte Steuersequenz an den Drucker :

TBLK:      .BYTE      01B,'T',000,035,'q'  
            ...  
BTOP      8,0,@TBLK,5

## MTOPT

Multi Text OutPut

91xx \_\_xx      MTOPT      DEV,POS,MTAB,EXT

**Erklärung:** Ausgabe aller Textstrings, deren Adressen in der MTOPT-Tabelle MTAB stehen. Update POS entsprechend EXT.

**Beachte:** 'MTAB' kann auch in einem anderen (64k) RACK-Bereich sein. Die Textstrings dazu müssen dann aber im gleichen Rack-Bereich wie MTOPT-Tabelle sei.

**Steuerzeichen:** Adressen < 0100 in der MTAB sind Steuerzeichen !

```

020                    Skip 1 Zeichen
021..03F    Skip N Zeichen        N = 1..01F
00D         New Line                (alte Spalte!)
0D1..0DF    New Line N            N = 1..0F
000                    Tabellen Ende
  
```

In den Textstrings dürfen auch <0A><0D><09> vorkommen.

**Beispiel:**      MTOPT      02,0108,@MTAB

Dieser Befehl erzeugt folgendes Bild:  
(Beachte dass alle Texte in der Spalte 8 starten)

	0	1	2
	0123456789ABCDEF0123456789ABCDEF01234567		
00			
01	Text eins                zwei		
02	Text drei    Text eins		
03			
04			
05	Dies ist ein Textstring		
06			

```

TAD1:    .TXT      'Dies ist ein Textstring'
MTX1:    .TXT      'Text eins'
MTX2:    .TXT      '<09>zwei'
MTX3:    .TXT      'Text drei'
MTAB:    .WORD     MTX1,MTX2,0D,MTX3,022,MTX1,0D3,TAD1,00
  
```

## HTOP


Horizontal Text OutPut

93xx \_\_xx      HTOP      DEV,POS,SRC,N,EXT

**Erklärung:**      Gebe das Zeichen SRC in der Position POS N-mal aus Update POS entsprechend EXT. Bei NP steht die POS auf dem letzten ausgegebenen Charakter (damit nicht gerollt wird am Zeilen/ Bildende).

N-max = 07FF      ( ganzer Bildschirm )

**Rahmen:**      Nur VIDEO !  
Ist das SRC-Zeichen eine Rahmen-Ecke, so wird die Linie folgerichtig weiter gezogen und mit der Gegen-Ecke beendet !

Zum Beispiel | erzeugt 

**Beispiel 1:**      Lösche 15 Zeichen von POS aus. Behalte alte Positon in POS:

HTOP      DEV,POS,020,15,OP

**Beispiel 2:**      Zeichne eine Linie '\*\*\*\*\*' über die ganze Zeile 5: (SIO+VIDEO)

HTOP      DEV,0500,"\*",80

**Beispiel 3:**      Zeichne einen Zusammenhängenden Rahmen: (Nur FARB-VIDEO)

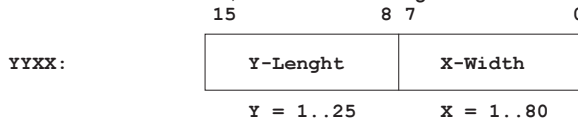
HTOP      DEV,01010, " [ " ,10 ;   
HTOP      DEV,01110, " [ " ,10 ;   
HTOP      DEV,01210, " [ " ,10 ; 

**Extends:**      Nur VIDEO! Nicht mit Rahmen!  
Folgende HTOP-Erweiterungen dienen vor allem der Soft-Window

Generierung: (AND,OR und XOR sind auch ohne RCT möglich!)

Ext	Funktion	Code
RCT	Rectangle	10
OR	OR Color	20
AND	AND Color	40
XOR	XOR Color	60

RCT: Lösche ein Rechteck, wobei N wie folgt neu definiert wird:



OR: Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV mit der OR-Funktion verknüpft z.B. TEXT hervorheben (+I-Bit) DEV = 088XX , SRC = 000

AND: Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV maskiert.  
 z.B. Gross -> Kleinschreibung (A..Z): DEV = 0FFxx , SRC = 020  
 z.B. Window-Schatten (Hintergrund = sw): DEV = 00Fxx , SRC = 0FF

XOR: Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV mit der XOR-Funktion verknüpft.  
 z.B. Hintergrund von Blau auf Hell-Rot: DEV = 0D0xx , SRC = 000

Beispiel 4: Markiere die Zeile 5 : vorher Grau auf Blau (COL=47), nacher Weiss auf Hell-Rot (COL=9F) -> XOR-COL=D8!

HTOP 0D800,0500,0,80,XOR

Beispiel 5: Oeffne ein SOFT-Window mit Schatten-Wurf auf DEV-0: Grösse in R10=YYXX, Position in POS, Hintergrund Blau

```

OPEN:  PUTD  0,1                ; Save Screenin Speicher 1
        ADD  0102,POS           ; Schatten 2-rechts, 1-
        unten                  ; HTOP
        0700,POS,0FF,R10,RCT,AND ; Schatten (Grau auf
        Schwarz)
        SUB  0102,POS           ; Original-POS
        HTOP 04F00,POS,' ',R10,RCT ; Clear Window (Blau)
        RTM  0                  ; ( Close mit GETD 0,1 )
  
```

## VTOP

Vertikal Text OutPut

94xx \_\_xx      VTOP      DEV,POS,SRC,N,EXT

**Erklärung:**      Gebe das Zeichen SRC in der Position POS N-mal vertikal aus.  
Update POS entsprechend EXT.

N = pos    von oben nach unten(rollt wenn nötig)  
N = neg    von unten nach oben(kein zurück rollen!)

**Extends:**      Wie beim HTOP sind folgende Extends für Window-Effekte möglich:

Ext	Funktion	Code
OR	OR Color	20
AND	AND Color	40
XOR	XOR Color	60

**OR:**      Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV mit der OR-Funktion verknüpft.

**AND:**      Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV maskiert.  
z.B. Window-Schatten (Hintergrund = sw): DEV = 00Fxx , SRC = 0FF

**XOR:**      Der auf dem Bildschirm stehende Text wird mit SRC und dessen Farbe wird mit COL in DEV mit der XOR-Funktion verknüpft.  
z.B. Hintergrund von Blau auf Hell-Rot: DEV = 0D0xx , SRC = 000

**Beispiel 1:**      Zeichne eine Linie " || " am linken und rechten Bildrand:

```
VTOP    DEV,00," || ",25    ; links
VTOP    DEV,80," || ",25    ; rechts
```

**Beispiel 2:**      Zeichne den rechten Schatten eines 10-Zeilen Windows:

```
VTOP    0700,POS,0FF,10,AND
```

## ZTOP\_

### Zahlen Text OutPut

96xx \_\_xx      ZTOP      DEV,POS,SRC,FORM,EXT  
 95xx \_\_xx      ZTOPD     DEV,POS,SRC:D,FORM,EXT

Erklärung:      Wandle die HEX-Zahl in SRC in eine ASCII-DEZ-Zahl gem. FORM.  
 Anzeige der DEZ-Zahl auf DEV an der Position POS. Update POS  
 entsprechend EXT.

Format:      FK      Mit dem Format-Byte kann die Fenster-Grösse definiert werden und  
 wo ein Dezimalpunkt eingefügt werden soll.

Das Format-Byte ist wie folgt aufgebaut: FK  
 F Fenster-Grösse  
 K Anzahl Kommastellen  
 Hat die Zahl (inkl Vorzeichen und Komma) im Fenster nicht Platz, so  
 wird das ganze Fenster '\*\*\*\*\*'.

Extends:      Folgende Extends können hinter die ZTOP-Befehle geschrieben  
 werden:

Ext	Funktion	Code
A	Zahl Absolut (nur bei WORT)	00
S	Zahl mit Signum (nur bei WORT)	20
L	Ausgabe Linksbündig	10
R	Ausgabe Rechtsbündig	00
G	Anzeige GROSS	80
ASC	Kopie nach ASCII-Puffer (Für RTIP)	40

Absolute:      A      Beim ZTOP ohne EXT oder mit EXT = 'A' wird immer eine Zahl  
 von 0..65535 ausgegeben.

Signed:      S      Beim ZTOP mit EXT = 'S' wird eine Zahl mit Vorzeichen von  
 +- 32767 ausgegeben.  
 Beim ZTOPD wird immer eine Zahl mit Vorzeichen von max  
 +- 99999999 ausgegeben.

- Links:      L    Bei EXT = 'L' wird die Zahl bei POS beginnend ausgegeben und POS danach ans Zahlen-Ende gesetzt. Das Fenster wird nicht weiter mit SPACES aufgefüllt.
- Rechts:     R    Bei keinem EXT oder EXT = 'R' wird die Zahl rechtsbündig ins Fenster geschrieben und leere Vorstellen mit SPACES gelöscht. POS wird danach ans Fensterende gesetzt.
- GROSS:      G    Bei EXT = 'G' gelten die gleichen Bedingungen wie beim GTOP.
- ASCII:      ASC    Bei EXT = 'ASC' wird eine Kopie der angezeigten Zahl in den ASCII-Puffer gelegt (für RTIP).
- Beispiel 1:            Schreib die Zahl ' -1234.567' mit drei Kommastellen in ein Fenster von 11 Zeichen rechtsbündig. Lege eine Kopie davon in den ASCII-Puffer Update die Position in POS mit 1 SPACE nach der Zahl:  
  
             ZTOPD    DEV,POS,-1234567,0B3,R,ASC,SP
- Beispiel 2:            Schreib die Zahl in REG 00 absolut und linksbündig:  
  
             ZTOP     DEV,POS,R00,050,NP,L

## XZTOP

Hex-Zahlen Text OutPut

97xx \_\_xx      XZTOP    DEV,POS,SRC,FORM,EXT

**Erklärung:**      Wandle die HEX-Zahl in SRC in eine ASCII-HEX-Zahl gem. FORM.  
Anzeige der HEX-Zahl auf DEV an der Position POS. Update POS  
entsprechend EXT.

**Format:**            FD Mit dem Format-Byte kann die Fenster-Grösse definiert werden  
und wieviele Digits ausgegeben werden sollen.  
Das Format-Byte ist wie folgt aufgebaut: FD  
F Fenster-Grösse  
D Anzahl Digits      (max 8)  
Wenn D>8 ist, werden D-8 Vornullen ausgegeben.

**Extends:**            Folgende Extends können hinter den XZTOP-Befehl geschrieben  
werden und funktionieren wie beim ZTOP:

Ext	Funktion	Code
L	Ausgabe Linksbündig	10
R	Ausgabe Rechtsbündig	00
G	Anzeige GROSS	80
ASC	Kopie nach ASCII-Puffer (Für RTIP)	40

**Beispiel 1:**        Zeige die 32-Bit Zahl in REG 01,00 an mit 2 führenden SPACES und  
update POS auf die nächste Zeile:

XZTOP    DEV,POS,R00,0B9,R,CRLF

**Beispiel 2:**        TOP GROSS '    1234' in ein Fenster von 7 Zeichen:

XZTOP    DEV,POS,01234,074,R,G



## TIP

### Text InPut

98xx \_\_x0      TIP DEV,POS,TADR,EXT

**Erklärung:**      Lese den eingegebenen Text von DEV nach TADR bis 'CR' getippt wird. Gebe ein ECHO an der Position POS aus (SIO nur bei EXT='E'). Werden mehr Zeichen als ASL eingegeben, werden die überzähligen einfach ignoriert. Beim TIP-Ende wird APO=0 gesetzt. Update POS entsprechend EXT.

**Extends:**      Nur für SIO-32 und 2K-SIO !

Ext	Funktion	Code
E	Echo für Terminal-Eingabe	20

Dabei wird jedes eingelesene Zeichen wieder ausgegeben. Der 'LF' als Eingabe wird immer ignoriert. Wird 'CR' als Abschluss eingegeben, so wird 'CR/LF' als Echo ausgegeben.

**UmTast:**      Im System können alle Tasten über die UmTast-Tabelle auf andere Tasten umgeleitet werden (z.B. Klein- auf Grossbuchstaben).

**Kontrolltasten:**      Tastencodes 00...01F sind Kontrolltasten. Wird eine dieser Tasten in der ersten TIP-Position (APO=0) betätigt, so wird der TIP sofort abgeschlossen und die ENTER-Taste (RETURN) muss nicht mehr betätigt werden. Der Code steht dann als erstes Zeichen im ASCII-Buffer. Wurden schon vorher Zeichen getippt, so wird die Taste wie andere behandelt und der Tip muss mit RETURN abgeschlossen werden.

Spezial-Tasten:	X'08      BACK SPACE	X'1E      FORWARD SPACE
	X'09      TABULATOR	X'1F      INSERT SPACE
	X'0D      RETURN	X'7F      DELETE CHARA

**Funktionstasten:**      Tastencodes 80...8F werden automatisch durch den Text im FKT-Buffer ersetzt und erlauben das Belegen von Funktionstasten mit beliebigem Text oder Spezial-Codes. Pro Taste sind im FKT-Buffer 31 Zeichen reserviert. Ein @ im FKT-Buffer wird als Auto-Return interpretiert.

**Beispiel:**      Lese den eingegebenen Text in den ASCII-Puffer:

TIP      DEV,POS,ASC

## JTIP

### Jump Text InPut

98xx 1\_x0 SAD JTIP DEV,POS,TADR,SAD,EXT

**Erklärung:** Lese den eingegebenen Text genau wie beim TIP, aber springe solange auf SAD bis 'CR' als TIP-Ende getippt wird.

**VIDEO** Die Tastatur wird nur bei jedem JTIP-Durchgang bedient. Der eingegebene Text wird vorzu auf TADR abgelegt. APO dient der Cursor-Steuerung und darf während dem JTIP nicht verändert werden (ACMP).  
TOP während JTIP möglich!

**SIO-32:** Der Text wird im Interrupt eingelesen und vorzu auf TADR abgelegt. APO dient als Text-Pointer und darf während dem JTIP nicht verändert werden (ACMP). !  
Kein TOP während JTIP möglich !

**2k-SIO:** Der Text wird nur bei jedem JTIP-Durchgang aus dem FIFO abgeholt und vorzu auf TADR abgelegt. Der JTIP springt beim ersten Aufruf immer auf SAD! Wird 'CR' empfangen, so wird über Interrupt das DELAY-Halt-Bit gelöscht (TIP mit Zeitüberwachung möglich).  
TOP während JTIP möglich!

**Abbruch:** Der JTIP kann frühzeitig mit CTIP,SETD,INID und RESD abgebrochen werden.

**Beispiel 1:** Warte bis eine beliebige Taste am VIDEO-0 gedrückt wird:

```

MOV      0,ASC ; Clear ASC
WT12:   CBR      ASC,><,0,CONTI ; Irgend etwas getippt ?
        JTIP     0,0406,ASC,WT12 ; Tastatur-Abfrage
CONTI:   CTIP    0 ; Lösche laufenden JTIP

```

**Beispiel 2:** Get-String Routine für die 2K-SIO mit Zeitüberwachung: Der Task belastet während der ganzen Eingabe-Zeit das System nicht!

```

GETS:   JTIP     8,0,ASC,WAIT ; Get String
        RTM      1 ; TIP-OK, skip BRA ERROR
WAIT:   DELAY   6000 ; TIME-OUT Zeit = 60 sec ; HALT
        CBR     TIM,<>,0,GETS ; TIME-OUT Zeit abgelaufen ?
TIMOUT: CTIP    8 ; Clear JTIP
        RTM     0 ; TIP-ERROR, BRA ERROR

```

## RTIP

Re-initialisiere Text InPut

99xx 00xx      RTIP      DEV,POS,TADR

**Erklärung:**      Ausgabe vom Text-String TADR auf DEV an der Position POS. Setze den Cursor auf die Position 'APO' Bereite die TIP-Parameter auf wie wenn dieser Text eingetippt worden wäre.

**Beachte:**      Nach RTIP muss immer ein TIP oder JTIP folgen !

**APO:**      Das APO-Register bestimmt, wo der Cursor hinkommt:  
 APO = 0      Cursor auf Text-Anfang  
 APO = N      Cursor auf Zeichen N  
 APO = OFF      Cursor am Text-Ende

**ZTOP:**      Wird eine Zahl mit dem ZTOP mit EXT='ASC' vorgegeben, kann diese mit dem RTIP einem TIP zum editieren übergeben werden.

**ACMP,ABR:**      Der RTIP erlaubt es, eine Eingabe zu wiederholen, die bei einem ACMP oder ABR-Befehl nicht erfolgreich war. Da diese Befehle immer das APO-Register nachführen, steht der Cursor danach automatisch auf der falschen Eingabe.

**Beispiel:**      TOP die Zahl in R00, lasse sie editieren oder mit RETURN bestätigen. Teste sie auf 0...200.0 und wiederhole den TIP bis sie in diesen Grenzen ist. Die neue Zahl steht nachher wieder in R00.

ERRI:	ZTOP	DEV,POS,R00,051,ASC,L,S	; Vorgabe der Zahl
	RTIP	DEV,POS,ASC	; TIP re-init
	TIP	DEV,POS,ASC,OP	; Zahl editieren
	ABR	051,R00,0,2000,ERRI	; Test Zahl auf MIN,MAX

## BTIP

Block Text InPut

9Dxx \_\_x0 SAD BTIP DEV,TBLK,N,SAD

Erklärung : Lese N Zeichen nach TBLK von der 2K-SIO. Spring solange auf SAD, bis N Zeichen empfangen wurden. Mit diesem Befehl können alle Zeichen von 00..FF ohne Einschränkungen empfangen werden.

Die Zeichen werden nur bei jedem BTIP-Durchgang aus dem FIFO abgeholt und vorzu auf TBLK abgelegt. Der BTIP springt beim ersten Aufruf immer auf SAD ! Wurden N Zeichen empfangen, so wird über Interrupt das DELAY-Halt-Bit gelöscht (BTIP mit Zeitüberwachung möglich). TOP während BTIP möglich !

Beispiel : Empfange 5-Byte Steuersequenz und schreibe sie in den ASC Puffer.

## **GRAPHIK-Befehle**

## GRAPHIK FGV

- FCV+FGV:** Die Graphik-Karte FGV bildet zusammen mit der Charakter-Karte FCV ein DEVICE. Die FGV wird mit einem speziellen Kabel an der FCV angeschlossen. Die Hardware-Adresse der FGV ist immer um 0100 höher als die der zugehörigen FCV-Karte.
- ACHTUNG:** Werden Graphik-Befehle auf ein DEVICE ausgeführt und die FGV ist nicht vorhanden, bleibt der Rechner stehen!
- CLEAR DEVICE:** Siehe auch CRLD-Befehl. Der Clear für Text-Schirm und Graphik-Schirm sind unabhängig voneinander!
- WINDOWS:** Wird das Hardware-Window Device auf dem Charakter-Bildschirm geöffnet, so wird genau gleich gross auch ein Graphik-Window gebildet.  
Folgende Einschränkungen ergeben sich jedoch Hardware-bedingt:
- Es können nicht gleichzeitig Graphik-Befehle im Window- und im Base-Device ausgeführt werden! Soll im Window-Device auch Graphik erzeugt werden, ist der Task auf dem Base-Device anzuhalten. Beim freigeben muss der ORG-Befehl auf dem Base-Device neu aufgerufen werden. (Die ORG-Position wird nicht doppelt geführt und bei Continuous-Befehlen kann die End-Position des letzten Graphik-Befehls verloren gehen.)
- Die FGV-Karte hat einen Speicher von 512k-Pixel. Daher bleiben im 640x500-Pixel Mode (320k-Pixel) nur noch 192k aktive Pixel. Ist das eröffnete Window grösser, so wird das Fenster zwar in der richtigen Grösse ausgeblendet, die unteren Pixel-Zeilen können jedoch nicht angesprochen werden! (Bei 25x80 Zeichen ist das in etwa das untere Drittel des Bildschirms)
- SOFT-Windows:** Die SCREEN-SAVE Befehle PUTD,GETD retten nur die Daten vom Charakter-Bildschirm! Die Graphik bleibt unverändert!
- SPLIT-SCREEN:** Die Aufteilung des Charakter-Bildschirms in UPPER-, BASE und LOWER-SCREEN ist für den Graphik-Bildschirm ohne Bedeutung! Er belegt immer den ganzen Schirm.
- FARBEN:** Die Farben des Graphik-Bildes werden auf der FCV mit einer EXOR-Funktion mit den Farben des Charakter-Bildes verknüpft. Wenn also z.B. der Charakter-Schirm einen blauen Hintergrund hat und auf dem Graphik-Schirm eine blaue Linie gezeichnet wird, erscheint sie auf dem Schirm schwarz.

## ACRTC-MODI

MODI: Die Graphik-Befehle basieren alle auf dem Befehlssatz des Advanced CRT Controller ACRTC HD 63484 von HITACHI. Da dieser Chip eine Vielfalt von Operations-Modis hat, würde eine vollständige Beschreibung aller Möglichkeiten den Rahmen dieser Beschreibung sprengen. Zur Programmierung von Graphiken müssen Sie sich daher unbedingt das folgende Buch von HITACHI besorgen:

**User's Manual HD63484 , 14-13A (680-1-311A) , HITACHI**

Folgende MODI können als Extensions hinter dem Befehl angegeben werden: (Es muss immer etwas angegeben werden, zB ...,0)

OPM	*	0 0 0	REPLACE
		0 0 1	OR
		0 1 0	AND
		0 1 1	EOR
		1 0 0	CONDITIONAL REPLACE (Read Data = CCMP)
		1 0 1	CONDITIONAL REPLACE (Read Data <> CCMP)
		1 1 0	CONDITIONAL REPLACE (Read Data < CL)
		1 1 1	CONDITIONAL REPLACE (Read Data > CL)
COL	*	0 0	When Patter RAM data = 0, Color Register 0 is used
		0 1	When Patter RAM data = 1, Color Register 1 is used
		1 0	When Patter RAM data = 0, drawing is suppressed
		1 1	When Patter RAM data = 1, Color Register 1 is used
			When Patter RAM data = 1, drawing is suppressed
			Pattern RAM contents are directly used as color data
AREA	*	0 0 0	Drawing is executed without checking
		0 1 0	Drawing suppressed outside Area
		1 1 0	Drawing suppressed inside Area
		x x x	(weitere Modes sind nicht unterstützt!)
REL (R)	*	0	Absolut Position
		1	Relative Position
C	*	0	Counterclockwise
		1	Clockwise

\* Für einfache Graphiken reicht es aus, wenn die Befehls-Modi (AREA,COL,OPM...) einfach immer 0 gesetzt werden.

Die MODIs können auch in die EQUAL-Liste aufgenommen werden und sind dann über deren Name ansprechbar:

```
REL      =      0400
C        =      0100
```

## ORG

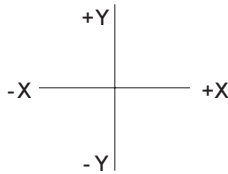
ORiGin

9Fxx 1000 MODI ORG      DEV,Y|X

**Erklärung:**      Setze den Null-Punkt für alle Graphik-Befehle auf Y|X. Dieser Befehl muss immer nach SETD aufgerufen werden!

**Beachte:**      X und Y sind immer als positive Zahlen anzugeben!  
 X sind die Anzahl Pixel von linken Bildschirm-Rand aus.  
 Y sind die Anzahl Pixel von oberen Bildschirm-Rand aus.

Nach dem ORG-Befehl gilt folgende Positions-Zählweise:



Beim ORG-Befehl wird die Pattern-Definition immer so gesetzt, dass ausgezogene Linien und voll ausgefüllte Flächen gezeichnet werden!  
 (PRA-0 = 0FFFF ; Pr-05..07 = 0000)

**Beispiel:**      Setze den Graphik-Nullpunkt genau in die Bildschirm-Mitte (VGA):

ORG      DEV,250|320

**KOORDINATEN:** Die Koordinaten eines Punktes werden bei allen Graphik-Befehlen immer als DOUBLE-WORD angegeben:      YYYYY'XXXX  
 Will man YYYYY und XXXX immediate angeben, können sie auch als zwei einzelne (DEZ-)Werte geschrieben werden, wenn man sie mit dem |-Zeichen (|=ALT124) getrennt aneinander schreibt:

Y=250 , X=320 :      250|320 == 00FA0140



## DOT

DOT

9Fxx 0000 MODI DOT      DEV,Y|X,MODI

MODI:	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Erklärung:      Setze ein Graphik-Pixel (DOT) an der Position Y|X.  
 Da bei allen Linien-Befehlen der letzte DOT nicht gezeichnet wird,  
 muss dieser bei Bedarf mit dem DOT-Befehl gesetzt werden.

Beispiel 1:      Setze ein rotes Pixel auf die Position Y=25,X=30:

DOT      0100,25|30,0

Beispiel 2:      Setze ein Pixel mit der Farbe in DEV und der Position  
 Y=R11,X=R10:

DOT      DEV,R10,0

## LINE

LINE

9Fxx 01x0 MODI LINE      DEV,sY|sX,eY|eX,MODI

F E D C    B A 9 8    7 6 5 4    3 2 1 0

**MODI:**

0 0 0 0	0 R 0 0	AREA		COL		OPM
---------	---------	------	--	-----	--	-----

**Erklärung:**      Zeichne eine Linie von sY|sX bis eY|eX. Die Linie ist ein Pixel breit. Gestrichelte Linien können mit Hilfe des Pattern-Rams erzeugt werden. (Der letzte DOT wird nicht gezeichnet!)

Bei REL ist sY|sX absolut und eY|eX eine Delta-Position bezogen auf sY|sX!

**Beispiel 1:**      Zeichne eine weiße 45-Grad Linie von sY|sX=10|10 bis eY|eX = 50|50:

LINE      0F00,10|10,50|50,0

**Beispiel 2:**      Zeichne eine Linie von sY|sX=10|20 mit einem dY|dX=5|5: (eY|eX kommt also auf 15|25)

LINE      DEV,10|20,5|5,REL

**Beispiel 3:**      Zeichne eine Linie von sY|sX=R11,R10 mit einem dY|dX in R13|R12: (eY|eX kommt auf R11+R13,R10+R12)

LINE      DEV,R10,R12,REL

## PLINE

### PolyLINE

9Fxx 02xx MODI PLINE DEV,sY|sX,n,SRC,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	R	0	0	AREA		COL		OPM			

**Erklärung:** Zeichne eine zusammenhängende Linie (Polyline) von sY|sX mit n weiteren Punkten aus der Tabelle in SRC. Pro Punkt wird ein Double-Word Eintrag in SRC benötigt! (Der letzte DOT wird nicht gezeichnet!)

Bei REL ist sY|sX absolut und alle weiteren Angaben eine Delta-Position bezogen auf den vorhergehenden Punkt.

**Beispiel:** Zeichne ein Dreieck mit der Breite 20 und der Höhe 15-Pixel auf die Position sY|sX in R11|R10:

```
DRK: .DOUBLE 00|20,15|-10,-15|-10 ; Relative Punkte
      PLINE DEV,R10,3,@DRK,REL
```

## CLINE

Continuous (poly)LINE

9Fxx 03x0 MODI CLINE      DEV,n,SRC,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0 0 0 0				0 R 0 0		AREA				COL				OPM	

**Erklärung:**      Zeichne vom Endpunkt des letzten Graphik-Befehls aus eine zusammenhängende Linie mit n weiteren Punkten aus der Tabelle in SRC. (Der letzte DOT wird nicht gezeichnet!)

Bei REL beziehen sich alle Positionen auf den vorhergehenden Punkt.

**Beispiel:**      Zeichne eine Kurve mit X=0..500 und dX=10 mit den Y-Koordinaten berechnet in RAM mit der Adresse in [R12]: (R10 used)

	MOV	0,R10		;	ERSTE X-KOORDINATE
	MOV	[R12]+1,R11		;	ERSTE Y-KOORDINATE
	DOT	DEV,R10,0		;	ESTER PUNKT
LOOP:	ADD	10,R10		;	NÄCHSTE X-KOORDINATE
	MOV	[R12]+1,R11		;	NÄCHSTE Y-KOORDINATE
	CLINE	DEV,1,R10,0		;	LINIE BIS ZUM NÄCHSTEN PUNKT
	CBR	500,>,R10,LOOP		;	FERTIG ?

## RCT

ReCTangle

9Fxx 04x0 MODI RCT      DEV,sY|sX,eY|eX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0 R 0	0	AREA		COL		OPM					

Erklärung:      Zeichne einen rechteckigen Rahmen mit dem Start-Punkt sY|sX und dem Diagonal-Punkt eY|eX:

Bei REL ist sY|sX absolut und eY|eX eine Delta-Position bezogen auf sY|sX!

Beispiel 1:      Zeichne einen Rahmen von 0|0 mit der Breite 400 und der Höhe 300 Pixel:

RTC      DEV,0|0,300|400,0

Beispiel 2:      Zeichne einen Rahmen von der Start-Position in R11|R10 mit der Breite 400 und der Höhe 300 Pixel:

RTC      DEV,R10,300|400,REL

## FRCT

Filled ReCTangle

9Fxx 05x0 MODI FRCT      DEV,sY|sX,eY|eX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	R	0	0	AREA		COL		OPM			

Erklärung:      Zeichne ein ausgefülltes Rechteck mit dem Start-Punkt sY|sX und dem Diagonal-Punkt eY|eX.

Bei REL ist sY|sX absolut und eY|eX eine Delta-Position bezogen auf sY|sX!

Beispiel:      Zeichne einen Balken mit der Breite 16 Pixel (= 2 Charakter) und der Höhe berechnet in R11:

```
MOV      16,R10                    ; BALKEN-BREITE
FRCT     DEV,1|1,R10,REL         ; ZEICHNE BALKEN
```

## CFRCT

Continuous Filled ReCTangle

9Fxx 0600 MODI CFRCT DEV,eY|eX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	R	0	0	AREA		COL		OPM			

**Erklärung:** Zeichne ein ausgefülltes Rechteck an den End-Punkt des letzten Graphik-Befehls und dem Diagonal-Punkt eY|eX.

Bei REL ist eY|eX eine Delta-Position bezogen auf den End-Punkt des letzten Graphik-Befehls.

**Beispiel:** Zeichne einen weiteren Balken mit der Breite 8 Pixel und der Höhe 50 Pixel:

CFRCT DEV,50|8,REL

## CRCL

CiRCLe

9Fxx 07x0 MODI CRCL DEV,cY|cX,R,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	0	0	C	AREA		COL		OPM			

Erklärung: Zeichne einen Kreis mit dem Mittel-Punkt cY|cX und dem Radius R.

Beispiel: Zeichne einen Kreis mit dem Radius 100 um den Nullpunkt:

CRCL DEV,0|0,100,0



## ARC

ARC (Kreissegment)

9Fxx 08xx MODI ARC      DEV,sY|sX,cY|cX,eY|eX,MODI

	F E D C	B A 9 8	7 6 5 4	3 2 1 0
<b>MODI:</b>	0 0 0 0	0 R 0 C	AREA   COL   OPM	

**Erklärung:**      Zeichne ein Kreissegment mit dem Start-Punkt sY|sX, dem Mittel-Punkt cY|cX und den Endpunkt eY|eX.

Bei REL ist sY|sX absolut und cY|cX,eY|eX sind Delta-Positionen bezogen auf sY|sX!

**Beispiel 1:**      Zeichne einen oberen Halbkreis mit dem Radius 100 Pixel:

ARC      DEV,0|0,0|100,0|200,C      ; CLOCKWISE

**Beispiel 2:**      Zeichne den unteren Halbkreis mit dem Radius 100 Pixel:

ARC      DEV,0|0,0|100,0|200,0      ; COUNTERCLOCKWISE

**Bemerkung:**      Da alle Linienbefehle den letzten DOT nicht zeichnen, fehlt jetzt immer noch ein Pixel in 0|200 !

## CARC

Continuous ARC

9Fxx 09x0 MODI CARC DEV,cY|cX,eY|eX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	R	0	C	AREA		COL		OPM			

**Erklärung:** Zeichne ein Kreissegment mit dem Start-Punkt gleich dem End- Punkt des letzten Graphik-Befehls, dem Mittel-Punkt cY|cX und dem Endpunkt eY|eX.

Bei REL sind cY|cX und eY|eX Delta-Positionen bezogen auf den End-Punkt des letzten Graphik-Befehls.

**Beispiel:** Zeichne ein Kuchen-Stück mit Radius = 100 Pixel und 90 Grad:

```

LINE    DEV,0|0,0|100,0    ; ERSTE LINIE
CARC    DEV,0|0,100|0,0    ; KREISSEGMENT
LINE    DEV,100|0,0|0,0    ; ZWEITE LINIE

```

## ELPS

### ELliPSe

9Fxx 0Axx MODI ELPS      DEV,cY|cX,b|a,dX,MODI

F E D C      B A 9 8      7 6 5 4      3 2 1 0

**MODI:**

0 0 0 0	0 0 0   C	AREA   COL   OPM	
---------	-----------	------------------	--

C

\* 0 Counterclockwise  
1 Clockwise

Erklärung:

Zeichne eine Ellipse mit dem Mittel-Punkt cY|cX, einem Radius in der X-Richtung von dX und dem Achsen-Verhältnis  
a : b = dX<sup>2</sup> : dY<sup>2</sup>

Dieser Befehl dient auch als Kreis-Befehl, wenn die Bildschirm-Verzerrung aufgehoben werden soll.

Beispiel:

Zeichne eine Ellipse mit den Achsen dX=100, dY=50:  
(a : b = 10000 : 2500)

ELPS      DEV,50|100,2500|10000,100,0

## CEARC

Continuous Ellips ARC

9Fxx 0Bxx MODI CEARC DEV,b|a,cY|cX,eY|eX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0 R 0 C	AREA		COL		OPM						

C \* 0 Counterclockwise  
1 Clockwise

Erklärung: Zeichne ein elliptisches Kreissegment mit dem Start-Punkt gleich dem End-Punkt des letzten Graphik-Befehls, dem Mittel-Punkt cY|cX, dem Endpunkt eY|eX und dem Achsen-Verhältnis  $a : b = dX^2 : dY^2$

Bei REL sind cY|cX und eY|eX Delta-Positionen bezogen auf den End-Punkt des letzten Graphik-befehls.

Bemerkung: Da der EARC mit dem Startpunkt 5 Parameter benötigen würde, gibt es diesen Befehl nur als Continuous-Befehl.

Beispiel: Zeichne eine geschlossene halbe Ellipse, doppelt so hoch wie breit: (a : b = 1 : 4)

```
LINE    DEV,0|0,0|100,0    ; BODEN, 100 PIXEL
CEARC   DEV,4|1,0|50,0|0,0 ; 1/2 ELLIPSE DARUEBER
```

## PTN

## PaTterN

9Fxx 0Dx0 MODI PTN      DEV,sY|sX,szYX,MODI

(MODI siehe CPTM-Befehl!)

SL , SD:

SL	SH	0 0 0	0 0 1	0 1 0	0 1 1
0					
SL	SD	1 0 0	1 0 1	1 1 0	1 1 1
0					
SL	SH	0 0 0	0 0 1	0 1 0	0 1 1
1					
SL	SD	1 0 0	1 0 1	1 1 0	1 1 1
1					

Erklärung: Das Graphik-Pattern definiert im Pattern-Ram mit der Rechteck-Grösse szYX wird an der Position sY|sX gezeichnet.

Beispiel: Zeichne einen Charakter aus dem Pattern-Ram mit 8x16 Pixel:

PTN      DEV,0|0,01008,060

## CPTN

Continuous PaTterN

9Fxx 0E00 MODI CPTN      DEV,szYX,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	SL	SD	AREA			COL	OPM					
<b>szYX:</b>	Size Y								Size X							

**Erklärung:** Das Graphik-Pattern definiert im Pattern-Ram mit der Rechteck-Grösse szYX wird an der End-Position des letzten Graphik-Befehls gezeichnet.

**Beispiel:** Zeichne einen weiteren Charakter aus dem Pattern-Ram mit 8x16 Pixel:

PTN      DEV,01008,060

## PAINT

### PAINT

9Fxx 0C00 MODI PAINT DEV,sY|sX,MODI

MODI :	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	E	AREA	0	0	0	0	0	0

E \* 0 The edge color is defined by the data in the EDG register.  
 1 The edge color is defined to be all colors except for the color in the EGD register

Erklärung: Fülle eine geschlossene Fläche gebildet durch Linien mit der Farbe=Vordergrund-Farbe von DEV mit der Hintergrund-Farbe in DEV.

Beachte: Die Beiden Color-Register CL0 und CL1 werden beide mit der Hintergrund-Farbe aus DEV geladen, das Edge-Color-Register mit der Vortergund-Farbe aus DEV!  
 Der Befehl füllt jede beliebige Form indem er sie mit horizontalen Linien auffüllt, kann sich jedoch maximal vier Unterbrechungs-Punkte merken, wo er weiter machen muss. Ist die Form zu komplex, müssen die fehlenden Segemente mit weiteren PAINT-Befehlen gefüllt werden.

Beispiel 1: Zeichne einen grünen Kreis und fülle ihn mit Rot auf:

```
CRCL    0200,50|50,50,0    ; WEISSER KREIS
PAINT   01200,50|50,0      ; MIT ROT FUELLEN
```

Beispiel 2: Fülle den selben Kreis mit grün auf:

```
PAINT   02200,50|50,0      ; MIT ROT FUELLEN
```

## GCPY

Graphic CoPY

9Fxx 0Fxx MODI GCPY      DEV,tY|tX,sY|sX,dY|dX,MODI

F E D C      B A 9 8      7 6 5 4      3 2 1 0

**MODI:**

0	0	0	R		S		DSD		AREA		0		0		OPM
---	---	---	---	--	---	--	-----	--	------	--	---	--	---	--	-----

**Erklärung:**      Der Graphik-Bereich mit dem Start-Punkt sY|sX und der Ausdehnung dY|dX wird nach tY|tX kopiert.  
 Mit Hilfe der Scan-Richtung S und der Dest-Scan-Direction DSD kann das Bild beliebig gedreht und gespiegelt werden.

**Beachte:**      Das \*REL-Bit R ist bei diesem Befehl das Bit-12!  
 Bei REL bezieht sich der Start-Punkt sY|sX auf den Punkt tY|tX.

**Beispiel:**      Rolle ein Graphik-Fenster mit 100x100 Pixel um ein Pixel nach links (Kurven-Schreiber):

GCPY      dev,0|0,0|1,100|100,01000 ; REL=01000 !



## WRDPR

WRite Drawing Parameter Register

9Fxx 13x0 MODI WRDPR DEV,n,SRC,MODI

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>MODI:</b>	0	0	0	0	0	0	0	0	0	0	0	0	R	N		

**Erklärung:** Lade das Drawing Parameter Register RN und n folgende mit den Daten von SRC.

**Beachte:** Die beiden Color-Register werden bei allen Graphik-Befehlen automatisch wie folgt geladen: (Ausnahme siehe PAINT)  
 CL0 = Background-Farbe aus DEV  
 CL1 = Foreground-Farbe aus DEV

Beim ORG-Befehl werden die Pattern RAM Controll Register Pr-05..07 alle auf 0000 gesetzt (ausgezogene Linien)!

Die Register Pr-0C bis Pr-1F (RWP,DP,CP) sind nicht beschreibbar!

**Beispiel:** Setze das Color-Compare Register (Pr-02) auf Rot:  
 (Die Farbe mus immer 4-fach definiert werden, 4-Pixel pro WORD)

WRDPR DEV,1,01111,02

## WRPTN

WRite PaTern Register

9Fxx 15x0 MODI WRPTN DEV,n,SRC,MODI

MODI:	F E D C	B A 9 8	7 6 5 4	3 2 1 0
	0 0 0 0	0 0 0 0	0 0 0 0	P R A

**Erklärung:** Lade das Pattern-RAM Register PRA und n folgende mit den Daten von SRC.

**Beachte:** Beim ORG-Befehl wird das PRA-0 immer auf 0FFFF gesetzt (ausgezogene Linien)!

**Beispiel:** Lade ein Pattern-Muster PRA-0..F aus einer Tabelle [R10]:

WRPTN DEV,16,[R10],0

## **ASCII-Befehle**

## ABR\_

Ascii → BinaRy

```
A0xx _0xx SAD ABR      K,DEST,MIN,MAX,SAD,EXT
B0xx 00xx SAD ABRD     K,DEST:D,MIN:D,MAX:D,SAD
A2xx 00xx SAD ABRF     DEST:F,MIN:F,MAX:F,SAD
B2xx 00xx SAD ABRL     DEST:L,MIN:L,MAX:L,SAD
```

Erklärung:

1. Suche im ASCII-Puffer ab Position APO nach einer DEZ-Zahl.
2. Wird eine Zahl gefunden, update APO, sonst springe nach SAD.
3. Wandle die ASCII-Zahl in eine Binär-Zahl.
4. Multipliziere sie mit  $K*10$  (bei Integer,  $K$ =Anz Komma-Stellen).
5. Konvertiere sie in eine HEX-Integer oder Floating Point Zahl.
6. Springe auf SAD wenn MIN oder MAX überschritten wird.
7. Werden die Grenzen eingehalten, schreibe die Zahl nach DEST.
8. Update APO ans Zahl-Ende.

Extends: Beim ABRD,ABRF,ABRL wird die Zahl immer mit Vorzeichen getestet und es gibt keine Extends.

Folgende Extends können hinter den ABR-Befehl geschrieben werden:

EXT	Funktion		Code
A	Zahl Absolut	0..65535	00
S	Zahl mit Signum	+/-32767	20

Format: K Anzahl Kommastellen.  
 K = 0..xxxF (es werden nur 4-Bits verarbeitet!)

Beachte: Alle ABR-Befehle lesen die ASCII-Zahl mit der selben Routine ein, erst dann wird sie in das gewünschte Format gewandelt. Diese Routine erkennt zum Beispiel folgende Zahlen richtig:

```
123      .356  -123.456   1E6  -123.456-E3
```

ACHTUNG: Es werden immer maximal 9 relevante Dezimalstellen verarbeitet, weitere Ziffern dienen nur zur Bildung des Exponenten! Dies betrifft vor allen den ABRL für LONG-Floating Point.

Beispiel:

Mit einem TIP wurde folgender ASCII-String in den ASC-Puffer gelesen (APO = 0):

```

      0           1
Position: 0123456789ABCDEF012345678
ASC:      QVW -123.45678 XYZ 153

```

Lade die erste Zahl, normiert in 1/1000 (3 Komma Stellen) nach REG 01,00. Teste ob Zahl in den Grenzen +- 500.000 ist:

```
ABRD    03,R00,-500000,500000,SAD
```

Im REG 01,00 steht nun FFFE,1DC0 (== -123456'DEZ).

APO ist jetzt 0E !

Lade jetzt die nächste Zahl, normiert in 1/10 (1 Komma-Stelle) nach REG 02. Teste auf 0...100.0:

```
ABR     1,R02,0,1000,SAD,S
```

Da jetzt die Testbedingung nicht erfüllt ist, springt der Macro Program Counter auf SAD.

In APO ist jetzt 014 !

Teste nun mit 0...200.0:

```
SAD:    ABR     1,R02,0,2000,SAD,S
```

In REG 02 steht nun 05FA (1530'DEZ).

In APO ist jetzt 017 !

Ein weiterer ABR geht jetzt immer auf SAD, da keine weitere Zahl mehr im ASCII-Puffer steht.

## XABR\_

heX-Ascii —> BinaRy

A1xx 00x0 SAD XABR DEST,MIN,MAX,SAD

B1xx 00X0 SAD XABRD DEST:D,MIN:D,MAX:D,SAD

Erklärung:

1. Suche im ASCII-Puffer ab Position APO nach einer HEX-Zahl.
2. Wird eine Zahl gefunden, update APO, sonst springe nach SAD.
3. Wandle die ASCII-Zahl in eine Binär-Zahl.
4. Springe auf SAD wenn MIN oder MAX überschritten wird.
5. Werden die Grenzen eingehalten, schreibe die HEX-Zahl nach DEST.
6. Update APO ans Zahl-Ende.

Beachte: Der XABR\_ liest alles sobald er eine Ziffer 0..F findet!

z.B. JOHANN ergibt die Hex-Zahl 0A

Beispiel: Mit einem TIP wurde folgender ASCII-String in den ASC-Puffer gelesen (APO = 0):

QVW 0A57B

Lade die HEX-Zahl nach REG 00 und teste auf 0...0B000:

XABR R00,0,0B000,SAD ; 0A57B => R 00

APO ist jetzt 09 !

## ACMP

Ascii CoMPare

A4xx SAD      ACMP      BTAB,DEST,SAD

Erklärung:      1. Suche im ASCII-Puffer ab Position APO nach einem Text.  
                   2. Wird ein Text gefunden, update APO, sonst springe nach SAD.  
                   3. Vergleiche den Text mit der Befehls-Tabelle BTAB.  
                   4. Steht kein identischer Text in der BTAB springe nach SAD.  
                   5. Wird der Befehl gefunden, schreibe den Wert darunter nach DEST.  
                   6. Update APO ans Text-Ende.

Beachte:        DEST kann auch direkt der Macro-Program-Counter MPC sein !

Beispiel:        Mit einem TIP wurde folgender ASCII-String in den ASC-Puffer  
                   gelesen (APO = 0):

HELP STATUS

Teste den ersten String im ASCII-Puffer, ob er in der BTAB  
 aufgeführt ist. Wenn ja, lade REG 00 mit dem entsprechenden Wert.  
 Wenn nein, springe nach SAD.

ACMP      @BTAB,R00,SAD

In REG 00 steht jetzt 2000      ; APO ist jetzt 07 !  
 Teste den nächsten String und springe auf die entsprechende  
 Adresse.

ACMP      @BTAB,MPC,SAD

Der Task steht jetzt auf 03000      ; APO ist jetzt 0E !  
 Jeder weitere ACMP springt jetzt auf SAD, da kein weiterer Text  
 mehr da ist.

Befehls-Tabelle:

BTAB:      .TXT      'DEBUG'      ; String 1





## TIME-Befehle

## TIME

get/set TIME

B4x\_            TIME        ART,ADRE

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

Code	ART	Uebergabe	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.	OYYY'YDNR	01990116
8	STIM	SET TIME	HHMM'SSZZ	16075198
9	SDAT	SET DATE	YYYY'MMDD	19900426
A	SDOW	SET LANGUAGE & DAY OF WEEK	0L0D	0004

Language:        Der Wochentag kann in mehreren Sprachen angezeigt werden:  
L: 0 = German , 1 = English , 2 = Italien , 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-Buffers '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

## FLOPPY-Befehle

## FLOPPY-FORMATE

### 720kB Disk:

Disk	Format	Sektor	Clust	Trac	Zylin.	Seite	Disk
Bytes		512	1 0 2 4	4 6 0 8	9 2 1 6	6 8 6 4 0	7 3 7 2 8 0
Sektoren			2	9	1 8	7 2 0	1 4 4 0
Cluster				4 . 5	9	3 6 0	7 2 0
Tracks					2	8 0	1 6 0
Zylinder							8 0

Boot Record : 1 Sektor  
 File Allocation Table : 2 \* 3 Sektoren  
 Root Directory : 7 Sektoren

### 1.44MB Disk:

Disk	Format	Sektor	Clust	Trac	Zylin.	Seite	Disk
Bytes		512	512	9216	18432	737280	1474560
Sektoren			1	18	36	1440	2880
Cluster				18	36	1440	2880
Tracks					2	80	160
Zylinder							80

Boot Record : 1 Sektor  
 File Allocation Table : 2 \* 9 Sektoren  
 Root Directory : 14 Sektoren

**Kompatibilität:** Mit der FDC-2-Karte ist es möglich 3 1/2" DOS-Disketten der Formate 720KB und 1.44MB zu lesen und zu schreiben. Die Format-Erkennung geschieht automatisch. Es stehen auch alle Möglichkeiten des Subdirectory-Handlings zur Verfügung.

**SYST-Belastung:** Da die Karte einen eigenen Mikroprozessor besitzt, wird das System selbst bei intensiven Befehlen wie FORMAT praktisch nicht belastet.

## DIRECTORY

Einträge: Maximale Anzahl Einträge pro Directory :

```

720KB : - Root Directory      112 Einträge
          - Subdirectory      224 Einträge
1.44MB: - Root Directory      224 Einträge
          - Subdirectory      224 Einträge
  
```

Als Eintrag gilt: FILE , SUBDIRECTORY , VOLUME-LABEL

Bei Subdirectories sind jeweils die ersten zwei Einträge fürs DOS reserviert.

Hinweis: Die Begrenzung der Anzahl Einträge in einem Subdirectory ist der einzige Punkt der Nichtkompatibilität zu DOS. (DOS erlaubt in Subdirectories beliebig viele Einträge)

Aufbau: Jeder Eintrag umfasst 32 Byte und ist folgendermassen aufgebaut:

Offset	Grösse	Bedeutung	
00h	8	Byte	File Name
08h	3	Byte	File Extension
0Bh	1	Byte	File Attribut
0Ch	10	Byte	reserviert (nicht gebraucht)
16h	1	Word	Zeit des letzten Update
18h	1	Word	Datum des letzten Update
1Ah	1	Word	Startclusternummer
1Ch	2	Word	File Grösse

File Name: 00h -> noch nie benützt, es folgt kein weiterer Eintrag!  
E5h -> gelöschter Eintrag

Attribut:

```

Bit 0   read only
  1     hidden
  2     system
  3     volume label
  4     subdirectory
  5     archive
  6     reserved
  7     reserved
  
```

Zeit:

```

Bit 0..4 2 Sekunden Inkrement
Bit 5..A Minuten
Bit B..F Stunden
  
```

Datum:

```

Bit 0..4 Tag
Bit 5..8 Monat
Bit 9..F Jahr relativ zu 1980
  
```

## PFAD

**Pfad:** Jeder Task hat im System seinen eigenen aktuellen Pfad, d.h. es kann mit mehreren Tasks (jeder hat einen anderen aktuellen Pfad) auf einer Floppy-Disk gleichzeitig gearbeitet werden, ohne jedesmal den Pfad angeben zu müssen.

**Beachte:** Jeder Task, der mit der Floppy arbeitet, muss sich zu Beginn mit dem Befehl CHDIR seinen eigenen Pfad setzen.

Beispiel: `PATH: .TXT 'A:\INDEL\SYSTEM\DATEN'`  
`CHDIR @PATH`

Weiter gelten beim Pfad die selben Regeln wie bei DOS.

**Drive Name:** A: B:  
 Wird kein Drive angegeben, wird der letzte weiter verwendet.

**Directory Name:** besteht aus Name und Extension genau gleich wie der File Name.  
 Beispiel: SUB.DIR

**File Name:** besteht aus Name und Extension.  
 Name: 1..8 beliebige Zeichen ausgenommen:  
 . " / \ [ ] | < > + = ; , und alle < 21h  
 Extension: 1..3 Zeichen wie oben  
 Beispiel: FILE.NAM

**NAME:** In der Befehlsbeschreibung soll unter 'NAME' folgendes verstanden werden :

NAME = [d:][Pfad] name  
 [] -> optionale Angabe eines Laufwerks bzw. eines Pfades.

Enthält NAME eine Laufwerk- bzw. Pfadangabe, so ist nach Ausführung des Befehls der aktuelle Pfad gleich dem Angegebenen  
 Enthält NAME keine Laufwerk- bzw. Pfadangabe, so wird der Befehl auf dem aktuellen Laufwerk im aktuellen Directory ausgeführt.

**Beispiele:** Anwendung aller Floppy-Disk Befehle finden Sie im File IFDOS !

## Disk ERRORS

ERRORS: Tritt ein Disk-Error auf, springt der Task auf seine ABORT- Adresse.  
Die ERROR-Nummer wird im Register APO übergeben:

APO	Beschreibung
1	File existiert nicht
2	File existiert schon
3	Die Floppy-Disk ist voll
4	-
5	File ist WRITE geschützt
6	-
7	Block-Nummer existiert nicht
8	Drive nach 1 sec nicht READY
9	Floppy-Disk ist WRITE geschützt
10 **	SEEK oder RECAL Fehler
11 **	READ oder WRITE Fehler
12	ungültiges Verzeichnis
13	ungültiger Filename
14	Verzeichnis voll
15	Verzeichnis nicht leer
16	File kann nicht in ein Verzeichnis gewandelt werden
17	unbekannte Diskkapazität
18	Diskcopy nur auf Disketten gleichen Formats erlaubt

\*\* DP8473 Daten werden übergeben :

R70	Status - 1	Status - 0
R71	---	Status - 2
R72	Track Nummer	Drive Nummer
R73	Sector Nummer	Kopf Nummer

**DELETE**

DELETE file

8Fx0           DELETE NAME

Erklärung:       Lösche das File 'NAME'.

Beispiel:         Lösche das File 'MSI.EXE' in 'A:\INDEL\NS32':

FILE:            .TXT       'A:\INDEL\NS32\MSI.EXE'

DELETE @FILE



## RENAME

RENAME file

8Exx            RENAME NAME,NEWNAME

Erklärung:      Ändere den Namen von 'NAME' auf 'NEWNAME'.

Beispiel:        Ändere den Filename 'MSI.EXE' auf 'NAME.NEU':

NFIL:            .TXT        'NAME.NEU'  
RENAME @FILE,@NFIL

## DISK

DISK space

8Dxx            DISK        DRIVE,DEST:D

Erklärung:        Rechne die freien Bytes auf DRIVE nach DEST.

Beispiel:         Rechne den freien Platz auf 'A:' und schreibe es ins R00:

DISK        'A:',R00

## FILE

FILE parameter

8Cxx            FILE        NAME,DEST:32-BYTE

Erklärung:        Die 32-Bytes des Directory-Eintrages des Files 'NAME' werden nach DEST gelesen.

Beispiel:         Der Directory-Eintrag von 'MSI.EXE' wird in den ASC-Puffer gelesen:

```
FILE:            .TXT        'A:\INDEL\NS32\MSI.EXE'  
FILE            @FILE,ASC
```

## GATR

Get ATtRIBUTE

8Bxx            GATR     NAME,DEST:B

Erklärung:        Kopiere das Attribut-Byte von 'NAME' nach DEST.

Beispiel:         Kopiere das Attribut von 'MSI.EXE' ins R00:

```
FILE:            .TXT        'A:\INDEL\NS32\MSI.EXE'  
GATR            @FILE,R00
```

## SATR

Set ATtribute

8Axx            SATR     NAME, SRC:B

Erklärung:        Kopiere SRC in das Attribut-Byte von 'NAME'.

Beispiel:         Setze neues Attribut-Byte von 'MSI.EXE':

```
FILE:            .TXT        'A:\INDEL\NS32\MSI.EXE'  
SATR            @FILE,020
```

## READ

READ file

88xx 00x0      READ      NAME,DEST,AZB:D

Erklärung:      Lese 'AZB'-Bytes von 'NAME' nach DEST.

Beachte:        AZB wird als Double-Word gelesen!

Beispiel:        Lese 'MSI.EXE' nach Adresse 0A000 in ganzer Länge:

FILE:            .TXT            'A:\INDEL\NS32\MSI.EXE'

READ            @FILE,@0A000,-1

## WRITE

WRITE file

88xx 01x0      WRITE      NAME, SRC, AZB:D

Erklärung:      Schreibe 'AZB'-Bytes von SRC nach 'NAME'.

Beachte:        AZB wird als Double-Word gelesen!

Beispiel:        Eröffne ein File (NAME in ASC) und schreibe ab Adresse  
0A000,02000 Bytes in dieses File:

WRITE      ASC, @0A000,02000

## APPEND

APPEND file

88xx 02x0      APPEND NAME,SRC,AZB:D

Erklärung:      Erweitere 'NAME' um 'AZB'-Bytes von SRC.

Beachte:        AZB wird als Double-Word gelesen!

Beispiel:        Erweitere das File (NAME in ASC) um [R44]-Bytes ab Adresse in R33:

APPEND ASC,[R33],R44



## RDBLK

ReaD Block

88xx 03x0      RDBLK    NAME,DEST,BLK#

Erklärung:      Lese den Block 'BLK#' aus 'NAME' nach DEST.  
Jeder Block ist 1024-WORD lang. Der erste Block ist BLK-0. Die  
überzähligen Bytes im letzten Block sind undefiniert!

Beispiel:        Lese den Block #2 aus dem File (NAME IN ASC) nach 0B000:

RDBLK    ASC,@0B000,2

## WRBLK

WRite BLock

88xx 04x0      WRBLK    NAME,SRC,BLK#

Erklärung:      Ueberschreibe den Block 'BLK#' in 'NAME' mit SRC.  
Jeder Block ist 1024-WORD lang. Der erste Block ist BLK-0. Das File  
kann mit diesem Befehl nicht verlängert werden !

Beispiel:        Ueberschreibe den Block #2 in 'MSI.EXE' von Adresse 0B000:

FILE:            .TXT        'A:\INDEL\NS32\MSI.EXE'  
WRBLK    @FILE,@0B000,2

## COPY

COPY file

88xx 0500 COPY NAME,NAME1

Erklärung: Kopiere 'NAME' nach 'NAME1'.  
Für NAME1 gilt dieselbe Definition wie für NAME, d.h. NAME1 kann auch optional Laufwerk und Pfad enthalten.

Beispiel: Kopiere 'MSI.EXE' nach 'MSI.BAK':

```
FILE: .TXT      'A:\INDEL\NS32\MSI.EXE'  
KOPF: .TXT      'B:\SYSTEM\BACKUP\MSI.BAK'  
  
COPY  @FILE,@KOPF
```

## DCOPY

Disk COPY

88xx 0600      DCOPY    SRCDRIVE,DESTDRIVE

Erklärung:      Kopiere die gesamte Disk SRCDRIVE auf die Disk DESTDRIVE.  
Es werden alle Files,inklusive Subdirectories kopiert. DCOPY ist nur auf Disketten vom gleichen Format erlaubt und ist nur mit zwei Drives möglich (DCOPY auf dem gleichen Drive durch wechseln der Floppy ist nicht möglich) !

Beispiel:        Kopiere 'A:' nach 'B':

DCOPY    'A:','B:'

## CHDIR

CHange DIRectory

88x0 0700 CHDIR [d:]PFAD

Erklärung: Wechsle das Verzeichnis.

Beispiel: Wechsle das Verzeichnis:

PFAD1: .TXT '\SYSTEM\NS32\2KSIO'

PFAD2: .TXT '2KSIO'

CHDIR @PFAD1 ;aktueller Pfad = '\SYSTEM\NS32\2KSIO'  
;wir befinden uns im Verzeichnis '2KSIO'

CHDIR '..' ;aktueller Pfad = '\SYSTEM\NS32'  
;wir befinden uns im Verzeichnis 'NS32'

CHDIR @PFAD2 ;aktueller Pfad = '\SYSTEM\NS32\2KSIO'  
;wir befinden uns im Verzeichnis '2KSIO'

## MKDIR

MaKe DIRectory

88x0 0800 MKDIR NAME

Erklärung: Erstelle das Verzeichnis 'NAME'.

Beispiel: Erstelle das Verzeichnis 'TEST':

DIRECT: .TXT 'TEST'

MKDIR @DIRECT

## RMDIR

ReMove DIRectory

88x0 0900      RMDIR      NAME

Erklärung:      Entferne das Verzeichnis 'NAME'.

Beispiel:        Entferne das Verzeichnis 'TEST':

DIRECT: .TXT 'TEST'

RMDIR      @DIRECT

## DIR

DIRectory

88xx 0Ax0 DIR PFAD,DEST,AZB

Erklärung: Lese AZB-Bytes der Einträge des Directories unter PFAD nach DEST.

Beispiel: Lese alle Directory-Einträge von 2KSIO nach 0A000:

PFAD: .TXT 'SYSTEM\NS32\2KSIO'

DIR@PFAD,0A000,-1



## PATH

PATH

88xx 0B00      PATH      PFAD,DEST

Erklärung:      Lese den aktuellen PfAD nach DEST.

Beispiel:      Lese den aktuellen Pfad in den ASC-Puffer:

PATH      0,ASC

## FORMAT

FORMAT disk

88xx 0C00      FORMAT DRIVE,ART

Erklärung:      Formatiere die Floppy im Laufwerk 'DRIVE' mit der Kapazität ART.  
ART = 0 -> 720KB ; ART = 1 -> 1.44MB, alle anderen Werte für ART  
sind ungültig !

Beispiel:      Formatiere die Floppy in Drive 'A:' auf 1.44MB:

## **MASTER/SLAVE- Protokoll**

## MASTER/SLAVE

Baud Rate: Die Baud Rate wird beim SETS und SETM wie folgt angegeben:

	7		4	3			0
BAUD :	odd	PEn	2SB	8DB	XON	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/PEGEL	EVEN	ODD
6	PARITY/PEGEL	DIS	EN
5	STOP BITS	1	2
4	DATA BITS	7	8
3	XON-XOFF	Nein	JA

PEGEL:      PARITY = EN , EVEN/ODD      Normale SIO-Pegel (RS232,RS422)  
               PARITY = DIS , ODD      Normales SIO-Pegel (RS232,RS422)  
               PARITY = DIS, EVEN      INVERSE SIO-Pegel (20mA passiv-Betrieb)

Wenn die MASTER/SLAVE-Karten im 20mA passiv-Betrieb arbeiten sollen (galvanische Trennung), muss PARITY = DIS,EVEN gesetzt werden (MASTER und SLAVE)!

Im RS232- oder RS422-Betrieb ist dieser Mode nicht sinnvoll, da die Pegel dann falsch sind!

## MASTER

- Slave-Nr.** Die Slave-Nummer kann 0..255 sein. Dabei sind jedoch die Nummern 128..255 für Erweiterungen reserviert (Transfer über GATEWAY).
- Adressen:**
- |                |         |    |          |                   |
|----------------|---------|----|----------|-------------------|
| PACE/6809      | 0000    | .. | 0FFFF    |                   |
| NS32016/SIO32  | 0'0000  | .. | 03'FFFF  | (0..3 in STAT)    |
| NS32016/2k-SIO | 00'0000 | .. | 0FF'FFFF | (3'tes ADRE-Byte) |
- Anz. Words:** Es können max 256 Words pro Transfer übertragen werden.
- TIME-OUT:** Die Time-Out Zeit belegt den 10-ms Timer und wird dementsprechend in 10-ms Einheiten angegeben. Läuft diese Zeit während PUT oder GET ab, so wird der versuchte Transfer abgebrochen und ein neuer Versuch gestartet.
- Anz. Versuche:** Wurde ein Transfer durch Time-Out oder durch Störungen abgebrochen, werden entsprechend dieser Angabe viele Versuche unternommen, den Transfer doch noch zustande zu bringen. Kommt trotzdem kein vollständiger Transfer zustande, springt der Task auf SAD und übergibt eine Error-Nummer im APO-Register.
- Errors:** Nach Anz. Versuchen springt der Task bei folgenden Errors auf SAD und in APO steht die ERROR-Nummer:
- | APO | Beschreibung                                    |
|-----|---|
| 030 | Slave antwortet überhaupt nicht!                |
| 031 | Framing Error                                   |
| 032 | Parity Error                                    |
| 033 | Overrun-Error                                   |
| 034 | Unzulässige Zeichen empfangen (not HEX-Input)   |
| 035 | Slave antwortet, aber Time-Out während Transfer |
- Steuer-Leitungen:** Die Steuerleitungen DTR,RTS,CTS,DSR und DCD werden genau wie bei TIP/TOP mit der 2k-SIO behandelt !  
Wird dies nicht gebraucht, CTS,DSR,DCD -> +5..15V.  
(Der DCD-Eingang wird bei 20mA passiv-Betrieb nicht beachtet.)
- XON/XOFF:** Wird der XON/XOFF-Betrieb beim SETS/SETM gewählt, so funktioniert er genau wie bei TIP/TOP mit der 2k-SIO.

## SETS

SET Slave

B5xx 0000      SETS      BAUD,SLAVE

Erklärung:      Initialisiere die SLAVE-SIO mit dem Uebertragungs-Format BAUD und  
schärfe sie auf die Slave-Nummer SLAVE.  
Enable den SLAVE-Interrupt!

Beispiel 1:      Setze die Baudrate 9600,n,7,1 , und die Slave-Nummer 1:  
(20mA passiv-Betrieb)

SETS      05,01

Beispiel 2:      Setze die Baudrate 9600,n,7,1 , und die Slave-Nummer 1:  
(RS232 oder RS422-Betrieb)

SETS      085,01

## SETM

### SET Master

B5xx 01x0      SETM      BAUD,TIMOUT,ANZ\_VERS

Erklärung:      Initialisiere die MASTER-SIO mit dem Uebertragungs-Format BAUD,  
setze die Time-Out Zeit auf TIMOUT und setze die Anzahl Versuche  
auf ANZ\_VERS.  
Enable den MASTER-Interrupt!

Beispiel 1:      Setze die Baudrate 9600,n,7,1 , die Time-Out Zeit auf 1 sec und die  
Anzahl Versuche pro Aufruf auf 5:  
(20mA passiv-Betrieb)

SETM      05,100,5

Beispiel 2:      Setze die Baudrate 9600,n,7,1 , die Time-Out Zeit auf 1 sec und die  
Anzahl Versuche pro Aufruf auf 5:  
(RS232 oder RS422-Betrieb)

SETM      085,100,5

## PUT

PUT data

B5xx 02xx SAD PUT SLAVE,ANZ,VON,NACH:D,SAD

Erklärung:       Sende ANZ-WORD von der Adresse VON zum SLAVE auf die Adresse NACH. Springe auf SAD wenn der Transfer nicht zustande kommt.  
(TIMOUT und ANZ\_VERS gemäss SETM ; ERROR Nummer in APO).

Beachte:         Die Adresse NACH gilt als Adresse im Slave und wird daher eine Adressierungs-Stufe höher angegeben als normal!  
Es wird immer ein DOUBLE-WORD für NACH benötigt!

Beispiel:         Sende REG-20..Reg-30 zum SLAVE-01 auf die Adresse 01'A000:  
PUT           01,010,R20,01A000,ERROR



## GET

GET data

B5xx 03xx SAD GET SLAVE,ANZ,VON:D,NACH,SAD

Erklärung: Hole ANZ-WORD auf der Adresse VON vom SLAVE auf die Adresse NACH Springe auf SAD wenn der Transfer nicht zustande kommt. (TIMOUT und ANZ\_VERS gemäss SETM ; ERROR Nummer in APO).

Beachte: Die Adresse VON gilt als Adresse im Slave und wird daher eine Adressierungs-Stufe höher angegeben als normal!  
Es wird immer ein DOUBLE-WORD für VON benötigt!

Beispiel: Hole 16-Words aus dem SLAVE-01 von Adresse 041'F070 in den ASCII-Buffer:

GET 01,16,041F070,ASC,ERROR

## PROTOKOLL

**Kompatibilität:** Das Master/Slave Protokoll ist kompatibel zu allen vorherigen Master/Slave-Versionen von INDEL AG und es können Master und Slaves jeder Generation an der selben Leitung betrieben werden.

**IBM-PC:** Ein Software-Treiber auf IBM-kompatiblen PCs ist vorhanden. Soll das Protokoll mit anderen Rechnern betrieben werden, fragen Sie uns oder verwenden Sie folgende Protokoll-Beschreibung!

**PUT DATA (Master an Slave)**

<b>MASTER:</b>	E O T	# L N Q R	S E L N	*	S O H T R A A A E C C	S O H T R A A A E C C	S O H T R A A A E C C	S D D D D E C C	T A A T S S	X T T X U U	A A A M M	E O T
<b>CSUM:</b>					Z-B-WW-B—WW	—WW//WW—						
<b>SLAVE:</b>				S L N K R				A C K				A C K

**GET DATA (Master an Slave)**

<b>MASTER:</b>	E O T	# L N Q R	S E L N	*	S O H T R A A A E C C	S O H T R A A A E C C	S O H T R A A A E C C	S D D D D E C C	T A A T S S	X T T X U U	A A A M M	A C K O T
<b>CSUM:</b>					Z-B-WW-B—WW	—WW//WW—						
<b>SLAVE:</b>				S L N K R				A C K	S D D D D E C C	T A A T S S	X T T X U U	A A A M M

Steuer-Zeichen:	#	2E	Slave-Nummer folgt
	ENQ	05	Enquiry
	ACK	06	Acknowledge
	SOH	01	Start of Heading
	ETX	03	End of Text
	EOT	04	End of Transmission
	NAK	15	Not Acknowledge

Der Transfer kann jederzeit mit EOT abgebrochen werden. Der Master überwacht den ganzen Transfer mit einer Timeout-Zeit (Timeout-Zeit = theoretische Übertragungs-Zeit \* 1.5). Wird diese Zeit überschritten (zB Slave antwortet nicht), wird mit EOT abgebrochen und ein neuer Aufruf gestartet.

Alle andern Zeichen werden in ASCII übertragen, damit der Transfer mit einem Standard-Terminal überwacht werden kann. Es wird immer das höchstwertige Byte als erstes gesendet.

SLNR: Die Slave-Nummer SLNR (00..7F) gibt an, welcher Rechner angesprochen werden soll (080..0FF sind reserviert).

STAT: Als Status sind folgende Bytes zulässig: S-I/O Karten

01 = PUT	ohne	RKNR		2K-SIO,SIO-32,M6809
02 = GET	ohne	RKNR		2K-SIO,SIO-32,M6809
R1 = PUT	ohne	RKNR,	R = Rack-Nummer	2K-SIO,SIO-32
R2 = GET	ohne	RKNR,	R = Rack-Nummer	2K-SIO,SIO-32
*09 = PUT	mit	RKNR		2K-SIO
*0A = GET	mit	RKNR		2K-SIO

RKNR,ADRE: Die Adresse ADRE gibt immer die niedrigste WORD-Adresse des Daten-Blocks an.

Bei STAT = 01,02 ist ADRE eine 16-Bit Adresse.

Bei STAT = R1,R2 bildet R und ADRE zusammen eine 20-Bit Adresse

Bei STAT = 09,0A bildet RKNR und ADRE zusammen eine 24-Bit ADR

Die Racknummer R wird eigentlich nur bei Adressen  $\geq 1'0000$  das Zusatzbyte RKNR nur bei Adressen  $\geq 10'0000$  benötigt. Es kann aber auch generell mit RKNR (STAT=09,0A) gefahren werden!

ANZW: ANZW gibt die Anzahl zu transferierenden WORDs an (01..FF). Ist ANZW=00, so werden 256-WORDS transferiert!

CHECKSUM: Die Checksum ist ein WORD gross. Die übertragenen BYTES oder WORDs werden ohne Beachtung des Ueberlaufes addiert.

Z	= CSUM wird h'0000 gesetzt
B	= ein Byte wird zu CSUM addiert (hex)
WW	= ein Word wird zu CSUM addiert (hex)

Steuerzeichen (SOH,ETX,ACK) werden nicht zu CSUM addiert!  
In der zweiten CSUM ist auch die CSUM vom Header enthalten!

## PUT - Beispiel

Sende 100-Worte (alle 0000) an Slave Nr. 01 auf Adresse h'041A000

		MASTER		SLAVE		CSUM	
		ASCII	HEX-BYTES	ASCII	HEX-BYTES	ADD	HEX
		EOT	04				
SLAVE-NR	#		23				
	01		30,31				
	ENQ		05				
				31	33,31		
				ACK	06		
PUT RACK-NR ADRESSE ANZ WORDS CSUM	SOH		01				0000
	09		30,39				+0009 = 0009
	41		34,31				+0041 = 004A
	A000		41,30,30,30				+A000 = A04A
	64		36,34				+0064 = A0AE
	ETX		03				+A0AE = 415C
				ACK	06		
DW 1 DW 2..99 DW 100 CSUM	STX		02				
	0000		30,30,30,30				+0000 = 415C
	...						...
	0000		30,30,30,30				+0000 = 415C
	ETX		03				
415C		41,31,35,43					
				ACK	06		
		EOT	04				

## GET - Beispiel

Hole 2-Worte (1234,5678) von Slave Nr. 01 von Adresse h'04189AB

	MASTER		SLAVE		CSUM	
	ASCII	HEX-BYTES	ASCII	HEX-BYTES	ADD	HEX
	EOT	04				
SLAVE-NR	#	23				
	01	30,31				
	ENQ	05				
			31 ACK	33,31 06		
GET	SOH	01				0000
RACK-NR	0A	30,41				+000A = 000A
ADRESSE	41	34,31				+0041 = 004B
ANZ WORDS	89AB	38,39,41,42				+89AB = 89F6
	02	30,32				+0002 = 89F8
CSUM	ETX	03				+89F8 = 13F0
	89F8	38,39,46,38				
			ACK	06		
DATA-W1			STX	02		
DATA-W2			1234	31,32,33,34		+1234 = 2624
			5678	35,36,37,38		+5678 = 7C9C
CSUM			ETX	03		
			7C9C	37,43,39,43		
			ACK	06		
	EOT	04				



## **3964R-Protokoll**

## 3964R

**Voraussetzungen:** Um einen 2k-SIO-Kanal als 3964R-Port laufen zu lassen, benötigen Sie eine 2k-SIO mit Rev. 2.80 oder höher und mindestens die Betriebssystemversion ISM 4.31 mit dem dazugehörigen Makroassembler Rev. 4.3, 910730.

**ACHTUNG:** Die Tasks, welche die 3964R-Befehle verwenden, müssen diese zu Beginn mit `!.INCLUDE 3964R.INC` dem Makroassembler bekannt machen.  
Das Modul 3964R.OBJ muss im Betriebssystem auf die Nummer gelinkt sein, die im 3964R.INC definiert ist oder umgekehrt.

**Device Nummer:** Die Devicenummern werden von 0 an gezählt, d.h. die erste 2k-SIO im System hat die 3964R-Devicenummer 0, die zweite die Nummer 1 usw. Die max. Anzahl 3964R.32k mit dem Equal NOPTS angegeben werden und ist somit theoretisch nicht begrenzt.

**Baud Rate:** Die Baud Rate wird beim 3964R-Protokoll 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	-	-



**Adressen:** Die Adressen im Kommunikationspartner werden lauf dem 3964R-Standard mit Hilfe von Datenbaustein und Datenwort angegeben. Für beide steht jeweils ein Byte zur Verfügung. In der Indel-Implementierung werden die beiden folgendermassen in ein Wort zusammengefasst:



Die Definitionen, wo die jeweiligen Datenbausteine im IPS-32 liegen, können im File 3964r.32k unter dem Label DBADR vorgenommen werden. Für jeden der 256 möglichen Datenbausteine ist hier ein Doppelwort reserviert.

**Anzahl Wort:** Theoretisch können bis zu 65535 Worte übertragen werden.

**Timeout:** Die Timeout-Ueberwachung belegt den 10ms Timer, d.h. während der Transfers kann dieser Timer nicht anderweitig verwendet werden.

**Errors:** Tritt während der Uebertragung ein Fehler auf, so springt der Task auf SAD und in APO steht die entsprechende Fehlernummer.

APO	Beschreibung
030	Partner antwortet nicht
031	Framing Error
032	Parity Error
033	Overrun Error
034	Reaktionstelegramm nicht empfangen
035	Errornummer im Reaktionstelegramm nicht gleich
Null	- > Errornummer im R70 low byte.

**Merkers:** Der jetzige Stand der Software fährt ohne Merker's, d.h. im Telegrammkopf wird OFFh für Merkerbyte und -bit gesendet.

**Steuer-Leitungen:** Die Steuerleitungen DTR, RTS, CTS, DSR und DCD werden genau wie bei TIP/TOP mit der 2k-SIO behandelt!  
Wird dies nicht gebraucht, CTS, DSR, DCD - > +5..15V.  
(Der DCD-Eingang wird bei 20mA passiv-Betrieb nicht beachtet.)

**XON/XOFF:** Dieser Betrieb ist mit 3964R nicht möglich.

**SET39M**

SET3964R Master

SET39M BAUD/DEV

Erklärung: Initialisiere die 2k-SIO Nr. DEV als 3964R - Master mit der Baudrate BAUD. Der Master besitzt bei einem Kommunikationskonflikt die Leitungspriorität.

Hinweis: Die Devicenummern werden hier von 0.. angegeben, d.h. die erste 2k-SIO im System hat die 3964R-Nummer0, die zweite die Nummer 1 usw.

Beispiel 1:: Setze die 2k-SIO Nummer 1 als 3964R-Master mit der Baudrate 38400,e,8,1.

SET39M 05701

**SET39S**

SET 3964R Slave

SET39S BAUD/DEV

Erklärung: Initialisiere die 2k-SIO Nr. DEV als 3164R - Slave mit der Baudrate BAUD. Der Slave besitzt bei einem Kommunikationskonflikt keine Leitungspriorität.

Hinweis: Die Devicenummern werden hier von 0.. angegeben, d.h. die erste 2k-SIO im System hat die 3964R-Nummer 0, die zweite die Nummer 1 usw.

Beispiel: Setze die 2k-SIO Nummer 3 als 3964R-Slave mit der Baudrate 600,n,7,2.

SET39S 02101

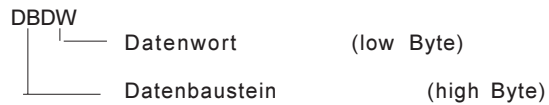
**AD\_39**

AusgabeDaten 3964R

AD\_39 DEV, ANZ, VON, NACH: W, SAD

Erklärung:        Sende ANZ Worte von der Adresse VON zum Kommunikationspartner auf die Adresse NACH. Springe auf SAD, wenn der Transfer nicht zustande kommt (Error Nummer im APO).

Hinweis:           Die Adresse NACH setzt sich folgendermassen zusammen:



Beispiel:         Sende R20..R30 zum Partner auf Datenbaustein 7, Datenwort 5.

AD\_39 DEV, 010, R20, 0705, ERROR

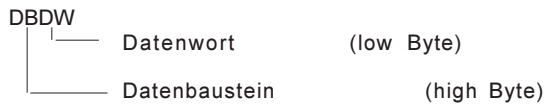
**ED\_39**

EingabeDaten 3964R

ED\_39 DEV, ANZ, VON: W, NACH, SAD

Erklärung: Hole ANZ Worte von der Adresse VON vom Kommunikatonspartner auf die Adresse NACH. Springe auf SAD, wenn der Transfer nicht zustande kommt (Error Nummer im APO).

Hinweis: Die Adresse VON setzt sich folgendermassen zusammen:



Beispiel: Hole 2 Worte vom Partner von Datenbaustein 3, Datenwort 0 in den ASC - Puffer.

ED\_39 DEV, 2, 0300, ASC, ERROR



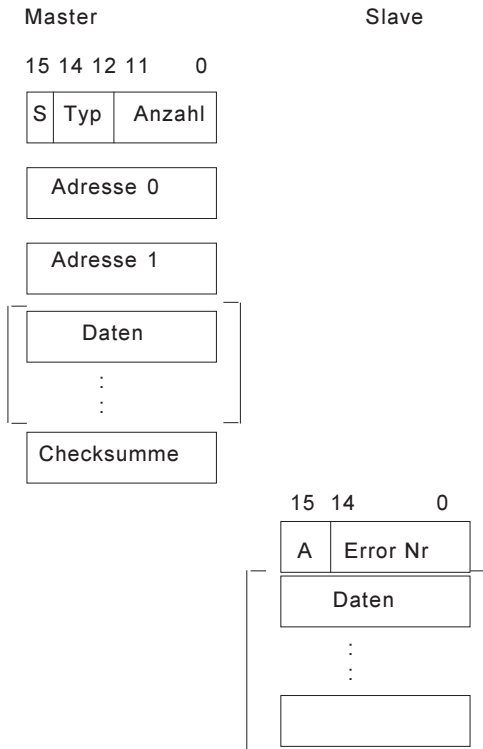
## **Info\_Master-Slave-Protokoll**

## 16-Bit Protokoll

**Voraussetzung:** Um diese Funktionen benutzen zu können, benötigen Sie die Masterkarten mit der Software Rev. 2.7 oder höher, für den InfoPC-Master das Modul info\_com.32k und für das Rack das Modul ips\_com.32k.

**Beschreibung:** Es wurden 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 Debugzwecke oder für Parameterdefinitionen verwendet werden.

**Aufbau des Protokolles:**





- Checksumme
- 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 Types Byte, Word oder DWord's. 0 entspricht  $2^{12} = 4096$ .
- Adresse 0: Lo-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 Datentypes.
- A: Antwortstatusbit der Slavekarte.  
A = 0 bedeutet: NACK, Checksumme war falsch.  
A = 1 bedeutet: ACK, Chesiumme 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 Befehlsblock wird für die Blockfunktionen 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 Kartenummer, Achsen oder Ausgängen

15 14 12 11 43 0

	Typ	Adresse	
--	-----	---------	--

**Anzahl Datenelemente:** Bit 15 Kennzeichen für Spezialblock 1, sonst 0  
 Bit 11-0 Anzahl zu schreibende Byte's / Word's / DWord's  
 Bereich von 0-4095. ( 0 = 4096 )

15 14 12 11 0

Spez.		Anzahl
-------	--	--------

**Quelladresse:** Byteadresse des Buffers der zu übertragenen Byte's / Word's oder DWord's.  
 (bei F\_PUTB ist diese Adresse im Speicher, bei F\_GETB auf der Karte)

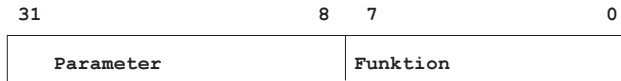
**Endadresse:** Destinationsbyteadresse an der die Daten geschrieben werden.  
 (bei F\_PUTB ist diese Adresse auf der Karte, bei F\_GETB im

## 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 Task 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.

**Übergabeparameter:** 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 Kommunikationskanal 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 wieder frei

Übergabeparameter: Kommunikationsdeskriptors

Rückgabe: -

Besonderes: Bei Fehler sprung auf Abort

Beispiel: Freigeben des vorher reservierten Kommunikationskanal.  
Deskriptor in P\_COMCH.

```

P_COMCH = R22           ; Kommunikationsdeskriptor (dword)
:
:
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-Byte's / -Word's / -DWord's 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.

**Übergabeparameter:** 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 Herz-Zähler des Siemensregler, 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       ; Buffer auf Reg R0
ASHD     1,SRCADR        ; auf Byteadresse anpassen
MOVD     07FE00040,ENDADR ; Adresse des Herz-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-Byte's / -Word's / -DWord's von der Quelladresse in der Karte zur Endadresse im Speicher. Beim Lesen eines Spezial-Blockes wird die Quelladresse als Parameter verwendet.

**Übergabeparameter:** Kommunikationsdeskriptor, Befehlsblock

**Rückgabe:** [Buffer]

**Besonderes:** bei Fehler sprung auf Abort

**Beispiel:** Lese den Herz-Zählerwert des SIMOVERT-Regler 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 ; Buffer auf Reg R0
ASHD 1,ENDADR ; auf Byteadresse anpassen
MOVD 07FE00040,SRCADR ; Adresse des Herz-Zählers

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



## **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 Siemensregler ist die Ansteuerung von verschiedenen Motortypen ohne Änderungen an der Hardware vorzunehmen. 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 Inkrementgebers. 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 Blockfunktionen 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

15 11 10 4 3 0

0 0 1 1 1	Nr	
-----------	----	--

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

15 12 11 0

1000	Anzahl
------	--------

Quelladresse: Byteadresse des Buffers der zu schreibenden Worte

Endadresse: Bit 3 - 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 10h = Siemensspezifikation

31 28 27 26 16 15 8 7 0

Befehl	Tog	Para-Nr.	Index	1100
--------	-----	----------	-------	------

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       ; Buffer 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'
```

## Uebergabeparameterblock

	Wertebereich	
Siemens Befehl	(word)	1 - 15
AchsenNummer	(word)	0 - 127
Parameter Nummer	(word)	0 - 4095
Parameter Index	(word)	0 - 255
Buffer 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 genauere Parameterbedeutungen und Funktionsbeschreibungen sehen Sie 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.

**Übergabeparameter:** Kommunikationsdeskriptor, Achsennummer, Parameternummer, Bufferadresse

**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
Buffer      = R34                ; (dword) Bufferadresse 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           ; Buffer setzen auf R0

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.

**Übergabeparameter:**   Kommunikationsdeskriptor, Achsennummer, Parameternummer, Bufferadresse

**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	;	Buffer setzen auf R0
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\_RWPARAM F\_RDPARAM

Lesen des Parameterwertes mit Index  
(read PARAM VALUE ARRAY)

Lesen eines 'word':      RCXP    KomDes,AchsenNr,'F\_RWPARAM'  
Lesen eines 'dword':    RCXP    KomDes,AchsenNr,'F\_RDPARAM'

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

**Übergabeparameter:**    Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Bufferadresse

**Rückgabe:**            [Buffer] word oder dword

**Besonderes:**        Bei Fehler sprung auf Abort. Abbruchgrund im APO Register.

**Simovort-Funktion:**    6 Read PARAM VALUE ARRAY

**Beispiel:**            Lesen Motorpolpaarzahl, 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.Polpaarzahl
MOV     1,Index              ; 1. Wert Lesen
ADDR    R4,Buffer            ; Buffer setzen auf R4
```

```
GGD     @T_RWPARAM,R20      ; get descriptor of F_RWPARAM
RCXP    P_COMCH,AchsenNr,R20 ; read word parameter value array
ZTOP    DEV,POS,R4,040      ; Ausgeben der Polpaarzahl
```

**T\_RWPARAM:**        .TXT 'F\_RWPARAM'

## 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.

**Übergabeparameter:** Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Bufferadresse

**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 der Grundabtastzeit
ADDR   R0,Buffer      ; Buffer setzen auf R0
```

```
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, die richtige Funktion (mit und ohne Index) anhand der Parametercharakteristik aufrufen.

**Übergabeparameter:** Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Bufferadresse, 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       ; Buffer setzen auf R0

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 Anzahl der Indexelemente.

**Übergabeparameter:** Kommunikationsdeskriptor, Achsennummer, Parameternummer, Parameterindex, Bufferadresse, 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    ; Buffer setzen auf Pchar
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 Anwender gewählte Kommandos  
(read/write user command)

Command lesen:           RCXP 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.

**Übergabeparameter:** Kommunikationsdeskriptor, Befehl, Achsennummer, Parameternummer, Parameterindex, Bufferadresse, 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      ; Buffer setzen auf Pchar
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, Bufferadresse, 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 Störtexliste
MOV      35,Index            ; Index für 'Ext. Fehler1'
ADDR    R0,Buffer           ; Buffer setzen auf R0
MOV      16,Anzahl           ; Lesen 16 Bytes

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

```

## **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
                PHI =     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
                .WORD01234,05678,START
                .DOUBLE   012345678,087654321
                .FLOAT    1.2,PHI,3.4E5
                .LONG     6.7,PHI,-8.9E-10

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

TEXT:          .TXT       '^
<000009> Text mit CR/LF
<000009> und ohne CR/LF^
<000009> New-Line'

```



# INDEX

# INDEX

## Symbole

.HX .....	31
.INI .....	26
.LS .....	31
.SY .....	31
20mA .....	228
3964R .....	240
3964R-PROTOKOLL .....	239

## A

ABA .....	41
ABC .....	42
ABORT .....	151
Abort .....	41
Abort-Adresse .....	70, 71
ABR_ .....	196
ABS_ .....	126
ABSolute .....	126
ACMP .....	199
ACRTC-MODI .....	175
AD_39 .....	244
ADD_ .....	118
ADDition .....	118
ADDR .....	134
ADDRESS calculation .....	134
@ADR .....	48, 49, 50
Adresse .....	48, 134
Adresse mit Register-Offset .....	49
Adressierungsarten .....	45
Adresstabelle .....	78
AND_ .....	110
APO .....	42
APPEND file .....	216
Arbeitsregister .....	40
ARC .....	185
Arithmetic Shift .....	115
ARITHMETIK-Befehle .....	117
ASC .....	57
Ascii → BinaRy .....	196
Ascii CoMPare .....	199
ASCII-Befehle .....	195
ASCII-Kontroll Register .....	42
ASCII-Puffer .....	57
ASCII-String .....	199
ASCII-Zahl .....	196, 198
ASH_ .....	115
ASL .....	42
Attribut .....	205

## B

Balken TOP .....	160
Baud Rate .....	143, 146, 228
BCD-Zahl .....	132, 133
Befehlblock .....	250
Befehlsaufbau .....	44
Beispiel .....	15
Bildschirm .....	140
BIT-Befehle .....	85
Block Text OutPut .....	161
BRA .....	74
BRanch Always .....	74
BRanch to Sub-Routine .....	75
BSR .....	75
BTOP .....	160, 161
BYTE .....	46, 106

## C

CARC .....	186
Carriage Return .....	155
CBIT .....	93
CBR .....	136, 138
CBRS_ .....	137
CEARC .....	188
CENTRONICS .....	145
CFRCT .....	183
CHange DIRectory .....	221
Change PARA DESCR .....	268
Change PARA VALUE .....	264
Change PARA VALUE ARRAY .....	266
CHDIR .....	221
Clear BIT .....	93
CLear Device .....	153
Clear TIP .....	154
CLINE .....	180
CLRD .....	153
CODE-File .....	31
COM_ .....	113
Compare and BRanch absolute .....	136
Compare and BRanch floating .....	138
Compare and BRanch signed .....	137
COMplement .....	113
CONVERT-Befehle .....	129
COPY file .....	219
CPTN .....	190
CRAM .....	36
CRCL .....	184
CRLF .....	155

CTIP ..... 154  
 CXP ..... 83

**D**

Datenbaustein ..... 241  
 Datenwort ..... 241  
 Datum ..... 202  
 DCOPY ..... 220  
 DEBUG-Stecker ..... 37, 38  
 Decimal Hex ConVert ..... 133  
 Decrement ..... 55  
 DELAY ..... 72  
 DELETE file ..... 208  
 Device ..... 140  
 DEZ-Zahl ..... 196  
 DHCV\_ ..... 133  
 DIR ..... 224  
 DIRECTORY ..... 205  
 DIRectory ..... 224  
 DIS/EN-Schalter ..... 36  
 Disk COPY ..... 220  
 DISK space ..... 210  
 DIV\_ ..... 121  
 DIVision ..... 121  
 DOS-Disketten ..... 204  
 DOT ..... 177  
 DOUBLEPRECISION ..... 47  
 DOUBLE-WORD ..... 46  
 Drive Name ..... 206  
 DUMP ..... 107

**E**

ED\_39 ..... 245  
 Ellipse ..... 187  
 ELPS ..... 187  
 EnableTime ..... 26  
 EPROM-Stecker ..... 38  
 EQUAL ..... 16  
 EQUAL-File ..... 31  
 ERRORS ..... 147, 207, 229  
 eXChange ..... 103  
 eXclusive OR ..... 112  
 EXeQute ..... 66  
 EXQ ..... 66

**F**

F\_FRECOM ..... 254  
 F\_PUTBxx ..... 255  
 F\_RCPARA ..... 267  
 F\_RDPARA ..... 265  
 F\_RDPARV ..... 262  
 F\_RESCOM ..... 253

F\_RPARAD ..... 268  
 F\_RTEXT ..... 270  
 F\_RUCOM ..... 269  
 F\_RWPARA ..... 265  
 F\_RWPARV ..... 262  
 F\_WCPARA ..... 267  
 F\_WDPARA ..... 266  
 F\_WDPARV ..... 264  
 F\_WPARAD ..... 268  
 F\_WTEXT ..... 270  
 F\_WUCOM ..... 269  
 F\_WWPARA ..... 266  
 F\_WWPARV ..... 264  
 Farbe ..... 140  
 FB ..... 60  
 FCV ..... 140  
 FCV+FGV ..... 174  
 FFSB ..... 97  
 FGV ..... 174  
 FILE parameter ..... 211  
 Find First Set Bit ..... 97  
 FLAG-Base ..... 60  
 FLOATINGPOINT ..... 47  
 Floating to Integer ..... 130  
 Floatingpointunit ..... 29  
 FLOPPY-Befehle ..... 203  
 FLOPPY-FORMATE ..... 204  
 FORMAT disk ..... 226  
 FRCT ..... 182  
 Funktionstasten ..... 170  
 Fx\_CPU-25 ..... 38

**G**

GATR ..... 212  
 GCPU-15 ..... 36  
 GCPY ..... 192  
 GET ..... 233  
 Get ATtRIBUTE ..... 212  
 GET data ..... 233  
 Get Program Number ..... 67  
 GETD ..... 157  
 GGA ..... 62  
 GGD ..... 64  
 GGP ..... 63  
 Globale Adressen - Befehle ..... 61  
 GPNR ..... 67  
 GRAPHIK-Befehle ..... 173  
 Graphik-Pattern ..... 189  
 Graphik-Pixel ..... 177  
 Gross-Text ..... 159  
 GTOP ..... 159  
 Gx\_CPU-15 ..... 37  
 Gx\_CPU-25 ..... 37

**H**

Haltbit .....	72
Haltbits .....	41
Haltwort .....	41, 72
HDCV_ .....	132
Hex Decimal ConVert .....	132
heX-Ascii → BinaRy .....	198
HEX-Zahl .....	169, 198
Hex-Zahlen Text OutPut .....	169
Horizontal Text OutPut .....	163
HTOP .....	163
HTW .....	41

**I**

IB .....	58
IBT .....	94
Immediate .....	46
INCLUDE .....	272
Include-File .....	16
Increment .....	55
INDEL.INI .....	26
Indirekt (Adresse mit Register-Offset) .....	50
Indirekt (Pointer indexed) .....	52
INFO_SMD .....	258
INIt Device .....	151
INPUT-Base .....	58
Integer to Floating .....	131
Invert BIT .....	94
Invertiere .....	94, 112, 113
ISEC .....	62

**J**

JAT .....	78
JEX .....	81
JEX-Modul .....	81
JMP .....	76
JOAB .....	71
JOhann ABort .....	71
JOhann KIll .....	69
JOhann Self ABort .....	70
JOhann Self KIll .....	68
JOKI .....	69
JSAB .....	70
JSKI .....	68
JSM .....	77
JST .....	79
JTIP .....	171
JuMP .....	76
Jump EXternal .....	81
Jump indirect Address-Table .....	78
Jump Text InPut .....	171

Jump to Subroutine .....	77
Jump to Subroutine indirect Address-Table .....	79

**K**

KIll .....	68, 69
KONSTANTEN .....	272
Kontrolltasten .....	170
Konvertiere .....	130, 131
KOORDINATEN .....	176, 272
Kopiere .....	102, 107
Kreis .....	184

**L**

Lade .....	102
Language .....	202
LBR_ .....	99
LINE .....	178
Line Feed .....	155
Linie .....	163, 166, 178
Linker .....	16
Listing .....	272
LISTING-File .....	31
Load Bit Range .....	99
Logic Shift .....	114
LOGIK-Befehle .....	109
LOOP-Counter .....	136
Lösche .....	68, 69, 93, 107, 153
LSH_ .....	114

**M**

Macro Base-Page .....	78
MaKe DIRectory .....	222
Maskiere .....	110
MASTER .....	229
MASTER/SLAVE .....	227
MB .....	106
MBIT .....	95
MIKRO-Programm .....	81
MINB .....	96
MKDIR .....	222
MOD_ .....	123
MODulus .....	123
Monitortask .....	28
MOV .....	102
MOV_ .....	130, 131
MOVE .....	102
Move BIT .....	95
Move Byte .....	106
Move INvert Bit .....	96
Move signum eXtended .....	105
Move Zero extended .....	104
MOVE-Befehle .....	101

MPC .....	41
MSI.EXE .....	31
MTOP .....	162
MUL_ .....	120
Multi Text OutPut .....	162
MULTiplikation .....	120
MX_ .....	105
MZ_ .....	104

**N**

NEG_ .....	127
NEGate .....	127
Negiere .....	127
NotSystem .....	38
NS32016-Register .....	82
Null-Punkt .....	176

**O**

OB .....	59
OFF(REG) .....	54
OFF@{POI} .....	52
OFF[REG] .....	54
OFF{POI} .....	51
OR_ .....	111
ORG .....	176
OUT-BASE .....	82
OUTPUT-Base .....	59

**P**

PAINT .....	191
Parameterblock .....	261
PATH .....	225
Pattern .....	189
PFAD .....	206
PLINE .....	179
{POI} .....	51, 52
PolyLINE .....	179
Position .....	140
Programm-Counter .....	41
Programm-Start .....	272
PROTOKOLL .....	234
PROTOKOLL 3964R .....	239
PSEUDO-Befehle .....	271
PTN .....	189
PUT .....	232
PUT data .....	232
PUTD .....	156

**Q**

Quadrat-Wurzel .....	125
QUO_ .....	122
Quotient .....	122

**R**

R00..R5F .....	40
R00..R7F .....	53
R60..R6F .....	40
R70..R7F .....	40
Rahmen .....	163
RAM-AUFTEILUNG .....	35
RCT .....	181
RCXP .....	84
RDBLK .....	217
READ .....	214
Read Block .....	217
READ file .....	214
Read PARADESCR .....	268
Read PARA VALUE .....	262
Read PARA VALUE ARRAY .....	265
Rechteck .....	183
ReCTangle .....	181
(REG) .....	54, 55, 56
REG .....	53
[REG] .....	54, 55, 56
REG(REG) .....	56
REG@@ADR .....	50
REG@ADR .....	49
REG[REG] .....	56
REGISTER .....	39
Register .....	53
Register indexed (mit Offset) .....	54
Register indexed mit Auto-Inc/Dec .....	55
Register indexed mit Register offset .....	56
REM_ .....	124
REMAinder .....	124
ReMove DIrectory .....	223
RENAME .....	209
RENAME file .....	209
RESD .....	152
RESet Device .....	152
Rest .....	124
RETRY .....	90
Return To Mainprogram .....	80
REX .....	82
load Registers and jump EXternal .....	82
REX-MODUL .....	82
RMDIR .....	223
RNR .....	41
ROT_ .....	116
Rotiere .....	116
RS232 .....	228
RS422 .....	228
RTM .....	80

**S**

S-I/O 32 .....	143
----------------	-----

SATR ..... 213  
 SBIT ..... 92  
 SBR ..... 98  
 SchalterDIS/EN ..... 36  
 SchalterINT/EXT ..... 36  
 Schatten ..... 164  
 Schiebe ..... 114, 115  
 SEC ..... 41  
 Set ATIRibute ..... 213  
 Set BIT ..... 92  
 Set Bit Range ..... 98  
 SET Device ..... 150  
 SET Master ..... 231  
 SET Slave ..... 230  
 SET39M ..... 242  
 SET39S ..... 243  
 SETD ..... 150  
 SETM ..... 231  
 SETS ..... 230  
 Setze ..... 92  
 Setzte ..... 111  
 Shift ..... 114  
 Siemens ..... 258  
 SIMOVERT ..... 257  
 SINGLEPRECISION ..... 47  
 2 Kanal S-I/O ..... 146  
 SLAVE ..... 228  
 Sonderzeichen ..... 144, 147  
 SOURCE-File ..... 31  
 Spezial-Tasten ..... 170  
 Spezialblock ..... 251  
 SPLIT-SCREEN ..... 141  
 SPO ..... 41  
 Sprachen ..... 202  
 Sprachwahl ..... 202  
 SPRUNG-Befehle ..... 73  
 SQR\_ ..... 125  
 Square Root ..... 125  
 Stackpointer ..... 41  
 Starte ..... 66  
 STK ..... 41  
 SUB\_ ..... 119  
 Subdirectories ..... 205  
 SUBtraction ..... 119  
 SYMBOL-File ..... 31  
 Symbol-File ..... 16  
 System-Register ..... 40  
 Systemleistung ..... 72

**T**

Task Register ..... 40  
 Task-Kontroll Register ..... 41  
 TASK-KONTROLL-Befehle ..... 65  
 Task-Nummer ..... 67, 69, 71

Tausche ..... 103  
 TBR0 ..... 86  
 TBR1 ..... 87  
 Test and BRanch if bit = 0 ..... 86  
 Test and BRanch if bit = 1 ..... 87  
 Test and HaIt if bit = 0 ..... 88  
 Test and HaIt if bit = 0 and branch if Timeout  
 90  
 Test and HaIt if bit = 1 ..... 89, 90, 91  
 Test and HaIt if bit = 1 and branch if Timeout  
 91  
 TEXT ..... 272  
 TEXT IN/OUT ..... 139  
 Text InPut ..... 170  
 Text OutPut ..... 158  
 THT0 ..... 88  
 THT1 ..... 89, 90, 91  
 THTT1 ..... 91  
 THTTO ..... 90  
 TIM ..... 41  
 TIME ..... 202  
 TIME-Befehle ..... 201  
 Timeout ..... 90, 91  
 Timer ..... 41, 72  
 TIP ..... 170  
 TOP ..... 158  
 TRANS.EXE ..... 32  
 Treiber ..... 234

**U**

Uhr-Zeit ..... 202  
 UmTast ..... 170  
 Unterprogramm ..... 75

**V**

VERGLEICHS-Befehle ..... 135  
 Vertikal Text OutPut ..... 166  
 VIDEO ..... 140  
 Vorzeichen ..... 105, 137, 167, 196  
 VTOP ..... 166

**W**

Waitstates ..... 36, 37, 38  
 Wandle ..... 130, 131, 132, 133  
 Window ..... 142, 156, 157, 164, 174  
 WORD ..... 46  
 WRBLK ..... 218  
 WRDPR ..... 193  
 WRITE ..... 215  
 WRite BLock ..... 218  
 WRITE file ..... 215  
 Write-Protect ..... 38  
 WRPTN ..... 194

**X**

XABR_ .....	198
XCH_ .....	103
XOFF .....	147
XON .....	147
XOR_ .....	112
XZTOP .....	169

**Z**

Zahlen Text OutPut .....	167
ZTOP_ .....	167
Zuweisungen .....	16





# 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		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
	Hex	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	0	_			0	@	P	`	p	Ç	É	ā		Ł	ll	α	≡
1	1	_		!	1	A	Q	a	q	ü	æ	í	⊥	⊥	β	±	
2	2	_	↑	"	2	B	R	b	r	é	Æ	ó	⊥	⊥	Γ	≥	
3	3	_	△	#	3	C	S	c	s	ā	ō	ú		†	ll	π	≤
4	4	■	∂	\$	4	D	T	d	t	ä	ö	ñ	†	-	↳	Σ	∫
5	5	■	§	%	5	E	U	e	u	ä	ö	Ñ	†	†	F	o	J
6	6	■		&	6	F	V	f	v	ä	ü	a	ll	†	π	μ	÷
7	7	■		,	7	G	W	g	w	ç	ü	o	ll	ll	ll	Υ	≈
8	8	■	↑	(	8	H	X	h	x	ê	ý	¿	ll	ll	Φ	o	
9	9	■	↓	)	9	I	Y	i	y	ë	ö	¬	ll	ll	J	Θ	·
10	A	■	→	*	:	J	Z	j	z	è	Ü	¬	ll	ll	Γ	Ω	·
11	B	■	←	+	;	K	[	k	{	ï	ϕ	½	ll	ll	■	δ	√
12	C	■	⊥	,	<	L	\	l		î	£	½	ll	ll	■	∞	∞
13	D	■	↔	-	=	M	]	m	}	ï	₩	ı	ll	=	■	φ	²
14	E	■	€	.	>	N	^	n	~	Ä	ϣ	«	⊥	ll	■	ε	■
15	F	■	D	/	?	O	_	o	_	Å	f	»	⊥	ll	■	∩	E <sub>B</sub>

