INDEL Operating System

I S M - 6.0

Version: PC

Reference Manual

Contents

Introduction

General

History: The INDEL-Operating-System ISM was created in 1980 to program complex machines, equipments and process controls. By continually adapting actual requirements, the Multitasks-Operating-System is efficient and simple in its handling.
It provides the user with 32 tasks in application-oriented programming language. We implemented practicable commands to make it possible for programming-laymen, machine engineers and operating electricians to read and to edit the working cycles.

The system is excellently suitable to program operating sequences, much more than just working up fixed connections (SPS).

System: The system itself is completely written in assembler for a CPU of National's NS32000. The user normally isn't confronted withthis item, except he wishes to implement his own critical time functions or interrupts. Often, INDEL AG realises and implements such custom-built functions as for example controls, interpretations and so on. Those functions are than available as REX-call-instructions or as new instructions respectively.

ISM-Tasks: The 32 Tasks are worked out quasi-parallel. This means that there is worked up one command in every Task and after this there is a change to the next Task. In every walkthrough, the Assembler-Module "USER-CPY" is worked out once; there may be implemented equipment-specific functions, such as for example an electronic mainshaft.

Register: Each Task has 128 own 16-bit registers (R00..R7F) which also may be used as 32-bit (R01,R00) or 64-bit (R03,R02,R01,R00) registers. 16 registers (R70..R7F) of the 128 are reserved for system-functions and are continuously occupied.

Commands: The Tasks are programmed in an own assembler-like language. The commands have symmetric addressing modes; this means that there is, for example, only one "move command" that can move data from anywhere to anywhere.

Timer: For each Task, two 16-bit timers inside its Task-registers are available: one 10ms-timer and one second-timer.

Flags: All tasks have 256 common flags. These make the coordinating and controlling of operating sequences possible. By switching on, the flags 0...127 are always reset. The flags 128...255 are kept in mind, even in case of a voltage breakdown, provided there is mounted a CRAM with battery.

I/sec The system performance ISEC indicates the number of instructions each Task can work out per second (for example, I/sec is displayed in the INDEL-Debugger ID). This also allows to determine the maximum reaction rate.
The following is obvious: the less Tasks are started, the faster they are worked out. It therefore seems reasonable to work out sequences which exclude one another in the same Task.

Example

Default

Hardware: To test the following example, you need a PC-Master (it doesn't matter which version) with a EXT-IO card.

Inputs:	0	RESET	TASK-0
	1	START	TASK-0
	2	STOP	TASK-0
	4+5	running light-mode 0..3	TASK-2
Outputs:	0	ALARM	TASK-0
	1	READY	TASK-0
	2	RUN	TASK-0
	4..7	flashing light	TASK-1
	8..15	running light	TASK-2

TASK-0: Initialize everything, works out the RESET, START and STOP-keys, starts and kills the Tasks 1+2 and controls the ALARM, READY and RUN-lamps.

TASK-1: Blinks between the outputs 4 to 7.

TASK-2: Lets the running light, depending on the mode, run as follows:

00	left
01	right
10	adds 1
11	subtracts 1

EQUAL: As there doesn't exist a linker for the task-programming, all common assignments are preferably written in one file (EQUAL). The assembler "MSI/O" then generates, among the listing (EQUAL.LS), also a symbol-file (EQUAL.SY), which can be loaded too when the Tasks are assembled. (The EQUAL-file also could be integrated as include-file in each Task.)

Files: You also can find the following source-files in your PCMASTER-index under PCMASTER\BEISPIEL\ISM\.

EQUAL

```
.TITLE          EQUAL-File for DEMO-Tasks

;*****************************************************************************
;*                                                                           *
;*                      Common assignments                                   *
;*                      for the DEMO-Tasks                                   *
;*                                                                           *
;*===========================================================================*
;*      Assemble:   MSI/O EQUAL        ; generates EQUAL.LS and .SL           *
;*****************************************************************************
; Rev.  1.0  920515-FB  basic version                                INDEL AG

;************** Task Start-Addresses ****************************************
                .LOC    08000           ; Program-area start
TASK_0:         .BLKW   0200            ; TASK-0, 0200 WORD size
TASK_1:         .BLKW   0100            ; TASK-0, 0100 WORD size
TASK_2:         .BLKW   0100            ; TASK-0, 0100 WORD size

;************** Hardware Addresses ******************************************
HW_DPR:         .EQU    0160000         ; DUAL-PORT RAM
HW_ADC:         .EQU    0080    ; {DPR} ; ADC-card  0
HW_POS:         .EQU    0280    ; {DPR} ; POSI-card 0
HW_PCR:         .EQU    0400    ; {DPR} ; free transfer-RAM on the PC/AT

;************** Common Pointers *********************************************
DPR:            .EQU    1               ; Poi-1 points at DUAL-PORT RAM
PCR:            .EQU    2               ; Poi-2 points at PC-RAM
ADC:            .EQU    3               ; Poi-3 points at ADC-card 0
POS:            .EQU    4               ; Poi-4 points at POS-card 0

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

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

;************** Inputs *****************************************************
I_RESET:        .EQU    0               ; RESET-key
I_START:        .EQU    1               ; START-key
I_STOP:         .EQU    2               ; STOP -key

I_MODE:         .EQU    4       ;+5     ; running light mode 0..3
```

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

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

O_LAUF:         .EQU    8       ;..15   ; running light 0..7

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

;************** EQUAL THE END *************************************************
```

TASK0

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

;*****************************************************************************
;*                                                                           *
;*                           Demo-Task 0                                     *
;*                        Reset, Start, Stop                                 *
;*                                                                           *
;*---------------------------------------------------------------------------*
;*         Assemble:   MSI TASK0 EQUAL         ; Generates TASK0.LS and .HX  *
;*****************************************************************************
; Rev.  1.0  920515-FB  Basic Version                                INDEL AG

;-------------- Local assignments --------------------------------------------
        .LOC    TASK_0                          ; TASK Start Address
TNr_1:  .EQU    R10                             ; Task-Number of Task-1
TNr_2:  .EQU    R12                             ; Task-Number of Task-2

;************** Basic-initialization *****************************************
;-------------- load common pointers for all Tasks ---------------------------
INIT:   MOVD    HW_DPR,2*DPR{0}                 ; DUALPORT RAM
        ADDR    HW_ADC{DPR},2*ADC{0}            ; ADC-BASE
        ADDR    HW_POS{DPR},2*POS{0}            ; POS-BASE
        ADDR    HW_PCR{DPR},2*PCR{0}            ; PC/AT RAM BASE

;-------------- Delete the DUALPORT RAM –-------------------------------------
        MOV     0,0400{PCR}                     ; Delete first PC-RAM cell
        DUMP    0400{PCR},07FC,1{PCR}           ; Delete complete PC-RAM

;************** Wait for RESET ***********************************************
W_RESET:SBIT    O_ALARM,OB                      ; ALARM-lamp on
        MOV     S_ALARM,STAT{PCR}               ; STATUS = ALARM to PC/AT
        THT0    I_RESET,IB                      ; Wait until RESET-key is pressed
        CBIT    O_ALARM,OB                      ; ALARM-lamp off
        THT1    I_RESET,IB                      ; Wait until RESET-key is released

;-------------- Start Task 1 and 2 -------------------------------------------
EXQ_1:  EXQ     TASK_1,TNr_1,EXQ_1              ; Start TASK 1
EXQ_2:  EXQ     TASK_2,TNr_2,EXQ_2              ; Start TASK 2

;************** Wait for START ***********************************************
READY:  SBIT    O_READY,OB                      ; READY-lamp on
        MOV     S_READY,STAT{PCR}               ; STATUS = READY to PC/AT
W_START:TBR1    I_RESET,IB,T_RESET              ; RESET-key operated ?
        TBR0    I_START,IB,W_START              ; START-key operated ?
```

```
;-------------- START- key operated ---------------------------------------------
T_START:CBIT    O_READY,OB                      ; READY-lamp off
        SBIT    O_RUN,OB                        ; START-lamp on
        THT1    I_START,IB                      ; Wait until START-key is released

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

;************** Flash with RUN-lamp until STOP ***********************************
BLINK:  IBIT    O_RUN,OB                        ; Flash with RUN-lamp
        MOV     50,TIM                          ; Load TIM with 500ms

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

;-------------- STOP-key operated ------------------------------------------------
T_STOP: CBIT    O_RUN,OB                        ; RUN-lamp off
        CBIT    F_RUN,FB                        ; RUN-FLAG off
        BRA     READY                           ; Are we ready again

;************** RESET- key operated **********************************************
T_RESET:CBIT    F_RUN,FB                        ; RUN-FLAG off
        CBIT    O_RUN,OB                        ; RUN-lamp off
        CBIT    O_READY,OB                      ; READY-lamp off
        SBIT    O_ALARM,OB                      ; ALARM-lamp on
        MOV     S_ALARM,STAT{PCR}               ; STATUS = ALARM to PC/AT
        JOAB    TNr_1                           ; Task-1 abort
        JOAB    TNr_2                           ; Task-2 abort

        THT1    I_RESET,IB                      ; RESET- key still operated?
        DELAY   100                             ; Wait 1 second!
        CBIT    O_ALARM,OB                      ; ALARM-lamp on
        BRA     EXQ_1                           ; Restart tasks

;************** TASK-0 THE END ***************************************************
```

TASK1

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

;*******************************************************************************
;*                                                                             *
;*                      Demo-Task 1                                            *
;*                  flash with Out BLK0..3                                     *
;*                                                                             *
;*******************************************************************************
;*       Assemble:   MSI TASK1 EQUAL           ; Generates TASK1.LS and .HX    *
;*******************************************************************************
; Rev.  1.0  920515-FB  Basic Version                                   INDEL AG

;------------- Local assignments -----------------------------------------------
        .LOC    TASK_1                          ; TASK Start Address

;************** Basic-initialization *******************************************
INIT:   MOV     ABORT,ABA                       ; Jump on ABORT if JOAB
        SBIT    O_BLK0,OB                       ; BLK-lamp 0 on
        SBIT    O_BLK2,OB                       ; BLK-lamp 2 on

;************** RUN / STOP *****************************************************
W_RUN:   THT0    F_RUN,FB                       ; Wait until RUN
        IBIT    O_BLK0,OB                       ; Invert all BLK-outputs
        IBIT    O_BLK1,OB
        IBIT    O_BLK2,OB
        IBIT    O_BLK3,OB
        DELAY   20                              ; Wait 20ms
        BRA     W_RUN

;************** Kill Task ******************************************************
ABORT:  SBR     O_BLK0,OB,4,0                   ; Flashing light off
        JSKI                                    ; KILL this Task

;************** TASK-1 THE END *************************************************
```

TASK2

```
.TITLE          **- Demo Task 2 -**
.SUBTITLE       running light
;****************************************************************************
;*                                                                          *
;*                          Demo-Task 2                                     *
;*              running light with Out LAUF 0..7                            *
;*                                                                          *
;*--------------------------------------------------------------------------*
;*      Assemble:   MSI TASK2 EQUAL             ; Generates TASK2.LS and .HX *
;****************************************************************************
; Rev.  1.0  920515-FB  Basic Version                                INDEL AG

;—————-------——— Local assignments ———————————————--------------------—————————-
        .LOC    TASK_2                          ; TASK Start address
MODE:   .EQU    R10                             ; MODE 0..3
LICHT:  .EQU    R12                             ; Running light register

;************** Basic-initialization ****************************************
INIT:   MOV     ABORT,ABA                       ; Jump on ABORT if JOAB
        MOV     0101,LICHT                      ; 2*8-Bit to 16-Bit register
        SBR     O_LAUF,OB,8,LICHT               ; Set 8 outputs from O_LAUF

;************** RUN / STOP **************************************************
W_RUN:  THT0    F_RUN,FB                        ; Wait until RUN
;—————— Mode 0..3 evaluate —————--------------------------————————————————-—
RUN:    LBR     I_MODE,IB,2,MODE                ; Read 2 INP from I_MODE
        JSM     MODE@EXQ_TAB                    ; execute MODE 0..3
        SBR     O_LAUF,OB,8,LICHT               ; Set 8 outputs from O_LAUF
        DELAY   3                               ; Wait 30ms!
        BRA     W_RUN                           ; Still RUN ?

EXQ_TAB:.WORD   LINKS                           ;0; move 1 left
        .WORD   RECHTS                          ;1; move 1 right
        .WORD   PLUS                            ;2; + 1
        .WORD   MINUS                           ;3; - 1

;============== Running light functions ====================================
LINKS:  ROT     1,LICHT                         ; Move 1 left
        RTM     0
RECHTS: ROT     -1,LICHT                        ; Move 1 right
        RTM     0
PLUS:   ADD     0101,LICHT                      ; + 1
        RTM     0
MINUS:  SUB     0101,LICHT                      ; - 1
        RTM     0

;************** Kill Task ***************************************************
ABORT:  SBR     O_LAUF,OB,8,0                   ; Running light off
        JSKI                                    ; KILL this task
;************** TASK-2 THE END **********************************************
```

Installation

MSI: To assemble the files, enter the following:

```
MSI /O    EQUAL              ; generates EQUAL.LS, .HX and .SY
MSI /O    TASK0    EQUAL     ; generates TASK0.LS, .HX and .SY
MSI /O    TASK1    EQUAL     ; generates TASK1.LS, .HX and .SY
MSI /O    TASK2    EQUAL     ; generates TASK2.LS, .HX and .SY
```

CONFIG: The PC-Master needs a configuration-file, for it is possible to adjust itself to the connected periphery-cards (see PC-Master documentation). Such a file is generated when you enter CONFIG, the number of the IO-cards, set it on 2, for example, and save under DEMO.PCM .

INDEL.INI: To test the Task, the project-file INDEL.INI must first be generated or adjusted (see also chap. TOOLS). It is needed by TRANS, SHOW and ID and looks approximately like this:

```
[Target]
System=PCMaster

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

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

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

TRANS: Now, the operating-system and the Tasks with TRANS can be loaded in the PC-Master. As in the file INDEL.INI AutoStart=yes is written, the task-0 (mostly called Monitortask) is automatically started. The other Tasks will not be started, until the RESET-key (INP-0) is pressed (see listing TASK0).

SHOW, ID: With the SHOW-Program, you can test and supervise the function of the IO-card. The INDEL-Debugger ID serves the installation and debugging of the Tasks on Source-Level.

TOOLS

INDEL.INI

FILE.INI — All tools of the INDEL AG get the configuration-data from a central '.INI' - file, the name of which can be transferred by calling the program as a parameter.

For example TRANS MyIni.ini

INDEL.INI — If there is no name specified for a certain parameter, all tools seek the configuration file INDEL.INI in the actual index.

The form of such a file is similar to the '.INI'-file structure of Windows. A title (application-name) is followed by the so-called keywords (key-names) which describe the single configuration-points:

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

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

It follows a description of the single items :

[Target]

System= — Defines the target system that is to be operated.

PCMASTER	-	the target system is a PC-Master
IPS-32	-	the target system is an Indel 19"-Rack
Default :		PCMASTER

[PCMaster]

Address= — Specification of the address on which the PC-Master is located (rotary switch values), for example CA00.
Default : D000

ConfigFile= — Name and path of Dualport RAM-configuration-file, creatid by CONFIG.EXE, for example c:\Project\test.pcm.
Default : CONFIG.PCM

WarmBoot=

NO	-	the target system is, in every case, first initialized and then stimulated with software
YES	-	the target system is only initialized and stimulated with software, if it isn't already busy or has broken down.
Default :		NO

EnableTime=	NO	- Probably used time-commands supply a wrong result
	YES	- In the PCMaster, PC-time and -date are available by the standard-time-commands.
	note :	This option always refers to all in the PC installed PCMasters. The TSR-driver gets the according addresses from SET PCMASTER = ... entry in Autoexec.bat.

FloatingPointValues=

	NO	The values in the DPR are represented in the usual fixed-point format.
	YES	The values are represented in the floating-point format. (This option is only available in connection with an INFO-Master.)

[IPS-32]

Baudrate= Baud-rate for the data transfer PC → IPS-32 rack

2400	2400 Baud	(Modem)
9600	9600 Baud	(Modem)
19200	19200 Baud	HST-Modem
38400	38400 Baud	Direct connection PC/AT → IPS-32 rack

DataBits= Number of Data-Bits per 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= Number of retries in case of transmission errors until there is an error message on screen.

5	5 retries

Timeout= Latency in ms until retry. Usually, this entry is not needed because the optimally time-out time is calculated according to the current baud-rate.

SlaveNumber= Slave-number of IPS-32 Rack

1	Slave-number 1

Port=

PC/AT	interface number
COM1	first interface
COM2	second interface

[Trans]

SystemSoftware=	Name and path of the system-software, for example c:\pcmaster\pcm.hex	
	Default :	PCM.HEX
SystemOffset=	A download-offset can be specified (only with Target=IPS-32). The offset is entered as word address in hex.	
	Default :	0
SystemDownload=	NO	- the system-software is not loaded in the target system.
	YES	- the system-software is loaded in the target system.
	Default :	YES
SystemVerify=	NO	- there is no comparison between source and target code.
	YES	- Source and target code are compared with each other and possible errors are indicated.
	Default :	NO
SystemAutostart=	NO	- the operating-system is started and immediately set on HALT. (For confirmed Indel Freaks this agrees with 'Init-Halt' with the Utility).
	YES	- the operating-system is started normally
	Default :	YES
DownLoad=	NO	- those, in the [ProjectFiles] possibly indicated files, are not automatically loaded in the target system.
	YES	- those, in the [ProjectFiles] possibly indicated files, are automatically loaded in the target system.
	Default :	NO
Verify=	NO	- there is no comparison between source and target code.
	YES	- Source and target code are compared with each other and possible errors are indicated.
	Default :	NO
Autostart=	NO	- the monitor task is started and immediately set on HALT.
	YES	- the monitor task is started
	Default :	NO
FloatingPointUnit=	NO	- the target system has no floating point unit.
	YES	- the target system has a floating point unit.
	Default :	NO

[Show]

ScreenMode=	2	- 25 lines, black and white mode on colour adapter
	3	- 25 lines, colour mode
	7	- 25 lines, monochrome mode
	258	- 43/50 lines, black and white mode on colour adapter
	259	- 43/50 lines, colour mode
	263	- 43/50 lines, monochrome mode

[Debug]

ScreenMode= see [Show]

RefreshRate= display-refresh-rate in sec
Default : 1

TabSize= tabulator characters (09) are extended in files to TABSIZE space characters.
Default : 8

maxInputs= The maximum number of inputs that an 'Inputs'-window manages can be entered. The number is rounded by the debugger to a multiple of 16 and can't go beyond 4096.
Default : 256

maxOutputs= As 'maxInputs' but for the outputs.

maxFlags= As 'maxInputs' but for the flags.

AutoTaskWndClose=

	YES	- the window of a task that doesn't existed any longer, is automatically deleted
	NO	- not automatically deleted
	Default :	YES

WatchCaseSensitiv=

	YES	- at the watches, capital or small letters are registered
	NO	- no observation of capital or small letters
	Default :	NO

SourceFileTrace=	YES	- the actual listing is always indicated (for example at a single step in another file)
	NO	- a listing change must be done by hand (with 'View' → 'Task source')

Verify=	YES	- By loading down of project files, the source code is compared with the target code. Possible differences are indicated.
	NO	- By loading down, there is no comparison between source and target code.
MemoryFilename=		- Off ID Rev. 1.32, it' s possible to write a MemoryDump directly into a file. Open MemoryDump (→ ALT-F10 → W). The Default-filename can be specified here.
NumberOfValues=		- Off ID Rev. 1.32, it' s possible to write a MemoryDump directly into a file. Open MemoryDump (→ ALT-F10 → W). The Default-number can be specified here.

+

[ProjectFiles]

FILE1= Here, those project files are entered that are loaded from TRANS.EXE in the target system or which should be known by the Debugger, respectively.

After '=', a Downloadoffset (in hex) can be entered because with the ISM-Compiler MSI.EXE, only Compilates in the address range 0..FFFF can be generated.

0	- The file is loaded in the range 00'0000 ... 00'FFFF.
10000	- The file is loaded in the range 01'0000 ... 01'FFFF.
Default :	0

MSI

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

MSI.EXE — The assembler for the ISM-5.0 operating system knows the following switches:

/O — Generates a symbol file NAME.SY
All assignments of the first file can therefore be used with the assembling of the further files.
This file is needed by INDEL-DEBUGGER "ID" to set watches.

/S — Generates a sorted list of all symbols in a listing-file NAME.LS.

/F — Off the Rev. ISM-5.0, the bit-commands can be executed faster with the addressing modes
Immediate,IB , Immediate,OB , Immediate,FB ,
if '/F' is specified with the assembling. The commands are then assigned to the new 17 command group; there are only the above-mentioned addressing modes possible, but they can be executed very fast.

/L — The debug flag /L shows the listing of all passes on the screen and serves only the debug finding in case of inexplicable pass errors.

/I — Passes and Include-files are indicated when assembling.

FILES:

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

Example: The machine has a common EQUAL-File, a common text file DTEXT and three tasks 0..2:

```
MSI /O    EQUAL
```

Generates EQUAL.LS and EQUAL.SY. The file EQUAL.HX isn't used if it doesn't contain tables that generate codes.

```
MSI /O    DTEXT    EQUAL
```

Takes over the assignments of symbol file EQUAL.SY, assembles the text in DTEXT and generates simultaneously with the .LS and .HX file also the new symbol file DTEXT.SY; this latter contains all assignments of EQUAL and the start addresses of texts. The three task-files can now use all assignments of EQUAL and all texts of DTEXT.

```
MSI       TASK0    DTEXT
MSI       TASK1    DTEXT
MSI       TASK2    DTEXT
```

TRANS

TRANS [IniFile.INI]

TRANS.EXE — This program allows the loading of operating software and the ISM-5.0 tasks in the target system PC-Master or IPS-32 rack.

INDEL.INI — The trans-program needs an .INI file which contains all specifications concerning the target-system and the project-files. If there is no special IniFile.INI defined, TRANS is automatically looking for INDEL.INI in the local directory.

Keynames: — TRANS is looking for the following keynames in INDEL.INI:

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

You will find an exact description of the entries under INDEL.INI at the beginning of this chapter.

FILES:

ConfigFile.PCM — If you are working with the PC-Master, TRANS needs the Dualport-Ram-configuration-file ConfigFile.PCM, generated by CONFIG. TRANS finds this file thanks to an entry in [PCMaster].

System.HEX — TRANS finds the operating-system System.HEX, that must be loaded, thanks to an entry in [TRANS].

Tasks.HX — If they are available, the ISM-5.0 task-programs tasks.HX, that must be loaded, are entered under [ProjectFiles].

ID

ID [IniFile.INI]

ID.EXE The INDEL-DEBUGGER ID allows the debugging of the ISM-5.0 Tasks in the target systems PC-Master or IPS-32 rack. You can work with several Tasks at the same time, without mutual influence.

INDEL.INI The ID-program needs an .INI file, which contains all specifications concerning the target-system and the project-files. If there is no special IniFile.INI defined, ID is automatically looking for INDEL.INI in the local directory.

Keynames: ID is looking for the following key-names in INDEL.INI:

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

You will find an exact description of the entries under INDEL.INI at the beginning of this chapter.

FILES:

Tasks.HX At the start, ID is looking for all task-files tasks.HX that are registered under [ProjectFiles] and creates a .MAP file, in which all start- and end-addresses of those programs are entered. This is how the ID can, at every time, assign and display the corresponding listing for every Task.

Tasks.LS The tasks.LS files are needed for the source-level debugging.

Tasks.SY If there is a watch-window opened, the debugger needs the corresponding symbol file.

CONFIG

CONFIG [ConfigFile.PCM]

CONFIG.EXE — With the CONFIG-Program, the Dualport-RAM configuration file is generated. Now the PC-Master (PC/AT) or Master-32 (IPS-32) knows all connected interface-cards and their operating modes.

PC-Master

ConfigFile.PCM — TRANS writes this file at the start in the PC-Master Dualport-RAM.

IPS-32

MASx.INC — The operating system for the IPS-32 rack needs, to manage each MASTER-32 card, also a Dualport-RAM configuration file with the names MAS1.INC to MAS3.INC. These files, in the .BYTE-Format, are appended at the end of file IOMAS32.32K.

CONVERT.EXE — The program CONVERT converts a ConfigFile.PCM file in a .BYT file ConfigFile.INC:

CONVERT ConfigFile

RAM-ORGANIZATION

PC-MASTER RAM

WORD-ADR	
00'0000.. 00'7FFF	System-Program
00'8000.. 01'BFFF	User-CRAM if it has 1MB Ram-ICs
00'8000.. 07'BFFF	User-CRAM if it has 4MB Ram-ICs
07'C000.. 07'FFFF	System-Ram and Task-Register
16'0000.. 16'03FF	Dualport Ram to PC/AT FirmWare
16'0400.. 16'07FE	Dualport Ram to PC/AT User range

INFO-MASTER RAM

WORD-ADR	
00'0000.. 00'7FFF	System-Program
00'8000.. 01'BFFF	User-CRAM if it has 1MB Ram-ICs
00'8000.. 07'BFFF	User-CRAM if it has 4MB Ram-ICs
07'C000.. 07'FFFF	System-Ram and Task-Register
16'0000.. 16'03FF	Dualport Ram to PC/AT FirmWare
16'0400.. 16'07FE	Dualport Ram to PC/AT User range

INFO-Interface

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

REGISTER

Task Register

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

R70..R7F: The registers R70..R7F are SYSTEM-REGISTERS and assigned fixly. They can be addressed as each other register (for example R7E) or with their names (for example MPC).

R60..R6F: The registers R60..R6F are occupied as ASCII-Buffer (Standard-occupation by SETD) in case of video- and ASCII-commands. If there are no such operations carried out, these registers can be occupied normally.

R00..R5F: The registers R00..R5F are the task-working-registers.

Task-Control Register

RNR,MPC: Both registers RNR and MPC build together the 32-bit Macro-Program-Counter.

HTW: The Haltw ord HTW (STOP) contains 8 conditional and 7 unconditional HALT-Bits.

BD	B0..B7	Conditional HALT, only if B15=0 is
UBD	B8..B11	Unconditional HALT
D	B12	Occupied by DEBUG
S	B13	Occupied by S-I/O
T	B14	Occupied by Timer (DELAY)
B	B15	Halt-inhibit-Bit for B0..B7

TIM: The system decrements the timer-register each 10 msec by 1, until it is 0000. It can be assigned w ith any commands, but it is also used by DELAY-command.

SEC: The system decrements the SEC-register each second by 1, until it is 0000.

ABA: An address is saved in the register ABA on w hich Johann jumps in case of an abort. If the address is 0000, the task is, in case of an abort, killed, and all used devices (VIDEO,SIO) are given free. If the address is not 0000, Johann jumps on (ABA). The rack-number RNR cannot be left in this case! Also SPO and HTW are reset, the devices remain reserved.

ABORT: * The address of the aborted command can be transferred to the stack w ith RTM 255 to be able to jump back on the command (Retry) w ith RTMO.

SPO: The stack-pointer SPO (Low er-Byte) indicates the stack-depth. The stack is empty if it' s 00. With 0FF (-1), the first place is occupied and so on. It is automatically served from JSR, BSR and interests the user only in special cases.

Kill Stack: MOV 0,SPO

With each ABORT, a copy is made from the low er- to the higher-Byte and the low er Byte is reset (= 00). The stack is now basicly deleted, but it can again be reconstructed w ith MHLB SPO,SPO (Abort in a subroutine, in that you w ish to return).

STK: The registers R76..R70 build the actual STACK. R76 is the first, R75 the second stack place, and so on.

CAUTION: The stack depth is not limited !

* Off System Rev. 5.11

ASCII-Control Register

ABC: This is the higher Byte in R7A and must be loaded with a special MOVE (for example MLHB "A",ABC).
If this character is entered by TIP or TOP on the keyboard, an abort is produced (the task jumps on the ABORT-address).
ABC is set to 01B (ESC) with INID or SETD.

APO: This is the lower Byte in R7A and must be loaded with a special MOVE (for example MLLB 0,APO).
This register is activated automatically by TIP, RTIP, ABR and ACMP and interests the user only in special cases.
APO is set to 00 with INID or SETD.
In case of abort (for example SETD, Floppy-commands an so on), an error number is set in APO, which shows the exact abort-reason.

ASL: This is the higher Byte in R79 and must be loaded with a special MOVE (for example MLHB 01F,ASL).
ASL is the maximum number of characters, that are read in with TIP. You thus can limit the size of the input-window in a screen-mask or make the ASCII-buffer longer or shorter (01F character == 010 REG). ASL is also active when, with TIP, the text is not saved in the ASCII-buffer, but in any RAM-buffer.
ASL is set to 01F with INID or SETD.

ASR: This is the lower Byte in R79 and must be loaded with a special MOVE (for example MLLB 060,ASR).
In ASR is the register located in which the ASC-addressed ASCII-buffer begins.
You can place the ASCII-buffer in any register-range.
ASR is set to 060 with INID or SETD.

ASC: ASC is an own addressing mode! (Does not generate a register number.)
With ASC, the ASCII-Buffer is addressed. The beginning of the ASCII- buffer is written down in ASR, the length in ASL.

ADDRESSING MODES

Format Of Commands

<table>
<tr><th>15 … 8</th><th colspan="2">7 … 0</th><th></th></tr>
<tr><td>Command-Code</td><td>SSSS</td><td>DDDD</td><td>command-head</td></tr>
<tr><td>Control-Bits</td><td>SSSS</td><td>DDDD</td><td>With more than 2 arguments</td></tr>
<tr><td colspan="3">SAD Jump Address</td><td>With conditional jumps</td></tr>
<tr><td colspan="3">DATA SRC</td><td></td></tr>
<tr><td colspan="3">DATA DEST</td><td></td></tr>
<tr><td>DATA SRC</td><td colspan="2">DATA DEST</td><td>B :B,Rxx,(Rxx),[Rxx]</td></tr>
</table>

BBBB = command-code
SSSS = addressing mode SRC
DDDD = addressing mode DEST
SAD = jump address (LABEL or address)
The control bits are used with text-commands (for example CR/LF) or as command-code extension (for example Floppy-commands).

Examples:

```
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
```

Addressing Modes

SRC	DEST	DATA		ADDRESSING MODE
0	0	WWWW WWWW	WWWW WWWW	WORD :W
1	1	LLLL LLLL	LLLL LLLL	D_WORD :D:F
		HHHH HHHH	HHHH HHHH	INT / FLOAT
2	2		QQQQ QQQQ	BYTE :B
3	3		0RRR RRRR	REG R00..R7F
3	3		1xxx xxxx	NOT USED
4	4		0RRR RRRR	(REG)
4	4		1RRR RRRR	[REG]
5	5	0000 0000	: 0RRR RRRR	OFF(REG)
5	5	0000 0000	: 1RRR RRRR	OFF[REG]
6	6	0000 0000	: 0BBB BBBB	OREG(BREG)
6	6	0000 0000	: 1BBB BBBB	OREG[BREG]
6	6	1NNN NNNN	: 0RRR RRRR	(REG)N
6	6	1NNN NNNN	: 1RRR RRRR	[REG]N
7	7	0000 0000	: 0RRR RRRR	REG@ADRE
7	7	0000 0000	: 1RRR RRRR	REG@@ADRE
8	8	AAAA AAAA		@ADRE
9	9	0000 0000		OFF{POI}
9	9	1000 0000		OFF@{POI}
A	A	none		ASC
B	B	MANTISSA		DOUBLE
		MANTISSA		-PRECISION
		MANTISSA		-FLOATING
		S: EXPONT	:MANT	-POINT
C		QQQQ QQQQ		BYTE
D		0RRR RRRR		REG
D		1		NOT USED
E		0RRR RRRR		(REG)
E		1RRR RRRR		[REG]
	C	none		IB
	D	none		OB
	E	none		FB
F	F			NOT USED

xxx

Immediate

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

Explanation: The command can be specified as a simple number. As far as the format isn't forced with :B, :W or :D, the MSI-assembler distributes the values in the command as follows:

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

The system expands BYTE and WORD-specifications in the command always with operational sign to DOUBLE and only than executes the operation!

Caution: The assembler changes in case of values > 07F automatically from BYTE to WORD, but not in case of values > 08000 from WORD to DOUBLE ! (Mostly, BYTE and WORD operations are executed!) In case of DOUBLE-instructions, this can lead to errors:

Examples:

MOV	0A0,R10	; R10	=	00A0	correct!
MOV	0A000,R10	; R10	=	A000	correct!
MOVD	0A000,R10	; R11,10	= FFFF'	A000	wrong??
MZWD	0A000,R10	; R11,10	= 0000'	A000	correct!
MOVD	0A000:D,R10	; R11,10	= 0000'	A000	correct!
MXWD	0A000,R10	; R11,10	= FFFF'	A000	demanded!
MXWD	0A000:D,R10	; R11,10	= FFFF'	A000	demanded!

```
        MOV     1E4,R10         ; R10       =       2710 = 10000
```

xxx.xx

FLOATING POINT Immediate

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

Explanation: If there is a number written with decimal point, the assembler automatically sets a floating point number (provided that approved within the command!).

SINGE-PREC: -3.4028235E-38 ... 3.4028235E38

DOUBLE-PREC: -2.225073858507201E-308 ... 2.225073858507201E308

Note: The command itself decides, whether SINGLE or DOUBLE PRECISION numbers must be put in. The specifications :F and :L have no influence and can be left out!

Caution: The MSI-assembler for PC/AT can only process exponents up to E38!

Examples:

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

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

@ADR

	Address
8	@ADR
Explanation:	Show s an address in a locale (64K) RACK-range. ADR = 0000 ... 0FFFF
Note:	This addressing mode is better used w ithin a listing only (@LABEL). Addresses outside the local 64K-range are addressed w ith the addressing modes REGISTER-INDEXED and POINTER-INDEXED!

Example:

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

REG@ADR

Address with Register-offset

7 REG@ADR

Explanation: On ADR, within a table, the value which is written in REG is displayed.

Note: The table must be in the direct nearness of the command!
‼ ADRE must be in the range of ±127. of MPC ‼

Example: R11 = 0003

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

REG@@ADR

Indirect (Address with Register-Offset)

7 REG@@ADR
a

Explanation: On ADR, within a table, the address which is written in REG is displayed. This address is addressed by the command.

Note: The table must be in the direct nearness of the command!
‼ ADRE must be in the range of ±127. of MPC ‼

OFFSET-REG: The offset-Register always contains a 16-Bit offset with operational sign. (-32768 ... 0 ... +32767).

Example: R11 = 0002

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

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

ADRE:   .WORD   01234
```

OFF{POI}

Pointer indexed

9 OFF{POI}

Explanation: All tasks have 12 common pointers (Pointer 0..11) and each task has 4 own, local pointers (pointer 12..15). Such a pointer always contains a 32-Bit (base-) address. Relative to this pointers, a data element can now be addressed with fixed offsets.

Note: The offset is always positive and must be in the range of 000 ... 07FF.

Load Pointer: For the pointers can load themselves, the pointer-0 always indicates the common pointer-table after start up; after the start of a single task (EXQ..), the pointer -12 indicates itself. This makes it possible to load the other pointers first and, on request, also pointer-0 respectively pointer-12.

Example: Load pointer 4 with the base 1' A000 and than write on the 16th place of this data-range the value 01234:
(The address of the pointer-4 = 8{0} , because of Double-Word entries!)

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

OFF@{POI}

Indirect (Pointer indexed)

9 OFF@{POI}

Explanation: All tasks have 12 common pointers (Pointer 0..11) and each task has 4 own, local pointers (pointers 12..15). Such a pointer always contains a 32-Bit (Base-)address. Relative to this pointers, an address can now be indicated with fixed offsets; this allows a data element to be addressed.

The WORD-address on OFF@{POI} refers to the rack in which the address-table is located!

Note: The offset is always positive and must be in the range of 000 ... 07FF.

Load pointer: See OFF{POI}

This addressing serves, for example, the indirect text output by text-table. The text can thereby be located in any (64k)RACK-range. By reloading the text-pointer, the whole machine can also be converted in another national language.

Example:

```
TEXT    =                   5               ; POINTER-occupation
ADDR    @TTAB,2*TEXT{0}                     ; Choosing the TEXT-table


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


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

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

REG

Register

3/D	REG
Explanation:	Every task has 128 registers (R00..R7F) that are addressed hereby. The registers R70..R7F can also be addressed with their names (see also SYSTEM-REGISTER).
Note:	With DOUBLE-WORD access, always two registers in series are addressed! With LONG- FLOATING access, always four registers in series are addressed!

Example:

```
        MOV   ABORT,ABA       ; LOAD ABA WITH THE ADR ABORT
        MOV   1300,TIM        ; LOAD TIMER WITH 1.3 SEC
ABORT:  MOVD  012345678,R10   ; R11 = 01234 , R10 = 05678
```

OFF[REG]

Register indexed (with Offset)

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

Explanation: The register (Rxx) contains an address that is addressed (with offset).

(REG) With paranthesis (Rxx), the register contains a 16-Bit address in the same (64k)RACK-range as the command.

[REG] With brackets [Rxx], the register contains a 32-Bit address.

OFFSET: In front of the paranthesis and brackets, an offset of maximal -128 ... +127 can be indicated to this address.

Example:

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

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

        ADDR    ASC,R10         ; R11,R10   = ADR OF ASCII-buffer
        MLLB    3[R10],R00      ; R00       = 7' TH CHARA IN ASC
        ...

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

[REG]N

Register indexed w ith Auto-Increment/Decrement

6	(REG)N
6	[REG]N

Explanation: N is automatically added to the address that the register contains; in case of decrement, it is added before, in case of increment after the operation.
!! POST-INCREMENT / PRE-DECREMENT !!

(REG) With paranthesis (Rxx), the register contains a 16-Bit address in the same (64k)RACK-range as the command.

[REG] With brackets [Rxx], the register contains a 32-Bit address.

N: N must be w ithin the range of -64...+63.

Example:

```
        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 with Register Offset

6 REG(REG)

6 REG[REG]

Explanation: The target address is formed by adding the base address in (Rxx) and the offset in Ryy.

(REG) With paranthesis (Rxx), the register contains a 16-Bit address in the same (64k)RACK-range as the command.

[REG] With brackets [Rxx], the register contains a 32-Bit address.

OFFSET-REG: The offset-register always contains a 16-Bit offset with operational sign (-32768....0....+32767).

Example:

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

MOV     R00(R10),R66        ; R66 = 02222

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


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

ASC

ASCII-Buffer

A	ASC
Explanation:	ASC indicates the ASCII-buffer, defined in the registers ASR (ASCII-register number) and ASL (ASCII-buffer size).
Note:	After INID or SETD, the registers R60..R6F form the ASCII-buffer! This addressing mode does not generate SRC/DEST-data in the command!

Example:

```
MLHB    10,ASL            ; MAX 10 CHARACTERS INPUT
TIP     DEV,POS,ASC       ; TEXT-INPUT IN THE ASCII-BUFFER
TIME    ATIM,ASC          ; TIME IN ASCII
TOP     DEV,POS,ASC       ; DISPLAY OF THE ASCII-BUFFER
```

IB

INPUT-Base

C	IB
Explanation:	IB indicates the first input-card.
Note:	May be specified only as second parameter (DEST) ! This addressing mode does not generate SRC/DEST-data in the command!

Example:

```
THT0    15,IB            ; WAIT UNTIL I-15 = 1
TBR1    128,IB,ERROR     ; ERROR IF I-128 = 1
```

OB

OUTPUT-Base

D	OB
Explanation:	OB indicates the first output-card (or OUT-COPY).
Note:	May be specified only as second parameter (DEST) ! This addressing mode does not generate SRC/DEST-data in the command!

Example:

```
SBIT    010,OB              ; SET THE OUTPUT 16

MOTOR   = 35                ; OUTPUT MOTOR ON
TBR0    MOTOR,OB,MOT_AUS    ; TEST IF MOTOR = OFF
```

FB

FLAG-Base

E	FB
Explanation:	FB indicates the first FLAG-Word.
Note:	May be specified only as second parameter (DEST) !
	This addressing mode does not generate SRC/DEST-data in the command!

Example:

```
THT0    13,FB    ; WAIT UNTIL F-13 = 1
CBIT    14,FB    ; SET F-14 = 0
```

Global Address - Commands

GGA

Get Global Address

B7_ 00_ GGA SRC, DEST:D

Explanation: Search the label with the name in SRC in the global variable table and write the word address (of the label) to DEST.

If there exist labels in different modules with the same name, the module name can be specified as additional search key. For a label can be recorded in the global variable table, it must be exported.

ERRORS: The task jumps by the following errors on its ABORT-address:
(The error-number stands in 'APO')

041 The label wasn’t found
042 The label has an uneven byte-address

Example 1: Write the address of the ISEC-counter to R20/21.

```
        GGA     @TX.ISEC, R20
TX_ISEC: .TXT 'V_SYISEC'
```

Example 2: Write the address of the system-busy-table to R0/R1.

```
        GGA     @TX_BUSY, R0
TX_BUSY: .TXT 'SYSTEM.V_BUSY'
```

GGP

Get Global Pointer

B7_ 02_ GGP SRC, DEST:D

Explanation: Search the label with the name in SRC in the global variable table, interpret the double word of the label' s address as byte-pointer, change this latter in a word-pointer and write the result to DEST.

If there exist labels in different modules with the same name, the module name can be specified as additional search key. For a label can be recorded in the global variable table, it must be exported.

ERRORS: The task jumps on its ABORT-address in case of the following errors:
(The error-number stands in 'APO')

041 The label wasn't found
042 The byte-pointer is uneven

Example: Write the pointer on the central 1ms timer to R0/R1.

```
          GGP     @TX_1MS, R0

TX_1MS:   .TXT 'P_TIM1MS'
```

P_TIM1MS is defined in the module INIT, for example as follows:

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

➔ R0/R1 = 01603EA

GGD

Get Global Descriptor

B7_ 01_ GGD SRC, DEST:D

Explanation: Search the label with the name in SRC in the global variable table and write the pointer on its descriptor to DEST.

If there exist labels in different modules with the same name, the module name can be specified as additional search key. For a label can be recorded in the global variable table, it must be exported.

ERRORS: The task jumps on its ABORT-address in case of the following errors:
(The error-number stands in 'APO')

041 The label wasn't found

Example : Use the library-function " F_EXQTSK" to start a Johann on address 045A000.

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

```
TX_EXQ:     .TXT. 'F_EXQTSK'
```

TASK-CONTROL-Commands

EXQ

EXeQute

0Cxx SAD EXQ SRC,DEST,SAD

Explanation: Start the program at SRC on the first free task and write the number of this task to DEST. All registers in the new task are deleted!

If there is no task free, jump to SAD.

Example 1: Start the first free task with the start-address ADRE. Calculate the new task-number to REG 00:

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

Example 2: Start a task on the double-word-address 045' A000:

```
MOVD    045A000,R10         ; R10 = TASK START-ADDRESS
BSR     EXQD                ; START THE TASK
...
```

Subroutine for the command EXQD:

```
        R10     = ADR:D
        R00..03 Used
EXQD:   EXQ     HALT,R00,ERROR    ; R00 = TASK-PROG NUMBER
        GPNR    R01               ; R01 = OWN PROG-NUMBER
        SUB     R01,R00           ; R00 = PNR-DIFFERENCE
        MUL     080,R00           ; R00 = REGISTER-SPACE
        ADDR    MPC,R02           ; R02 = ADR OF OWN MPC
        MOVD    R10,R00[R02]      ; START THE TASK ON 45' A000
        RTM     0
HALT:   BRA     HALT              ; TASK STOPS
```

GPNR

Get Program NumbeR

0Bx0	GPNR DEST
Explanation:	Write the own Task-Number to DEST.
Example:	Calculate the own task-number to REG 00:
	GPNR R00

JSKI

Johann Self KIll

00x0	JSKI
Explanation:	Delete the ow n task and set all reserved devices free.
Example:	Delete the ow n task:
	JSKI

JOKI

JOhann KIll

0Fx0 JOKI SRC

Explanation: Delete the task with the Task-Number in SRC and set all its reserved devices free.

Example: Delete task number. 5:

JOKI 5

JSAB

Johann Self ABort

0Dxx	JSAB

Explanation: Set the own task on its Abort-Address ABA. Save the current stack pointer (into R77-HIGH-Byte) and set it (R77- LOW-Byte) to 00.

If ABA = 0000, delete the task and set all its reserved devices free.

Example: Jump on ABA (Kill Stack):

JSAB

JOAB

JOhann ABort

0Exx	JOAB SRC
Explanation:	Set the task with the Task-Number in SRC on its Abort-Address ABA. Save its current stack pointer (into R77-HIGH-Byte) and set it (R77-LOW-Byte) to 00. If ABA = 0000 delete the task and set all its reserved devices free.
Example:	Abort the task with the number in R00: JOAB R00

DELAY

DELAY

A5x0 DELAY SRC

Explanation: Set the 10ms Timer ‘TIM’ with the value in SRC and set the Delay-Halt-Bit T in the Halt-Word HTW. The timer-interrupt deletes this Halt-Bit if TIM = 0000.

Note: Because the task is on HALT during the delay, the system is relieved by one task in this time. The system performance can thus be considerably increased by brilliant application of this command!

Critical commands are for example: GTOP, TIP, HTOP, TIME

Example 1: Set the output 15 for 1 second to one:

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

Example 2: Time-large-screen-display (only RACK-Version):

```
LOOP:   TIME  ATIM,ASC      ; Get time (ASCII)
GTOP    DEV,POS,ASC         ; Large-screen-display
DELAY   100                 ; System-relief 1-SEC
BRA     LOOP
```

Jump-Commands

BRA

BRanch Always

F___ SAD   BRA   SAD

Explanation: Jump to the address SAD. Only the displacement SAD-momentary address is filed in the command.
With SAD only one LABEL can be specified!
Displacement   max.   ± 07FF   (1-WORD command)

Example: Jump to LABEL:

LABEL: BRA   LABEL

BSR

Branch to Sub-Routine

E___ SAD BSR SAD

Explanation: Save the actual MPC in the stack and jump on the address SAD. Only the displacement SAD-momentary address is filed in the command.
With SAD only one LABEL can be specified!
Displacement max. ± 07FF (1-WORD Command)

Example: Call up an under-program named SUBROUT:

```
BSR     SUBROUT
```

JMP_

JuMP

01x0	JMP	SRC
02x0	JMPD	SRC:D

Explanation: Jump on the address SRC.
Here, each addressing mode can be used with SRC!

Example 1: Jump on the address 0A000 in the actual (64k)RACK-range:

```
JMP     0A000          ; RNR unchanged !
```

Example 2: Jump on address 04000 in rack-3:

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

JSM

Jump to Subroutine

03x0 JSM SRC

Explanation: Save the actual MPC in the stack and jump on the address SRC in the actual (64k)RACK-range.
Here, each addressing mode can be used w ith SRC!

Example 1: Jump in the subroutine on ADRE:

```
JSM     ADRE
```

Example 2: Jump on the address that stands in 011(R22):

```
JSM     011(R22)
```

JAT

Jump indirect Address-Table

06__ JAT AT

Explanation: Jump on the address that stands under AT in the address table.
AT max. 0...0FF (1-WORD command)
The address of ATAB is set in the INIT with the pointer (HWMCB) on the Macro Base-Page.

Example: MPC = (33(ATAB))

JAT 033

JST

Jump to Subroutine indirect address-Table

07__ JST AT

Explanation: Save the actual MPC in the stack and jump on the address that stands under AT in the address table.
AT max. 0...0FF (1-WORD Command)
The address of ATAB is set in the INIT with the pointer (HWMCB) on the Macro Base-Page.

Example: MPC = (33(ATAB))

JST 033

RTM

Return To Main program

04__ RTM N

Explanation: Return to the main program at the end of a subroutine. Thereby, N words of the main program are skipped.
N max. ± 07F

Example: Return to the main program and skip the next five words:

```
RTM     5
```

JEX

Jump EXternal

0800 MSAD   JEX   MSAD

Explanation: Jump in a MICRO-program with the address MSAD.

If MSAD < 02000,	than	CXP	MSAD(JEX-MODULES)
If MSAD >= 02000,	than	JSR	MSAD

CPU-Register: The NS32016-registers are loaded as follows:
R7 = Address of REG 00 of the calling up task
R6 = Address of JEX COMMAND (byte address)
R5 = Address of NEXT COMMAND (word address)
All CPU-registers may be changed!

The JEX-module is determined by the INIT (for example HWJMD = MOD-5).

Example 1: Call MICRO-ROUTINE on address 0100:B of the REX-Module:

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

Example 2: Call LOCAL-MICRO-ROUTINE that stands on address MICRO:

```
        JEX     MICRO           ; PC = 2*MICRO      (BYTE-ADR)
        ...
MICRO:  .BYTE   012,00          ; RET0              NS32000-Micro
```

NSB.EXE: The program NSB compiles a NS32000 assembler-program (NAME.LST) in a .BYTE-File (NAME.BYT) that can be integrated with .INCLUDE.

REX

load Registers and jump EXternal

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

Explanation: Jump in a MICRO-Program with the address MSAD and transfer the parameter to SRC and DEST.

If MSAD < 02000,	than	CXP	MSAD(REX-MODULES)
If MSAD >= 02000,	than	JSR	MSAD

CPU-Register: The NS32016-registers are loaded as follows:
R7 = Address of REG 00 of the calling up task
R6 = Address of REX COMMAND (byte address)
R5 = Address of NEXT COMMAND (word address)
R4 = Address of SRC
R3 = Address of DEST
R2 = Address of DEST
R1 = Contents of SRC:D
R0 = Contents of DEST:D
All CPU-registers may be changed!

The REX module is determined by the INIT (for example HWJMD = MOD-5).

Example 1: Call REX-MODULE PC(MOD)=0A and transfer the contents of REG00 and the constant 045 to the micro program:

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

Example 2: Calculate the WORD-address of the OUT-BASE into REG 01,00 (since FB,IB,OB are not allowed as SRC, the ADDR-command is not possible!):

```
REX     R00,OB,D_ADR                ; is also possible for FB,IB...
```

Micro-program: Address from DEST to SRC !

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

NSB.EXE: The program NSB compiles a NS32000 assembler-program (NAME.LST) in a .BYTE-File (NAME.BYT) that can be integrated with .INCLUDE.

CXP

Call eXternal Procedure

B7_03_ CXP DESC:D

Explanation: Jump in the micro-procedure with the descriptor DESC.
DESC must first be loaded with GGD.

CPU-Register: The NS32016-registers are loaded as follows:
R7 = Address of REG 00 of the calling up task
R6 = Address of JEX COMMAND (byte address)
R5 = Address of NEXT COMMAND (word address)
All CPU-registers may be changed!

Example : Call MICRO-ROUTINE "MEIN_PROC" that is defined in any module.

```
GGD     @TX_MEIN, R10
CXP     R10

TX_MEIN:.TXT 'MEIN_PROC'
```

RCXP

load Registers and Call eXternal Procedure

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

Explanation: Jump in the MICRO-Procedure with the descriptor DESC and transfer the parameter to SRC and DEST.
DESC must first be loaded with GGD.

CPU-Register: The NS32016-registers are loaded as follows:
R7 = Address of REG 00 of the calling up task
R6 = Address of REX COMMAND (byte address)
R5 = Address of NEXT COMMAND (word address)
R4 = Address of SRC
R3 = Address of DEST
R2 = Address of DEST
R1 = Contents of SRC:D
R0 = Contents of DEST:D
All CPU-registers may be changed!

Example : Use the library-function " F_EXQTSK" to start a Johann at the address 045A000.

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

TX_EXQ: .TXT. 'F_EXQTSK'
```

BIT-Commands

TBR0

Test and BRanch if bit = 0

10xx SAD TBR0 OFF,BASE,SAD

Explanation: Test the Bit (Offset,Base) and jump on SAD if the Bit = 0.

TBSR0: * If the command jumps on SAD, the return address can, with RTM 255, be got in the stack and then, with RTM 0 (beyond the TBR-command), be jumped back (Equivalent a TBSR-command).

Example 1: Jump on LABEL if the input 35 = 0:

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

Example 2: Jump on LABEL if the flag with the number in R00 is not set:

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

* Off System Rev. 5.11

TBR1

Test and BRanch if bit = 1

11xx SAD TBR1 OFF, BASE, SAD

Explanation: Test the Bit (Offset,Base) and jump on SAD if the Bit = 1.

TBSR1: * If the command jumps on SAD, the return address can, with RTM 255, be got in the stack and then, with RTM 0 (beyond the TBR-command), be jumped back (Equivalent a TBSR-command).

Example 1: Jump on LABEL if the input 15 = 1:

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

Example 2: Jump on LABEL if there is a negative number in R10 (Bit 15 = signum of the number = 1 if negative):

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

* Off System Rev. 5.11

THT0

Test and HalT if bit = 0

15xx	THT0	OFF,BASE

Explanation: Halt if the Bit (Offset,Base) = 0.

Example 1: Wait until the FLAG 5 = 1:

```
THT0    5,FB
```

Example 2: Wait until the input 35 is set:

```
THT0    35,IB
```

THHT1

Test and HalT if bit = 1

16xx THT1 OFF,BASE

Explanation: Halt if the Bit (Offset,Base) = 1.

Example 1: Wait, until the FLAG w ith the number in R00 is deleted:

```
THT1    R00,FB
```

Example 2: Wait until the input 5 is not set any more:

```
THT1    5,IB
```

THTT0

Test and HalT if bit = 0 and branch if Timeout

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

Explanation: Halt (if the Bit (Offset,Base) = 0) as long as either the Bit (Offset, Base)= 1 or the time (TIME) has run down. If the time (TIME) has run down, jump on SAD and write ERRORNR to R70.

RETRY: * If the command jumps on SAD, the address can be got, with RTM 255, from THTT-command itself on the stack and (for example after an error message) be jumped back on the command with RTM 0 (Retry).

Example: Wait 1 sec. max until the input 5 = 1. In case of Timeout, jump to LABEL and write 7 in R70.

THTT0 5, IB, 1000, 7, LABEL

* Off System Rev. 5.11

THTT1

Test and HalT if bit = 1 and branch if Timeout

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

Explanation: Halt (if the Bit (Offset,Base) = 1) as long as either the Bit (Offset, Base)= 0 or the time (TIME) has run down. If the time (TIME) has run down, jump on SAD and write ERRORNR to R70.

RETRY: * If the command jumps on SAD, the address can be got, with RTM 255, from THTT-command itself on the stack and (for example after an error message) be jumped back on the command with RTM 0 (Retry).

Example: Wait 1 sec. max until the input with the number in R10 = 0. In case of Timeout, jump to LABEL and write 8 in R70.

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

* Off System Rev. 5.11

SBIT

Set BIT

12xx SBIT OFF,BASE

Explanation: Set the Bit (Offset,Base) = 1.

Remark: This READ-MODIFY-WRITE command is executed in the interlocked-mode and therefore can, even in multiprocessor-operation, not be interrupted by another CPU. That is why it is also used for the communication of several CPUs on the BUS by FLAGs.
(Only SBIT- and CBIT-commands!)

Example 1: Set the output 45 on 1:

```
SBIT    45,OB
```

Example 2: Set the Bit with the number in R00 in the register R10:

```
SBIT    R00,R10
```

Example 3: Set the flag 128:

```
SBIT    128,FB
```

CBIT

Clear BIT

13xx	CBIT	OFF,BASE

Explanation: Clear the Bit (Offset,Base) = 0.

Remark: This READ-MODIFY-WRITE command is executed in the interlocked-mode and therefore can, even in multiprocessor-operation, not be interrupted by another CPU. That is why it is also used for the communication of several CPUs on the BUS by FLAGs.
(Only SBIT- and CBIT-commands!)

Example 1: Clear flag 5:

```
CBIT    5,FB
```

Example 2: Clear Bit 15 in REG 33:

```
CBIT    15,R33
```

IBIT

Invert BIT

14xx IBIT OFF,BASE

Explanation: Invert the Bit (Offset,Base); 1 → 0 ; 0 → 1 .

Example: Flash with the output 155 in intervals of 1 second:

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

MBIT

Move BIT

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

Explanation: Copy the Bit (OFF,BASE) to Bit (OFF2,BASE2).

Example: Copy the input-Bit 045 to the output 5:

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

MINB

Move INvert Bit

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

Explanation: Copy the inverted Bit from (OFF1,BASE1) to Bit (OFF2,BASE2).

Example: Copy the inverted Bit 1 of R22 to FLAG 5:

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

FFSB

Find First Set Bit

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

Explanation: Test N Bits off Bit (OFF,BASE) on ‘ 1’ and give the number of the first set Bit to DEST. If none of those Bits is set, set DEST = 0FFFF.

ATTENTION: Even though N can be specified with 1..32, only up to 25 Bits are processed, according to the Start-Bit. The CPU first gets the Bits to be processed by means of a double-word transfer out of the memory into the internal register. This means that the Bit-range can move within 4 Bytes only. Therefore, N is limited as follows:

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

Example 1: Search for the Bit-number of the first set Bit in REG 00 and give it to REG 22:

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

Example 2: Search for the first set FLAG in the range FL-45..54 and write the Bit-number in R10: If, for example, FL-50 is the first set flag, R10 gets 5!

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

SBR_

Set Bit Range

1Dxx 00xx	SBR	OFF,BASE,N,SRC
	SBRD	OFF,BASE,N,SRC:D

Explanation: Copy N Bits of SRC to Bit (OFF,BASE) and following.
SBR N = 1..16
SBRD N = 1..32

ATTENTION: Even though N can be specified with 1..32, only up to 25 Bits are processed, according to the Start-Bit. The CPU first gets the Bits to be processed by means of a double-word transfer out of the memory into the internal register. This means that the Bit-range can move within 4 Bytes only. Therefore, N is limited as follows:

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

Note: With this command, the correct order is DEST,N,SRC !

Example: Copy 24 Bits of REG 01,00 to the outputs off Output-Bit 045:

SBRD 045,OB,24,R00

LBR_

Load Bit Range

1Fxx 00xx	LBR	OFF,BASE,N,DEST
	LBRD	OFF,BASE,N,DEST:D

Explanation: Copy N Bits off Bit (OFF,BASE) flush-right to DEST and fill in the residual Bits in DEST with '0'.

LBR N = 1..16
LBRD N = 1..32

ATTENTION: Even though N can be specified with 1..32, only up to 25 Bits are processed, according to the Start-Bit. The CPU first gets the Bits to be processed by means of a double-word transfer out of the memory into the internal register. This means that the Bit-range can move within 4 Bytes only. Therefore, N is limited as follows:

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

Example: Load 30 inputs off input-Bit 045 to REG 01,00:

```
LBRD    045,IB,24,R00
```

MOVE-Commands

MOV_

MOVe

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

Explanation: Copy the contents of SRC to DEST.

ATTENTION: MOVF and MOVL go to TRAP-3 if the numbers are not floating-point numbers!

Example: Load R00 with the contents of the address 'ADRE':

```
MOV     @ADRE,R00
```

XCH_

eXCHange

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

Explanation: Exchange the contents of SRC and DEST.

Example: Exchange the contents of R00 and 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

Explanation: Copy the contents of SRC to DEST and fill in the residual Bits (in DEST) with 0.

Example 1: Copy the first sign of the ASCII-buffer to R10 and delete the higher Byte in R10 as well as the whole R11:

```
MZBD    ASC,R10
```

Example 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

Explanation: Copy the contents of SRC to DEST and fill in the residual Bits (in DEST) with the operational sign of SRC.
SRC = positive : fill in 0
SRC = negative : fill in 1

Examples: 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

Explanation: Copy one Byte from SRC to DEST. The other Byte in DEST is left unchanged if DEST is in CRAM-range.
L = Lower Byte H = Higher Byte

Example 1: Limit the ASCII-Buffer to 10 signs:

```
MLHB    10,ASL
```

Example 2: Overwrite the second character in the ASCII-buffer with 'A':

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

DUMP

Dump

0Axx	DUMP	SRC,N,DEST

Explanation: Copy N 16-Bit words from SRC to DEST (copies ascending!).

Example 1: Copy 01000..013FF to 02000..023FF:

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

Example 2: Clear memory 0A000..0BFFF:

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

LOGIC-Commands

AND_

AND

28xx	AND	SRC,DEST
28xx	ANDD	SRC:D,DEST:D

Explanation: Delete all Bits in DEST that are deleted in SRC.

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

Example: Mask R00 with 0FF00:

```
AND     0FF00,R00
```

OR_

OR

29xx	OR	SRC,DEST
39xx	ORD	SRC:D,DEST:D

Explanation: Set all Bits in DEST that are set in SRC.

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

Example: Set all Bits in R00 that are set in ADRE:

```
OR      @ADRE,R00
```

XOR_

eXclusive OR

2Axx	XOR	SRC,DEST
3Axx	XORD	SRC:D,DEST:D

Explanation: Invert all Bits in DEST that are set in SRC.

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

Example: Invert BIT 4 and 2 in R33:

```
XOR     014,R33
```

COM_

COMplement

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

Explanation: Copy the inverted SRC to DEST.

Example: Invert all Bits in R22:

```
COM     R22,R22
```

LSH_

Logic SHift

2Dxx	LSH	N,DEST
3Dxx	LSHD	N,DEST:D

Explanation: Shift DEST N times left (N=pos) or right (N=neg) and fill in the new Bits with '0'.

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

Example: Shift R00 5 times left:

```
                                ; R00 = 0001
LSH         5,R00               ; Shift left
                                ; R00 = 0020
```

ASH_

Arithmetic SHift

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

Explanation: Shift DEST N times left (N=pos) and fill in the new Bits with 0 or shift DEST N times right (N=neg) and extend with the operational sign of DEST.

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

Example: Divide R00 by four. Operational sign remains:

```
                        ; R00   = 0FF00   (-256)
ASH     -2,R00          ; Shift right, remain operational sign
                        ; R00   = 0FFC0   (-64)
```

ROT_

ROTate

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

Explanation: Rotate DEST N times left (N=pos) or right (N=neg).

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

Example: Rotate R22 5 times left:

```
                          ; R22 =    1000
ROT      5,R22            ; Rotate left
                          ; R22 =    0002
```

ARITHMETIC-Commands

ADD_

ADDition

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

Explanation: Add SRC and DEST to DEST.

Example 1: Add 1 to R00:

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

Example 2: Add Pi to 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

Explanation: Subtract SRC from DEST to DEST.

Example: Subtract 1 from R23,22

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

MUL_

MULtiplication

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

Explanation: Multiply SRC with DEST to DEST.

Example: Multiply R23,22 with 3.3:

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

DIV_

DIVision

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

Explanation: Divide DEST by SRC to DEST.

+ 10 / + 3 = + 3
- 10 / + 3 = - 4 * * Rounds differently than QUO!
+ 10 / - 3 = - 4 *
- 10 / - 3 = + 3

Example: Divide R25,24,23,22 with 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

Explanation: Calculate the quotient of DEST/SRC to DEST.

```
+  10 QUO   +  3    = + 3
-  10 QUO   +  3    = -  3 *
+  10 QUO   -  3    = -  3 *
-  10 QUO   -  3    = + 3
```

* Rounds differently than DIV !

Example: Calculate the quotient of R22 / 033:

```
QUO     033,R22
```

MOD_

MODulus

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

Explanation: Calculate the rest of DEST/SRC to DEST.

```
+  10 MOD   +  3   = + 1
-  10 MOD   +  3   = + 2 *
+  10 MOD   -  3   = -  2 *
-  10 MOD   -  3   = -  1
```

* Rounds differently than REM!

Example: Calculate R22 MOD R00 to R22:

```
MOD     R00,R22
```

REM_

REMainder

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

Explanation: Calculate the rest of DEST/SRC to DEST.

```
+   10 REM  +  3 =          +         1
- 10 REM   +3  = - 1 *      * Rounds differently than MOD !
+   10 REM  -  3 =          +         1 *
- 10 REM   - 3  = - 1
```

Example: Calculate the rest of the DIV R22 / 3 to R22:

```
REM     3,R22
```

SQR_

SQuare Root

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

Explanation: Calculate the square root of SRC to DEST.

Example: Calculate the square root of 2 to 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

Explanation: Calculate the absolute value of SRC to DEST.
neg → pos ; pos remains positive !

Example: Calculate the absolute value of R00 to 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

Explanation: Calculate the negative value of SRC to DEST.
neg → pos ; pos → neg

Example: Negate the value in R25,24,23,22:

```
NEGL    R22,R22
```

CONVERT-Commands

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

Explanation: Change the floating point number in SRC to an integer number in DEST.

Example: Convert the floating point number R25,24,23,22 in an integer number 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

Explanation: Change the integer number in SRC to a floating point number in DEST.

Example: Convert the integer number 123 in a floating point number to 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

Explanation: Change the HEX-number in SRC to a decimal number (BCD-number) in DEST.

Example: Change the HEX-value in R22 to a decimal value:

```
HDCV    R22,R22
```

DHCV_

Decimal Hex ConVert

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

Explanation: Change the decimal number (BCD-number) in SRC to a HEX-number in DEST.

Example: Change the decimal-value in R22 to a HEX-value:

```
DHCV    R22,R22
```

ADDR

ADDRess calculation

5Dxx ADDR SRC,DEST:D

Explanation: Calculate the address of SRC to DEST (Double-Word Address).

Example1: Calculate the address of REG 00 to REG 00/R01:

```
ADDR    R00,R00
```

Example2: Calculate the address of the ASCII-BUFFER to R01,00:

```
ADDR    ASC,R00
```

Compare- Commands

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

Explanation: Compare SRC with DEST and jump to SAD if the condition is fulfilled. The operational sign is not tested (08000>07FFF!)

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

Example 1: Jump to SAD if R10,11 = R22,23:

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

Example 2: Pass through a LOOP 125 times:

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

Example 3: Search the end of the text in the ASCII-Buffer:

```
        ADDR    ASC,R0                  ; Address of the ASCII-BUFFER
LOOP:   CBR     000FF,&Z,[R0],EOTL      ; Test Lower-Byte
        CBR     0FF00,&N,[R0]+1,LOOP    ; Test Higher-Byte, Address+1
EOTH:                                   ; End of the text in the High-Byte -1[R4]
EOTL:                                   ; End of the text in the 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

Explanation: Compare SRC with DEST and jump to SAD if the condition is fulfilled.
The operational sign is tested (08000<07FFF!)

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

Note: The comparisons = and <> may also be entered by CBRS and CBRSD. However, they are changed automatically in a normal CBR or CBRD.

Example 1: Jump to SAD if R22 is positive:

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

Example 2: Jump to SAD if R22,23 is negative:

```
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
```

Explanation: Compare SRC with DEST and jump to SAD if the condition is fulfilled.

BEF	COND	Function
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

Example 1: Jump to SAD if R22,23 >= 123.456 E15:

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

Example 2: Jump to SAD if R10..13 < PHI:

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

TIME-Commands

TIME

get/set TIME

B4x_ TIME ART,ADRE

Condition:
- PCMASTER Firmware Rev.1.56 or higher
- TRANS.EXE 1.7 or higher
- INI-file-entry in rubric [PCMaster] enable time = YES

Explanation: Set or read time, date, day of week or day number.

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

Language: The day of week can be shown in several languages:
L: 0 = German , 1 = English , 2 = Italian , 3 = French

If the system is designed multilingual, LANGUAGE can serve as (battery-stored) language-selection-Bit for the whole system!

Day of week: D: 1 = Monday , 2 = Tuesday ... 7 = Sunday

Note: The length of the ASCII-buffer 'ASL' is not tested! The turn of the year and the leap-year is, also after the year 2000, automatically and correctly processed.

Example: Always shows the actual time. Since the time is shown in second-intervals only, the system can be relieved essentially if a DELAY-command is inserted:

```
LOOP:   TIME    ATOT,ASC        ; DATE , TIME , DOW , DNR
        TOP     DEV,POS,ASC     ; DISPLAY ON SCREEN
        DELAY   100             ; SYSTEM-RELIEF 1-SEC
        BRA     LOOP
```

PC-Interface-Commands

PCCOM

General: These commands serve the access on the PC's serial interfaces, starting from the PCMaster's macro. It was paid attention, that the selection of a serial PC-Interface (COMx) can happen quite compatible to the selection of a INDEL 2K-SIO.

Conditions: To select a PC-Interface from PCMaster,
two drivers of the PC must first be loaded in the correct order.

1. COMDRIVE.COM Rev. 2.03 or higher

   Driver of the serial PC-interfaces.

2. PCMIRQ.EXE Rev. 1.00 or higher

   Driver of the PC-Master interrupts.

The tasks which use the PCCOM-commands, must announce these with '.INLCUDE PCCOM.INC' to the macro-assembler at the beginning. The module PCCOM.OBJ must be linked to the very number in the operating system that is defined in the PCCOM.INC or vice versa.

3 more parameters can be handed over to the driver PCMIRQ by means of INDEL.INI:

[PCMaster]	
IRQNumber=	Number of the PCMaster-Interrupts (adjustable by jumper) Default : 11
[PCMIRQHandler]	
COMInputBufferSize=	Size of the input-buffer in Bytes Default : 256
COMOutputBufferSize=	Size of the output-buffer in Bytes Default : 256

Attention: The overflow of the input- or output-buffer is not supervised; this means that it is up to you to choose the adequate sizes according to the requests.

Device Number: The counting of the device numbers is started with 0, what means that the PC-interface COM1 has the PCCOM-device number 0, COM2 the number 1 and so on. (But at the moment, only two interfaces are supported.)

Baud Rate: The baud rate is specified as follows:

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

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

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

Errors : Transmission errors of any case (for example parity error and so on) are not yet identified; this means, that the corresponding task is, in case of such an error, not yet set on abort automatically.

Control-Lines: The control-lines DTR,RTS,CTS,DSR can, by means of special commands, be changed or questioned, respectively.

Information:

- With all PCCOM-commands, the system requires the macro-registers R70 and R71; this means, that the depth of stack decrements by two places.
- Because the processing of the PCCOM-commands happens completely asynchronous to the system, it is not allowed to use immediate values for certain parameters!!!

```
Example: COMBTOP    1,    01B,    1
                    |     |       I
                    |     |       Number of sign: immediate allowed
                    |     Text: immediate NOT allowed
                    Device: immediate allowed
```

COMSETD

COM SET Device

COMSETD BAUD/DEV

BAUD/DEV: immediate allowed

Explanation : Initialize the PC-SIO number. DEV with the baud-rate BAUD. From now on, only the initialized task has access to this SIO.

Information : The counting of the device-numbers begins with 0; this means, that the first PC-SIO in the system has the device-number 0, the second the number 1 and so on.

Example : Initialize the PC-SIO COM2 with the baud-rate 9600,n,8,1.

COMSETD 01501

COMRESD

COM RESet Device

COMRESD DEV

DEV: immediate allowed

Explanation : Set the PC-SIO Nr. DEV free again .

Information : The counting of the device-numbers begins with 0; this means, that the first PC-SIO in the system has the device-number 0, the second the number 1 and so on.

Example : Set the PC-SIO COM1 free of Johann's miseries.

COMRESD 0

COMTOP

COM Text OutPut

COMTOP DEV,TADR

DEV: immediate allowed
TADR: immediate not allowed

Explanation : Output of the text-string TADR to DEV.

Example: Write the text ' By Zeus, where are the women ?' to the PC-SIO COM2

COMTOP 1,@ZEUS

ZEUS: .TXT ' By Zeus, where are the women?'

COMBTOP

COM Block Text OutPut

COMBTOP DEV,TBLK,N

DEV: immediate allowed
TBLK: immediate not allowed
N: immediate allowed

Explanation : Output of N signs of the text block TBLK to DEV.

Example : Send this 5 byte control sequence to the ink jet printer :

```
COMBTOP 1,@TBLK,5

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

COMTIP

COM Text InPut

COMTIP DEV,TADR

DEV: immediate allowed

Explanation : Read signs of DEV and write them to TADR until CR comes or until ASL - signs were read. At the end of TIP, APO=0.

Example : Read the incoming signs of COM1 in the ASC-buffer :

COMTIP 0,ASC

COMJTIP

COM Jump Text InPut

COMJTIP DEV,TADR,SAD

DEV: immediate allowed
SAD: immediate allowed

Explanation : Read signs of DEV as with TIP, but jump on SAD as long as CR is received or ASL - signs were read. To show the system that it is about a new JTIP, you first have to set R70=0; this is for technical reasons.

Example : Wait, until any sign is received :

```
        MOV     0,ASC
        MOV     0,R70           ; new JTIP
WAIT:   CBR     0,<>,ASC,CONT   ; anything written ?
        JTIP    1,ASC,WAIT      ; COM2 - inquiry
CONT:
```

COMSST

COM Set line STatus

COMSST DEV,STAT

DEV: immediate allowed
STAT: immediate not allowed

Explanation : Set the level of the control-lines RTS and DTR.

Bit 0 → DTR, Bit 1 → RTS

Example : Set DTR=1 and RTS=0 of COM2 :

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

COMGST

COM Get line STatus

COMGST DEV,STAT

DEV: immediate allowed

Explanation : Read the level of the control-lines CTS and DSR.

Bit 0 → DSR, Bit 1 → CTS

Example : Read DSR and CTS of COM1 in register R10 :

COMSST 0,R10

Info Master-Slave Protocol

16-Bit Protocol

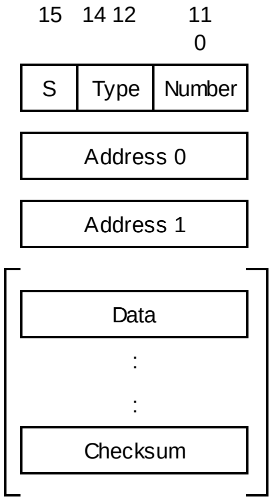
Condition: To be able to use these functions, you need the master-card with the Software Rev. 2.7 or higher, for the Info PC-Master the module info_com.32k and for the rack the module ips_com.32k.

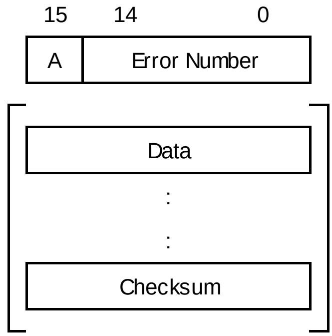
Description: The new functions F_GETB8, F_PUTB8, F_GETB16, F_PUTB16,F_GETB32 and F_PUTB32 were implemented to guarantee a defined data communication with the different processor systems (big- and little-Endian) in the future. This functions should be used only for debug purposes or for parameter definitions.

Format of Protocol:

Master

Slave





S: Determines the Put, Get-Art.

0 = Normal Put Get.
1 = Special Put, Get.

With S = 1, the address is used as command or as parameter. You see a possible application in connection with the SIMOVERT-functions (only in the German manual).

Type: The following values are permitted as data type:

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

Number: The number of data to be received or transmitted of the type Byte, Word or DWords. 0 corresponds to 2^12 = 4096.

Address 0: Low-Word of the memory address or a word parameter depending on the condition of S.

Address 1: Hi-Word of the memory address or a word parameter depending on the condition of S.

Data: Byte, Word or DWord, depending on the defined data type.

A: Answer status bit of the slave-card.

A = 0 means: NACK, checksum was wrong.
A = 1 means: ACK, checksum was right.

Error Number: Number of the error. Zero means: no error.

Checksum: The checksum is as long as a WORD. It is formed by the complement of the word sum of the transmitted words. This means checksum + word sum = -1 (0FFFF).

Construction of the Command Block

Address of the card (word)
Number of data elements (word)
Source address (dword)
End address (dword)

Description: This command block is used for the block-functions F_GETB8, F_PUTB8,F_GETB16, F_PUTB16,F_GETB32 and F_PUTB32, which transmit a memory range from or to the card. Depending on the special block identifier, the source and the end address get another meaning. You can see an example in the SIMOVERT master drive-functions (only in the German manual).

Address of the card:

Bit 14-12 type of card:

- 0 = res
- 1 = Analog Inp (ADC, PT100, FAD ..)
- 2 = IO (16-Bit IO, Valve-IO ..)
- 3 = Posi,DAC (4K-Pos, DAC)
- 6 = Special card (DEnd, Ultrasound..)

Bit 11-4 Address:
0-255, Choice of the card number, axis or outputs

15	14 12	11 4	3 0
	Type	Address	

Number of data elements:

Bit 15 identifier for special block 1, otherwise 0
Bit 11-0 number of bytes / words / Dwords to be written
Range of 0-4095. (0 = 4096)

15	14 12	11 0
Spec.	Type	Number

Source address: Byte address of the buffer of the bytes / words or the Dwords to be transmitted (with F_PUTB, this address is in the memory, with F_GETB, it is on the card).

End address: Destination byte address to which the data is written.
(with F_PUTB, this address is on the card, with F_GETB, it is in the memory)

Special feature: With the identifier of the special block, the addresses get another meaning. With the function F_PUTB, this concerns the end address, with F_GETB, the source address. The other meaning is defined as followed.

Bit	31 - 8	Is used as additional parameter value.
Bit	7 - 0	Determine the function service routine on the card
		0 - 127 are reserved for INDEL functions
		128 - 255 are free for the user

31 … 8	7 … 0
Parameter	Function

Error code in the APO Register

APO = 1	: Line disconnection between the cards. (Link Down)
APO = 2	: Card does not answer. Probably not connected.
APO = 3	: Check-sum error. The transmission was not faultless.
APO = 4	: Time-out error.
APO = 5	: Error number of the answer was <> 0.

Description: The APO register is only fed with the error number, if the task jumps on the ABORT -address; otherwise, it remains unchanged.

F_RESCOM

To reserve a channel

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

Description: Reserves a communication channel to the defined master. A communication channel is needed to guarantee the data communication without interruptions of other tasks. The reserved channel must be set free again, because only a limited number of descriptors is available. Only in case of the InfoPC-Master, the communication descriptor can be used as pointer on the data structure of the communication. This is, how, for example, the exact error number that was sent back by the card can be found out.

Transfer Parameter: Info-Master: Master number, InfoPC-Master:

Return: Communication descriptor

Special feature: With the InfoPC-Master, the master number doesn't make any difference. In case of an error: jump on abort.

Example: Reserve and set free again a communication channel to the master card 2.

```
P_COMCH     = R22                ; Communication descriptor (dword)
   :
   :
GGD    @T_RSCOM,R20              ; get descriptor of F_RESCOM
RCXP   P_COMCH,2,R20             ; reserve a channel to master 2
   :
   :
GGD    @T_FRCOM,R20              ; get descriptor of F_FRECOM
RCXP   P_COMCH,0,R20             ; give channel free again, important !
```

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

F_FRECOM

Set a channel free

```
RCXP    KomDes,0,'F_FRECOM'
```

Description: Set the reserved communication channel to the Master free again.

Transfer Parameter: Communication deskriptor

Return: -

Special feature: In case of an error: jump on abort

Example: Set the previously reserved communication channel free.
Descriptor in P_COMCH.

```
P_COMCH  = R22                ; Communication descriptor (dw ord)
   :
   :
GGD      @T_FRCOM,R20         ; get descriptor of F_FRECOM
RCXP     P_COMCH,0,R20        ; set the channel free


T_FRCOM:    .TXT    'F_FRECOM'
```

F_PUTBxx

Write 8/16/32-Bit block

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

Description: Writes n-bytes / -words / -Dwords from the source address in the memory to the end address in the chosen card. When writing a special-block, the end address is used as parameter. You can see a special-block-example in the chapter of the SIMOVERT master drive-functions (only in the German manual).

Transfer parameter: Communication descriptor, command block

Return: -

Special feature: These functions should only be used for debug-purposes. An uncontrolled writing can lead to a 'crash' of the card and should therefore only be used with sufficient knowledge.
In case of error: jump on abort. Reason to break off is in the APO register (1-5).

Example: Set the Hertz-counter of the Siemens controller, with the axis number 1, to 0. The Word-counter value is on the address 07FE00040.

```
                                ; The channel to the master was already
                                ; reserved.
                                ; The descriptor is in P_COMCH.

CrdAdr   = R24                  ; Address of the card  (word)
Anzword  = R25                  ; Number of words      (word)
SRCADR   = R26                  ; Source address       (dword)
ENDADR   = R28                  ; End address          (dword)
  :
  :
MOV      03810,CrdAdr           ; Siemens controller, axis 1
MOV      01,Anzword             ; write 1 word
ADDR     R0,SRCADR              ; buffer on Reg R0
ASHD     1,SRCADR               ; adapt to the byte address
MOVD     07FE00040,ENDADR       ; address of the Hertz-counter
MOV      0,R0                   ; set the counter to 0

GGD      @T_PUTB16,R20          ; get descriptor of F_PUTB16
RCXP     P_COMCH,CrdAdr,R20     ; write the block
```

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

F_GETBxx

Read 8/16/32-Bit-Block

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

Description: Reads n-bytes / -words / -Dwords from the source address in the card to the end address in the memory . When reading a special-block, the source address is used as parameter.

Transfer parameter: Communication descriptor, command block

Return: [Buffer]

Special feature: In case of error: jump on abort.

Example: Read the Hertz-counter value of the Siemens controller with the axis number 3. The word-counter value is on the address 07FE00040.

```
                                 ; The channel to the master was already
                                 ; reserved.
                                 ; The descriptor is in P_COMCH.

CrdAdr    = R24                  ; Address of the card  (word)
Anzword   = R25                  ; Number of words      (word)
SRCADR    = R26                  ; Source address       (dword)
ENDADR    = R28                  ; End address          (dword)
   :
   :
MOV       03830,CrdAdr           ; Siemens controller, axis 3
MOV       01,Anzword             ; Read 1 word
ADDR      R0,ENDADR              ; Buffer on Reg R0
ASHD      1,ENDADR               ; Adapt to the byte address
MOVD      07FE00040,SRCADR       ; Address of the Hertz-counter

GGD       @T_GETB16,R20          ; Get descriptor of F_GETB16
RCXP      P_COMCH,CrdAdr,R20     ; Read counting result


T_GETB16:  .TXT    'F_GETB16'
```

INFO_SIO - Commands

INFO_SIO

General: 2 INFO_SIO - cards are supported by the standard-firmware what means 4 SIO-channels. These channels are addressed by the device - numbers 0..3.

Conditions: INFO-PCMaster Firmware Rev. 2.88 or higher.

Attention: The INFO_SIO - commands were implemented by .NEWINST of macro assembler. The commands are implemented in the module SIO_NSIO; this means that the module-number of SIO_NSIO (see file INFO.IND) must be assigned to the equal MOD_NSIO in the file SIO_NSIO.INC.

DataFrame: The Transmission Format is specified in the usual INDEL - design. The Baud rate is an exception: it is freely selectable up to 115200 Baud, thus, also devices with exotic Baud rates can be addressed (see command SIOSETD).

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

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

Errors: Johann jumps on its abort-address in case of the following errors (the error - number stands in APO - register):

INFO_SIO Communication - Error
001 LinkDown (no closed INFO-link available)
002 INFO_SIO - card does not answer
003 Check sum - error (the INFO-link is bad, the communication to the INFO_SIO - card is seriously disturbed -> check error-counter)
004 Timeout (when communicating with the INFO_SIO-card, a timeout occured)
005 internal communication error (call INDEL)

INFO_SIO protocol - error

010	Put/Get without having selected a protocol before (should happen only in case of self-protocol-implementations)
020	Receive abort character (only with SIOTIP)
021	Framing error
022	Parity error
023	Overrun error
024	Input-buffer overflow
025	DSR with SIOSETD = 0
026	Timeout with SIOTOP, SIOBTOP, SIOTIP or SIOBTIP

Debug Note: The INFO_SIO - commands were partly implemented with macro what means that a single step in an INFO_SIO - command leads you and your Johann somewhere in the system's profundity. We therefore advise you, not to debug the INFO_SIO - commands with single step, but to set a breakpoint immediately after the command and work off the command with F9 (go).

Special Signs: A text-entry is closed, when either the desired number of signs (SIOBTIP) or the desired end string (SIOTIP) has arrived. If, with SIOTIP, no end string was defined, this end string thus behaves compatible to TIP on the 2K-SIO; this means, that the LF (0A) and the 00 - character are ignored and the input is completed, as soon as a CR (0D) has arrived. If the abort-character 'ABC' is received with SIOTIP, the input buffer is cleared and Johann jumps on abort.

Buffer: The INFO_SIO has, per channel, 2KByte input- and 2KB output buffer. The following max. block length are allowed :

SIOTOP	512 Bytes
SIOBTOP	512 Bytes
SIOTIP	255 Bytes (because ASL has the size of only one Byte)
SIOBTIP	512 Bytes

OUT: The output buffer always receives data from SIOTOP or SIOBTOP, provided it is not full, even when the output data channel is decelerated by CTS or XOFF. The buffer can be deleted with SIOSETD only.

INPUT : The input buffer is always ready to receive, provided it is not full. TIP takes data from the input buffer only. The buffer can be deleted by SIOSETD. It is also deleted with FRAMING, PARITY and OVERRUN -Error.

DTR: The output DTR is used to decelerate the input device. Because the input buffer is always open, DTR (-15V) only decelerates if this buffer leaves free 256 signs at least. If this value is fallen below , also DTR gets active again (+15V).

RTS: The RTS output always gets active (+15V) if there is data in the output buffer.

CTS: Via the CTS input, the output data can be decelerated. As soon as it is inactive (-15V), the output is stopped (the processing sign is still transmitted).
If it is not needed: CTS -> +5..15V.

DSR: If, with SIOSETD, the DSR input is inactive (-15V), Johann jumps on its ABORT address. This is used to recognize if there is an output device connected and ready to receive data (end of paper). After this, the DSR works as the CTS line.
If it is not needed: DSR -> +5..15V.

DCD: This input is, in the modem mode, used as data carrier detect. If the DCD is inactive (-15V), the input channel is switched off and it isn't possible to receive any wrong or undefined signs. If it is active (+15V), the input is switched through normally.
If it is not needed: DCD -> +5..15V.

XON/XOFF: In case of XON/OFF-mode, there are the control lines processed, but also XON (011) and XOFF (014). If XON/XOFF is switched off, those signs are handled as others are. If XON/XOFF is switched on, the user doesn't note anything; the receiving XON/XOFF doesn't reach the INP buffer.

XOFF is sent if the INP buffer leaves free 256 signs at least.
If XOFF is received, the output is stopped immediately
(the processing sign is still transmitted).

XON is sent if XOFF was sent before and the INP buffer has capacities again; even though after Power-On with the first SIOSETD (only, if XON/XOFF was chosen!).
If XON is received and there is still data in the OUT-buffer, the transmitting will be continued.

Note: The XON/XOFF - mode is not yet supported in the acutel INFO_SIO revision.

Note: The control lines CTS,DSR,DCD are also processed in the 20mA, RS422 and XON/XOFF-mode.
If they are not needed: all on +5..15V !

SIOSETD

SIO SET Device

SIOSETD DEV, BAUD:D

Explanation : With the SIOSETD - command, the SIO-channel DEV is reserved for this Johann. In the High-Byte of DEV, you can define the DataFrame (see Introduction). In the Low-Byte, the channel number is specified (0..3).
With BAUD, you can select any baud rate (up to 115200 Baud); also exotics are allowed, as for example 1326 baud. Note, that the baud rate must be specified as double word.

If the DSR input is inactive (-15V) when executing SIOSETD, Johann jumps on abort (APO = 025).

With SIOSETD, the input and output buffers are deleted.

Example: Reserve the SIO - channel 0 on card 1 (->channel 2). Initialize the transfer-format with 19200 Baud, E,8,1.

SIOSETD 0C002,19200:D

SIORESD

SIO RESet Device

SIORESD DEV

Explanation : Set a SIO-channel, reserved with SIOSETD, free again.

Example : Set channel 2 free again

SIORESD 2

SIOTOP

SIO Text OutPut

SIOTOP DEV,TADR,EADR,TIMOUT:D

Explanation : Output of text-string TADR (ended with 00 char) + EADR (ended with 00 char) on DEV.

With EADR, any terminating string can be defined, for example CR,LF ...

SIOTOP writes the character string in the output buffer and returns immediately. However, if the buffer is full and is not processed any more (because, for example, CTS or DSR are inactive -> it isn't possible to send), a max. waiting period in ms can be defined. TIMOUT = 0 means -> no time-out monitoring.

We recommend to specify a time-out period in any case for an unnecessary blocking of the task can be prevented.

Note: max. length of the output string (TADR+EADR) = 512 signs

Note: The time-out period must be defined as a double word. The Time-out monitoring is assisted only off INFO_SIO Rev. 1.10 and INFO-PCMaster Firmware Rev. 2.91.

Example : Write the text 'Oh, du meine SIO' on channel 1 and end with CRLF. Note, that 0A0D is defined as double word for a 00 char is automatically generated.

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

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

SIOBTOP

SIO Block Text OutPut

SIOBTOP DEV,TBLK,N,TIMOUT:D

Explanation : Output of N signs out of the text - block TBLK on the channel DEV. With this command, all signs from 00..FF can be given without restrictions.

SIOBTOP writes the character string in the output buffer and returns immediately. However, if the buffer is full and is not processed any more (because, for example, CTS or DSR are inactive -> it isn't possible to send), a max. waiting period in ms can be defined. TIMOUT = 0 means -> no time-out monitoring.

We recommend to specify a time-out period in any case for an unnecessary blocking of the task can be prevented.

Note: The number of signs N is limited to max. N=512.

Note: The time-out period must be defined as a double word. The Time-out monitoring is assisted only off INFO_SIO Rev. 1.10 and INFO-PCMaster Firmware Rev. 2.91.

Example : Send this 5-Byte control sequence to the printer that is connected to channel 1.

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

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

SIOTIP

SIO Text InPut

SIOTIP DEV,TADR,EADR,TIMOUT:D

Explanation : Read the signs from DEV to TADR, until the TIP-end-identification-string is arrived or until TIMOUT ms are run down.

With EADR, any end-identification-string can be defined, for example CR,LF .... Without end-string-specification, the command reacts as the TIP on the 2K-SIO (0A and 00 char are ignored, 0D applies as end-identification)

Note: The time-out period must be defined as a double word.

Example : Read signs from channel 0 to the ASC-buffer until CRLF is arrived or 2 sec are run down. Note, that 0A0D is defined as double word for a 00 char is automatically generated.

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

SIOSTAT

SIO STATus

SIOSTAT DEV,STRUCT

Explanation : Read the SIO-Channel status to STRUCT

STRUCT :	offset 0	Modem status register (MSR)
	offset 1	Actual number of signs in the input buffer
	offset 2	Actual number of signs in the output buffer

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

Example : Read Status from channel 2. Write MSR to R0, input buffer to R1 and output buffer to R2.

```
SIOSTAT   2,R0
```

SIOBTIP

SIO Block Text InPut

SIOBTIP DEV,TBLK,N,TIMOUT:D

Explanation : Read signs from DEV, until either N signs were read or until TIMOUT ms are run down. With this command, all signs from 00..FF can be read in without any restrictions.

Note: The time-out period must be defined as a double word

Example : Read 6 signs from channel 3 to R00..R02, wait max. 500ms.

SIOBTIP 3,R0,6,500:D

PSEUDO-Commands

PSEUDO COMMANDS

```
TITLE:                      .TITLE      "**- PSEUDO COMMANDS -**"
                            .SUBTITLE   "- Common -"

LISTING:                    .LINE 85                ; 85 Lines / Pages
                            .NOLIST                 ; Listing off
                            .LIST                   ; Listing on
                            .EJECT                  ; New page

FILE:                       .INCLUDE    NEXTFIL     ; Load additional file

ADDRESS:                    .LOC        01000       ; Start of program

ASSIGNMENTS:    WT1:        .EQU   012345678        ; With .EQU or =
                WT2         =      -WT1             ; Is filed in DW
                FW1         =      123.456          ; Is filed in LONG
                FW2         .EQU   -123.456
                PI          =      3.1415926536
                RL1:        .EQU   033(R77)         ; R77 - relative
                RL2         =      044(R22)         ; R22 - relative
                BUFFER      =      ASC
                NAME        =      R11              ; NAME = R11

NUMBERS:        DZ1:        .EQU   999              ; Decimal
                DZ3:        .EQU   1E3              ; Exponent
                HX1:        .EQU   0ABCD            ; Hex
                FL1:        .ECU   123.456E15       ; Floating point
                FL2:        .ECU   -123.456E-15
                FL3:        .ECU   2.0              ; Decimal point = floating!

COORDINATES:    YYXX:       .EQU   1234|5678        ; DW out of two decimal numbers
                                                    ; (| = ALT124 )

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

BLOCKS:                     .BLKB    Number of bytes        ; Byte block
                            .BLKW    Number of words        ; Word block
                            .BLKD    Number of double words ; Double word block

TEXT:                       .TXT        '^
                <000009>    Text with   CR/LF
                <000009>    and without CR/LF^
                <000009>    New-Line'
                            .TXT        ' <09>special sign<0D><0A>'
```

INDEX

ASCII-SET

Special Signs

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-Character

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	á					
1	1			!	1	A	Q	a	q	ü	æ	í					
2	2			"	2	B	R	b	r	é	Æ	ó					
3	3			#	3	C	S	c	s	â	ô	ú					
4	4			$	4	D	T	d	t	ä	ö	ñ					
5	5			%	5	E	U	e	u	à	ò	Ñ					
6	6			&	6	F	V	f	v	å	û	ª					
7	7				7	G	W	g	w	ç	ù	º					
8	8			(	8	H	X	h	x	ê	ÿ	¿					
9	9			)	9	I	Y	i	y	ë	Ö						
10	A			*	:	J	Z	j	z	è	Ü						
11	B			+	;	K	[	k	{	ï		½					
12	C			,	<	L	\	l			î	£	¼					
13	D			-	=	M	]	m	}	ì		¡					
14	E			.	>	N	^	n	~	Ä		«					
15	F			/	?	O	_	o	_	Å		»					