Manual

Toolbox

Order-No.: VIPA CP4-HB74E
Rev. 00/33

The information contained in this manual is supplied on an "as-is" basis and is not guaranteed in any way. The contents is subject to change without notice. The hardware (software) described in this manual is supplied on the basis of a general or a specific license (company license) and it may only be used or copied for purposes meeting the conditions of this license. We will claim compensation for damages.

Any information that became available after this manual was printed is provided in a file located on the accompanying floppy disk. To read the floppy (if it was supplied) insert the VIPA driver diskette #1 into your drive A and enter the following command:

TYPE README.TXT

Please use the 'NOTEPAD' to view the file in Windows®, and in OS/2 you can use 'E'.

Ohmstraße 4, D-91074 Herzogenaurach
Tel.: +49 (91 32) 744-0
Fax.: +49 (91 32) 744-144
EMail: info@vipa.de

Hotline: +49 (91 32) 744-114

About this manual

This manual describes the Utility-Software for the CP386 (CP3) and CP486 (CP4) modules.

As operating system we recommend MS-DOS 5.00 or later. The installation and the description for this operating system is available from the respective manual supplied by Microsoft or. in the advanced manuals available from MARKT UND TECHNIK and DATA BECKER. Information on system related functions may be obtained from the book PC INTERN available from DATA BECKER. VIPA has developed utilities and tools for operating the CP under the MS-DOS operating system. The software described in these chapters is supplied on VIPA diskette CP4-SW593.

Overview

The drivers described in chapters 1, 2 and 3 provide the means for communications between the CP module and the PLC. These three drivers may be used in conjunction with any CP3 or CP4 module.

The first three chapters describe the installation of the respective driver and the communication functions provided by the driver. Here you can also find the interfacing for PASCAL (not CP486NT) and C and a description of the operation with WINDOWS.

Chapter 1: Linkage with PLC by CP386COM

This chapter contains the discription of the CP386COM-driver. This driver supports only single-prozessor operations with block to block requests.

Chapter 2: Linkage with PLC by CP486COM

The CP486COM-driver that is the subject of this chapter was developed from the CP386COM. This driver supports multi-processor operations. Block requests are processed according to the FIFO-principle.

Chapter 3: Linkage with PLC by CP486NT

This contains the description of the CP486NT. This driver is specific to WINDOWS NT. The range of functions is nearly identical to those of the CP486COM.

Chapter 4: MS-DOS-utilities for Solid-state disk-operation

Chapter 4 contains a description of the solid-state disk driver and its integration into the system. The chapter also describes the programs required to operate a solid-state disk, e.g. the formatting, read and write programs.

Chapter 5: Auxiliary programs

This chapter contains a description of the support programs required for coupling different computers and to display the process image. It also contains a note on the test-program for the system that is available as an option.

Contents

1 Linkage with PLC by CP386COM

1 Linkage with PLC by CP386COM

1.1 General description

Data transfer between CP and PLC is supported by handling modules on PLC side and by software interrupts on CP side. Following routines are available:

Bank No	Function	Operation on PLC side	Operation on CP side
Bank 0	PLC job: read data from CP (PLC active)	Handling module (FB3)	Interrupt service routine
Bank 1	PLC job: send data to CP (PLC active)	Handling module (FB3)	Interrupt service routine
Bank 2	CP job: read data from PLC (CP active)	Cyclically called handling module (FB1)	Software interrupt
Bank 3	CP job: send data to PLC (CP active)	Cyclically called handling module (FB1)	Software interrupt
Bank 7	Transfer process image to CP	Cyclically called handling module (FB1)	Software interrupt or direct access to bank

Tab. 1-1: Overview bank function by CP386com

Following data structures in the PLC can be accessed on CP side:

- single elements in format byte, words and doublewords
  DB, DX, markers, inputs, outputs, timer, counter, flag word
- data blocks
  DB, DX, FB, FX, OB, PB, SB, BA, BB, BT, BS

Following PLC accesses to CP are possible:

- the linkage supports all types of MS-DOS device-oriented accesses.
- all jobs issued by the PLC are based on MS-DOS device functions.

Functions described below are available upward for CP386COM version 1.00 (Software CP4-SW593 version 2.x) and handling module version 2.00 (CP4-SW977 and CP4-SW978 version 2.x). The program CP386COM is named in the following description as COM-driver.

1.1.1 MS-DOS Driver Program

For communication via banks between PLC and CP a specific communication driver must be loaded in the CP. This driver is specific for the communication with the VIPA handling modules. The driver supplies functions which are easy to handle. The user needs no detailed information concerning structure and operation of the banks. The driver contains software to control all banks. At the moment it currently supports:

- banks 0 and 1 (PLC active, CP passive).
- banks 2 und 3 (PLC passive, CP active)
- bank 7 (process image).

The driver automatically supports all functions for all banks, no further configuration is necessary for particular banks.

1.1.2 Driver Installation

It is recommended to load the driver at the best during the start of the CP, that means already in AUTOEXEC.BAT. As well a later call is possible.

Notice, that as a rule , the handling modules for synchronization are called during the restart. They just wait for a fixed time for a reaction of the CP. Therefore PLC and CP remain unsynchronized if the driver is not started within this time and no communication is possible.

Call: CP386com.exe [/ini] [/txx] [/ixx] [/notsr] [/?] [/h]

The driver is a resident program (TSR-Utility) and allocates about 26 KByte of main memory (program and data). The driver can be loaded only once at a time. Any further call causes a message, that the driver has already been installed. A removal of the driver from main memory (de-installation) is not pssible, thereto the CP has to be rebooted.

1.1.3 Driver options (Revision 1.6 and following):

/INI or /ini

With this option-switch the communication bank is initialized, pending jobs are stopped and the whole bank is cleared. The driver is not installed. This initialization can be as often as necessary.

/Txx or /txx

This option specifies the timeout in seconds for the driver. The default timeout value is 10seconds. Possible values for xx: 1sec .. 30sec

Default: 10sec

/Ixx or /ixx

This option specifies the number of the software-interrupt for the communication via bank 2 and 3 (CP active mode).

Possible values: 78h, 7Ah .. 7Fh.

Default: 78h.

/NOTSR or /notsr

Using this option when calling the driver in the command line causes the program to be installed nonresident. The program will not be finished, no further DOS-commands can be entered or programs can be started subsequently. This option is only meaningful, if communication ensues exclusively via banks 0 and 1 (CP passive). By pressing the F10-key and subsequent confirmation with "j" the driver will be removed again.

/?, /H or /h

This option shows a list of all possible options of the driver.

The COM-driver is designed to work with CP-modules of the VIPA GmbH exclusively and can be installed on these systems solely. Loading the driver on different CP-systems, even on i386 or i486 processor, causes an immediate system-hang-up.

1.1.4 Reserved interrupts

The driver uses several software interrupts for operation and communication with the applications software of theCP-module:

- INT 1Ch Timer-Interrupt and INT 28 DOS-Idle-Interrupt

  The so called ticker-interrupt with number 1Ch, as well as the so called DOS-idle interrupt are used for routine and cyclical check of the bank. By this, it is regularly checked, whether the PLC tries to synchronize the banks recently. After executing the CP-specific functions, the initial interrupt service routine is called.

- INT 74h (IRQ 12):

  The CP uses the hardware-IRQ 12, which occupies software-interrupt 74h. This interrupt will be triggered, if BASP is active in PLC, or if the highest memory location of every bank (byte1023) is written by the PLC. The usage of the interrupt permits fast reaction of the CP to a request by the PLC. After processing the CP-specific functions the initial interrupt service routine is called. In this manner, different devices can use IRQ 12.

- INT 78h Service-Interrupt:

  This interrupt is to be used by applications software in the CP to call functions of the driver, for example data exchange with the PLC via banks 2 and 3. Different functions can be triggered by corresponding assigment of the processor registers. If INT 78 is called with register values, which are invalid for the CP communication, the initial interrupt service routine is called.

1.1.5 Different Data Representation in Memory

For the transfer of data between CP and PLC, the different representation of words and doublewords (extended words) on CP and PLC (programmable controllers) has to be taken into account.

Unlike CP´s the PLC stores the datatype word in a different form in memory, High-Byte and Low-Byte are stored reverse. In doublewords all 4 bytes are stored in exact reverse order. If data of type word, doubleword is exchanged between PLC and CP, it´s quite natural that an interchange has to be undertaken, otherwise data is wrong after transmission. As far this is possible in a meaningful way the COM-driver processes this adaptation automatically.

For data exchange via banks 0/1 the user has to execute the interchange on its own, either on the CP or on the PLC. The driver processes the interchange automatically for the banks 2/3 and bank 7 in all cases.

Operation:

- During transmission of bytes no interchange takes place.
- During transmission of words High- Byte and Low-Byte are interchanged.
- During transmission of extended words all 4 bytes are reversed according to their order.

Data representation in PLC (programmable controller):

Address n	Byte	Representation Byte
Address n	High-Byte	Representation Word
Address n+1	Low-Byte	
Address n	High-Byte High-Word	Repres. Doubleword
Address n+1	Low-Byte High-Word	
Address n+2	High-Byte Low-Word	
Address n+3	Low-Byte Low-Word	

Data representation in CP

Address n	Byte	Representation Byte
Address n	Low-Byte	Representation Word
Address n+1	High-Byte	
Address n	Low-Byte Low-Word	Repres. Doubleword
Address n+1	High-Byte Low-Word	
Address n+2	Low-Byte High-Word	
Address n+3	HighByte High-Word	

1.2 PLC Jobs for CP (Functions for Bank 0 and 1)

1.2.1 Overview

All PLC-jobs are transacted via banks 0 and 1. In this way the CPU can use series of MS-DOS-functions. Die PLC-instructions are processed in background via interrupts, as soon as this driver is being installed. In this way no additional software for the CP is necessary to operate the PLC-jobs.

The driver permits to call several MS-DOS system functions from the PLC. Hereby all parameters and information are exchanged transparent between PLC and MS-DOS. The CP-driver software is solely useful, to record the passed parameters correctly in the processor-register and to transfer the returned values in a suitable form to the PLC. The only fundamental restriction is, that only one part of the MS-DOS system functions can be called by the PLC. It´s not recommendable to call all functions by the PLC, because a great number of functions cannot be used meaningful by the PLC at all. For further reasons a breakdown of the operating system can occur if a series of functions is used in a non-adequate way. Therefore the driver software only supports such functions, which can be used meaningful by the PLC. As function numbers for calling, exactly these function numbers of the MS-DOS system function are to be specified.

Bank no.	Func. no.		Function
	hex	dez	
1	$0D	13	reset all disk drives
1	$0E	14	select disk drive
0	$19	25	determine current disk drive
1	$39	57	set up directory
1	$3A	58	delete directory
1	$3B	59	change directory
0	$47	71	determine current directory
1	$3C	60	create file
1	$5A	90	create file without overwriting
1	$3D	61	open file
1	$68	104	write file physically to disk (without close)
1	$3E	62	close file
1	$41	65	delete file
1	$56	86	rename file
1	$42	66	set file pointer
0	$C2	194	read file pointer (no MS-DOS system function) !!
0	$3F	63	read from file or device
1	$40	64	write to file or device
0	$2A	42	read date
0	$2C	44	read time
1	$4B	75	execute program
0	$30	48	determine MS-DOS version
0	$59	89	read detailed error information
1	$FF	255	optional interrupt

Tab. 1-2: Overview MS-DOS system functions

1.2.2 Interface concept for banks 0 and 1

These two banks serve for reading and writing of data from or to the CP respectively. If the PLC tries to read data from the CP or write data, it has to call the suitable handling module (SEND or FETCH and RECEIVE). As a result these handling modules provide a job unit in bank 0 or bank 1. A maximum of one job can be entered in the banks 0 und 1 at a time. The size of the data to be transferred ranges from one word up to 504 words. The structure of the job unit inside bank 0 and bank 1 is absolutely identical. The distinction reading or writing is only due to the bank number.

On the CP-side, there is a job catalog deposit. As soon as the CP registers a job in bank 0 or in bank 1, it takes the job number from the job unit and searches for the respective parameter block on its side of the job catalog. In this catalog it is deposit, what should happen with the data, which e.g. will be transferred from the PLC to the CP. The same happens for reading correspondingly, that means, the CP searches in the bank by means of the job number, whether a catalog is filed on its side. If yes, it makes the requested data available corresponding to its catalog.

1.2.3 Processing a Write Job

The PLC-applications software calls the handling module SEND. At this point the PLC programmer sets the parameter for the job number, the transmission length in words, as well as the source of data in the PLC. The handling module checks these specifications. If the specifications are correct, it verifies, whether the bank is unassigned. Unassigned means, whether the bank reports a running job. In this case the send job would be rejected. If the bank is available the handling module creates a job unit and stores the data to be written subsequent to the job unit in the bank and sets the job status to 'job is running'. This is the identification for the CP, that a new job to be executed is waiting in the bank. Accordingly, if the job is executable, the CP resets after executing the job, the identification 'job is running' and sets instead of one of the identifications 'job finished with error' or 'job finished without error'. If an error occurred, The CP reports a corresponding error code. With the handling module CONTROL the user gets information about the status of the running job or the last job.

1.2.4 Processing a Read Job:

With the handling module FETCH the PLC-applications software passes a read job to the CP. This handling module verifies as well as the handling module SEND the specified parameters, creates a job unit in bank 0 and sets the status to 'job is running'. By this the CP detects the existence of a new job and executes this job, as already described under 'Write'. If the CP is able to supply these data, it stores the data subsequent to the job unit in bank 0 and sets the status to 'finished without error'. Additionally it sets the identification to 'data for receive available'. With the handling module CONTROL the PLC applications software allows reading the status instantly. If the data is prepared by the CP, it can be transmitted to the PLC via handling module RECEIVE. If this was successful, the handling module RECEIVE resets the identification 'data for receive available'. From this time a new receive job can be entered by the PLC software.

In the case, that the PLC applications software tries to access a new write or read job, while a write or read job is still running, the PLC applications software receives the identification 'interface busy''. This identifications are placed only by handling modules, they are of no account for the communication with the CP.

The interface

The interface (Data construct in banks) isn't suitable for multi processor operating in the PLC in the current version. Always one and the same CPU may access a CP in the PLC to every time!

1.2.5 Parameterization of Handling Modules:

The handling modules SEND (FB3), CONTROL (FB4), FETCH (FB5) and RECEIVE (FB6) are parameterized as follows

If the bank number of the CP doesn't agree with then number in the handling block, the CPU stopps with QVZ.

1.2.5.1 FB3 (SEND), send job to CP

This handling module transfers a data block of up to 504 words from a DB to the CP. For identification purposes a job number is also sent to the CP. The handling module supplies the result by means of a display word in a marker word to the PLC application program. Parameterization errors are signalled via a marker byte. The handling module is directly and indirectly parameterizable:

```
Module#FB3
BSTNAME     #SEND
BIB
BEZ    #INSS            D:KY
BEZ    #A-NR      D:KY
BEZ    #DOSP      D:KY
BEZ    #ANZW      D:KY
BEZ    #QT/N      D:KY
BEZ    #QANF      D:KF
BEZ    #QLAE      D:KF
BEZ    #PAFE      A:BY
```

Transfer parameters:

INSS:	D	KY	IN	Code if direct or indirect parameterization
				= 0 direct parameterization via formal operands
				≠ 0 indirect parameterization - transfer parameters are filed in opened DB
			SS	number of basic bank (must be divisible by 8)
A-NR:	D	KY		If direct parameterization left byte has the job number (1..127) and right byte the function number for CP-driver. If indirect parameterization, A-NR has the DW-no, from which on the parameters are in the open DB. Thereby the content of the parameter is evaluated as word.
DOSP:	D	KY		DOS-parameter which is also transferred at certain functions. It can be a handle, an access or a drive-number.

ANZW: D KY left byte: reserve
right byte: contains the MW-no. where the display word is to be stored (permitted are MW0..MW198)
Display word can have following information at SEND status:
Bit .0 =0
.1 job runs (job transferred without errors)
.2 =0
.3 job finished with error
.4 F-Nr. 2 raised to 0
2 raised to 1
2 raised to 2
2 raised to 3
Error numbers are dual encoded.
Error number 1 interface occupied by PLC (job runs)
6 interface occupied by CP

QT/N: D KY left byte: reserve
right byte: source module no. (2...255), DB-no. of the module with data to be transferred is specified

QANF: D KF Initial address in DB (0...32761), the DW-no. is specified from which on the data to be transferred are filed in the DB.

QLAE: D KF Number of data words to be transferred (1...504)

PAFE: A BY Markerbyte, in which the PAFE-message is transferred to PLC program (permitted 0...255)
Error acknowledge message of handling module:
= 0 no error occurred
≠ 0 error occurred, error number in PAFE-Byte:
3 basic bank number is not divisible by 8
5 bank is not synchronized yet by the CP
10 invalid job number (out of 1...127)
12 no DB open for indirect parameterization
13 source module is not existent
14 source module too short
15 QLAE is invalid (out of 1...504)
16 DB for indirect parameterization too short
18 invalid source module no. (out of 2...255)
19 invalid source initial address (out of 0...32761)
20 invalid marker word no. for ANZW (out of 0...198)

Attention: If the bank number of the CP does not coincide with the number parameterized in this module, the CPU goes in stop with QVZ!

Parameter storage in a DB if parameterization is indirect:

A-NR points to the beginning

DL	DR
INSS	
A-NR	F-NR
DOSP	
ANZW	
QT/N	
QANF	
QLAE	

PAFE is not parameterizable indirectly

0 can be parameterized at all other formal operands (SS, DOSP, ANZW, QT/N, QANF, QLAE) because these are not evaluated at indirect parameterization.

Scratch pads used: MB200-255

1.2.5.2 FB4 (CONTROL), show CP status

This handling module outputs the status of a write or read job. For identification purposes a job number is also sent to the CP. The handling module supplies the result by means of a display word in a marker word to the PLC application program. Parameterization errors are signalled via a marker byte. The handling module is directly and indirectly parameterizable:

```
Module#FB4
BSTNAME     #CONTROL
BIB
BEZ    #INSS      D:KY
BEZ    #A-NR      D:KF
BEZ    #DOSP      D:KY
BEZ    #RWAW      D:KY
BEZ    #PAFE      A:BY
```

Transfer parameters:

INSS:	KY	IN	Code if direct or indirect parameterization
			= 0 direct parameterization via formal operands
			≠ 0 indirect parameterization - transfer parameters are filed in opened DB
		SS	number of basic bank (must be divisible by 8)

A-NR: If direct parameterization right byte has the job number (0..127)
If indirect parameterization, A-NR has the DW-No, from which on the parameters are in the open DB. Thereby the content of the parameter is evaluated as word.
For job number 1...127, the status of the suitable job is read.
For job number 0, the status of the actually running or finally executed job is read.

DOSP: left byte: reserve
right byte: contains MW-no., where DOS-parameter is to be stored.

RWAW: RW = Control for read job in bank 0
= Control for write job in bank 1
AW contains the MW-no. where the display word is to be stored (permitted are MW0..MW198)
Display word can have following information:
Bit .0 receive meaningful
.1 job runs (job transferred without errors)
.2 finished without error
.3 job finished with error
Error number is dual encoded.
Error number 4 not defined job status on the CP
5 no job under this job number
6 interface occupied by CP
Bits 8-15 contain a probable error number of the CP

PAFE: A BY Markerbyte, in which the PAFE-message is transferred to PLC program (permitted 0...255)
Error acknowledge message of handling module:

= 0	no error occurred
≠ 0	error occurred, error number in PAFE-Byte:
3	basic bank number is not divisible by 8
4	bank is not existent (acknowledgement delay at bank access)
5	bank is not synchronized yet by the CP
10	invalid job number (out of 1...127)
12	no DB open for indirect parameterization
16	DB for indirect parameterization too short
20	invalid marker word no. for ANZW (out of 0...198)
21	invalid marker word no. for DOSP (out of 0...198)

Parameter storage in a DB if parameterization is indirect:

A-NR points to the beginning

DL DR
INSS
A-NR
DOSP
RWAW

PAFE is not parameterizable indirectly

0 can be parameterized at all other formal operands (SS, RWAW) because these are not evaluated at indirect parameterization.

Scratch pads used: MB200-255

1.2.5.3 FB5 (FETCH) data request to CP

This handling module transfers the read job to the CP. For identification purposes a job number is also sent to the CP. The handling module supplies the result by means of a display word in a marker word to the PLC application program. Parameterization errors are signalled via a marker byte. The handling module is directly and indirectly parameterizable

```
Module#FB5
BSTNAME     #FETCH
BIB
BEZ    #INSS      D:KY
BEZ    #A-NR      D:KY
BEZ    #DOSP      D:KY
BEZ    #LAE       D:KF
BEZ    #ANZW      D:KY
BEZ    #PAFE      A:BY
```

Transfer parameters:

INSS: IN Code if direct or indirect parameterization
- = 0 direct parameterization via formal operands
- ≠ 0 indirect parameterization - transfer parameters are filed in opened DB

SS Number of basic bank (must be divisible by 8)

A-NR: If direct parameterization left byte has the job number (1..127) and right byte the function number for CP-driver. If indirect parameterization, A-NR has the DW-No, from which on the parameters are in the open DB. Thereby the content of the parameter is evaluated as word.

DOSP: DOS-parameter which is also transferred at certain functions. It can be a handle, an access or a drive-number.

LAE: Number of data words to be read (corresponding to the passed function no. for the DOS-driver, e.g. number of data which are to be read from a file

ANZW: left byte: reserve
right byte: contains the MW-no. where the display word is to be stored (permitted are MW0..MW198)
Display word can have following information at FETCH status:
Bit .0 =0
.1 job runs (job transferred without errors)
.2 =0
.3 job finished with error (job was not transferred)
Error numbers are dual encoded.
Error number 1 interface occupied by PLC (job runs)
6 interface occupied by CP

PAFE: Markerbyte, in which the PAFE-message is transferred to PLC program (permitted 0...255)
Error acknowledge message of handling module:
= 0 no error occurred
≠ 0 error occurred, error number in PAFE-Byte:
3 basic bank number is not divisible by 8
5 bank is not synchronized yet by the CP
10 invalid job number (out of 1...127)
12 no DB open for indirect parameterization
16 DB for indirect parameterization too short
20 invalid marker word no. for ANZW (out of 0...198)

Attenttion:
If the bank number of the CP does not coincide with the number parameterized in this module, the CPU goes in stop with QVZ!

Parameter storage in a DB if parameterization is indirect:

A-NR points to the beginning

DL	DR
INSS	
A-NR	F-NR
DOSP	
LAE	
ANZW	

PAFE is not parameterizable indirectly
0 can be parameterized at all other formal operands (SS, DOSP, LAE, ANZW) because these are not evaluated at indirect parameterization.

Scratch pads used: MB200-255

1.2.5.4 FB6 (RECEIVE), receive data from CP

This handling module transfers a data block of up to 504 words from the CP to a DB. Before calling the RECEIVE module, the CP must be informed by means of the FETCH handling module about data which it requires. For identification purposes a job number is also sent to the CP. This job number is returned together with the data.

The handling module supplies the result by means of a display word in a marker word to the PLC application program. Parameterization errors are signalled via a marker byte. The handling module is directly and indirectly parameterizable:

```
Module#FB6
BSTNAME     #RECEIVE
BIB
BEZ    #INSS      D:KY
BEZ    #A-NR      D:KF
BEZ    #ANZW      D:KY
BEZ    #ZT/N      D:KY
BEZ    #ZANF      D:KF
BEZ    #ZLAE      D:KF
BEZ    #PAFE      A:BY
```

Transfer parameters:

INSS: IN Code if direct or indirect parameterization
= 0 direct parameterization via formal operands
≠ 0 indirect parameterization - transfer parameters are filed in opened DB
SS number of basic bank (must be divisible by 8)

A-NR: If direct parameterization, A-NR has the job number (0..127). For a number 1...127 it is checked whether data being prepared by the CP, have the same job number. Data are taken over only if they have the same job number. In case of job number 0, data are taken over by the CP in any case.
If indirect parameterization, A-NR has the DW-No, from which on the parameters are in the open DB. Thereby the content of the parameter is evaluated as word.

ANZW: left byte: reserve
right byte: contains the MW-no. where the display word is to be stored (permitted are MW0..MW198)
Display word can have following information at SEND status:
Bit .0 =0
.1 job runs (still no data received from the CP)
.2 job finished without error (data are in DB)
.3 job finished with error (error no. in Bit 4...7)
Error numbers are dual encoded.
Error number 2 no data existent
3 no data present for this job
4 not defined job status on the CP
6 interface occupied by the CP

ZT/N: left byte: reserve
right byte: target module no. (2...255), DB-no. of the module with data to be transferred is specified

ZANF: Initial address in DB (0...32761), the DW-no. is specified from which on the data to be transferred are filed in the DB.

ZLAE: Number of data words (1...504) at least to be transferred, CP passes only so many data words as it also supplies.

PAFE: Markerbyte, in which the PAFE-message is transferred to PLC program (permitted 0...255)
Error acknowledge message of handling module:
= 0 no error occurred
≠ 0 error occurred, error number in PAFE-Byte:
3 basic bank number is not divisible by 8
4 bank not existent (acknowl. delay at access)
5 bank is not synchronized yet by the CP
10 invalid job number (out of 1...127)
12 no DB open for indirect parameterization
13 source module is not existent
14 source module too short
15 ZLAE is invalid (out of 1...504)
16 DB for indirect parameterization too short
18 invalid target module no. (out of 2...255)
19 invalid target initial address (out of 0...32761)
20 invalid marker word no. for ANZW (out of 0...198)

Attention:

Error number 4 in PAFE is dedicated for future improvements. In the moment, it is generally not possible to recognize QVZ via software in the CPUs for the AG115. In this version, the CPU goes in QVZ in stop, if the bank number of the CP does not coincide with the number being parameterized in this module.

Parameter storage in a DB if parameterization is indirect:

A-NR points to the beginning

DL	DR
INSS	
A-NR	
ANZW	
ZT/N	
ZANF	
ZLAE	

PAFE is not parameterizable indirectly

0 can be parameterized at all other formal operands (SS,, ANZW, ZT/N, ZANF, ZLAE) because these are not evaluated at indirect parameterization.

Data storage in a DB:

ZANF points to the beginning

DL	DR
A-Nr	F-Nr
No. of word being read	
Data being read by the CP are filed from here on	

Scratch pads used: MB200-255

1.2.5.5 Parameterization of File Accesses via Handles

The COM-driver allows full access to all drives of the CP and supports access to directories. For every file access a drive and/or directory name can be specified and the same rules are valid, as known from MS-DOS.

To access files under MS-DOS numbers, the so called handles are used. The amount of available handles and herewith the maximum number of open files at the same time is specified by the entry FILES = n in the CONFIG.SYS. We recommend to set the FILES-parameter at a minimum of 20, better however to 25 or 30.

By means of handles not only files on a mass storage, like hard disk, RAM disk or disk can be accessed. MS-DOS offers the opportunity to access via handles so called devices as well, like printer and serial interfaces. For a series of devices standard handles have already been defined. These devices need not to be opened before they are accessed. By direct read or write functions data can be read or output. Thus the PLC can in easy way directly access printer, screen and keyboard of the CP.

Predefined Standard-Handles

Handle	Device	access mode
00	standard-input (keyboard),	read only
01	standard-output (screen),	write only
02	standard-error (screen too),	write only
03	standard-auxiliary (V24-interface),	read and write
04	standard-printer (auxiliary),	write only

1.2.5.6 File Names

Like already mentioned, file names can be specified as well with drive and/or path specification. The CP software does not affect this. Directories are to be assigned, as usual, with a backslash (ASCII-Code 92, 5C hex). The character can also be entered correctly with the PG. The maximum length of a path specification is up to 64 characters. A file specification can consist of a maximum of 2 characters for the disk drive specification, up to 64 characters for the path specification as well as 8 characters for the file name and 3 characters for the extension, at the whole up to 78 characters.

Ist also possible to use a fixed form for a file name, consisting of 8 characters, a point, and 3 characters for the extension. If the file name is shorter than 8 characters or the extension shorter than 3 characters, it is possible to preset the positions not used with blanks (ASCII-Code 32, 20 hex).

For MS-DOS system functions, which need a file name as parameter, the file name must be closed up with the character ASCII zero (ASCII-code 0). For this reason the file name should be terminated with character ASCII-zero when a file name is passed from the PLC to the CP. If the file name is odd-numbered, this is unconditionally required. As the handling modules can enter only a block of words into a bank, it is necessary to fill up the file name with ASCII zero to an even-numbered length to achieve a block of words. If a file name is specified without terminating ASCII zero, the CP software completes the missing zero.

1.2.6 Function Description

1.2.6.1 Reset All Disk Drives (Disk Reset)

This functon enables to store all modified and non-saved file buffers physically to the drives.

Parameterization of FB3: F-Nr 13 ($0D hex)

1.2.6.2 Select Disk

Parameterization of FB3: F-Nr 14 ($0E hex)
DOSP Number of requested disk drive

Drive number	
0	A:
1	B:
2	C:

For this function drive number 0 corresponds to drive A:, composite to other functions like "get current directory".

1.2.6.3 Get Disk

The function outputs the number of the current (default) disk drive. The corresponding disk drive character "A", "C", ... is filed to byte 6.

Parameterization of FB5: F-Nr 25 ($19 hex)

Parameterization of FB6: ZT/N no. of DB for disk drive data
ZANF position of data word in DB
ZLAE length of drive data in the DB in words (1)

Content of DB: DW1 A-NR F-Nr
DW2 number of words being read
DW3 drive character drive number

drive character	drive number
A:	0
B:	1
C:	2

For this function drive number 0 corresponds to drive A:, composite to other functions like "get current directory".

1.2.6.4 Create Directory

Parameterization of FB3:	F-Nr	57 ($39 hex)
	QT/N	no. of DB with directory name
	QANF	position of directory name in the DB
	QLAE	length of directory name in the DB in words
Content of DB:	DW1	directory name
	DW2	...
	DW3	...
	...	

Note:
The directory name must be terminated with 0-byte, if it is of odd-numbered length (is not necessary for even-numbered length).

1.2.6.5 Delete Directory

Parameterization of FB3:	F-Nr	58 ($3A hex)
	QT/N	no. of the DB with directory name
	QANF	position of directory name in DB
	QLAE	length of directory name in the DB in words
Content of DB:	DW1	directory name
	DW2	...
	DW3	...
	...	

Note:
The directory name must be terminated with 0-byte, if it is of odd-numbered length (is not necessary for even-numbered length).

Specifying a disk drive in the directory name allows to delete a directory also in a not logged-on drive.

This function is finished with an error if the specified directory is the current directory or if the specified directory contains files.

1.2.6.6 Set Current Directory

Parameterization of FB3:	F-Nr	59 ($3B hex)
	QT/N	no. of DB with the directory name
	QANF	position of directory name in DB
	QLAE	length of directory name in the DB in words
Content of DB:	DW1	directory name
	DW2	...
	DW3	...
	...	

Note:
The directory name must be terminated with 0-byte, if it has an odd-numbered length (is not necessary for even-numbered length).

Regard that the current drive cannot be changed by means of this function. The directory can be, of course, affixed with a drive specification, but the current directory remains adjusted on the previous value on the drive logged-on. Only when the directed drive is accessed e.g. by the function "Select Disk", then the required drive is set.

1.2.6.7 Get Current Directory

Parameterization of FB5:	F-Nr	71 ($47 hex)
	DOSP	disk drive number
Parameterization of FB6:	ZT/N	no. of DB with the directory name
	ZANF	position of directory name in DB
	ZLAE	length of directory name in the DB in words
Content of DB:	DW1	A-NR F-Nr
	DW2	number of words being read
	DW3	directory name
	DW4	...
	...	

1.2.6.8 Create File/Rewrite Existing File

Parameterization of FB3:	F-Nr	60 ($3C hex)
	DOSP	attribute of new file
	QT/N	no. of DB with the file name
	QANF	position of file name in DB
	QLAE	length of file name in the DB in words
Content of DB:	DW1	file name
	DW2	...
	DW3	...
	...	
Parameter:		
Attribute:	00	normal
	01	read-only
	02	hidden
	04	system
	File attributes can be added up:	
	e.g. attribute 03 => file is read-only and hidden.	
Return of FB3:	DOSP Handle of the new file	

Note:
Does a file with the specified name already exist, then it is cut to zero length, i.e. all present data are deleted.

1.2.6.9 Create New File

Parameterization of FB3:	F-Nr	90 ($5A hex)
	DOSP	attribute of the new file
	QT/N	no. of DB with the file name
	QANF	position of file name in DB
	QLAE	length of file name in the DB in words
Content of DB:	DW1	file name
	DW2	...
	DW3	...
	...	
Parameter:		
Attribute:	00	normal
	01	read-only
	02	hidden
	04	system
	File attributes can be added up:	
	e.g. attribute 03 => file is read-only and hidden.	
Return of FB3:	DOSP	Handle of the new file

Note:
Does a file with the specified name already exist, then it is cut to zero length, i.e. all present data are deleted.

1.2.6.10 Open File

Parameterization of FB3:	F-Nr	61 ($3D hex)
	DOSP	access mode
	QT/N	no. of DB with file name
	QANF	position of file name in DB
	QLAE	length of file name in DB in words
Content of DB:	DW1	file name
	DW2	...
	DW3	...
	...	
Parameter:		
Access mode:	00	open file for reading
	01	open file for writing
	02	open file for reading and writing

Return of FB3: DOSP handle of the new file

Note:
After opening the file, the access mode can no more be changed, just after closing and renewed opening it is possible to apply another access mode. Net accesses are possible for this function but are not taken into account.
(SHARE.EXE must be loaded)

1.2.6.11 Write Physically a File to Disk (Commit File)

This function ensures a physical transfer of all modified internal data buffers of a CP file to the drive and updating of date and time of the last modification in the directory and updating of the file size. This function is equivalent to file closing and renewed opening.

Parameterization of FB3: F-Nr 104 ($68 hex)
DOSP handle number of file to be written

This function does not transfer any data from the PLC to the CP.

1.2.6.12 Close File

Parameterization of FB3: F-Nr 62 ($3E hex)
DOSP handle number of file to be closed

This function does not transfer any data from the PLC to the CP.

1.2.6.13 Delete File

This function deletes a file on a CP drive. The file needs not to be open before deletion. It is even possible to delete a file without error message which is open somewhere else and is still processed. The user has to take care that no files being in the moment accessed are deleted. This task is realized for networks by the network management software.

Parameterization of FB3:	F-Nr	65 ($41 hex)
	QT/N	no. of DB with file name
	QANF	position of file name in DB
	QLAE	length of file name in the DB in words
Content of DB:	DW1	file name
	DW2	...
	DW3	...
	...	

1.2.6.14 Rename File

Parameterization of FB3:	F-Nr	86 ($56 hex)
	QT/N	no. of DB with file name
	QANF	position of file names in DB
	QLAE	length of file names in the DB in words
Content of DB:	DW1	original file name, zero character, new
	DW2	file name
	DW3	...
	...	

The file must not be opened before renaming.

As data both file names are to be transferred connected, first the original file name and then the new file name. Both names must be separated by at least an ASCII-zero character. As data length must be defined the length of both file names including all zero characters.

This function can be used to move a file into another directory (move file). Therefore, only the name of the required target directory must be specified in the new file name.

Note:

Regard that moving a file is possible only within a disk drive.

1.2.6.15 Set File Pointer

Parameterization of FB3:	F-Nr	66 ($42 hex)
	DOSP	POS (high-order byte), handle (low-order byte)
	QT/N	no. of DB with file pointer
	QANF	position of file pointer in DB
	QLAE	length of data record (2 words)
Content of DB:	DW1	high-order word of file pointer
	DW2	low-order word of file pointer
Parameter:		
POS:	0	abs. position of file start
	1	rel. position from current position (signed)
	2	rel. position from file end (signed)

Note:

Note that the value of the file pointer is always to be regarded as the specification of a byte-position.

Length of a file can be detected by means of this function if 02 is entered as function code and 0 as new relative position of file end. Finally, the position being at the same time the number of data can be achieved via "get file pointer".

1.2.6.16 Get File Pointer

Parameterization of FB5:	F-Nr	194 ($C2 hex)
	DOSP	handle
Parameterization of FB6:	ZT/N	no. of DB for required data
	ZANF	target position in DB
	ZLAE	2
Content of DB:	DW1	A-NR F-Nr
	DW2	number of words being read
	DW3	high-order word of file pointer
	DW4	low-order word of file pointer

The file pointer is returned again as a doubleword. Thus, the digit 2 is also to be specified as number of data.

Note

The value of the file pointer is always to be regarded as the specification of a byte-position.

1.2.6.17 Read File or Device

Parameterization of FB5:	F-Nr	63 ($3F hex)
	DOSP	handle of the file
Parameterization of FB6:	ZT/N	no. of DB for data to be read
	ZANF	target position in DB
	ZLAE	number of data words to be read (2)
Content of DB:	DW1	A-NR F-Nr
	DW2	number of words being read
	DW3	data word 1
	DW4	data word 2
	...	

The number of words to be read from the file is not allowed to be higher than 504, otherwise the function is aborted with errors.

An exchange of bytes in a data word or doubleword is not provided in this function. All data are transferred unchanged from the CP to the PLC. In most of the cases it is delt with ASCII-files where an exchange is proved anyway to be not necessary. If required, the exchange must be done on the CP or PLC side, depending on the demands.

1.2.6.18 Write File or Device

Parameterization of FB3:	F-Nr	64 ($40 hex)
	DOSP	handle of file
	QT/N	no. of DB with data to be written
	QANF	position of data in DB
	QLAE	length of data record to be written in words

The number of words to read from the file is not allowed to be higher than 504, otherwise the function is aborted with errors.

An exchange of bytes in a data word or doubleword is not provided in this function. All data are transferred unchanged from the PLC to the CP. In most of the cases it is delt with ASCII-files where an exchange is proved anyway to be not necessary. If required, the exchange must be done on the CP or PLC side, depending on the demands.

If the function was terminated without errors but the written number is lower than the required, then a partial write error has probably occurred during the execution, or the character ^Z ASCII-code 26, 1A hex has been written to a character device (standard output).

1.2.6.19 Get Date

Parameterization of FB5:	F-Nr	42 ($2A hex)
Parameterization of FB6:	ZT/N	no. of DB for the date to be read
	ZANF	target position in DB
	ZLAE	number of data words to be read (3)

Content of DB:	DW1	A-NR	F-Nr
	DW2	number of words being read	
	DW3	year	
	DW4	month	day
	DW4	week-day	-----

Parameter:

Year	1980 ... 2099
Month	1 ... 12
Day	1 ... 31
Week-day	0 ... 6, (0=Sunday, 1= Monday, ...)

1.2.6.20 Get Time

Parameterization of FB5:	F-Nr	44 ($2C hex)
Parameterization of FB6:	ZT/N	no. of DB for the time to be read
	ZANF	target position in DB
	ZLAE	number of data words to be read (2)

Content of DB:	DW1	A-NR	F-Nr
	DW2	number of words being read	
	DW3	hour	minutes
	DW4	seconds	hundredth seconds

Parameter:

Hour	0 ... 23
Minute	0 ... 59
Second	0 ... 59
Hundredth second	0 ... 99

The function returns after an error-free termination two words as number of data. All values are to be interpreted as bytes.

1.2.6.21 Program Execute

Direct commands can be passed to the CP via this function.

Parameterization of FB3:	F-Nr	75 ($4B hex)
	QT/N	no. of DB with the MS-DOS-command line
	QANF	position of command line in DB
	QLAE	length of command line in DB in words

MS-DOS is no Multi-Tasking operating system enabling concurrent execution of several programs. As a rule, the main memory of a personal computer is too much limited as to load a series of resident programs with extensive data areas. This function can be called in the moment only if no other program is running on the CP apart from the COM-driver and other resident utilities. The COM-driver CP386COM.EXE must be started hereto in non-resident operation (option /NOTSR when calling CP386COM.EXE).

This function is finished when the called program is terminated with or without errors. Consequently, the bank stays disabled for other jobs during the program run. This must be considered when calling another function!

1.2.6.22 Get MS-DOS Version

Parameterization of FB5:	F-Nr	48 ($30 hex)
Parameterization of FB6:	ZT/N	no. of DB for data to be read
	ZANF	target position in DB
	ZLAE	number of data words to be read (3)

Content of DB:	DW1	A-NR F-Nr
	DW2	number of words being read
	DW3	main number subnumber
	DW4	OEM-number user number
	DW5	serial number

The function returns 3 words as data number after an error-free completion. The version main number is entered to data byte 0, version subnumber to data byte 1 and the OEM identification is entered to data byte 2. The bytes 3 and 5 return a 24 bit application serial number. Thereof the highest-order byte is filed to byte 3 and the low-order bytes to byte 4 and 5.

1.2.6.23 Get Detailed Error Information

Parameterization of FB5:	F-Nr	89 ($59 hex)
Parameterization of FB6:	ZT/N	no. of the DB for data to be read
	ZANF	target position in DB
	ZLAE	number of data words to be read (3)

Content of DB:	DW1	A-NR	F-Nr
	DW2	number of words being read	
	DW3	error code	
	DW4	error class	remedy
	DW5	error location	------

The function outputs MS-DOS error codes in the data record after an error-free completion. Error code is entered in data byte 0 and 1, in data byte 2 the error class, in data byte 3 the remedy and in data byte 4 the error position (see table).

Table with error codes:

01	invalid function number
02	file not found
03	path (directory) not found
04	too many open files, remedy: increase number of files in CONFIG.SYS
05	access refused attempt to modify a write-protected file
06	invalid handle, there is no opened file for the specified handle
07	memory control blocks destroyed MS-DOS inoperable, system must be rebooted
08	no memory existent
09	invalid memory control block
10 (0Ah)	invalid environment
11 (0Bh)	invalid program format program is incorrect structured or file has no program
12 (0Ch)	invalid access code, wrong access mode input when opening file
13 (0Dh)	invalid data
14 (0Eh)	invalid unit
15 (0Fh)	invalid disk drive, a non-existing disk drive has been responded
16 (10h)	invalid command
17 (11h)	not the same device
18 (12h)	no more files can be created, directory is full
19 (13h)	disk is write protected
20 (14h)	unknown device
21 (15h)	disk drive not ready. No disk inserted
22 (16h)	unknwn command
23 (17h)	data error (CRC-error) checksum of disk/hard disk sector wrong; sector probably defect
24 (18h)	length of request structure wrong
25 (19h)	seek error, positioning error, file pointer was positioned beyond end of file
26 (1Ah)	unknown media type, (disk is not in MS-DOS format)

27 (1Bh)	sector not found
28 (1Ch)	printer reports paper out
29 (1Dh)	write error
30 (1Eh)	read error
31 (1Fh)	general error
32 (20h)	file sharing violation
33 (21h)	file locking violation
34 (22h)	invalid disk change
35 (23h)	FCB not available
36 (24h)	file sharing buffer overflow
80 (50h)	file already exists
82 (52h)	directory cannot be created
83 (53h)	Int 24 error (handling of critical errors)
112 (70h)	size error, invalid number of data (e.g. trial to read or write more than 504 words)
113 (71h)	time exceeded during communication. For about 10 sec the CP was unable to access the bank.

Code table for error classes:

01	no resources available (memory or handles)
02	no error, but actual status (disabled region in a file), which is expected to disappear.
03	authorisation problem
04	internal error in system software
05	hardware error
06	system software error, no error of active process (as missing configuration files)
07	application program error
08	file or element not found
09	file or element has a faulty type or format
0A	file or element access disabled
0B	wrong disk in disk drive, faulty data sectors or error of storage medium
0C	other error

Code table for recommended measures

01	function repeat several times.Then ask user whether to abort or to ignore the error.
02	function repeat several times time-delayed between single attempts. Then ask the user whether to abort or to ignore the error.
03	correct information by user input (usually caused by invalid file name or disk drive specification).
04	abort application correctly (close opened files, disable file locking)
05	stop application immediately without 'ordering'.
06	ignore error
07	repeat after the user is prompted to correct the error.

Code table for error locations

01	unknown
02	block device or disk drive emulation (RAM disk)
03	network
04	serial device
05	memory

1.2.6.24 General Interrupt

The function enables to call general interrupts of the CP, e.g. VGA-BIOS-interrupts, keyboard interrupt, mouse interrupt etc. Because of various parameterization opportunities it is not possible to supply all registers with parameters. Four data words are passed to this function which are loaded correspondingly in registers AX, BX, CD and DX. The interrupt number is to be stored to the DOSP-parameter. All interrupt numbers are permitted.

After the function is executed, the value returned to register AX is entered in DOSP-parameter.

Parameterization of FB3:	F-Nr	255 ($FF hex)
	DOSP	interrupt number
	QT/N	no. of DB with data to be written
	QANF	data position in DB
	QLAE	length of data record to be written in words
DB content:	DW1	data word for register AX
	DW2	data word for register BX
	DW3	data word for register CX
	DW4	data word for register DX
Return of FB3:	DOSP	data word from register AX

This call should only be used by experienced DOS-programmers because this call enables arbitrary DOS-accesses.

1.3 CP-Jobs for PLC (Functions for Bank 2, 3 and 7)

1.3.1 Overview

The driver program CP386COM serves jobs initiated by the PLC, as well as jobs initiated by the CP386. The driver supplies a series of functions for the banks 2, 3 and 7. With these functions data can be read from the PLC or written to the PLC respectively from a running application on the CP. All functions are called by means of software interrupt 78h.

Tranfer and return of parameters is realized exclusively in the processor registers when calling the interrupt. Register assignment is included in the description of functions. Interfaces are implemented for Turbo-Pascal, Turbo-C and C++, as well as Microsoft-C. Also functions for reading and writing of data to the PLC, status call and abort functions are realized. Functios are differed by exchanging single elements and data blocks when data are transmitted. Functios are handled as "jobs". When calling a function, a job number is returned which is used to call the processing status. Up to 127 jobs can processed at the same time in each of banks 2 and 3.

Following functions are available:

Bank used	Function number	Function
none	$00	status call
2	$21	read a single element from the PLC
2	$21	read a block from the PLC
2	$20	status call of read job
2	$28	abort all read jobs
3	$31	write a single element into the PLC
3	$31	write a block into the PLC
3	$30	status call of write job
3	$38	abort all write jobs
7	$70	status call for process image
7	$71	read a process image area

1.3.2 Installation of Bank Software for Linking PLC and CP

1.3.2.1 PLC Side: Handling Modules

Handling modules FB1 and FB2 have to be loaded in the PLC to enable communication with the CP. Handling module FB1 is called up in OB1 and handling module FB2 in the restart modules (OB21 and OB22).

Example for calling up FB1 in OB1:

```
        Module#OB1
        BIB
0000            ;SPA FB 1
        NAME    #CP-L/S
        ANSS    =KY 2,32
        PAA     =KF +1
        PAFE    =MB 99
0005            ;BE
```

Transfer parameters:

ANSS: KY AN Number of jobs to be at most processed on the bank when calling up a handling module

SS Number of basic bank

PAA: KF Update ident of process images on the bank when calling up the handling module

≠ 0 Process images are updated

= 0 Process images are **not** updated

PAFE: MB Error acknowledgement message of handling module

= 0 no error occurred

≠ 0 error occurred. Error number is sent in PAFE-byte

Error number:

1 Number of jobs to be at most processed when calling up a handling module is 0.

2 Number of jobs to be at most processed when calling up a handling module is higher than 127.

3 Basic bank number is not divisible by 8

5 Bank is not synchronized yet by CP.

6 For a block job being the first call, no further job is allowed to be in the bank.

7 A further block job is only allowed to be the first job in the bank.

Scratch pads used: MB200-MB255

Example for calling up FB2 in OB21:

```
        Module#OB21
        BIB
0000            ;SPA FB 2
        NAME    #SYNCHRON
        SSNR    =KY +32
        WART    =KF +0
        PAFE    =MB 98
0005            ;BE
```

Transfer parameters:

SSNR: KF Number of basic bank

WART:
- = 0 FB-SYNCHRON does not wait until every single bank is synchronized by CP
- ≠ 0 FB-SYNCHRON waits at every single bank until the CP has synchronized this bank

PAFE: BY Error acknowledgement message of handling module
- = 0 no error occurred
- ≠ 0 error occurred:
- 3 basic bank number is not divisible by 8.

Scratch pads used: MB200-MB255

1.3.3 Driver Functions via Software Interrupt

1.3.3.1 CP Status Call

This function outputs various arbitrary status information via the CP. Moreover, it can be used by the applications software to check whether the CP software driver is loaded. Further information returned is the output status of hard and software, CPU identification etc.

Register	**IN**	high	**OUT**	low
AX	$00	$C386		
BX		VGA-Bios	Bios	
CX		CP	CPU	
DX		CP-Status	PLC-Status	

AX C386 hex code for CP software is loaded
BH output status of CP VGA-BIOS
BL output status of CP-BIOS
CH output status of driver software (CP)
CL code of CPU in PLC (valid if banks are synchronised)
DH CP status register (IO-address 280h)
DL PLC status register (IO-address 281h)

Codes of the output statuses (version numbers) for BIOS, VGA and driver are BCD-coded in one byte. That is, the value 10 (hex) is equivalent to version 1.0; 15 (hex) is equivalent to version number 1.5 and 1A (hex) is equivalent to version 1.10.

This function does not execute any initializations in the banks.

1.3.3.2 Read a Single Element from the PLC

With this function a single data type (bit, byte, word,...) can be read from the PLC. The function is just starting the job and does not wait for the PLC to execute it, but returns immediately, to where it was called. Therefore the data can be read not before the function 'status call' was executed.

Register	high	**In**	low	**Out**
AX	$21	typ		status
BX	size	bst		
CX	adr			
DX	-	bit		

typ element area (type) of data for single element in PLC (DB, MB, ...)
size code of data size (bit, byte, ...)
bst module number, is to be set only for data area (type) DB or DX
if data area (type) is absolute, the higher order bits of adr are here.
adr initial address in area
bit Bit-number if element size is bit or semaphore.

status < 0 error code because of an error
error numbers of PLC are summed up with FFF0h
1..127 job number to call status

Note:
This function does not return data! If the job status is 'finished without error', the data can be passed by calling the function status call.

To ensure, that a finished reading job will not be overwritten with a new job before data are passed, the job is blocked. After starting a job, its status is to be checked as long as the job is finished with or without error. If the status of the job is 'finished without error', data will be copied to the CP with the address specified. If no status call is executed, the job remains blocked, and possibly no further read jobs can be started, even if all jobs in the bank are finished.

1.3.3.3 Read a Block from the PLC

With this function a whole block of data can be read from the PLC. The function is just starting the job, and does not wait until the PLC is executing it, but returns immediately to where it was called. Therefore the data can be read not before the function 'status call' was executed.

Register	high	**In**	low	**Out**
AX	$21	typ		status
BX	size	bst		
CX	adr			
DX	len			

typ element area (type) of data for single element in the PLC (DB, MB, ...)
size data size of block data (ident. whether single bytes are to be exchanged)
0F(hex) data block of bytes (no exchange)
1F(hex) data block of words (exchange of high- and low-byte)
2F(hex) data block of doublewords (exchange of all 4 bytes)
bst module number, is to be set for element areas (type) DB, DX , FX ...
if element area (type) is absolute, then here are high-order bits of adr.
adr initial address in the area
len number of data in words !! (also if size is byte or doubleword)

status < 0 error number because of error
= 0 job number to call status

Note:
For reasons of executing an automatical adjustment of data during the transmission, the type of the data in a block (bytes, words, doublewords) has to be specified. One block can only contain data of the same type. Concernig words and doublewords for every single data an exchange of the bytes is executed correspondingly.

This function does not return data! If the job status is 'finished without error', the data can be passed by calling the function status call.

A block read job can only be started, if bank 2 is empty, that means, no variable read jobs and no block read job may be in processing status.

To enable, that a finished reading job will not be overwritten with a new job before data are passed, the job is blocked. After starting a job, its status is to be checked as long as the job is finished with or without error. If the status of the job is 'finished without error', data will be copied to the CP with the address specified. If no status call is executed, the job remains blocked, and possibly no further read jobs can be started.

1.3.3.4 Write a Variable to the PLC

This function enables to write single data (bit, byte, word,...) to the memory of the PLC. When calling, the address of a variable to be written must be specified. The function transmits its value to the bank and does not wait for the PLC to read the data, but returns immediately to where it was called.

Register	high	**In**	low	**Out**
AX	$31	typ		status
BX	size	bst		
CX	adr			
DX	-	bit		
SI	offset			
DS	segment			

typ	element area (type) of data for single element in PLC (DB, MB, ...)	
size	code of data size (Bit, Byte, ...)	
bst	module number, is to be set only for element area (type) DB or DX if element area (type) is absolute, here are the high-order bits of adr.	
adr	initial address in area	
bit	bit number if data size is bit or semaphor.	
offset	offset of variable address (in CP)	
segment	segment of variable address (in CP)	
status	< 0	error number because of error
	129-255	job number to call status

Note:
Different to read jobs, a write job is not being blocked, nevertheless the job status as well should be as long called as the job is finished with or without error.

Depending on the element size, the pointer to the data in the CP is to interpreted differently:

bit or semaphore

Pointer is the address of a byte, the bit is read by bit number 0.

Byte, left byte, right byte

The pointer is the address of a byte. The byte is read from the memory cell.

Word

The pointer is the address of a word. High- and low-byte are exchanged at the transmission.

Doubleword/extended word

The pointer is the address of a extended word, the extended word is read and the order of all 4 bytes is reverse.

1.3.3.5 Write a Block to the PLC

With this functions a whole data block can be transmitted to the PLC. When calling, a pointer to the data block is to be specified. The function writes the data to the bank and returns immediately to where it was called. There is no delay for the PLC to read the data. Subsequently the data block is competely available in the CP and could be overwritten for example.

Register	high	**In**	low	**Out**
AX	$31	typ		status
BX	size	bst		
CX	adr			
DX	len			
SI	offset			
DS	segment			

typ	data type of data for block element in PLC (DB, MB, ...)
size	data size of block data
	0F(hex) data block of bytes
	1F(hex) data block of words
	2F(hex) data block of doublewords
bst	module number where DB, DX, FB is relevant,
	contains at typ = absolute the high-bits of adr
adr	initial address in the area
len	number of data in words !!
offset	offset of block address (in CP)
segment	segment of block address (in CP)
status	< 0 error number because of error
	128 job number to call status

Note:
For reasons of executing an automatical adjustment of data during the transmission, the type of the data in a block (bytes, words, doublewords) has to be specified. One block can only contain data of the same type. Concernig words and doublewords for every single data, an exchange of the bytes is executed correspondingly.

A block write job can only be started, if bank 3 is empty, that means, no variable write jobs and no block write job may be in processing status.

A block write job is blocking the bank. After starting a job, its status is to be checked as long as the job is finished with or without error. If the status of the job is 'finished without error', data will be copied to the PLC with the address specified. If no status call is executed, the job remains blocked, and possibly no further read jobs can be started.

1.3.3.6 Read Job Status

With this function, the status of an earlier started job can be called. For read jobs, variable and block read jobs, this function copies data to a specified address to the CP, if the job status is 'finished without error'.

Register	high	**In**	low	**Out**
AX	$20/$30	a_nr		status
SI	offset			
DS	segment			

fn function number for status call
$20 for read jobs
$30 for write jobs

a_nr job number

offset offset of data address (in CP)
to be specified only for read jobs (variables and block).

segment segment of data address (in CP)
to be specified only for read jobs (variables and block).

status
- < 0 job finished with error
  error messages of PLC are added with FF00h.
- 1 job still in process
- 2 job status not defined
- 3 job finished without error

The following procedure is advisable for the status call:

- If the job is still "in processing", the status function has to be called as long as the status changes.
- Concerning write jobs (bank 3): if the job was "finished without error", then the data were written to the PLC. For block elements, the bank has been released.
- Concerning read jobs (bank 2): If the job is 'finished without error', and a pointer was specified for the data, the data will be copied to the specified address in the CP. Depending on the specified data size, an exchange of bytes will be executed, if neccessary. The job block will be released, in order to enable the execution of new jobs. If the address NULL (0:0) is specified as a pointer, no data will be copied, but the job will be released as well.
- If the job status is 'not defined' the job was already finished earlier, but was not overwritten by a new job. If this job was a read job and a pointer unequal to NULL was specified, the data will be copied to the specified destination address.
- If the job is "finished with error", then the job block was enabled if a read job or a block job is concerned.

- For a read job for single variables, the pointer is to be interpreted differently, depending on the element size:
  - Bit or semaphore

    The pointer is the address of a byte, the bit will be written to bit number 0, the whole byte will be overwritten.
  - Byte, left byte, right byte

    The pointer is the address of a byte. The byte will be written to the storage cell.
  - Word

    The pointer is the address of a word. High and low-byte will be interchanged during transmission.
  - Doubleword/extended word:

    The pointer is the address of a exetended word, the order of all 4 bytes will be interchanged and be written to the extended word.

- For a read job for a data block, the pointer is to be interpreted differently, depending on the element size:
  - Byte:

    The pointer is the address of a block of bytes, the individual bytes will be transmitted unchanged from the PLC.
  - Word:

    The pointer is the address of a block of words. High and low byte will be interchanged for every individual word during transmission.
  - Doubleword/extended word:

    The pointer is the address of a block of exetended words. All 4 bytes of every individual exetended word will be interchanged during transmission.

1.3.3.7 Abort All Jobs of a Bank

Register	high	**In**	low	**Out**
AX	fn			status

fn function number for status call
$28 abort all read jobs
$38 abort all write jobs

status < 0 error number because error occurred
0 all jobs were aborted.

All not yet finished write or read jobs can be aborted by means of this function. It must not be differed between variable and block jobs. Also if there were no jobs active in the bank, the function answers with the return value 0.

1.3.3.8 Read Status of Process Image

Register	high	**In**	low	**Out**
AX	$70	-		status

status 0 no process image available
1..255 current process image counter

The current value of the process image counter (bank 7 address 3FEh) can be read by means of this function. If the value is 0 then no process image is available.

1.3.3.9 Read Area of Process Image

Register	high	**In**	low	**Out**
AX	$71	typr		status
BX	adr			
CX	len			
SI	offset			
DS	segment			

typ	data type of process image (EB, MB)
adr	initial address in area
len	number of data in bytes or words (depending on TYPE)!!
offset	offset of data address (in CP)
segment	segment of data address (in CP)

status	< 0	error number because of error
	= 0	process image not available
	> 0	counter for process image (as for status function)

This function is used to read an area of the current process image. When accessing single areas, then the length of the area is supervised, e.g. cannot be read from EB126 with the length of 4 bytes because only 128 byte EB are available.

The length is input in words for timer and counter access, and in bytes for all other types. For timer and counter the high- and low byte of every word is exchanged also at the transfer, so that the data in the CP can be correctly processed as words.

By setting the type ABSOLUT, an optional process image sector can be read, also affecting other areas. The length is given in bytes, also if it is read from timer or counter range. If a range of timer or counter is read, then high- and low byte is changed again, too !!

06	counter (length in words)
07	timer (length in words)
08	marker (length in bytes)
09	EB (length in bytes)
0A	AB (length in bytes)
0F	absolute access to process image (length in bytes)

1.3.3.10 Error Numbers of CP for Banks 2, 3 and 7

hex	dec.	description
FFFF	-1	invalid data type
FFFE	-2	length error (e.g. address too big, bit number too high)
FFFD	-3	invalid data size (wrong value at single or block job)
FFFC	-4	data type for this CPU not possible
FFFB	-5	bank full, 127 single jobs in the bank, or at least 1 single element- and a block job are to be started.
FFFA	-6	bank access disabled for 10 sec (PLC stop probably)
FFF9	-7	job/bank is still blocked (job status was not checked in order to de-block the job).
FFF8	-8	wrong job number for status function (e.g. job no > 255)
FFF7	-9	faulty source data pointer (address NULL was entered in write job)
FFF6	-10	job not in process (not used !!!!)
FF	255	invalid function call
EE	238	job stopped during initialization

1.3.4 Interface for Turbo-Pascal (from Version 4.0)

To facilitate calling functions of COM-driver from Pascal programs, a Turbo-Pascal-Unit has been created which makes available all functions of the service interrupt INT 78 to be easy called. For every driver function an adequate Pascal-procedure is defined which supplies registers, calls interrupts and returns values. Thus, also users being not familiar with system-oriented programming on CP, are able to utilize fully all driver feasibilities.

All required functions, data types and constants are included in the Unit CP386LIB. This has to be entered with "USES CP386Lib" into the application program if it should be used in a Pascal-program. It must be ensured that the compiled Unit CP386LIB.TPU is contained in the directory where Turbo-Pascal traces for units. Setting occurs via the menu items "Options| Directories| EXE & TPU-directory" (cf. Manual or Help-Functions for Turbo-Pascal).

Following sections show only a survey of the individual functions. For a detailed description including all important information see function descriptions of the previously described sections.

1.3.4.1 Function CP-Status Call

FUNCTION **CP_Info** (AR inforec : CP386InfoRec) : INTEGER;

Data structures:

```
TYPE CP386InfoRec =
       RECORD
         CP_id : WORD;                (* identification: CP386 value= $C386 *)
         VGA_ver, BIOS_ver : BYTE;    (* version numbers: VGA-BIOS and BIOS *)
         DRV_ver: BYTE;               (* identification: software version *)
         CPU_AG : BYTE;               (* identification CPU in PLC *)
         CP_reg, S5_reg : BYTE;       (* CP- and PLC-status register *)
       END;
```

Data structure for the general status info function and the components are defined corresponding to CP values.

This function calls the driver function "general status information", but previously it is checked whether the driver is installed. If not, the function returns the value -1. If the COM-driver is installed, value 0 is returned and the components of info-structure inforec are set adequate. The component CP_id is always identified with the value $C386.

1.3.4.2 Read a Single Element from the PLC

FUNCTION **CP_read**_AG(size, typ, bst : BYTE; adr : longint; bit : BYTE) : INTEGER;

size: data size of single elements
typ: data type for single elements
bst: module number
adr: address in the module or absolute address
bit: bit number

More details see chapter 1.3.4.10.

Return: job number or negative number if there is an error

This function calls the driver function "read a single element from the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
VAR   a_nr,                                    (* job number for read job *)
      stat : INTEGER;                          (* momentaneous job status *)
      wert : BYTE;                             (* value read from the PLC *)
BEGIN

(* start job *)
   a_nr := CP_read_AG(LBYTE_ELM, DB_SNG, 10, 1, 0);

   IF a_nr < 0                                       (* error occurred *)
   THEN WriteLn('job finished with error: ', a_nr);

   ELSE  BEGIN                                       (* a_nr contains job number
*)
      REPEAT
            stat := CP_stat_AG(a_nr, Addr(value)); (* job status/fetch data *)
      UNTIL stat <> REQ_WRKN;   (* as long as job is ready with or without
errors *)

      CASE stat  OF
        REQ_NO_ERR:     WriteLn('date: ', value, ' has been read');
        REQ_UNDEF:      WriteLn('job status nondefined,date: ',value, ' read');
        ELSE            WriteLn('job is ready with error: ', stat);
      END;

   END;
```

1.3.4.3 Read a Block from the PLC

FUNCTION **CP_readn_AG** (size, typ, bst:BYTE; adr:longint; len:WORD):integer;

size: data size of block elements
typ: data type of block elements
bst: module number
adr: address in module or absolute address
len: number of data in words

More details see chapter 1.3.4.10.

Return: job number or negative number if there is an error

This function calls the driver function "read a block from the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number 0 is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
VAR   a_nr,                                          (* job number for read job*)
      stat : INTEGER;                                (* momentaneous job status *)
      buff : ARRAY[1..50 ] OF INTEGER                (* data read from the PLC *)
BEGIN

(* start job *)
   a_nr := CP_readn_AG(W_BLOCK, DB_BLK, 5, 10, 100);

   IF a_nr < 0                                       (* error occurred *)
   THEN WriteLn('job finished with error: ', a_nr);

   ELSE  BEGIN                                       (* a_nr contains job number *)
      REPEAT
            stat := CP_stat_AG(a_nr, Addr(buf)); (* job status/fetch data *)
      UNTIL stat <> REQ_WRKN;  (* as long as job is ready with or without errors
*)

      CASE stat  OF
          REQ_NO_ERR:   WriteLn(buffer was read');
          REQ_UNDEF:    WriteLn('job status nondefined, buffer was read');
          ELSE           WriteLn('job is ready with error: ', stat);
      END;

   END;
```

1.3.4.4 Write a Single Element to the PLC

FUNCTION **CP_write_AG** (size, typ, bst : BYTE; adr: longint; bit: BYTE; p: POINTER)
: integer;

size:	data size	
typ:	data type single elements	
bst:	module number	
adr:	address in module or absolute address	
bit:	bit number	
p:	pointer to date to be written	
	for data type bit, semaphore or byte:	pointer to a byte
	for data type word:	pointer to a word
	for data type doubleword:	pointer to a doubleword

More details see chapter 1.3.4.10.

Return: job number or negative number if there is an error

This function calls the driver function "write a single element into the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number is returned as function value.

Recommended calling method

```
VAR   a_nr,                                    (* job number for read job*)
      stat : INTEGER;                          (* momentaneous job status *)
      wert : BYTE;                             (* value to be written *)
BEGIN
 ...
(* start job *)
   wert := $7E;
   a_nr := CP_write_AG(BYTE_ELM, DB_SNG, 10, 1, 0, Addr(value));

   IF a_nr < 0                                 (* error occurred *)
   THEN WriteLn('job finished with error: ', a_nr);

   ELSE  BEGIN                                 (* a_nr contains job number *)
      REPEAT
            stat := CP_stat_AG(a_nr, NIL);     (* read job status *)
      UNTIL stat <> REQ_WRKN;  (* as long as job is ready with or without errors
*)

      CASE stat OF
          REQ_NO_ERR:   WriteLn('date: ', value, ' was written');
          REQ_UNDEF:     WriteLn('job status nondefined.');
          ELSE           WriteLn('job is ready with error: ', stat);
      END;
   END;
```

1.3.4.5 Write a Block into the PLC

FUNCTION **CP_writen_AG** (size, typ, bst : BYTE; adr : longint; len : word;
p : POINTER) : integer;

size: data size of block elements
typ: data type block elements
bst: module number
adr: address in module or absolute address
len: number or data in words
p: pointer to the data block to be written

More details see chapter 1.3.4.10.

Return: job number or negative number if there was an error

This function calls the driver function "write a block into the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number $80 (hex) is returned as function value.

Recommended calling method

```
VAR   a_nr,                                        (* job number for read job*)
      stat : INTEGER;                              (* momentaneous job status *)
      i : INTEGER;
      buff : ARRAY[1..100 ] OF BYTE;               (* data to be written *)
BEGIN
 ...
   FOR i := 1 TO 100 DO
            buff[i] := i;                          (* preset data buffer *)
(* start job *)
   a_nr := CP_writen_AG(B_BLOCK, DB_BLK, 5, 10, 100, Addr(buff));

   IF a_nr < 0                                     (* error occurred *)
   THEN WriteLn('job finished with error: ', a_nr);

   ELSE  BEGIN                                     (* a_nr contains job number *)
      REPEAT
            stat := CP_stat_AG(a_nr, NIL);       (* read job status *)
      UNTIL stat <> REQ_WRKN;  (* as long as job is ready with or without errors
*)

      CASE stat OF
          REQ_NO_ERR:        WriteLn('buffer was written');
          REQ_UNDEF:         WriteLn('job status nondefined.');
          ELSE               WriteLn('job is ready with error: ', stat);
      END;
   END;
```

1.3.4.6 Read Job Status

FUNCTION **CP_stat_AG** (a_nr : INTEGER; p: POINTER): INTEGER;

a_nr: job number of the job to be tested
p: pointer to date or data block in CP-memory
(to be preset only for read jobs with single and block element)
for data type bit, semaphore or byte pointer to a byte
for data type word, pointer to a word
for data type doubleword, pointer to a doubleword
for data type block, pointer to buffer for data block

Return: job status or negative number if there is an error

This function calls the driver function "read job status". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If there was an error during the job execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job status (see chapter 1.3.4.10) is returned.

1.3.4.7 Abort all Jobs of a Bank

FUNCTION **CP_cncl_AG** (a_nr : BYTE): INTEGER;

a_nr: identification for bank 2 or 3
$00 abort all still active jobs of bank 2
$80 abort all still active jobs of bank 3

Return: 0 or negative number if there is an error

This function calls the driver function "abort all jobs of a bank". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If there was an error during the job execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, i.e. all jobs have been aborted, then 0 is returned.

1.3.4.8 Read Status of Process Image

FUNCTION **CP_stat_PA** : BYTE;

Return: process image counter

This function calls the driver function "status call process image". This function returns the process image counter.

1.3.4.9 Read Area of Process Image

FUNCTION **CP_read_PA** (typ : BYTE; adr, len : WORD; p : POINTER) : INTEGER;

typ: data type process image
adr: address in the area or absolute address
len: number of data (bytes or words) depending on the type
p: pointer to a data buffer in the storage

More details see chapter 1.3.4.10.

Return: process image counter

This function calls the driver function "read a process image area". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the current value of the process image counter is returned.

1.3.4.10 Constants of Turbo Pascal

Following constants are already predefined. It is recommended to use these constants also in the program text for reasons of clearness and better readability. Moreover, adaptations attended to possible later changes of the COM-driver can be carried out easier.

1.3.4.10.1 Predefined constants for data sizes

```
CONST
     BIT_ELM   0x00      = $00;    (* bit *)
     SEMA_ELM  0x01      = $01;    (* bit as semaphore *)
     BYTE_ELM  0x02      = $02;    (* byte *)
     LBYTE_ELM 0x02      = $02;    (* left byte of a word *)
     RBYTE_ELM 0x03      = $03;    (* right byte of a word *)
     WORD_ELM  0x04      = $04;    (* word *)
     DWORD_ELM 0x05      = $05;    (* doubleword *)
     BLOCK_ELM 0x07      = $07;    (* block *)
```

1.3.4.10.2 Predefined constants for data types for single elements

```
CONST
     DB_SNG 0x00         = $00;    (* DB *)
     DX_SNG 0x01         = $01;    (* DB in external memory *)
     BA_SNG 0x02         = $02;    (* BA *)
     BB_SNG 0x03         = $03;    (* BB *)
     BS_SNG 0x04         = $04;    (* BS *)
     BT_SNG 0x05         = $05;    (* BT *)
     Z_SNG  0x06         = $06;    (* counter *)
     T_SNG  0x07         = $07;    (* timer *)
     MB_SNG 0x08         = $08;    (* marker *)
     EB_SNG 0x09         = $09;    (* input area *)
     AB_SNG 0x0A         = $0A;    (* output area *)
     PB_SNG 0x0B         = $0B;    (* P-peripherals *)
     QB_SNG 0x0C         = $0C;    (* Q-peripherals *)
     ABS_SNG0x0F         = $0F;    (* absolute memory *)
```

1.3.4.10.3 Predefined constants for data types for block elements

```
CONST
     DB_BLK  0x00        = $00;    (* data module *)
     DX_BLK  0x01        = $01;    (* DB in external memory *)
     BA_BLK  0x02        = $02;    (* BA *)
     BB_BLK  0x03        = $03;    (* BB *)
     BS_BLK  0x04        = $04;    (* BS *)
     BT_BLK  0x05        = $05;    (* BT *)
     FB_BLK  0x06        = $06;    (* FB *)
     FX_BLK  0x07        = $07;    (* FB in external memory *)
     OB_BLK  0x08        = $08;    (* OB *)
     PB_BLK  0x09        = $09;    (* PB *)
     SB_BLK  0x0A        = $0A;    (* SB *)
     ABS_BLK 0x0F        = $0F;    (* absolute memory *)
```

1.3.4.10.4 Predefined constants for data type for block elements

```
CONST
     B_BLOCK  0x0F       = $07;    (* type: block with bytes  *)
     W_BLOCK  0x1F       = $17;    (* type: block with words  *)
     D_BLOCK  0x2F       = $27;    (* type: block with extended
                                   words *)
```

1.3.4.10.5 Identifications for job status

```
CONST
     REQ_WRKN 0x01       = $01;    (* job in processing *)
     REQ_UNDEF  0x02     = $02;    (* job status not defined *)
     REQ_NO_ERR  0x03    = $03;    (* job ready without errors *)
```

1.3.4.10.6 Predefined constants for data types for process image

```
CONST
     Z_PA  0x06          = $06;    (* counter*)
     T_PA  0x07          = $07;    (* timer *)
     MB_PA 0x08          = $08;    (* marker *)
     EB_PA 0x09          = $09;    (* input area *)
     AB_PA 0x0A          = $0A;    (* output area *)
     ABS_PA0x0F          = $01F    (* absolute block in PA *)
```

1.3.4.10.7 Predefined constants for error messages: bank 2, 3 and 7

```
CONST
```

ERR_S5_TYP = $01; (* invalid element type *)

With a single-element access with the element type DX_SNG, BA_SNG, BB_SNG, BT_SNG or QB_SNG or with a block element access with element type DX_BLK, BA_BLK, BB_BLK, BT_BLK or FX_BLK the programme tried to access data in a PLC of the type 115U. However, these element types do not exist in this PLC type.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

ERR_S5_BST = $02; (* module not available *)

With a single-element access with element type DB_SNG or with a block element type DB_BLK the programme tried to access a not existing module.

Correction: To create data block in the PLC or to correct parameter „bst“ in the function call of the CP-user software.

ERR_S5_ELM = $03; (* element not available *)

With a single-element access with element type DB_SNG or with a block element access with element type DB_BLK the programme tried to access data in a data block which are not available.

Correction: To extend the data block in the PLC correspondingly or to correct the parameter „adr“ or „len“ in the function call of the CP user software.

With a single-element access with element type Z_SNG or T_SNG the programme tried to access timer or counter with a number > 127.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

With a single-element access with element type MB_SNG the programme tried to access flags with a number > 199 with the size of element Byte, with number > 198 with the size of element word or with number > 196 with the size of ement douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

With a single-element access with element type EB_ - or AB_SNG the programme tried to access the process image of the I/O range with number > 127 with the element size Byte, with number > 126 with element size word or with number > 124 with element size douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

With a single-element access with element type PB_SNG the programme tried to access elements of the P-peripherals with number > 255 with element size Byte, with number > 254 with element size word or with number > 252 with element size douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

ERR_S5_SIZE = $04; (* invalid element size *)

With a single-element access with element type Z_SNG or T_SNG the programme tried to access timer or counter, whereas the parameter element size was not set to word access (WORD_ELM).

Correction: To correct the parameter „size“ in the function call of the CP user software.

With a single-element access with element type MB_SNG or ABS_SNG the programme tried to access flags or absolute addresses with the parameter element size RBYTE_ELM.

Correction: To correct the parameter „size“ in the function call of the CP user software.

With a single-element access with element type EB_SNG or AB_SNG the programme tried to access inputs or outputs in the process image with the parameter element size SEMA_ELM or RBYTE_ELM.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

With a single-element access with element type PB_SNG the programme tried to access the P-peripherals with the parameter element size BIT_ELM, SEMA_ELM or RBYTE_ELM.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

With a reading single-element access with element type ABS_SNG the programme tried to read absolute addresses with element size SEMA_ELM. This type of access is only possible in writing under absolute addressing! When single bits are to be read then the element size BIT_ELM has to be used.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

ERR_S5_BIT = $05; (* Bit-number too high *)

With a single-element access with element type MB_SNG or ABS_SNG and the element size BIT_ELM or SEMA_ELM the programme tried to access a flag bit or an absolute address bit with a bit number > 7 (15).

Correction: To correct the parameter „bit“ in the function call of the CP user software.

With a single-element access with element type EB_SNG or AB_SNG the programme tried to access an I/O-BIT with a bit number > 7.

Correction: To correct the parameter „bit“ in the function call of the CP user software.

ERR_S5_STRT = $06; (* invalid starting address *)

With a block element access with element type „module“_BLK the programme tried to transfer blocks via modules whereas the relative starting address in the block is> 32767.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

ERR_S5_LEN = $07; (* invalide block length *)

With a block element access under all element types the programme tried to transfer blocks with a length > 504.

Correction: To correct the parameter „len“ in the function call of the CP user software.

ERR_S5_ADR = $08; (* Address too big *)

With a single- or block element access with element type ABS_SNG the programme tried to addess an address > FFFFh in a PLC of the typee 115U. However, the CPUs (up to CPU 944) have an address range of only 64 KB.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

ERR_S5_QVZ = $09; (* QVZ/ADF in the PLC with reading/writing *)

The programme tried to access an address range which is physically not available.

The PLCs of the type 135 and 155 make this error message available. A PLC of the type 115U would be set to STOP in this case.

Correction: To correct the parameters „typ“ or „adr“ in the function call of the CP user software.

ERR_S5_944 = $0A; (* CPU 944: module in prog.bank *)

With a block element access with element type „module“_BLK the programme tried to access a module which is not in the data block. (This only concerns the CPU 944 form the PLC type 115U)

Correction: To create a module in the PLC in the data block bank (via BIB-Nr. 19285) or to correct the function call in the CP user software.

1.3.5 Interface to Turbo-C (2.0 and C++ from 1.0), Microsoft-C 6.0

To facilitate calling functions of COM-driver from C-programs, a library file has been created which makes available all functions of the service interrupt INT 78 to be easy called. For every driver function a respective C-function is defined which supplies registers, calls interrupts and returns values. Thus, also users being not familiar with system-oriented programming on CP, are able to utilize fully all driver feasibilities.

Data types and constants for element sizes, element types and error numbers as well as function prototypes of of functions in ANSI-C-style described in the following are defined in the Include File "CP386DEF.H". The Include-File must be quoted in the application program.

All required functions are implemented in the CP386LIB.C file. The CP386LIB.O file is also to be implemented if it is to be used in a program. Depending on the programming environment and version, the file is to be packed into the project file (Turbo-C) or Depencie List (Microsoft-C) or into the Make-File. For detailed information see the respective manuals.

Reference: in "CP386LIB.H" is byte defined as unsigned char word defined as unsigned short.

1.3.5.1 Function CP Status Call

Data structures:

```
typedef struct {
     word CP_id;         /* identification: CP386 value = $C386  */
     byte VGA_ver, BIOS_ver;  /* version numbers:(VGA)-BIOS      */
     byte DRV_ver;            /* identification:software version */
     byte CPU_AG;             /* identification CPU in PLC       */
     byte CP_reg, S5_reg;     /* CP- and PLC-status registers    */
     } CP386_InfoBlk;         /* info-block  */
```

Data structure for the general status info function, the components are preset according to values of the CP486.

1.3.5.2 Read a Single Element from the PLC

int CP_read_AG(byte size, byte typ, byte bst, unsigned long adr, byte bit);

size:	data size
typ:	data type single elements
bst:	module number
adr:	address in module or absolute address
bit:	bit number

More details see chapter 1.3.5.10.

Return: job number or negative number if there is an error

This function calls the driver function "read a single element from the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
int a_nr;                        /* job number for read job*/
int stat;                        /* momentaneous job status */
int time_count=4000;             /* timeout counter (4 seconds) */
          byte wert ;            /* value read from the PLC */
/* start job */
   a_nr = CP_read_AG(LBYTE_ELM, DB_SNG, 10, 1, 0);

   if(a_nr < 0)                  /* error occurred */
     printf("job finished with error: %d\n", a_nr);
   else {                        /* a_nr contains job number */
      do {
         stat = CP_stat_AG(a_nr, &wert);       /* job status/fetch data */
      } while((time_count > 0)&&(stat == REQ_WRKN));
/* as long as job is ready with or without errors*/
      switch(stat)  {
       case REQ_NO_ERR:
          printf("date: %d was read\n", wert);
            break;
       case REQ_UNDEF:
          printf("job status nondefined, date: %d read\n", wert);
            break;
       default:   printf("job is ready with error: %d\n", stat);
      }
   }
```

1.3.5.3 Read a Block from the PLC

int CP_readn_AG(byte size, byte typ, byte bst, unsigned long adr, word len);

size: data size of block elements
typ: data type block elements
bst: module number
adr: address in module or absolute address
len: number of data in words

More details see chapter 1.3.5.10.

Return: job number 0 or negative number if there is an error

This function calls the driver function "read a block from the PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number 0 is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
int a_nr;                              /* job number for read job*/
int stat;                              /* momentaneous job status */
int time_count=4000;                   /* timeout counter (4 seconds) */
int buff[100] ;                        /* value read from the PLC */

/* start job */
   a_nr = CP_readn_AG(W_BLOCK, DB_BLK, 5, 10, 100);

   if(a_nr < 0)                        /* error occurred */
     printf("job finished with error: %d\n", a_nr);

   else {                              /* a_nr contains job number */
       do {
          stat = CP_stat_AG(a_nr, &buff);     /* job status/fetch data */
       } while((time_count > 0)&&(stat == REQ_WRKN));
                        /* as long as job is ready with or without errors */

      switch(stat)  {
        case REQ_NO_ERR:      printf("Data have been readed\n", value);
                  break;
        case REQ_UNDEF: printf("job status nondefined\n");
                  break;
        default:        printf("job is ready with error: %d\n", stat);
      }
   }
```

1.3.5.4 Write a Single Element into the PLC

int CP_write_AG(byte size, byte type, byte bst, unsigned long adr, byte bit, void far *p);

size:	data size	
typ:	data type single elements	
bst:	module number	
adr:	address in module or absolute address	
bit:	bit number	
p:	pointer to the date to be written in the CP-memory	
	for data type bit, semaphore or byte	pointer to a byte
	for data type word	pointer to a word
	for data type doubleword	pointer to a doubleword

More details see chapter 1.3.5.10.

Return job number or negative number is there is an error

This function calls the driver function "write a single element into a PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
int a_nr;                   /* job number for read job*/
int stat;                   /* momentaneous job status */
int time_count=4000;        /* timeout counter (4 seconds) */
byte wert = 0x5A;           /* value read from the PLC */
/* start job */
a_nr = CP_write_AG(LBYTE_ELM, DB_SNG, 10, 1, 0, &wert);
if(a_nr < 0)                                  /* error occurred */
     printf("job finished with error: %d\n", a_nr);
   else
{                           /* a_nr contains job number */
     do
     {
        stat = CP_stat_AG(a_nr, NULL);        /* read job status */
     } while((time_count > 0)&&(stat == REQ_WRKN));
               /* as long as job is ready with or without errors */
     switch(stat)
     {
          case REQ_NO_ERR:    printf("date: %d was written\n", value);
                              break;
          case REQ_UNDEF:     printf("job status nondefined\n");
                              break;
          default:            printf("job is ready with error: %d\n", stat);
     }
 }
```

1.3.5.5 Write a Block into the PLC

int CP_writen_AG(byte size, byte typ, byte bst, unsigned long adr, word len, void far *p);

size:	data size of block elements
typ:	data type block elements
bst:	module number
adr:	address in module or absolute address
len:	number of data in words
p:	pointer to the data block to be written in the CP-memory

More details see chapter 1.3.5.10.

Return: job number $80 or negative number if there is an error

This function calls the driver function "write a block into a PLC". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If the driver has detected an error during the execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job number $80 (hex) is returned as function value.

Recommended calling method

To process correctly driver functions, the following scheme should be adhered when executing functions. Otherwise the bank, for example, can be blocked (cf. sections about driver functions).

```
int a_nr;                           /* job number for read job*/
int stat;                           /* momentaneous job status */
int time_count=4000;                /* timeout counter (4 seconds) */
int i;
byte buff[100];                     /* data to be written */

for(i = 0; i< 100; i++)
        buff[i] = (byte)i;          /* preset data buffer */

/* start job */
   a_nr = CP_writen_AG(B_BLOCK, DB_BLK, 5, 10, 100, &buff);
   if(a_nr < 0)                              /* error occurred */
     printf("job finished with error: %d\n", a_nr);
   else {                                    /* a_nr contains job number */
      do {
         stat = CP_stat_AG(a_nr, NULL);      /* read job status */
      } while(stat == REQ_WRKN);
                     /* as long as job is ready with or without errors */
      switch(stat)  {
          case REQ_NO_ERR:    printf("data have been written\n", value);
                              break;
          case REQ_UNDEF:     printf("job status nondefined\n");
                              break;
          default:            printf("job is ready with error: %d\n", stat);
      }
   }
```

1.3.5.6 Read Job Status

int CP_stat_AG(int r, void far *p);

a_nr: job number of the job to be tested
p: pointer to date or data block in CP-memory
(only for read jobs with single and block element
pointer to a byte for data variables bit, semaphore or byte
pointer to a word for data type word
pointer to a doubleword for data type doubleword
pointer to buffer for data block for data type block

Return: job status or negative number if error

This function calls the driver function "status call for job". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If there was an error during the job execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the job status (see chapter 1.3.5.10) is returned.

1.3.5.7 Abort All Jobs of a Bank

int CP_cncl_AG(int a_nr);

a_nr: code for bank 2 or 3
$00 abort all still active jobs of bank 2
$80 abort all still active jobs of bank 3

This function calls the driver function "abort all jobs of a bank". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If there was an error during the job execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, i.e. all jobs have been aborted, then 0 is returned.

1.3.5.8 Read Status of Process Image

byte CP_stat_PA();

Return: process image counter

This function calls the driver function "status call process image". The function returns the process image counter.

1.3.5.9 Read Area of Process Image

int CP_read_PA(byte typ, word adr, word len, void far *p);

typ: data type single elements
bst: module number
adr: address in module or absolute address
len: number of data (bytes or words) depending on the type
p: pointer to data buffer in CP-memory

More details see chapter 1.3.5.10.

Return: process image counter

This function calls the driver function "read an area of process image". The registers are preset according to the transferred parameters when calling up. Meaning of the parameters is described in the section of driver function. If there was an error during the job execution, then the respective error message (negative number) is returned as function value. If the function can be executed without errors, the current value of the process image counter is returned.

1.3.5.10 Constants for C

Following constants are already predefined. It is recommended to use these constants also in the program text for reasons of clearness and better readability. Moreover, adaptations attended to possible later changes of the COM-driver can be carried out easier.

1.3.5.10.1 predefined constants for data sizes

```
#define BIT_ELM     0x00      /* bit */
#define SEMA_ELM    0x01      /* bit as semaphore */
#define BYTE_ELM    0x02      /* byte */
#define LBYTE_ELM   0x02      /* left byte of a word */
#define RBYTE_ELM   0x03      /* right byte of a word */
#define WORD_ELM    0x04      /* word */
#define DWORD_ELM   0x05      /* doubleword */
#define BLOCK_ELM   0x07      /* block */
```

1.3.5.10.2 predefined constants for data types for single elements

```
#define DB_SNG      0x00      /* DB */
#define DX_SNG      0x01      /* DB in external memory */
#define BA_SNG      0x02      /* BA */
#define BB_SNG      0x03      /* BB */
#define BS_SNG      0x04      /* BS */
#define BT_SNG      0x05      /* BT */
#define Z_SNG       0x06      /* counter */
#define T_SNG       0x07      /* timer */
#define MB_SNG      0x08      /* marker */
#define EB_SNG      0x09      /* input area */
#define AB_SNG      0x0A      /* output area */
#define PB_SNG      0x0B      /* P-peripherals */
#define QB_SNG      0x0C      /* Q-peripherals */
#define ABS_SNG     0x0F      /* absolute memory */
```

1.3.5.10.3 predefined constants for data types for block elements

```
#define DB_BLK      0x00      /* data module */
#define DX_BLK      0x01      /* DB in external memory */
#define BA_BLK      0x02      /* BA */
#define BB_BLK      0x03      /* BB */
#define BS_BLK      0x04      /* BS */
#define BT_BLK      0x05      /* BT */
#define FB_BLK      0x06      /* FB */
#define FX_BLK      0x07      /* FB in external memory */
#define OB_BLK      0x08      /* OB */
#define PB_BLK      0x09      /* PB */
#define SB_BLK      0x0A      /* SB */
#define MB_BLK      0x0B      /* MB */
#define ABS_BLK     0x0F      /* absolute memory */
```

1.3.5.10.4 predefined constants for data type for block elements

```
#define B_BLOCK     0x0F      /* type: block with bytes  */
#define W_BLOCK     0x1F      /* type: block with words  */
#define D_BLOCK     0x2F      /* type: block with extended words */
```

1.3.5.10.5 identifications for job status

```
#define REQ_WRKN    0x01      /* job in processing */
#define REQ_UNDEF   0x02      /* job status not defined */
#define REQ_NO_ERR  0x03      /* job ready without errors */
```

1.3.5.10.6 predefined constants for data types for process image

```
#define Z_PA        0x06      /* counter*/
#define T_PA        0x07      /* timer */
#define MB_PA       0x08      /* marker */
#define EB_PA       0x09      /* input area */
#define AB_PA       0x0A      /* output area */
#define ABS_PA      0x0F      /* absolute block in PA */
```

1.3.5.10.7 predefined constants for error messages: bank 2, 3 and 7

CONST

ERR_S5_TYP = $01; (* invalid element type *)

With a single-element access with the element type DX_SNG, BA_SNG, BB_SNG, BT_SNG or QB_SNG or with a block element access with element type DX_BLK, BA_BLK, BB_BLK, BT_BLK or FX_BLK the programme tried to access data in a PLC of the type 115U. However, these element types do not exist in this PLC type.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

ERR_S5_BST = $02; (* module not available *)

With a single-element access with element type DB_SNG or with a block element type DB_BLK the programme tried to access a not existing module.

Correction: To create data block in the PLC or to correct parameter „bst“ in the function call of the CP-user software.

ERR_S5_ELM = $03; (* element not available *)

With a single-element access with element type DB_SNG or with a block element access with element type DB_BLK the programme tried to access data in a data block which are not available.

Correction: To extend the data block in the PLC correspondingly or to correct the parameter „adr“ or „len“ in the function call of the CP user software.

With a single-element access with element type Z_SNG or T_SNG the programme tried to access timer or counter with a number > 127.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

With a single-element access with element type MB_SNG the programme tried to access flags with a number > 199 with the size of element Byte, with number > 198 with the size of element word or with number > 196 with the size of ement douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

With a single-element access with element type EB_ - or AB_SNG the programme tried to access the process image of the I/O range with number > 127 with the element size Byte, with number > 126 with element size word or with number > 124 with element size douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

With a single-element access with element type PB_SNG the programme tried to access elements of the P-peripherals with number > 255 with element size Byte, with number > 254 with element size word or with number > 252 with element size douple word.

Correction: To check the parameter „adr“ in the function call of the CP user software for valence.

ERR_S5_SIZE = $04; (* invalid element size *)

With a single-element access with element type Z_SNG or T_SNG the programme tried to access timer or counter, whereas the parameter element size was not set to word access (WORD_ELM).

Correction: To correct the parameter „size“ in the function call of the CP user software.

With a single-element access with element type MB_SNG or ABS_SNG the programme tried to access flags or absolute addresses with the parameter element size RBYTE_ELM.

Correction: To correct the parameter „size“ in the function call of the CP user software.

With a single-element access with element type EB_SNG or AB_SNG the programme tried to access inputs or outputs in the process image with the parameter element size SEMA_ELM or RBYTE_ELM.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

With a single-element access with element type PB_SNG the programme tried to access the P-peripherals with the parameter element size BIT_ELM, SEMA_ELM or RBYTE_ELM.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

With a reading single-element access with element type ABS_SNG the programme tried to read absolute addresses with element size SEMA_ELM. This type of access is only possible in writing under absolute addressing! When single bits are to be read then the element size BIT_ELM has to be used.

Correction: To correct the parameter „typ“ in the function call of the CP user software.

ERR_S5_BIT = $05; (* Bit-number too high *)

With a single-element access with element type MB_SNG or ABS_SNG and the element size BIT_ELM or SEMA_ELM the programme tried to access a flag bit or an absolute address bit with a bit number > 7 (15).

Correction: To correct the parameter „bit“ in the function call of the CP user software.

With a single-element access with element type EB_SNG or AB_SNG the programme tried to access an I/O-BIT with a bit number > 7.

Correction: To correct the parameter „bit“ in the function call of the CP user software.

ERR_S5_STRT = $06; (* invalid starting address *)

With a block element access with element type „module“_BLK the programme tried to transfer blocks via modules whereas the relative starting address in the block is> 32767.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

ERR_S5_LEN = $07; (* invalide block length *)

With a block element access under all element types the programme tried to transfer blocks with a length > 504.

Correction: To correct the parameter „len“ in the function call of the CP user software.

ERR_S5_ADR = $08; (* Address too big *)

With a single- or block element access with element type ABS_SNG the programme tried to addess an address > FFFFh in a PLC of the typee 115U. However, the CPUs (up to CPU 944) have an address range of only 64 KB.

Correction: To correct the parameter „adr“ in the function call of the CP user software.

ERR_S5_QVZ = $09; (*QVZ/ADF in the PLC with read-/writing*)

The programme tried to access an address range which is physically not available.

The PLCs of the type 135 and 155 make this error message available. A PLC of the type 115U would be set to STOP in this case.

Correction: To correct the parameters „typ“ or „adr“ in the function call of the CP user software.

ERR_S5_944 = $0A; (* CPU 944: module in prog.bank *)

With a block element access with element type „module“_BLK the programme tried to access a module which is not in the data block. (This only concerns the CPU 944 form the PLC type 115U)

Correction: To create a module in the PLC in the data block bank (via BIB-Nr. 19285) or to correct the function call in the CP user software.

1.3.6 Storage of Process Images to Bank 7

The process image can also be directly read out by the user. Following survey shows how bank 7 is structured. Direct access is very fast:

```
Address in the
bank (hex)

Byte   0 process image EB 0          -+
       .                             ¦
       .                             ¦ 128 byte PAE 0-127
       .                             ¦
Byte 127 process image EB 127        -+
Byte 128 process image AB 0          -+
       .                             ¦
       .                             ¦ 128 byte PAA 0-127
       .                             ¦
Byte 255 process image AB 127        -+
Byte 256 marker Byte 0               -+
       .                             ¦
       .                             ¦ 256 byte marker 0-255
       .                             ¦
Byte 511 marker Byte 255             -+
Byte 512/513 Timer 0   (high/low)    -+
       .                             ¦
       .                             ¦ 128 words timer 0-127
       .                             ¦
Byte 766/767 timer 127 (high/low)    -+
Byte 768/769 counter  0 (high/low)   -+
       .                             ¦
       .                             ¦ 127 words counter 0-126
       .                             ¦
Byte 1020/1021 counter 126 (high/low)-+
Byte 1022 count byte 1)                 +    count byte
Byte 1023 trigger interrupt on CP
```

Annotation:

All values of this bank are refreshed when the handling module CP L/S is called up (if this is enabled on the formal operand of the handling module). After every data refreshing in the bank 7 the handling module increments the count byte by 1. CP recognizes by this count byte whether data are valid and how often they have been refreshed since the last reading. Data are then valid when the count byte content is involved in the range dual 1...255. iegt. In the case of overflow the count byte starts again with 1.

The handling module Synchron sets the current counter, address 3FE in bank 7 to 0. By that the CP recognizes that the data in bank 7 are not valid in the moment.

This bank needs not to be deleted by the CP if 0 is contained in the count byte (address 3FE of bank).

This bank can only be write accessed by the handling module.

The handling module for refreshing data of bank 7 does not trigger any interrupt.

1.4 Operation of the CP386COM in a WINDOWS environment

From the tool disk version 2.2 onwards a programme library for MS-WINDOWS 3.1 with the following data is available:

The header file CP386WIN.H and the OBJ-file CP386WIN.OBJ.

The file CP386WIN.H contains the necessary definitions for an operation on WINDOWS.

The file CP386WIN.OBJ contains the communication functions on WINDOWS. The functions have to be called as described in chapter 6.4 for DOS (**exception:** CP_stat_AG)

Changes in the function call:

CP_stat_AG:	CP_stat_AG(byte r) with r = Order number.

New functions:

CP_init(void):	Creates a data area for the communication on Dual Port RAM and returns a pointer on this area.
CP_exit(void):	Sets free the data area. This command has to be called at the end of the programme.

Note:

In the SYSTEM.INI under the section [386Enh] the Dual Port RAM area has to be excluded with the command *EMMExclude* = ... from the WINDOWS memory management in addition to the entry in the CONFIG.SYS!

(This is valid for all cases where the CP486 runs on WINDOWS 3.1 because WINDOWS does not exclude the Dual Port RAM independently!)

Tool disk 2.2 contains an example for the operation under Windows.

2 Linkage with PLC by CP486COM

2 Linkage with PLC by CP486COM

2.1 General description

The data transfer between the CP486 and the PLC is controlled by means of handler modules on the PLC side and by means of software-interrupts on the CP-side. The following routines are available:

		Operation on the PLC-side	Operation on the CP-side
Page frame 2	CP-request: read/write data from/to PLC **(CP486 active)**	handler module is invoked cyclically (FB1)	Software interrupt for DOS call to driver for WIN
Page frame 7	Transfer process image to CP	handler module is invoked cyclically (FB1)	Software interrupt or direct access to the page frame

Tab. 2-1: Routines

The following data structures in the PLC may be accessed from the CP:

- single elements of the type bit, byte, word and double word, DB, DX, BA, BB, BT, BS, flag, inputs, outputs, timers, counters
- Data blocks DB, DX, MB, T, Z, BA, BB, BT, BS, FB, FX, OB, PB, SB

The following functions are available from version 3.00 of the CPX86 (software CP4-SW593 version 3.00) and version 3.00 of the handler module (CP4-SW977 and CP4-SW978 version 3.00).

The following description refers to the CPX86 program as COM-driver.

The following CPUs were tested: CPU943B, CPU944, CPU945, CPU928, CPU928B, CPU946/947.

The following types were tested:

Type	read	write	Notes
Bit	X	X	
Word left-hand byte	X	X	
Word right-hand byte	X	X	
Word	X	X	
Double word	X	X	
Word (block)	X	X	specify length in words
Double word (block)	X	X	specify length in words
BS-area Bit	X	X	
BS-area left-hand byte	X	X	
BS-area right-hand byte	X	X	
BS-area word	X	X	
BS-area double word	X	X	
BS-area word (block)	X	X	specify length in words
Counter word (1 counter)	X	X	

Type	read	write	Notes
Counter double word (2 counters)	X	X	Counters are rotated
Counter word (block n-counter)	X	X	
Timer word (1 timer)	X	X	
Timer double word (2 timer)	X	X	timers are rotated
Timer word (block n-timer)	X	X	
Flag bit	X	X	
Flag byte	X	X	
Flag word	X	X	
Double flag word	X	X	
Flag byte (block)	X	X	specify length in byte
Flag word (block)	X	X	specify length in byte
Input bit	X	X	
Input byte	X	X	
Input word	X	X	
Input double word	X	X	
Input byte (block)	X	X	specify length in words
Input word (block)	X	X	specify length in words
Output bit	X	X	
Output byte	X	X	
Output word	X	X	
Output double word	X	X	
Output byte (block)	X	X	specify length in words
Output word (block)	X	X	specify length in words
peripheral bit	X	X	
peripheral byte	X	X	
peripheral word	X	X	
peripheral double word	X	X	
peripheral byte (block)	X	X	specify length in words
peripheral word (block)	X	X	specify length in words
Absolute address byte	X	X	The right-hand byte is read for CPUs with word addressing
Absolute address word	X	X	
Absolute address double word	X	X	
Absolute address block	X	X	specify data length in byte
FB - block	X		specify data length in byte
OB - block	X		specify data length in byte
PB - block	X		specify data length in byte

Type	read	write	Notes
SB - block	X		specify data length in byte

Tab. 2-2: Overview of the types that where tested

Runtime of the FB in the different CPUs, in ms:

CPU designation	Idle	Transfer of 100 DW	Transfer of 200 DW
944	0,2	1,6	2,2
928	1,0	4,5	5,2
928B	0,2	1,8	3,1
945	0,02	0,55	1,0
946/947	0,125	1,0	1,6

Tab. 2-3: Runtime of the FB in the different CPUs

Changes to the V2.0 driver

The constant PB_SNG was changed to PY_SNG.

The constant QB_SNG was changed to QY_SNG.

The constant BLOCK_ELM is no longer supported, use the constant B_BLOCK instead.

Constants identified by _SNG and _BLK have the same significance, i.e. both may be used for block requests as well as single requests.

Application programs that read or write from/to the periphery, Q-periphery, flag blocks and absolute addresses must be changed in accordance with the new list of constants and re-compiled.

Block requests with more than 245 words may cause a "page frame blockage" indication from the driver for the PLC cycle. In this case, the request must be reissued.

For multi-processor operations, the CP requests are defined by means of the CP_CALL function. On the PLC it is necessary to call FB 10 (read/write page frame) in OB 1 and the start-up OBs require FB 20 (Synchron). In multi-processor operations, it is possible to implement communications from a CP with up to 4 CPUs.

Single processor operations may use the functions CP_read_AG, CP_write_AG, CP_readn_AG, CP_writen_AG or the function CP_CALL. If the CP_CALL function is used the CPU-No. must always be 0. The PLC program may use FB 1 or FB 10 and for synchronization purposes, it may use FB2 or FB12.

2.2 Installation of the page frame software

2.2.1 PLC-side: handler modules

The handler modules FB1/FB10 and FB2/FB12 must be loaded into the PLC to facilitate communications with the CP486. Handler module FB1 is started in OB1, and FB2 in the restart modules (OB20, OB21 and OB22).

2.2.1.1 FB1/FB10 (CP-L/S), read and write from/to the CP

Name	Format	Description
ANSS	KY	Number of requests
PAA	KF	Process image identifier
PAFE	MB	Flag byte for error messages

Tab. 2-4: Parameter list for starting FB1/FB10

ANSS AN The maximum number of requests that should be processed in the page frame when the handler module is started

SS The number of the base page frame

PAA Update the identifier of the process images in the page frame when the handler module is being started

= 0 process images must not be updated

≠ 0 process images will be updated

The value that is specified here is transferred to the process image and consists of the page frame number + 1. Valid page frame numbers range from 4 - 7.

PAFE Error messages from the handler module

= 0 no error occurred

≠ 0 an error occurred. The error number is supplied in the PAFE-byte

1 The maximum number of requests that should be processed when a handler module is started is 0.

2 The maximum no. of requests that should be processed when a handler module is started is larger than 127.

3 The base page frame number is not divisible by 8

5 The page frame has not been synchronized by the CP.

9 Page frame for process image not located in a valid area.

Scratch pads used: MB200-MB255

2.2.1.2 FB2/FB12 (SYNCHRON), Synchronization CP and AG

Name	Format	Description
SSNR	KF	Number of the base page frame
WART	KF	Type of synchronization
PAA	KF	Process image identifier
PAFE	BY	Flag byte for error messages

Tab. 2-5: Parameter list for starting FB2/FB12

SSNR Base page frame number

WART = 0 FB-SYNCHRON does not wait until the CP has synchronized every individual page frame

≠ 0 FB-SYNCHRON waits until the CP has synchronized every individual page frame

PAA Number of page frame where process image must be stored. Valid range is between 4..7.

PAFE Error message from the handler module

= 0 no error occurred

≠ 0 an error was detected:

3 number of base page frame is not divisible by 8.

Scratch pads used: MB200-MB255

2.2.2 CP486-side: MSDOS driver program

A special communication driver must be loaded into the CP486 to facilitate communications between the PLC and the CP by means of page frames. This driver has special features for communicating with VIPA handler modules. It provides a set of simple functions; i.e. the user no longer needs to have detailed knowledge on the structure and the operation of page frames. The software required for the control of the page frames is included with the driver. The software currently supports page frame 2 (AG passive, CP active) and page frame X (process image). The driver provides all page frame functions automatically and requires no configuration.

2.2.2.1 Driver installation

The driver is loaded when the CP486 is started; i.e. it is included in the file CONFIG.SYS. Please note that the PLC usually starts the handler modules for synchronization purposes when a re-start occurs. These modules will only wait a limited amount of time for a reaction from the CP if the WART parameter was not set. The PLC will indicate "not synchronized" until the driver is started in the CP. The following entry starts the driver:

```
DEVICE = CP486COM.EXE
```

This driver occupies app. 26KB of main memory (program and data) and must only be loaded once. Any further attempts to load the driver are rejected with a message indicating that the driver has already been installed. This driver can only be removed from memory by re-booting the CP.

The COM-driver was specially developed for the CP486 module supplied by VIPA GmbH and must not be used on other systems. If the driver is installed on other PC-systems, including the i386 or i486 processors, the operation of these systems is not guaranteed. The driver usually recognizes that the processor is not a CP.

2.2.2.2 DOS interrupts used by the driver

The driver occupies a number of software interrupts , which are used for communicating with the application software in the CP module:

- INT 1Ch Timer-Interrupt

  Regular cyclic checks of the page frame are controlled by the "ticker-interrupt" 1Ch as well as the DOS-idle interrupt. For example, this may be used on a regular basis to check whether the PLC wishes to re-synchronize the page frames. Once the CP-specific functions have been completed the original interrupt handler routine is executed (interrupt service routine).

- INT 74h (IRQ 12):

  The CP486 uses hardware-IRQ 12 that occupies the software-interrupt 74h. This interrupt is always activated when BASP is active in the PLC or when the highest memory location of any page frame (byte 1023) was written by the PLC. By using this interrupt, the CP can react quickly to requests received from the PLC. Upon completion of the CP-specific function, the original interrupt service routine is executed. For this reason, it is possible that a number of devices use IRQ 12.

- INT 78h Service-Interrupt:

  This interrupt must be used by the application software in the CP to access the driver functions, e.g. where data must be transferred to/from the PLC via page frame 2. It is possible to access different functions by specifying different parameters for the processor registers. If INT 78 is supplied with register values that are not valid for CP486 communications, the original interrupt service routine will be executed.

2.2.3 Various representations of data in memory

The different rules for representing words and double words (long words) in the CP and in the PLC must be met when data is transferred between the CP and the PLC.

Data words are stored differently in the CP than in the PLC. The positions of the most significant byte (high-byte) and the least significant byte (low-byte) have been swapped. In the case of double words, the sequence of all 4 bytes has been reversed. Where data is transferred between the PLC and the CP, the position of relevant bytes must be swapped at some time, as the transferred data would otherwise be invalid. Wherever possible, the COM driver will adjust the data automatically as required.

The driver will perform an automatic swap for all data transfers to/from page frame 2 and 7.

- Data is not modified while it is being transferred.
- The most significant and least significant bytes of words are swapped during transfer.
- The sequence of all 4 bytes of a long data word is reversed during transfer.

Representation of data in the PLC

Addressesn	byte	representation byte
Address n	high-byte	
Address n+1	low-byte	representation byte
Address n	high-byte high-word	
Address n+1	low-byte high-word	
Address n+2	high-byte low-word	representation double word
Address n+3	low-byte low-word	

Representation of data in the CP

Address n	byte	representation byte
Address n	low-byte	
Address n+1	high-byte	representation word
Address n	low-byte low-word	
Address n+1	high-byte low-word	
Address n+2	low-byte high-word	representation double word
Address n+3	high-byte high-word	

2.3 CP486-Requests for PLC (Page Frame 2 and 7 Functions)

2.3.1 Overview

The CP486 driver handles the requests that were initiated from the CP. This driver provides a number of functions for page frames 2 and 7. These functions may be used to read data from a PLC or write data to a PLC from the application program running on the CP486 . All these functions are initiated by means of software interrupt 78h. From version 3.10 this calling structure has been expanded to include a structure for the transfer of data.

When the interrupt is activated the required parameters are sent and received via the processor registers. The function description contains a description of the allocation of these registers. The interfaces cater for Turbo-Pascal, Turbo-C and C++ as well as Microsoft-C. Functions are available for reading and writing data from/to the PLC, for interrogating statuses and for termination functions. Data transfer functions are classified according to the type of transfer. Here we differentiate between single transfers and blocked transfers. Different functions are performed on the basis of "requests". When a function is activated, it will return a request number, which may be used to interrogate the status of the respective process. Page frame 2 can handle up to 90 simultaneous requests.

The following functions are available

Page frame	Function no.	Function
none	$00	Status request
2	$21	Read/write a single element to/from the PLC
2	$21	Read/write a block to/from the PLC
2	$20	Status request for the read/write request
2	$28	Termination of all read/write requests
2	$7f	Initiation by means of a structure
7	$70	Status request for the process image
7	$71	Read the process image area

Tab. 2-6: Function description

2.3.2 Driver functions controlled by software interrupts

2.3.2.1 CP-status request

The application software can use this function to determine which CPx86 drivers have been loaded.

Register	IN	high	OUT	low
AX	$00	$C386		
BX				
CX				
DX				

AX C386h flag indicating that the CP-software has been loaded.

The function returns a 0 if the driver has been loaded, otherwise it returns a -1.
This function does not initialize page frames.

2.3.2.2 Reading a single element from the PLC

You may use this function to read a single data element (bit, byte, word, ...) from the PLC. The function initiates the request and immediately returns to the caller. You can then read the actual data by means of the "status request" function (see chapter 2.3.2.6).

Register	high	In	low	Out
AX	$21	type		status
BX	size	bst		
CX	adr			
DX	-	bit		

Parameters

type	Data element type for single elements in the PLC (DB, MB...) (see chapter 2.3.3.11.2)	
size	Element size indicator (bit, byte ...) see chapter 2.3.3.11.1	
bst	Module number, only for element type DB or DX. For element type "absolute", this contains the most significant bits of adr.	
adr	Start address in the range	
bit	Bit number when element size is bit.	
status	< 0	error number, when an error has occurred. Error numbers of the PLC are added to FF00h
	1..255	Request number which may be used to interrogate the status

Note

This function does not return any data! Any data may be retrieved by issuing a call to the status request function if the request was "completed without error".

Read requests that have been completed are blocked to ensure that it is not overwritten by a new request before the returned data has been retrieved. Once a request has been started, its status must be interrogated until the request returns "completed with error" or "completed without error". If the returned status is "completed without error" the respective data is copied to the specified address in the CP. If the status is not interrogated, the request remains locked and no further read requests can be started, even if all requests for the page frame have been completed.

2.3.2.3 Reading a block from the PLC

You can use this function to read an entire block of data from the PLC. The function initiates the request and returns immediately to the caller. You can access the returned data by means of the "status request" function (see chapter 2.3.2.6).

Register	high	In	low	Out
AX	$21	type		status
BX	size	bst		
CX	adr			
DX	len			

Parameters

type	Element type for the data of a single element in the PLC (DB, MB...) (see chapter 2.3.3.11.2)	
size	Element size of block data. Indicator whether individual bytes are swapped:	
	07h	Data block consisting of bytes (no swapping)
	17h	Data block consisting of words (high and low byte are swapped)
	27h	Data block consisting of double words (all 4 bytes are exchanged)
bst	Module number, element type DB, DX , FX. For element type "absolute", this contains the most significant bits of adr.	
adr	Start address in the range	
len	Data length in words or bytes. The value in the PLC determines the length in bytes or words, e.g. timer, counter, DW length specified in words or for flags and outputs, in bytes.	
status	< 0	error number, when an error has occurred
	1..255	request number which may be used to interrogate the status

Note

The type of data must be specified in the block (bytes, words, double words) to ensure that data can be aligned automatically during transfer. Each block can only contain data of the same type. For words and double words, the data bytes are swapped as required.

This function does not return any data! Any data may be retrieved by issuing a call to the status request function if the request was "completed without error".

Read requests that have been completes are blocked to ensure that it is not overwritten by a new request before the returned data has been retrieved. Once a request has been started, its status must be interrogated until the request returns "completed with error" or "completed without error". If the returned status is "completed without error" the respective data is copied to the specified address in the CP. If the status is not interrogated, the request remains locked and no further read requests can be started, even if all requests for the page frame have been completed.

2.3.2.4 Writing a variable to the PLC

This function is used to write a single data element (bit, byte, word,...) into the memory of the PLC. The call to this function must include the address of the variable that must be written. The function writes the value into the page frame and does not wait for the PLC to fetch the data, it returns to the caller immediately.

Register	high	low	Out
	In		
AX	$21	type	status
BX	size	bst	
CX	adr		
DX	-	bit	
SI	offset		
DS	segment		

Parameters

type	Data element type for single elements in the PLC (DB, MB ...) (see chapter 2.3.3.11.2)	
size	Element size indicator (bit, byte, ...), see chapter 2.3.3.11.1	
bst	Module number, only for element type DB or DX For element type "absolute", this contains the most significant bits of adr.	
adr	Start address in the range	
bit	Bit-number if the element size is bit.	
offset	Offset of the variables address (in the CP)	
segment	Segment of the variable address (in the CP)	
status	< 0	error number, when an error has occurred
	1-255	request number which may be used to interrogate the status

Note

For write requests, the page frame is locked in the same manner as for read requests. For this reason the requests status should also be interrogated until the returned status is "completed with error" or "completed without error".

The element size determines how the pointer to the data in the CP is interpreted.

- Bit:
  The pointer represents the address of a byte. The bit is read from bit-number 0.
- Byte, left byte, right byte:
  the pointer represents the address of a byte. The byte is read from the memory location.
- Word:
  The pointer represents the address of a word. High and low byte are swapped during the transfer.
- Double word/long word:
  The pointer represents the address of a long word. The long word is read and the sequence of all 4 bytes is reversed.

2.3.2.5 Writing a block to the PLC

You can write an entire data block into the PLC by means of this function. The call to this function must include a pointer to the data block that must be written. The function writes the data into the page frame and returns immediately to the caller. It does not wait until the PLC has fetched the data. The data block in the CP is again available and could, for instance, be overwritten.

Register	high In low	
AX	$21	type
BX	size	bst
CX	adr	
DX	len	
SI	offset	
DS	segment	

Parameters

type	Element type of the data for block elements in the PLC (see chapter 2.3.3.11.3)	
size	Element size of the block data. Indicator whether single bytes must be swapped.	
	07h	Data block consisting of bytes
	17h	Data block consisting of words
	27h	Data block consisting of double words
bst	Module number, only for element types DB, DX, FB. For element type "absolute", this contains the most significant bits of adr.	
adr	Start address in the range	
len	Number of data elements in words or bytes (see reading blocks)	
offset	Offset of the block address (in the CP)	
segment	Segment of the block address (in the CP)	
status	< 0	error number, when an error has occurred
	1..255	request number which may be used to interrogate the status

Note

The type of data in the block (bytes, words, double words) must be specified so that the data may be aligned automatically . A block may only contain data of a single type. In the case of words and double words, the data bytes are swapped as required.

A block write request locks the page frame. Once a request has been started, its status must be interrogated until the request returns "completed with error" or "completed without error". If the status returned by the request is "complete without error", the data is copied into the specified address in the PLC. If the status is not interrogated, the request remains locked and no further read requests can be started, even if all requests for the page frame have been completed.

2.3.2.6 Reading the status of a request

This function returns the status of a request that was started earlier. For read request variables and block read requests, the function will also copy the data into a specified address in the CP, if the returned status is "complete without error".

Register	high	**In**	low	**Out**
AX	$20	a_no		status
SI	offset			
DS	segment			

Parameters

fn	Function number for the status request	
a_no	request number	
offset	Offset of the data in the PC. offset must only be specified for read requests (variables and block).	
segment	Segment of the data address in the PC. segment must only be specified for read requests (variables and block).	
status	< 0	request "complete with error" Error messages from the PLC are added to FF00h.
	1	request processing "not completed"
	2	request status "undefined"
	3	request "complete without error"

Procedure for status requests

If the request returns processing "not completed", the status function must be called until the status changes.

- For write requests: if the request returns "complete without errors", the data was written into the PLC.
- For read requests: if the request returns "complete without errors" and a pointer to the data was specified, then the data was successfully copied into the respective address in the CP. The data bytes were swapped in accordance with the if the specified size of the data.
- If the status returned by the request is "undefined", then the request has already been terminated but it has not yet been overwritten by a new request. If this was a read request and the specified pointer was not equal to NULL, then the returned data is copied into the specified destination address.
- If the status returned by the request is "complete with errors" then the request block was released if the request was a read request or a block request.

- The pointer for read requests must be interpreted according to the specified size of the element:
    - Bit :
      The pointer represents the address of a byte. The bit is written to bit-number 0 and the entire byte is overwritten.
    - Byte, left byte, right byte:
      the pointer represents the address of a byte. The byte is written into the memory location.
    - Word:
      The pointer represents the address of a word. The high and low bytes are swapped during transfer.
    - Double word/long word:
      The pointer represents the address of a long word. The long word is read and the sequence of all 4 bytes is reversed.

- The pointer for read requests for a data block must be interpreted according to the specified size of the element:
    - Byte:
      The pointer represents the address of a block of bytes. All the bytes are transferred from the PLC and their sequence is not changed.
    - Word:
      The pointer represents the address of a block of words. The high and low bytes of every word are swapped during transfer.
    - Double word/long word:
      The pointer represents the address of a block of long words. The sequence of all 4 bytes of every long word is reversed during transfer.

2.3.2.7 Terminate all requests of a page frame

Register	high	**In** low	**Out**
AX	fn		status

Parameters

fn	Function number for status-request	
	$28	termination of all read requests
status	< 0	error number, when an error has occurred
	0	all requests were terminated.

Note

This function may be used to terminate any write or read request that has no been completed.

The function applies equally to variable and to block requests. It always returns a value of 0, even if no requests were active in the page frame.

2.3.2.8 Read process image status

Register	high	**In** low	**Out**
AX	$70	-	status

Parameters

status	0	process image not available
	1..255	up to date process image counter

Note

This function may be used to read the up to date value of the process image counter (page frame 7 address 3FEh). A process image is not available if the returned value is 0.

2.3.2.9 Read a from the process image area

This function may be used to read from the current process image area. The length of the area is monitored while it is being accessed.

Register	high	In	low	Out
AX	$71	type		status
BX	adr			
CX	len			
SI	offset			
DS	segment			

Parameters

type	Element type of the data of the process image (EB, MB)	
adr	Start address in the rage	
len	Number of data elements in bytes or words (depending on type)	
offset	Offset of the data address in the PC	
segment	Segment of the data address in the PC	
status	< 0	error number, when an error has occurred
	= 0	no process image available
	> 0	counter for process image (as for the status function)

Note

The length for timer and counter accesses is specified in words, and for all other types it is specified in bytes. The high and low bytes of timer and counter words are swapped so that the data may be processed by the PC.

If the type " is specified as "absolut", any section spanning arbitrary partitions of the process image may be read. The length is specified in bytes. If the range for a timer or counter is read, the high and low bytes are swapped.

No.	Type
06	Counter (specify length in words)
07	Timer (specify length in words)
08	Flag (length in bytes)
09	EB (length in bytes)
0A	AB (length in bytes)
0F	Absolute access to process image (length in bytes)

Tab. 2-7: Element types for process image

2.3.2.10 Error numbers of the CP for page frames 2, 3 and 7

hex	dec.	Description
FFFFh	-1	invalid element type
FFFEh	-2	Incorrect length (e.g. address too large, bit number too large)
FFFDh	-3	Invalid element size (incorrect value for single or for block request)
FFFCh	-4	Element type not available for the current CPU
FFFBh	-5	Page frame full, new requests can not be entered
		Request and block request must be started
FFFAh	-6	No access to a page frame for 10 sec. (PLC probably stopped)
FFF9h	-7	Request/page frame still locked
		(request status has not been interrogated to remove the lock).
FFF8h	-8	incorrect request number for status function (e.g. request no. > 255)
FFF7h	-9	incorrect source data pointer
		(write request with NULL address)
FFF6h	-10	Request not processed (unused!)
FFh	255	Invalid function call
EEh	238	Request terminated during initialization

Tab. 2-8: Error numbers of the CP for page frames 2, 3 and 7

2.3.3 Turbo-Pascal interface (from Version 4.0)

A Turbo-Pascal-Unit was created to provide access to the COM driver by means of function calls. This unit makes all the functions of the service interrupts INT 78 available. Every function of the driver is provided with an equivalent Pascal procedure that manages the registers, calls the interrupt and returns values as required. In this way, users that are not accustomed to low-level programming may also make use of the facilities provided by the driver.

All required functions, data types and constants are contained in the CP486LIB unit. This unit must be included in the respective Pascal application program by means of "USES CP486Lib". The user must also ensure that the compiled unit CP486LIB.TPU is located in the directory where Turbo-Pascal searches for units. The respective settings are made via menu items "Options | Directories | EXE & TPU-directory" (refer to the Turbo Pascal manual or help functions).

The following sections provide a short overview of the available functions. A detailed description with all the relevant information can be found in the functional description located in preceding paragraphs.

2.3.3.1 Function CP status request

```
FUNCTION CP_Info (VAR inforec : CP486InfoRec) : INTEGER;
```

Data structures

```
TYPE CP486InfoRec =
       RECORD
          CP_id : WORD;               (* Id: CP486 value = $C386     *)
          VGA_ver, BIOS_ver : BYTE;   (*                             *)
          DRV_ver: BYTE;              (*                             *)
          CPU_AG : BYTE;              (*                             *)
          CP_reg, S5_reg : BYTE;      (*                             *)
       END;
```

Data structure for the general status info function. The components are completed according to the values of the CP486.

This function calls the driver function "general status information". The function is preceded by an installation check of the driver. If the driver was not installed, the function returns a value of -1. If the COM driver was installed properly, the function returns a value of 0. The CP_id component always contains the identifier $C386, other components are not changed.

2.3.3.2 Reading a single element from the PLC

```
FUNCTION CP_read_AG (size, type, bst : BYTE; adr : longint;
                     bit : BYTE):INTEGER;
```

Parameter

size	Element size of single elelements (see chapter 2.3.3.11.1)
type	Element type for single elements (see chapter 2.3.3.11.2)
bst	Module number
adr	Address in the module or absolute address
bit	Bit number

Returns

Request number or negative number for errors.

This function calls the driver function "read a single element from the PLC". The call assigns the specified parameters to the respective registers. Chapter 2.3.3.11 describes the significance of the individual parameters. If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is completed without errors, the request number is returned as function value.

Calling procedure

The following scheme should be applied when processing the driver function, otherwise the page frame might, be locked:

```
VAR   a_no,                     (* request number for read request *)
      stat : INTEGER;           (* instantaneous request status    *)
      value : BYTE;             (* value read from PLC             *)
BEGIN

(*Start the request *)
   a_no := CP_read_AG(LBYTE_ELM, DB_SNG, 10, 1, 0);

   IF a_no < 0                  (*an error occurred                *)
   THEN WriteLn(''Request terminated due to error: ', a_no)

   ELSE  BEGIN                  (*a_no contains the request number *)
      REPEAT
            stat := CP_stat_AG(a_no, Addr(value)); (*fetch request
                                                     status/data *)
      UNTIL stat <> REQ_WRKN;   (*repeat until request is complete
                                with or without errors             *)

      CASE stat OF
        REQ_NO_ERR:WriteLn('Data: ', value, ' was read');
        REQ_UNDEF: WriteLn('Request-status undefined, data:
      ',value,' was read');
        ELSE WriteLn('Request completed with error: ', stat)
      END;
   END;
END.
```

2.3.3.3 Reading a block from the PLC

```
FUNCTION CP_readn_AG (size, type, bst:BYTE; adr : longint;
                      len :  WORD) : integer;
```

Parameters

size	Data type of the block elements (see chapter 2.3.3.11.4)
type	Element type of the block elements (see chapter 2.3.3.11.3)
bst	Module number
adr	Address in the module or absolute address
len	Length of data in words or bytes, depending on element type

Returns

Request number or negative number for errors.

This function calls the driver function "Read a block from the PLC". The call assigns the values supplied in parameters to the registers. Parameters are described in chapter 2.3.3.11 . If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, the request number is returned as function value.

Calling procedure

The following scheme should be applied when processing the driver function, otherwise the page frame might, be locked:

```
VAR   a_no,                  (*Request number for the read request *)
      stat : INTEGER;           (*up to date request status *)
      buff : ARRAY[1..100 ] OF INTEGER(*data read from the PLC *)
BEGIN
 (*Start the request *)
   a_no := CP_readn_AG(W_BLOCK, DB_BLK, 5, 10, 100);

   IF a_no < 0                   (*an error has occurred          *)
   THEN WriteLn(' Request completed with errors: ', a_no)

   ELSE  BEGIN                 (*a_no contains the request number *)
      REPEAT
            stat := CP_stat_AG(a_no, Addr(buff)); (*Fetch the
                                          request status/data *)
      UNTIL stat <> REQ_WRKN;  (*repeat until the request completes
                                  with or without error       *)
      CASE stat  OF
          REQ_NO_ERR:       WriteLn(' Buffer read ');
          REQ_UNDEF:        WriteLn(' Undefined request status
                                     buffer was read');
          ELSE              WriteLn(' Request completed with error:
                                    ', stat);
      END;
   END;
END.
```

2.3.3.4 Writing a single element to the PLC

```
FUNCTION CP_write_AG (size, type, bst : BYTE; adr: longint;
                      bit: BYTE; p: POINTER): integer;
```

Parameter

size	Element size (see chapter 2.3.3.11.1)	
type	Element type single elements (see chapter 2.3.3.11.2)	
bst	Module number	
adr	Address in the module or absolute address	
bit	Bit number	
p	Pointer to write data	
	if element size is bit or byte:	pointer to a byte
	if element size is word:	pointer to a word
	if element size is double word:	pointer to a double word

Returns

Request number or negative number for errors.

This function calls the driver function "Write a single element to the PLC". The call assigns the specified parameters to the respective registers. Parameters are described in chapter 2.3.3.11 . If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, the request number is returned as function value.

Calling procedure

```
VAR   a_no,                (*Request number for the write request *)
      stat : INTEGER;          (*up to date request status        *)
      value : BYTE;            (*value to be written              *)
BEGIN
 ...
(*Start the request *)
   value := $7E;
   a_no := CP_write_AG(LBYTE_ELM, DB_SNG, 10, 1, 0, Addr(value));

   IF a_no < 0                 (*an error has occurred            *)
   THEN WriteLn(' Request completed with errors: ', a_no)

   ELSE  BEGIN                 (*a_no contains the request number *)
      REPEAT
            stat := CP_stat_AG(a_no, NIL);     (* read the request
                                                     status    *)
      UNTIL stat <> REQ_WRKN;  (* repeat until request completes
                                  with or without error          *)

      CASE stat OF
          REQ_NO_ERR: WriteLn('Data: ', value,' was written');
          REQ_UNDEF:  WriteLn('Undefined request status.');
          ELSE        WriteLn('Request completed with error: ',
                              stat)
      END;
   END;
End.
```

2.3.3.5 Writing a block to the PLC

```
FUNCTION CP_writen_AG (size, type, bst : BYTE; adr : longint;
                       len : word; p : POINTER) : integer;
```

Parameters

size	Data type of the block elements (see chapter 2.3.3.11.4)
type	Element type block elements (see chapter 2.3.3.11.3)
bst	Module number
adr	Address in the module or absolute address
len	Length of data in words or bytes, depending on element type
p	Pointer to write data

Returns

Request number or negative number for errors.

This function calls the driver function "Write a block to the PLC". The call assigns the specified parameters to the respective registers. Parameters are described in chapter 2.3.3.11.

If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, the request number is returned as function value.

Calling procedure

```
VAR   a_no,                 (*Request number for the read request *)
      stat : INTEGER;          (*up to date request status        *)
      i : INTEGER;
      buff : ARRAY[1..100 ] OF INTEGER;(* data to be written      *)
BEGIN
 ...
   FOR i := 1 TO 100 DO
            buff[i] := i;      (* assign data to data buffer      *)
(* Start the request *)
   a_no := CP_writen_AG(B_BLOCK, DB_BLK, 5, 10, 100, Addr(buff));

   IF a_no < 0                (* an error has occurred           *)
   THEN WriteLn(' Request completed with errors: ', a_no)

   ELSE  BEGIN                (* a_no contains the request number *)
      REPEAT
            stat := CP_stat_AG(a_no, NIL);      (* read the request
                                                      status      *)
      UNTIL stat <> REQ_WRKN;       (* repeat until request
                                 completes with or without errors *)

      CASE stat OF
          REQ_NO_ERR: WriteLn('Buffer written');
          REQ_UNDEF: WriteLn(' Undefined request status.');
          ELSE      WriteLn(' Request completed with error:', stat)
      END;
   END;
END.
```

2.3.3.6 Read the status of a request

```
FUNCTION CP_stat_AG (a_no : INTEGER; p: POINTER): INTEGER;
```

Parameter

a_no	Request number of the respective request
p	Pointer to the data or data block located in the PC's memory (must only be - completed for read requests with single or block) if element size is bit or byte — pointer to a byte if element size is word — pointer to a word if element size is double word — pointer to a double word if element size is block — pointer to buffer for data block

Returns

Request status or negative number when an error has occurred.

This function calls the driver function "Status request for a request". The call assigns the specified parameters to the respective registers. Parameters are described in chapter 2.3.3.11 . If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, the request status (see chapter 2.3.3.11.4) is returned.

2.3.3.7 Terminating all requests of a page frame

```
FUNCTION CP_cncl_AG (a_no : BYTE): INTEGER;
```

Parameter

a_no	Identifier for page frame 2 $28 terminate all active requests of page frame 2

Returns

0 or negative number if an error has occurred.

This function calls the driver function "Terminate all requests of a page frame". The call assigns the specified parameters to the respective registers. Parameters are described in chapter 2.3.3.11 If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, i.e. all requests were terminated properly, then the request returns a value of 0.

2.3.3.8 Read satus of process image

```
FUNCTION CP_stat_PA : BYTE;
```

Returns

Process image counter

This function calls the driver function "Request status of process image". The function returns the process image counter.

2.3.3.9 Read the process image area

```
FUNCTION CP_read_PA(type : BYTE; adr, len : WORD; p : POINTER) :
                                    INTEGER;
```

Parameters

type	Element type process image (see chapter 2.3.3.11.6)
adr	Address in the area or absolute address
len	Length of data (bytes or words) depending on type
p	Pointer to data buffer in memory

Returns

Process image counter

This function calls the driver function "Read an area of the process image ". The call assigns the specified parameters to the respective registers. Parameters are described in chapter 2.3.3.11. If the driver should detect an error, an appropriate error message (negative number) is returned as a function value. If the function is processed without errors, the request returns the up to date value of the process image counter.

2.3.3.10 Standard functions

```
FUNCTION CP_CALL (VAR setwert : CPX86_PARAMETER_REC) : INTEGER;
```

Data structure

```
TYPE CPX86_PARAMETER_REC =
     RECORD
       Elementtyp       : byte;     (* Element type see chapter 2.3.3.11.2 or.
                                       2.3.3.11.3 *)
       Auftrag          : byte;     (* Request type see chapter 2.3.3.11.7*)
       Bausteinnummer   : byte;     (*Number of the data module or 0 for
                                     flags, outputs etc. *)
       Kennung          : byte;     (*Element size or data type see
                                       chapter 2.3.3.11.1 or 2.3.3.11.4 *)
       Adresse          : integer;  (* Number of the first data element *)
       Len              : integer;  (* Length of the data *)
       ptr              : pointer;  (* Pointer to DOS-data always "nil" *)
       ptr_win          : pointer;  (* Pointer to WIN-data always "nil" *)
       upro_zeiger      : longint;  (*Pointer to subroutine, unused *)
       CPU              : byte;     (* CPU number, 0-3, 0 for a single CPU 0*)
       Reserved         : byte;     (* Reserved byte*)
       Fehler           : integer;  (* Returned byte for errors*)
       case Integer of
         0:      (DatenByte   : ARRAY[1..1000] of byte);
                   (* Data transfer from/to the procedure, in byte (8 Bit) *)
         1:      (DatenWort   : ARRAY[1..500] of integer);
                   (* Data transfer from/to the procedure, as integer (16 Bit)*)
         2:      (DatenDoppel : ARRAY[1..250] of longint);
                   (* Data transfer from/to the procedure, as Longint (32 Bit)*)
     END;
```

Data structure for the standard function. Values are assigned to the components as required by the function that is being executed.

This function can be used instead of all the previously described functions.

The functions write_AG or writen_AG enters all write data into the respective ARRAY and issues a call to the function CP_CALL.

The functions read_AG or. readn_AG return all data after the function CP_CALL has been completed (as function CP_stat_AG) in the respective ARRAY.

Calling procedure

```
(*Turbo-Pascal example for communicating via page frame 2 with 2 CPUs *)

         program Test;

         USES DOS, crt, CP486LIB;

         VAR
             ret1,ret2     : integer;
             a_no          : BYTE;
             i,zaehl       : WORD;
             setwert       : cpx86_parameter_rec;

         begin

          clrscr;
          writeln;
          writeln(' This DEMO requires FB1 on the PLC side,');
          writeln(' which is called cyclically by OB1');
          delay(2000);

         repeat
         begin
                                  (* Structure for predefined request *)
             setwert.elementtyp := DB_SNG;(* Element type acc. to table    *)
             setwert.auftrag := READ_AG;     (*Request type                 *)
             setwert.bausteinnummer := 10;   (* Number of the data module  *)
             setwert.kennung := W_BLOCK;     (* Elemet size acc. to table  *)
             setwert.adresse := 0;           (* Start address in the PLC   *)
             setwert.len     := 50;          (* Length for data transfer   *)
             setwert.cpu     := 0;           (* CPU-number 0 for first CPU *)

             ret1 := CP_CALL(setwert);       (* Transfer data to driver    *)

                    (* Structure for request has been preset *)
             setwert.elementtyp := DB_SNG;   (* Element type acc. to table *)
             setwert.auftrag := READ_AG;     (* Request type               *)
             setwert.bausteinnummer := 10;   (* Number of the data module  *)
             setwert.kennung := W_BLOCK;  (* Element size acc. to table *)
             setwert.adresse := 0;                  (* Start address in PLC *)
             setwert.len     := 50;                 (* Length of transfer   *)
             setwert.cpu     := 1;        (* CPU number 1 for second CPU   *)

             ret2 := CP_CALL(setwert);    (* Transfer data to driver       *)

            IF ret1 < 0                       (* An error has occurred      *)
             THEN write(' Request completed with errors: ',ret1,'   ')
             ELSE
              begin
              a_no := ret1;              (* Transfer the request number    *)
              repeat
                setwert.elementtyp := a_no; (*Request number into structure*)
                setwert.auftrag := statr_AG; (* Request type               *)
                setwert.cpu     := 0;              (* Set CPU number       *)
                setwert.ptr     := nil;            (* Delete the pointer   *)

                ret1 := cp_call(setwert);    (* Transfer data to driver    *)
              until (ret1 <> 1);

              if ret1 = 3 then          (* Function executed without error *)
                begin
                 gotoxy(1,6);             (* Position the cursor           *)
                 write('Data :   ');
                 for i := 1 to 50 do
                  begin
```

```
                    write(setwert.datenwort[i]:5,'   '); (* Display the data
                                                          on screen    *)
                  end;
                end;
               end;


  IF ret2< 0                              (* An error has occurred          *)
   THEN write(' Request terminated with error: ',ret2,'   ')
   ELSE
    begin
    a_no := ret2;                         (* Transfer request number         *)
    repeat
      setwert.elementtyp := a_no;         (* Set request no. into structure  *)
      setwert.auftrag := statr_AG;        (* Request type                    *)
      setwert.cpu     := 1;               (* Enter CPU number                *)
      setwert.ptr     := nil;             (* Clear the pointer               *)

      ret2 := cp_call(setwert);           (* Transfer data to driver         *)
    until (ret1 <> 1);

    if ret2 = 3 then               (* Function completed without error      *)
      begin
       gotoxy(1,6);                (* Position the cursor                   *)
       write('Daten :   ');
       for i := 1 to 50 do
        begin
         write(setwert.datenwort[i]:5,'   ');( * Display data on screen      *)
       end;
    end;
   end;


end;

until keypressed;

end.
```

2.3.3.11 Constants

The following constants have already been defined. It is recommended that these are used, as they are more readable and result in a clearer program. The resulting program can then easily be adapted to cater for changes in the COM driver.

2.3.3.11.1 Constants for element sizes

```
CONST
      BIT_ELM     = $00;      (* bit                          *)
      SEMA_ELM    = $01;      (* bit as semaphore             *)
      BYTE_ELM    = $02;      (* byte                         *)
      LBYTE_ELM   = $02;      (* left byte of a word          *)
      RBYTE_ELM   = $03;      (* right byte of a Word *)
      WORD_ELM    = $04;      (* word                         *)
      DWORD_ELM   = $05;      (* double word                  *)
      BLOCK_ELM   = $07;      (* block                        *)
```

2.3.3.11.2 Constants for element types of single elements

```
CONST
      DB_SNG      = $00;      (* DB                           *)
      DX_SNG      = $01;      (* DB in external memory        *)
      BA_SNG      = $02;      (* BA                           *)
      BB_SNG      = $03;      (* BB                           *)
      BS_SNG      = $04;      (* BS                           *)
      BT_SNG      = $05;      (* BT                           *)
      Z_SNG       = $06;      (* Counter                      *)
      T_SNG       = $07;      (* Timer                        *)
      MB_SNG      = $08;      (* Flag                         *)
      EB_SNG      = $09;      (* Input area                   *)
      AB_SNG      = $0A;      (* Output area                  *)
      PY_SNG      = $0B;      (* P-periphery                  *)
      QY_SNG      = $0C;      (* Q-periphery                  *)
      ABS_SNG     = $0F;      (* Absolute memory              *)
      FB_SNG      = $10;      (* FB                           *)
      FX_SNG      = $11;      (* FB in external memory        *)
      OB_SNG      = $12;      (* OB                           *)
      PB_SNG      = $13;      (* PB                           *)
      SB_SNG      = $14;      (* SB                           *)
      RB_FREE     = $0E;      (* Code for request block free *)
```

2.3.3.11.3 Constants for element types of block elements

```
CONST
      DB_BLK      = $00;          (* DB                     *)
      DX_BLK      = $01;          (* DB in external memory  *)
      BA_BLK      = $02;          (* BA                     *)
      BB_BLK      = $03;          (* BB                     *)
      BS_BLK      = $04;          (* BS                     *)
      BT_BLK      = $05;          (* BT                     *)
      Z_BLK       = $06;          (* Counter                *)
      T_BLK       = $07;          (* Timer                  *)
      MB_BLK      = $08;          (* Flag                   *)
      EB_BLK      = $09;          (* Input area             *)
      AB_BLK      = $0A;          (* Output area            *)
      PY_BLK      = $0B;          (* P-periphery            *)
      QY_BLK      = $0C;          (* Q-periphery            *)
      ABS_BLK     = $0F;          (* Absolute memory  *)
      FB_BLK      = $10;          (* FB                     *)
      FX_BLK      = $11;          (* FB in external memory  *)
      OB_BLK      = $12;          (* OB                     *)
      PB_BLK      = $13;          (* PB                     *)
      SB_BLK      = $14;          (* SB                     *)
```

2.3.3.11.4 Constants for the data type of block elements

```
CONST
      B_BLOCK     = $07;      (* Type: Block of bytes       *)
      W_BLOCK     = $17;      (* Type: Block of words       *)
      D_BLOCK     = $27;      (* Type: Block of long words  *)
```

2.3.3.11.5 Flag for request status

```
CONST
      REQ_WRKN    = $01;      (* Request being processed    *)
      REQ_UNDEF   = $02;      (* Undefined request status   *)
      REQ_NO_ERR  = $03;      (* Request compl. without err.*)
```

2.3.3.11.6 Constants for the element types of the process image

```
CONST
      Z_PA        = $06;      (* Counter                    *)
      T_PA        = $07;      (* Timer                      *)
      MB_PA       = $08;      (* Flag                       *)
      EB_PA       = $09;      (* Input area                 *)
      AB_PA       = $0A;      (* Output area                *)
      ABS_PA      = $0F;      (* absolute block in the PA   *)
```

2.3.3.11.7 Request type for CP__Call

```
CONST
      GET_INFO          = $00
      STATR_AG          = $20
      READ_AG           = $21
      WRITE_AG          = $21
      CNCLR_AG          = $28
      STAT_PA           = $70
      READ_PA           = $71
      HANDLE_Struct     = $7F       (* dx:bx = pointer to CP486_
                                       Parameter_Rec *)
```

2.3.3.11.8 Constants for error messages: page frame 2 and 7

```
CONST
```

`ERR_S5_TYP = $01;` (* invalid element type *)

An attempt was made to access the data of a PLC of the 115U series during a single element request with element types DX_SNG, BA_SNG, BB_SNG, BT_SNG or QB_SNG, or for a block element request with element types DX_BLK, BA_BLK, BB_BLK, BT_BLK or FX_BLK. These element types do not exist in this type of PLC.

Remedy: modify the "`type`" parameter in the function call to the PC application software.

`ERR_S5_BST = $02;` (* Module does not exist *)

An attempt was made to access a module that does not exist whilst using a single element request with element type DB_SNG or. a block element type DB_BLK.

Remedy: define a data module in the PLC or modify the "`bst`" parameter in the function call to the PC-application software.

`ERR_S5_ELM = $03;` (* Element does not exist *)

An attempt was made to access non-existing data in a DB for a single element request using element type DB_SNG or block element request with element type DB_BLK.

Remedy: increase the length of the data module in the PLC or modify the parameters "`adr`" or "`len`" in the function call to the PC application software.

An attempt was made to access timer or counter numbers > 127 using element types Z_SNG or T_SNG with a single element request.

Remedy: modify the "`adr`" parameter in the function call to the PC application program.

An attempt was made to access a flag with a number > 199 for an element size specified as byte and an element type MB_SNG, a number > 198 for an element size specified as word or a number > 196 for an element size specified as double word.

Remedy: check the value specified for the "`adr`" parameter in the function call to the PC application software.

An attempt was made to access the process image of the I/O area with a number > 127 for an element size specified as byte and an element type EB_ - or AB_SNG, a number > 126 for an element size specified as word or a number > 124 for an element size specified as double word.

Remedy: Check the value specified for the "`adr`" parameter in the function call to the PC application software.

An attempt was made to access the P-periphery with a number > 255 for an element size specified as byte, a number > 254 for an element size specified as word or a number > 252 for an element size specified as double word.

Remedy: check the value specified for the "`adr`" parameter in the function call to the PC application software.

`ERR_S5_SIZE = $04;` (* invalid element size *)

An attempt was made to access timers or counters with element type Z_SNG or T_SNG using a single element request and the parameter for the element size not set for word access (WORD_ELM).

Remedy: modify the "`size`" parameter in the function call to the PC application software.

An attempt was made to access a flag or an absolute address using element type MB_SNG or ABS_SNG with a single element request and the parameter for the element size RBYTE_ELM.

Remedy: modify the "`size`" parameter in the function call to the PC application software.

An attempt was made to access the inputs or the outputs in the process image using element type EB_SNG or AB_SNG with a single element request and the parameter for the element size SEMA_ELM or RBYTE_ELM.

Remedy: modify the "`type`" parameter in the function call to the PC application software.

An attempt was made to access the P periphery using element type PB_SNG and the parameter for element size set to BIT_ELM, SEMA_ELM or RBYTE_ELM.

Remedy: modify the "`type`" parameter in the function call to the PC application software.

An attempt was made to read from absolute addresses with a single element request and element type set to ABS_SNG as well as an the element size of SEMA_ELM. This type of access with absolute addressing is only permitted for write operations! Where single bits must be read the element size should be BIT_ELM.

Remedy: modify the "`type`" parameter in the function call to the PC application software.

`ERR_S5_BIT = $05;` (* Bit number too high *)

An attempt was made to access a flag or an absolute address bit with a single element request and an element type of MB_SNG or ABS_SNG as well as the element size BIT_ELM or SEMA_ELM and a bit number > 7 (15).

Remedy: modify the "`bit`" parameter in the function call to the PC application software.

An attempt was made to access an I/O bit with a single element request and element type EB_SNG or AB_SNG as well as a bit number > 7.

Remedy: modify the "`bit`" parameter in the function call to the PC application software.

`ERR_S5_STRT = $06;` (* invalid start address *)

An attempt was made at a block transfer by means of modules with a block element request and an element type "Baustein"_BLK, where the relative start address in the block is > 32767.

Remedy: modify the "`adr`" parameter in the function call to the PC application software.

`ERR_S5_LEN = $07;` (* illegal block length *)

An attempt was made at a block transfer with a length > 504 words and a block element request to any element type.

Remedy: modify the "`len`" parameter in the function call to the PC application software.

`ERR_S5_ADR = $08;` (* address too large *)

An attempt was made to access an address > FFFFh in a PLC of the 115U-series with a single or a block element request and element type ABS_SNG. These CPUs (through CPU 944) only have an addressing capacity of 64 KB.

Remedy: modify the "`adr`" parameter in the function call to the PC application software.

`ERR_S5_QVZ = $09;` (* QVZ/ADF in the PLC during read/write *)

An attempt was made to access an addressing area that does not exist.
This error message is provided by the PLCs of the 135 and the 155-series. A 115-series PLC would be placed into the stop condition.

Remedy: modify the "`type`" or. "`adr`" parameter in the function call to the PC application software.

`ERR_S5_944 = $0A;` (* CPU 944: module in prog.bank *)
An attempt was made to access a module that is not located in the DB bank using a block element request and an element type of "Baustein"_BLK.
(only relevant for the CPU 944 of the PLC-type 115U)

Remedy: install the PLC's module in the DB-bank (by means of BIB-No. 19285) or modify the function call to the PC application software.

2.3.4 Turbo-C Interface (2.0 and C++ from 1.0), Microsoft-C 6.0

C-language access to the COM driver is available by means of a library file that provides all the functions of the INT service interrupt. A C-function has been defined for every function of the driver. This C-function supplies parameters to registers, executes the call to the interrupt, and returns the necessary values. Thus, even those users that do not have the ability to write low-level programs for the CP may use all the facilities of provided by the driver.

The include file "CP486DEF.H" contains the definition of the data types and constants for element size, element types and error numbers. It also contains the ANSI-C prototypes for the following functions. The include file must appear in the application program.

All the required functions have been implemented in the file CP486LIB.C. When these are to be used in a program the file CP486LIB.OBJ must also be linked to the program. Depending on the programming environment and version this file must be entered into the project file (Turbo-C), the dependency list (Microsoft-C) or into the Make-file. The relevant details are available from the applicable manuals.

Note

In "CP486LIB.H" byte was defined as unsigned char and word as unsigned short.

2.3.4.1 CP-status request function

Data structures

```
int CP_info (p)
          typedef struct
          {
            word CP_id;                 /* Identifier: CP486 value = $C386  */
            byte VGA_ver, BIOS_ver;     /*  */
            byte DRV_ver;               /*  */
            byte CPU_AG;                /*  */
            byte CP_reg, S5_reg;        /*  */
          } CP486_InfoBlk;              /*  */
```

Data structure for the general status-info function, where the components are completed according to the values of the CP486.

This function calls the driver function "general status information". The function is preceded by an installation check for the driver. If the driver was not installed the function returns -1. If the driver was installed the function returns 0. The CP_id component always contains the identifier $C386, the other components are not affected.

2.3.4.2 Reading a single element from the PLC

```
int CP_read_AG (byte size, byte type, byte bst, unsigned long adr,
                byte bit);
```

Parameters

size	Element size (see chapter 2.3.4.11.1)
type	Element type single elements (see chapter 2.3.4.11.2)
bst	Module number
adr	Address in the module or absolute address
bit	Bit number

Returns

Request number or negative number for errors.

This function calls the driver function "Read a single element from the PLC". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11.

If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, the request returns the request number as function value.

Calling procedure

The following scheme should always be used to ensure that the driver function is processed properly, otherwise the page frame might be locked (see chapter 2.3.4.11):

```
int a_no;           /* Request number for the read request */
int stat;                /* current request status */
int time_count=4000;/* Timeout timer (= 4 seconds) */
byte wert ;         /* value read from the PLC */

/* Start the request */
a_no = CP_read_AG(LBYTE_ELM, DB_SNG, 10, 1, 0);

if(a_no < 0)        /* An error has occurred */
  printf("Request terminated by an error: %d\n", a_no);

else
{                         /* a_no contains the request number */
  do
  {
    delay(1);                           /* wait 1 ms */
    stat = CP_stat_AG(a_no, &wert);     /* Fetch requested status/data */
    /* until timer has expired or request was completed with or without error */
  } while((time_count-- > 0) && (stat == REQ_WRKN));

  switch(stat)
  {
    case REQ_NO_ERR:
      printf("Data: %d was read\n", wert);
    break;
    case REQ_UNDEF:
      printf("Undefined request status, Data: %d was read\n", wert);
    break;
    default:
      printf("Request completed with error: %d\n", stat);
  }
}
```

2.3.4.3 Reading a block from the PLC

```
int    CP_readn_AG    (byte    size,    byte    type,    byte    bst,
                       unsigned long adr, word len);
```

Parameters

size	Data type of the block elements (see chapter 2.3.4.11.4)
type	Element type block elements (see chapter 2.3.4.11.3)
bst	Module number
adr	Address in the module or absolute address
len	Length of data in words or bytes, depending on element type

Returns

Request number or negative number for errors

This function calls the driver function "Read a block from the PLC". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11. If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, the request returns the request number as function value.

Calling procedure

The following scheme should always be used to ensure that the driver function is processed properly, otherwise the page frame might be locked:

```
int a_no;                  /* Request number for the read request */
int stat;                  /* current request status */
int time_count=4000;/* Timeout timer (= 4 seconds) */
int buff[100];                    /* value read from the PLC */

/* Start the request */
a_no = CP_readn_AG(W_BLOCK, DB_BLK, 5, 10, 100);

if(a_no < 0)        /* An error has occurred */
  printf("Request terminated by an error: %d\n", a_no);

else
{                                 /* a_no contains the request number */
  do
  {
    delay(1);                              /* wait 1 ms */
    stat = CP_stat_AG(a_no, &buff);     /* Fetch requested status/data */
    /* until timer has expired or request was completed with or without error */
  } while((time_count-- > 0) && (stat == REQ_WRKN));

  switch(stat)
  {
    case REQ_NO_ERR:
      printf("Data was read\n"");
    break;
    case REQ_UNDEF:
      printf("Undefined request status \n");
    break;
    default:
      printf("Request completed with error: %d\n", stat);
  }
}
```

2.3.4.4 Writing a single element to the PLC

```
int CP_write_AG (byte size, byte type, byte bst, unsigned long adr,
                 byte bit, void far *p);
```

Parameter

size	Element size (see chapter 2.3.4.11.1)	
type	Element type single elements (see chapter 2.3.4.11.2)	
bst	Module number	
adr	Address in the module or absolute address	
bit	Bit number	
p	Pointer to write data in PCs memory	
	if element size is bit or byte	pointer to a byte
	if element size is word	pointer to a word
	if element size is double word	pointer to a double word

Returns

Request number or negative number for errors.

This function calls the driver function "Write a single element to the PLC". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11. If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, the request returns the request number as function value.

Calling procedure

The following scheme should always be used to ensure that the driver function is processed properly, otherwise the page frame might be locked (see chapter 2.3.4.11):

```
int a_no;           /* Request number for the read request       */
int stat;           /* current request status */
int time_count=4000;       /* Time-out timer (= 4 seconds) */
byte wert = 0x5A;          /* value read from the PLC */
/* Start the request */
a_no = CP_write_AG(LBYTE_ELM, DB_SNG, 10, 1, 0, &wert);
if(a_no < 0)                      /* An error has occurred */
  printf("Request terminated by an error: %d\n", a_no);
else
{                                 /* a_no contains the request number */
  do
  {
    delay(1);                           /* wait 1 ms */
    stat = CP_stat_AG(a_no, NULL);      /* Read the requested status */
    /* until timer has expired or request was completed with or without error */
  } while((time_count-- > 0) && (stat == REQ_WRKN));
  switch(stat)
    {
    case REQ_NO_ERR:
      printf("Data: %x written\n",      wert);
    break;
    case REQ_UNDEF:
      printf("Undefined request status \n");
    break;
    default:
      printf("Request completed with error: %d\n", stat);
    }
}
```

2.3.4.5 Writing a block to the PLC

```
int CP_writen_AG (byte size, byte type, byte bst, unsigned long adr,
                  word len, void far *p);
```

Parameters

size	Data type of the block elements (see chapter 2.3.4.11.4)
type	Element type block elements (see 2.3.4.11.3)
bst	Module number
adr	Address in the module or absolute address
len	Length of data in words or bytes, depending on element type
p	Pointer to write data block in PC memory

Returns

Request number or negative number for errors.

This function calls the driver function "Write a block to the PLC". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11.

If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, the request returns the request number as function value.

Calling procedure

The following scheme should always be used to ensure that the driver function is processed properly, otherwise the page frame might be locked:

```
int a_no;           /* Request number for the read request */
int stat;           /* current request status              */
int i;
int time_count=4000;       /* Timeout timer (= 4 seconds) */
int buff[100];             /* write data */
for(i = 0; i< 100; i++)
buff[i] = (byte)i;         /* assign default values to data buffer */

/* Start the request */
a_no = CP_writen_AG(B_BLOCK, DB_BLK, 5, 10, 100, &buff);

if(a_no < 0)        /* An error has occurred */
  printf("Request terminated by an error: %d\n", a_no);
else
{                                /* a_no contains the request number */
  do
  {
    delay(1);                    /* wait 1 ms */
    stat = CP_stat_AG(a_no, NULL);      /* Read the requested status */
    /* until timer has expired or request was completed with or without error */
  } while((time_count-- > 0) && (stat == REQ_WRKN));

  switch(stat)  {
  case REQ_NO_ERR:
    printf("Data was written\n");
  break;
  case REQ_UNDEF:
    printf("Undefined request status \n");
  break;
  default:
    printf("Request completed with error: %d\n", stat);
  }
}
```

2.3.4.6 Interrogate the status of a request

```
int CP_stat_AG (int r, void far *p);
```

Parameters

a_no	Request number of the request that must be interrogated	
p	Pointer to data or data block in PCs memory only for read requests with single or block elements	
	if element size is bit or byte	pointer to a byte
	if element size is word	pointer to a word
	if element size is double word	pointer to a double word
	if element size is block	pointer to buffer for data block

Returns

Request number or negative number for errors.

This function calls the driver function "Interrogate status for a request". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11. If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, the request returns the request status (see chapter 2.3.4.11.5).

2.3.4.7 Terminate all requests for a page frame

```
int CP_cncl_AG (int a_no);
```

Parameters

Identifier for page frame 2

$28 terminate all active requests of page frame 2

This function calls the driver function "Terminate all requests for a page frame". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11 If the driver detects an error, the respective error message (negative number) is returned as function value. If the function is processed without errors, i.e. all requests were terminates successfully, then the request returns 0.

2.3.4.8 Read process image status

```
byte CP_stat_PA ();
```

Returns

Process image counter

This function calls the driver function "Status request process image". The function returns the process image counter.

2.3.4.9 Read process image area

```
int CP_read_PA (byte type, word adr, word len, void far *p);
```

Parameters

type	Element type single elements (see chapter 2.3.4.11.2)
bst	Module number
adr	Address in the module or absolute address
len	Data length (bytes or words) depending on type
p	Pointer to data buffer in CP memory

Returns

Process image counter.

This function calls the driver function "Read an area of the process image". The function enters the parameters supplied into the registers. The parameters are described in chapter 2.3.4.11. If an error occurs when the function is being executed, the respective error message (negative number) is returned as function value. If the function is processed without errors, then the request returns the up to date value of the process image counter.

2.3.4.10 Standard function

```
unsigned far pascal CP_Call(CP486_PARAMETER far *cp);
typedef struct
{
unsigned char Elementtyp;     // Element type see chapter 2.3.4.11.2 or
                                 2.3.4.11.3.
unsigned char Auftrag;        // Type of request see chapter 2.3.4.11.7
unsigned char Bausteinnummer; // Number of the data module or 0 for flags,
                           outputs etc.
unsigned char Kennung;        // Element size or data type see chapter
                           2.3.4.11.1 or 2.3.4.11.4.
unsigned short Adresse;       // Start address in the element
unsigned short Len;           // Number of elements transferred
unsigned char far *ptr;       // 16:16 Pointer to data, DOS. Set ptr to 0 when
                              // using data in the structure.
unsigned char far *ptr_win;   // 16:16 Pointer to data, Windows. Set to 0
                              // when using 32bit flat model 32bit data.
unsigned short (far pascal *ConfirmFunction)(unsigned char Auftragsnummer);
                              // Reverse call function when request completes.
unsigned char Cpu;            // CPU Number in PLC 0 - 3, always 0 for CPU
unsigned char reserved;       // Reserved
unsigned short Fehler;        // Error code. 0 = no error
unsigned char Daten[1];       // Space required for data. used when
                              // ptr==0 and ptr_win==0.

}CP486_PARAMETER;
```

The above is the data structure for standard functions. The components are pre-set in accordance with the requirements of function that must be executed.

If available, the pointer ptr_win must, be a segment 'offset pointer'.

For stat_AG The element Len returns the length of the data area (call).

Once stat_AG completes without errors, the element Len contains the number of bytes read.

This function can be used in place of all previously described functions.

All write data for the functions write_AG or. writen_AG is entered into the respective ARRAY and the function CP_Call is executed.

The functions read_AG or. readn_AG return all data after the function CP_CALL has been completed (as function CP_stat_AG) in the respective ARRAY.

2.3.4.11 Constants

The following constants are have been predefined. It is recommended that these are used, as they are more readable and result in a clearer program. The resulting program can then easily be adapted to cater for changes in the COM driver.

2.3.4.11.1 Constants for element size

```
#define BIT_ELM     0x00      /* Bit                          */
#define SEMA_ELM    0x01      /* Bit as semaphore             */
#define BYTE_ELM    0x02      /* Byte                         */
#define LBYTE_ELM   0x02      /* Left byte of a word          */
#define RBYTE_ELM   0x03      /* Right byte of a word         */
#define WORD_ELM    0x04      /* Word                         */
#define DWORD_ELM   0x05      /* Double word                  */
#define BLOCK_ELM   0x07      /* Block                        */
```

2.3.4.11.2 Constants for element types of single elements

```
#define DB_BLK      0x00      /* DB                           */
#define DX_BLK      0x01      /* DB in external memory        */
#define BA_BLK      0x02      /* BA                           */
#define BB_BLK      0x03      /* BB                           */
#define BS_BLK      0x04      /* BS                           */
#define BT_BLK      0x05      /* BT                           */
#define Z_BLK       0x06      /* Counter                      */
#define T_BLK       0x07      /* Timer                        */
#define MB_BLK      0x08      /* Flag                         */
#define EB_BLK      0x09      /* Input area                   */
#define AB_BLK      0x0A      /* Output area                  */
#define PY_BLK      0x0B      /* P-periphery                  */
#define QY_BLK      0x0C      /* Q-periphery                  */
#define ABS_BLK     0x0F      /* Absolute memory              */
#define FB_BLK      0x10      /* FB                           */
#define FX_BLK      0x11      /* FB in external memory        */
#define OB_BLK      0x12      /* OB                           */
#define PB_BLK      0x13      /* PB                           */
#define SB_BLK      0x14      /* SB                           */
#define RB_FREE     0x0E      /* Code for request block free*/
```

2.3.4.11.3 Constants for the element types of block elements

```
#define DB_SNG      DB_BLK          /* DB                      */
#define DX_SNG      DX_BLK          /* DB in external memory */
#define BA_SNG      BA_BLK          /* BA                      */
#define BB_SNG      BB_BLK          /* BB                      */
#define BS_SNG      BS_BLK          /* BS                      */
#define BT_SNG      BT_BLK          /* BT                      */
#define Z_SNG       Z_BLK           /* Counter                 */
#define T_SNG       T_BLK           /* Timer                   */
#define MB_SNG      MB_BLK          /* Flag                    */
#define EB_SNG      EB_BLK          /* Input area              */
#define AB_SNG      AB_BLK          /* Output area             */
#define PY_SNG      PY_BLK          /* P-periphery             */
#define QY_SNG      QY_BLK          /* Q-periphery             */
#define ABS_SNG     ABS_BLK         /* Absolute memory         */
#define FB_SNG      FB_BLK          /* FB                      */
#define FX_SNG      FX_BLK          /* FB in external memory */
#define OB_SNG      OB_BLK          /* OB                      */
#define PB_SNG      PB_BLK          /* PB                      */
#define SB_SNG      SB_BLK          /* SB                      */
```

2.3.4.11.4 Constants for the data type of block elements

```
#define B_BLOCK     0x07      /* Type: Block of bytes       */
#define W_BLOCK     0x17      /* Type: Block of words       */
#define D_BLOCK     0x27      /* Type: Block of long words  */
```

2.3.4.11.5 Identifier for request status

```
#define REQ_WRKN    0x01      /* Request being processed    */
#define REQ_UNDEF   0x02      /* Undefined request status   */
#define REQ_NO_ERR  0x03      /* Request compl. without err */
```

2.3.4.11.6 Constants for element types of process images

```
#define Z_PA        0x06          /* Counter                 */
#define T_PA        0x07          /* Timer                   */
#define MB_PA       0x08          /* Flag                    */
#define EB_PA       0x09          /* Input area              */
#define AB_PA       0x0A          /* Output area             */
#define ABS_PA      0x0F          /* absolute block in PA  */
```

2.3.4.11.7 Request type for CP__Call

```
#define GET_INFO        = 0x00
#define STATR_AG        = 0x20
#define READ_AG         = 0x21
#define WRITE_AG        = 0x21
#define CNCLR_AG        = 0x28
#define STAT_PA         = 0x70
#define READ_PA         = 0x71
#define HANDLE_Struct   = 0x7F      /* dx:bx = pointer to CP486_
                                       Parameter_Rec */
```

2.3.4.11.8 Constants for error messages: page frame 2, 3 and 7

```
#define ERR_S5_TYP = 0x01;
```
/* invalid element type */

An attempt was made to access the data of a PLC of the 115U series during a single element request with element types DX_SNG, BA_SNG, BB_SNG, BT_SNG or QB_SNG or for a block element request with element types DX_BLK, BA_BLK, BB_BLK, BT_BLK or FX_BLK. These element types do nor exist in this type of PLC.

Remedy: modify the "`type`" parameter in the function call to the PC application software.

```
#define ERR_S5_BST = 0x02;
```
/* Module does not exist */

An attempt was made to access a module that does not exist whilst using a single element request with element type DB_SNG or. a block element type DB_BLK.

Remedy: define a data module in the PLC or modify the "`bst`" parameter in the function call to the PC-application software.

```
#define ERR_S5_ELM = 0x03;
```
/* Element does not exist */

An attempt was made to access non-existing data in a DB for a single element request using element type DB_SNG or block element request with element type DB_BLK.

Remedy: increase the length of the data module in the PLC or modify the parameters "`adr`" or "`len`" in the function call to the PC application software.

An attempt was made to access timer or counter numbers > 127 using element types Z_SNG or T_SNG with a single element request.

Remedy: modify the "`addr`" parameter in the function call to the PC application program

An attempt was made to access a flag with a number > 199 for an element size specified as byte and an element type MB_SNG, a number > 198 for an element size specified as word or a number > 196 for an element size specified as double word.

Remedy: check the value specified for the "`adr`" parameter in the function call to the PC application software.

An attempt was made to access the process image of the I/O area with a number > 127 for an element size specified as byte and an element type EB_ - or AB_SNG, a number > 126 for an element size specified as word or a number > 124 for an element size specified as double word.

Remedy: Check the value specified for the "adr" parameter in the function call to the PC application software.

An attempt was made to access the P-periphery with a number > 255 for an element size specified as byte, a number > 254 for an element size specified as word or a number > 252 for an element size specified as double word.

Remedy: check the value specified for the "`adr`" parameter in the function call to the PC application software.

```
#define ERR_S5_SIZE = 0x04; /* invalid element size */
```

An attempt was made to access timers or counters with element type Z_SNG or T_SNG using a single element request and the parameter for the element size not set for word access (WORD_ELM).

Remedy: modify the "`size`" parameter in the function call to the PC application software.

An attempt was made to access a flag or an absolute address using element type MB_SNG or ABS_SNG with a single element request and the parameter for the element size RBYTE_ELM.

Remedy: modify the "`size`" parameter in the function call to the PC application software.

An attempt was made to access the inputs or the outputs in the process image using element type EB_SNG or AB_SNG with a single element request and the parameter for the element size SEMA_ELM or RBYTE_ELM.

Remedy: modify the "type" parameter in the function call to the PC application software.

An attempt was made to access the P periphery using element type PB_SNG and the parameter for element size set to BIT_ELM, SEMA_ELM or RBYTE_ELM.

Remedy: modify the "type" parameter in the function call to the PC application software.

An attempt was made to read from absolute addresses with a single element request and element type set to ABS_SNG as well as an the element size of SEMA_ELM. This type of access with absolute addressing is only permitted for write operations! Where single bits must be read the element size should be BIT_ELM.

Remedy: modify the "type" parameter in the function call to the PC application software.

`#define ERR_S5_BIT = 0x05;` /* Bit number too high */

An attempt was made to access a flag or an absolute address bit with a single element request and an element type of MB_SNG or ABS_SNG as well as the element size BIT_ELM or SEMA_ELM and a bit number > 7 (15).

Remedy: modify the "bit" parameter in the function call to the PC application software.

An attempt was made to access an I/O bit with a single element request and element type EB_SNG or AB_SNG as well as a bit number > 7.

Remedy: modify the "bit" parameter in the function call to the PC application software.

`#define ERR_S5_STRT = 0x06;` /* invalid start address */

An attempt was made at a block transfer by means of modules with a block element request and an element type "Baustein"_BLK, where the relative start address in the block is > 32767.

Remedy: modify the "adr" parameter in the function call to the PC application software.

`#define ERR_S5_LEN = 0x07;` /* invalid block length */

An attempt was made at a block transfer with a length > 504 words and a block element request to any element type.

Remedy: modify the "len" parameter in the function call to the PC application software.

`#define ERR_S5_ADR = 0x08;` /* address too large */

An attempt was made to access an address > FFFFh in a 115U series PLC with a single or a block element request and element type ABS_SNG. These CPUs (through CPU 944) only have an addressing capacity of 64 KB.

Remedy: modify the "`adr`" parameter in the function call to the PC application software.

`#define ERR_S5_QVZ = 0x09;` /* QVZ/ADF in the PLC during read/write */

An attempt was made to access an addressing area that does not exist.
This error message is provided by the PLCs of the 135 and the 155-series. A 115-series PLC would be placed into the stop condition.

Remedy: modify the "`type`" or. "`adr`" parameter in the function call to the PC application software

`#define ERR_S5_944 = 0x0A;` /* CPU 944: module in prog.bank */

An attempt was made to access a module that is not located in the DB bank using a block element request and an element type of "Baustein"_BLK.
(only relevant for the CPU 944 of the PLC-type 115U)

Remedy: install the PLC's module in the DB-bank (by means of BIB-No. 19285) or modify the function call to the PC application software.

2.3.4.12 Storage of process image in page frame 7

The user may also access the process image directly. The following overview shows a map of page frame 7. This provides very quick access to the data:

```
Address in
page frame (hex)

Byte   0 Process image EB 0                    +
       .                                       |
       .                                       | 128 Byte PAE 0-127
       .                                       |
Byte 127 Process image EB 127                  +
Byte 128 Process image AB 0                    +
       .                                       |
       .                                       | 128 Byte PAA 0-127
       .                                       |
Byte 255 Process image AB 127                  +
Byte 256 Flag byte 0                           +
       .                                       |
       .                                       | 256 Byte flag 0-255
       .                                       |
Byte 511 Flag byte 255        +
Byte 512/513 Timer 0   (high/low)              +
       .                                       |
       .                                       | 128 Words timer 0-127
       .                                       |
Byte 766/767 Timer 127 (high/low)              +
Byte 768/769 Counter  0 (high/low)             +
       .                                       |
       .                                       | 127 Words counter 0-126
       .                                       |
Byte 1020/1021 Counter 126 (high/low)          +
Byte 1022 Counter byte 1)                      + Count byte
Byte 1023 Interrupt the CP
```

Note

All values pertaining to this page frame are refreshed when the handler module FB1/FB10 is called (provided that this operation was enabled in the formal operator of the handler module). The counter byte is incremented by the handler module every time the data is refreshed. This counter byte can be used by the PC to detect whether the data is valid and how often it was refreshed since the last read operation. Data is valid if the counter byte contains a number between 1...255. The counter is restarted from 1 when it exceeds the upper limit.

The handler module "Synchron" resets the running counter at address 3Feh of page frame 7. This indicates to the PC that the data in page frame 7 is currently not valid.

The PC must not delete this page frame if the counter byte (at address 3FEh of the page frame) contains a 0.

The handler module accesses this page frame for write operations only.

The handler module does not issue an interrupt when the data of page frame 7 must be refreshed.

2.4 Operation of the CP486COM in a WINDOWS environment

As of version 2.2 of the tool diskette a programming library containing the following files is available for MS-WINDOWS 3.1:

- The header file CP486WIN.H and the OBJ file CP486WIN.OBJ.
- The file CP486WIN.H contains the definitions required for the WINDOWS environment.
- The file CP486WIN.OBJ contains the communication functions for WINDOWS. These functions are accessed as described for DOS in chapter 2.3.4. (Exception: CP_stat_AG)

Changes to the calling structure of the function

CP_stat_AG: CP_stat_AG (byte r) where r = request number.

New functions

CP_init (void): defines a data area for communications via page frame 4 and returns a pointer to this area.

CP_exit (void): releases the data area. This command is obligatory at the end of the program.

Note

In addition to the entry in the CONFIG.SYS file, the page frame area must be excluded from WINDOWS memory management by means of an *EMMExclude* = ...entry into the [386Enh] section of the SYSTEM.INI file.!

This is always necessary when the CP486 is installed in a WINDOWS 3.1 environment as WINDOWS does not automatically exclude the page frame area.

The tool diskette 2.2 contains a sample for the operation in a WINDOWS environment.

3 Linkage with PLC by CP486NT

3 Linkage with PLC by CP486NT

3.1 General description

The data transfer between the CP486 and the PLC is controlled by means of handler modules on the PLC side and by means of software-interrupts on the CP-side. The following routines are available:

		Operation on the PLC-side	Operation on the CP-side
Page frame 2	CP-request: read/write data from/to PLC **(CP486 active)**	handler module is invoked cyclically (FB1)	Software interrupt for DOS call to driver for WinNT
Page frame 7	Transfer process image to CP	handler module is invoked cyclically (FB1)	Software interrupt or direct access to the page frame

Tab. 3-1: Routines

The following data structures in the PLC may be accessed from the CP:

- single elements of the type bit, byte, word and double word, DB, DX, BA, BB, BT, BS, flags, inputs, outputs, timers, counters
- Data blocks DB, DX, MB, T, Z, BA, BB, BT, BS, FB, FX, OB, PB, SB

The following functions are available from version 3.00 of the CPX86 (software CP4-SW593 version 3.00) and version 3.00 of the handler module (CP4-SW973 version 3.00).

The following description refers to the CPX86 program as COM-driver.

3.2 Installation of the page frame software

3.2.1 PLC-side: handler modules

The handler modules FB1 and FB2 must be loaded into the PLC to facilitate communications with the CP486. Handler module FB1 is started in OB1, and FB2 in the restart modules (OB20, OB21 and OB22).

Example for the call to FB1 in the OB1

```
          Baustein#OB1
          BIB
0000              ;SPA FB 1
          NAME    #CP-L/S
          ANSS    =KY 2,32
          PAA     =KF +0
          PAFE    =MB 99
0005              ;BE
```

Name	Format	Description
ANNS	KY	Number of requests
PAA	KF	Process image identifier
PAFE	MB	Flag byte for error messages

Tab. 3-2: Parameter list for starting FB1

ANSS

AN The maximum number of requests that should be processed in the page frame when the handler module is started

SS The number of the base page frame

PAA

Update the identifier of the process images in the page frame when the handler module is being started

= 0 process images must not be updated

≠ 0 process images will be updated

The value that is specified here is transferred to the process image and consists of the page frame number + 1. Valid page frame numbers range from 4 - 7.

PAFE

Error messages from the handler module

= 0 no error occurred

≠ 0 an error occurred. The error number is supplied in the PAFE-byte

1 The maximum number of requests that should be processed when a handler module is started is 0.

2 The maximum no. of requests that should be processed when a handler module is started is larger than 127.

3 The base page frame number is not divisible by 8

5 The page frame has not been synchronized by the CP.

9 Page frame for process image not located in a valid area.

Scratch pads used: MB200-MB255

Example for calling FB2 from OB21:

```
          Baustein#OB21
          BIB
0000              ;SPA FB 2
          NAME    #SYNCHRON
          SSNR    =KF +32
          WART    =KF +0
          PAA     =KF +7
          PAFE    =MB 98
0005              ;BE
```

Name	Format	Description
SSNR	KF	Number of the base page frame
WART	KF	Type of synchronization
PAA	KF	Process image identifier
PAFE	BY	Flag byte for error messages

Tab. 3-3: Parameter list for starting FB2

SSNR Base page frame number

WART
= 0 FB-SYNCHRON does not wait until the CP has synchronized every individual page frame

≠ 0 FB-SYNCHRON waits until the CP has synchronized every individual page frame

PAA Number of page frame where process image must be stored.
Valid range is between 4..7.

PAFE Error message from the handler module
= 0 no error occurred
≠ 0 an error was detected:
3 number of base page frame is not divisible by 8.

Scratch pads used: MB200-MB255

3.2.2 Various representations of data in memory

The different rules for representing words and double words (long words) in the CP and in the PLC must be met when data is transferred between the CP and the PLC.

Data words are stored differently in the CP than in the PLC. The positions of the most significant byte (high-byte) and the least significant byte (low-byte) have been swapped. In the case of double words, the sequence of all 4 bytes has been reversed. Where data is transferred between the PLC and the CP, the position of relevant bytes must be swapped at some time, as the transferred data would otherwise be invalid. Wherever possible, the COM driver will adjust the data automatically as required.

The driver will perform an automatic swap for all data transfers to/from page frame 2 and 7.

- Bytes are not modified during transfer.
- The most significant and least significant bytes of words are swapped during transfer.
- The sequence of all 4 bytes of a long data word is reversed during transfer.

Representation of data in the PLC

Address n	byte	representation byte
Address n	high-byte	representation word
Address n+1	low-byte	
Address n	high-byte high-word	representation double word
Address n+1	low-byte high-word	
Address n+2	high-byte low-word	
Address n+3	low-byte low-word	

Representation of data in the CP

Address n	byte	representation byte
Address n	low-byte	representation word
Address n+1	high-byte	
Address n	low-byte low-word	representation double word
Address n+1	high-byte low-word	
Address n+2	low-byte high-word	
Address n+3	high-byte high-word	

3.3 Operation of the CP486COM in a Windows-NT environment

As of version 2.2 of the tool diskette a programming library containing the following files is available for WINDOWS-NT 3.51:

Include-Files

CP486DEF.H	General definitions and prototypes
CPHWDEF.H	Definitions for CP-modules
DRVFCALL.H	Definition of the driver calls for DevIOCtrl
WMKTYPES.H	Definition of the type of calls for different compilers and operating systems

C-Files

CPTEST.C	Demo program for reading and writing from/to the page frame

Library files

CPWK.LIB	Import library with the exporting functions of the DLLs
CPWKNT.DLL	dynamic library files for NT

Page frame driver

CPWK.SYS	Page frame driver for Windows-NT 3.51

Operating system

CPWK.INI	Script file for regini.exe
REGINI.EXE	makes entries into the registration script of the NT system and must be used in conjunction with "CPWK.INI"

3.3.1 Installing the page frame driver into Windows-NT

Before the page frame driver is installed Windows-NT 3.51 must be installed on the VIPA 486-DX.

The following steps are necessary to register and activate the page frame driver under Windows-NT 3.51:

- Copy the file "CPWK.SYS" into the directory:

  `\WINNT35\SYSTEM32\DRIVERS`

- Copy the files "REGINI.EXE" and "CPWK.INI" into a directory you have created.
- You must register the driver with your NT-system by means of the program REGINI. Choose "Run" from your START menu and execute the file "REGINI.EXE" with the script-file "CPWK.INI" as command line parameter. REGINI installs the driver into your system.
- Restart your system to initialize the driver.
- Start the page frame driver by selecting the *Main group, Control panel, Hardware* and search the displayed list of hardware units for the entry:

  "VIPA Dual Port S5 Device Driver".

  Enter the required "starting-mode" for the page frame driver and start the driver. The "Status" column of the hardware panel now displays "started".

At this point the page frame driver is ready for operation and may be accessed from the applications by means of the function calls described below.

You may use the console application "CPTEST.EXE" to verify that the driver operates properly . Start the application via the *Main group* and the "MS-DOS-input request". Please note that the library file "CPWK.DLL" must be located in the same directory.

3.3.2 Microsoft-Visual C V2.0, V4.0 interface

C-language access to the COM driver is available by means of a library file that provides all the functions. A C-function has been defined for every function of the driver.

These functions may be used by applications running on the CP486 to read data from or write data to the PLC. Data is transferred by means of a structure that you must complete as required.

The include-file "CP486DEF.H" defines the data type and constants for element size, element types and error numbers. The function prototypes for the following functions are also defined here in ANSI-C format. The application program must contain a reference to the include-file.

All required functions are provided by the DLL file (CPWKNT.DLL). The DLL must be located in the same directory as your application. During compilation you must also link the import library "CPWK.LIB" to your application. Depending on the programming environment and version this file must be entered into the dependence-list (project list) or into the "make-file". Please refer to the respective manuals for details.

All the functions below require the type definitions specified in "WMKTYPES".

```
#define WENTRY_C        pascal
#define W_POINTER       *
```

3.3.2.1 CP-function initialization

```
unsigned  short  WENTRY_C CP_init (void)
```

Input parameters

none

Returns

0	no errors	
CP_NO_DRIVER	error, see chapter	3.3.4

This function must be executed before any other CP-functions are accessed. It checks whether the Windows-NT driver has been installed and then opens and initializes the device interface.

3.3.2.2 CP-function exit

```
void  WENTRY_C CP_exit (void)
```

Input parameters

none

Returns

none

This function closes the device-interface and must be called before the program is ended.

3.3.2.3 CP-function read

```
unsigned short WENTRY_C CP_startread (CP386_PARAMETER
         W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	pointer to a structure containing the entries about the read data for the PLC.

Returns

`0`	no error
<> `0`	system error
`cp → Fehler`	error, see chapter 3.3.4

This function starts a read operation. The respective variables of the structure for the read operation must be assigned as required before the function is executed (see chapter 3.3.3).

Structure variables

Input

Element type	`cp → Elementtyp`	single or block elements
Module number	`cp → Bausteinnummer`	only for module elements
Data type	`cp → Kennung`	Data type for single or block-elements
Length	`cp → Len`	Number of elements that must be read or the bit number for reading single bit elements

Result

Request number	`cp → Handle`	Unique request number
Status	`cp → Fehler`	Status of the connection

This function may be used to read a single element or a block element. The function initiates the request and immediately returns to the caller. Any data can only be retrieved by means of a call to the status function "CP_pollread".

Note

Read requests that have been completed are blocked to ensure that it is not overwritten by a new request before the returned data has been retrieved. Once a request has been started, its status must be interrogated until the request returns "completed with error" or "completed without error".

The status request returns the data at the end of the structure starting from "cp → Daten[0]". . If the status is not interrogated, the request remains locked and no further read requests can be started, even if all requests for the page frame have been completed.

3.3.2.4 CP-function read complete check

`unsigned short WENTRY_C` **`CP_pollread`** `(CP386_PARAMETER W_POINTER cp)`

Input parameters

`CP386_PARAMETER W_POINTER cp`	pointer to the structure containing the entries required for the data read from the PLC.

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

You can use this function to check whether the data requested by a read request is available. A "CP_startread" must always precede this function.

The function modifies a few variables in the respective structure and returns the requested data.

Structure variables

Status	`cp → Fehler`	connection status
Daten	`cp → Daten[0]`	this is where the requested data is returned

The function enters the data into the data buffer "cp → Daten" located at the end of the structure "CP386_PARAMETER". You must reserve a data buffer providing enough space for the expected data. You may specify the size of the data buffer as follows:

```
CP_386_PARAMETER *cp;
int bufferSize=1024;
.
.
cp=(CP386_PARAMETER *) malloc(sizeof(CP386_PARAMETER *) +
bufferSize));
```

Note

Individual bytes of data words and data blocks are swapped during the transfer.

The sequence of the 4 bytes of double words and data blocks is reversed.

3.3.2.5 CP-function stop a read operation

```
unsigned short WENTRY_C CP_stopread (CP386_PARAMETER W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	pointer to a structure containing entries about the data that must be read from the PLC.

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

This function terminates an active read request.

3.3.2.6 CP-function write

```
unsigned short WENTRY_C CP_startwrite (CP386_PARAMETER
         W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	pointer to a structure containing entries about the write data for the PLC.

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

This function starts a write request. The entries in the variables of the respective structure must be completed before the function is called. (see chapter 3.3.3).

Structure variables

Entry		
Element type	`cp → Elementtyp`	single or block elements
Module number	`cp → Bausteinnummer`	only for module elements
Data type	`cp → Kennung`	Data type for a single or a block element
Element offset	`cp → Adresse`	Offset of the element type
Length	`cp → Len`	Number of elements that must be written or the bit number for single bit elements
Data	`cp → Daten`	write data
Results		
Request number	`cp → Handle`	Unique request number
Status	`cp → Fehler`	Connection status

This function is used to write a single data element or a data block to the memory of the PLC. The function writes the respective value into the page frame and does not wait for the PLC to fetch the data, it returns to the caller immediately. For this reason the status can only be retrieved by means of the status function "CP_pollwrite".

Note

Once a write request has been started, its status must be interrogated until the request returns "completed with error" or "completed without error". Data is always returned from "p → Daten[0]" at the end of the structure. If the status is not interrogated, the request remains locked and no further requests can be started.

3.3.2.7 CP-function poll write complete

```
unsigned short WENTRY_C CP_pollwrite (CP386_PARAMETER
         W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	Pointer to a structure containing the required entries about the write data for the PLC.

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

This function checks whether the data for a write request has been retrieved or not. This function must always be preceded by a "CP_startwrite". The respective entries in the variables of the structure must have been completed (see chapter 3.3.3). The function supplies the write data to the Windows NT-driver and modifies the following variable in the structure.

Status	`cp → Fehler`	connection status

The function retrieves the data from the buffer "cp → Daten" located at the end of the structure "CP386_PARAMETER". You must reserve a data buffer providing enough space for the expected data. The data must also be available before the call to the function "CP_startwrite" is issued. You may specify the size of the data buffer as follows:

```
CP_386_PARAMETER *cp;
int bufferSize=1024;
.
.
cp=(CP386_PARAMETER *) malloc(sizeof(CP386_PARAMETER *) + bufferSize));
```

Note

The bytes of data words and data blocks are swapped during transfer. The 4 byte sequence of double word data and data blocks is reversed.

3.3.2.8 CP-function stop write operation

```
unsigned short WENTRY_C CP_stopwrite (CP386_PARAMETER
         W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	Pointer to a sstructure containing the write data for the PLC.

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

This function stops any currently active write request.

3.3.2.9 CP-function general call

```
unsigned short WENTRY_C CP_Call (CP386_PARAMETER W_POINTER cp)
```

Input parameters

`CP386_PARAMETER W_POINTER cp`	Pointer to the structure containing the entries in a structure of the type "CP386_PARAMETER"

Returns

`0`	no error
<> 0	System error
`cp → Fehler`	Error, see chapter 3.3.4

This function can be used in place of all the above function calls. You must supply the relevant function number (see chapter 3.3.4) as a request type before issuing the call.

Request types

The following request types are available for "cp → Auftrag":

`CP_START_READ`	Start read operation
`CP_POLL_READ`	Request read status and read data
`CP_START_WRITE`	Start write operation
`CP_POLL_WRITE`	Request write status and write data
`CP_GETKACHEL`	Read page frame information
`CP_SETKACHEL`	Set base page frame number

In addition to these functions you may also use the following to read the process image:

`READ_PA`	Read process image
`STAT_PA`	Read the status of the process image

Note

The process image function reads an area from the current process image. The length of the individual areas is checked when these are accessed, i.e. you can not request 4 bytes from EB126 as only 128 EB bytes are available.

The length for timers or counters is specified in words, all others are specified in bytes. The high and the low byte of timers and counters are also swapped during transfer to ensure that the data words may be processed properly in the CP.

Any required area spanning arbitrary partitions of the process image may be read if the type is specified as "ABS_SNG". The respective length is specified as bytes, even if the area includes timers or a counters. If a timer or counter area is read, the high and low byte are again swapped!

Element types for "cp → Elementtyp" and length for "cp → Len"

Z_SNG	Counter (length in words)
T_SNG	Timer (length in words)
MB_SNG	Flags (length in bytes)
EB_SNG	Inputs (length in bytes)
AB_SNG	Outputs (length in bytes)
ABS_SNG	Absolute access to process image (length in bytes)

3.3.2.10 Set CP-page frame parameter

```
unsigned short WENTRY_C CP_setkachel (KACHEL_STRUCT W_POINTER
         kachel)
```

Input parameters

`KACHEL_STRUCT W_POINTER kachel`	Pointer to a structure containing the entries for the page frame that must be set.

Returns

`0`	no error
<> 0	System error
`kachel → Fehler`	Error, see chapter 3.3.4

The structure must be completed before the function is called (see chapter 3.3.3). The base page frame number may be supplied in "KACHEL_STRUCT".

The base page frame number "kachel → Kachelbasis" for the CP4BG6x may be set to any value that is divisible by 8 and that lies between 0 and 240. For the CP4BG7x the base page frame number must be divisible by 16.

3.3.2.11 Read CP- page frame parameters

```
unsigned short WENTRY_C CP_getkachel (KACHEL_STRUCT W_POINTER
         kachel)
```

Input parameters

`KACHEL_STRUCT W_POINTER kachel`	Pointer to a structure that is used to store the data about the page frame.

Returns

`0`	no error
<> 0	System error
`kachel → Fehler`	Error, see chapter 3.3.4

Before this function is called the respective structure should be initialized to contain "0"'s. The function returns information about the page frame (see chapter 3.3.3).

3.3.3 Description of the structures

This section contains a description of the structures that are available as a data interface when the CP-functions are used. Type-definitions are contained in include-files CP486DEF.H and CPHWDEF.H. Please note the rules for the definitions that you should use to supply parameters to structures, as explained in chapter 3.3.4.

Extract from "CP486DEF.H":

```
typedef struct {
unsigned char Elementtyp;           //Accessible element types of the PLC
unsigned char Auftrag;              //Request type (only required for the
                                    //function "CP_call".
unsigned char Bausteinnummer;       //Number of the module, else 0 for flags,
                                    //outputs etc.
unsigned char Kennung;              //specifies the element width (bit, byte,
                                    //word, double word)
unsigned short Adresse;             //Start address in the element, for write
                                    //operations
unsigned short Len;                 //Number of elements transferred or
                                    //bit number for single element bit
unsigned char far *ptr;             //16:16 Realmode-pointer to data DOS.
                                    //set to NULL when using the data in the
                                    //structure "ptr"
unsigned char far *ptr_win;         //16:16 Realmode pointer to data
                                    //Windows. For 32 Bit Flat-Model set 32Bit
                                    //data to NULL
unsigned short far pascal*ConfirmFunction)
(unsigned char Auftragsnummer);
                                    //Recall function when request has completed
unsigned char Cpu                   //CPU Number in the PLC 0 - 3, always 0 for
                                    //a single CPU
unsigned char Handle;               //Handle for the request
unsigned short Fehler;              //Error code. 0 = no error, else see the
                                    //constants for error messages page frame 2
unsigned char Daten[1];             //Space for data. Used when
                                    //ptr==NULL and ptr_win==NULL
}CP386_PARAMETER;
```

The following structure elements can not be used for Windows-NT:

- *ptr
- *ptr_win
- *ConfirmFunction

Use the data buffer instead of *ptr or. *ptr_win. Ensure that "cp → Daten" is allocated as required by the read or write data (refer to the function description for "CP_pollread" and "CP_pollwrite").

Extract from "CPHWDEF.H":

```
typedef struct
{

  unsigned short Cb;                //Length of the structure, in bytes
                                    //(allocated internally)
  unsigned short Fehler;            //Error code 0=no error
  unsigned short Seriennummer;      //Serial number of the CP
  short Kachelbasis;                //Definition of base page frame
  short KachelAnzahl;               //Number of page frames
  short TreiberVersion;             //Driver version 100 -> 1.00
  shortCPTyp;                       //CP3, CP4, BG81, ..
  short KachelTyp;                  //Page frame, type

}KACHEL_STRUCT;
```

3.3.4 General definitions and definitions of errors

The following constants have been defined. It is recommended that these are used, as they are more readable and result in a clearer program. The resulting program can then easily be adapted to cater for changes in the page frame driver.

3.3.4.1 General definitions

Function numbers

must only be used for function call "CP_Call" and entered into "cp → Auftrag"

```
#define CP_START_READ         Start a read request
#define CP_POLL_READ          Start a read poll request
#define CP_START_WRITE        Start a write request
#define CP_POLL_WRITE         Start a write poll request
#define CP_GETKACHEL          Request to read page frame info
#define CP_SETKACHEL          Request to write page frame info
#define READ_PA               Read process image
#define STAT_PA               Read the status of the process image
```

Element types for single elements

must be entered into "cp → Elementtyp"

```
#define DB_SNG  0x00     //DB
#define DX_SNG  0x01     //DB in external memory
#define BA_SNG  0x02     //BA
#define BB_SNG  0x03     //BB
#define BS_SNG  0x04     //BS
#define BT_SNG  0x05     //BT
#define Z_SNG   0x06     //Counter
#define T_SNG   0x07     //Timer
#define MB_SNG  0x08     //Flag
#define EB_SNG  0x09     //Input area
#define AB_SNG  0x0A     //Output area
#define PY_SNG  0x0B     //P-periphery
#define QY_SNG  0x0C     //Q-periphery
#define ABS_SNG 0x0F     //Absolute memory
#define FB_SNG  0x10     //FB
#define FX_SNG  0x11     //FB in external memory
#define OB_SNG  0x12     //OB
#define PB_SNG  0x13     //PB
#define SB_SNG  0x14     //SB
```

Element types for block elements
must be entered into "cp → Elementtyp"

```
#define DB_BLK          0x00        //DB
#define DX_BLK          0x01        //DB in external memory
#define BA_BLK          0x02        //BA
#define BB_BLK          0x03        //BB
#define BS_BLK          0x04        //BS
#define BT_BLK          0x05        //BT
#define Z_BLK           0x06        //Counter
#define T_BLK           0x07        //Timer
#define MB_BLK          0x08        //Flag
#define EB_BLK          0x09        //Input area
#define AB_BLK          0x0A        //Output area
#define PY_BLK          0x0B        //P-periphery
#define QY_BLK          0x0C        //Q-periphery
#define ABS_BLK         0x0F        //Absolute memory
#define FB_BLK          0x10        //FB
#define FX_BLK          0x11        //FB in external memory
#define OB_BLK          0x12        //OB
#define PB_BLK          0x13        //PB
#define SB_BLK          0x14        //SB
```

Data type for single elements
must be entered into "cp → Kennung"

```
#define BIT_ELM          0x00      //Bit
#define BYTE_ELM         0x02      //Byte
#define LBYTE_ELM        0x02      //left byte of a word
#define RBYTE_ELM        0x03      //right byte of a word
#define WORD_ELM         0x04      //Word
#define DWORD_ELM        0x05      //Double word
#define BLOCK_ELM        0x07      //Block
```

Data type for block elements
must be entered into "cp → Kennung"

```
#define B_BLOCK         0x07        //Type: Block of bytes
#define W_BLOCK         0x17        //Type: Block of words
#define D_BLOCK         0x27        //Type: Block of long words
```

Identifier for request statuses
These messages are returned in "p → Fehler" by a request that is interrogated using a poll-function

```
#define REQ_NO_ERR      0x00        //Request compl. without error
#define REQ_WRKN        0x01        //Request being processed
#define REQ_UNDEF       0x02        //Undefined request status
```

3.3.4.2 Error definitions

General error definition:

"CP_init" function only when an error has occurred

```
#define CP_NO_DRIVER     1    //NT-driver not installed
```

Error messages for page frame 2 and 7:
returned in "p → Fehler"

General

```
#define ILL_TYPE     100  //Illegal element type
#define ERR_LEN      101  //Incorrect length
#define LL_ELMSZ     102  //Illegal element size
#define CPU_ERR      103  //Element type not valid for this CPU
#define ERR_KFULL    104  //Page frame full
#define ERR_COORD    105  //Page frame access not possible
#define ERR_BLKD     106  //Page frame locked
#define ERR_REQNR    107  //Incorrect request number
#define ERR_DPTR     108  //Invalid source-/destination data
                          //pointer
#define ERR_NOREQ    109  //Request not being processed
#define ERR_ILL_CALL 110  //Illegal function call
```

PLC-specific

These messages are returned by a request that is interrogated by means of a poll function when an error has occurred

```
#define ERR_S5_TYP 0xFF01
```
// illegal element type

An attempt was made to access the data of a PLC of the 115U series during a single element request with element types DX_SNG, BA_SNG, BB_SNG, BT_SNG or QB_SNG, or for a block element request with element types DX_BLK, BA_BLK, BB_BLK, BT_BLK or FX_BLK. These element types do not exist in this type of PLC.

Remedy: modify the structure variable "Elementtyp" before calling the function in the application.

```
#define ERR_S5_BST 0xFF02
```
// Module does not exist

An attempt was made to access a module that does not exist whilst using a single element request with element type DB_SNG or. a block element type DB_BLK.

Remedy: define a data module in the PLC or modify the structure variable "Bausteinnummer" before calling the function in the application.

`#define ERR_S5_ELM 0xFF03` // Element does not exist

An attempt was made to access non-existing data in a DB for a single element request using element type DB_SNG or block element request with element type DB_BLK.

Remedy: increase the length of the data module in the PLC or modify the structure variables "Adresse" or "Len" before calling the function in the application software.

An attempt was made to access timer or counter numbers > 127 using element types Z_SNG or T_SNG with a single element request.

Remedy: modify the parameter "Adresse" before calling the function in the application program.

An attempt was made to access a flag with a number > 199 for an element size specified as byte and an element type MB_SNG, a number > 198 for an element size specified as word or a number > 196 for an element size specified as double word.

Remedy: check the structure variable "Adresse" before calling the function in the application.

An attempt was made to access the process image of the I/O area with a number > 127 for an element size specified as byte and an element type EB_ - or AB_SNG, a number > 126 for an element size specified as word or a number > 124 for an element size specified as double word.

Remedy: check the structure variable "Adresse" before calling the function from the application.

An attempt was made to access the P-periphery with a number > 255 for an element size specified as byte, a number > 254 for an element size specified as word or a number > 252 for an element size specified as double word.

Remedy: check the structure variable "Adresse" before calling the function from the application.

`#define ERR_S5_SIZE 0xFF04` // illegal element size

An attempt was made to access timers or counters with element type Z_SNG or T_SNG using a single element request and the structure variable "Kennung" was not set for word access (WORD_ELM).

Remedy: modify the structure variable "Kennung" before calling the function from the application.

An attempt was made to access a flag or an absolute address using element type MB_SNG or ABS_SNG with a single element request and the structure variable "Kennung" containing RBYTE_ELM.

Remedy: modify the structure variable "Kennung" before calling the function in the application.

An attempt was made to access the inputs or outputs of the process image using element type EB_SNG or AB_SNG and the structure variable "Kennung" containing SEMA_ELM or RBYTE_ELM.

Remedy: modify the structure variable "Elementtyp" before calling the function in the application.

An attempt was made to access the P periphery using element type PB_SNG and the structure variable "Kennung" set to BIT_ELM, SEMA_ELM or RBYTE_ELM.

Remedy: modify the structure variable "Elementtyp" before calling the function in the application.

An attempt was made to read from absolute addresses with a single element request and element type set to ABS_SNG as well as an the element size of SEMA_ELM. This type of access with absolute addressing is only permitted for write operations! Where single bits must be read the element size should be BIT_ELM.

Remedy: modify the structure variable "Elementtyp" before calling the function in the application.

`#define ERR_S5_BIT 0xFF05` // Bit number too high

An attempt was made to access a flag or an absolute address bit with a single element request and an element type of MB_SNG or ABS_SNG as well as the element size BIT_ELM or SEMA_ELM and a bit number > 7 (15).

Remedy: modify the structure variable "Len" before calling the function in the application.

An attempt was made to access an I/O bit with a single element request and element type EB_SNG or AB_SNG as well as a bit number > 7.

Remedy: modify the structure variable "Len" before calling the function in the application.

`#define ERR_S5_STRT 0xFF06` // invalid start address

An attempt was made at a block transfer by means of modules with a block element request and an element type "Baustein"_BLK, where the relative start address in the block is > 32767.

Remedy: correct the structure variable "Adresse" before calling the function from the application.

`#define ERR_S5_LEN 0xFF07` // illegal block length

An attempt was made at a block transfer with a length > 504 words and a block element request to any element type.

Remedy: modify the structure variable "Len" before calling the function in the application.

`#define ERR_S5_ADR 0xFF08` // address too large

An attempt was made to access an address > FFFFh in a PLC of the 115U-series with a single or a block element request and element type ABS_SNG. These CPUs (through CPU 944) only have an addressing capacity of 64 KB.

Remedy: correct the structure variable "Adresse" before calling the function from the application.

`#define ERR_S5_QVZ 0xFF09` // QVZ/ADF in the PLC during read/write

An attempt was made to access an addressing area that does not exist.

This error message is provided by the PLCs of the 135 and the 155-series. A 115-series PLC would be placed into the stop condition.

Remedy: correct the structure variable "Elementtyp" or "Adresse" before calling the function from the application.

`#define ERR_S5_944 0xFF0A` // CPU 944: module in prog.bank

An attempt was made to access a module that is not located in the DB bank using a block element request and an element type of "Baustein"_BLK. (only relevant for the CPU 944 of the PLC-type 115U)

Remedy: install the PLC's module in the DB-bank (by means of BIB-No. 19285) or modify the function call in the application.

#define ERR_S52SHRT 0xFF0B // Area too small

#define ERR_S5_BITSIZE 0xFF0C // Length of transfer for bit elements not 1

3.3.5 Sample program

The following example illustrates the use of these functions. Any information for the page frame driver is read and displayed first. Then the structure elements for reading from or writing to the page frame are initialized. The purpose is to read or write 10 words from/to the data module in DB10 starting at word 0. The first data byte is modified before the data is written.

```
#include <windows.h>
#include <cp486def.h>
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
  unsigned short *db;
  KACHEL_STRUCT   ks;
  CP386_PARAMETER *cp;

  if(!CP_init())                // initialize CP-functions
  {
    ks.Cb = sizeof(KACHEL_STRUCT);

    if(!CP_getkachel(&ks))      // read page frame info
    {
      printf("Driver version %d, page frame start %d,number %d, serial number %d\n",
        ks.driver version,ks.base page frame,ks.Number of page frames,ks.serial number);
    }
    else
      printf("No page frame driver");

    if(cp=(CP386_PARAMETER*)malloc(sizeof(CP386_PARAMETER*) +1024))
    {                                   // Cp-Structure initialized
      cp->Elementtyp=DB_BLK;            // Data module-data block
      cp->Bausteinnummer=10;            // Module number 10
      cp->Kennung=W_BLOCK;              // Word-block
      cp->Adresse=0;                    // from offset 0 in the DB
      cp->Len=10;                       // read 10 words
      cp->Cpu=0;                        // start read request

      db=(unsigned short*)cp->Daten;

      if(!CP_startread(cp))        // start read request ->  0 OK,  != 0 System error
      {
        printf("Read started\n");
        if(cp->Fehler==0) //cp->Fehler is initialized to 0 by start function
        {
          printf("Handle number: %d\n",cp->Handle);

          while(!CP_pollread(cp))      //  0 OK,  != 0 System error
          {
            if(cp->Fehler==REQ_NO_ERR) // cp->Fehler =  REQ_NO_ERR if data available
            {
              printf("Data read : %d %d %d\n",db[0],db[1],db[2]);
              break;
            }
            else
```

```
            {                   // REQ_WRKN, REQ_UNDEF or PLC-specific error
              printf("CpFehler = %d\n",cp->Fehler);
              Sleep(20);        // wait 20 ms and try again
            }
          }
        }
        else
          printf("CpFehler = %d\n",cp->Fehler);  // general error (100-110)
      }
      else
        printf("No start-read\n");

      (*db)++;                // increment data[0]

      if(!CP_startwrite(cp))  // start write ->  0 OK,  != 0 System error
      {              // cp->Fehler is initialized to 0 by the start function
        printf("Write started\n ");
        if(cp->Fehler==0)
        {
          printf("Handle number %d \n",cp->Handle);
          while(!CP_pollwrite(cp))     //  0 OK,  != 0 System error
          {
            if(cp->Fehler == REQ_NO_ERR)
            {
              printf("Data written! \n");
              break;
            }
            else
              // REQ_WRKN, REQ_UNDEF or PLC-specific error
              printf("CpFehler = %d\n",cp->Fehler);
          }
        }
        else
          printf("CpFehler = %d\n",cp->Fehler);  // general error (100-110)
      }
      else
        printf("No Start-write\n");

       CP_exit();
    }
    else
      printf("No CP486\n");
  }
  else
    printf("No Init");
}
```

4 MS-DOS-Utilities for Solid-State disk operations

4 MS-DOS-Utilities for solid-state disk operations

4.1 Solid-state disk driver

The BIOS contains the standard driver functions for the control of standard hardware components. Any additional hardware or modified hardware must also be serviced by drivers. MS-DOS has provisions for adding hardware drivers as required to avoid having to change the BIOS for every change to the system hardware. This is achieved by means of an entry into the CONFIG.SYS file.

The CP3/4 provides a number of different solid-state disks.

- Chip-based solid-state disk (IC3, IC4):
  Die chip-based solid-state disk has a capacity of 256KB or. 1MB. The chip-based solid-state disk may be fitted with SRAM, EPROM, and PEROM.
- Memory-based solid-state disk boards
  Memory-based boards are available with capacities of 128KB, 512KB and 1MB. Cards are fitted with OTP-ROM- or SRAM (up to board revision V17).
- Auxiliary solid-state disk board
  Different auxiliary boards with capacities up to 7MB and fitted with FLASH-, SRAM- and EPROM are available.

The driver for the respective solid-state disk must be installed into the system file CONFIG.SYS to make the disk available as a physical drive.

Different drivers are available:

SDRAM.SYS	Driver for SRAM-based solid-state disks This driver provides read and write functions for solid-state disks.
SDROM.SYS	Driver for solid-state disks using EPROM's, FLASH-PROM's and OTP-ROM's This driver can only be used to read from the solid-state disk.
SDPEROM.SYS	Driver for solid-state disks using EPROM's You can read from or write to this solid-state disk. The useful life of the EEPROM's depends mainly on the number of write cycles (1.000 to 10.000 , depending on the type). For this reason we recommend that you install these drivers only for a short period of time when the data is written to the EEPROM's or when it is being modified. Subsequently the SDROM.SYS driver should be used to avoid further write cycles.

The respective driver is installed into the CONFIG.SYS file as follows:

```
device=[d:][path]SDxxx.SYS[comment][/bb][comment]
       [/llll][comment][/pp]
```

Parameter description

`[c:][path]` specifies the drive and the driver SDxxx.SYS, where
xxx defines the type of memory used for the solid-state disk. The following three drivers are available, SDRAM, SDROM and SDPEROM.

`[comment]` any text which must not contain the slash "/" or the enter key.

`/bb` base address of the solid-state disk (64 KB steps)
(e.g. /C0, corresponding to the physical start address C00000h).

`/llll` size of the solid-state disk (1 KB steps)
(e.g. /256, corresponding to a disk capacity of 256 KB = 262144 byte).

`/pp` parameter for SDPEROM.SYS: block size of the PEROM's
64 byte for AT29MC010 IC's
128 byte for AT29C010 and for AT29MC040 IC's
256 byte for AT29MC040 IC's

Examples:

Solid-state disk using 1MB EPROM's

```
DEVICE = \DEVICE\sdrom.sys base-address=/c0 size=/1024
```

Explanation The driver is located in the subdirectory "DEVICE" located on the boot drive. The solid-state disk uses read-only memory. For this reason the SDROM.SYS driver was selected. The base address of the solid-state disk is C00000h. The size of the solid-state disk is 1024KB = 1 MByte.

Solid-state disk using 256KB SRAM

```
DEVICE = \DEVICE\sdram.sys base-address =/c0 size=/256
```

Explanation The driver is located in the subdirectory "DEVICE" located on the boot drive. The solid-state disk uses read-write memory. For this reason the SDRAM.SYS driver was selected. The base address of the solid-state disk is C00000h. The size of the solid-state disk is 256KB.

Memorycard solid-state disk using 1MB ROM:

```
DEVICE = \DEVICE\sdrom.sys base-address=/80 size=/1024
```

Explanation The driver is located in the subdirectory "DEVICE" located on the boot drive. The solid-state disk uses read-only memory. For this reason the SDROM.SYS driver was selected. The base address of the memorycard slot is 800000h. The size of the solid-state disk is 1024KB = 1 MByte.

Memorycard solid-state disk using 128KB SRAM:

```
DEVICE = \DEVICE\sdram.sys base-address=/80 size=/128
```

Explanation The driver is located in the subdirectory "DEVICE" located on the boot drive. The solid-state disk uses read-write memory. For this reason the SDRAM.SYS driver was selected. The base address of the memorycard slot is 800000h. The size of the solid-state disk is 128KB.

Note

If the solid-state disk was previously activated by means of the VIPA-SETUP, it does not have to be installed in the CONFIG.SYS file. In this case the drive letter A: was already assigned to the disk. Any additional entry in the CONFIG.SYS file would assign a second drive letter.

It is possible to install a number of solid-state disks by repeating the entries in the CONFIG.SYS file with the respective modifications to the parameters. In this way a contiguous block of memory can be divided into a number of logical solid-state disk drives.

The operating system installs all the drivers in the CONFIG.SYS file. Every storage device receives a drive letter (e.g. D: or E: etc.). These drive letters are displayed on the screen during the boot process of the system.

Every virtual drive extends the resident part of DOS by some 900 bytes which are required for the driver.

Solid-state SRAM disks must be initialized by means of the FORMATSD program. Where the RAM-disk has a battery backup, this operation is only required once, otherwise the formatting operation must be performed each time the system is turned on. RAM-disks with battery backup will retain data for a certain period of time after the system has been turned off.

RAM-disks with battery backup are bootable disks if an operating system is installed on the disk.

4.2 Formatting program for the SRAM solid-state disk

SRAM solid-state disks, much like a normal diskette,must be formatted before they may be used as to store any data. The solid-state disk is formatted by means of the FORMATSD.EXE program. In MS-DOS this program is executed as follows:

```
[c:][path]FORMATSD d:[/D:xx][/S]
```

These parameters are described below

`[c:][path]`	drive and path that contains the FORMATSD program
`d:`	drive letter for the solid-state disk drive that must be formatted
`/D:xx`	this parameter defines how much space should be reserved for the directory. xx may have any value between 1 and 99. If this parameter was omitted the default for 64 directory entries is used.
`/S`	this parameter copies the system files from the MS-DOS boot drive to the solid-state disk. This parameter is required if the new solid-state disk should be bootable.

This program erases all files from the specified drive!

The FORMATSD program must be executed before any other program or system command (e.g. DIR) can access the solid-state disk drive. If this is not done, it may not be possible to format the solid state disk to the required disk size.

4.3 Solid-state disk generator

The SDGEN program generates the binary files for EPROM's, FLASH-PROM's and PEROM's of the memory. This program is executed as follows:

```
[c:][path] SDGEN
```

The program requests the following parameters:

EPROM-size in bit (0, 512 ,1M, 2M, 4M, 8M):

Here you must specify the size of the EPROM's (e.g. 1M for a 27C010 EPROM). The program creates files of the required size for an EPROM-programmer. If the parameter is specified as 0, the program generates a single file as required by SDLOAD.

gesplittet (J/N) splitted files are required for 16 bit solid-state disks. The solid-state disk of the CP3/4 PC is a 16 bit disk, the memorycard is an 8 bit wide disk. The 16 bit solid-state disk always contains 2 EPROM's (odd and even byte) connected in parallel.

Quellaufwerk source disk for the solid-state disk

Zieldateiname destination file name for binary files used for programming the EPROM's.

Depending on the size of the EPROM's the program will generate a number of files with the specified destination name. A sequential number is assigned to the extension of these files (filename.xxx). ODD and EVEN portions of splitted files are identified by an O or an E in the first letter of the extension (filename.Oxx or. filename.Exx).

Note

When a bootable solid-state disk must be created by means of the MS-DOS-RAM-disk RAMDRIVE.SYS, the label must first be removed from the RAM-disk. This can be done using the system program LABEL. Then you must copy the operating system files (IO.SYS, MSDOS.SYS and COMMAND.COM) to the empty RAM-Disk. Now you may use the SDGEN program to transfer the remaining files.

Before you transfer any data by means of SDLOAD you must first generate one large file by means of SDGEN which can then be loaded using SDLOAD.

All direct transfers of MS-DOS-Ramdisks by means of SDLOAD require an intermediate file.

4.4 Solid-state disk loader

The SDLOAD.EXE program is available for loading data records into the solid-state disk. This loader must be used for FLASH-PROM's and for PEROM's. FLASH-PROM's can only be erased completely before they are reloaded completely using this program. PEROM's can only withstand a rather limited number of write cycles. If you were to use a standard DOS copy program to transfer the data, certain sectors of the directory and elsewhere would be rewritten repeatedly. This would reduce the useful life of these components drastically. In this case a complete data transfer is advisable as every sector is only accessed once.

In MS-DOS the SDLOAD program is started as follows:

```
[c:][path]SDLOAD
```

where `[c:][path]` specify the drive and the path for the SDLOAD program.

Once the program is started the following list of components is displayed on the screen:

```
The following FLASH-PROM's or EEPROM's/PEROM's may be programmed:
        1  Am28F010-150,P28F010-150
        2  Am28F020-150,P28F020-150
        3  Am28F040-150,P28F040-150
        4  AT28C010-150
        5  AT28C040-150
        6  AT29C010-150
        7  AT29C040-150
        8  AT29MC010-150
        9  AT29MC040-150
Please enter the number of the ROM-type you are using:
```

The respective components must be selected by entering the relevant number and pressing the enter key.

You must then enter the quantity of components:

```
Please enter the quantity of the above components (2, 4, 6 or 8):
```

Here you must enter the number of IC's and press the enter key.

This is followed by the request for the base address for the solid-state disk board. This is entered as a hexadecimal address:

Possible values are : 800000
840000
880000
...
FC0000

You may now transfer the contents of a file that was generated by means of SDGEN or the contents of a logical drive into the solid-state disk. In the case of a file the respective file name must be entered. Where the data from a drive must be transferred the drive letter must be entered.

File names must be entered as follows: `[d:][path]file-name`
Drive letters must be entered as: `d:`

Files must have been created by means of SDGEN and must contain the entire contents of the drive as a single file. These files may have a name like: SDISK.000

The capacity of the destination drive mustbe equal to or higher than the size of the source file or the source drive.

SDLOAD transfers the contents of the drive or the specified file without modifications. A drive that is not bootable will not become bootable after the transfer by means of SDLOAD. If this is required you will first have to create a file using SDGEN.

4.5 Sample applications for the solid-state disk

4.5.1 Implementing a solid-state disk

Purpose to create an SRAM solid-state disk at a base address C00000h and with a size of 256KB (consisting of 2 x 128KB SRAM's (1MBit SRAM's)).

The board is fitted with 2 SRAM's of 128KB each and the jumpers are adjusted accordingly.

The system is booted from the fixed disk (drive C:). You require the programs SDRAM.SYS, FORMATSD.EXE (located in the subdirectory C:\SD) and a text editor for these operations.

Enter the following lines at the end of the C:\CONFIG.SYS file:

```
DEVICE = C:\SD\SDRAM.SYS  base=/C0 size=/256
```

Reboot the system by simultaneously depressing the keys <CTRL>, <ALT> and <DEL>. The system will return the following messages during the boot procedure:

SILICON DISK installed as drive D: . Vx.x Date RAMDISK FROM ADDRESS C00000h

The drive is now available and only needs to be formatted by means of the following command:

```
C:\SD\FORMATSD D:  /D:32  /S
```

The FORMATSD.EXE program installs a DOS file structure on drive D: (SRAM-disk). The directory provides space for 32 entries. Once the program has formatted the RAM-disk without errors it may be used as MS-DOS-drive. The /S parameter transfers the system files to the SRAM-disk so that the SRAM-disk may be used as boot drive.

4.5.2 Implementing a FLASH-PROM solid-state disk

Purpose to create a FLASH-PROM-solid-state disk at a base address C00000h and with a size of 1MByte (consisting of 4 x 256KB FLASH-PROM's (2MBit FLASH-PROMS).

Here we use a solid-state disk board with 4 x 256KB FLASH-PROM's (2MBit FLASH-PROM's) and 2 x 512KB SRAM's (4MBit SRAM's). The base address of the FLASH-PROM-disk is set to C00000h and the size to 1MB. Die base address of the SRAM-disk is set to 800000 h and it size is also set to 1MB.

The system is booted from the fixed disk (drive C:). You require the programs SDROM.SYS, SDRAM.SYS, FORMATSD.EXE and SDLOAD.EXE (located in the subdirectory C:\SD) and a text editor.

Enter the following lines at the end of the C:\CONFIG.SYS file:

```
DEVICE = C:\SD\SDRAM.SYS base=/80 size=/1024
DEVICE = C:\SD\SDROM.SYS base=/C0 size=/1024
```

Reboot the system by simultaneously depressing the keys <CTRL>, <ALT> and <DEL>. The system will return the following messages during the boot procedure:

```
RAM-DISK installed as drive D: .  Vx.y  Date
RAMDISK LOCATD AT 80 0000H

RAM-DISK installed as drive E:.  Vx.y  Date
ROMDISK LOCATED AT C0 0000H
```

An image of program memory is created on drive D: (SRAM-disk). For this purpose the SRAM-disk must first be formatted by means of the command

```
SD\FORMATSD D: /D:32 /S
```

Then you must copy all the files required on the FLASH-PROM-disk to drive D: . This concludes the creation of the FLASH-PROM-disk image and it may now be tested. For this purpose the solid-state disk located at a base address 80h 0000h must be selected in setup.

The program SDLOAD is used to transfer the image to the FLASH-PROM-disk using the following parameters:

```
C:\SD\SDLOAD
```

```
The following FLASH-PROM's or EEPROM's/PEROM's may be programmed:
        1  Am28F010-150,P28F010-150
        2  Am28F020-150,P28F020-150
        3  Am28F040-150,P28F040-150
        4  AT28C010-150
        5  AT28C040-150
        6  AT29C010-150
        7  AT29C040-150
        8  AT29MC010-150
        9  AT29MC040-150
Please enter the number of the ROM-type you are using: 2
Please enter the quantity of the above components (2,4,6,8): 4

Please enter the base address that you have set up on the solid-state disk
board. This is entered as a hexadecimal value: C00000

You may either transfer the contents of a file that was generated by means of
SDGEN or the contents of a logical drive to the solid-state disk.
Enter the file name or the drive letter: D:
```

The contents of drive D: is transferred into the FLASH-PROM's. If the program terminates without errors, the FLASH-PROM disk is available as drive E: . Both drives D: and E: have identical contents, however, drive E: is write protected. If you change the BIOS settings so that the solid-state disk at address C0h 0000h is the boot disk, then the system may be booted from the solid-state disk.

4.5.3 Creating program memory using EPROM's

Purpose to create an EPROM-solid-state disk consisting of 2 x 512KB EPROM's.

Option 1: by means of a diskette

For this operation you require a formatted and bootable diskette (format a: /s). The disk must be formatted on a computer that is running the same operating system that will be installed on the solid-state disk. Next you must also transfer all the files that should be loaded into the solid-state disk to the diskette. You must ensure that the 1,44MB diskette contains only those files that are definitely required for the solid-state disk. This master is then used to create the files for the 2 EPROM's. For this purpose you must start the program SDGEN on your development computer and enter the following parameters, provided that the diskette with the data for the solid-state disk is located in drive A:

```
C:\SD\SDGEN
EPROM-size in bit (0,512,1M,2M,4M,8M)          : 4M
SPLIT IN ODD-EVEN (J/N)                        : J
SOURCE DRIVE (A: .... F:)                      : A:
DESTINATION FILE NAME (8 char. max.)           : EPROM
```

The program creates the files EPROM.O00 and EPROM.E00. These consist of binary files for the EPROM programmer. Each file is intended for a single EPROM. These EPROM's are programmed and they are then inserted into the EPROM sockets as follows:

up to V32:	EPROM.E00 in IC4 (even)
	EPROM.O00 in IC3 (odd)
from V33:	EPROM.E00 in IC2 (even)
	EPROM.O00 in IC1 (odd)

If you now select the solid-state drive ROM at address C00000h as BIOS drive A:, then you may boot the system from the solid-state disk. Once the system is turned on and booted this storage module is available as drive A:.

Note

The following procedure must be used to create an update for the solid-state disk:

Once the new file has been transferred to an existing diskette, the diskette must be defragmented (start Defrag). The defrag operation stores the files in contiguous form on the diskette. Spaces that exist between blocks of data are overwritten. These records are then transferred byte by byte to the solid state disk. In this way the data occupies as little space as possible on the solid-state disk and all available memory is used effectively when the 1MB boundary of the solid-state disk is reached.

Option 2: by means of the SRAM-disk

Install an SRAM-disk consisting of 2 x 512KB SRAM's (4MBit SRAM's) on the PC-board and set the jumpers accordingly. Set the base address to C00000h.

The system is then booted from fixed disk C:. You require the programs SDROM.SYS, SDRAM.SYS, FORMATSD.EXE and SDGEN.EXE (located in the subdirectory C:\SD) and a text editor. Use the text editor to enter the following lines to the end of the C:\CONFIG.SYS file:

```
DEVICE = SD\SDRAM.SYS base=/C0 size=/1024
```

Reboot the system by simultaneously depressing the keys <CTRL>, <ALT> and <DEL>. The system will return the following messages during the boot procedure:

```
RAM-DISK installed as drive D: .  Vx.y  Date
RAMDISK LOCATD AT C00000h
```

The master for the memory is created in drive D: (SRAM-disk). The SRAM-disk must first be formatted by means of the command

```
SD\FORMATSD D: /D:32 /S
```

Now you must copy the respective data to drive D:. This completes the creation of the EPROM-disk and it may be tested. For this purpose you must change the setup to the solid-state disk ROM located at address C00000 and delete the following line from the CONFIG.SYS file:

```
DEVICE = SD\SDRAM.SYS base=/C0 size=/1024
```

This master may now be used to create the 2 EPROM's. Start the SDGEN program and enter the following parameters:

```
C:\SD\SDGEN
EPROM-SIZE in bit (0,512,1M,2M,4M,8M)   : 4M
SPLIT IN ODD-EVEN                  (J/N): J
SOURCE-DRIVE (A: .... F:)               : D:
DESTINATION FILE-NAME (8 char. Max.)    : EPROM
```

The program creates the files EPROM.O00 and EPROM.E00. These consist of binary files for the EPROM programmer. Each file is intended for a single EPROM. These EPROM's are programmed and they are then inserted into the EPROM sockets as follows:

up to V32:	EPROM.E00 in IC4 (even)
	EPROM.O00 in IC3 (odd)
from V33:	EPROM.E00 in IC2 (even)
	EPROM.O00 in IC1 (odd)

If you now select the solid-state drive ROM at address C00000h as BIOS drive A:, then you may boot the system from the solid-state disk.

Once the system is turned on and booted this storage module is available as drive E:.

4.5.4 Creating a FLASH-PROM solid-state disk with MS-DOS solid-state disk

Purpose to create a 2 MByte FLASH-PROM-solid-state disk on the solid-state disk board at a base address C00000h (consisting of 8 x 256KB FLASH-PROM's (2MBit FLASH-PROM's)).

Here we use a solid-state disk board with 8 x 256KB FLASH-PROM's (2MBit FLASH-PROM's). The base address is set to C00000h and the size to 2MB.

The system is booted from the fixed disk (drive C:). You require the programs SDROM.SYS, SDGEN.EXE, and SDLOAD.EXE (located in the subdirectory C:\SD) and a text editor.

Enter the following lines at the end of the C:\CONFIG.SYS file:

```
DEVICE = C:\DOS\RAMDRIVE.SYS 2042 512 64 /E
DEVICE = C:\SD\SDROM.SYS base=/C0 size=/2048
```

Reboot the system by simultaneously depressing the keys <CTRL>, <ALT> and <DEL>. The system will also display the following messages during the boot procedure:

Microsoft RAMDrive Version x.y - virtual drive D:

Disk size: 2042K

Sector size: 512 Byte

Block size: 1 Sector

Number of directory entries: 64

ROM SILICON DISK installed as drive E:. Vx.y Date

Drive D: is used as the master for the ROM-solid-state disk. For this purpose the label must be removed as this occupies space that is required by the operating system. The respective command is:

```
LABEL D:
```

The following message is displayed:

```
Volume in drive D is MS-RAMDRIVE
Volume label (11 characters, ENTER for none)?
```

Do not enter a volume label but simply press the ENTER key. You will now be prompted whether you:

```
Delete current volume label (Y/N)?
```

Answer "Y" and press the ENTER key.

The LABEL program terminates (MS-DOS prompt). Next you must copy all the required files to drive D: (adhere to the following sequence if the drive must be bootable):

1. Io.sys
2. Msdos.sys
3. command.com
4. remaining files and directories in any sequence

The Io.sys and Msdos.sys files are hidden files and they can therefore not be transferred by the COPY command of DOS.

Files may be transferredby means of the DOSSHELL or Norton Commander. At this point the master for the FLASH-PROM is complete and the program SDGEN is executed with the following parameters:

```
C:\SD\SDGEN

EPROM-SIZER in bit  (0,512,1M,2M,4M,8M): 0
SPLIT IN ODD-EVEN                 (J/N): N
SOURCE DRIVE (A: .... F:)              : D:
DESTINATION FILE NAME (8 char. max.)   : EPROM
```

The program creates a file called EPROM.000. (A note foe experts: SDGEN creates the bootsector and both FAT entries when it generates the file. The disk is thus bootable). The file may then be transferred to FLASH-PROM by means of the SDLOAD program. The program SDLOAD is executed with the following parameters:

```
C:\SD\SDLOAD

    The following FLASH-PROM's or EEPROM's/PEROM's may be programmed:
            1  Am28F010-150,P28F010-150
            2  Am28F020-150,P28F020-150
            3  Am28F040-150,P28F040-150
            4  AT28C010-150
            5  AT28C040-150
            6  AT29C010-150
            7  AT29C040-150
            8  AT29MC010-150
            9  AT29MC040-150
    Please enter the number of the ROM-type you are using: 2

    Please enter the quantity of the above components (2,4,6,8): 8

    Please enter the base address that you have set up on the solid-state disk
    board. This is entered as a hexadecimal value: C00000

    You may either transfer the contents of a file that was generated by means
    of SDGEN or the contents of a logical drive to the solid-state disk.
    Enter the file name or the drive letter: C:\EPROM.000
```

The contents of the file C:\EPROM.000 is transferred into the FLASH-PROM's. If the program terminates without errors, the FLASH-PROM disk is available as drive E: . Both drives D: and E: have identical contents, however, drive E: is write protected. If you change the BIOS settings so that the solid-state disk at address C0h 0000h is the boot disk, then the system may be booted from the solid-state disk.

5 Auxiliary programs

5 Auxiliary programs

5.1 CPLINK program for coupling computers

5.1.1 General

This program may be used to transfer files via a serial interface into and from the CP3/4. It is thus possible to enter data and programs into CP3/4 modules without a diskette drive or an exchangeable memory card. Any recorded data can also be retrieved via the same link..

Available commands appear on the status line (bottom line). Commands that are "grayed out" are not available. Other key combinations and shortcuts are available from the menu system.

You can access the menu line (top line) by means of <F10>. From here you can select the required menu entry using the cursor keys and the enter key. A quicker method is to select the menu item by means of the key combination <Alt+selection-key>. In this case the selection-key is the highlighted capital letter. This displays a sub-menu from where the required command may be selected by means of a further selection key.

The file list windows are available for processing directories. Once the link between the two computers has been established you may switch between the two windows by means of the <Tab> key. The active window is denoted by a double frame. Here you may select files by means of the <cursor keys> (<space key>) and you can access subdirectories (<enter key>).

5.1.2 Function description

Info
Provides information on the version number of the program and the remaining memory space.

Ende (quit)
Terminates the program and disconnects any existing link.

Selektieren (select)
Marks all the files that match the search criteria entered into the mask.

Unselektieren (deselect)
Deselects all the files that were selected by means of the mask.

Betrachten (view)
The file selected by the selection bar is displayed. Data may be displayed in hexadecimal or in ASCII.

Kopieren (copy)
If the program is linked to a second computer the selected files are transferred to that computer.

Umbenennen (rename)
Modifies the name of the selected file.

Löschen (delete)
The selected file is deleted after the operator confirms the action.

Laufwerk wechseln (change drive)
You may select a different drive from a list of drives.

Verzeichnis erstellen (create a new directory)
This command creates a new subdirectory.

Verzeichnis umbenennen (rename subdirectory)
Modifies the name of an existing subdirectory.

Verzeichnis löschen (delete subdirectory)
Deletes an empty subdirectory from the directory tree.

Verbindung aufbauen (connect)

Here you may select the interface (COM1/COM2) and the operating mode of the computer (active/passive). It is also possible to establish a slower connection. This may be necessary when a relatively slow computer is being used or in cases where the link suffers from interference. You must always connect an active computer to a passive computer. The active computer is under the control of the user.

Verbindung unterbrechen (disconnect)

The link between two computers is terminated.

Optionen einstellen (options)

Here you may select the number of display lines and the color.

Dateibetrachter (view file)

This function may be used to view the contents of a selected file.

Change viewer	<F2>
Scroll through text:	<Up/Down>
	<Left/Right>
Page by page	<Page Up/Page Down>
Start/end of line	<Home>, <End>
Start/End of file	<Ctrl+Page Up/Down>
Quit viewer	<ESC>

Kommandozeilen-Parameter (command-line parameters)

You may enter connection parameters as command-line parameters when you start the program.

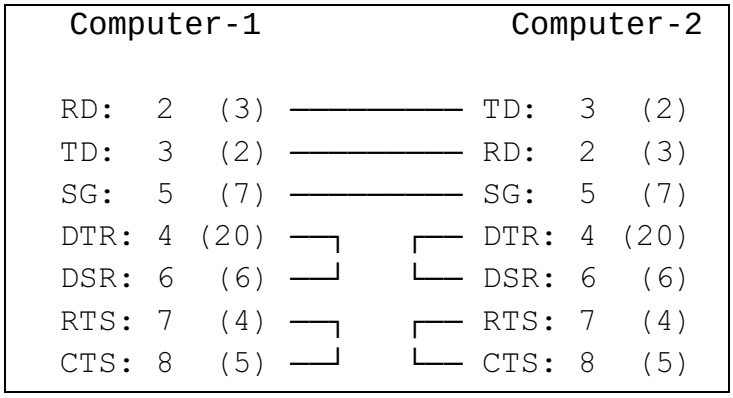
The respective command is as follows:

CPLINK [COMx A/P S/L]

where COMx defines the interface (x: 1 or 2) and A/P defines the operating mode (active or passive). The third parameter defines the data rate, i.e. whether the data is transferred faster or slower. If you select to enter command-line parameters, you must always enter all three parameters!

5.1.3 Interconnecting cables

Serial interfaces on PC's may be provided by a 9-pin or a 25-pin plug. The following cables are required to link computers for data transfers via the serial interface:



The numbers above specify the pin of a 9-(or 25)-pin connector.

5.2 Graphic display program for the PLC process image

The current version of the S5KOP program is available to display the process image from the processor of the PLC.

The size of the MS-DOS program S5KOP.EXE with the version 1.1 dated 8.4.1991 is 75677 bytes. The PLC must contain the VIPA handler module FB1 (from CP386COM or CP486COM) for the CP3/4.

From MS-DOS the program is started by means of the command "S5KOP <Return>". The program then displays an introductory screen which may be removed by depressing a key on the keyboard. You are now presented with the main menu.

The menu item "process image" is activated by the <F1> key. Here you may select a cyclic display by means of function keys:

<F1> display the inputs
<F2> display the outputs
<F3> display the flags 0..127
<F4> display the flags 128..255
<F5> display the timers
<F6> display the counters
<F8> display the page frame

The display cycle may be paused by means of the pause key <F7>. The pause can be terminated by depressing <F7> again.

Once you have selected a process image by means of the keys F1 .. F6 or F8, the selected process image is displayed until you change the selection.

The display is "frozen" when you depress the pause key <F7> ("Pause") until you press another key.

You can quit from the process image menu and return to the main menu by pressing <ESC>. Here you can press <ESC> again to return to the DOS command line.

5.3 System test program

The system test program is available as an optional diskette containing the original Quadtel software (Software pack SW583, MS-DOS diskette, 3.5", 720KB). This software is sold for single user applications only, i.e. a license is required for each and every system where the driver will be used.

This software pack contains a diagnostic program for the CP3/4. The diagnostic software runs under MS-DOS and provides a number of tests for all the important system components. These tests may be executed individually or as a batch.

Annex

Annex

A List of tables

B Index