isel automation
Von der Komponente zum System

isel IMC4
Programming the IMC4 V2.xx

Software-Manual

970393 BE001
09/2000

The information, technical data and dimensions contained in this publication comply with the latest state of the art (1/1999). Nevertheless, any printing errors and mistakes cannot be ruled out. Suggestions for improvement and indications to errors are always welcomed.

The software and hardware designations of the corresponding companies, which are used in our publications, are generally either patented or protected by trademarks.

Table of Contents

1 Introduction

1.1 Preface

Our company ***isel****automation GmbH & Co.KG* has been known for drive systems and computerised numerical control systems (CNC) with stepper motors for many years. Stepper motors in the lower and medium performance ranges have generally gained acceptance all over the world. We can virtually not imagine state-of-the-art automation technology without drive systems in the medium accuracy and dynamic response range. As a rule, stepper motor control systems do not use closed-loop control circuits so that expensive sensor and electronic evaluation systems are not necessary. In addition to a simple design and start-up, this also provides a good price-performance ratio.

We have been offering control systems based on interface cards (interface cards 4.0/5.0/EP1090) for many years; they are used in many areas of production, automation, research and further education. In continuation of this product line, the IMC4 control system has been developed. The IMC4 is a micro-processor based Microstep control system designed to control up to 4 axes. For data transfer, the IMC4 is coupled via a serial interface, as our interface cards, too.

To maintain a certain compatibility, the commands to control the control system are used similarly to the CNC and DNC modes of the interface card series 4.0/5.0. This description will give you an overview of the CNC and DNC commands implemented in the control.

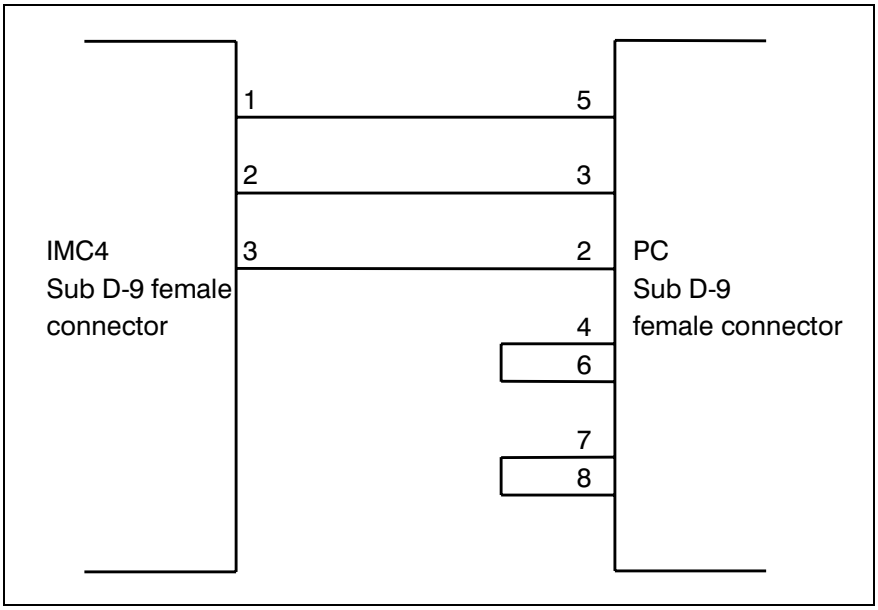
1.2 Connecting the Control System to a PC

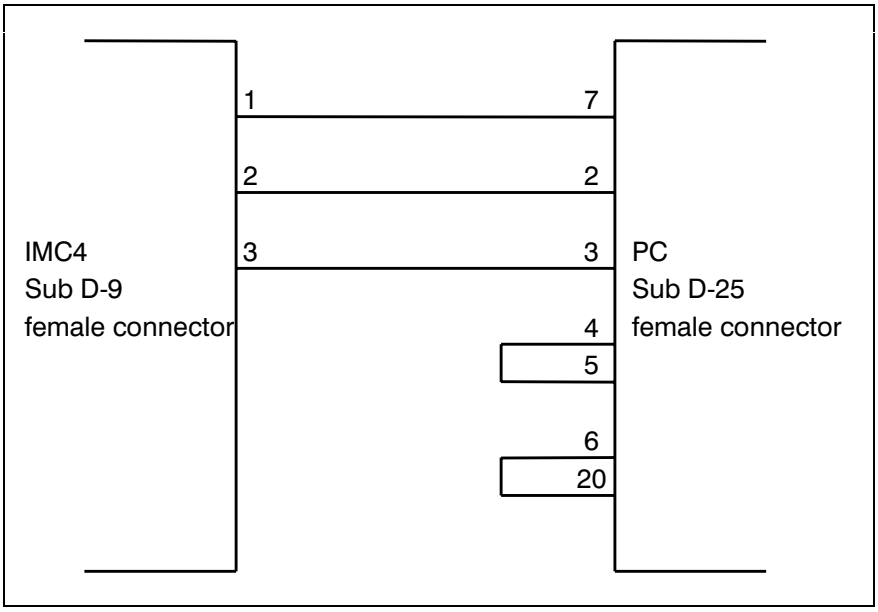
For data transfer between IMC4 and control computer, a serial interface to RS232 is used. The connection is provided via a 3-wire line. A software protocol provides for error-free transfer of the characters.

The following parameters are defined on the IMC4 as the data transfer parameters:

Baud rate:	19,200 (9,600 possible with appropriate jumper settings)
Data bit:	8
Stop bit:	1
Parity:	none

A special 3-wire connection is used as the connection line to the PC:





1.3 Conversion Factors

The distances to be traversed are transferred to the control system in steps. In contrast to the transfer of metric dimensions [mm], this kind of representation allows faster decoding and execution, since the control system represents all positions internally as steps. The conversion of the distances to be traversed has to be carried out by the control computer. For this conversion, the following information about the connected mechanical system is required:

Spindle lead:
- lead of the spindle installed in the mechanical system
- specifies the distance traversed by the slide (at a revolution of the spindle)
- leads of 2.5, 4, 5 and 10mm are typical

Steps/revolution:
- number of increments per motor revolution
- the IMC4 control system carries out the calculation in half-steps, i.e. 400 increments per motor revolution

Gear ratio:
- is only needed when geared stepper motor drives are used

The conversion from mm into steps can be carried out using the following formula:

$$\text{Steps} = \frac{\text{Dist. to be traversed}}{\text{Spindle lead}} * \text{Steps/rev.} * \text{gear ratio}$$

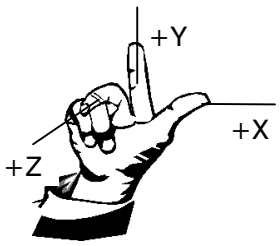
For velocities, the control system expects a specification in steps (increments) per second. To convert the velocity specifications usually used for mechanical systems (mm/s or m/min), the following formulas can be used:

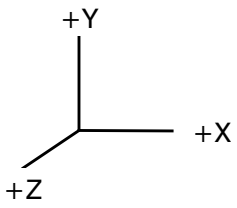
$$\text{V(step/sec)} = \frac{\text{V[mm/s]} * \text{steps/rev.} * \text{gear ratio}}{\text{spindle lead}}$$

$$\text{V(step/sec)} = \frac{\text{V[m/min]} * \text{steps/rev.} * \text{gear ratio}}{\text{spindle lead}} * \frac{60}{1000}$$

1.4 Coordinate System and Reference Points

The definition of the coordinate systems and the reference points is an essential prerequisite for programming machining movements within the working area of machine tools. Acc. to DIN 66217, a right-handed, right-angled coordinate system with the axes X, Y and Z is used. The Z axis is identical with the axis of the machining spindle. The positive direction of the Z axis extends from the workpiece towards the tool.





To create the machining movements, either the tool or the workpiece can be moved. But irrespective of that, the coordinate system will always refer to the workpiece. It is thus not essential for the programming whether the tool or the workpiece is moved. The programmer will always assume that the tool moves relatively to the workpiece not moving.

In addition to the coordinate system, reference points also play an important part in the programming of machine tools. The most important reference points will be mentioned here:

Machine zero:
- is fixed in the origin of the plant coordinate system
- is given by the plant design and cannot be changed

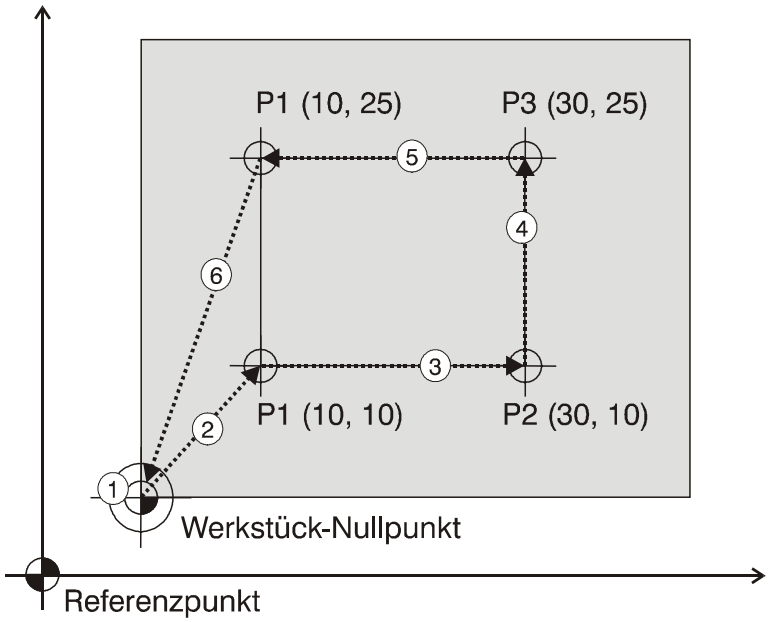
Reference point:
- point which is defined by limit switches and can be located by them; is in most cases situated in an outer corner of the working area and often identical with machine zero
- can also be set to a fixed distance to the machine zero and will then remain unchanged
- reference point approach is in most cases possible using an accuracy of one path increment

Workpiece zero:
- origin of the workpiece coordinate system
- can be freely selected by the programmer and is used as the reference point for the programming

For particulars and specifications regarding your plant, please refer to the relevant Operator's Guides and/or hardware and mechanical manuals.

1.5 Relative and Absolute Coordinate Specifications

Distances to be traversed can be programmed using either relative (incremental dimensions) or absolute coordinates.
When specifying relative coordinates, the reference point will be the current tool position. The coordinates will then constitute the distance to the point of the current position to be approached next. In contrast, when specifying absolute coordinates, the workpiece zero point will be used as the reference point. The coordinates will then constitute the distance from the point to be next approached from workpiece zero. This is illustrated by the example below:



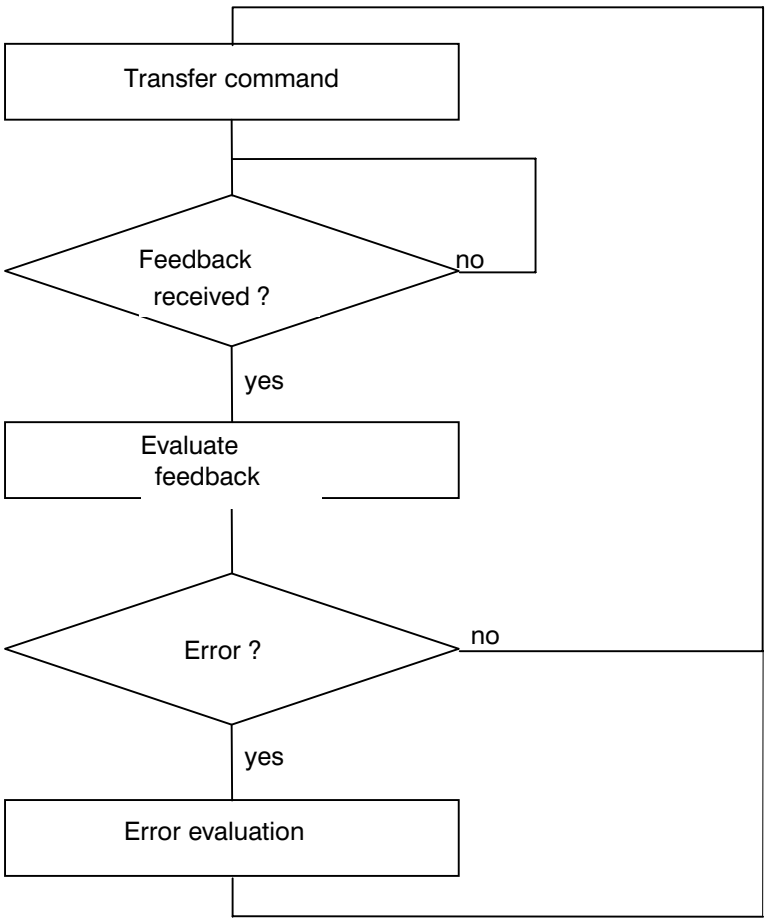
4 drill holes are to be made in a workpiece at the points P1, P2, P3 and P4. The points are to be approached in the order P1 --> P2 --> P3 --> P4.
For machining, the following relative and absolute coordinates will be used:

	absolute	relative
1	0, 0	0, 0
2	10, 10	10, 10
3	30, 10	20, 0
4	30, 25	0, 15
5	10, 25	-20, 0
6	0, 0	-10,-25

2 DNC Mode and its Commands

2.1 DNC Command Structure

In DNC mode, the data records and commands transferred by a control computer are evaluated and executed directly. To this aim, a so-called initialisation is required in the beginning of the data communication. This initialisation consists of the data opening character @, the device number (0 = default) and the number of the axes to be traversed. After the initialisation, the program steps are transferred to the control system individually and are directly executed by the control system. To check the data transfer or to signal any errors during the data transfer, appropriate ASCII characters are transmitted to the control computer as a feedback. This so-called software handshake procedure is realised as follows:



First, a command is transferred to the control system. The control system will decode and execute the command and will then generate either an appropriate acknowledgement or an error character.

This feedback is evaluated by the control computer. If an error has occurred, an error evaluation and elimination must be carried out. Then the next command can be transferred to the control system in the same way.

The scope of commands available in DNC mode of the IMC4 control system is described in the following.
The programming examples in Turbo Pascal and Microsoft C are intended for better understanding.

2.2 IMC4 Commands in DNC Mode

2.2.1 Initialisation; Setting the Number of Axes

Command: Set number of axes

Application: The control system is re-initialised by transferring the number of axes.

Structure: @<GN><axes><CR>

@	= data opening character
<GN>	= device number, default = 0
<axes>	= axis specification, see below
<CR>	= Carriage Return to compete the command

Notation: @07, @08

Explanation: The control system is addressed using @0; the numerical value following contains the axis configuration. Each axis is internally represented by a bit of a binary value, resulting in the following values:

1 --> X axis
3 --> X+Y axes
7 --> X+Y+Z axes
8 --> A axis

Restriction: The combinations @00, @02, @04, @06 and @09 are not permitted.
CAUTION: The A axis must always be initialised separately.

Programming example: see Appendix B 3.1. “Reference point approach; Initialising the Axes”

2.2.2 Reference Point Approach

Command: Reference point approach

Application: The control system will traverse all specified axes to their zero points (reference points). With *isel* systems, the reference points are always defined in a reasonable order, but can also be adapted accordingly using appropriate initialisation commands.

Structure: @<GN>R<axes><CR>

@ = data opening character
<GN> = device number, default = 0
R or r = reference point approach command
<axes> = axis specification, see below
<CR> = Carriage Return to complete the command

Notation: @0R7, @0r7, @0R8

Explanation: The control system is addressed using @0. "R" specifies that referencing is required. The following numerical value defines the axes to be referenced. Each axis is internally represented by a bit of the binary value, resulting in the following values:
1 --> X axis
2 --> Y axis
3 --> X+Y axes
4 --> Z axis
5 --> X+Z axes
6 --> Y+Z axes
7 --> X+Y+Z axes
8 --> A axis
The order of execution is defined as follows:
Z axis --> Y axis --> X axis --> A axis

After the reference point approach has been carried out, the control system will send its acknowledgement character and will wait for the commands to come. The control system will only be able again to execute commands after the reference point approach has been carried out by the connected mechanical system.

Restriction: This command can only be used after the control system has been initialised using the command “Set number of axes” and is limited to the axis configuration specified there. If the axes are specified not correctly, error message "3" will appear. If the control system is operated in 3D mode, the command will switch back to 2.5-dimensional mode.

CAUTION: The A axis must always be referenced separately.
CAUTION: If the reference point switch is not connected, the appropriate axis is selected continuously. It is, however, possible to simulate the reference point switch by pressing the STOP button.

Programming example: see Appendix B 3.1. Reference Point Approach, Initialising the Axes

2.2.3 Defining the Referencing Velocities

Command: Set referencing velocity

Application: This command defines the reference point approach velocity (further referred to as referencing velocity) for each axis separately.

Structure:

@<GN>d<Gx><CR>	(x)
@<GN>d<Gx>,<Gy><CR>	(x-y)
@<GN>d<Gx>,<Gy>,<Gz><CR>	(x-y-z)
@<GN>d<Gx>,<Gy>,<Gz>,<Ga><CR>	(x-y-z-a)

@	= data opening character
<GN>	= device number, default = 0
d	= command “Set referencing velocity”
<Gx>	= referencing velocity x
<Gy>	= referencing velocity y
<Gz>	= referencing velocity z
<Ga>	= referencing velocity a
<CR>	= Carriage Return to complete the command

Notation: @0d2500, @0d2400,3000, @0d1000,3000,2000, @0d1000,3000,2000,2000

Explanation: If no information regarding the referencing velocity is transferred by the control system, a default value will be used. Any values that have been changed will not be stored when the control system is turned off.

Restriction: The specified velocities must be within the valid range of values for the velocities.
A referencing velocity chosen too high, in conjunction with a spindle lead also chosen too high, may damage the reference point switches due to the existing mass inertia. The control system needs a switching hysteresis from the connected zero-position switch. This should be observed when connecting electronic zero sensors.

Programming example: see Appendix B 3.2. Setting the Referencing Velocity

2.2.4 Executing a Relative Movement

Command: Relative movement

Application: The control system will generate a relative movement depending on the transferred number of steps and on the transferred stepping velocity. The traversing movement is carried out immediately.

Structure:
@<GN>A<Sx>,<Gx>,<Sy>,<Gy>,<Sz1>,<Gz1>,<Sz2>,<Gz2><CR>
@<GN>A<Sx>,<Gx>,<Sy>,<Gy>,<Sz>,<Gz>,<Sa>,<Ga><CR>

@	= data opening character
<GN>	= device number, default = 0
A or a	= command “Relative Movement”
<Sx>	= number of steps x
<Gx>	= velocity
<Sy>	= number of steps y
<Gy>	= velocity
<Sz>, <Sz1>	= number of steps z
<Sz2>	= number of steps z, 2nd movement with 2.5D, 3 axes
<Gz>, <Gz1>	= velocity
<Gz2>	= velocity z, 2nd movement with 2.5D, 3 axes
<Sa>	= number of steps a, with 4 axes
<Ga>	= velocity, with 4 axes
<CR>	= Carriage Return to complete the command

Notation:

@0A 5000,900	(x axis only)
@0A 50,900,20,9000	(x and y axes)
@0A 30,800,10,900,4,90,-4,30	(x, y and z axes, with 3 axes)
@0A 30,800,10,900,4,90,-4,30	(x, y, z and a axes, with 4 axes)

Explanation: The control system is addressed using @0; "A" or "a" specify that a relative movement is to be carried out. The control system expects a number of pairs consisting of the number of steps and the velocity for each axis. The movement is carried out using relative dimensions, i.e. referred to the last position. The number of axes must match with the number of axes, i.e. one parameter pair for x operation, two parameter pairs for xy operation, four parameter pairs for xyz operation and four parameter pairs for xyza operation. The individual numbers must be separated by commas. For the z axis, two pairs of numbers are expected when 3 axes are used in 2.5D mode, since for machining applications the situation "Traverse, lower tool and then lift" often occurs. In 2.5D interpolation mode, first the movements of the x and y axes are carried out (interpolated linearly), then the z axis is traversed first by the values specified in z1 and then by the values specified in z2. If only one axis is required to be moved, the values for all axes initialised must nevertheless be transferred. When doing so, "0" must be specified as the number of steps for the axes not moved. After execution of this command, the control system will send the handshake character ("0") as the feedback. The control system will only be able again to execute new commands after this command has been executed.

Restriction: This command can only be used after the number of axes has been set. Apart from this, the control system will not check whether the movement goes beyond the admissible range of the connected mechanical system.

In 2.5D interpolation mode, the velocity specification of the axis with the longest way is accepted as the path velocity, and the velocity of the remaining axes is adapted accordingly depending on the distance ratio. In contrast to that, in 3D interpolation mode, the velocity specification of the x axis is used as the setting value for the path velocity.

Programming example: see Appendix B 3.3. Relative Movement

2.2.5 Executing an Absolute Movement

Command: Move to absolute position

Application: The control system will approach the specified position at the velocities specified. The traversing movement will be carried out immediately.

Structure:
@<GN>M<Sx>,<Gx>,<Sy>,<Gy>,<Sz1>,<Gz1>,<Sz2>,<Gz2><CR>
@<GN>M<Sx>,<Gx>,<Sy>,<Gy>,<Sz>,<Gz>,<Sa>,<Ga><CR>

@	= data opening character
<GN>	= device number, default = 0
M	= command "Absolute Movement"
<Sx>	= position x
<Gx>	= velocity
<Sy>	= position y
<Gy>	= velocity
<Sz>, <Sz1>	= position z
<Sz2>	= position z, 2nd movement always = 0
<Gz>, <Gz1>	= velocity
<Gz2>	= velocity
<Sa>	= position a, with 4 axes
<Ga>	= velocity, with 4 axes
<CR>	= Carriage Return to complete the command

Notation:

@0M 5000,900	(x axis only)
@0M 50,900,20,9000	(x and y axes)
@0M 30,800,10,900,4,90,0,30	(x,y and z axes, with 3 axes)
@0M 30,800,10,900,4,90,4,30	(x,y,z and a axes, with 4 axes)

Explanation: The control system is addressed using @0. "M" specifies that an absolute position will follow. For reasons of compatibility with the relative position command, two pairs of numbers are also here expected for the z axis when 3 axes are used. The second position specification of the z position must then, however, be zero and will be ignored. After the command has been carried out, the control system will response with the handshake character. The control system will only be able again to execute a new command if this command has been carried out.

Restriction: This command can only be used if the number of axes has been set. The control system will not check whether the movement goes beyond the range of the connected mechanical system.

Programming example: see Appendix B 3.4. Absolute Movement

2.2.6 Interrogating the Actual Position

Command: Position interrogation

Application: The control system will return the current actual position of all axes to the higher-level computer.

Structure: @<GN>P<CR>

@	= data opening character
<GN>	= device number, default = 0
P	= position inquiry command
<CR>	= Carriage Return to complete the command

Notation: @0P

Explanation: The control system is addressed using @0. "P" specifies that a position interrogation is carried out. The control system confirms this with the handshake character and then outputs the position values of all axes in the hexadecimal format (up to 3 axes: 18 hexadecimal digits, with 4 axes: 24 hexadecimal digits).

The structure of the returned position is as follows:

e.g.: 00010002000FFFFFFE for 3 axes

Position x = 000100, hexadec. in the complement on 2, corresponds to 256 dec.
Position y = 02000F, hexadec. in the complement on 2, corresponds to 131087 dec.
Position z = FFFFFE, hexadec. in the complement on 2, corresponds to -2 dec.

e.g.: 000B00044000000FFE003040 for 4 axes

Position x = 000B00, hexadec. in the complement on 2, corresponds to 2816 dec.
Position y = 044000, hexadec. in the complement on 2, corresponds to 278528 dec.
Position z = 000FFE, hexadec. in the complement on 2, corresponds to 4094 dec.
Position a = 003040, hexadec. in the complement on 2, corresponds to 12352 dec.

Restriction: This command can only be used if no traversing command takes place (if the plant is at a standstill). The control system cannot check whether the actual position corresponds to the current position of the connected mechanical system, since no control loop exists.

CAUTION: With a maximum of 3 axes, in all cases the positions for three axes are returned by the function, irrespective of the number of axes defined. In the case of four axes, in all cases positions for 4 axes are returned.

Programming example: see Appendix B 3.6. Position Interrogation

2.2.7 Setting Zero

Command: Set zero at the current position

Application: The control system will save the current position as the virtual zero point for the specified axis/axes. The next Absolute Movement commands will use this virtual zero point as the new reference point.

Structure: @<GN>n<axes><CR>

@	= data opening character
<GN>	= device number, default = 0
n	= command "Set Zero"
<axes>	= axis specification, see below
<CR>	= Carriage Return to complete the command

Notation: @0n7, @0n1, @0n8

Explanation: The control system is addressed using @0. "n" specifies that a zero offset is to be carried out. After the command, the control system are reported the axes for which a zero offset is to be carried out. Each axis is internally represented by a bit of a binary value, resulting in the following values:

1 --> X axis
2 --> Y axis
3 --> X+Y axes
4 --> Z axis
5 --> X+Z axes
6 --> Y+Z axis
7 --> X+Y+Z axes
8 --> A axis

After the command has been executed, the control system will send an appropriate feedback.

Restriction: The virtual zero point is only important for the Absolute Movement command. The virtual zero point has no influence on the relative positioning, since a relative traversing vector is specified here.

CAUTION: The zero offset for the A axis must always be carried out separately.

Programming example: see Appendix B 3.5. Zero Offset

2.2.8 Enabling/Disabling the 3D Interpolation

Command: Enable/disable 3D linear interpolation

Application: The control system extends the 2.5D interpolation of the standard operating system to 3-dimensional interpolation. This command can be used to enable/disable this interpolation depending on the particular task.

Structure: @<GN>z<status><CR>

@	= data opening character
<GN>	= device number, default = 0
z	= command "3D Interpolation"
<status>	= 0 --> disable, 1 --> enable
<CR>	= Carriage Return to complete the command

Notation: @0z1, @0z0

Explanation: The control system is prepared for the new command using the data opening character @0. "z1" will change the interpolation from 2D to 3D operation. This statement is modal, i.e. all relative and absolute movements are carried out three-dimensionally. The specification of z2 parameters with 3 axes are ignored in the case of these traversing movements. The velocity specification of the interpolation must be carried out with the x specification. In the case of 4 axes, the 4th axis is traced.

Programming example: see Appendix B 3.8. 3D Interpolation

2.2.9 Plane Selection for Circular Interpolation

Command: Plane selection

Application: Setting of the interpolation plane for the circular interpolation. Circles are only defined in one plane. The default plane for the circular interpolation is the XY plane. However, it is also possible to define any other plane configuration as the circle plane using the Plane Selection command.

Structure: @<GN>e<plane><CR>

@	= data opening character
<GN>	= device number, default = 0
e	= command "Set circle plane"
<plane>	= plane specification, see below
<CR>	= Carriage Return to complete the command

Notation: @0e1, @0e0

Explanation: The control system is addressed using @0. "e" specifies that the plane for circular interpolation is to be set. The following numerical value defines the plane as follows:

0 --> XY plane
1 --> XZ plane
2 --> YZ plane

Restriction: This command has modal effect, i.e. a plane selection for the circular interpolation remains stored until it is overwritten by a new plane selection.

Programming example: see Appendix B 3.7. Plane Selection

2.2.10 Setting the Circle Direction for the Circular Interpolation

Command: Set circle direction

Application: Setting of the circle direction for the circular interpolation. The circular interpolation is initiated by two successive commands. The first command defines the direction of the circle, and the second command (see 2.2.11.) transfers the interpolation parameters.

Structure: @<GN>f<direction><CR>

@	= data opening character
<GN>	= device number, default = 0
f	= command "Set "Circle Direction"
<direction>	= 0 --> CW (clockwise), -1 --> CCW (counter-clockwise)
<CR>	= Carriage Return to complete the command

Notation: @0f-1, @0f0

Explanation: The control system is addressed using @0. "f" specifies that the direction for the circular interpolation is to be set. The following numerical value defines the direction as follows:

0 --> CW (circular interpolation arc CW)
-1 --> CCW (circular interpolation arc CCW)

Restriction: The direction for the circular interpolation must always be programmed before any circular movement is programmed.

Programming example: see 2.2.11 Circular Interpolation, 2.3 Calculating the Circle Parameters, Appendix A 1, Software Routines for Calculating the Parameters with the Circle Command using Turbo Pascal
Appendix A 2. Software Routines for Calculating the Parameters with the Circle Command in C
Appendix B 3.7. Plane Selection

2.2.11 Circular Interpolation

Command: Circular interpolation

Application: Processing of circles and arcs at constant path velocity. The circular interpolation is initiated by two successive commands. The first command defines the circle direction (see 2.2.10.), and the second command transfers the interpolation parameters.

Structure: @<GN>y<B>,<V>,<D>,<Xs>,<Ys>,<Rx>,<Ry><CR>

@	= data opening character
<GN>	= device number, default = 0
<B>	= arc length in steps
<V>	= velocity
<D>	= interpolation parameters
<Xs>	= starting point x
<Ys>	= starting point y
<Rx>	= direction x
<Ry>	= direction y
<CR>	= Carriage Return to complete the command

Notation: @0y400,1500,119,-141,141,-1,-1

Explanation: The control system is addressed using @0. "y" specifies that a circular interpolation is to be carried out. The arc length specifies the length of the arc between starting and end points of the circular interpolation in steps. For the velocity, all integer values within the valid range of values for velocities are permitted. The interpolation parameter has to be transferred, because the control system cannot calculate this parameter by itself due to its memory capacity. The parameters Xs and Ys specify the starting point of the arc relatively to the circle centre point. Rx and Ry specify in which quadrant of the circle the interpolation starts. After the command has been executed, the control system will response with the handshake character ("0") as the feedback. The control system will only be able again to execute new commands if this command has been carried out.

CAUTION: To calculate the parameters, please refer to the Section "Calculating the Parameters for Circular Interpolation".

Restriction: This command can only be used if the number of axes has been set. Apart from that, the control system will not check whether the movement goes beyond the range of the connected mechanical system.

Programming example: see 2.2.10 Setting the Circle Direction for the Circular Interpolation
2.2.9 Plane Selection for Circular Interpolation,
2.3 Calculating the Circle Parameters,
Appendix A 1, Software Routines for Calculating the Parameters for the Circle Command in Turbo Pascal
Appendix A 2, Software Routines for Calculating the Parameters for the Circle Command in C

2.2.12 Starting a Stopped Movement

Command: Start

Application: A stopped movement will be continued.

Structure: @<GN>S<CR>

@	= data opening character
<GN>	= device number, default = 0
S or s	= Start command
<CR>	= Carriage Return to complete the command

Explanation: The control system is addressed using @0. "S" specifies that a stopped movement is to be restarted, completing the execution of the movement remaining. After the command has been executed, the control system will either response with the handshake character ("0") as the feedback or output an error message if no movement to be traversed is left in the memory. The control system will only be able again to execute new commands after the remaining movement has been traversed.

Restriction: This command makes only sense if a movement has been stopped and the control system is in Stop mode. In this mode, the current status of the START button can be interrogated using the Diagnosis command (see 2.2.19 Diagnosis), and it is also possible to create a Start command by the operating program when the START button is pressed. In DNC mode, the operation using the keys of the control system is only possible in this way, since the control system does not directly react to the START button.

Programming example: -

2.2.13 Reading Ports

Command: Read port

Application: This command can be used to determine the current status of logical or physical input ports via the serial interface.

Structure: @<GN>b<port No.><CR>

@	= data opening character
<GN>	= device number, default = 0
b	= command "Read Port"
<port No.>	= port number, see below
<CR>	= Carriage Return to complete the command

Notation: @0b0, @0b1

Explanation: The control system is addressed using @0. "b" specifies that the status of an input port is to be determined. Then the port number is transferred and the command is completed with Carriage Return. The control system will response with the software handshake "0" followed by two characters that specify a hexadecimal value corresponding to the current status of the input port. For the IMC4 control system, the following ports are defined with the following functionalities:

Port	Status	Function	
0	00 - FF	User I/O (also still possible with 65531)	
		Bit0	Stop button
		Bit1	Start button
		Bit2	User input 1
		Bit3	User input 2
		Bit4	Emergency Stop switch
		Bit5	ON button
		Bit6	Cover switch
		Bit7	Keyswitch
1	00	Cover is open	
	01	Cover is closed	
2	00	Spindle is switched off	
	01	Spindle is switched on	
3	00	Motor currents are switched off	
	01	Motor currents are switched on	

Restriction: The port states are only returned if the control system responses with the software handshake "0". The command can only be used if no movement is executed.

Programming example: see Appendix B 3.9. Reading Ports

2.2.14 Writing Ports

Command: Write port

Application: This command can be used to write defined values to logical or physical output ports via the serial interface.

Structure: @<GN>B<port No.>,<value><CR>

@	= data opening character
<GN>	= device number, default = 0
B	= command "Write Port"
<port No.>	= port number, see below
<Value>	= new port value
<CR>	= Carriage Return to complete the command

Notation: @0B1,1

Explanation: The control system is addressed using @0. "B" specifies that the value of an output port is to be set. The port number and the new port value are then transferred separated by a comma and the command is completed with Carriage Return. The control system responses with the software handshake "0" if the execution has been successful or with an error message if wrong port numbers and/or wrong values have been transferred. For the IMC4 control system, the following ports are defined with the following functionalities:

Port	Value	Function
0	0 - 255	User I/O (also still possible with 65529)
		Bit0 reserved
		Bit1 reserved
		Bit2 reserved (relay, spindle)
		Bit3 User output 1 (relay, 230 V, max. 100 W)
		Bit4 User output 2 (relay, not installed)
		Bit5 User output 3 (opto)
		Bit6 User output 4 (opto)
		Bit7 reserved
1	0	Cover may not be opened
	1	Cover may be opened
2	0	Spindle OFF
	1	Spindle ON
3	0	Motor currents OFF
	1	Motor currents ON

Restriction: The port values will only be overwritten if the control system responses with software handshake "0". This command can only be used if no movement is carried out. The reserved bits of port 0 cannot be overwritten by the user.

Programming example: see Appendix B 3.10. Writing Ports

2.2.15 Turning On/Off Test Mode

Command: Test mode ON/OFF

Application: This command can be used to turn on/turn off the test mode depending on the particular task.

Structure: @<GN>T<status><CR>

@	= data opening character
<GN>	= device number, default = 0
T	= command "Test Mode ON/OFF"
<status>	= 0 --> OFF, 1 --> ON
<CR>	= Carriage Return to complete the command

Notation: @0T1, @0T0

Explanation: The control system is prepared for a new command using the data opening character @0. "T1" is used to turn on the test mode, and "T0" is used to turn it off. After the command has been executed, the control system will response with the handshake character ("0"). In test mode, the control system will treat the reference point approach and the limit switches other than in normal mode. If a reference point approach command is received in normal mode, the control system will not carry out a reference point approach in the actual sense, but will set the current point as the reference point. The limit switches are continued to be monitored, but can be overtravelled. This is very useful if an axis stands on a limit switch after the plant has been turned on and must be cleared.

Restriction: This command can only be used if no movement is executed.

Programming example: -

2.2.16 Polling the Status Data of the Control System

Command: Poll status data

Application: Polling important status data of the control system in order to represent the current status, as well as for error location and diagnosis.

Structure: @<GN>H<CR>

@	= data opening character
<GN>	= device number, default = 0
H	= command "Poll Status Data"
<CR>	= Carriage Return to complete the command

Notation: @0H

Explanation: The control system is prepared for a new command using the data opening character @0. "H" causes the control system to send back information about the current status in plaintext format. At the end of this information, the control system will response with the handshake character ("0"). The information is already output in ASCII format formatted line by line so that it can be displayed, e.g. in a terminal window directly on the screen of a control computer. This information includes the status of the limit switches and of the plant control elements.

Restriction: To call this function, a sufficiently large receive buffer (min. 512bytes) must be provided in the control computer to make sure that no information is lost.

Programming example: -

2.2.17 Requesting the Version Data of the Control System

Command: Request version data

Application: Requesting important version data of the control system.

Structure: @<GN>V<CR>

@	= data opening character
<GN>	= device number, default = 0
V	= command "Interrogate Version Data"
<CR>	= Carriage Return to complete the command

Notation: @0V

Explanation: The control system is prepared for a new command using the data opening character @0. "V" causes the control system to send back information about the version of the control system in plaintext format. At the end of this information, the control system will response with the handshake character ("0"). The information is already output in ASCII format formatted line by line so that it can be displayed, e.g. in a terminal window directly on the screen of a control computer. This information includes the status of the limit switches and of the plant control elements.

Restriction: To call this function, a sufficiently large receive buffer (min. 512bytes) must be provided in the control computer to make sure that no information is lost.

Programming example: -

2.2.18 Initialising Parameters

Command: Initialise Parameters

Application: Initialising axis and referencing directions

Structure: @<GN>I<code><value><CR>

@	= data opening character
<GN>	= device number, default = 0
I	= Initialisation command
<code>	= ASCII characters to differ different parameters, see below
<value>	= new value for the parameter
<CR>	= Carriage Return to complete the command

Notation: @0ID3, @0IR1, @0IS1

Explanation: The control system is prepared for a new command using the data opening character @0. "I" tells the control system that an initialisation is to be carried out. The initialisation is followed by an identifier for the parameter and the new value, as well as by Carriage Return as the command end. The following parameters can be initialised:

Code	Value	Function
D	1	Negate direction of X axis
	2	Negate direction of Y axis
	4	Negate direction of Z axis
	8	Negate direction of A axis
		(all combinations are possible by adding the values above)
R	1	Negate referencing direction of X axis
	2	Negate referencing direction of Y axis
	4	Negate referencing direction of Z axis
	8	Negate referencing direction of A axis
		(all combinations are possible by adding the values above)
S	0-255	Set internal hardware status byte
		Bit assignment: Bit0: Enable/disable cover opening
		Bit1-7: Not assigned

Restriction: The internal hardware status byte should not be used by the user and is only mentioned here for the sake of completeness. Its use is reserved for appropriate drivers from iselautomation.

Programming example: -

2.2.19 Diagnosis

Command: Diagnosis

Application: Polling diagnosis data of the control system.

Structure: @<GN>D<code1><code2><CR>

@	= data opening character
<GN>	= device number, default = 0
D	= Diagnosis command
<code1>	= ASCII characters to differ different parameters, see below
<code2>	= ASCII characters to differ different parameters, see below
<CR>	= Carriage Return to complete the command

Notation: @0DRp, @0DRn, @0DS0

Explanation: The control system is prepared for a new command using the data opening character @0. "D" tells the control system that a diagnosis is to be carried out. The diagnosis is followed by two identifiers for the parameter and by Carriage Return as the command end. The control system will response with the software handshake "0" followed by two characters specifying a hexadecimal value that corresponds to the current value of the parameter. The following parameters can be polled:

Code1	Code2	Function
R	p	Poll positive limit switches
R	n	Poll negative limit switches
P	0	Poll input port 0
P	1	Poll input port 1
O	0	Poll output port 0
S	0	Poll hardware status

Bit assignment of the limit switches:

Bit0:	X axis
Bit1:	Y axis
Bit2:	Z axis
Bit3:	A axis

Bit assignment of the hardware status:

Bit0:	Status of spindle
Bit1:	Status of Start button
Bit2:	Status of Stop button
Bit3:	Status of Emergency Stop button
Bit4:	Status of keyswitch
Bit5:	Status of cover
Bit6:	Not used
Bit7:	Status of output stages/power supply

Restriction: The diagnostic functions should not be used by the user and are only mentioned here for the sake of completeness. The use is reserved for appropriate drivers from iselautomation.

Programming example: -

2.2.20 Carrying Out a Self-test

Command: Self-test

Application: This statement will initiate a self-test of the control system. This test includes the movement of the axes, as well as the interface test and the output of information about the version.

Structure: @<GN>?<CR>

@	= data opening character
<GN>	= device number, default = 0
?	= Self-test command
<CR>	= Carriage Return to complete the command

Notation: @0?

Explanation: The control system is addressed using @0. "?" specifies that a self-test is to be carried out. The command is completed with Carriage Return. The control system will then output information about the version, will test the movement of the motors and carry out an interface test. To test the interface, the control system will first output the ASCII character set. If a character is received via the interface, the control system will change to Echo mode and then return all characters received to the control computer.

Restriction: To call this function, a sufficiently large receive buffer must be provided in the control computer to make sure that no information is lost. Therefore, it only makes sense to call this function within a terminal program or a terminal function. The self-test can only be quitted by turning off the control system or by software reset (char(254)). The plant must then be re-initialised.

Programming example: -

The self-test can also be initiated if the Start button is pressed when turning on the control system and is released only after the self-test has been started.

2.2.21 Check and Control Codes

Check and control codes provide direct access to the functional sequence of the control system via the serial interface. The commands sent are evaluated directly in the control system's receive routine and are then executed. Special control codes are provided for the following functionalities:

Function: Software stop char(253)

A positioning movement in DNC mode (relative or absolute) can be stopped by a stop command without any step losses. A start command executed thereafter (by transferring @0S, see 2.2.12 "Starting a Stopped Movement") will complete the interrupted functional sequence. Furthermore, it is possible to read back the currently reached position after a stop command using the command "Interrogate Position". This functionality can also be achieved by pressing the Stop button. If a movement has been successfully stopped, the control system will create an additional feedback signal "F".

The function is called by transferring char(253) via the serial RS232 interface.

Function: Software reset char(254)

The control system interrupts all activities immediately and carries out a software reset internally. The plant must then be re-initialised and referencing be carried out.

The function is called by transferring char(254) via the serial RS232 interface.

Function: Software break char(255)

A positioning movement in DNC mode (relative or absolute) can be completed using a break command. This means that the remainder of the movement will be lost.

The function is called by transferring char(255) via the serial RS232 interface.

see also Chapter 4 "Error Messages of the IMC4"

2.3 Calculating the Circle Parameters

2.3.1 The Parameters for the Circular Interpolation

The circular interpolation is initiated by two successive commands (see 2.2.10, 2.2.11). The first command defines the circle direction, and the second command is used to transfer the interpolation parameters.

Circle direction: @<GN>f<direction><CR>

@	= data opening character
<GN>	= device number, default = 0
f	= command "Set Circle Direction"
<direction>	= 0 --> CW (clockwise), -1 --> CCW (counter-clockwise)
<CR>	= Carriage Return to complete the command

Circular interpolation: @<GN>y<B>,<V>,<D>,<Xs>,<Ys>,<Rx>,<Ry><CR>

@	= data opening character
<GN>	= device number, default = 0
<B>	= arc length in steps
<V>	= velocity
<D>	= interpolation parameter
<Xs>	= starting point x
<Ys>	= starting point y
<Rx>	= direction x
<Ry>	= direction y
<CR>	= Carriage Return to complete the command

An especially adapted differential algorithm acc. to Bresenham is used internally in the control system to create arcs. This kind of algorithms is very often used in micro-processor applications since high processing speeds are achieved here with a low calculation expenditure.

The meaning and how to calculate the parameters for circular interpolation will be explained in the following. An appropriate example is to be found in Section 2.3.6. of this Description.

2.3.2 Calculating the Arc Length

The arc length <B> specifies the length of the arc between the starting point and the end point of the arc in steps and is used internally in the control system as a running variable for the differential algorithm. The calculation of the arc length can be carried out in different ways and will be explained in the following.

- Simple approximation formula
  For simple circle applications that contain, e.g. only quadrants, semi or full circles, the arc length can be calculated using the following formula:

$$B = 4 * R * \frac{E - A}{Pi}$$

B – arc length in steps
R – arc radius in steps
A – starting angle in arc dimension
E – end angle in arc dimension

The result must be rounded to the next integer value. To eliminate the inaccuracy for processing on the control system, the next positioning movement should be programmed as an absolute movement.

- Calculation using the software routine

An exact calculation of the arc length can be achieved using simple software routines. Appropriate examples are to be found in the C and Pascal routines in the Appendix (see A 1, A 2) of this Description.

2.3.3 Calculating the Interpolation Parameter

The interpolation parameter is used by the control system as the starting value for the differential register of the algorithm used by the control system to create a circle. The calculation of the parameter is carried out on the PC side using appropriate software routines. This will take unnecessary burden of calculation from the control system, thus increasing the processing speed. Sample routines to calculate the parameter are to be found in the Appendix (see A 1. and A 2.) of this Description.

2.3.4 Arc Starting Point

The starting point of the arc constitutes the distance from the circle centre along X and Y in steps using relative coordinates (i.e. the circle centre point is assumed as an imaginary centre point for the calculation). The calculation can be carried out using the appropriate circle functions.

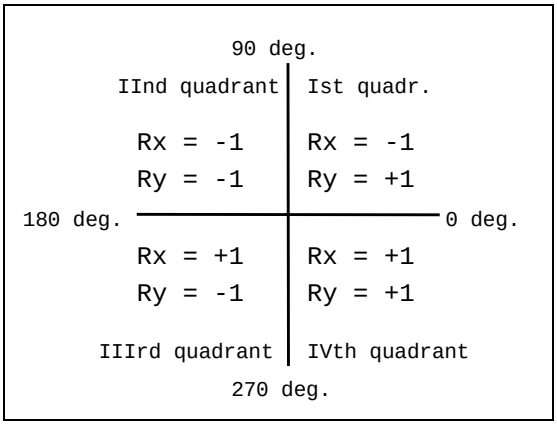
Xs - X coordinate of the starting point relative to the centre point
Ys - Y coordinate of the starting point relative to the centre point
R - radius in steps
A - starting angle as an arc

$$Xs = R * \cos(A)$$
$$Ys = R * \sin(A)$$

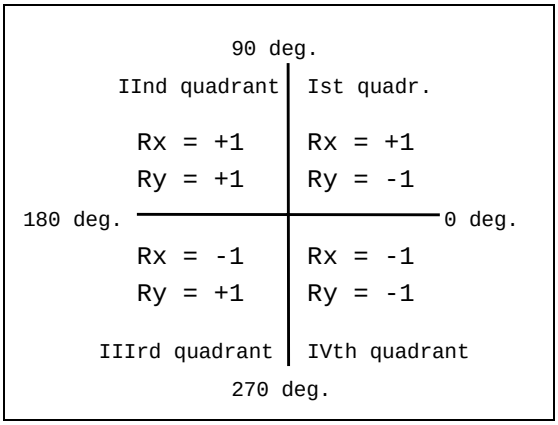
2.3.5. Directions in the Starting Point of the Circular Interpolation

To carry out the interpolation algorithm, the control system needs an information on in which quadrant the arc starts and which signs are to be used internally in the control system for certain calculations. This information is provided to the control in the form of the parameters Rx and Ry. This is carried out based on the following definitions:

Arcs CCW



Arcs CW



2.3.6 Calculation Example for Circular Interpolation

An example will be calculated in the following to illustrate how to calculate the circle command parameter.

An arc CCW with a radius of 200 steps is to be traversed at a velocity of 1,500 steps/s. The starting angle is 135 degrees, the end angle 225 degrees. Please note that all path specifications must be specified in steps and all angle specifications in arc dimension to carry out the calculation.

Given:	Radius	R = 200	Searched:	Arc length	B
	Starting angle	A = 135*Pi/180 = 2.3562		Starting point X	Xs
	End angle	E = 225*Pi/180 = 3.9267		Starting point Y	Ys
	Speed	V = 1500		Direction X	Rx
	Direction	CCW		Direction Y	Ry
				Interpolation parameter	D

Arc length B (see 2.3.2.):

B = 4 * R * (E - A)/Pi
B = 4 * 200 * (3.9267 - 2.3562) / Pi
B = 4 * 200 * 0.4999 = 399.9245
B = 400

Starting point Xs and Ys (see 2.3.4):

Xs = R * cos(A) = 200 * cos(2.3562) = -141.4221
Xs = -141

Ys = R * sin(A) = 200 * sin(2.3562) = 141.4205
Ys = 141

Directions Rx and Ry (see 2.3.5):

Starting angle 135 degrees, direction of rotation CCW
Rx = -1 Ry = -1

Interpolation parameter D (see 2.3.3.):

D = (Rx*Ry*R + Rx*Ry*total(R-1) - Rx*total(Xs+(Rx-Ry)/2) + Ry*total(Ys+(Rx+Ry)/2))/2

Total(R-1) = total(199) = 199*(199+1) = 39800

Total(Xs+(Rx-Ry)/2) = total(-141+(-1-(-1))/2) = total(-141) = 141*(-141+1) = -19740

Total(Ys+(Rx+Ry)/2) = total(141+(-1-(-1))/2) = total(141) =141*(141+1) = 20022

D = ((-1)*(-1)*200 + (-1)*(-1)*39800 - (-1)*(-19740) + (-1)*20022) / 2

D = (200 +39800 - 19740 - 20022) / 2 = 119

The appropriate commands would be:

@0f-1
@0y400,1500,119,-141,141,-1,-1

see also: 2.2.10 Setting the Circle Direction for the Circular Interpolation, 2.2.11 Circular Interpolation, Appendix A 1 Software Routines for Calculating the Parameters with the Circle Command in Turbo Pascal, Appendix A 2 Software Routines for Calculating the Parameters with the Circle Command in C

3 CNC Mode and Its Commands

3.1 CNC Command Structure

Operated in CNC mode, the control system stores all transmitted commands in the internal data memory. For activation, the command "Store CNC data field" must be transferred after the standard initialisation. Then the data field is transferred and completed with the command "End of data field".

The program can now be enabled without any further communication with the control computer using an external Start command (pressing the Start button).

As a memory medium, FlashPROMs (non-volatile, electrically programmable and erasable memories) are used on the IMC4. Similarly to EPROMs, these memories are programmed in the system with the appropriate information by certain programming cycles. Erasing them corresponds to programming them with a default value. The appropriate cycles are carried out by the memory circuits automatically. While these cycles are executed, no access to the circuits is possible so that the programming and erasure require relatively much time. Before these chips are reprogrammed, they have generally to be deleted. To do so, an erasing cycle must be carried out by the IMC4 as follows:

- Operate the emergency switch and release it again
- Turn the key switch to the ON position
  !!! The key can't be removed in this position
- Depress the STOP button and hold it in this position
- Now, with the STOP button depressed, depress the ON button
- After 20 seconds, the internal CNC memory is erased
- Switch off the machine with the main

After erasure, the new program can be written to the memories. If a program or part of the program has already been stored in the memories, the command "Store CNC data field" will result in an error message.

The commands of the IMC4 control system will be listed below and be explained in brief. For a detailed explanation of some commands, please refer to the appropriate DNC mode command, since the meanings and number of the parameters often correspond to those in DNC mode.

If an error has occurred when transmitting and storing a CNC data field, the CNC program stored until then will be marked as invalid and can not be executed. In this case, the error in the program must be eliminated accordingly and the FlashPROM be erased before the data field can be transferred anew for saving.

3.2 The Commands of the IMC4 in CNC Mode

3.2.1 Saving the CNC Data Field

Command: Save CNC data field

Application: This statement is intended to initialise the transfer of storable commands and must be programmed in the beginning of CNC mode.

Structure: @<GN>i<CR>

@	= data opening character
<GN>	= device number, default = 0
i	= command "Save CNC data field"
<CR>	= Carriage Return to complete the command

Notation: @0i

Explanation: The control system is addressed using @0. "i" specifies that a CNC data field is to be stored. The command is completed with Carriage Return. Then, until the End-of-Data Field command comes or an error occurs, the control system will only accept CNC commands. The command is acknowledged with an appropriate feedback signal. All of the following storable commands are stored in the FlashPROM.

Restriction: This command can only be used if the control system has been initialised and no movement is being carried out. If a program is already stored in the memories, an error message is output.

Programming example: see Appendix D

3.2.2 Reference Point Approach in CNC Mode

Command: Reference point approach

Application: The control system will save a movement of all axes towards their zero points (reference points). With *isel* systems, the reference points of the axes are always defined in a reasonable default order, but can be adapted accordingly using appropriate initialisation commands.

Structure: 7<axes><CR>

7	= command code for reference point approach
<axes>	= axis specification, see below
<CR>	= Carriage Return to complete the command

Notation: 77, 78

Explanation: "7" specifies that referencing is to be carried out. The following numerical value defines the axes that will carry out a reference point approach. When doing so, each axis is internally represented by a bit of a binary value, resulting in the following values:

1 --> X axis
2 --> Y axis
3 --> X+Y axes
4 --> Z axis
5 --> X+Z axes
6 --> Y+Z axes
7 --> X+Y+Z axes
8 --> A axis

The order of execution is defined as follows:
Z axis --> Y axis --> X axis --> A axis

After the reference point approach has been carried out, the next CNC command is read from the memory and executed.

Restriction: This command is limited to the axis configuration initialised. If wrong axes are specified, error message "3" will occur. If the control system is operated in 3D mode, this command will switch back to 2.5-dimensional mode.

CAUTION: The A axis must always be referenced separately.

CAUTION: If the reference point switch is not connected, the appropriate axis is selected continuously. It is, however, possible to cancel referencing by pressing the Stop key.

Programming example: see Appendix D

3.2.3 Relative Movement in CNC Mode

Command: Relative movement

Application: The control system will store a relative movement according to the transferred number of steps and the transferred stepping velocity.

Structure:
0<Sx>,<Gx>,<Sy>,<Gy>,<Sz1>,<Gz1>,<Sz2>,<Gz2><CR>
0<Sx>,<Gx>,<Sy>,<Gy>,<Sz>,<Gz>,<Sa>,<Ga><CR>

0	= command code for relative movement
<Sx>	= number of steps x
<Gx>	= velocity
<Sy>	= number of steps y
<Gy>	= velocity
<Sz>, <Sz1>	= number of steps z
<Sz2>	= number of steps z, 2nd movement with 2.5D, 3 axes
<Gz>, <Gz1>	= velocity
<Gz2>	= velocity z, 2nd movement with 2.5D, 3 axes
<Sa>	= number of steps a, with 4 axes
<Ga>	= velocity, with 4 axes
<CR>	= Carriage Return to complete the command

Notation:

05000,900	(x axis only)
050,900,20,9000	(x and y axes)
030,800,10,900,4,90,-4,30	(x,y and z axes, with 3 axes)
030,800,10,900,4,90,-4,30	(x,y,z and a axis, with 4 axes)

Explanation: "0" specifies that a relative movement is to be carried out. The control system will now expect a number of pairs for each axis, which consists of number of steps and velocity. The distances are specified using relative dimensions, i.e. with reference to the last position. The number of positions must match with the number of axes, i.e. one parameter pair in x mode, two parameter pairs in xy mode, four parameter pairs for xyz mode and four parameter pairs for xyza mode. The individual numbers must be separated by commas. For the z axis, two pairs of numbers are expected when 3 axes are used in 2.5D mode, since for machining applications the situation "Traverse, lower tool and then lift" often occurs. In 2.5D interpolation mode, first the movements of the x and y axes are carried out (interpolated linearly), then the z axis is traversed first by the values given in z1 and then by the values specified in z2. If only one axis is required to be moved, the values for all axes initialised must nevertheless be transferred. When doing so, "0" must be specified as the number of steps for the axes not moved. After execution of this command, the control system will send the handshake character ("0") as the feedback.

Restriction: The control system will not check whether the movement goes beyond the admissible range of the connected mechanical system.

In 2.5D interpolation mode, the velocity specification of the axis with the longest way is accepted as the path velocity and the velocity of the remaining axes be adapted according to the distance ratio. In contrast to that, in 3D interpolation mode, the velocity specification of the x axis will be used as the setting value for the path velocity.

Programming example: see Appendix D

3.2.4 Absolute Movement in CNC Mode

Command: Move to absolute position

Application: The control system will store an absolute movement according to the specified velocities and positions.

Structure:
m<Sx>,<Gx>,<Sy>,<Gy>,<Sz1>,<Gz1>,<Sz2>,<Gz2><CR>
m<Sx>,<Gx>,<Sy>,<Gy>,<Sz>,<Gz>,<Sa>,<Ga><CR>

m	= command code for absolute movement
<Sx>	= position x
<Gx>	= velocity
<Sy>	= position y
<Gy>	= velocity
<Sz>, <Sz1>	= position z
<Sz2>	= position z, 2nd movement always = 0
<Gz>, <Gz1>	= velocity
<Gz2>	= velocity
<Sa>	= position a, with 4 axes
<Ga>	= velocity, with 4 axes
<CR>	= Carriage Return to complete the command

Notation:

m5000,900	(x axis only)
m50,900,20,9000	(x and y axes)
m30,800,10,900,4,90,0,30	(x,y and z axis, with 3 axes)
m30,800,10,900,4,90,4,30	(x,y,z and a axes, with 4 axes)

Explanation: "m" specifies that an absolute position will follow. For reasons of compatibility with the relative positioning command, for the z axis two pairs of numbers are expected also here when 3 axes are used. The second position specification of the z position, however, must then be zero and will be ignored. After storing, the control system will response with the handshake character.

Restriction: The control system will not check whether the movement goes beyond the admissible range of the connected mechanical system.

Programming example: see Appendix D

3.2.5 Setting Zero in CNC Mode

Command: Set zero at virtual point

Application: The control system will store a command to set the current position as the virtual zero point for the specified axis/axes when executing the CNC program. The next following Traverse Absolutely movements will then refer to this virtual zero.

Structure: n<axes><CR>

n	= Set Zero command code
<axes>	= axis specification, see below
<CR>	= Carriage Return to complete the command

Notation: n7, n1, n8

Explanation: "n" specifies that a zero offset is to be carried out. After the command code, the control system are communicated the axes for which a zero offset is to be carried out. Each axis is internally represented by a bit of a binary value, resulting in the following values:

1 --> X axis
2 --> Y axis
3 --> X+Y axes
4 --> Z axis
5 --> X+Z axes
6 --> Y+Z axes
7 --> X+Y+Z axes
8 --> A axis

After storing, the control system will response with an appropriate feedback.

Restriction: The virtual zero point is only important for the Absolute Movement command. The virtual zero point has no influence on the relative positioning, since a relative traversing vector is specified here.

CAUTION: The zero offset for the A axis must always be carried out separately.

Programming example: see Appendix D

3.2.6 Enabling/Disabling 3D Interpolation in CNC Mode

Command: Enable/disable 3D-linear interpolation

Application: The control system will save the command in order to be able to extend the 2.5D interpolation of the operating system to 3-dimensional interpolation. This command can be used to enable/disable this interpolation depending on the particular task.

Structure: z<status><CR>

z	= 3D interpolation command code
<status>	= 0 --> disable, 1 --> enable
<CR>	= Carriage Return to complete the command

Notation: z1, z0

Explanation: "z1" is used to change the interpolation from 2D to 3D operation. The statement is modal, i.e. all relative and absolute movements are carried out three-dimensionally. The specification of z2 parameters with 3 axes will be ignored in the case of these traversing movements. The velocity specification of the interpolation must be carried out with the x specification. In the case of 4 axes, the 4th axis are traced.

Programming example: see Appendix D

3.2.7 Plane Selection for Circular Interpolation in CNC Mode

Command: Plane selection

Application: Setting of the interpolation plane for the circular interpolation. Circles are only defined within one plane. The default plane for the circular interpolation is the XY plane. However, it is also possible to define any plane configuration other than the circle plane using the Plane Selection command.

Structure: e<plane><CR>

e	= command code for "Set Circle Plane"
<plane>	= plane specification, see below
<CR>	= Carriage Return to complete the command

Notation: e1, e0

Explanation: "e" specifies that the plane for the circular interpolation is to be set. The following numerical value defines the plane as follows:

0 --> XY plane
1 --> XZ plane
2 --> YZ plane

Restriction: This command has modal effect, i.e. a plane selection for the circular interpolation remains stored until it is overwritten by a new plane selection.

Programming example: see Appendix D

3.2.8 Setting the Circle Direction for the Circular Interpolation in CNC Mode

Command: Set Circle Direction

Application: Setting the circle direction for the circular interpolation. The circular interpolation is initiated by two successive commands. The first command defines the direction of the circle, and the second command (see 3.2.9.) transfers the interpolation parameters.

Structure: f<direction><CR>

f	= command "Set "Circle Direction"
<direction>	= 0 --> CW (clockwise), -1 --> CCW (counter-clockwise)
<CR>	= Carriage Return to complete the command

Notation: f-1, 0f0

Explanation: The control system is addressed using @0. "f" specifies that the direction for the circular interpolation is to be set. The following numerical value defines the direction as follows:

0 --> CW (circular interpolation arc CW)
-1 --> CCW (circular interpolation arc CCW)

Restriction: The direction for the circular interpolation must always be programmed before any circular movement is programmed.

Programming example: see Appendix D

3.2.9 Circular Interpolation in CNC Mode

Command: Circular interpolation

Application: Saving motion commands for circles and arcs at constant path velocity. The circular interpolation is initiated by two successive commands. The first command defines the circle direction (see 3.2.8.), and the second command transfers the interpolation parameters.

Structure: y<B>,<V>,<D>,<Xs>,<Ys>,<Rx>,<Ry><CR>

y	= command code for circular interpolation
<B>	= arc length in steps
<V>	= velocity
<D>	= interpolation parameter
<Xs>	= starting point x
<Ys>	= starting point y
<Rx>	= direction x
<Ry>	= direction y
<CR>	= Carriage Return to complete the command

Notation: y400,1500,119,-141,141,-1,-1

Explanation: "y" specifies that a circular interpolation is to be saved. The arc length specifies the length of the arc between starting and end points of the circular interpolation in steps. For the velocity, all integer values within the valid range of values for velocities are permitted. The interpolation parameter has to be transferred, because the control system cannot calculate this parameter by itself due to its memory capacity. The parameters Xs and Ys specify the starting point of the arc relatively to the circle centre point. Rx and Ry specify in which quadrant of the circle the interpolation starts. After the command has been executed, the control system will response with the handshake character ("0") as the feedback.

CAUTION: For calculating the parameters, please read the Section "Calculating the Parameters for Circular Interpolation".

Restriction: The control system will not check whether the movement goes beyond the admissible range of the connected mechanical system.

Programming example: see Appendix D

3.2.10 Loops and Branches in CNC Mode

Command: Loop, branch

Application: Saving loops and branches. Loops are intended to summarise movement sequences of the same kind. Thanks to this feature, the memory available in the control system is used more efficiently. Branches can be used to jump after a logical decision to a certain block in the program.

Structure: 3<number>,<offset><CR>

3	= command code for loop, branch	
<number>	= number of loops	
	Loop:	0 < number of loops < 32768
	Branch:	always 0
<offset>	= branch destination	
	Loop:	-1 >= branch destination >= -32768
	Branch:	-32768 <= branch destination <= 32767
<CR>	= Carriage Return to complete the command	

Notation:

3 25,-1	Repeat the last command 25 times
3 0,-5	Jump 5 steps back
3 0,5	Skip the next 4 commands
3 6,-5	Repeat the last 5 commands 6 times

Explanation: If the control system finds the command "Loop/branch" during the execution of the CNC program, first the number of loops is checked to decide whether it is a loop or branch command. In the case of a loop command, a loop counter is set up, loaded with default values, and the command counter is corrected by the offset specified. The commands up to the next loop counter are now repeated, and the loop counter is decremented until it has reached zero. Then the program continues with the execution of the first command after the loop. Loops may be nested with a nesting depth of 15. The required counters are managed on an appropriate loop stack. In the case of a branch, the offset will be understood as a relative branch destination within the NC program and the command counter be corrected by the offset.

Restriction: Branching before the start or after the end of the data field is not permitted. Forward loops are permitted. A loop will always repeat the last n commands. At least one command must be repeated. Loops may be nested; the maximum nesting depth is 15. Leaving a loop via a branch is not permitted.

Programming example: see Appendix D

3.2.11 Time Delays in CNC Mode

Command: Time delay

Application: Storing time delays.

Structure: 5<time><CR>

5	= time delay command code
<time>	= time in 1/10 sec
<CR>	= Carriage Return to complete the command

Notation: 3 50 delay of 5 seconds

Explanation: If the control system finds the Time delay command during the execution of the CNC program, the next command in the CNC program will only be executed after the delay time has expired. The appropriate time will be specified in 1/10 seconds.

Restriction: A time delay cannot be cancelled by pressing the Stop key of the control system.

Programming example: see Appendix D

3.2.12 Setting the Port in CNC Mode

Command: Set output port

Application: Defined enabling/disabling of existing output ports.

Structure: p<port No.>,<bit No.>,<value><CR>

p	= command code "Set Port"
<port No.>	= port number
<bit No.>	= bit number, 0 - 7 --> bit by bit, 128 --> byte by byte
<Value>	= new value
<CR>	= Carriage Return to complete the command

Notation:

p2,128,1	Port 0, setting to "1" by bytes
p2,0,1	Port 0 , setting bit 0 to 1

Explanation: "p" specifies that the value of an output port is to be set. Then port number, bit number and new port value are transferred separated by commas, and the command is completed with Carriage Return. The control system will respond with the software handshake "0" if the saving operation has been successfully completed or with an error message if wrong port numbers and/or values have been transferred. For the IMC4 control system, the following ports are defined with appropriate functionalities:

Port	Bit	Value	Function
0	0 - 7	0 - 255	User I/O (also still possible with 65529)
1	0	0	Cover may not be opened
	0	1	Cover may be opened
2	0	0	Spindle OFF
	0	1	Spindle ON
3	0	0	Motor currents OFF
	0	1	Motor current ON

Restriction: The setting of the port outputs is carried out in the control system as defined in the program. Setting or deleting outputs while a command is being executed, e.g. during a positioning movement, is not possible.

Programming example: see Appendix D

3.2.13 Reading a Port and Branching in CNC Mode

Command: Read input port

Application: Reading an input port and branching within the program. Thanks to the branching, it is possible to branch to a certain block after a logic comparison.

Structure: o<port No.>,<bit No.>,<value>,<offset><CR>

o	= command code "Set Port"
<port No.>	= port number
<bit No.>	= bit number, 0 - 7 --> by bits, 128 --> by bytes
<value>	= comparison value
<Offset>	= branch destination -32768 <= branch destination <= 32767
<CR>	= Carriage Return to complete the command

Notation:

o2,128,1,-1	Wait until port 0 <> 1
o2,0,1,-1	Wait until Port0, bit0 = 0
o2,0,1,3	If Port0, Bit0 == 1, command counter += 3

Explanation: "o" specifies that the value of an input port is to be read and the program will be adapted according to the value. Then port number, bit number, comparison value and command offset are transferred separated by commas, and the command is completed with Carriage Return. The control system will respond with the software handshake "0" if the saving operation has been successfully completed or with an error message if wrong port numbers and/or values have been transferred. During the program execution, the appropriate port is polled and logically compared with the intended value either bit by bit or byte by byte. If the logic comparison is true, branching by the offset is carried out; otherwise, the next command is executed as intended by the program. For the IMC4 control system, the following ports are defined with the appropriate functionalities:

Port	Bit	Status	Function
0	0 - 7	00 - FF	User I/O (also still possible with 65531)
1	0	00	Cover is open
	0	01	Cover is closed
2	0	00	Spindle is turned off
	0	01	Spindle is turned on
3	0	00	Motor currents are turned off
	0	01	Motor currents are turned on

Restriction: The port inputs are polled in the control system as defined in the program. Polling inputs while a command is being executed, e.g. during a positioning movement, is thus not possible.

Programming example: -

3.2.14 End of Data Field in CNC Mode

Command: End of data field

Application: This command is used to mark the end of a CNC data field and serves to complete the data transfer and to save storable commands.

Structure: 9<CR>

9	= command code "End of Data Field"
<CR>	= Carriage Return to complete the command

Notation: 9

Explanation: "9" specifies that the end of the CNC data field transferred is reached. The command is completed with Carriage Return. The control system will response with the software handshake "0" if the saving operation has been successfully carried out or with an error message. In addition to the marking of the data field as a valid CNC program, status information (e.g. the current referencing velocity) are stored in the FlashProm. The control system will then be in DNC mode again and will accept the appropriate commands.

Restriction: A CNC data field must be completed with the end-of-data field command; otherwise, the saved CNC program will not be valid and cannot be executed.

Programming example: see Appendix D

4 Error Messages of the IMC4

After each transferred command, the control system will response with an appropriate feedback. These codes are transferred as ASCII characters and can thus be evaluated easily. Based on the character transferred, error sources and causes can be determined.
The individual error codes are described in the following.

Code	Description
0	Handshake character - No error; the command has been executed correctly. - The next command can be transferred.
1	Error in transmitted number - The control system has received a numerical specification that could not be interpreted correctly. - The numerical value transmitted is outside the admissible range or contains illegal characters.
2	Limit switch error - As a result of the traversing movement, a limit switch has responded. The current movement has been cancelled. This is done by stopping the movement without braking ramp. As a result, the actual positions of the control system are no longer correct; step losses may have occurred. - The reference point approach of a stepper motor axis has been carried out not correctly or has not yet been carried out at all. **CAUTION:** After a limit switch error, the control system has to be re-initialised and a reference point approach to be carried out.
3	Illegal axis specification - The control system has been transmitted an axis specification for a command to be executed, which contains an axis not defined. - In commands that contain axis specifications you should only use combinations of axes that are initialised.
4	No axes defined - Before movements or, generally, commands are transferred to the control system which have a number of parameters which depends on the number of axes, the Set Axes command must be transferred in order to be able to set the internal axis parameters correctly.
5	Syntax error - A command has been transmitted with errors. - The command used does not exist or cannot be executed by this control system. - Check whether all transferred commands are correct.
6	End of memory - You have tried to transfer more commands in CNC mode, than can be stored in the control system.
7	Illegal number of parameters - The control system has received more or less parameters for the command than needed. - Check whether the number of parameters required for the command is correct in conjunction with the number of axes.

8	Command to be saved is correct - The control system has been transferred a command that is not available as a CNC command.
9	Plant error - The power supply of the plant is not yet turned on. - The safety relay of the plant is not active. - The output stages and/or the safety circuit could not be turned on, since the cover is still open. - An Emergency Stop situation has occurred. **CAUTION:** After an Emergency Stop situation, the control system must be re-initialised and a reference point approach be carried out.
A	not used by this control system
B	not used by this control system
C	not used by this control system
D	Illegal velocity - The admissible limits for velocity specifications have not been observed. - Check whether all velocity specifications are correct.
E	not used by this control system
F	User stop - The user has actuated the Stop key on the control system and the movement currently active has been stopped. The command execution can be resumed either by pressing the Start key or using the Start command @0s.
G	Invalid data field - The control system has been transferred a Start command although no remaining movement to be traversed is left in the memory, i.e. although no stop function has been carried out beforehand. - You have tried to transfer a CNC program although a program or parts of a program are still contained in the memory.
H	Cover error - You have tried to execute a command that is not permissible with open cover.
=	not used by this control system

A1 Software Routines for Calculating the Parameters with the Circle Command in Turbo Pascal

Sample program for calculating the circle parameters in Turbo Pascal:

```
{ ============================================================ }
{ ================= Kreisberechnung für IMC4 ================= }
{ ============================================================ }

program test_kreis;

uses Crt;

const ccw=16; cw=0;
      yle=8;  ygt=0;
      yeq=4;
      xle=2;  xgt=0;
      xeq=1;

var sxarr     : array [ 0 .. 32 ] of real;
    syarr     : array [ 0 .. 32 ] of real;
    sx, sy    : real;

    msteps, richtung, speed,
    mmRadius, sRadius,
    mmStartX, mmStartY,
    sStartX, sStartY,
    wStart, wEnd,
    no, dq,
    S_Winkel, E_Winkel,
    a_w, e_w  : real;
    str1      : string[80];
    str2      : string[80];


function Summe(xx:real):real;
begin
  if(xx>0) then
    Summe:=xx*(xx+1)
  else
    Summe:=-xx*(xx-1)
end; { Ende Summe }

function formel:real;
begin
  if(richtung=1) then       { kreis ccw }
    Formel:=( sx*sy*sRadius+sx*sy*Summe(sRadius-1.0)
    -sx*Summe(sStartX+(sx-sy)/2.0)+sy*Summe(sStartY+(sx+sy)/2.0))/2

  else
    Formel:=(-sx*sy*sRadius-sx*sy*Summe(sRadius-1.0)
    -sx*Summe(sStartX+(sx+sy)/2.0)+sy*Summe(sStartY+(sy-sx)/2))/2;

end; { Ende Formel }

procedure initsxsyarr;
var i:integer;
begin
  for i:=0 to 32 do begin sxarr[i]:=13; syarr[i]:=13; end;

  sxarr[ccw+xgt+ygt]:= -1;    syarr[ccw+xgt+ygt]:= +1;
  sxarr[ccw+xgt+yeq]:= -1;    syarr[ccw+xgt+yeq]:= +1; { CCW-Quadrant   I }

  sxarr[ccw+xle+ygt]:= -1;    syarr[ccw+xle+ygt]:= -1;
  sxarr[ccw+xeq+ygt]:= -1;    syarr[ccw+xeq+ygt]:= -1; { CCW-Quadrant  II }
```

```
  sxarr[ccw+xle+yle]:= +1;    syarr[ccw+xle+yle]:= -1;
  sxarr[ccw+xle+yeq]:= +1;    syarr[ccw+xle+yeq]:= -1; { CCW-Quadrant III }

  sxarr[ccw+xgt+yle]:= +1;    syarr[ccw+xgt+yle]:= +1;
  sxarr[ccw+xeq+yle]:= +1;    syarr[ccw+xeq+yle]:= +1; { CCW-Quadrant  IV }

  sxarr[cw+xgt+yle]:= -1;    syarr[cw+xgt+yle]:= -1;
  sxarr[cw+xgt+yeq]:= -1;    syarr[cw+xgt+yeq]:= -1;   { CW-Quadrant  IV }

  sxarr[cw+xle+yle]:= -1;    syarr[cw+xle+yle]:= +1;
  sxarr[cw+xeq+yle]:= -1;    syarr[cw+xeq+yle]:= +1;   { CW-Quadrant III }

  sxarr[cw+xle+ygt]:= +1;    syarr[cw+xle+ygt]:= +1;
  sxarr[cw+xle+yeq]:= +1;    syarr[cw+xle+yeq]:= +1;   { CW-Quadrant  II }

  sxarr[cw+xgt+ygt]:= +1;    syarr[cw+xgt+ygt]:= -1;
  sxarr[cw+xeq+ygt]:= +1;    syarr[cw+xeq+ygt]:= -1;   { CW-Quadrant   I }
end; { Ende initsxsyarr }

procedure calcsxsy;
var  i,xx,yy:integer;
begin
  i:=0;

  if richtung=1
    then i:=i+ccw
  else i:=i+cw;
  if (sStartX>=-0.5) and (sStartX<=0.5)  then i:=i+xeq;
  if sStartX<-0.5 then i:=i+xle;
  if (sStartY>=-0.5) and (sStartY<=0.5) then i:=i+yeq;
  if sStartY<-0.5 then i:=i+yle;

  sx:=sxarr[i];
  sy:=syarr[i];

end; { Ende calcsxsy }

{ Hauptprogramm }
begin
  WriteLn('Test der Kreisberechnung IMC4');
  WriteLn('============================');
  Write('Schritte pro mm     :'); ReadLn(MSteps);
  Write('Richtung(0=cw/1=ccw):'); ReadLn(Richtung);
  Write('Radius              :'); ReadLn(mmRadius);
  Write('Anfangswinkel       :'); ReadLn(wStart);
  Write('Endwinkel           :'); ReadLn(wEnd);
  Write('Geschwindigkeit:    :'); ReadLn(Speed);

  if(richtung>=1.0) then richtung:=1.0;
  if(richtung<=0.0) then richtung:=0.0;

  { Winkel in Bogenmass umrechnen }
  S_Winkel:=wStart*pi/180.0;
  E_Winkel:=wEnd*pi/180.0;

  { Startpunkt relativ zum Mittelpunkt in mm }
  mmStartX:=cos(S_Winkel)*mmRadius;
  mmStartY:=sin(S_Winkel)*mmRadius;

  { Startpunkt und Radius in Schritten }
  sStartX:=mmStartX*msteps;
  sStartY:=mmStartY*msteps;
  sRadius:=mmRadius*msteps;

  initsxsyarr;
  calcsxsy;

  { Bogenlaenge in Schritten }
  if(richtung=1) then     { circle ccw }
```

```
    begin
    a_w:=s_winkel;
    e_w:=e_winkel+0.00001 { Korrektur wegen Rundungsfehlern };
    while(a_w<0) do begin a_w:=a_w+2.0*pi; e_w:=e_w+2.0*pi; end;
    if(a_w>e_w) then
      begin WriteLn('Fehler Startwinkel > Endwinkel !!!'); halt; end;
    {Berechnung der Bogenlänge}
    while(a_w>=pi/2.0) do begin a_w:=a_w-pi/2;
                                e_w:=e_w-pi/2; end;
    no:=0.0;
    while(e_w-a_w>=pi/2.0) do begin e_w:=e_w-pi/2.0;
                                    no:=no+2.0*sRadius; end;
    if(e_w>pi/2.0) then begin no:=no+2.0*sRadius;
                              e_w:=e_w-pi/2.0; end;
    no:=no+sRadius*(cos(a_w)-cos(e_w)+sin(e_w)-sin(a_w));
    end
  else                    { circle cw }
    begin
    a_w:=s_winkel;
    e_w:=e_winkel-0.00001 { Korrektur wegen Rundungsfehlern };
    while(a_w>0) do begin a_w:=a_w-2.0*pi;
                          e_w:=e_w-2.0*pi; end;
    if(a_w<e_w) then
      begin WriteLn('Fehler Startwinkel < Endwinkel !!!'); halt; end;
    {berechnung der Bogenlänge}
    while(a_w<=-pi/2.0) do begin a_w:=a_w+pi/2;
                                 e_w:=e_w+pi/2; end;
    no:=0.0;
    while(a_w-e_w>=pi/2.0) do begin e_w:=e_w+pi/2.0;
                                    no:=no+2.0*sRadius; end;
    if(e_w<-pi/2.0) then begin e_w:=e_w+pi/2.0;
                               no:=no+2.0*sRadius; end;
    no:=no+sRadius*(cos(a_w)-cos(e_w)+sin(a_w)-sin(e_w));
  end;

  if(no<0) then no:=-no;
  if no<1. then begin
    WriteLn('Fehler Bogenlaenge < 1 Schritt !!!'); halt; end;

  dq:=Formel;               { Interpolationsparameter berechnen }

  str2:='@0y';
  str(no:0:0,str1);
  str2:=str2+str1+',';      { @0y2000, }
  str(speed:0:0,str1);
  str2:=str2+str1+',';      { @0y2000,1000, }
  str(dq:0:0,str1);
  str2:=str2+str1+',';      { @0y2000,1000,1000, }
  str(sStartX:0:0,str1);
  str2:=str2+str1+',';      { @0y2000,1000,1000,0, }
  str(sStartY:0:0,str1);
  str2:=str2+str1+',';      { @0y2000,1000,1000,0,2000, }
  str(sx:0:0,str1);
  str2:=str2+str1+',';      { @0y2000,1000,1000,0,2000,1, }
  str(sy:0:0,str1);
  str2:=str2+str1;          { @0y2000,1000,1000,0,2000,1,-1 }

  Writeln; WriteLn('Ausgabe:');
  if(richtung=1) then WriteLn('@0f-1')  { circle ccw }
  else WriteLn('@0f0');                 { circle cw }
  WriteLn(str2);                        { output string to display }

end. { Ende Hauptprogramm }
```

A2 Software Routines for Calculating the Parameters with the Circle Command in C

Sample program for calculating the circle parameters in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


#define CONST_PI    (double)3.141592653589793
#define CONST_2PI   (double)(CONST_PI * 2.0)
#define CONST_PId2  (double)(CONST_PI / 2.0)
#define CONST_GRAD  (double)(180.0 / CONST_PI)
#define CONST_DEG   (double)(CONST_PI / 180.0)


/*********************************************************************/
/************** Erzeugung der Kreisparameter für IMC4 in C ***************/
/*********************************************************************/

                            /***********************************
                             *                ToLong           *
                             ***********************************/
long ToLong(double a)
{ if (a<0.0) return((long)(a-0.5));
  else return((long)(a+0.5));
}

                            /***********************************
                             *            TravelCCW            *
                             ***********************************/
/* Normierung in den 1.Quadranten CCW */
double TravelCCW(double xs, double ys, double radii)
{ double no=0.; double t;
  while((xs<0.)|(ys<0.))
  { /* rotate 90 deg cw */
    t=ys; ys=-xs; xs=t;
    /* we did 2xR Steps */
    no+=radii*2.;
  }
  no+=radii-xs+ys;
  return(no);
}

                            /***********************************
                             *             TravelCW            *
                             ***********************************/
/* Normierung in den 1.Quadranten CW */
double TravelCW(double xs, double ys, double radii)
{ double no=0.; double t;
  while((xs<0.)|(ys>0.))
  { /* rotate 90 deg ccw */
    t=ys; ys=xs; xs=-t;
    /* we did 2xR Steps */
    no+=radii*2.;
  }
  no+=radii-xs-ys;
  return(no);
}
```

```
                            /***********************************
                             *          Bahnlaenge             *
                             ***********************************/
/* Bahnlänge des Kreises */
double Bahnlaenge(double startx,double starty,double endx,double endy,
                  int cidir,double radius)
{
  double nno;

  switch(cidir)
  {
    case 0:                 /* cw */
      nno=TravelCW(endx,endy,radius)-TravelCW(startx,starty,radius);
      if(nno<0.) nno+=8.*radius; return(nno);
      break;
    case 1:                 /* ccw*/
      nno=TravelCCW(endx,endy,radius)-TravelCCW(startx,starty,radius);
      if(nno<0.) nno+=8.*radius; return(nno);
      break;
    break;
  }
}


                            /***********************************
                             *              Summe              *
                             ***********************************/
double Summe(double i)          /* Teilberechnung zu Formel*/
{ if(i<=0.) return(-i*(i-1.));
  else return(i*(i+1.));
}


                            /***********************************
                             *              Formel             *
                             ***********************************/
/* Berechnung Interpolationsparameter DQ*/
double Formel(double sx,double sy, double radius,
              double x, double y, int ri)
{
  switch(ri)
  {
    case 1:     /* ccw */
      return( (sx*sy*radius+sx*sy*Summe(radius-1.)
             -sx*Summe(x+(sx-sy)/2.)+sy*Summe(y+(sx+sy)/2.))/2. );
    break;
    case 0:     /* cw */
      return( (-sx*sy*radius-sx*sy*Summe(radius-1.)
             -sx*Summe(x+(sx+sy)/2.)+sy*Summe(y+(sy-sx)/2.))/2. );
    break;
  }
}
```

```
                          /**********************************
                           *              Circle            *
                           **********************************/

/*************************************************************/
/* Parameter:                                                */
/*************************************************************/
/* ebene              - Ebene für Zirkularinterpolation      */
/* cidir              - Richtung des Kreisbogens (0=cw, 1=ccw) */
/* endx, endy         - Endpunkt des Kreisbogens             */
/* startx, startx     - Startpunkt des Kreises               */
/* MoveSpeed          - Geschwindigkeit für Ausgabe          */
/*************************************************************/
void Circle(int ebene, int cidir,
          double startx,double starty,
          double endx,double endy,
          double radius,
          int MoveSpeed)
{
   double sx,sy,fo,no;

   switch(cidir)             /* sx und sy in Abhängigkeit von der Kreis- */
    {                        /* richtung bestimmen */
      case 0:/* cw */
      if( startx>=0. & starty>0. ) { sx=1.;  sy=-1.; }
      if( startx<0. & starty>=0. ) { sx=1.;  sy=1.;  }
      if( startx<=0. & starty<0. ) { sx=-1.; sy=1.;  }
      if( startx>0. & starty<=0. ) { sx=-1.; sy=-1.; }
        break;
      case 1: /* ccw */
      if( startx>0. & starty>=0. ) { sx=-1.; sy=1.;  }
      if( startx<=0. & starty>0. ) { sx=-1.; sy=-1.; }
      if( startx<0. & starty<=0. ) { sx=1.;  sy=-1.; }
      if( startx>=0. & starty<0. ) { sx=1.;  sy=1.;  }
        break;
    }

                           /* Berechnung Interpolationsparameter */
   fo=(double)Formel(sx,sy,radius,startx,starty,cidir);

                           /* Berechnung der Bahnlaenge */
   no=(double)Bahnlaenge(startx,starty,endx,endy,cidir,radius);
   if(no<0.) no=-no;

                           /* Korrektur Richtung für Ausgabe */
   if(cidir==1) cidir=-1;

       fprintf(stderr,"@0e%d\n",(int)ebene);
       fprintf(stderr,"@0f%1d\n",cidir);
       fprintf(stderr,"@0y%ld,%d,%ld,%ld,%ld,%ld,%ld\n",
                  ToLong(no),
                  (int)MoveSpeed,
                  ToLong(fo),
                  ToLong(startx),
                  ToLong(starty),
                  ToLong(sx),
                  ToLong(sy));
}
```

```
/*****************************************************************************/
/* Hauptprogramm                                                             */
/*****************************************************************************/
void main()
{
  double msteps=80.;
  double Radius=0.;
  double Startwinkel=0.;
  double Endwinkel=0.;
  int Ebene=0;
  int Richtung=0;
  int Speed=1500;
  int CNC=0;

  double radius=0.;
  double startwinkel=0.;
  double endwinkel=0.;
  double startx, starty, endx, endy;

  printf("\nParameter für Kreisberechnung IMC4:");
  printf("\nSchritte pro mm      : "); scanf("%lf",&msteps);
  printf(  "Ebene(0=xy/1=xz/2=yz): "); scanf("%d",&Ebene);
  printf(  "Richtung(0=cw/1=ccw) : "); scanf("%d",&Richtung);
  printf(  "Radius               : "); scanf("%lf",&Radius);
  printf(  "Startwinkel          : "); scanf("%lf",&Startwinkel);
  printf(  "Endwinkel            : "); scanf("%lf",&Endwinkel);
  printf(  "Geschwindigkeit      : "); scanf("%d",&Speed);

  radius=Radius*msteps;

  startwinkel=(Startwinkel)*CONST_DEG;
  endwinkel=(Endwinkel)*CONST_DEG;

  startx = radius*cos(startwinkel);
  starty = radius*sin(startwinkel);

  endx = radius*cos(endwinkel);
  endy = radius*sin(endwinkel);

  Circle(Ebene,Richtung,startx,starty,endx,endy,radius,Speed,CNC);

}
/*****************************************************************************/
/* Hauptprogramm Ende                                                        */
/*****************************************************************************/
```

B1 Programming the IMC4 in Turbo Pascal

These sample programs will show how the IMC4 can be programmed for easy tasks using, e.g. the programming language Turbo Pascal. In addition, the sample programs will serve for better understanding of the serial selection of the IMC4 and its scope of functions and to initiate your own applications, also in other programming languages.

B2 Unit for Serial Data Transfer in Turbo Pascal

First, we would like to introduce you in brief a unit whose functions can be used in the sample programs below and which provides easy handling of the interface.

```
{*********************************************}
{* unit SERIO.PAS                            *}
{* Turbo Pascal unit für serielle Kommunikation *}
{*********************************************}

unit Serio;

interface                    {* Interface *}
uses Crt, Dos;               {* System Includes *}
  type                       {* Vordefinierte Aufzählungstypen *}
    ComType    = (COM1, COM2);
    BaudType   = (B110, B150, B300, B600, B1200, B2400, B4800,
                  B9600, B19200, B38400, B57600, B115200);
    ParityType = (None, Odd, Even);
    LengthType = (D7, D8);
    StopType   = (S1, S2);
                             {* Prozeduren und Funktionen *}
  procedure InitCom (ComNumber  : ComType;
                     BaudRate   : BaudType;
                     ParityBit  : ParityType;
                     DataLength : LengthType;
                     StopBits   : StopType );

  procedure ExitCom (ComNumber : ComType);
  function  ComDataReceived (ComNumber : ComType) : boolean;
  function  ReadComData (ComNumber : ComType) : char;
  procedure WriteComData (ComNumber : ComType; OutByte : char);
  procedure WriteComString (ComNumber : ComType; OutString : string);


implementation               {* Implementation *}

  type                       {* globale Typdefinition zu Speicherung *}
    IntBlock = record        {* einer Interruptvektoradresse *}
      IntOffset  : integer;  {* mit Offsetadresse, *}
      IntSegment : integer;  {* Segmentadresse *}
      IntNumber  : byte;     {* und Interruptnummer *}
    end;

    I8250 = record      {* globale Tydefintion für Registersatz i8250 *}
      DLL : integer;    {* divisor latch low register (if LCR bit7 = 1) *}
      DLH : integer;    {* divisor latch high register (if LCR bit7 = 1) *}
      THR : integer;    {* transmit holding register *}
      RBR : integer;    {* receive holding register *}
      IER : integer;    {* interrupt enable register *}
      LCR : integer;    {* line control register *}
      MCR : integer;    {* modem control register *}
      LSR : integer;    {* line status register *}
      MSR : integer;    {* modem status register *}
    end;


  const                      {* Konstantendefinitionen *}
    IntDS : integer = 0;
    ComPort : array [COM1..COM2] of I8250 =
      ((DLL : $3F8 ; DLH : $3F9 ; THR : $3F8 ; RBR : $3F8 ;
        IER : $3F9 ; LCR : $3FB ; MCR : $3FC ; LSR : $3FD ; MSR : $3FE),
```

```
      (DLL : $2F8 ; DLH : $2F9 ; THR : $2F8 ; RBR : $2F8 ;
       IER : $2F9 ; LCR : $2FB ; MCR : $2FC ; LSR : $2FD ; MSR : $2FE));
    ComBufferSize = $03ff;

  var                          { globale Variablen *}
    ComBuffer : array [COM1 .. COM2, 0..(ComBufferSize)] of byte;
    ComBufferWrite, ComBufferRead : array [COM1 .. COM2] of integer;
    ComBlock : array [COM1 .. COM2] of IntBlock;

{*************************************************************************}
{* InstallComInt                                                         *}
{* Interrupthandler installieren, alte Vektoradresse speichern und neue  *}
{* Adresse in Vektortabelle eintragen                                    *}
{*************************************************************************}
  procedure InstallComInt (IntNumber : byte; IntHandler : integer;
                           var Block : IntBlock);
  var
    Regs : Registers;
  begin
    IntDS := DSeg;                {* Datensegment merken *}
    Block.IntNumber := IntNumber; {* Interruptnummer merken *}
    Regs.AH := $35;               {* Int21 Fktn35 Interruptvektor lesen *}
    Regs.AL := IntNumber;         {* Interruptnummer *}
    MSDos (Dos.Registers(Regs));  {* Int21 aufrufen *}
    Block.IntSegment := Regs.ES;  {* ES enthält Segmentadresse *}
    Block.IntOffset  := Regs.BX;  {* BX enthält Offsetadresse *}
    Regs.AH := $25;               {* Int21 Fktn25 Interruptvektor setzen *}
    Regs.AL := IntNumber;         {* Interruptnummer *}
    Regs.DS := CSeg;              {* Segmentadresse neuer Handler *}
    Regs.DX := IntHandler;        {* Offsetadresse neuer Handler *}
    MSDos (Dos.Registers(Regs));  {* Int21 aufrufen *}
  end;

{*************************************************************************}
{* UnInstallComInt                                                       *}
{* Interrupthandler deinstallieren, alte Vektoradresse eintragen         *}
{*************************************************************************}
  procedure UnInstallComInt (var Block : IntBlock);
  var
    Regs : Registers;
  begin
    Regs.AH := $25;               {* Int21 Fktn25 Interruptvektor setzen *}
    Regs.AL := Block.IntNumber;   {* Interruptnummer *}
    Regs.DS := Block.IntSegment;  {* Segmentadresse alter Handler *}
    Regs.DX := Block.IntOffset;   {* Offsetadresse alter Handler *}
    MSDos (Dos.Registers(Regs));  {* Int21 aufrufen *}
  end;

{*************************************************************************}
{* Com1IntHandler                                                        *}
{* Interrupthandler für Com1, Intnr. 12                                  *}
{*************************************************************************}
  procedure Com1IntHandler (Flags, CS, IP, AX, BX, CX, DX,
                            SI, DI, DS, ES, BP : word);
  interrupt;
  begin                          {* Zeichen von Schnittstelle lesen *}
    ComBuffer[COM1, ComBufferWrite[COM1]] := Port[ComPort[COM1].RBR];
                                 {* Schnittstellenpuffer handeln *}
    ComBufferWrite[COM1] := (ComBufferWrite[COM1] + 1) and ComBufferSize;
    inline ($FA);                {* Maschinencode CLI,Interruptflag löschen *}
    Port[$20] := $20;            {* Interruptende Interruptkontroller 8259 *}
  end;

{*************************************************************************}


{* Com2IntHandler                                                        *}
{* Interrupthandler für Com2, Intnr. 11                                  *}
{*************************************************************************}
  procedure Com2IntHandler (Flags, CS, IP, AX, BX, CX, DX,
                            SI, DI, DS, ES, BP : word);
  interrupt;
  begin                          {* Zeichen von Schnittstelle lesen *}
    ComBuffer[COM2, ComBufferWrite[COM2]] := Port[ComPort[COM2].RBR];
                                 {* Schnittstellenpuffer handeln *}
```

```
    ComBufferWrite[COM2] := (ComBufferWrite[COM2] + 1) and ComBufferSize;
    inline ($FA);             { Maschinencode CLI, Interruptflag löschen }
    Port[$20] := $20;         { Interruptende Interruptkontroller 8259 }
  end;

{************************************************************************}
{* InitCom                                                              *}
{* Initialisierung der seriellen Schnittstelle                          *}
{************************************************************************}
  procedure InitCom; { (ComNumber  : ComType;
                        BaudRate   : BaudType; ParityBit  : ParityType;
                        DataLength : LengthType; StopBits   : StopType; }
  const
    BaudReg : array [B110 .. B115200] of word =
      ($0417, $0300, $0180, $00C0, $0060, $0030,
       $0018, $000C, $0006, $0003, $0002, $0001);
    ParityReg : array [None..Even] of byte =
      ($00, $08, $18);
    LengthReg : array [D7 .. D8] of byte =
      ($02, $03);
    StopReg : array [S1 .. S2] of byte =
      ($00, $04);
  var
    Regs : Registers;
  begin
    if ComNumber = COM1 then
    begin
      InstallComInt($0C, Ofs(Com1IntHandler), ComBlock[COM1]);
      Port[$21] := Port[$21] and $EF
    end
    else if ComNumber = COM2 then
    begin
      InstallComInt($0B, Ofs(Com2IntHandler), ComBlock[COM2]);
      Port[$21] := Port[$21] and $F7
    end;

    Port[ComPort[ComNumber].LCR] := $80; { switch to write latch reg }
    Port[ComPort[ComNumber].DLH] := Hi (BaudReg [BaudRate]);
    Port[ComPort[ComNumber].DLL] := Lo (BaudReg [BaudRate]);
    Port[ComPort[ComNumber].LCR] := $00 or
                                    ParityReg [ParityBit] or
                                    LengthReg [DataLength] or
                                    StopReg [StopBits];
    Port[ComPort[ComNumber].IER] := $01; { enable interrupts }
    Port[ComPort[ComNumber].MCR] := $01 or { raise DTR }
                                    $02 or { raise RTS }
                                    $08;   { raise OUT2 }
    ComBufferWrite[ComNumber] := 0;
    ComBufferRead[ComNumber]  := 0;
 end;
```

```
{*************************************************************************}
{* ExitCom                                                               *}
{* Rücksetzen der seriellen Schnittstelle                                *}
{*************************************************************************}
  procedure ExitCom; { (ComNumber : ComType) }
  var
    Regs : Registers;
  begin
    if ComNumber = COM1 then
      Port[$21] := Port[$21] or $10
    else if ComNumber = COM2 then
      Port[$21] := Port[$21] or $08;
    Port[ComPort[ComNumber].LCR] := Port[ComPort[ComNumber].LCR] and $7F;
    Port[ComPort[ComNumber].IER] := 0; { disable interrupts }
    Port[ComPort[ComNumber].MCR] := 0; { lower DTR, RTS and OUT2 }
    UnInstallComInt(ComBlock[ComNumber]);
  end;

{*************************************************************************}
{* ComDataReceived                                                       *}
{* Abfrage Zustand Empfangspuffer                                        *}
{*************************************************************************}
  function ComDataReceived; { (ComNumber : ComType) : boolean; }
  begin
    ComDataReceived := ComBufferWrite[ComNumber]<>ComBufferRead[ComNumber];
  end;

{*************************************************************************}
{* ReadComData                                                           *}
{* Zeichen aus dem Empfangspuffer lesen                                  *}
{*************************************************************************}
  function ReadComData; { (ComNumber : ComType) : char; }
  begin
    while ComBufferWrite[ComNumber] = ComBufferRead[ComNumber] do
      Delay(10);
    ReadComData := char(ComBuffer[ComNumber, ComBufferRead[ComNumber]]);
    ComBufferRead[ComNumber] := (ComBufferRead[ComNumber] + 1) and
                                ComBufferSize;
  end;

{*************************************************************************}
{* WriteComData                                                          *}
{* Zeichen senden                                                        *}
{*************************************************************************}
  procedure WriteComData; { (ComNumber : ComType; OutByte : char); }
  begin
    while ((Port[ComPort[ComNumber].LSR] and $20) <> $20) do Delay(1);
    Port[ComPort[ComNumber].THR] := byte(OutByte);
  end;

{*************************************************************************}
{* WriteComString                                                        *}
{* Zeichenkette senden                                                   *}
{*************************************************************************}
  procedure WriteComString; { (ComNumber : ComType; OutString : string); }
  var i : byte;
  begin
    for i:=1 to length(OutString) do WriteComData(ComNumber,OutString[i]);
  end;

end.
{*************************************************************************}
{* Ende der Unit Serio                                                   *}
{*************************************************************************}
```

B3 Sample Programs in Turbo Pascal

B3.1 Reference Point Approach; Initialising the Axes

```
{*************************************************************************}
{* IMC4 - Test Referenzfahrt, Achseninitialisierung                      *}
{*************************************************************************}

program imc4_r;
  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}
  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}

{*************************************************************************}
{* Rückmeldung von der Steuerung lesen                                   *}
{*************************************************************************}

  procedure Rueckmeldung (ComNumber : ComType);

  var
    antwort : char;

  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}

{*************************************************************************}
{* Hauptprogramm                                                         *}
{*************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* Schnittstelle initialisieren *}
                             {* Com1, 19200 Baud, keine Parität *}
                             {* 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzfahrt in X-Achse ausführen *}
    WriteComString(SerioPort,'@0R1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzfahrt in X-Y-Achse ausführen *}
    WriteComString(SerioPort,'@0R3'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzfahrt in X-Y-Z-Achse ausführen *}
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
                             {* 4te Achse initialisieren *}
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzfahrt in A-Achse ausführen *}
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}
 end.                        {* und Ende *}

{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.2 Setting the Referencing Velocity

```
{************************************************************************}
{* IMC4 - Test Referenzgeschwindigkeit einstellen                       *}
{************************************************************************}

program imc4_d;

  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}
  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}

{************************************************************************}
{* Rückmeldung von der Steuerung lesen                                  *}
{************************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}

{************************************************************************}
{* Hauptprogramm                                                        *}
{************************************************************************}

 begin
    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* Schnittstelle initialisieren *}
                             {* Com1, 19200 Baud, keine Parität *}
                             {* 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);
                             {* 1 Achse initialisieren *}
    WriteComString(SerioPort,'@01'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzgeschw. setzen X-Achse *}
    WriteComString(SerioPort,'@0d3000'+chr($0D));
    Rueckmeldung(SerioPort);
                             {* 2 Achsen initialisieren *}
    WriteComString(SerioPort,'@03'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzgeschw. setzen X-Y-Achse *}
    WriteComString(SerioPort,'@0d3000,3000'+chr($0D));
    Rueckmeldung(SerioPort);
                             {* 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzgeschw. setzen X-Y-Z-Achse *}
    WriteComString(SerioPort,'@0d3000,3000,3000'+chr($0D));
    Rueckmeldung(SerioPort);
                             {* 4 Achse initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Referenzgeschw. setzen X-Y-Z-A-Achse*}
    WriteComString(SerioPort,'@0d3000,3000,3000,3000'+chr($0D));
    Rueckmeldung(SerioPort);
    ExitCom(SerioPort);      {* Schnittstelle schliessen *}
 end.                        {* und Ende *}

{************************************************************************}
{* Ende Hauptprogramm                                                   *}
{************************************************************************}
```

B3.3 Relative Movement

```
{*********************************************************************}
{* IMC4 - Test relative Bewegung                                     *}
{*********************************************************************}

program imc4_a;

  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}



{*********************************************************************}
{* Rückmeldung von der Steuerung lesen                               *}
{*********************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}



{*********************************************************************}
{* Hauptprogramm                                                     *}
{*********************************************************************}
 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, COm1, 19200 Baud*}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* nur X-Achse, 1 Achse initialisieren *}
    WriteComString(SerioPort,'@01'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* 5000 Schritte, 1000 Schritte/s *}
    WriteComString(SerioPort,'@0A5000,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y-Achse, 2 Achsen initialisieren *}
    WriteComString(SerioPort,'@03'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R3'+chr($0D)); Rueckmeldung(SerioPort);
                             {* X 5000, Y3000 Schritte, 1000 Schritte/s *}
    WriteComString(SerioPort,'@0A5000,1000,3000,10000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
                             {* X5000, Y3000, Z1000, 1000 Schritte/s *}
    WriteComString(SerioPort,
                  '@0A5000,1000,3000,1000,1000,1000,-1000,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y+Z+A-Achse, 4 Achse initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);
                             {* X5000, Y3000, Z1000, A500, 1000 S/s *}
    WriteComString(SerioPort,
                  '@0A5000,1000,3000,1000,1000,1000,500,1000'+chr($0D));
    Rueckmeldung(SerioPort);
```

```
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}
{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.4 Absolute Movement

```
{*********************************************************************}
{* IMC4 - Test Bewegung zur Position                                 *}
{*********************************************************************}

program imc4_a;

  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}



{*********************************************************************}
{* Rückmeldung von der Steuerung lesen                               *}
{*********************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}



{*********************************************************************}
{* Hauptprogramm                                                     *}
{*********************************************************************}
 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, COm1, 19200 Baud*}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* nur X-Achse, 1 Achse initialisieren *}
    WriteComString(SerioPort,'@01'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Pos(X5000), 1000 Schritte/s *}
    WriteComString(SerioPort,'@0M5000,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y-Achse, 2 Achsen initialisieren *}
    WriteComString(SerioPort,'@03'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R3'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Pos(X5000,Y3000), 1000 Schritte/s *}
    WriteComString(SerioPort,'@0M5000,1000,3000,10000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Pos(X5000,Y3000,Z1000), 1000 S/s *}
    WriteComString(SerioPort,
                  '@0M5000,1000,3000,1000,1000,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y+Z+A-Achse, 4 Achse initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Pos(X5000,Y3000,Z1000,A500), 1000 S/s *}
    WriteComString(SerioPort,
                  '@0M5000,1000,3000,1000,1000,1000,500,1000'+chr($0D));
    Rueckmeldung(SerioPort);
```

```
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}

{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.5 Zero Offset

```
{************************************************************************}
{* IMC4 - Test Nullpunktverschiebung                                    *}
{************************************************************************}

program imc4_a;

  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}



{************************************************************************}
{* Rückmeldung von der Steuerung lesen                                  *}
{************************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}



{************************************************************************}
{* Hauptprogramm                                                        *}
{************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* nur X-Achse, 1 Achse initialisieren *}
    WriteComString(SerioPort,'@01'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}
    WriteComString(SerioPort,'@0A5000,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* Nullpunkt setzen *}
    WriteComString(SerioPort,'@0n1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}
    WriteComString(SerioPort,'@0M-5000,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y-Achse, 2 Achsen initialisieren *}
    WriteComString(SerioPort,'@03'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R3'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}
    WriteComString(SerioPort,'@0M5000,1000,3000,10000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* Nullpunkt setzen *}
    WriteComString(SerioPort,'@0n3'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung nach (0,0) *}
    WriteComString(SerioPort,'@0M0,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}
```

```
    WriteComString(SerioPort,
                   '@0M5000,1000,3000,1000,1000,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                              {* Nullpunkt setzen *}
    WriteComString(SerioPort,'@0n7'+chr($0D)); Rueckmeldung(SerioPort);
                              {* Bewegung nach (0,0,0) *}
    WriteComString(SerioPort,'@0M0,1000,0,1000,0,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                              {* X+Y+Z+A-Achse, 4 Achse initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);
                              {* Bewegung *}
    WriteComString(SerioPort,
                   '@0M5000,1000,3000,1000,1000,1000,500,1000'+chr($0D));
    Rueckmeldung(SerioPort);

                              {* Nullpunkt setzen *}
    WriteComString(SerioPort,'@0n7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0n8'+chr($0D)); Rueckmeldung(SerioPort);
                              {* Bewegung *}
    WriteComString(SerioPort,
             '@0A-5000,1000,-3000,1000,-1000,1000,-500,1000'+chr($0D));
    Rueckmeldung(SerioPort);

    ExitCom(SerioPort);       {* Schnittstelle schliessen *}

 end.                         {* und Ende *}

{************************************************************************}
{* Ende Hauptprogramm                                                   *}
{************************************************************************}
```

B3.6 Position Interrogation

```
{************************************************************************}
{* IMC4 - Test Positionsabfrage 3+4 Achsen                              *}
{************************************************************************}


program imc4_p;

  uses Crt, Serio;           {* Serio ist eigene Unit für RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}


{************************************************************************}
{* Rückmeldung von der Steuerung lesen                                  *}
{************************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(ComNumber); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(SerioPort); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}


{************************************************************************}
{* Position 6byte hex char --> dezimal string                           *}
{************************************************************************}
  procedure Umwandeln( VAR PStr : string);
  var
    Pos : LongInt;
    i : Integer;
  begin
    if PStr[1]>'7' then Val('$FF'+PStr,Pos,i) else Val('$00'+PStr,Pos,i);
    Str(Pos,PStr);
  end;


{************************************************************************}
{* Position lesen 3 Achsen                                              *}
{************************************************************************}
  procedure PositionLesen3 (ComNumber : ComType);
  var
    i : Integer;
    PosStr : string;
  begin
    Writeln;
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position X = '+ PosStr);
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position Y = '+ PosStr);
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position Z = '+ PosStr);
  end;


{************************************************************************}
{* Position lesen 4 Achsen                                              *}
{************************************************************************}
  procedure PositionLesen4 (ComNumber : ComType);
  var
    i : Integer;
    PosStr : string;
```

```
  begin
    Writeln;
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position X = '+ PosStr);
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position Y = '+ PosStr);
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position Z = '+ PosStr);
    PosStr[0]:=#6; for i:=1 to 6 do PosStr[i]:=ReadComData(ComNumber);
    Umwandeln(PosStr); Writeln('Position A = '+ PosStr);
  end;


{*************************************************************************}
{* Hauptprogramm                                                         *}
{*************************************************************************}
 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);

                             {* Bewegung *}
    WriteComString(SerioPort,'@0M123,500,456,500,-789,500,0,30'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* Position abfragen *}
    WriteComString(SerioPort,'@0P'+chr($0D)); Rueckmeldung(SerioPort);
    PositionLesen3(SerioPort);

                             {* X+Y+Z+A-Achse, 4 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@08'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R8'+chr($0D)); Rueckmeldung(SerioPort);

                             {* Bewegung *}
    WriteComString(SerioPort,'@0M12,500,34,500,-56,500,78,500'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* Position abfragen *}
    WriteComString(SerioPort,'@0P'+chr($0D)); Rueckmeldung(SerioPort);
    PositionLesen4(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}
{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.7 Plane Selection

```
{*************************************************************************}
{* IMC4 - Test Ebenenwahl                                                *}
{*************************************************************************}

program imc4_e;

  uses Crt, Serio;           {* Serio ist eigene Unit füür RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}

{*************************************************************************}
{* Rückmeldung von der Steuerung lesen                                   *}
{*************************************************************************}

  procedure Rueckmeldung (ComNumber : ComType);

  var
    antwort : char;

  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}

{*************************************************************************}
{* Hauptprogramm                                                         *}
{*************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);

                             {* YZ-Ebene für Kreisinterpolation *}
    WriteComString(SerioPort,'@0e2'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}

{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.8 3D Interpolation

```
{*************************************************************************}
{* IMC4 - Test 3d-Interpolation Ein/Aus                                   *}
{*************************************************************************}

program imc4_z;

  uses Crt, Serio;           {* Serio ist eigene Unit füür RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}


{*************************************************************************}
{* Rückmeldung von der Steuerung lesen                                   *}
{*************************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}


{*************************************************************************}
{* Hauptprogramm                                                         *}
{*************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* X+Y+Z-Achse, 3 Achsen initialisieren *}
    WriteComString(SerioPort,'@07'+chr($0D)); Rueckmeldung(SerioPort);
    WriteComString(SerioPort,'@0R7'+chr($0D)); Rueckmeldung(SerioPort);
                             {* 3d-Interpolation Ein *}
    WriteComString(SerioPort,'@0z1'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung *}
    WriteComString(SerioPort,
                  '@0M5000,1000,3000,1000,1000,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);
                             {* 3d-Interpolation Aus *}
    WriteComString(SerioPort,'@0z0'+chr($0D)); Rueckmeldung(SerioPort);
                             {* Bewegung nach (0,0,0) *}
    WriteComString(SerioPort,'@0M0,1000,0,1000,0,1000,0,1000'+chr($0D));
    Rueckmeldung(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}

{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.9 Read Port

```
{*************************************************************************}
{* IMC4 - Test Port lesen                                                *}
{*************************************************************************}

program imc4_b;
  uses Crt, Serio;           {* Serio ist eigene Unit füür RS232 *}
  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}
    Pstr : string;
    i : Integer;
    Pwert : byte;

{*************************************************************************}
{* Rückmeldung von der Steuerung lesen                                   *}
{*************************************************************************}
  procedure Rueckmeldung (ComNumber : ComType);
  var
    antwort : char;
  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}

{*************************************************************************}
{* Hauptprogramm                                                         *}
{*************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* Port 0 abfragen *}
    WriteComString(SerioPort,'@0b0'+chr($0D));
    Rueckmeldung(SerioPort);

                             {* 2 Zeichen von Schnittstelle lesen *}
    Pstr[0] := #2;
    for i:=1 to 2 do Pstr[i]:=ReadComData(SerioPort);

    Writeln;                 {* Portwert ausgeben hex *}
    Writeln('Portwert: ' + Pstr + ' hex');
    Val(Pstr,Pwert,i);       {* Umwandeln string hex --> byte *}
    Str(Pwert,Pstr);         {* Umwandeln byte --> string dezimal *}
                             {* Portwert ausgeben dezimal *}
    Writeln('Portwert: ' + Pstr + ' dezimal');

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}

{*************************************************************************}
{* Ende Hauptprogramm                                                    *}
{*************************************************************************}
```

B3.10 Write Port

```
{************************************************************************}
{* IMC4 - Test Port schreiben                                           *}
{************************************************************************}

program imc4_B;

  uses Crt, Serio;           {* Serio ist eigene Unit füür RS232 *}

  var
    SerioPort : ComType;     {* Variable für RS232-Schnittstelle *}

{************************************************************************}
{* Rückmeldung von der Steuerung lesen                                  *}
{************************************************************************}

  procedure Rueckmeldung (ComNumber : ComType);

  var
    antwort : char;

  begin
    antwort := ReadComData(SerioPort); {* Zeichen von RS232 lesen *}
    if antwort<>'0' then begin         {* wenn Fehler, dann *}
      Writeln('Fehler : '+antwort);    {* Fehlermeldung und *}
      ExitCom(ComNumber); Halt;        {* Programmabbruch   *}
    end;                               {* sonst kein Fehler *}
  end;                                 {* und zurück *}

{************************************************************************}
{* Hauptprogramm                                                        *}
{************************************************************************}

 begin

    SerioPort := COM1;       {* Com1 verwenden, für Com2 COM2 einsetzen *}
                             {* RS232 initialisieren, Com1, 19200 Baud, *}
                             {* keine Parität, 8 Datenbit, 1 Stopbit *}
    InitCom(SerioPort, B19200, None, D8, S1);

                             {* Ausgabeport 0 schreiben *}
                             {* Wert 11000101 bin, C5 hex, 197 dez *}
    WriteComString(SerioPort,'@0B0,197'+chr($0D));
    Rueckmeldung(SerioPort);

    ExitCom(SerioPort);      {* Schnittstelle schliessen *}

 end.                        {* und Ende *}

{************************************************************************}
{* Ende Hauptprogramm                                                   *}
{************************************************************************}
```

C1 Sample Program for Programming the IMC4 in C

The sample programs below will show how the IMC4 can be programmed for easy tasks, e.g. using the programming language Microsoft C. In addition, this will serve for better understanding of the IMC4 and its scope of functions and to initiate your own applications, also in other programming languages.

```
/*********************************************************************/
/***                          IMC4TEST.C                           ***/
/*********************************************************************/
/*** Zweck : Test manuelles Verfahren mit Steuerung IMC4 ueber      ***/
/***         Funktionstasten mit Positionsabfrage und variabler     ***/
/***         Geschwindigkeit                                        ***/
/*********************************************************************/

/*********************************************************************/
/* globale Definitionen und Vereinbarungen                           */
/*********************************************************************/

/***********************************/
/**INCLUDES*************************/
/***********************************/
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include <dos.h>

/***********************************/
/**DEFINES**************************/
/***********************************/

#ifndef MK_FP                /* make far pointer funktion wie in Turbo C */
#define MK_FP(seg,ofs) \
              ((void far *)(((unsigned long)(seg)<<16)|(unsigned)(ofs)))
#endif

#define FALSE (0==1)         /* define bools wie in Pascal */
#define TRUE (1==1)
#define BOOL int

                             /* Konstanten der Achsen definieren */
#define StepsX 400
#define StepsY 400
#define StepsZ 400
#define SteigungX 5.
#define SteigungY 5.
#define SteigungZ 5.

                             /* Interruptvektoren com1, com2 */
#define Com1Vect 0x0c
#define BASE1 0x3f8
#define Com2Vect 0x0b
#define BASE2 0x2f8
                             /* Register Schnittstellencontroller 8250 */
#define XF8 BASE
#define XF9 BASE+1
#define XFA BASE+2
#define XFB BASE+3
#define XFC BASE+4
#define XFD BASE+5
#define XFE BASE+6
```

```
                                /* Funktionen Schnittstellencontroller 8250 */
#define SelectDivisorRegs outp(XFB,inp(XFB)|0x80)
#define SelectDataRegs outp(XFB,inp(XFB)&0x7F)
#define Set38400Bd outp(XF8,0x03);outp(XF9,0x00)
#define Set19200Bd outp(XF8,0x06);outp(XF9,0x00)
#define Set9600Bd  outp(XF8,0x0C);outp(XF9,0x00)
#define Set4800Bd  outp(XF8,0x18);outp(XF9,0x00)
#define Set2400Bd  outp(XF8,0x30);outp(XF9,0x00)
#define Set1200Bd  outp(XF8,0x60);outp(XF9,0x00)
#define Set600Bd   outp(XF8,0xC0);outp(XF9,0x00)
#define Set300Bd   outp(XF8,0x80);outp(XF9,0x01)
#define ComDataAvailable ((inp(XFD)&1)==1)
#define ComTransmitPossible ((inp(XFD)&0x20)==0x20)
#define Sende(c) { while(!ComTransmitPossible);  \
                   outp(XF8,(char)((int)(c)&0x00ff)); }
#define ComData inp(XF8)
#define LineState inp(XFD)

                                /* Groesse Empfangspuffer definieren */
#define BufferSize 0x03ff

                                /* Beep */
#define beep putch(0x07)

/************************************/
/******FUNKTION PROTOTYPES***********/
/************************************/
int InitializeRs232(long,int);
char Empfange(void);
void EnableComInt(int);
void DisableComInt(int);
void CloseRs232();
void _interrupt _far RSInterruptProc(void);
void (_interrupt _far * OldRSInt)();
void Rueckmeld(void);

void StringOut(char *);
void GetActPos(long *,long *,long *);
void ClrTastBuffer(void);
long HexToLong(unsigned char *);
void ReceivePosition(long *, long *, long *);

/************************************/
/**BOOLS*****************************/
/************************************/
BOOL BufferOverflow=FALSE;
BOOL RSError=FALSE;

/************************************/
/**INTEGERS**************************/
/************************************/
int ComPort;
int BASE;
int ComVect;
int W_Ptr=0;
int R_Ptr=0;
int Tastbuf=0;
int TeachSpeed=500;
unsigned int key_entry;
int scancode=0;
```

```
/************************************/
/***LONG INTEGERS*******************/
/************************************/
long xpos,ypos,zpos;
long Baudrate = 19200;


/************************************/
/**FLOATS***************************/
/************************************/
double AchsfaktorX,AchsfaktorY,AchsfaktorZ;


/************************************/
/**CHARACTERS***********************/
/************************************/
char RSBuffer[BufferSize];
char outstr[100];
char c;

                              /* Tabellen zur Umwandlung Hexzahlen */
unsigned char low_atab[128] =
           {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0,0,0,0,0,0,
            0,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };

unsigned char high_atab[128] =
           {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,0x90,0,0,0,0,0,0,
            0,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };

/*********************************************************************/
/**ENDE VEREINBARUNGSTEIL*********************************************/
/*********************************************************************/


/*********************************************************************/
/* ClrTastBuffer :  Tastaturpuffer loeschen                          */
/*********************************************************************/
void ClrTastBuffer(void)
{
  _disable();                     /* wenn Schreib- und Lesepointer auf  */
  *(int far *) MK_FP(0x40,0x1a)=  /* auf den Tastaturpuffer gleich sind */
  *(int far *) MK_FP(0x40,0x1c);  /* gilt dieser als leer               */
  _enable();
}
```

```
/*********************************************************************/
/* HexToLong :  Umwandeln einer Positionsrueckmeldung                */
/*********************************************************************/
long HexToLong(unsigned char *src)
{
  unsigned char B[4];

  B[2] =(unsigned char) high_atab[*(src++)];
  B[2]|=(unsigned char)  low_atab[*(src++)];
  B[1] =(unsigned char) high_atab[*(src++)];
  B[1]|=(unsigned char)  low_atab[*(src++)];
  B[0] =(unsigned char) high_atab[*(src++)];
  B[0]|=(unsigned char)  low_atab[*(src)];
  if(B[2]&0x80) B[3] =(unsigned char) 0xff;
  else B[3] =(unsigned char) 0x00;
  return(*(long *)(&B[0]));
}

/*********************************************************************/
/* ReceivePosition :  Empfangen einer Positionsrueckmeldung          */
/*********************************************************************/
void ReceivePosition(long *x, long *y, long *z)
{
   int i;
   char buffer[10];

   for(i=0;i<6;i++) buffer[i]=Empfange(); buffer[6]='\0';
   *x=(long)HexToLong((char *)buffer);
   for(i=0;i<6;i++) buffer[i]=Empfange(); buffer[6]='\0';
   *y=(long)HexToLong((char *)buffer);
   for(i=0;i<6;i++) buffer[i]=Empfange(); buffer[6]='\0';
   *z=(long)HexToLong((char *)buffer);
}

/*********************************************************************/
/* GetActPos :  Position abfragen und anzeigen                       */
/*********************************************************************/
void GetActPos(long *xpos,long *ypos,long *zpos)
{
  long xxpos,yypos,zzpos;    /* Achtung die Positionen werden als*/
                             /* Folge von Hexzahlen (char) uebertragen*/
  double x,y,z;

  StringOut("@0P");          /* Befehl Position abfragen */
  Rueckmeld();               /* Rückmeldung einlesen */
                             /* Position einlesen */
  ReceivePosition(&xxpos,&yypos,&zzpos);

  x=(double)xxpos/AchsfaktorX;
  *xpos=xxpos;               /* aktuelle X-Position merken */

  y=(double)yypos/AchsfaktorY;
  *ypos=yypos;               /* aktuelle Y-Position merken */

  z=(double)zzpos/AchsfaktorZ;
  *zpos=zzpos;               /* aktuelle Z-Position merken */

                             /* Ausgabe Position */
  printf("\nPosition:\tX : %.2lf\tY : %.2lf\tZ : %.2lf\n",x,y,z);
}
```

```
/*********************************************************************/
/* InitializeRs232 :  Initialisierung RS232                          */
/*********************************************************************/
int InitializeRs232(long Baud,int com)
{
  CloseRs232();              /* Schnittstelle schliessen */
  outp(XFB,0x03);            /* 8DatenBit,1Stopbit,Keine Parität */
  SelectDivisorRegs;         /* Divisor Register auswählen */
  switch(Baud)               /* Baudrate setzen */
  {
    case 300   : Set300Bd;   break;
    case 600   : Set600Bd;   break;
    case 1200  : Set1200Bd;  break;
    case 2400  : Set2400Bd;  break;
    case 4800  : Set4800Bd;  break;
    case 9600  : Set9600Bd;  break;
    case 19200 : Set19200Bd; break;
    case 38400 : Set38400Bd; break;
  }

  SelectDataRegs;            /* Datenregister auswählen */
  outp(XFE,0);
  EnableComInt(com);         /* Schnittstelleninterrupt erlauben */
  outp(0x20,0x20);           /* Interruptcontroller 8259 */

  R_Ptr=0;W_Ptr=0;           /* Pufferzeiger initialisieren */
}

/*********************************************************************/
/* Empfange :  Zeichen empfangen bzw. aus Pufer lesen */
/*********************************************************************/
char Empfange()
{
  static char dat;
  static long i=0;

  while (R_Ptr==W_Ptr)       /* auf ein Zeichen warten */
    {
      if (kbhit()            /* Schleife kann mit ESC abgebrochen werden */
       {
         int c=getch();      /* Taste lesen */
         if (c==27)          /* bei ESC Abbruch */
          {
            return(0);       /* 0 als Fehlerkennung */
          }
       }
      i++;                   /* Laufzeitbegrenzung */
      if (i==0x00ffffff)     /* time out */
        {
           return(0);        /* 0 als Fehlerkennung */
        }
    }
  if (W_Ptr != R_Ptr)        /* wenn Zeichen empfangen */
    {
      dat=RSBuffer[R_Ptr++]; /* Zeichen aus Ringpuffer lesen */
      R_Ptr&=BufferSize;     /* Ringpuffer behandeln */
      return(dat);           /* Zeichen rückgeben */
    }
  else return(0xff);         /* kann eigentlich nie eintreten */
}
```

```
/*********************************************************************/
/* RSInterruptProc :  Interruptroutine fuer RS232 Interrupt          */
/*********************************************************************/
void _interrupt _far RSInterruptProc()
{
  static int error;
  static long time;

  error=LineState;           /* Linestatus lesen */
  time=0L;
  if((error&0x1c)==0)        /* kein Fehler */
    {
      RSError=FALSE;
                             /* Daten lesen, Pufferzeiger incrementieren */
      RSBuffer[W_Ptr++]=ComData;
    }
  else                       /* Zeichen noch nocht vollständig empfangen */
    {                        /* Warten mit Laufzeitbegrenzung */
      while( ( ((error&0x1c)!=0) && ((error&0x01)!=1) )
             &&
             ( ((time++)!=0x0fffffff) ) ) error=LineState;
                             /* Daten lesen, Pufferzeiger incrementieren */
      RSBuffer[W_Ptr++]=ComData;
                             /* Laufzeitfehler testen */
      if(time==0x0fffffff) RSError=TRUE;
    }
  W_Ptr&=BufferSize;         /* Ringpuffer handeln */
  if( ( ( ( W_Ptr+1 ) & BufferSize ) - R_Ptr ) == 0)
    {                        /* if writepointer+1 = readpointer */
      BufferOverflow=TRUE;   /* Bufferoverflow*/
    }
  outp(0x20,0x20);           /* PIC 8259 */
}

/*********************************************************************/
/* EnableComInt :  Schnittstelleninterrupt erlauben                  */
/*********************************************************************/
void EnableComInt(int com)
{
  outp(XFB,inp(XFB)&0x7f);
  outp(XFC,0x0b);
  outp(XF9,0x05);
  if(com==1) outp(0x21,inp(0x21)&0xef);
  else if(com==2) outp(0x21,inp(0x21)&0xf7);
}

/*********************************************************************/
/* DisableComInt :  Schnittstelleninterrupt verbieten                */
/*********************************************************************/
void DisableComInt(int com)
{
  if(com==1) outp(0x21,inp(0x21)|0x10);     /* OCW1 */
  else if(com==2) outp(0x21,inp(0x21)|0x08);

  outp(XFB,0x80);
  outp(XF8,0x00);
  outp(XF9,0x00);
  outp(XFA,0x00);
  outp(XFC,0x00);
  outp(XFD,0x00);
  outp(XFE,0x00);
}
```

```
/*********************************************************************/
/* CloseRs232 :  Schnittstelle rücksetzen                            */
/*********************************************************************/
void CloseRs232()
{
  outp(XF8,0x00);            /* Standardwerte 8250 Register */
  outp(XF9,0x00);
  outp(XFA,0x01);
  outp(XFB,0x80);
  outp(XF8,0x02);
  outp(XF9,0x00);
  outp(XFB,0x00);
  outp(XFC,0x00);
  outp(XFD,0x60);
  outp(XFE,0x00);
}


/*********************************************************************/
/* StringOut :  Ausgabe eines Befehls                                */
/*********************************************************************/
void StringOut(char *str)
{
   while(*str) { Sende(*str); str++;}
   Sende(0x0d);
}


/*********************************************************************/
/* Rueckmeld :  Rueckmeldung empfangen                               */
/*********************************************************************/
void Rueckmeld()
{
  int c;
  c=Empfange();
  if(c!=48) printf("Fehler : %c\n",c);
}



/*********************************************************************/
/* HAUPTPROGRAMM ANFANG                                              */
/*********************************************************************/

main( int argc, char **argv, char **envp )
 {
  char c='0';
  int i;

  printf("\nIsel-IMC4-XY-Test");
  printf("\n=================");

  if(argc)                   /* Kommadozeilenparameter auswerten */
    {
       sscanf(*++argv,"%c",&c);
       if(c=='1') ComPort=1;
       else if(c=='2') ComPort=2;
       else ComPort=1;
    }
  else ComPort=1;            /* Default com1 */
```

```
                            /* Interruptvektor und Basisadresse Rs232 */
if(ComPort==1)
  {
    printf("Eingestellte Schnittstelle ist Com1 !\n");
    BASE=BASE1; ComVect=Com1Vect;
  }
else if(ComPort==2)
  {
    printf("Eingestellte Schnittstelle ist Com2 !\n");
    BASE=BASE2; ComVect=Com2Vect;
  }
else
  {
    printf("Eingestellte Schnittstelle ist Com1 !\n");
    BASE=BASE1; ComVect=Com1Vect;
  }

                            /* disable Schnitttstelleninterrupt */
DisableComInt(ComPort);
                            /* alten Interruptvektor merken */
OldRSInt=_dos_getvect(ComVect);
                            /* Int.vektor auf eigene Routine setzen */
_dos_setvect(ComVect,RSInterruptProc);

W_Ptr=0; R_Ptr=0;
                            /* Schnittstelle neu initialisieren */
InitializeRs232(Baudrate,ComPort);

printf(" Baudrate ist %d bit/sec\n",Baudrate);

                            /* Normierung der Achsen */
AchsfaktorX=(double)StepsX/SteigungX;
AchsfaktorY=(double)StepsY/SteigungY;
AchsfaktorZ=(double)StepsZ/SteigungZ;

StringOut("@03");           /* Steuerung initialisieren XY-Achse */
Rueckmeld();                /* Rückmeldung empfangen */
StringOut("@0R3");          /* Referenzfahrt XY-Achse */
Rueckmeld();                /* Rückmeldung empfangen */

                            /* Tastaturbelegung anzeigen */
printf("Tastaturbelegung :\n\n");
printf("X-Achse :  + F1     - F2\n");
printf("Y-Achse :  + F3     - F4\n");
printf("Speed   :  + PgUp   - PgDn\n");
printf("Ende    :  ESC\n");

                            /* aktuelle Position anzeigen */
GetActPos(&xpos,&ypos,&zpos);

                            /* Hauptschleife */
while ((scancode!=1)&&(!BufferOverflow)&&(!RSError))
  {
    if (kbhit()!=0)         /* Schleife Tastaturabfrage*/
      {
                            /* Tastaturabfrage */
        key_entry=_bios_keybrd(_KEYBRD_READ);
        scancode=(key_entry>>8)&0xff;

        switch (scancode)
          {
            case 1:         /* ESC */
                break;
```

```
case 59:       /* F1 */
  {
    printf("  Start +X\n");
    sprintf(outstr,"@0A1000000,%d,0,%d",
                   TeachSpeed,TeachSpeed);
    StringOut(outstr);
    while(inp(0x60)==59) ClrTastBuffer();
    Sende(255);
    Rueckmeld();
    printf("  Stop\n");
    GetActPos(&xpos,&ypos,&zpos);
    break;
  }

case 60:       /* F2 */
  {
    printf("  Start -X\n");
    sprintf(outstr,"@0A-%ld,%d,0,%d",
                   xpos,TeachSpeed,TeachSpeed);
    StringOut(outstr);
    while(inp(0x60)==60) ClrTastBuffer();
    Sende(255);
    Rueckmeld();
    printf("  Stop\n");
    GetActPos(&xpos,&ypos,&zpos);
    break;
  }

case 61:       /* F3 */
  {
    printf("  Start +Y\n");
    sprintf(outstr,"@0A0,%d,1000000,%d",
                   TeachSpeed,TeachSpeed);
    StringOut(outstr);
    while(inp(0x60)==61) ClrTastBuffer();
    Sende(255);
    Rueckmeld();
    printf("  Stop\n");
    GetActPos(&xpos,&ypos,&zpos);
    break;
  }

case 62:       /* F4 */
  {
    printf("  Start -Y\n");
    sprintf(outstr,"@0A0,%d,-%ld,%d",
                   TeachSpeed,ypos,TeachSpeed);
    StringOut(outstr);
    while(inp(0x60)==62) ClrTastBuffer();
    Sende(255);
    Rueckmeld();
    printf("  Stop\n");
    GetActPos(&xpos,&ypos,&zpos);
    break;
  }
```

```
              case 73:       /* Page Up */
                {
                  TeachSpeed+=50;
                  if(TeachSpeed>5000) { beep; TeachSpeed=5000;}
                  putch(13);
                  printf("Speed : %04d",TeachSpeed);
                  break;
                }
              case 81:       /* Page Down */
                {
                  TeachSpeed-=50;
                  if(TeachSpeed<30) { beep; TeachSpeed=50;}
                  putch(13);
                  printf("Speed : %04d",TeachSpeed);
                  break;
                }
              default:       /* Falsche Taste */
                {
                  beep;
                  printf("  Falsche Eingabe !\n");
                  break;
                }
            }                /* end switch */
        }                    /* end if */
    }                        /* end while */

  printf("Ende\n");

  DisableComInt(ComPort);    /* Schnittstelleninterrupt disable */
  CloseRs232();              /* Schnittstelle schliessen */
                             /* Interruptvektor rücksetzen */
 _dos_setvect(ComVect,OldRSInt);
}

/*********************************************************************/
/* HAUPTPROGRAMM ENDE                                                */
/*********************************************************************/
```

D1 Example for Programming the IMC4 in CNC Mode

This example will show that an easy programming of the IMC4 for easy tasks can also be achieved using BASIC. With version 2.00 and higher, the IMC4 can be set to a data transfer speed of 9,600 baud. It is thus possible to program the control system using simple BASIC statements (as with the isel interface card series). The transfer of the CNC commands can naturally also be carried out in other programming languages using appropriate interface functions.

In D2, "CNC Program Sample", you will find a short sequencing program written in pseudo-code and in the appropriate CNC syntax, which is to be transferred to the control system. This program can be stored in an ASCII file in a simple fashion. In Appendix D3, you will find a short BASIC program with which such a file can be transferred to the control system as a CNC program. The program has been designed deliberately very simple; for normal operation, the evaluation of the feedback messages and the appropriate responses should be subjected to a more comprehensive treatment by the program. For the error messages and possible causes, please refer to Chapter 4 "Error Messages of the IMC4".

D2 CNC Sample Program

D2.1 Pseudo-Code

The pseudo-code used here will only serve to illustrate the program execution and will at the same time be self-explaining. The values provided in () each constitute a parameter for a command. Sections put in {} are to be understood as program blocks. The coordinates have been specified in millimetres and the velocities in steps/second. In the case of linear movements, X is used for the movement of the X axis and Y for the movement of the Y axis. For the Z axis, 1 parameter is specified in the case of absolute movements, and 2 parameters for relative movements. In the case of circular movements, R stands for the radius in millimetres, A for the starting angle and E for the end angle in degrees.

```
3_Achsen_initialisieren (XYZ);
Referenzgeschwindigkeit X(1000) Y(1000) Z(1000);
CNC_Programm (Anfang)
{
  Schleife (endlos)
  {
    Referenzfahrt (XYZ);
    3d_Mode (Aus);
    Spindel (Ein);
    Zeitverzögerung (1s);
    Bewege_Absolut X(120) Y(120) Z(-60) Geschwindigkeit(4000);
    Schleife (5)
    {
      Schleife (5)
      {
        Bewege_Relativ X(5) Y(0) Z1(-10) Z2(10) Geschwindigkeit(4000);
      }
      Bewege_Relativ X(0) Y(5) Z1(0) Z2(0) Geschwindigkeit(4000);
      Schleife (5)
      {
        Bewege_Relativ X(-5) Y(0) Z1(-10) Z2(10) Geschwindigkeit(4000);
      }
      Bewege_Relativ X(0) Y(5) Z1(0) Z2(0) Geschwindigkeit(4000);
    }
    Bewege_Relativ X(0) Y(0) Z1(0) Z2(0) Geschwindigkeit(4000);
    Bewege_Absolut X(0) Y(0) Z(0) Geschwindigkeit(4000);
    Spindel (Aus);
    Zeitverzögerung (1s);
    3d_Mode (Ein);
    Bewege_Relativ X(50) Y(50) Z1(-10) Z2(0) Geschwindigkeit(2000);
    Schleife (5)
    {
      Bewege_Relativ X(5) Y(0) Z1(0) Z2(0) Geschwindigkeit(1000);
    }
    Schleife (5)
    {
      Bewege_Relativ X(0) Y(5) Z1(0) Z2(0) Geschwindigkeit(1000);
    }
    Zeitverzögerung (0.5s);
    Bewege_Absolut X(120) Y(120) Z(-60) Geschwindigkeit(4000);
    Kreisebene (XY);
    Kreisrichtung (Uhrzeigersinn);
    Kreis R(20) A(360) E(180) Geschwindigkeit(1000);
    Kreis R(20) A(180) E(0) Geschwindigkeit(1000);
    Kreisrichtung (Gegenuhrzeigersinn);
    Kreis R(20) A(0) E(180) Geschwindigkeit(1000);
    Kreis R(20) A(180) E(360) Geschwindigkeit(1000);
    Bewege_Absolut X(120) Y(120) Z(-60) Geschwindigkeit(4000);
    Kreisebene (YZ);
    Kreisrichtung (Uhrzeigersinn);
    Kreis R(20) A(360) E(180) Geschwindigkeit(1000);
    Kreis R(20) A(180) E(0) Geschwindigkeit(1000);
    Kreisrichtung (Gegenuhrzeigersinn);
    Kreis R(20) A(0) E(180) Geschwindigkeit(1000);
```

```
    Kreis R(20) A(180) E(360) Geschwindigkeit(1000);
    Bewege_Absolut X(120) Y(120) Z(-60) Geschwindigkeit(4000);
    Kreisebene (XZ);
    Kreisrichtung (Uhrzeigersinn);
    Kreis R(20) A(360) E(180) Geschwindigkeit(1000);
    Kreis R(20) A(180) E(0) Geschwindigkeit(1000);
    Kreisrichtung (Gegenuhrzeigersinn);
    Kreis R(20) A(0) E(180) Geschwindigkeit(1000);
    Kreis R(20) A(180) E(360) Geschwindigkeit(1000);
    Kreisebene (XY);
    Bewege_Relativ X(20) Y(0) Z1(0) Z2(0) Geschwindigkeit(1000);
    Bewege_Relativ X(0) Y(20) Z1(0) Z2(0) Geschwindigkeit(1000);
    Bewege_Relativ X(0) Y(0) Z1(-20) Z2(0) Geschwindigkeit(1000);
    Bewege_Relativ X(-20) Y(-20) Z1(20) Z2(0) Geschwindigkeit(1000);
    Bewege_Relativ X(-20) Y(0) Z1(-20) Z2(0) Geschwindigkeit(1000);
    Bewege_Relativ X(0) Y(-20) Z1(-20) Z2(0) Geschwindigkeit(1000);
    Zeitverzögerung (0.5s);
    Bewege_Relativ X(20) Y(0) Z1(0) Z2(0) Geschwindigkeit(2000);
    Bewege_Relativ X(0) Y(20) Z1(0) Z2(0) Geschwindigkeit(2000);
    Bewege_Relativ X(0) Y(0) Z1(-20) Z2(0) Geschwindigkeit(2000);
    Bewege_Relativ X(-20) Y(-20) Z1(20) Z2(0) Geschwindigkeit(2000);
    Bewege_Relativ X(-20) Y(0) Z1(-20) Z2(0) Geschwindigkeit(2000);
    Bewege_Relativ X(0) Y(-20) Z1(-20) Z2(0) Geschwindigkeit(2000);
  }
CNC_Programm (Ende);
```

D2.2 isel CNC Code

In the case of a spindle lead of 10 mm, the pseudo-code specified in D2.1, with a step resolution of 400 steps/revolution, the following CNC program would result (for converting mm into steps, see Section 1.3 "Conversion Factors", for calculating the parameters for circular movements, see Section 2.3 "Calculating the Circle Parameters"):

```
@07
@0d1000,1000,1000
@0i
77
z 0
p2,1,1
510
m2400,4000,2400,4000,-1200,4000,-1200,4000
0100,4000,0,4000,-200,1000,200,4000
35,-1
00,4000,100,4000,0,4000,0,4000
0-100,4000,0,4000,-200,1000,200,4000
35,-1
00,4000,100,4000,0,1000,0,4000
35,-6
00,4000,0,4000,1000,4000,0,4000
m0,4000,0,4000,0,4000,0,4000
p2,1,0
510
z 1
01000,2000,1000,2000,-200,2000,0,21
0100,1000,0,1000,0,1000,0,21
35,-1
00,1000,100,1000,0,1000,0,21
35,-1
55
m2400,4000,2400,4000,-1200,4000,-1200,4000
e0
f0
y1600,1000,-200,400,-0,-1,-1
f0
y1600,1000,-200,-400,0,1,1
f-1
y1600,2000,-200,400,0,-1,1
```

```
f-1
y1600,2000,-200,-400,0,1,-1
m2400,4000,2400,4000,-1200,4000,-1200,4000
e2
f0
y1600,1000,-200,400,-0,-1,-1
f0
y1600,1000,-200,-400,0,1,1
f-1
y1600,2000,-200,400,0,-1,1
f-1
y1600,2000,-200,-400,0,1,-1
m2400,4000,2400,4000,-1200,4000,-1200,4000
e1
f0
y1600,1000,-200,400,-0,-1,-1
f0
y1600,1000,-200,-400,0,1,1
f-1
y1600,2000,-200,400,0,-1,1
f-1
y1600,2000,-200,-400,0,1,-1
e0
0400,1000,0,1000,0,1000,0,21
00,1000,400,1000,0,1000,0,21
00,1000,0,1000,-400,1000,0,21
0-400,1000,-400,1000,400,1000,0,21
0400,1000,400,1000,0,1000,0,21
0-400,1000,0,1000,-400,1000,0,21
00,1000,-400,1000,-400,1000,0,21
p0,128,0
55
0400,2000,0,2000,0,2000,0,21
00,2000,400,2000,0,2000,0,21
00,2000,0,2000,-400,2000,0,21
0-400,2000,-400,2000,400,2000,0,21
0400,2000,400,2000,0,2000,0,21
0-400,2000,0,2000,-400,2000,0,21
00,2000,-400,2000,400,2000,0,21
30,-69
9
```

D3 BASIC Sample for Transferring a CNC Program

To transfer the CNC program, the following short BASIC program can be used, with which the file name of the CNC program could be transferred as a command line interpreter:

```
REM Beispielprogramm Download im CNC-Mode
REM ================================================

REM ----------------------Kommandozeilenparameter zum Einlesen vorbereiten
  OPEN COMMAND$ FOR INPUT AS #2
REM ---------------------------serielle Schnittstelle COM1 Initialisieren
REM -----------------------9800 Baud, keine Parität, 8 Datenbit, 1 Stopbit
  OPEN "com1:9600,N,8,1" FOR RANDOM AS #1
REM -----------für COM2  OPEN "com2:9600,N,8,1" FOR RANDOM AS #1 verwenden
REM ------------------------------------------------------Bildschirm löschen
  CLS
REM ----------------------------------------------------Dateilänge ermitteln
  flen = LOF(2)
  fpos = 0
REM ------------------------------------------------------Bilschirmausgabe
  LOCATE 10, 10
  PRINT "Prozentsatz übertragener Zeilen: "
  LOCATE 12, 10
  PRINT "letzte bekannte Rückmeldung: "
REM -----------Hauptschleife für Datenübertragung, solange nicht Dateiende
  DO WHILE NOT (EOF(2))
REM ---------------------------------------------------Zeile aus Datei lesen
  LINE INPUT #2, a$
REM ---------------------------------------------------Prozentsatz ausrechnen
  fpos = fpos + LEN(a$) + 2
  fproz = INT((100 * fpos / flen) + .5)
REM ----------------------------------Prozentsatz auf Bildschirm ausgeben
  LOCATE 10, 50: PRINT USING "###"; fproz;
REM ------------------------------------------Befehl an IMC4 übertragen
  PRINT #1, a$
REM -------------------auf Rückmeldung warten, Abbruch mit ESC ermöglichen
       DO WHILE LOC(1) < 1
             c$ = INKEY$
             IF c$ = CHR$(27) THEN END
       LOOP
REM ----------------------------------Rückmeldung von Schnittstelle lesen
  b$ = INPUT$(1, 1)
REM ----------------------------------Rückmeldung auf Bildschirm ausgeben
  LOCATE 12, 50: PRINT b$;
REM ------------------------------------------------------Fehler auswerten
  IF b$ <> "0" THEN PRINT "Fehler:"; b$: INPUT d$
REM ------------------------------------------------------Schleifenende
  LOOP
REM ------------------------------------------------------------------Ende
  END

REM ------------------------------------------------------------------------
REM Programmende
REM ------------------------------------------------------------------------
```

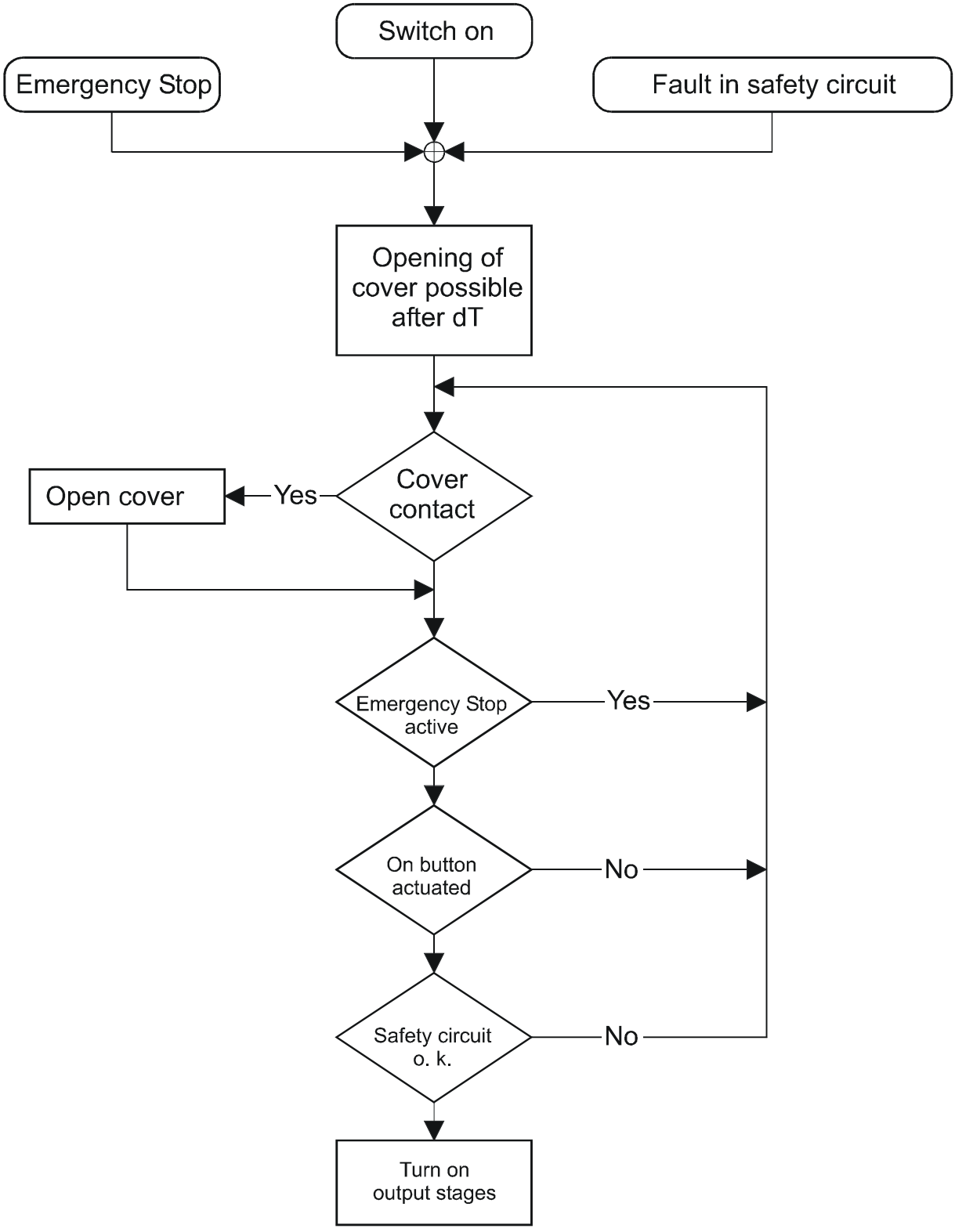
E1 Control Elements of the IMC4

The control elements of the IMC and their functions will be described here in brief for the sake of completeness. For an illustration of the functions, please refer to the following flowcharts.

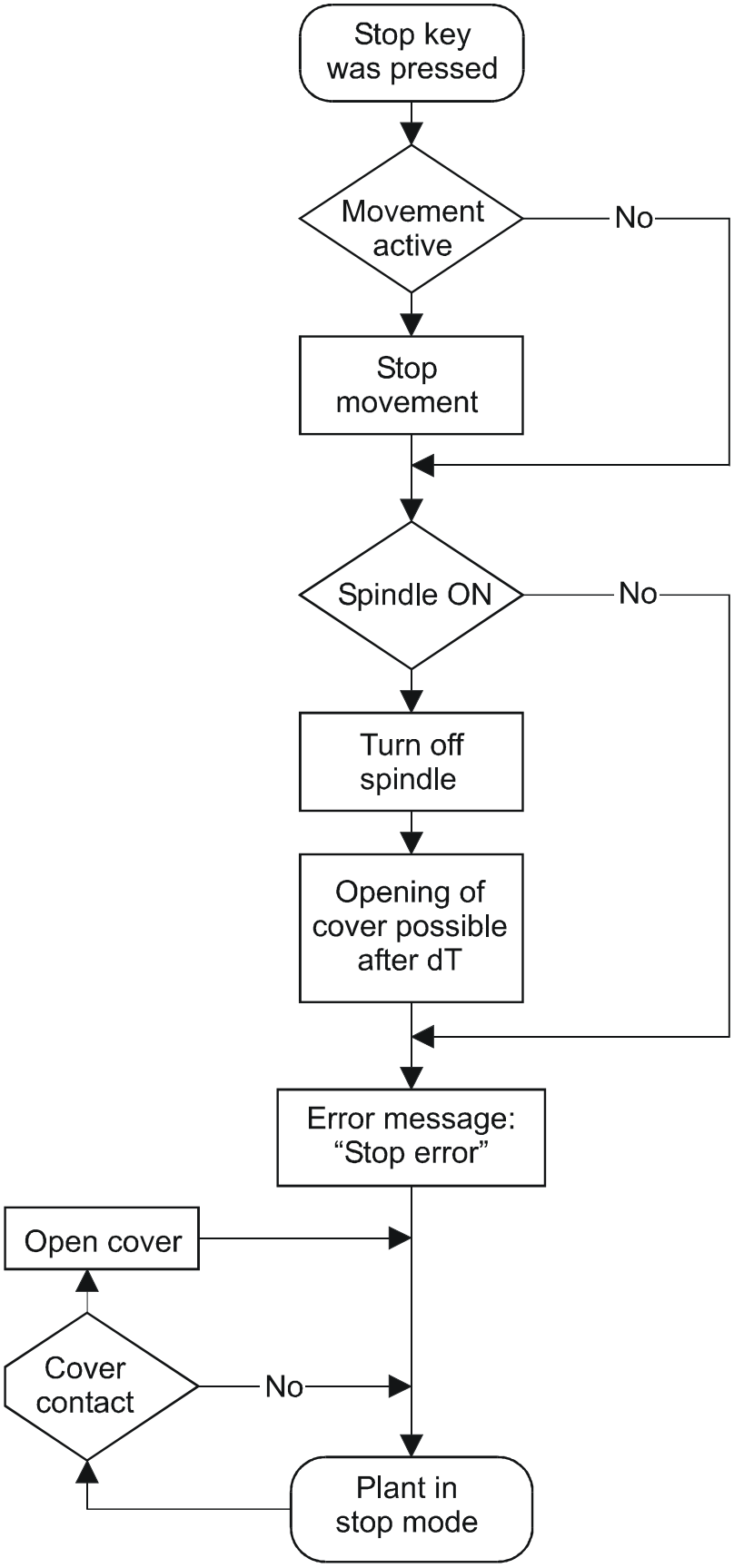
Control Element	Function
Emergency Stop switch	Emergency Stop function, safety relay - When actuated, output stage is switched off, all outputs are switched off, software reset - If Emergency Stop is active, the output stages cannot be connected.
ON button	Turning on the output stages, self-test, erasing the FlashProms - In initial state, intended to turn off the output stages - When simultaneously holding down the Start button, self-test is initiated - When simultaneously holding down the Stop key, the FlashProms are erased.
Keyswitch	Switchover from test mode to Automatic mode, erasing FlashProms - In Automatic mode, traversing is not possible with the cover open - In Test mode, traversing is possible with the cover open - To erase the FlashProms, the switch must be set to "Test".
Start button	Starting CNC programs, movements, erasing the FlashProms - Starting stored CNC programs - Starting stopped movements in CNC mode - Erasing FlashProms in the initial state of the control system
Stop button	Stopping movements, self-test - Stopping movements in CNC and DNC modes; when the spindle is running, it is then switched off - Running self-test if Stop is actuated when turning on the output stages.
Cover button	Opening the protection cover - When the button lights, the protection cover can be opened.

E2 Functionalities of the Control Elements

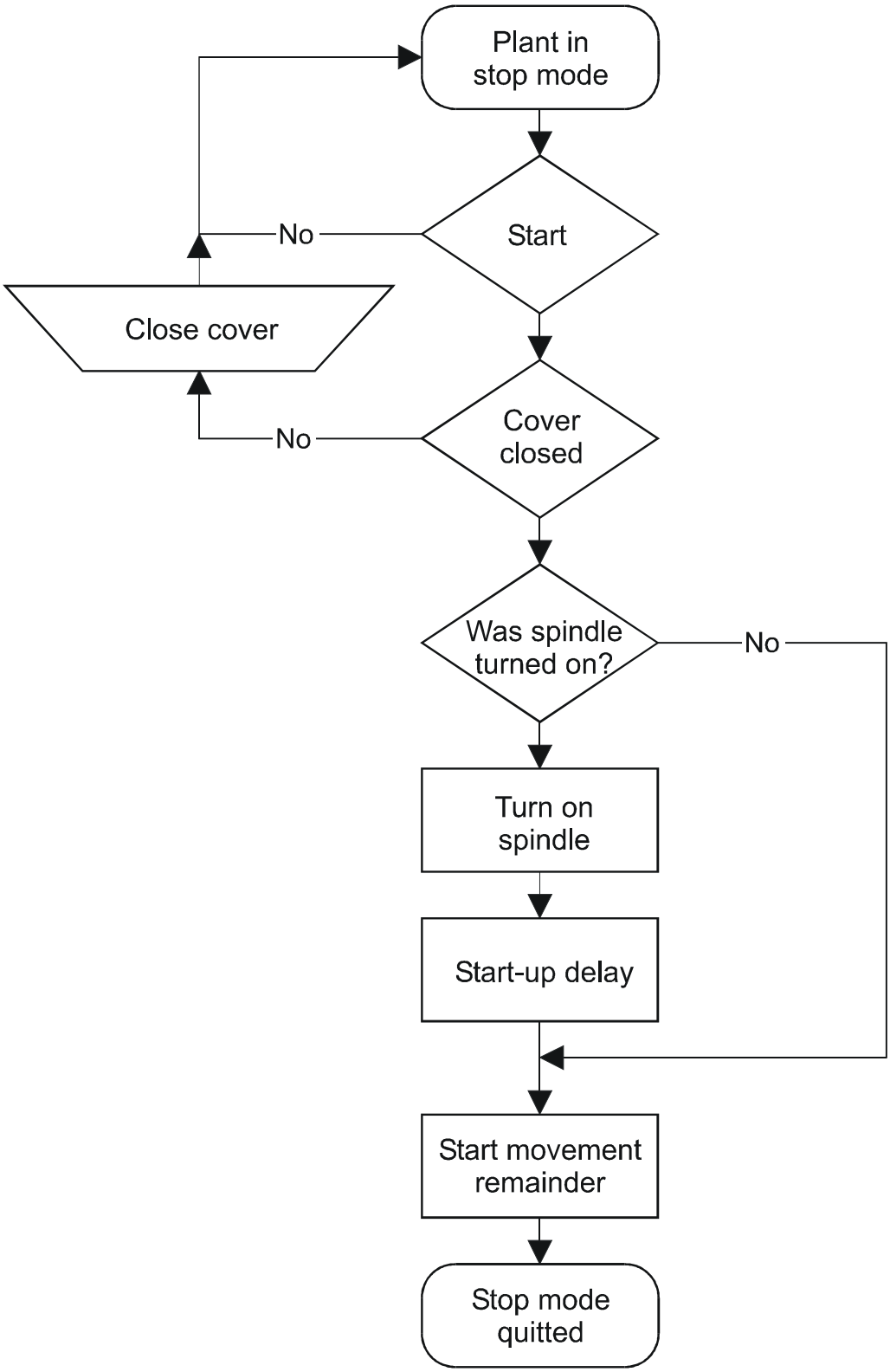
E2.1 Turning On the Output Stages



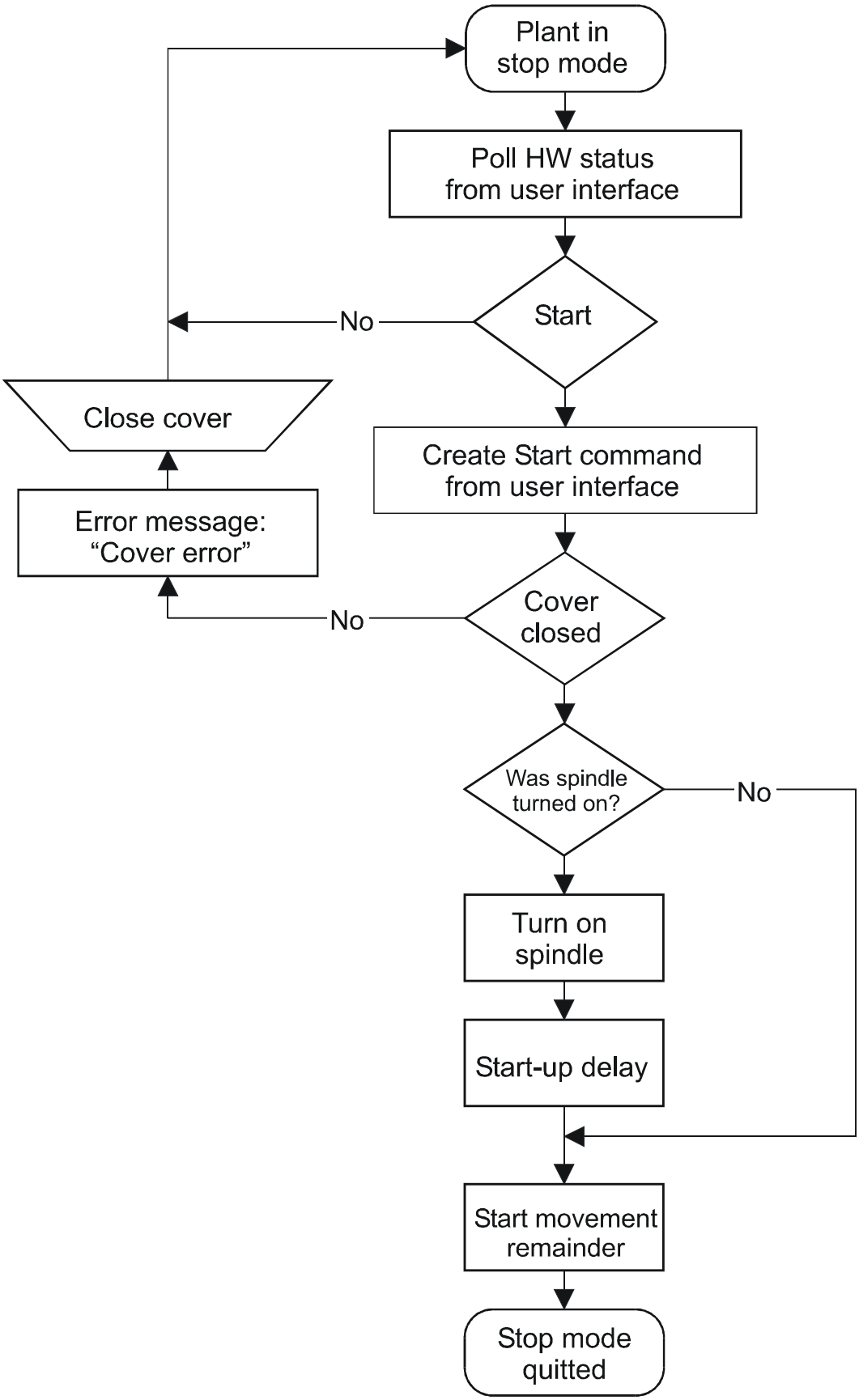
E2.2 Function of Stop Button



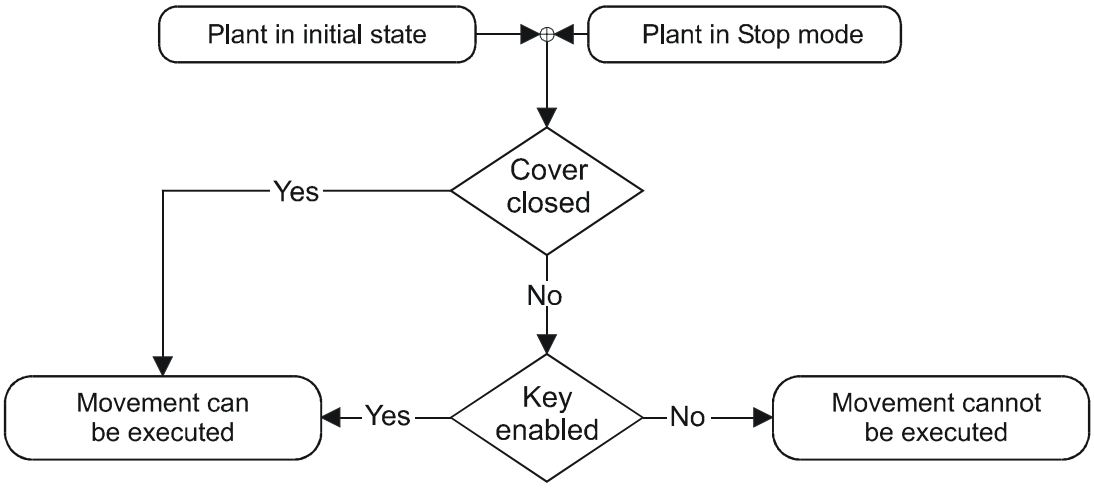
E2.3 Function of Start Button in CNC Mode



E2.4 Function of Start Button in DNC Mode



E2.5 Function of Keyswitch



E2.6 Erasing the FlashProms