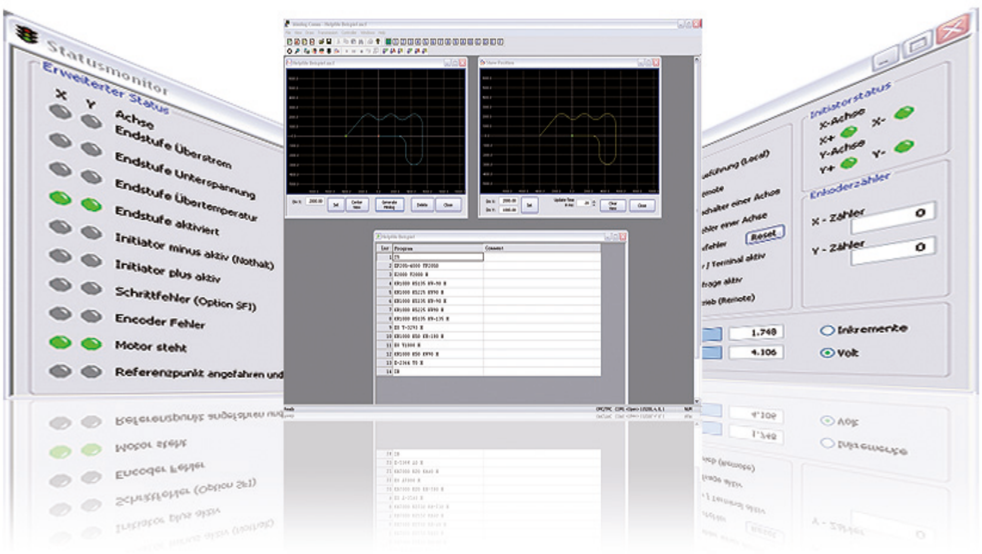
Minilog MCC

Programming Manual for MCC



MANUAL 1240-A008 EN

phytron
Extreme. Precision. Positioning.

Programming Manual MINILOG
for the Controllers
MCC-1, MCC-2 and MCC-2 LIN

TRANSLATION OF THE GERMAN ORIGINAL MANUAL

Version	Modification
7	p.29 Bit 0 corrected Parameters P40 to P42 (MCC-2 LIN) added valid from serial number 15XXXXXXXX
8	p.49 chap. 5 "P27 to P49 are special parameters for MCC-2” removed

© 2018

All rights with:

Phytron GmbH

Industriestraße 12

82194 Gröbenzell, Germany

Tel.: +49(0)8142/503-0

Fax: +49(0)8142/503-190

Every possible care has been taken to ensure the accuracy of this technical manual. All information contained in this manual is correct to the best of our knowledge and belief but cannot be guaranteed. Furthermore we reserve the right to make improvements and enhancements to the manual and / or the devices described herein without prior notification.

We appreciate suggestions and criticisms for further improvement. Please send your comments to the following E-mail address: doku@phytron.de

Contents

1 Structure of the Minilog Instructions

1.1 Instruction Code

Xrvalue	**X**	The bold characters represent the instruction code and must be used unchanged. In this example: **X** represents the motion instruction code for relative positioning of the X-axis. **This manual shows only the instruction codes for the single axis controller (MCC-1), where the axis is generally named "X".** **For multiple axis controllers (also Master/slave up to 8 axes) the corresponding characters X, Y,... or 1, 2, … must be used.**
	r	Small letter require the input of the characters or values which are described in the column *Meaning*. In this example: r = running direction + or – .
	value	In this example a running distance of 1000 is fed in. The corresponding unit (e.g. steps) of the particular input is defined by **parameters**. For the specific parameters refer to chap. 5.
Example: X+1000		Relative motion instruction for the X-axis: Go 1000 steps to the + direction.

Important:

- **All characters and signs, belonging to one single instruction, must be written without a blank.**
- **The instructions themselves must be separated by a blank.**
- **Leading zeros in an instruction are ignored (Example: the instruction A001S is realized as A1S)**
- **Instructions, which cannot be used in the program <u>and</u> direct mode, are marked with:**
    1. **Instruction only used in the program (ONLY PROG)**
    2. **Instruction only used in the PC direct mode (ONLY PC)**

Exception: In the instruction group "System Status (ONLY PC)", chapter 2.12, the first program name must be separated by a blank from the second program name or the following alphanumeric part of the instruction code.

1.2 Design of MiniLog Programs

- MiniLog programs consist of up to 2000 program lines. The program lines are numbered consecutively by MiniLog-Comm.
- The single instructions in each line must be separated by blank characters.
- Do not insert extra blank characters within an instruction.
- The instructions will be executed serially.
- By means of the line numbers jump instructions or subroutines can be defined.
- Parameter and register values should be defined at the beginning of a program.
- Line, parameter and register numbers may be entered with or without preceding zeros.

  Example: R0001 or R1
- A line break is used in the program by a carriage return **(CR) : 0x0D**

  Example: A1S T500 A1R **0x0D**
  A2S T500 A2R **0x0D**

All programming instructions are assembled in the MiniLog programming manual.

1.3 Addressing Mode

For instructions, where at least one register is used as an operator, two addressing modes are available: The **Direct Addressing Mode** and the **Indirect Addressing Mode**. In this programming manual the meaning of the basic instruction is always explained for the **Direct Addressing Mode**. The variations of the **Indirect Addressing Mode** are listed for the sake of completeness. The first named register within an instruction code is always the destination register for the result.

Example for Direct Addressing Mode:

Instruction	***Meaning***
Rnn**BE**nn**–**mm	The status of the inputs nn to mm is written as a binary value into the register nn. **Example: R1BE1–8** Status of the inputs 1 to 8 is e.g. **1010 0101**. This binary value is written into the register 1. After the Instruction was carried out, the register content is 165 decimal.

Example for Indirect Addressing Mode:

Instruction	***Meaning***
R[Rnn**]BE**nn**–**mm	**Indirect Addressing Mode:** The status of the inputs nn to mm is written as a binary value into the register which is addressed by the register **[Rnn]**. Example: **R1S10 [R1]BE1–8** The addressing register **[R1]** is set to 10. The status of the inputs 1 to 8 is e.g. 1010 0101. This binary value is written into the register 10, which was addressed by the register 1. After the instruction was carried out, the content of the register 10 is 165 decimal.

Adressing with Label

In case of jump calls (page 15) and subroutine calls (page 32) the start or destination line can be set in the instruction code with a label (*la*), which is assigned to this program line. A label is defined between two * and can have up to 6 alphanumeric signs. Max. 100 labels can be used in one program.

Example: *[label name]*

Program name:

Program names [name] in the instruction code can have up to 8 alphanumeric signs.

1.4 Conditional Instructions

The execution of some instructions (e.g. jumps or subroutine calls) can be combined with a condition. Before a conditional jump etc. can be used, the condition byte has to be set, for example by an input request (see chap. 2.5) or a register comparison (see chap. 2.14).

Possible states of the condition byte:

E = Condition fulfilled **N** = Condition not fulfilled

The state of the condition byte remains stored until it is changed again.

All instructions which set no condition delete the condtion request.

1.5 Data and Telegram Format

Data format: No Parity

1 Stopbit

8 Bit ASCII-Code

57 600 Baud

The **send telegram** from PC via RS232 is defined as:

Without check-sum: <STX>| Address | Instruction | <ETX>

With check-sum: <STX> | Address | Instruction | Separator | Checksum | <ETX>

The **response telegram** (always for address 0-9, A-F) is defined as:

<STX> **|** ACK **|** Answer **|** <ETX> or

<STX> **|** ACK **|** <ETX> or

<STX> **|** NAK **|** <ETX>

	Meaning
<STX>	<STX> (Start of Text, 02_H): It is exclusively used as the start code for a new telegram
Address	Address of the controller, the range of the address byte is 0 to 9 and A to F (30_H…39_H and 41_H…46_H). Additional the Broadcast[1] address @ (40_H) is used.
Instruction	MINILOG instruction code
Separator	: Colon ($3A_H$) as separator, to distinguish between usable data and checksum.
Checksum	Upper byte of the checksum value (see below for the algorithm to calculate the checksum)
	Lower byte of the checksum value (calculation see below)
<ETX>	(End of Text, 03_H) this code indicates the end of the telegram.
ACK	(Acknowledge 06_H), the instruction has been confirmed.
NAK	(Negative Acknowledge 15_H), the instruction has been negatively confirmed.
Answer	Answer as number or string, f.ex. E or N

[1] Broadcast: All axes receive and evaluate the telegram. To avoid bus-conflicts caused by the response of all axes nearly within the same time, the response of the controllers is suppressed by addressing with “@”.

The checksum CS is defined by summing up all bytes, beginning with the address byte and including the separator (:) in an exclusive-OR-operation:

CS = address $\oplus$ data byte 1 $\oplus$ data byte 2 . . . $\oplus$ data byte n $\oplus$ separator

The checksum is calculated as one 8-bit binary value (00_H to FF_H). This byte is taken apart in its upper and lower byte (nibbles). After the HEX values of the two nibbles have been transferred to the corresponding two ASCII characters (0 to 9 instead of 0_H to 9_H and A to F instead of A_H to F_H, that means to each nibble 30_H or rather 37_H is mathematically added), the checksum is written in the telegram.

The MCC also calculates (Exclusive OR) the checksum of the received data. The telegram will be rejected if a difference to the received checksum is detected, and the error is confirmed by NAK.

If there is no need to validate the contents of the telegram, the checksum monitoring can be set off. Instead of the checksum bytes, **two X** characters will be accepted, e. g.:

<STX>| 1 | X | + | 1 | 0 | 0 | : | X | X <ETX>

2 MINILOG Instructions

2.1 Outputs

Instruction	***Meaning***
	Set Outputs
Annnz	Set one or several outputs at the same time. nnn, mmm, xxx → number (ID) of the output
Annnzmmmzxxxz	z = S → set z = R → reset **Example: A1S2R3S** Output **1** and **3** ON, output **2** OUT
	Read Output Status
AGn**R**	Read the state of the output groupes n. (ONLY PC) **Example: AG2R** The 2nd output groupe is read **Response : <STX><ACK>nnnnnnnn<ETX>** (ONLY PC) n = 0 Output OFF n = 1 Output ON
	Set the output group outputs
AGn**S**zzzzzzzz	Set the output group n=1 or 2, z= 0 or 1. z must always have 8 places **Example: AG1S10101001** The 1. output group is set with the information '10101001'
	Read Output Status
ARnnn;mmm;xxx	The state of the outputs nnn, mmm, xxx is read. (ONLY PC) **Response : <STX><ACK>nnn<ETX>** n = 0 Output OFF n = 1 Output ON **Important:** Set a **;** between the output numbers.

2.2 A/D Converter

Instruction	***Meaning***
ADn**R**	A/D converter setting is read. n→ A/D converter address: n=1 or n=2 **Response : <STX><ACK>[0 to 1023]<ETX>** (ONLY PC) 0 to 1023 = 0 to 5 V

2.3 Reset

Instruction	***Meaning***
CR	The controller is reset by the interface.
CT	The display of the terminal is deleted via interface. **Response : <STX><ACK><ETX>** (ONLY PC)

2.4 Write Instructions via Serial Interface

Instruction	***Meaning***
	Informations can be carried out via the 3 serial interfaces (X31, X32, X9). The writing is carried out without formatting. s = 1 → interface nameRS232 /Com X5)
Ds **<t**ext**>**	The bracketed expression is carried out.
Ds**R**nn	The content of the register nn is carried out.
Ds**R[R**nn**]**	The content of the register which is addressed by register nn is carried out.
Dsx**P**mm	Parameter mm of the axis x is carried out. mm = 1 to 45 → number (ID) of the parameter x = 1 to 8 or X,Y,Z,W,5,6,7,8 → axis ID **Example: D24P10** The parameter 10 of the axis 4 is carried out via the interface X32 .

2.5 Input requests

Instruction ***Meaning***

Logical AND

E^nnzmmzxxz The inputs nn, mm, xx are tested as AND condition.
Only if the AND condition is fulfilled the condition byte is set.
Otherwise the condition byte is reset.

nn. mm. xx → input number
z = S → input ON
z = R → input OFF

Example: E^1S2R3S
The input states 1, 2 and 3 are read out. If input 1 is set, input 2 reset and input 3 set, the AND condition is fulfilled and the condition byte is set. Now a conditional jump or a conditional call of a subprogram can be carried out.

Response: <STX><ACK> E <ETX> or
<STX><ACK> N <ETX> (ONLY PC)

Logical OR

Evnnzmmzxx The inputs nn, mm, xx are tested as OR condition.
Only if the OR condition is fulfilled the condition byte is set.
Otherwise the condition byte is reset.

nn. mm. xx → input number
z = S → input ON
z = R → input OFF

Example: Ev1S2R3S

The input states **1, 2 and 3** are read out. If input 1 is set or input 2 reset or input 3 set, the AND condition is fulfilled and the condition byte is set. Now a conditional jump or a conditional call of a subprogram can be carried out.

Response: <STX><ACK> E <ETX> or
<STX><ACK> N <ETX> (ONLY PC)

Wait for Condition Fulfilled

Ennz Wait for the preset input condition.
The program stops until the preset input condition is fulfilled. The condition byte is not affected. (ONLY PROG)

Instruction	***Meaning***
Ennzmmz	When reading the status of several inputs, one input after the other is read out (no AND linking). The condition byte is not affected. (ONLY PROG) **Example:** **E1S2R3S** The status of the inputs 1, 2 and 3 are read. After the input 1 is set, the input 2 is read. After the input 2 is reset, the input 3 is read. After the input 3 set, the reading Instruction is done and the program continues. After the instruction end the inputs1 and 2 can have another state.
	Definition of the inputs and outputs only for MCC-1
EASnnnnnnnn	The MCC-1 controller has eight digital inputs and outputs, electrically insulated and bidirectional. Which I/Os are input or output can be defined by the user via MiniLog programming. n = allocation → input or output n = 1 → input n = 0 → output **Example:** **EAS00000011** 1 to 6 are outputs, 7 and 8 are used as inputs.
	Read input group
EGn**R**	The input group n is read. (ONLY PC) n=1 to 8 **Response : <STX><ACK>nnnnnnnn<ETX>** n = 0 input is reset n = 1 input is set
ERnn;mm;xx	The Status of the inputs nn, mm, xx is read (ONLY PC). **Response: <STX><ACK>nnn<ETX>** n = 0 input is reset n = 1 input is set **Important:** Set a **;** between the input numbers.

2.6 Program Manipulation at Emergency Stop (ONLY PROG)

Instruction	***Meaning***
FNznr	The program line is defined at which the program has to be continued in the case of an emergency stop.
FN*la**	The program line, at which the program has to be continued in the case of an emergency stop, is defined by a label.
FP[name**]**	Indicates the program for an emergency stop. In the case of an emergency stop, a jump to the named program is initiated.

2.7 Program Interruption

Instruction	***Meaning***
H	The program waits here until all axes have stopped. (ONLY PROG)

2.8 System Adaption during Program Execution

Instruction	***Meaning***
	Number of Axes
IAR	The number of existing axes is requested (ONLY PC). **Response: <STX><ACK>n<ETX>**
	Automatic Start
IBSname	The name of the start program is written into the Auto Start register. If the REMOTE/LOCAL switch is in the LOCAL position the program execution starts here. **Response: <STX><ACK><ETX>** or **<STX><NAK><ETX>** (ONLY PC)
IBR	The name of the Auto Start program is read (ONLY PC). **Response: <STX><ACK>name<ETX>**
	Read/Set Baudrate (ONLY PC)
ICn**S**baud	Set the baudrate for the MCC interfaces. n = 1 $\rightarrow$ COM 1 of MCC baud = Baudrate (9600, 19200, 38400, 57600 or 115200 Baud)
ICn**R**	Baudrate setting of the MCC interfaces is read out. n = 1 $\rightarrow$ COM 1 of MCC

Instruction	***Meaning***
	Remote/Local Reversing (ONLY PC)
IFR	The controller is reversed to the Remote function. If a program is running, it is canceled. If the switch is positioned to Local, the position Remote is simulated. **Response: <STX><ACK><ETX>**
IFL	The controller is reversed to the function Local, if the Remote/Local switch is on the position Local. If the switch is on Remote position, it is not reversed. **Response: <STX><ACK><ETX>**
	Directory of RAM
IPn	Read out the n program *name* of the program list from the RAM. If no program name exists, the response NAK is shown (ONLY PC). **Response: <STX><ACK>name<ETX>** **Response: <STX><NAK><ETX> no name available**
	Information of the transmission protocol
ITR	Read out the state of the RS interface transmission protocol
ITSn	Define the RS interface transmission protocol n=0 Instruction transmission **without** checksum n=1 Instruction transmission **with** checksum
	Information operator panel
ITTSn	Define type of operator panel n=0 drive without operator panel n=1 drive with operator panel BT5 n=2 drive with operator panel TP11
	Version Request
IVR	The software version of the controller is read (ONLY PC). **Response: <STX><ACK>Software Version<ETX>**

2.9 Jump Instructions (ONLY PROG)

Instruction	***Meaning***
	Relative Jump
N+nn **N–**nn	Relative jump forward (+) or backward (–). Distance: nn program lines.
N+Rnn **N–R**nn **N+R[R**nn**]** **N–R[R**nn**]**	Relative jump forward (+) or backward (–). Distance: number of program lines, defined by the content of register nn.
	Absolute Jump
Nnn	Absolute jump to program line number nn.
N*la*****	Absolute jump. Destination program line number: marked by the label *la*.
NRnn **NR[R**nn**]**	Absolute jump. Destination program line number: defined by the content of register nn.
NP[name**]**	Absolute jump to program *name*. Program starts at line number 1.
NP[name**]N**nn	Absolute jump to program *name*. Program starts at line number nn.
NP[name**]NR**nn **NP[**name**]NR[R**nn**]**	Absolute jump to program *name*. The program start line number is defined by the content of register nn.
NP[name**]N***la*****	Absolute jump to program *name*. The program start line is marked by the label ***la***.
	Conditional Jump Relative / E = Condition Fulfilled
NE+nn **NE–**nn	Relative jump forward (+) or backward (–). Distance: nn program lines.
NE+Rnn **NE–R**nn **NE+R[R**nn**]** **NE–R[R**nn**]**	Relative jump forward (+) or backward (–). Distance: number of program lines, defined by the content of register nn.
	Conditional Jump Absolute / E = Condition Fulfilled
NEnn	Absolute jump to program line number nn.
NE*la*****	Absolute jump. Destination program line: marked by the label ***la***.
NERnn **NER[R**nn**]**	Absolute jump. Destination program line number: defined by the content of register nn.
NEP[name**]**	Absolute jump to program *name*. Program starts at line number 1.
NEP[name**]N**nn	Absolute jump to program *name*. Program starts at line number nn.

Instruction	***Meaning***
NEP[name**]NR**nn **NEP[**name**]NR[R**nn**]**	Absolute jump to program *name*. The program start line number is defined by the content of register nn.
NEP[name**]N***la*****	Absolute jump to program *name*. The program start line is marked by the label ***la***.
	Conditional Jump Relative / N = Condition Not Fulfilled
NN+nn **NN–**nn	Relative jump forward (+) or backward (–). Distance: nn program lines.
NN+Rnn **NN–R**nn **NN+R[R**nn**]** **NN–R[R**nn**]**	Relative jump forward (+) or backward (–). Distance: number of program lines, defined by the content of register nn.
	Conditional Jump Absolute / N = Condition Not Fulfilled
NNnn	Absolute jump to program line number nn.
NN*la*****	Absolute jump. Destination program line number: marked by the label ***la***.
NNRnn **NNR[R**nn**]**	Absolute jump. Destination program line number: defined by the content of register nn.
NNP[name**]**	Absolute jump to program *name*. Program starts at line number 1.
NNP[name**]N**nn	Absolute jump to program *name*. Program starts at line number nn.
NNP[name**]NR**nn **NNP[**name**]NR[R**nn**]**	Absolute jump to program *name*. The program start line number is defined by the content of register nn.
NNP[name**]N***la*****	Absolute jump to program *name*. The program start line is marked by the label ***la***.

2.10 Repeating of Program Lines

Instruction	***Meaning***
NWnn	The program line is nn times repeated.
NWRnn **NWR[R**nn**]**	The program line is repeated as often as defined by the content of register nn. **Response: <STX><ACK><ETX>** (ONLY PC)

2.11 Passwort

Instruction	***Meaning***
PA␣	The controller is activated, if there is no password. Then it is not possible to lock the controller.
PAname	The password protected controller is activated.
PSname	This comand allocates the controller a password . It has maximum 8 alphanumeric signs.
	Activation Status for Programs, Parameters and Registers
PWSp	Set activation status Programs, parameters and registers can be activated or locked by a password protected controller. p → Activation status for programs, parameters and registers p = 0 all activated p = 1 Program R/W locked p = 2 Parameter R/W locked p = 4 Register R/W locked p is between 0 and 7 Example: PWS**5** Program and register locked (1+4=5)
PWR	Read activation status The answer are two digits. **<ACK> sp** s □ Activation status of the controller s = 0 Controller locked s = 1 Controller activated p → Activation status for programs, parameters and registers see above **Example: PWR** **<ACK> 15** s = 1 → Controller activated p = 5 → Program and register locked (1+4=5)

2.12 Ending or Interruption of a Program Call (ONLY PROG)

Instruction	***Meaning***
PE	The actual program is ended and the system waits for another changing of the **REMOTE/LOCAL** switch. If the program was started via computer, the system goes back to the **COMPUTER MODE**.

2.13 Program and Data Management (ONLY PC)

Instruction	***Meaning***
	Delete Programs and Data
QDP*.*	All programs in the RAM are deleted
QDR	All registers in the RAM are set to zero.
	Read Program Line
QPname **N**nn**R**	The program line nn of the program *name* is read.
	Program Start by line
QPname **N**nn**A**	The program *name* is started from line nn.
	Program Stop
QPE	If the QPE Instruction is sent by the computer, it causes a jump back to the initial program level from which the program has been started.
	Program Transmission with Read Out
QPname **S**byte	The program *name* is to be transmitted block wise. During the transmission, the whole control sequence must be observed. The name must have 8 characters. The line number is not transmitted.

name → maximal 8 characters,
byte → number of bytes to be transmitted

1. Computer:

<STX>controller address QPname Sbyte**<ETX>**
The program transmission sequence is started.

2. Controller response:

STX><ACK>O<ETX>
if the program does not exist in the controller, and the controller's RAM capacity is sufficient.

<STX><ACK>E<ETX>
If the program exists in the controller and must be overwritten.

How to overwrite:
1) Backup of all programs of the MCC on the PC.
2) Delete all programs in the Flash RAM of the MCC.
3) Rewrite the programs (including revised prog.) to the MCC.

Instruction	***Meaning***
	3. Program transmission: • Start: **<STX>controller address block 1<ETX>** Block 1 is 256 byte long and starts with program name <ETB> .The program name must have 8 characters! • Further blocks (block 1+x) always must have 256 bytes and are embedded in **<STX>controller address block 1+x<ETX** • Last block: 0x04 (EOT) must be the last character. If the last block is shorter than 256 bytes, the rest of the block must be filled with EOT. Example: **<STX>controller address block end <EOT><EOT>… <EOT><EOT><ETX>** Controller response after each block: **<STX><ACK><ETX>**
	Read Program with Request
QPname **R**	The program *name* is to be read from the controller unit. The name must have 8 characters. The program is to be read by the line. **Request and send** 1. Computer: **<STX>controller address QP**name **R<ETX>** The program *name* is to be read. 2. Controller response: **<STX><ACK>O**lnr**<ETX>** If the program is available, the controller unit reports the character O and the number of program lines lnr . 3. Computer: **<STX>controller address J<ETX>** The computer receives the first line of the controller. 4. Contoller response: **<STX>data program line x<ETX>** The data are read line by line by indicating the line number. Number 3. and 4. are repeated as long as all lines are received. The transmission is finished by appending 0x04(EOT) to the last line. Example for last line: **<STX>Data last program line …<EOT><EXT>**

2.14 Registers

- The MCC-2 controllers contain 1000 memory locations used to store variables, called **Registers** within MiniLog programs.
- The registers are numbered R1 to R1000.
- In each register numbers with up to ten digits can be entered. Decimal values are also programmable. Before and after the decimal point up to seven digits may be entered. The total number of digits must not exceed 8.
- If possible, the registers should be programmed in the first program lines.

Write value into a register:	**R**nnnn**S**zz
Read value of a register:	**R**nnnn**R**

Explanations:	**R**	instruction code: register
	nnnn	register number
	S	Write (Schreiben)
	zz	number (maximum 10 digits)

- Within the program registers can be used for indirect input of positions. Combined with arithmetic calculations registers can be used as counters during program run.
- For all logic combinations or arithmetic calculations with registers please notice: The computed value will always be written into the first register named in the instruction.

<u>Example</u>:	Add the values of two registers
R18+R2	Value of register 2 is added to value of register 18. The result will be stored in register 18.

Compare register values

As the result of a comparison, a condition byte will be set by the program:

E = condition fulfilled,

N = condition not fulfilled.

The status of the condition byte can be used for a conditional jump, subroutine instructions or other operations.

<u>Example</u>:	Comparison of a register value with a number and conditional jump
R999=1 NE11 N77	If register 999 contains the value 1, jump to line 11, if not, jump to line 77.

2.15 Register Instructions

Instruction	***Meaning***
	Register Value Integer
Rnn.z	The digits after the decimal point of the register nn are deleted without truncation of the value. z = 0 – 6 digits after the decimal point
	Set Outputs with Register Value
Rnn**BA**nn**–**mm **R[R**nn**]BA**nn**–**mm	The content of the register nn is set as a binary value to the controller outputs nn to mm.
	Load Register with Input Status
Rnn**BE**nn**–**mm **R[R**nn**]BE**nn**–**mm	The status of the inputs nn to mm is written as a binary value into the register nn. **Example:** **R1BE1–8** → Input status: **1010 0101** Result: **165**
	Load Register with Hexadecimal Value
Rnn**BS**value **R[R**nn**]BS**value	The register nn is set to the value. The data are fed in hexadecimal **Example:** **R1BS1FA** The register 1 is set to the hexadecimal value 1FA. After the instruction was carried out, the content of the register 1 is 506 decimal.
	Shift Register Bit by Bit
Rnn**BL**m **R[R**nn**]BL**m	The content of the register nn is shifted the number of m digits to the left (MSB ←). The right side of the register is filled in with zero. m = 1 to 27 → maximal value of the register content. **Example:** **R1S168 R1BL2** The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. After the register content was shifted the number of 2 digits to the left, the binary value is **1010100000** which corresponds to the decimal value 672. **Response: <STX><ACK><ETX>** (ONLY PC)

Instruction ***Meaning***

Rnn**BR**m
R[Rnn**]BR**m

The content of the register nn is shifted the number of m digits to the right (→ LSB). The left side of the register is filled in with zero.

m = 1 to 27 → maximal value of the register content.

Example: R1S168 R1BR2

The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. After the register content was shifted (R1BL2) the number of 2 digits to the right, the binary value is **101010** which corresponds to the decimal value 42.

Register Bit Check

Rnn**BT**m
R[Rnn**]BT**m

The content of the register nn is regarded as a binary value. The digit in the position m of the binary value is checked. If the corresponding bit has been set, the condition byte is set. Otherwise the condition byte is reset.

m = 0 to 27 → maximal value of the register content.

Example: R1S168 R1BT4

The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. The Instruction R1BT4 checks the 4^{th} digit from the right side (m ← LSB) of the binary value. The condition byte is set, because the 4^{th} digit has the value 1.

Response: <STX><ACK> E <ETX> or

<STX><ACK> N <ETX> (ONLY PC)

Logical Register Operations

Logic AND

Rnn**B^**value
R[Rnn**]B^**value

A logical **AND** operation is carried out with the content of the register nn and the hexadecimal value. The condition byte is set if the result is zero. Otherwise it is reset.

Example: R1BS2A8 R1B^1A0

The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1B^1A0** has been carried out, the content of the register 1 is 160 decimal.

	Decimal	Hex	Binary
	680	**2A8**	**1010101000**
	416	**1A0**	**0110100000**
Result:	**160**	**0A0**	**0010100000**

Rnn**B^R**mm
R[Rnn**]B^R**mm
Rnn**B^R[R**mm**]**
R[Rnn**]B^R[R**mm**]**

A logical **AND** operation is carried out with the content of the register nn and the content of register mm. The condition byte is set if the result is zero. Otherwise it is reset.

Instruction	***Meaning***
	Logic OR
Rnn**Bv**value **R[R**nn**]Bv**value	A logical **OR** operation is carried out with the content of the register nn and the hexadecimal value. The condition byte is set if the result is zero. Otherwise it is reset.

Example: **R1BS2A8 R1Bv1A0**

The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1Bv1A0** has been carried out, the content of the register 1 is 936 decimal.

	Decimal	**Hex**	**Binary**
	680	**2A8**	**1010101000**
	416	**1A0**	**0110100000**
Result:	**936**	**3A8**	**1110101000**

Rnn**BvR**mm **R[R**nn**]BvR**mm **R**nn**BvR[R**mm**]** **R[R**nn**]BvR[R**mm**]**	A logical **OR** operation is carried out with the content of the register nn and the content of register mm. The condition byte is set if the result is zero. Otherwise it is reset.

Response: <STX><ACK><ETX> (ONLY PC)

	Logical Exclusive OR
Rnn**BX**value **R[R**nn**]Bx**value	A logical **XOR** operation is carried out with the content of the register nn and the hexadecimal value. The condition byte is set if the result is zero. Otherwise it is reset.

Example: **R1BS2A8 R1BX1A0**

The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1BX1A0** has been carried out, the content of the register 1 is 776 decimal.

	Decimal	**Hex**	**Binary**
	680	**2A8**	**1010101000**
	416	**1A0**	**0110100000**
Result:	**776**	**308**	**1100001000**

Rnn**BXR**mm **R[R**nn**]BXR**mm **R**nn**BXR[R**mm**]** **R[R**nn**]BXR[R**mm**]**	A logical **XOR** operation is carried out with the content of the register nn and the content of register mm. The condition byte is set if the result is zero. Otherwise it is reset
	Compare Register Content with Number Values
Rnn=value **R[R**nn**]**=value	The content of register nn is compared with a number (value). The condition byte is set if equality has been detected. Otherwise it is reset.
Rnn#value **R[R**nn**]**#value	The content of register nn is compared with a number (value). The condition byte is set if inequality has been detected. Otherwise it is reset.

Instruction	***Meaning***
Rnn>value R[Rnn]>value	The content of register nn is compared with a number (value). The condition byte is set if the register value is higher. Otherwise it is reset.
Rnn<value R[Rnn]<value	The content of register nn is compared with a number (value). The condition byte is set if the register value is lower. Otherwise it is reset.
	Compare Register Content
Rnn=Rmm R[Rnn]=Rmm Rnn=R[Rmm] R[Rnn]=R[Rmm]	The content of register nn is compared with the content of register mm. The condition byte is set if equality has been detected. Otherwise it is reset.
Rnn#Rmm R[Rnn]#Rmm Rnn#R[Rmm] R[Rnn]#R[Rmm]	The content of register nn is compared with the content of register mm. The condition byte is set if inequality has been detected. Otherwise it is reset.
Rnn>Rmm R[Rnn]>Rmm Rnn>R[Rmm] R[Rnn]>R[Rmm]	The content of register nn is compared with the content of register mm. The condition byte is set if the value of register nn is higher. Otherwise it is reset.
Rnn<Rmm R[Rnn]<Rmm Rnn<R[Rmm] R[Rnn]<R[Rmm]	The content of register nn is compared with the content of register mm. The condition byte is set if the value of register nn is lower. Otherwise it is reset.
	Response for all relations: **<STX><ACK> E <ETX>** or **<STX><ACK> N <ETX>** (ONLY PC)
	Arithmetical Register Operations
	Addition
Rnn+value R[Rnn]+value	The value is added to the content of register nn.
Rnn+Rmm Rnn+R[Rmm] R[Rnn]+Rmm R[Rnn]+R[Rmm]	The content of register mm is added to the content of register nn.
	Subtraction
Rnn–value R[Rnn]–value	The value is subtracted from the content of the register nn.
Rnn–Rmm Rnn–R[Rmm] R[Rnn]–Rmm R[Rnn]–R[Rmm]	The content of register mm is subtracted from the content of the register nn.
	Multiplication
Rnn*value R[Rnn]*value	The content of register nn is multiplied by the value.

Instruction	***Meaning***
Rnn*Rmm R[Rnn]*Rmm Rnn*R[Rmm] R[Rnn]*R[Rmm]	The content of the register nn is multiplied by the content of register mm.
	Division
Rnn:value R[Rnn]:value Rnn/value R[Rnn]/value	The content of register nn is divided by the value.
Rnn:Rmm Rnn:R[Rmm] R[Rnn]:Rmm R[Rnn]:R[Rmm]	The content of register nn is divided by the content of register mm.
	The content of register nn is divided by the value.
Rnn/Rmm Rnn/R[Rmm] R[Rnn]/Rmm R[Rnn]/R[Rmm]	The content of register nn is divided by the content of register mm.
	Trigonometric Functions
RnnSIN RnnCOS RnnTAN	Sinus, Cosinus or Tangant is evaluated from the value of the register nn and the result is written back to the register nn.
	Square Root
RnnQW	The square root is evaluated from the value of the register nn and written back to the register nn.
	Random Number
RnnRAND	The register nn is set with the random number in the range 0 to 4294967296 (2^{32}).
	Read Register
RnnR R[Rnn]R	The content of register nn is read (ONLY PROG). **Response: <STX><ACK>value<ETX>** **Response for all arithmetical operations:** **<STX><ACK><ETX>** (ONLY PC)
	Write Register with Decimal Values:
RnnSvalue R[Rnn]Svalue	Register nn is set to the value.
	with Register Values:
RnnSRmm R[Rnn]SRmm	Register nn is set to the value of register mm.

Instruction	*Meaning*
Rnn**SR[R**mm**]** **R[R**nn**]SR[**mm**]**	
	with Parameter Values
Rnn**SXP**mm **R[R**nn**]SXP**mm	Register nn is set to the parameter mm of axis x.
	with line number emergency stop
Rnn**SN** **R[R**nn**]SN**	The register nn is set with the line number, in which an emergency stop started. **Example :** **Lnr 001 FN10** **Lnr 002 X+1000** **Lnr 003 X–1000 H N2** **Lnr 010 R1SN** In this example an emergency stop program is defined in line 10. The x-axis drives 1000 steps in + and – direction. In case of an emergency stop during an axis drive, the program continues in line 10 and the axis is stopped. With the instruction **R1SN** the register 1 is set with the line number, in which the emergency stop started. Then, it is possible to interpret at which positioning the emergency stop started.
	with the Timer Ticker Value
Rnn**STT**	The register nn is written with the timer ticker value.
	with the Program Line Number
Rnn**SZ**	The program line number at which this instruction is called is written into the register nn.
R[Rnn**]SZ**	The program line number at which this instruction is called is written into the register which is adressed by the register nn. **Example :** **Lnr 041 R1S10** **Lnr 042 R[R1]SZ** In this example the register 1 is written with the value 10. The register 10 is written with the instruction **R[R1]SZ** by the actual line number. The register content 10 is now 42. This instruction can be used for automatic start functions.
	Write Register via Inputs
Rnn**SE**mm**–**xx.k **R[R**nn**]SE**mm**–**xx.k	A BCD value is written via the inputs mm to xx into the register nn. k = number of digits after the decimal point. **Example: R1SE1–8.1** The inputs 1 to 8 have e.g. the status: **1001 0011**. The result is 9.3.

Instruction	***Meaning***
	Change Register with Terminal
Rnn**ST**	Register nn is displayed in line 4 from position 10. New data are input at the cursor position. The register nn is rewritten by pressing the ENTER key of the terminal. **Example : R41ST** In this example the register 41 is displayed in the 4th line of the terminal display from the 10th position and is ready for editing. **Response: <STX><ACK><ETX> if terminal available** **<STX><NAK><ETX> if no terminal available** (ONLY PC)
Rnn**ST.**z	Register nn is displayed in line 4 with z digits after the decimal point (z=0 to 6). New data are input at the cursor position. The register nn is rewritten by pressing the ENTER key of the terminal. **Example : R2ST.6** The register 2 is displayed with 6 digits after the decimal point on the terminal and is rewritten.
Rnn**ST**y	Register nn in line y is displayed (y=1 to 4) and is rewritten after new input with ENTER key. **Example : R2ST3** The register 2 is displayed in the 3rd line on the terminal from position 1 and is rewritten.
Rnn**ST**y.z	Register nn in line y (y=1 to 4) is displayed with z digits after the decimal point (z=0 to 6) and is rewritten after new input with ENTER key. **Example : R2ST3.4** The register 2 is displayed in the 3rd line with 4 digits after the decimal point from position 1 on the terminal and is rewritten.
Rnn**ST**y;m	Register nn in line y (y=1 to 4) is displayed from position m (m=1 to 20) and is rewritten after new input with ENTER key. **Example : R1ST3;6** The register 1 is displayed in the 3rd line from position 6 and is newly written.
Rnn**ST**y;m.z	Register nn in line y (y=1 to 4) is displayed with z digits after the decimal point (z=0 to 6) from position m (m=1 to 20) and is rewritten after new input with ENTER key. **Example : R2ST2;2.6** The register 2 is displayed in the 2nd line from position 2 with 6 digits after the decimal point from postion 1 on the terminal and is rewritten.

Instruction	***Meaning***
	with A/D converter values
Rnn**SAD**y **R[R**nn**]SAD**y	Register nn is written by the A/D converter value. y=1 to 2: A/D converter channel
	Display Register with Terminal
Rnn**W**y	The value of register nn is displayed in line y from position 1 (y=1 to 4). **Example : R2W2** Register 2 is displayed in line 2 from position 1.
Rnn**W**y.z	The value of register nn is displayed in line y from position 1 (y=1 to 4) with z digits after the decimal point (z=0 to 6). **Example : R1W4.6** The register 1 is displayed in the 4th line from the 1st position with 6 digits after the decimal point.
Rnn**W**y;m	The value of register nn is displayed in line y from position m (y=1 to 4, m= 1 to 20). **Example : R2W3;5** The register 2 is displayed in the 3rd line from the 5th position.
Rnn**W**y;m.z	The value of register nn is displayed in line y from position m (y=1 to 4, m= 1 to 20, z=0 to 6) with z digits after the decimal point. **Example : R7W2;5;3** The register 7 is displayed in the 2nd line from the 5th position with 3 digits after the decimal point. **Answer: <STX><ACK><ETX>** (ONLY PC)

2.16 System Status (ONLY PC)

Instruction	***Meaning***
	System Status General
S	Axes check and request for the number of axes.
	Response:<STX><ACK>n IO <ETX>
	n = number of axes
	System Status Binary
SB	Read system status in binary format ($\mathbf{d_{B\,=}}$ 0 or 1).
	Response: <STX><ACK>$\mathbf{d_{B8}}$.......... $\mathbf{d_{B1}}$<ETX>
	d_B 1 = 1 → Program Run d_B 2 = 1 → Software Remote d_B 3 = 1 → Emergency limit switch of an axis d_B 4 = 1 → Power stage failure of an axis d_B 5 = 1 → Error programming (reset after status request) d_B 6 = 1 → Terminal is activated d_B 7 = 1 → SRQ has been set d_B 8 = 1 → Computer call
	System Status Extended
SE	Read system status in hexadecimal code. Two bytes (4 hexadecimal digits d_H) are available per axis: 1. + 2. byte for the x-axis, 3. + 4. byte for the y-axis.
	Response:<STX><ACK>$\mathbf{d_{HX}d_{HX}d_{HX}d_{HX}d_{HY}d_{HY}d_{HY}d_{HY}}$<ETX>
	Bit 0 = 1 → Power stage error Bit 1 = 1 → Power stage under voltage Bit 2 = 1 → Power stage overtemperature Bit 3 = 1 → Power stage is actived Bit 4 = 1 → Initiator – is activated (emergency stop) Bit 5 = 1 → Initiator + is activated Bit 6 = 1 → Step failure (only with option SFI = Step Failure Indication) Bit 7 = 1 → Encoder error Bit 8 = 1 → Motor stands still Bit 9 = 1 → Reference point is driven and OK (is reset at stop by initiator)
	(Bit 10 to Bit 15 not reserved)
	If Bit 0 to Bit 2 are set at the same time, no power stage is connected. Otherwise, only one error is possible at the time.

Instruction	***Meaning***
SH	**System Status Axes**
	Axis test with status axes output.
	Response:<STX><ACK> E <ETX>, if all axes are stopped.
	<STX><ACK> N <ETX>, if any axis is running.
	System Status Decimal
ST	Read system status as decimal number.
	Response: <STX><ACK>value <ETX> value = number between 0 and 255
	0 = End of program in the LOCAL MODE 1 = Program run 2 = Software Remote 4 = Emergency limit switch of an axis 8 = Power stage failure of an axis 16 = Error programming (reset after status request) 32 = Terminal or Enable is activated 64 = SRQ has been set 128 = Computer Mode
	Initiators
SUI	Read status of initiators (limit switches).
	Response:<STX><ACK>I=n <ETX>
	n = 0 → Axis is free, no initiator has reacted n = + → Initiator + direction has reacted n = – → Initiator – direction has reacted n = 2 → Both initiators have reacted (that means: wrong polarity of the initiators, broken wire or no 24 V supply voltage)
	Synchronous Start
S1	Prepare the synchronous start of the axes
S0	Execute the synchronous start of the axes.

2.17 Store Data into Flash EPROM

Instruction	***Meaning***
	Store programs and axis parameters (ONLY PC)
SA	Axis parameters are stored into the EPROM.

2.18 Time Loops

Instruction	***Meaning***
Tvalue **TR**nn **TR[R**nn**]**	The value for time loops (value, content of register nn or register [Rnn]) is preset in ms. The program waits here until the preset time has run out. **Response: <STX><ACK><ETX>** (ONLY PC)
TTSvalue **TTSR**nn **TTSR[R**nn**]**	The timer is loaded with a time (ms) value (value, content of register nn or register [Rnn]). The timer counts down to zero. The program is not interrupted. **Response: <STX><ACK><ETX>** (ONLY PC)
TT=0	The timer is compared with zero. If the timer is equal to zero the condition byte is set. Otherwise it is reset. Timer = 0 : the preset time is passed.
TT>value **TT>R**nn **TT>R[R**nn**]** **TT<**value **TT<R**nn **TT<R[R**nn**]**	The timer is compared with the preset value (value, content of register nn or register [Rnn]). If the timer value is higher/lower than the preset value (condition fulfilled) the condition byte is set. Otherwise it is reset. **Response: <STX><ACK>E<ETX>** or **<STX><ACK>N<ETX>** (ONLY PC)

2.19 Subroutines (ONLY PROG)

Instruction	***Meaning***
	Break Off Subroutine
UA	Break off all subroutines and set stack. The program can be continued with a jump instruction.
	End of Subroutine
UE	The subroutine is finished and the program is continued at the program line where this subroutine has been called.
	Call of Subroutine
Unn	The subroutine with the start line nn is called. The subroutine can be ended by means of the instruction UE**.**
URnn **UR[R**nn**]**	The register nn or [Rnn] contains the start line of the called subroutine. The subroutine is ended with the instruction UE.
U*la*	The subroutine starts at that line which is indicated by the label ***la***. The subroutine is ended by the instruction UE.
UP[name**]**	The subroutine *name* (start line number 1) is called. The subroutine is ended by the instruction UE.
UP[name**]N**nn	The subroutine *name* (start line number nn) is called. The subroutine is ended by the instruction UE.
UP[name**]NR**nn **UP[**name**]NR[R**nn**]**	The subroutine *name* starts at that program line which is stored in the register nn or [Rnn]. The subroutine is ended by the instruction UE.
UP[name**]N***la*****	The subroutine *name* starts at that line which is indicated by the label ***la***. The subroutine is ended by the instruction UE.
	Conditional Subroutine Call
	All instruction variants described above are available for the conditional subroutine call. The instructon call is only completed by the letter „E“ for condition fulfilled or „N“ for condition not fulfilled.
	"E" = Condition fulfilled
Uenn	see **U**nn, page 32
UERnn **UE[R**nn**]**	
UE*la*	see **U***la*, page 32
UEP[name**]**	see **UP[**name**]**, page 32
UEP[name**]N**nn	

Instruction	***Meaning***
UEP[name**]NR**nn **UEP[**name**]NR[R**nn **]**	
UEP[name**]N***la*	see **UP[**name**]N***la*****, page 32
	"N" = Condition not fulfilled
Unnn	see **U**nn, page 32
UNRnn **UNR[R**nn**]**	
UN*la*	see **U***la*, page 32
UNP[name**]**	see **UP[**name**]**, page 32
UNP[name**]N**nn	
UNP[name**]NR**nn **UNP[**name**]NR[R**nn **]**	
UNP[name**]N***la*	see **UP[**name**]N***la*****, page 32

2.20 Terminal Instructions (also by PC in case of terminal connection)

Instruction	***Meaning***
<>Wy	Erase text on line y (y=1 to 4)
<text>Wy	Display the text in line y from position 1 (y=1 to 4)
<text>Wy;m	Display the text in line y from position m (y=1 to 4; M=1 to 20) **Response: <STX><ACK><ETX>**

2.21 Axes Instructions

Instruction	***Meaning***
XC **YC**	Reset x-axis Reset y-axis **Response: <STX><ACK><ETX>** (ONLY PC)
	Axis Status Request
X=E **X#E**	Axis request on power stage error. Check (**=**) if a power stage error has occurred or check (**#**) if the power stage is operating normally. The error message “Failure” is requested.
X=H **X#H**	Axis request on stillstand. Check (**=**) if the axis is in standstill or check (**#**) if the axis is in motion. The condition byte is set when the condition is fulfilled. Otherwise it is reset.
X=I+ **X=I–**	Axis request on initiator status. The condition byte is set when the axis has come to a standstill at the initiator or the initiator is not connected. Otherwise it is reset.
X=M **X#M**	Axis request on power stage error. Check power stage (**=**), if a **Step failure** has occurred or has not (**#**) occurred. The condition byte is set, when the condition is fulfilled. Otherwise it is reset. This instruction applies only to control units with optional **S**tep **F**ailure **I**ndication (SFI) board.
X=N **X#N**	Axis request on emergency stop. Check (**=**) if the axis has come to a standstill (or not (**#**)) at an emergency switch. The condition byte is set when the condition is fulfilled. Otherwise it is reset. **Response: <STX><ACK>E<ETX>** or **<STX><ACK>N<ETX>** (ONLY PC)
	Wait until Set Point is reached
X>value **X>R**nn **X>R[R**nn**]**	The axis X is positioned and the program waits until the value of the counter XP21 is higher than the preset value (value, content of register nn or register [Rnn]). If the XP21 value is higher or the axis has come to a standstill the program is continued. **Example:** **Inr 005 XP21S0 XP14S2000 XL+** **Inr 006 X>5000 XP14S1000** **Inr 007 X>10000 XS XP14S2000**

Instruction	***Meaning***
	The axis is to be moved 10000 steps with 2000 Hz. After 5000 steps, the frequency is lowered to 1000 Hz and is set to 2000 Hz again after the standstill of the axis. At the instruction **X>5000** the program is stopped and will be continued after the position 5000 is reached or the axis has been stopped by an emergency stop.
X<value **X<R**nn **X<R[R**nn**]**	The axis x is positioned and the program waits until the value of the counter xP21 is lower than the preset value (value, content of register nn or register [Rnn]). If the xP21 value is lower or the axis has come to a standstill the program is continued. **Response: <STX><ACK><ETX>** (ONLY PC), if the axis has come to a standstill or the position condition is fulfilled. Otherwise the program waits.
	Switching Power Stages **Activate**
XMA	The power stage of axis X is activated.
	Deactivate
XMD	The power stage of axis X is deactivated.
	Axis parameter
XPmm**R**	The parameter mm of axis x is read out. (Only PROG) **Response : <STX><ACK>value<ETX>** mm = Parameter ID (ONLY PROG)
XPmm**S**value **XP**mm**SR**nn **XP**mm**SR[R**nn**]**	The parameter mm of axis x is loaded with the preset value (value, the content of register nn or register [Rnn]). mm = Parameter ID
	Initialisation/Reference Search Run To initialize an axis, a reference search run has to be carried out. The initiators, also called limit switches, serve as reference point. The axis moves to an initiator. When the initiator signal is identified, the motor stops and moves as long in the opposite direction until there is no more initiator signal. In case of initiator offset setting the offset distance is run and the axis is stopped. This point is called MØP (mechanical zero point) or reference point.
X0–	The axis moves to the initiator of the – direction.
X0+	The axis moves to the initiator of the + direction.
X0–I	The axis moves in – direction and stops with the zero pulse of the **i**ncremental encoder. Only incremental, no SSI Encoder!

Instruction	***Meaning***
X0+I	The axis moves in + direction and stops with the zero pulse of the **i**ncremental encoder. Only incremental, no SSI Encoder! **Response: <STX><ACK><ETX>** (ONLY PC)
X0–^I	The axis moves to the initiator of the – direction. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI Encoder!
X0 +^I	The axis moves to the initiator of the + direction. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI Encoder!
	Free Running
XLr	The axis is started and runs as long as it is stopped by the instruction xS or by a limit switch. r = + or – running direction
	Relative Positioning
Xrvalue **X**r**R**nn **X**r**R[R**nn**]**	The axis runs the distance relatively which is preset by value, the content of register Rnn or register [Rnn]. r = + or – running direction
	with stop instruction via input
Xrvalue**vE**nnz **X**r**R**nn**vE**nnz **X**r**R[R**nn**]vE**nnz	The axis runs relatively with its creep speed the distance which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops the positioning. r = + or – running direction z = S → input set z = R → input reset
Xrvalue**vvE**nnz **X**r**R**nn**vvE**nnz **X**r**R[R**nn**]vvE**nnz	The axis runs relatively with its high speed the distance which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops the positioning. r = + or – running direction z = S → input set z = R → input reset

Instruction	***Meaning***
	Absolute Positioning Related to the MØP
XArvalue **XA**r**R**nn **XA**r**R[R**nn**]**	The axis runs, in relation to the mechanical zero point MØP (XP20) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. r = + or – running direction
	with stop instruction via input
XArvalue**vvE**nnz **XA**r**R**nn**vvE**nnz **XA**r**R[R**nn**]vvE**nnz	The axis runs with high speed, in relation to the mechanical zero point MØP to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops axis run. r = + or – running direction z = S → input set z = R → input reset
	Absolute Positioning Related to the EL0P
Xervalue **XE**r**R**nn **XE**r**R[R**nn**]**	The axis runs, in relation to the electronical zero point (ELØP) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. r = + or– running direction z = S → input set z = R → input reset
	With stop instruction via input
XErvalue**vvE**nnz **XE**r**R**nn**vvE**nnz **XE**r**R[R**nn**]vvE**nnz	The axis runs with high speed, in relation to the electroncal zero point (ELØP) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops axis run. r = + or – running direction z = S → input set z = R → input reset
	Axis Stop
XS	All running instructions are cut off. The axis stops with the preset ramp.
XSN	The axis stopps with the preset emergency stop ramp (parameter P7).

2.22 Function Keys Read Out on Terminal BT24 (also by PC)

Instruction	***Meaning***
	Conditional Keyboard Read Out
#vFn	If the function key n is being depressed, the conditional byte is set. Otherwise it is reset. n = function key F1 to F6
#vnmx	If the key n or m or x is being depressed, the conditional byte is set. Otherwise it is reset. n, m, x = 0 to 9 (key 0 to 9) n, m, x = L (key CURSOR LEFT) n, m, x = R (key CURSOR RIGHT) n, m, x = U (key CURSOR UP) n, m, x = D (key CURSOR DOWN) n, m, x = H (key CURSOR HOME) n, m, x = B (key SCROLL) n, m, x = C (key CLEAR) n, m, x = E (key ENTER) n, m, x = P (key PRINT) n, m, x = ? (key ?) n, m, x = + (key +) n, m, x = – (key –) n, m, x = . (key .)

Example: **ZNR 005 #vH1? NN–0**
The BT24 keyboard is scanned as long as the key **H, 1** or **?** is beeing depressed. The conditonal byte is reset, if the keys **HOME,1** or **?** are not depressed. By the instruction **NN–0** the programm jumps to the line hold of line 5.

Important: The key ENTER is not defined for a read out.

Response: <STX><ACK> E <ETX> or
<STX><ACK> N <ETX> (ONLY PC)

3 List of Minilog Instructions

4 List of DIN Instructions

The controller programm can also be defined by the DIN instructions for process conditions and special functions. These standard instructions DIN 66025 can be used in one program with all the MINILOG instructions.

Instruction	***Meaning***
	G Instructions (Process Conditions)
G00, **G0**	Coordinate setting course Positioning with the speed as high as possible (high speed activation) without interpolation by parameter 14.
G01, **G1**	Set the linear interpolation
G04Tnn, **G4T**nn	Program interrupts with term, programed or defined in the controller The program continues automatically. n= in seconds with digits after the decimal point Abortion via input 2
G05, **G5**	Halt: the program waits for standstill of all axes and after that is continued
G20Lnn	Unconditional jump to line nn
G20L+nn	Unconditional jump by nn lines in + direction
G20L-nn	Unconditional jump by nn lines in – direction
G20*label**	Unconditional jump to *label*
G20L*label**	Unconditional jump to *label*
G20LP[name]	Unconditional jump to program name in line 1
G21z**L**nn	Conditional jump to line nn z=E or N
G21z**L+**nn	Conditional jump by nn lines in + direction z = E or N
G21z**L-**nn	Conditional jump by nn lines in – direction z = E or N
G21z***label**	Conditional jump to label z = E oder N
G21z**L***label**	Conditional jump to label z = E or N

Instruction	***Meaning***
G21z**LP**[name]	Conditional jump to program name in line 1 z = E or N
G22Lnn	Call the subroutine program nn Subroutine is marked by G98Lnn in the program
G22*label*****	Call the subroutine program *label*
G22P[name]	Call the subroutine program [name]
G23Lnn	Stop the subroutine at once and return to line nn
G23*label*****	Stop the subroutine at once and return to *label*
G74	Initialisation of all axes – direction
G74 x	Initialisation of one axis x= X or Y
G79Lnn	Automatic subroutine call at the end of the program line Subroutine is marked by G98Lxx in the program
G80	End of the automatic subroutine call G79
G90	Positioning absolut value in relation to the reference counter parameter 20
G91	Incremental positioning
G92	Set the absolute counter (zero offset) parameter 20
G98Lnn	Subroutine beginning and declaration nn Subroutine name maximum 6 characters
G99	Subroutine end
	M instructions (Additional Functions)
M00, **M0**	Programmed halt The program is continued by setting input 2
M01, **M1**	Programmed halt, if input 3 is ON The program is continued by setting input 2
M02, **M2**	End of program
M03, **M3**	Spindle ACTIVATED, clockwise rotation output 1 on; output 2 off
M04, **M4**	Spindle ACTIVATED, counterclockwise rotation

Instruction	*Meaning*
	output 1 off; output 2 on
M05, **M5**	Spindle quick STOP output 1 off; output 2 off
M07, **M7**	Cooling 2 on output 3 off; output 4 on
M08, **M8**	Cooling 1 on output 3 on; output 4 off
M09, **M9**	Cooling off output 3 off; output 4 off
M10	Tool holder on; output 5 on
M11	Tool holder off; output 5 off
M68	Clamp component; output 6 on
M69	Unclamp component ; output 6 off

5 Parameters

For operating a stepper motor controller several presettings as speed, acceleration ramps or waiting time are required. These presettings are called **Parameters**.

Default parameters are stored which can be used in several applications at delivery. You can read and edit these parameters with MiniLog-Comm.

Several counters are also contained in the list of parameters, which will be continuously actualized by the program. The counters can be read and some of them can be edited, too.

- For each axis separate parameters have to be set. Insert an X or Y to mark the axis in front of the parameter number (also valid: 1 or 2).

  Example: XP15 is the acceleration ramp value for axis X.

- Parameters (e.g. speeds) may be modified several times within a program, too.
- Parameter values can be entered or read.
- P48 and P49 can only be read.
- P19 to P22 are counters. They will be actualized by the program during axis movement.

5.1 List of Parameters

No.	Meaning	Default
P01	Type of movement 0 = rotational Rotating table, 1 limit switch for mechanical zero (referencing) 1 = linear for XY tables or other linear systems, 2 limit switches: Mechanical zero and limit direction – Limit direction +	0
P02	Measuring units of movement 1 = step 2 = mm 3 = inch 4 = degree	1
P03	Conversion factor for the thread 1 step corresponds to ... If P03 = 1 (steps) the conversion factor is 1. Computing the conversion factor: $Conversion\,factor = \frac{Thread}{Number of\ steps\ per revolution}$ <u>Example</u>: 4 mm thread pitch 200-step motor = 400 steps/rev. in the half step mode $Conversion\,factor = \frac{4}{400} = 0.01$	1
P04	Start/stop frequency The start/stop frequency is the maximum frequency to start or stop the motor without ramp. At higher frequencies, step losses or motor stop would be the result of a start or stop without ramp. The start/stop frequency depends on various factors: type of motor, load, mechanical system, power stage. The frequency is programmed in Hz.	400
P05 **P06**	not used	
P07	Emergency stop ramp The frequency is programmed in 4000-Hz/sec-steps.	100 000

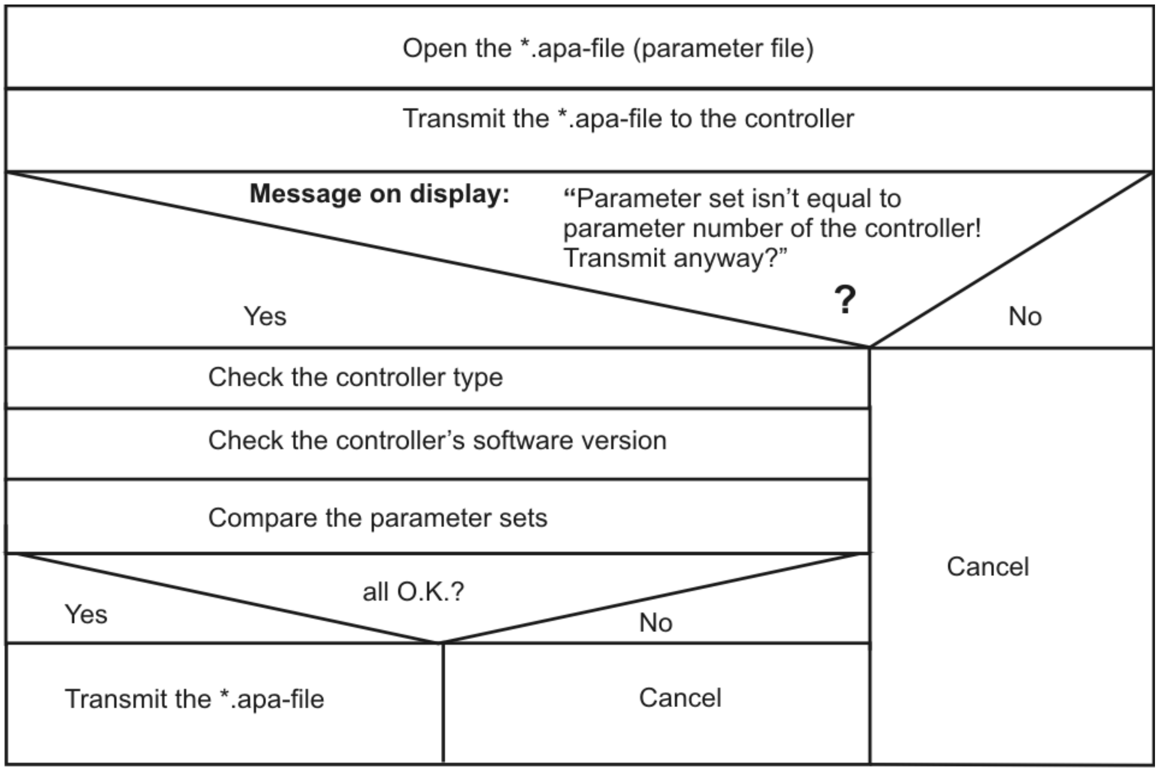
No.	Meaning	Default
P08	f_{max} MØP (mechanical zero point) Run frequency during initializing (referencing) Enter in Hz (integer value)	4000
P09	Ramp MØP Ramp during initializing, associated to parameter P08 Enter in 4000-Hz/sec-steps	4000
P10	f_{min} MØP Run frequency for leaving the limit switch range Enter in Hz	400
P11	MØP offset for limit switch direction + Distance between reference point MØP and limit switch activation Unit: is defined in parameter P02	0
P12	MØP offset for limit switch direction – Distance between reference point MØP and limit switch activation Unit: is defined in parameter P02	0
P13	Recovery time MØP Time lapse during initialization Enter in msec	20
P14	f_{max} Run frequency during program operation Enter in Hz (integer value) (40 000 maximum)	4000
P15	Ramp for run frequency (P14) Input in 4000-Hz/sec-steps (4000 to 500 000 Hz/sec)	4000
P16	Recovery time position Time lapse after positioning Input in msec	20
P17	Boost (defined in P42) 0 = off 1 = on during motor run 2 = on during acceleration and deceleration ramp <u>Remarks:</u> The boost current can be set in parameter P42. You can select with parameter P17 in which situation the controller switches to boost current. P17 = 1 means, the boost current always is switched on during motor run. During motor standstill the controller switches to stop current.	0

No.	Meaning	Default
P18	not used	
P19	Electronical zero counter Used for setting operating points. At standstill of the axis, P19 can be read or programmed during program execution.	0
P20	Mechanical zero counter This counter contains the number of steps referred to the mechanical zero (MØP). Can be read at axis standstill. If the axis reaches the MØP, P20 will be set to zero.	0
P21	Absolute counter Encoder, multi turn and also for single turn. The value of P22 is extended to P21 by software. The encoder counters have a fixed resolution, e.g. 10 bit (for single-turn encoders: the resolution is bits per turn), then the read value repeats. A saw tooth profile of the the numerical values is produced during a continuous motor running. This course is "straightened" by software. P20 and P21 will be scaled to the same value per revolution by P3 and P39 and are therefore directly comparable, see P36.	0
P22	Encoder counter Indicates the true encoder position.	0
P23	Axial limitation pos. direction + If the number of steps is reached, the run in + direction is aborted. 0 = no limitation	0
P24	Axial limitation neg. direction – If the number of steps is reached, the run in – direction is aborted. 0 = no limitation	0
P25	Compensation for play Indicates the step number, the target position in the selected direction is passed over and afterwards is started in reverse direction. 0 = no compensation for play	0
P26	not used	
P27	Initiator type 0 = PNP normally closed contact (NCC) 1 = PNP normally open contact (NOC)	0

No.	Meaning			Default
P 28 to **P33** not used				
P34	Encoder type 0 = no 1 = incremental 2 = serial interface SSI binary Code 3 = serial interface SSI Gray Code Connect the **correct** encoder type! Do not parameterize an incremental encoder as SSI. Danger of damage!			0
P35	Encoder resolution for SSI encoder Enter max. encoder resolution in bit (max. 31Bit)			10
P36	Encoder function 0 = counter			0
P37	not used			
P38	Encoder preferential direction of rotation 0 = + (positive) 1 = – (negative)			0
P39	Encoder conversion factor 1 increment corresponds to ...			1
		MCC-2 / MCC-1 in 0.1 A steps	**MCC-2 LIN in 0.04 A steps**	**MCC-1 or MCC-2 / MCC-2 LIN**
P40	**Stop current** Values Input	0 to 2.5 A 0 to 25	0 to 1 A 0 to 25	2 / 2
P41	**Run current** Values Input	0 to 2.5 A 0 to 25	0 to 1 A 0 to 25	6 / 6
P42	**Boost current** Values Input	0 to 2.5 A 0 to 25	0 to 1 A 0 to 25	10 / deactivated
P43	Current delay time in msec			20
P44	not used			
P45	Step resolution 1 to 256 1 = Full step 10 = 1/10 step 2 = Half step 16 = 1/16 step 4 = 1/4 step 128 = 1/128 step 8 = 1/8 step 256 = 1/256 step			4

No.	Meaning	Default
P46	Current Shaping (CS), also see appendix A 0 = Off 1 = On Recommended setting: P46 = 1	1
P47	Chopper frequency 0 = low 1 = high The chopper frequency value depends on P46: If P46 = 0, then is applied: P47 = 0: 16 kHz P47 = 1: 22.5 kHz If P46 = 1, then is applied: P47 = 0: 50 kHz P47 = 1: 75 kHz Recommende P47 = 1	1
P48	Power stage type (read only) 0 = linear 1 = chopper	(read only)
P49	Power stage temperature in °C (read only) (only for linear power stage type)	(read only)

5.2 Parameter Set Transmission to the Controller



6 Programming Example

6.1 General

Line number	Program	Comment
LNo1	E^1R2R NN+1 X=H NE+1 XS H A1R2R	Reading 2 inputs, if both are 0 and the motor is running, then stop the motor if not, continue to next line. If the motor is out of action, reset output 1 and 2
LNo2	E^1S2R NN+1 X=H NN+1 XL+ A1S	If the first input is 1 and motor is out of action, then start running in + direction and set output 1
LNo3	E^1R2S NN+1 X=H NN+1 XL- A2S	Input 2 = running in – direction and set output 2, if the motor runs.
LNo4	E^3S NN+1 X=H NN+1 N+3	If input 3 = 1 and motor is out of action, then reference run on initiator, then continue program in line 1.
LNo5	E^4S NN-4 X=H NN-4 N+3	If input 4 = 1 and motor is out of action, then positioning relatively.
LNo6	N1	Return to line 1
LNo7	X0- A3S H A3R N1	Execute reference run on initiator – direction and wait until motor is out of action, then return to line 1. Set output 3 during reference run
LNo8	X+1000 A4S	Positioning 1000 steps in + direction. Set output 4 during positioning.
LNo9	E^5S1 NN+1 XS H A4R N1	Wait here until input 5 1, then stop motor and return to program start, if positioning is finished.
LNo10	X=H NN-1 A4R N1	Positioning finished ? If yes, then reset output 4 and return to line 1

6.2 A/D Converter

Program	Comment
START	
R2SAD1	Register 2 is set with AD card 0 Ch 1
R3SAD2	Register 3 is set with AD card 0 Ch 2
R2W2	The value of register 2 is displayed in line 2
R3W3	The value of register 3 is displayed in line 3
N*START*	Jump back to Start

7 Storing Programs, Parameters and Registers

Programs and parameters can be edited with MiniLog-Comm, transferred to the controller and stored. During program run registers and counters can be modified by the program. As long as the controller is powered these data are stored. After switching off the controller, these data will be handled dependent on the built-in type of memory components:

Flash-EPROM Memory	Register or counter values modified by the program will not be stored when you switch off the controller. If these data are further required, they should be stored with MiniLog-Comm before switching off and transmitted to the controller again.
RAM Memory	The first 100 registers are stored in a nonretentive RAM Advantage: fast access Disadvantage: data get lost when powered off As from register 101 the data are stored in a serial RAM **(SRAM):** Advantage: data remain stored after powered off and are available after power on Disadvantage: slow access

8 Current Shaping CS

Current Shaping (CS) is a circuitry method for delivering a true phase current which corresponds for a wide range of frequencies to a selected current shape.
If the stepper motor is driven without CS, the true current differs from the specified current, even in the lower speed frequencies.

The 1/20 sine wave mode results in a current deviation as shown in the following figure, for average speed:

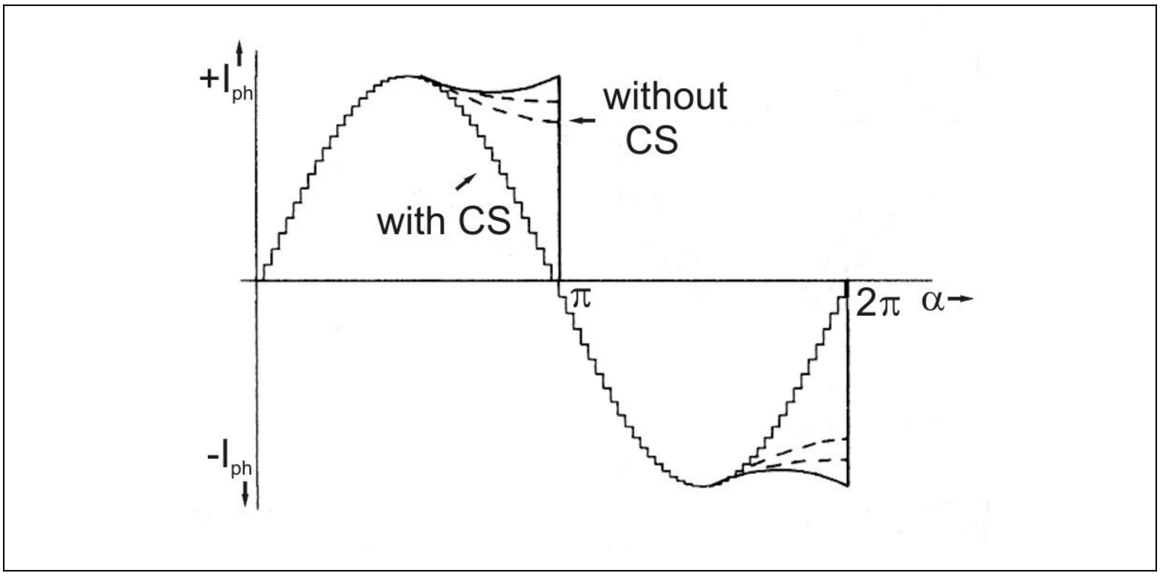


Fig. 1: Current shaping CS

These typical deformations can be observed for all types of curves. They are caused by the stepper motor inductance and the generator feedback which increases with the motor speed.

The resulting ‚current queue’ makes precise current regulation possible by Current Shaping (CS= 4 quadrant current regulation), only. The amplitude of the ‚current queue’ varies strongly remarked during one revolution and may provoke a motor resonance effect which causes step losses or desynchronization of the motor.

If the CS function is activated, the ‚current queue’ disappears and the resulting current is close to the ideal shape.

We recommend to use CS in higher current and speed frequency ranges.

The CS function can be activated by the parameter P46 (see chap. 5.1.).

9 Index

Phytron GmbH
Industriestraße 12 – 82194 Gröbenzell
T+49-8142-503-0   F +49-8142-503-190
www.phytron.eu