isel-IMC4
Programmierung der IMC4 V2.xx

Software-Beschreibung

970393 BD001
09/2000

Die in dieser Druckschrift enthaltenen Informationen, technischen Daten und Maßangaben entsprechen dem neuesten technischen Stand (1/1999). Dennoch vorhandene Druckfehler und Irrtümer können jedoch nicht ausgeschlossen werden. Für Verbesserungsvorschläge und Hinweise auf Fehler sind wir dankbar.

Die in unseren Druckschriften verwendeten Soft- und Hardwarebezeichnungen der jeweiligen Firmen unterliegen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz.

Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

Unsere Firma ***isel****automation* ist seit vielen Jahren bekannt für Antriebssysteme und CNC-Steuerungen mit Schrittmotoren. Weltweit haben sich Schrittmotoren im kleinen und mittleren Leistungsbereich durchgesetzt. In Antriebsystemen mit mittleren Genauigkeits- und Dynamikanforderungen sind sie quasi nicht mehr wegzudenken. Schrittmotorsteuerungen arbeiten im Allgemeinen nicht mit geschlossenen Regelkreisen, so dass teure Sensoriken und Auswerteelektroniken entfallen. Dadurch ergibt sich neben einfachem Aufbau und Inbetriebnahme auch ein günstiges Preis-Leistungs-Verhältnis.

Seit vielen Jahren werden Steuerungssysteme auf Interfacekartenbasis (Interfacekarte 4.0/5.0/EP1090) von uns angeboten und finden in vielen Bereichen der Produktion, Automatisierung, Forschung und Weiterbildung ihren Einsatz. In Fortführung dieser Produktlinie wurde die Steuerung IMC4 entwickelt. Die IMC4 ist eine Microstep-Steuerung auf Mikroprozessor-Basis zur Ansteuerung von bis zu 4 Achsen. Zur Datenübertragung wird die IMC4 wie unsere Interfacekarten über eine serielle Schnittstelle angekoppelt.

Um eine gewisse Kompatibilität zu waren werden die Befehle analog zum CNC- und DNC-Modus der Interface-Kartenserie 4.0/5.0 zur Bedienung der Steuerung verwendet. Diese Beschreibung soll einen Einblick in die realisierten CNC- und DNC-Befehle der Steuerung geben.

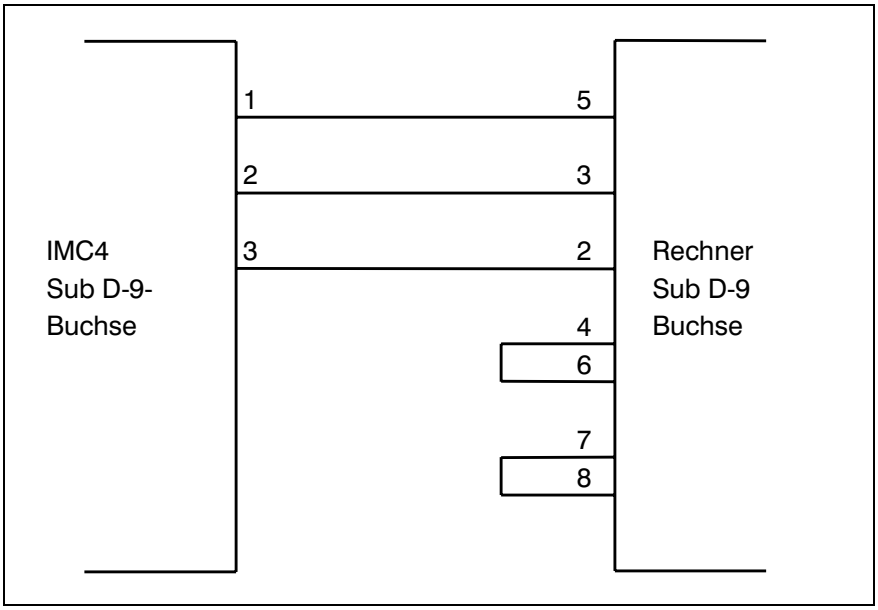
1.2 Anschluss der Steuerung an einen PC

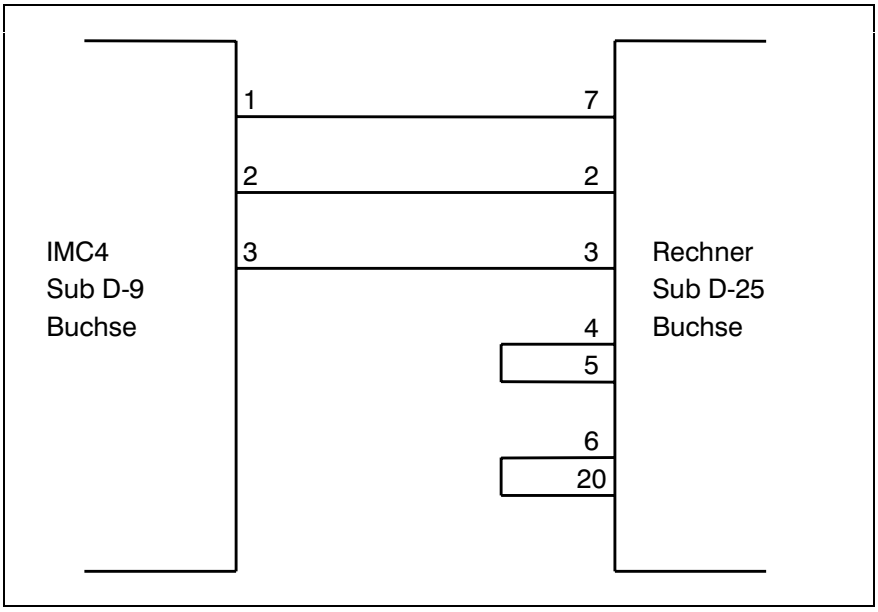
Zur Datenübertragung zwischen IMC4 und Steuerrechner wird eine serielle Schnittstelle nach RS232 eingesetzt. Die Verbindung ist über eine 3-Draht-Leitung realisiert. Ein Softwareprotokoll ermöglicht die fehlerfreie Übertragung der Zeichen.

Als Datenübertragungsparamter sind auf der IMC4 folgende Parameter festgelegt:

Baudrate:	19200 (9600 möglich durch entsprechende Jumpereinstellung)
Daten-Bit:	8
Stop-Bit:	1
Parität:	keine

Als Verbindungsleitung zum PC findet eine spezielle 3-Drahtverbindung Anwendung:





1.3 Umrechnungsfaktoren

Die Übergabe der Verfahrwege an die Steuerung erfolgt in Schritten. Im Gegensatz zur Übergabe von metrischen Maßen [mm] ermöglicht diese Art der Darstellung eine schnellere Dekodierung und Ausführung, da die Steuerung intern alle Positionen in Schritten darstellt. Die Umrechnung der Verfahrwege in Schrittmotorschritte muss vom Steuerrechner übernommen werden. Für diese Umrechnung werden folgende Informationen über die angeschlossene Mechanik benötigt:

Spindelsteigung:
- Steigung der in der Mechanik eingebauten Spindel
- gibt den Weg an, den sich der Schlitten bewegt (bei einer Umdrehung der Spindel)
- üblich sind Steigungen von 2.5, 4, 5 und 10 mm

Schritte/Umdrehung:
- Anzahl der Inkremente pro Umdrehung des Motors
- die Steuerung IMC4 rechnet in Halbschritten, dies bedeutet 400 Inkremente pro Motorumdrehung

Getriebefaktor:
- wird nur bei Schrittmotorantrieben mit Getriebe benötigt

Die Umrechnung von mm in Schritte kann mit folgender Formel erfolgen:

$$\text{Schritte} = \frac{\text{Verfahrweg in mm}}{\text{Spindelsteigung}} * \text{Schritte/Umdrehung} * \text{Getriebefaktor}$$

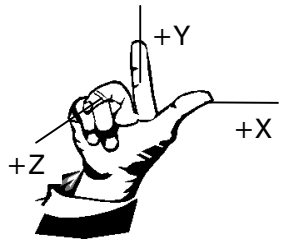
Für Geschwindigkeiten erwartet die Steuerung eine Angabe in Schritten (Inkremente) pro Sekunde. Die Umrechnung von in der Mechanik üblichen Geschwindigkeitsangaben (mm/s oder m/min) kann mit folgenden Formeln erfolgen:

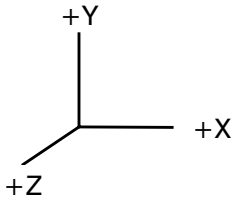
$$\text{V(Schritt/sek)} = \frac{\text{V[mm/s]} * \text{Schritte/Umdrehung} * \text{Getriebefaktor}}{\text{Spindelsteigung}}$$

$$\text{V(Schritt/sek)} = \frac{\text{V[m/min]} * \text{Schritte/Umdrehung} * \text{Getriebefaktor}}{\text{Spindelsteigung}} * \frac{60}{1000}$$

1.4 Das Koordinatensystem und Bezugspunkte

Die Festlegung von Koordinatensystemen und Bezugspunkten ist eine notwendige Voraussetzung für die Programmierung von Bearbeitungsbewegungen innerhalb des Arbeitsraumes von Werkzeugmaschinen. In Übereinstimmung mit der DIN 66217 wird ein rechtshändiges, rechtwinkliges Koordinatensystem mit den Achsen X, Y und Z verwendet. Die Z-Achse ist identisch mit der Achse der Bearbeitungsspindel. Die positive Richtung der Z-Achse verläuft vom Werkstück zum Werkzeug.





Zur Erzeugung der Bearbeitungsbewegungen kann entweder das Werkzeug oder das Werkstück bewegt werden. Das Koordinatensystem bezieht sich aber unabhängig davon immer auf das Werkstück. Somit ist für die Programmierung unabhängig davon, ob sich Werkzeug oder Werkstück bewegen. Der Programmierer geht immer davon aus, dass sich das Werkzeug relativ zum Koordinatensystem des stillstehenden Werkstücks bewegt.

Neben dem Koordinatensystem spielen Bezugspunkte bei der Programmierung von Werkzeugmaschinen eine wesentliche Rolle. Die wichtigsten seien hier genannt:

Maschinennullpunkt:
- liegt im Ursprung des Anlagenkoordinatensystems unveränderlich fest
- ist durch die Anlagenkonstruktion vorgegeben und kann nicht verändert werden

Referenzpunkt: - durch Endschalter auffindbarer, festgelegter Punkt, der meist in einer äußeren Ecke des Arbeitsraumes liegt und oft identisch mit dem Maschinennullpunkt ist

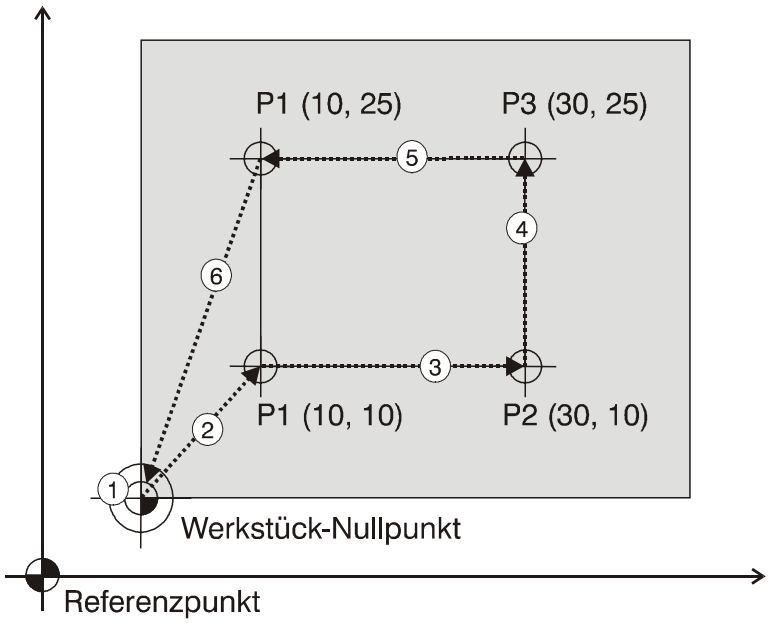
- kann auch auf einen festen Abstand zum Maschinennullpunkt eingestellt werden und bleibt dann unverändert
- kann meist mit der Genauigkeit von einem Weginkrement angefahren werden

Werkstücknullpunkt:
- Ursprung des Werkstückkoordinatensystems
- kann vom Programmierer frei gewählt werden und ist Bezugspunkt für die Programmierung

Die Festlegungen Ihre Anlage betreffend entnehmen Sie bitte den entsprechenden Bedienanleitungen und/oder Hardware- und Mechanikbeschreibungen.

1.5 Relative und Absolute Koordinatenangaben

Verfahrwege können in relativen (Kettenmaß) oder absoluten Koordinaten programmiert werden. Bei der Angabe von relativen Koordinaten ist der Bezugspunkt die aktuelle Werkzeugposition. Die Koordinaten stellen dann den Abstand des nächsten anzufahrenden Punkts von der aktuellen Position dar. Im Gegensatz dazu dient bei der Angabe von absoluten Koordinaten der Werkstücknullpunkt als Bezugspunkt. Die Koordinaten stellen dann den Abstand des nächsten anzufahrenden Punkts vom Werkstücknullpunkt dar. Folgendes Beispiel soll dies verdeutlichen:



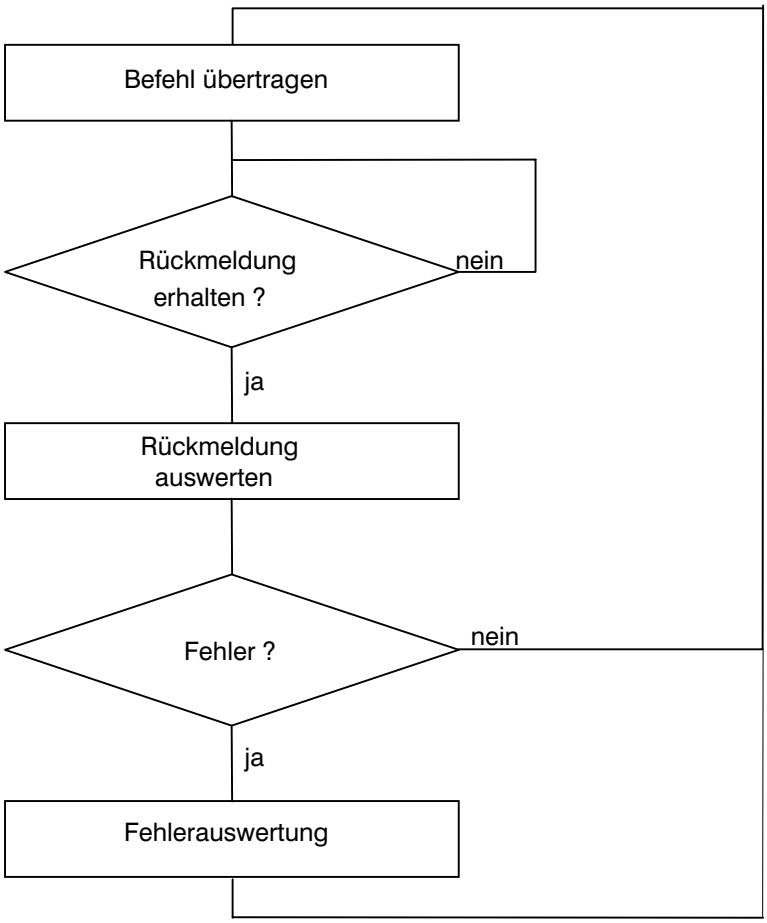
Es sollen 4 Bohrungen in ein Werkstück an den Punkten P1, P2, P3 und P4 eingebracht werden. Dabei sollen die Punkte in der Reihenfolge P1 --> P2 --> P3 --> P4 angefahren werden.
Für die Abarbeitung ergeben sich folgende relativen und absoluten Koordinaten:

	absolut	relativ
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 Der DNC-Modus und seine Befehle

2.1 Befehlsaufbau der DNC-Befehle

Im DNC-Modus betrieben, werden die von einem Steuerrechner übergebenen Datensätze, bzw. Befehle direkt ausgewertet und ausgeführt. Hierzu ist zu Beginn der Datenkommunikation eine sogenannte Initialisierung notwendig. Sie besteht aus dem Dateneröffnungszeichen @, der Gerätenummer (0 = Standard) und der Anzahl der zu verfahrenden Achsen. Anschließend werden der Steuerung die Programmschritte einzeln übergeben und von ihr direkt ausgeführt. Zur Überprüfung der Datenübertragung bzw. Meldung von aufgetretenen Fehlern werden über die Schnittstelle entsprechende ASCII-Zeichen an den Steuerrechner zurückgesendet. Dieses sogenannte Software-Handshake-Verfahren wird folgendermaßen realisiert:



Zunächst wird ein Befehl an die Steuerung übertragen. Der Befehl wird von der Steuerung dekodiert und abgearbeitet, anschließend generiert die Steuerung ein entsprechendes Quittierungs- oder Fehlerzeichen.

Diese Rückmeldung wird vom Steuerrechner ausgewertet. Ist ein Fehler aufgetreten muss eine entsprechende Fehlerauswertung und -beseitigung durchgeführt werden. Anschließend kann der nächste Befehl auf gleiche Art und Weise an die Steuerung übertragen werden.

Im Folgenden wird der Befehlsumfang des DNC-Modus der Steuerung IMC4 beschrieben.
Die angegebenen Programmierbeispiele in Turbo Pascal und Microsoft C sollen dem besseren Verständnis dienen.

2.2 Die Befehle der IMC4 im DNC-Modus

2.2.1 Initialisierung, Achsenanzahl setzen

Befehl: Achsenanzahl setzen

Zweck: Durch Übergeben der Achsenanzahl wird die Steuerung neu initialisiert.

Aufbau: @<GN><Achsen><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
<Achsen>	= Achsenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @07, @08

Erläuterung: Die Steuerung wird durch „@0“ adressiert, der anschließende Zahlenwert beinhaltet die Achsenkonfiguration. Dabei wird intern jede Achse durch ein Bit eines Binärwertes repräsentiert und somit ergeben sich folgende Werte:

1 --> X-Achse
3 --> X+Y-Achse
7 --> X+Y+Z-Achse
8 --> A Achse

Beschränkung: Nicht zulässig sind die Kombinationen „@00“, „@02“, „@04“, „@06“, sowie „@09“.
ACHTUNG: Die A-Achse muss immer separat initialisiert werden.

Programmierbeispiel: siehe Anhang B 3.1. Referenzfahrt, Achseninitialisierung

2.2.2 Referenzfahrt

Befehl: Referenzfahrt

Zweck: Die Steuerung verfährt alle angegebenen Achsen an ihre Nullpunkte (Referenzpunkte).Die Referenzpunkte der Achsen sind bei *isel*-Systemen immer in einer sinnvollen Defaultanordnung festgelegt, können aber durch entsprechende Initialiserungsbefehle angepasst werden.

Aufbau: @<GN>R<Achsen><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
R oder r	= Befehl Referenzfahrt
<Achsen>	= Achsenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0R7, @0r7, @0R8

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „R“ gibt an, dass eine Referenzfahrt ausgeführt werden soll. Der anschließende Zahlenwert definiert die Achsen, die eine Referenzfahrt durchführen sollen. Dabei wird intern jede Achse durch ein Bit eines Binärwertes repräsentiert und somit ergeben sich folgende Werte:
1 --> X Achse
2 --> Y Achse
3 --> X+Y Achse
4 --> Z Achse
5 --> X+Z Achse
6 --> Y+Z Achse
7 --> X+Y+Z Achse
8 --> A Achse
Die Reihenfolge der Ausführung ist dabei wie folgt festgelegt:
Z-Achse --> Y-Achse --> X-Achse --> A-Achse

Nach erfolgter Referenzfahrt sendet die Steuerung ihr Quittierungszeichen und wartet auf die nächsten Befehle. Die Steuerung kann erst wieder Befehle verarbeiten, nachdem die Referenzfahrt durch die Mechanik ausgeführt worden ist.

Beschränkung: Der Befehl kann erst verwendet werden, nachdem eine Initialisierung der Steuerung durch den Befehl „Achsenanzahl setzen“ erfolgt ist und beschränkt sich auf die dort vorgegebene Achsenkonfiguration. Bei falscher Achsangabe erfolgt die Fehlerrückmeldung „3“. Befindet sich die Steuerung im 3D-Modus, schaltet der Befehl wieder in die 2,5 dimensionale Ausführung zurück.

ACHTUNG: Die A-Achse muss immer separat referenziert werden.
ACHTUNG: Bei nicht angeschlossenem Referenzschalter, wird die entsprechende
Achse permanent angesteuert. Durch Betätigen des Stop-Tasters besteht jedoch die Möglichkeit den Referenzschalter zu simulieren.

Programmierbeispiel: siehe Anhang B 3.1. Referenzfahrt, Achseninitialisierung

2.2.3 Referenzgeschwindigkeit festlegen

Befehl: Referenzgeschwindigkeit setzen

Zweck: Der Befehl definiert, getrennt für jede Achse die Geschwindigkeit, mit der eine Referenzfahrt ausgeführt wird.

Aufbau:

@<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)

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
d	= Befehl Referenzgeschwindigkeit setzen
<Gx>	= Referenzgeschwindigkeit x
<Gy>	= Referenzgeschwindigkeit y
<Gz>	= Referenzgeschwindigkeit z
<Ga>	= Referenzgeschwindigkeit a
<CR>	= Carriage Return als Befehlsabschluss

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

Erläuterung: Werden der Steuerung keine Informationen zur Referenzgeschwindigkeit übergeben, erfolgt die Ausführung mit einem Default-Wert. Ein geänderter Wert bleibt beim Ausschalten nicht erhalten.

Beschränkung: Die angegebenen Geschwindigkeiten müssen innerhalb des gültigen Wertebereiches für Geschwindigkeiten liegen.
Eine zu hoch gewählte Referenzgeschwindigkeit kann in Verbindung mit einer großen Spindelsteigung, durch die vorhandene Massenträgheit, zu einer Beschädigung der Referenzschalter führen. Die Steuerung benötigt eine Schalthysterese des angeschlossenen Nulllagenschalters. Dies ist bei Anschluss von elektronischen Nullsensoren zu beachten!

Programmierbeispiel: siehe Anhang B 3.2. Referenzgeschwindigkeit einstellen

2.2.4 Ausführen einer relativen Bewegung

Befehl: Bewegung relativ

Zweck: Die Steuerung generiert entsprechend der übergebenen Schrittanzahl und Schrittgeschwindigkeit eine relative Bewegung. Die Verfahrbewegung wird sofort ausgeführt.

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

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
A oder a	= Befehl Bewegung relativ
<Sx>	= Schrittanzahl x
<Gx>	= Geschwindigkeit
<Sy>	= Schrittanzahl y
<Gy>	= Geschwindigkeit
<Sz>, <Sz1>	= Schrittanzahl z
<Sz2>	= Schrittanzahl z, 2. Bewegung bei 2.5d, 3 Achsen
<Gz>, <Gz1>	= Geschwindigkeit
<Gz2>	= Geschwindigkeit z, 2. Bewegung bei 2.5d, 3 Achsen
<Sa>	= Schrittanzahl a, bei 4 Achsen
<Ga>	= Geschwindigkeit, bei 4 Achsen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:

@0A 5000,900	(nur x-Achse)
@0A 50,900,20,9000	(x und y-Achse)
@0A 30,800,10,900,4,90,-4,30	(x,y und z-Achse, bei 3 Achsen)
@0A 30,800,10,900,4,90,-4,30	(x,y,z und a-Achse, bei 4 Achsen)

Erläuterung: Die Steuerung wird durch „@0“ adressiert; „A“ oder „a“ gibt an, dass eine relative Bewegung erfolgen soll. Die Steuerung erwartet nun für jede Achse ein Zahlenpaar bestehend aus Schrittanzahl und Geschwindigkeit. Die Bewegung erfolgt im Relativmaß, d.h. bezogen auf die letzte Position. Die Anzahl der Angaben muss mit der Achsenzahl übereinstimmen, d.h. ein Parameterpaar bei x-Betrieb, zwei Parameterpaare bei xy-Betrieb, vier Parameterpaare für xyz-Betrieb und vier Parameterpaare für xyza-Betrieb. Die einzelnen Zahlen müssen durch Kommata getrennt werden. Für die z-Achse werden bei 3 Achsen und 2,5D zwei Zahlenpaare erwartet, da für Bearbeitungsanwendungen sehr häufig die Situation „Fahren, Werkzeug absenken und anschließend anheben“ vorkommt. Im 2.5D-Interpolationsbetrieb werden zuerst die Bewegungen der x- und y-Achse verfahren (linear interpoliert), anschließend wird die z-Achse zuerst um die in z1-angegebenen und dann um die in z2-angegebenen Werte verfahren. Besteht die Notwendigkeit nur eine Achse zu bewegen, so sind dennoch für alle initialisierten Achsen Werte zu übertragen. Dabei ist für die Schrittzahlen der nicht bewegten Achsen 0 anzugeben. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter („0“). Die Steuerung kann erst wieder nach Ausführung des Befehles neue Befehle verarbeiten.

Beschränkung: Der Befehl kann erst verwendet werden, nachdem die Achsenanzahl gesetzt worden ist. Darüber hinaus prüft die Steuerung nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Im 2.5D-Interpolationsbetrieb wird die Geschwindigkeitsangabe der Achse mit dem längsten Weg als Bahngeschwindigkeit übernommen und die Geschwindigkeit der anderen Achsen entsprechend dem Wegeverhältnis angepasst. Dem gegenüber wird im 3D-Interpolationsbetrieb die Geschwindigkeitsangabe der x-Achse als Vorgabewert für die Bahngeschwindigkeit herangezogen.

Programmierbeispiel: siehe Anhang B 3.3. Relative Bewegung

2.2.5 Ausführen einer absoluten Bewegung

Befehl: Bewegung zur absoluten Position

Zweck: Die Steuerung fährt mit den angegebenen Geschwindigkeiten an die angegebene Position. Die Verfahrbewegung wird sofort ausgeführt.

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

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
M	= Befehl Bewegung absolut
<Sx>	= Position x
<Gx>	= Geschwindigkeit
<Sy>	= Position y
<Gy>	= Geschwindigkeit
<Sz>, <Sz1>	= Position z
<Sz2>	= Position z, 2. Bewegung immer = 0
<Gz>, <Gz1>	= Geschwindigkeit
<Gz2>	= Geschwindigkeit
<Sa>	= Position a, bei 4 Achsen
<Ga>	= Geschwindigkeit, bei 4 Achsen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:

@0M 5000,900	(nur x-Achse)
@0M 50,900,20,9000	(x und y-Achse)
@0M 30,800,10,900,4,90,0,30	(x,y und z-Achse, bei 3 Achsen)
@0M 30,800,10,900,4,90,4,30	(x,y,z und a-Achse, bei 4 Achsen)

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „M“ gibt an, dass eine Absolut-Position folgt. Aus Kompatibilitätsgründen zum relativen Positionierbefehl werden auch hier bei 3 Achsen für die z-Achse zwei Zahlenpaare erwartet. Die zweite Positionsangabe der z-Position muss dann jedoch Null sein und wird ignoriert. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter. Die Steuerung kann erst wieder nach Ausführung des Befehles neue Befehle verarbeiten.

Beschränkung: Der Befehl kann erst verwendet werden, nachdem die Achsenanzahl gesetzt worden ist. Die Steuerung prüft nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Programmierbeispiel: siehe Anhang B 3.4. Absolute Bewegung

2.2.6 Abfragen der Ist-Position

Befehl: Positionsabfrage

Zweck: Die Steuerung gibt die momentane Ist-Position aller Achsen an den übergeordneten Rechner zurück.

Aufbau: @<GN>P<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
P	= Befehl Positionsabfrage
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0P

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „P“ gibt an, dass eine Positionsabfrage erfolgt. Die Steuerung bestätigt dies mit dem Handshake-Charakter und gibt anschließend im hexadezimalen Format die Positionswerte aller Achsen aus (bei bis zu 3 Achsen 18 hexadezimale Ziffern, bei 4 Achsen 24 hexadezimale Ziffern).

Der Aufbau der rückgemeldeten Position ist folgendermaßen:

z. B.: 00010002000FFFFFFE für 3 Achsen

Position x = 000100, hexadezimal im 2er-Komplement, entspricht 256 dezimal.
Position y = 02000F, hexadezimal im 2er-Komplement, entspricht 131087 dezimal.
Position z = FFFFFE, hexadezimal im 2er-Komplement, entspricht -2 dezimal.

z. Bsp:. 000B00044000000FFE003040 für 4 Achsen

Position x = 000B00, hexadezimal im 2er-Komplement, entspricht 2816 dezimal.
Position y = 044000, hexadezimal im 2er-Komplement, entspricht 278528 dezimal.
Position z = 000FFE, hexadezimal im 2er-Komplement, entspricht 4094 dezimal.
Position a = 003040, hexadezimal im 2er-Komplement, entspricht 12352 dezimal.

Beschränkung: Der Befehl kann nur verwendet werden, wenn keine Verfahrbewegung stattfindet (wenn sich die Anlage im Stop-Zustand befindet). Die Steuerung kann nicht prüfen, ob die Ist-Position der aktuellen Position der Mechanik entspricht, da kein Regelkreis vorhanden ist.

ACHTUNG: Es werden bei bis zu 3 Achsen immer die Positionen für drei Achsen durch die Funktion zurückgegeben, unabhängig von der Anzahl der definierten Achsen. Bei 4 Achsen werden immer Positionen für 4 Achsen zurückgegeben.

Programmierbeispiel: siehe Anhang B 3.6. Positionsabfrage

2.2.7 Nullpunkt setzen

Befehl: Nullpunkt am aktuellen Punkt setzen

Zweck: Die Steuerung speichert die momentane Position als virtuellen Nullpunkt für die angegebene(n) Achse(n). Die nächsten „Verfahre absolut“-Anweisungen berücksichtigen diesen virtuellen Nullpunkt als neuen Bezugspunkt.

Aufbau: @<GN>n<Achsen><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
n	= Befehl Nullpunkt setzen
<Achsen>	= Achsenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0n7, @0n1, @0n8

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „n“ gibt an, dass eine Nullpunktverschiebung vorgenommen werden soll. Nach dem Befehl werden der Steuerung die Achsen mitgeteilt, für die eine Nullpunktverschiebung durchgeführt werden soll. Dabei wird intern jede Achse durch ein Bit eines Binärwertes repräsentiert und somit ergeben sich folgende Werte:

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

Die Steuerung meldet sich nach erfolgter Ausführung mit einer Rückmeldung.

Beschränkung: Der virtuelle Nullpunkt hat nur für den Befehl „Verfahre absolut“ eine Bedeutung. Relativpositionierung wird vom virtuellen Nullpunkt nicht beeinflusst, da hier ein relativer Verfahrvektor angegeben wird.

ACHTUNG: Die Nullpunktverschiebung für die A-Achse muss immer separat durchgeführt werden.

Programmierbeispiel: siehe Anhang B 3.5. Nullpunktverschiebung

2.2.8 3D-Interpolation Ein-/Ausschalten

Befehl: 3D-Linearinterpolation Ein-/Ausschalten

Zweck: Die Steuerung erweitert die 2.5d Interpolation des Standard-Betriebssystemes auf eine 3-dimensionale Interpolation. Durch Verwenden des Befehles kann diese Interpolation gezielt aus- und eingeschaltet werden.

Aufbau: @<GN>z<Status><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
z	= Befehl 3D-Interpolation
<Status>	= 0 --> Ausschalten, 1 --> Einschalten
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0z1, @0z0

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „z1“ ändert die Interpolation von 2D- auf 3D-Betrieb. Die Anweisung wirkt modal, d. h. alle relativen und absoluten Bewegungen werden dreidimensional ausgeführt. Die Angabe von z2-Parametern bei 3 Achsen in diesen Verfahrbewegungen wird ignoriert. Die Geschwindigkeitsangabe der Interpolation muss bei der x-Angabe erfolgen. Bei 4 Achsen wird die 4. Achse entsprechend nachgeführt.

Programmierbeispiel: siehe Anhang B 3.8. 3D-Interpolation

2.2.9 Ebenenwahl für Kreisinterpolation

Befehl: Ebenenwahl

Zweck: Einstellen der Interpolationsebene für die Kreisinterpolation. Kreise sind nur innerhalb einer Ebene definiert. Die Defaultebene für die Kreisinterpolation ist die XY-Ebene. Durch den Ebenenwahl-Befehl besteht hier jedoch die Möglichkeit jede andere Ebenenkonfiguration als Kreisebene zu definieren.

Aufbau: @<GN>e<Ebene><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
e	= Befehl Kreisebene setzen
<Ebene>	= Ebenenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0e1, @0e0

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „e“ gibt an, dass die Ebene für die Kreisinterpolation eingestellt werden soll. Der anschließende Zahlenwert definiert die Ebene in folgender Weise:

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

Beschränkung: Diese Befehl ist modal wirksam, d.h. eine Ebenenwahl für die Kreisinterpolation bleibt erhalten bis sie von einer erneuten Ebenenwahl überschrieben wird.

Programmierbeispiel: siehe Anhang B 3.7. Ebenenwahl

2.2.10 Einstellen der Kreisrichtung für die Kreisinterpolation

Befehl: Kreisrichtung einstellen

Zweck: Einstellen der Kreisrichtung für die Kreisinterpolation. Die Kreisinterpolation wird durch zwei aufeinanderfolgende Befehle ausgelöst. Der erste Befehl legt die Kreisrichtung fest, im Zweiten (s. 2.2.11.) werden die Interpolationsparameter übergeben.

Aufbau: @<GN>f<Richtung><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
f	= Befehl Kreisrichtung einstellen
<Richtung>	= 0 --> CW (Uhrzeigersinn), -1 --> CCW (gegen Uhrzeigersinn)
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0f-1, @0f0

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „f“ gibt an, dass die Richtung für die Kreisinterpolation eingestellt werden soll. Der anschließende Zahlenwert definiert die Richtung in folgender Weise:

0 --> CW (Kreisinterpolation im Uhrzeigersinn)
-1 --> CCW (Kreisinterpolation entgegen dem Uhrzeigersinn)

Beschränkung: Die Richtung für die Kreisinterpolation ist prinzipiell vor jeder Kreisbewegung zu programmieren.

Programmierbeispiel: siehe 2.2.11 Kreisinterpolation, 2.3 Die Berechnung der Kreisparameter,
Anhang A 1, Software-Routinen zur Berechnung der Param. beim Kreisbefehl in Turbo-Pascal
Anhang A 2. Softwareroutinen zur Berechnung der Parameter beim Kreisbefehl in C
Anhang B 3.7. Ebenenwahl

2.2.11 Kreisinterpolation

Befehl: Kreisinterpolation

Zweck: Bearbeiten von Kreisen und Kreisbögen mit konstanter Bahngeschwindigkeit. Die Kreisinterpolation wird durch zwei aufeinanderfolgende Befehle ausgelöst. Der erste Befehl legt die Kreisrichtung fest (s. 2.2.10.), im Zweiten werden die Interpolationsparameter übergeben.

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

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
<B>	= Bogenlänge in Schritten
<V>	= Geschwindigkeit
<D>	= Interpolationsparameter
<Xs>	= Startpunkt x
<Ys>	= Startpunkt y
<Rx>	= Richtung x
<Ry>	= Richtung y
<CR>	= Carriage Return als Befehlsabschluss

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

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „y“ gibt an, dass eine Kreisinterpolation ausgeführt werden soll. Die Bogenlänge gibt die Länge des Bogens in Schritten zwischen dem Start- und Endpunkt der Kreisinterpolation an. Als Geschwindigkeitsangaben sind alle ganz zahligen Werte innerhalb des gültigen Wertebereiches für Geschwindigkeiten zulässig. Der Interpolationsparameter muss übergeben werden, da die Steuerung aufgrund ihrer Speicherkapazität nicht in der Lage ist diesen Parameter selbst zu berechnen. Die Parameter Xs und Ys geben den Startpunkt des Bogens relativ zum Kreismittelpunkt an. Rx und Ry geben an, in welchem Quadranten des Kreises die Interpolation startet. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter („0“). Die Steuerung kann erst wieder nach Ausführung des Befehles neue Befehle verarbeiten.

ACHTUNG: Zur Berechnung der Parameter lesen Sie bitte das Kapitel „Berechnung der Parameter für die Kreisinterpolation“.

Beschränkung: Der Befehl kann erst verwendet werden, nachdem die Achsenanzahl gesetzt worden ist. Darüber hinaus prüft die Steuerung nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Programmierbeispiel: siehe 2.2.10 Einstellen der Kreisrichtung für die Kreisinterpolation
2.2.9 Ebenenwahl für Kreisinterpolation,
2.3 Die Berechnung der Kreisparameter,
Anhang A 1, Software-Routinen zur Berechnung der Param. beim Kreisbefehl in Turbo-Pascal
Anhang A 2, Software-Routinen zur Berechnung der Parameter beim Kreisbefehl in C

2.2.12 Starten einer angehaltenen Bewegung

Befehl: Start

Zweck: Eine angehaltene Bewegung soll fortgeführt werden.

Aufbau: @<GN>S<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
S oder s	= Befehl Start
<CR>	= Carriage Return als Befehlsabschluss

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „S“ gibt an, dass eine angehaltene Bewegung gestartet und somit der Rest der eigentlichen Bewegung zur Ausführung gebracht werden soll. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter („0“) oder mit einer Fehlermeldung, falls kein Bewegungsrest im Speicher vorhanden ist. Die Steuerung kann erst wieder nach Ausführung der restlichen Bewegung neue Befehle verarbeiten.

Beschränkung: Der Befehl ist nur sinnvoll, wenn eine Bewegung angehalten wurde und sich die Steuerung im Stopp-Modus befindet. In diesem Modus kann über den Befehl „Diagnose“ (siehe 2.2.19 Diagnose) der aktuelle Status des Start-Tasters abgefragt und bei dessen Betätigung vom Bedienprogramm ein Startbefehl erzeugt werden. Im DNC-Modus ist eine Bedienung über die Tasten der Steuerung nur auf diese Weise möglich, da die Steuerung nicht direkt auf die Starttaste reagiert.

Programmierbeispiel: -

2.2.13 Lesen von Ports

Befehl: Port lesen

Zweck: Der Befehl ermöglicht den aktuellen Zustand von logischen oder physikalischen Eingangsports über die serielle Schnittstelle zu ermitteln.

Aufbau: @<GN>b<Portnr><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
b	= Befehl Port lesen
<Portnr>	= Portnummer s.u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0b0, @0b1

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „b“ gibt an, dass der Status eines Eingabeports ermittelt werden soll. Anschließend wird die Portnummer übermittelt und der Befehl mit Carriage Return abgeschlossen. Die Steuerung antwortet mit dem Software-Handshake „0“ gefolgt von zwei Zeichen, die einen Hexadezimalwert angeben, der dem aktuellen Status des Eingangsports entspricht. Für die Steuerung IMC4 sind folgende Ports mit entsprechenden Funktionalitäten definiert:

Port	Zustand	Funktion	
0	00 - FF	User E/A (auch noch mit 65531 möglich)	
		Bit0	Stoptaster
		Bit1	Start-Taster
		Bit2	User Input 1
		Bit3	User Input 2
		Bit4	Not-Ausschalter
		Bit5	Eintaster
		Bit6	Haubenschalter
		Bit7	Schlüsselschalter
1	00	Haube ist geöffnet	
	01	Haube ist geschlossen	
2	00	Spindel ist ausgeschaltet	
	01	Spindel ist eingeschaltet	
3	00	Motorströme sind ausgeschaltet	
	01	Motorströme sind eingeschaltet	

Beschränkung: Eine Rückgabe der Portzustände erfolgt nur, wenn die Steuerung mit dem Software-Handshake „0“ antwortet. Der Befehl kann nur verwendet werden, wenn keine Bewegung abgearbeitet wird.

Programmierbeispiel: siehe Anhang B 3.9. Port lesen

2.2.14 Schreiben von Ports

Befehl: Port schreiben

Zweck: Der Befehl erlaubt es, logische oder physikalische Ausgangsports mit definierten Werten über die serielle Schnittstelle zu beschreiben.

Aufbau: @<GN>B<Portnr>,<Wert><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
B	= Befehl Port schreiben
<Portnr>	= Portnummer s.u.
<Wert>	= neuer Portwert
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0B1,1

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „B“ gibt an, dass der Wert eines Ausgabeports gesetzt werden soll. Anschließend wird die Portnummer und der neue Portwert getrennt durch Komma übermittelt und der Befehl mit Carriage Return abgeschlossen. Die Steuerung antwortet mit dem Software-Handshake „0“, falls die Ausführung erfolgreich war, oder mit einer Fehlermeldung, falls falsche Portnummern und/oder Werte übergeben wurden. Für die Steuerung IMC4 sind folgende Ports mit entsprechenden Funktionalitäten definiert:

Port	Wert	Funktion
0	0 - 255	User E/A (auch noch mit 65529 möglich)
		Bit0 reserviert
		Bit1 reserviert
		Bit2 reserviert (Relais, Spindel)
		Bit3 User Output 1 (Relais, 230 V, max. 100 W)
		Bit4 User Output 2 (Relais, nicht bestückt)
		Bit5 User Output 3 (Opto)
		Bit6 User Output 4 (Opto)
		Bit7 reserviert
1	0	Haube darf nicht geöffnet werden
	1	Haube darf geöffnet werden
2	0	Spindel ausschalten
	1	Spindel einschalten
3	0	Motorströme ausschalten
	1	Motorströme einschalten

Beschränkung: Ein Überschreiben der Portwerte erfolgt nur, wenn die Steuerung mit dem Software-Handshake „0“ antwortet. Der Befehl kann nur verwendet werden, wenn keine Bewegung abgearbeitet wird. Die reservierten Bits des Port 0 können vom Anwender nicht überschrieben werden.

Programmierbeispiel: siehe Anhang B 3.10. Port schreiben

2.2.15 Test-Modus Ein-/Ausschalten

Befehl: Test-Modus Ein-/Ausschalten

Zweck: Durch Verwenden des Befehles kann der Test-Modus gezielt aus- und eingeschaltet werden.

Aufbau: @<GN>T<Status><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
T	= Befehl Test-Modus Ein-/Ausschalten
<Status>	= 0 --> Ausschalten, 1 --> Einschalten
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0T1, @0T0

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „T1“ schaltet den Test-Modus ein „T0“ schaltet den Test-Modus aus. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter („0“). Im Test-Modus behandelt die Steuerung die Referenzfahrt und die Endschalter anders als im normalen Betrieb. Wenn im Test-Modus ein Befehl Referenzfahrt empfangen wird, führt die Steuerung keine Referenzfahrt im eigentlichen Sinne aus sondern setzt den aktuellen Punkt als Referenzpunkt. Die Endschalter werden weiterhin überwacht können aber überfahren werden. Dies ist sehr nützlich wenn eine Achse nach dem Einschalten der Anlage in einem Endschalter steht und freigefahren werden muss.

Beschränkung: Der Befehl kann nur verwendet werden, wenn keine Bewegung abgearbeitet wird.

Programmierbeispiel: -

2.2.16 Abfragen der Statusdaten der Steuerung

Befehl: Statusdaten abfragen

Zweck: Abfrage wichtiger Statusdaten der Steuerung zur Darstellung des aktuellen Zustandes und zur Fehlersuche und -diagnose.

Aufbau: @<GN>H<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
H	= Befehl Statusdaten abfragen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0H

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „H“ veranlasst die Steuerung Informationen über den aktuellen Status im Klartextformat zurückzusenden. Am Ende dieser Informationen antwortet die Steuerung mit dem Handshake-Charakter („0“). Die Informationen werden im ASCII-Format bereits zeilenweise formatiert ausgegeben, so dass sie z.B. in einem Terminalfenster direkt auf dem Bildschirm eines Steuerrechners dargestellt werden können. Diese Informationen umfassen den Zustand der Endschalter und der Bedienelemente einer Anlage.

Beschränkung: Für den Aufruf der Funktion muss ein genügend großer Empfangspuffer (mindestens 512 Byte) auf Seite des Steuerrechners zur Verfügung stehen, damit keine Informationen verloren gehen.

Programmierbeispiel: -

2.2.17 Abfragen der Versionsdaten der Steuerung

Befehl: Versionsdaten abfragen

Zweck: Abfrage wichtiger Versionsdaten der Steuerung.

Aufbau: @<GN>V<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
V	= Befehl Versionsdaten abfragen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0V

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „V“ veranlasst die Steuerung Informationen über Version der Steuerung im Klartextformat zurückzusenden. Am Ende dieser Informationen antwortet die Steuerung mit dem Handshake-Charakter („0“). Die Informationen werden im ASCII-Format bereits zeilenweise formatiert ausgegeben, so dass sie z.B. in einem Terminalfenster direkt auf dem Bildschirm eines Steuerrechners dargestellt werden können.

Beschränkung: Für den Aufruf der Funktion muss ein genügend großer Empfangspuffer (mindestens 512 Byte) auf Seite des Steuerrechners zur Verfügung stehen, damit keine Informationen verloren gehen.

Programmierbeispiel: -

2.2.18 Initialisierung von Parametern

Befehl: Parameter initialisieren

Zweck: Initialisierung von Achs- und Referenzrichtungen.

Aufbau: @<GN>I<Code><Wert><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
I	= Befehl Initialisierung
<Code>	= ASCII-Zeichen zu Unterscheidung verschiedener Parameter s.u.
<Wert>	= neuer Wert für den Parameter
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0ID3, @0IR1, @0IS1

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „I“ teilt der Steuerung mit, dass eine Initialisierung vorgenommen werden soll. Anschließend folgen eine Kennung für den Parameter sowie der neue Wert und Carriage Return als Befehlsende. Folgende Parameter können initialisiert werden:

Code	Wert	Funktion
D	1	Richtung X-Achse negiert
	2	Richtung Y-Achse negiert
	4	Richtung Z-Achse negiert
	8	Richtung A-Achse negiert
		(durch Addition der Werte sind alle Kombinationen möglich)
R	1	Referenzrichtung X-Achse negiert
	2	Referenzrichtung Y-Achse negiert
	4	Referenzrichtung Z-Achse negiert
	8	Referenzrichtung A-Achse negiert
		(durch Addition der Werte sind alle Kombinationen möglich)
S	0-255	Setzen des internen Hardwarestatusbytes
		Bitbelegung: Bit0: Enable/Disable Haubenöffnung
		Bit1-7: nicht benutzt

Beschränkung: Das interne Hardwarestatusbyte sollte vom Anwender nicht benutzt werden und ist hier nur der Vollständigkeit halber aufgeführt. Die Benutzung bleibt entsprechenden Treibern der Fa. iselautomation vorbehalten.

Programmierbeispiel: -

2.2.19 Diagnose

Befehl: Diagnose

Zweck: Abfragen von Diagnosedaten der Steuerung.

Aufbau: @<GN>D<Code1><Code2><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
D	= Befehl Diagnose
<Code1>	= ASCII-Zeichen zu Unterscheidung verschiedener Parameter s.u.
<Code2>	= ASCII-Zeichen zu Unterscheidung verschiedener Parameter s.u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0DRp, @0DRn, @0DS0

Erläuterung: Die Steuerung wird mit dem Dateneröffnungsteil „@0“ auf einen neuen Befehl vorbereitet. „D“ teilt der Steuerung mit, dass eine Diagnose vorgenommen werden soll. Anschließend folgen zwei Kennungen für den Parameter und Carriage Return als Befehlsende. Die Steuerung antwortet mit dem Software-Handshake „0“ gefolgt von zwei Zeichen, die einen Hexadezimalwert angeben, der dem aktuellen Wert des Parameters entspricht. Folgende Parameter können abgefragt werden:

Code1	Code2	Funktion
R	p	positive Endschalter abfragen
R	n	negative Endschalter abfragen
P	0	Eingangsport 0 abfragen
P	1	Eingangsport 1 abfragen
O	0	Ausgangsport 0 abfragen
S	0	Hardwarestatuseingangsport abfragen

Bitbelegung Endschalter:

Bit0:	X-Achse
Bit1:	Y-Achse
Bit2:	Z-Achse
Bit3:	A-Achse

Bitbelegung Hardwarestatus:

Bit0:	Status Spindel
Bit1:	Status Starttaste
Bit2:	Status Stoptaste
Bit3:	Status Not-Aus
Bit4:	Status Schlüsselschalter
Bit5:	Status Haube
Bit6:	nicht benutzt
Bit7:	Status Endstufen/Stromversorgung

Beschränkung: Die Diagnosefunktionen sollten vom Anwender nicht benutzt werden und sind hier nur der Vollständigkeit halber aufgeführt. Die Benutzung bleibt entsprechenden Treibern der Fa. ***isel****automation* vorbehalten.

Programmierbeispiel: -

2.2.20 Selbsttest durchführen

Befehl: Selbsttest

Zweck: Diese Anweisung löst einen Selbsttest der Steuerung aus. Diese Test umfasst die Bewegung der Achsen sowie den Schnittstellentest und die Ausgabe von Versionsinformationen.

Aufbau: @<GN>?<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
?	= Befehl Selbsttest
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0?

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „?“ gibt an, dass ein Selbsttest durchgeführt werden soll. Der Befehl wird mit Carriage Return abgeschlossen. Die Steuerung gibt anschließend Versionsinformationen aus, testet die Bewegung der Motoren und führt einen Schnittstellentest aus. Zum Testen der Schnittstelle gibt die Steuerung zunächst den ASCII-Zeichensatz aus. Wird über die Schnittstelle ein Zeichen empfangen so wechselt die Steuerung in den Echo-Modus und gibt alle empfangenen Zeichen an den Steuerrechner zurück.

Beschränkung: Für den Aufruf der Funktion muss ein genügend großer Empfangspuffer auf Seite des Steuerrechners zur Verfügung stehen, damit keine Informationen verloren gehen. Der Aufruf dieser Funktion ist nur innerhalb eines Terminalprogramms oder einer Terminalfunktion sinnvoll. Der Selbsttest kann nur durch Ausschalten der Steuerung oder Software-Reset (char(254)) beendet werden. Anschließend ist eine Neuinitialisierung der Anlage notwendig.

Programmierbeispiel: -

Der Selbsttest kann auch ausgelöst werden, wenn beim Einschalten der Steuerung die Starttaste betätigt und erst nach Beginn des Selbsttest losgelassen wird.

2.2.21 Kontroll- und Steuercodes

Kontroll- und Steuercodes ermöglichen den direkten Eingriff in den Funktionsablauf der Steuerung über die serielle Schnittstelle. Dabei werden die jeweils gesendeten Kommandos ohne Verzögerung direkt in der Empfangsroutine der Steuerung ausgewertet und anschließend ausgeführt. Für folgende Funktionalitäten stehen spezielle Steuercodes zu Verfügung:

Funktion: Softwarestop char(253)

Eine Positionierbewegung im DNC-Modus (relativ oder absolut), kann durch einen Stopbefehl angehalten werden, ohne das Schrittverluste auftreten. Ein danach ausgeführter Startbefehl (durch Übergabe von „@0S“, siehe 2.2.12 Starten einer angehaltenen Bewegung) beendet den unterbrochenen Funktionsablauf. Außerdem kann nach einem Stopbefehl mit Hilfe des Befehles „Positionsabfrage“ die aktuelle erreichte Position rückgelesen werden. Diese Funktionalität kann auch durch Betätigen des Stoptasters erreicht werden. Wurde eine Bewegung erfolgreich angehalten erzeugt die Steuerung eine zusätzliche Rückmeldung „F“.

Die Funktion wird durch Übergabe eines char(253) über die serielle RS232-Schnittstelle aufgerufen.

Funktion: Software-Reset char(254)

Die Steuerung unterbricht sofort alle Aktivitäten und führt intern einen Software-Reset durch. Anschließend muss die Anlage wieder neu initialisiert und eine Referenzfahrt durchgeführt werden.

Die Funktion wird durch Übergabe eines char(254) über die serielle RS232-Schnittstelle aufgerufen.

Funktion: Software-Break char(255)

Eine Positionierbewegung im DNC-Modus (relativ oder absolut), kann durch einen Breakbefehlbeendet werden. Dies bedeutet, dass der Rest der Bewegung vergessen wird.

Die Funktion wird durch Übergabe eines char(255) über die serielle RS232-Schnittstelle aufgerufen.

siehe auch: 4 Die Fehlermeldungen der IMC4

2.3 Die Berechnung der Kreisparameter

2.3.1 Die Parameter für die Kreisinterpolation

Die Kreisinterpolation wird durch zwei aufeinanderfolgende Befehle (s. 2.2.10, 2.2.11) ausgelöst. Der erste Befehl legt die Kreisrichtung fest, im Zweiten werden die Interpolationsparameter übergeben.

Kreisrichtung: @<GN>f<Richtung><CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
f	= Befehl Kreisrichtung einstellen
<Richtung>	= 0 --> CW (Uhrzeigersinn), -1 --> CCW (gegen Uhrzeigersinn)
<CR>	= Carriage Return als Befehlsabschluss

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

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
<B>	= Bogenlänge in Schritten
<V>	= Geschwindigkeit
<D>	= Interpolationsparameter
<Xs>	= Startpunkt x
<Ys>	= Startpunkt y
<Rx>	= Richtung x
<Ry>	= Richtung y
<CR>	= Carriage Return als Befehlsabschluss

Steuerungsintern wird ein speziell angepasster Differenzalgorithmus nach Bresenham benutzt um Kreisbögen zu erzeugen. Diese Art Algorithmus wird sehr oft in Mikroprozessoranwendungen benutzt, da man hier mit geringem Rechenaufwand hohe Ausführungsgeschwindigkeiten erreicht.

Hier soll nun die Bedeutung und Berechnung der Parameter für die Kreisinterpolation erläutert werden. Ein entsprechendes Beispiel finden Sie unter Punkt 2.3.6. dieser Beschreibung.

2.3.2 Die Berechnung der Bogenlänge

Die Bogenlänge <B> gibt die Länge des Kreisbogens in Schritten zwischen dem Startpunkt und dem Endpunkt des Bogens an und wird steuerungsintern als Laufvariable für den Differenzalgorithmus benutzt. Die Berechnung der Bogenlänge in Schritten kann auf unterschiedlichen Wegen erfolgen und soll im Folgenden erläutert werden.

- Einfache Näherungsformel
  Für einfache Kreisanwendungen, die z.B. nur Viertel-, Halb-, oder Vollkreise enthalten kann die Bogenlänge in Schritte mit folgender Formel berechnet werden:

  B - Bogelänge in Schritten
  R - Radius des Bogens in Schritten
  A - Anfangswinkel im Bogenmaß
  E - Endwinkel im Bogenmaß

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

Das Ergebnis ist auf den nächsten ganz zahligen Wert zu runden. Um für die Abarbeitung auf der Steuerung die Ungenauigkeit zu eliminieren sollte die nächste Positionierung als absolute Bewegung programmiert werden.

- Berechnung mittels Softwareroutine

Eine genaue Berechnung der Bogenlänge in Schritten kann mittels einfacher Softwareroutinen erreicht werden. Beispiele dazu finden Sie in den C- und Pascal-Routinen im Anhang (siehe A 1, A 2) dieser Beschreibung.

2.3.3 Die Berechnung des Interpolationsparameters

Der Interpolationsparameter dient der Steuerung als Startwert für das Differenzregister des von der Steuerung verwendeten Algorithmus zur Kreiserzeugung. Die Berechnung des Parameters erfolgt auf PC-Seite mittels entsprechender Softwareroutinen. Dies entlastet die Steuerung von unnötigem Rechenaufwand und erhöht somit die Abarbeitungsgeschwindigkeit. Beispielroutinen zur Berechnung des Parameters finden Sie im Anhang (siehe A 1. und A 2.) dieser Beschreibung.

2.3.4 Startpunkt des Kreisbogens

Der Startpunkt des Kreisbogens stellt den Abstand in X und Y in Schritten vom Kreismittelpunkt in relativen Koordinaten dar (d.h. der Kreismittelpunkt wird für die Berechnung als gedachter Nullpunkt angenommen). Die Berechnung kann mittels der entsprechenden Kreisfunktionen erfolgen.

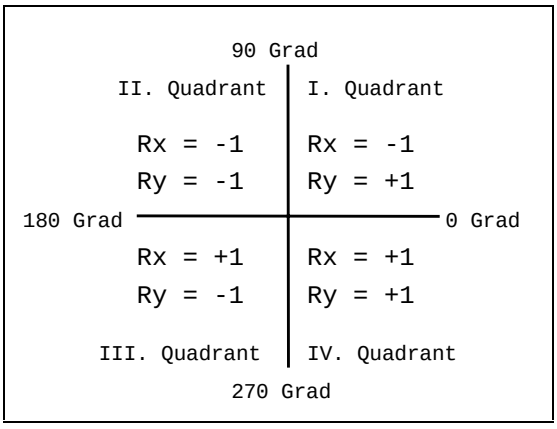
Xs - X-Koordinate des Startpunktes relativ zum Mittelpunkt
Ys - Y Koordinate des Startpunktes relativ zum Mittelpunkt
R - Radius in Schritten
A - Anfangswinkel im Bogenmaß

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

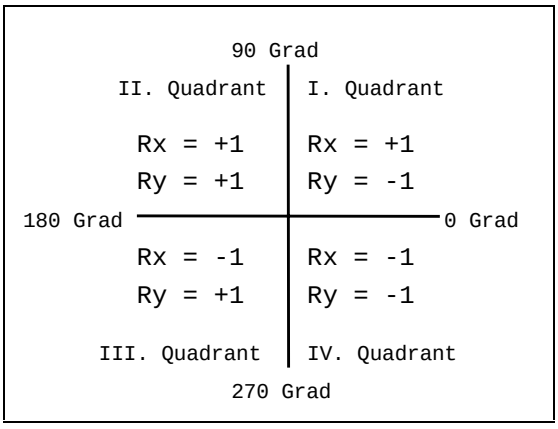
2.3.5. Richtungen im Startpunkt der Kreisinterpolation

Zur Ausführung des Interpolationsalgorithmus benötigt die Steuerung eine Information darüber, in welchem Quadranten der Bogen beginnt und welche Vorzeichen steuerungsintern für bestimmte Berechnungen benutzt werden sollen. Diese Informationen werden der Steuerung in Form der Parameter Rx und Ry zur Verfügung gestellt. Dabei gelten folgende Festlegungen:

Kreisbögen entgegen dem Uhrzeigersinn (CCW)



Kreisbögen im Uhrzeigersinn (CW)



2.3.6 Berechnungsbeispiel zur Kreisinterpolation

Zur Vertiefung der Berechnung der Parameter des Kreisbefehls soll hier ein kurzes Beispiel berechnet werden.

Es soll ein Kreisbogen entgegen dem Uhrzeigersinn mit dem Radius von 200 Schritten mit einer Geschwindigkeit von 1500 Schritten/s verfahren werden. Der Anfangswinkel sei 135 Grad, der Endwinkel 225 Grad. Beachten Sie bitte, dass für die Berechnung alle Wegangaben in Schritten und alle Winkelangaben im Bogenmaß zur Verfügung stehen müssen.

geg:	Radius	R = 200	ges:	Bogenlänge	B
	Startwinkel	A = 135*Pi/180 = 2.3562		Startpunkt X	Xs
	Endwinkel	E = 225*Pi/180 = 3.9267		Startpunkt Y	Ys
	Speed	V = 1500		Richtung X	Rx
	Richtung	CCW		Richtung Y	Ry
				Interpolationsparameter	D

Bogenlänge B (siehe 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

Startpunkt Xs und Ys (siehe 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

Richtung Rx und Ry (siehe 2.3.5):

Startwinkel 135 Grad Drehrichtung CCW
Rx = -1 Ry = -1

Interpolationsparameter D (siehe 2.3.3.):

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

Summe(R-1) = Summe(199) = 199*(199+1) = 39800

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

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

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

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

Die Befehle würden lauten:

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

siehe auch: 2.2.10 Einstellen der Kreisrichtung für die Kreisinterpolation, 2.2.11 Kreisinterpolation, Anhang A 1 Software-Routinen zur Berechnung der Parameter beim Kreisbefehl in Turbo-Pascal, Anhang A 2 Software-Routinen zur Berechnung der Parameter beim Kreisbefehl in C

3 Der CNC-Modus und seine Befehle

3.1 Befehlsaufbau der CNC-Befehle

Im CNC-Modus betrieben, speichert die Steuerung alle übersendeten Befehle im internen Datenspeicher. Zur Aktivierung ist nach der Standard-Initialisierung der Befehl „CNC-Datenfeld speichern“ zu übertragen. Anschließend wird das Datenfeld übergeben und mit dem Befehl „Datenfeld-Ende“ abgeschlossen.

Das Programm kann nun ohne weitere Kommunikation mit dem Steuerrechner durch einen externen Start-Befehl (Betätigen der Starttaste) aktiviert werden.

Als Speichermedium kommen auf der IMC4 FlashPROM’s (nichtflüchtige elektrisch schreib- und löschbare Speicher) zum Einsatz. Diese Speicher werden, ähnlich einem EPROM, im System durch bestimmte Programmierzyklen mit den entsprechenden Informationen beschrieben. Das Löschen entspricht dem Beschreiben mit einem Default-Wert. Die notwendigen Zyklen werden dabei selbständig von den Speicherschaltkreisen ausgeführt. Während der Ausführung dieser Zyklen kann nicht auf die Schalkreise zugegriffen werden, so dass das Programmieren und Löschen relativ zeitaufwendig ist. Vor einem Wiederbeschreiben der Chips müssen diese prinzipiell gelöscht werden.

Dazu muss ein Löschzyklus wie folgt durch die IMC4 ausgeführt werden:

- NOT-AUS betätigen und wieder entriegeln
- Schlüsselschalter in EIN-Stellung schalten
- (! Der Schlüssel lässt sich in dieser Stellung nicht abziehen)
- STOP-Taster betätigen und im betätigtem Zustand halten
- Jetzt bei betätigtem STOP-Taster den EIN-Taster betätigen
- Nach 20 Sekunden ist der maschineninterne CNC-Speicher gelöscht
- Maschine ausschalten

Nach dem Löschen können die Speicher mit einem neuen Programm überschrieben werden. Ist auf den Speichern bereits ein Programm oder ein Teil eines Programmes abgelegt führt der Befehl „CNC-Datenfeld speichern“ zu einer Fehlermeldung.

Im Folgenden werden die speicherbaren Befehle der Steuerung IMC4 aufgelistet und kurz erläutert. Eine Detailerklärung kann für einige Befehle unter dem entsprechendem Befehl des DNC-Modus nachgeschlagen werden, da die Bedeutung und Anzahl der Parameter oft denen des DNC-Modus entsprechen.

Ist während der Übertragung und Speicherung eines CNC-Datenfeldes ein Fehler aufgetreten wird das bis dahin abgespeicherte CNC-Programm als ungültig markiert und kann nicht abgearbeitet werden. Der Fehler im Programm muss dann entsprechend beseitigt und das FlashPROM gelöscht werden, ehe das Datenfeld erneut zu Abspeicherung übertragen werden kann.

3.2 Die Befehle der IMC4 im CNC-Modus

3.2.1 CNC-Datenfeld speichern

Befehl: CNC-Datenfeld speichern

Zweck: Diese Anweisung dient als Initialisierung für die Übertragung von speicherbaren Befehlen und ist zu Beginn des CNC-Modus zwingend erforderlich.

Aufbau: @<GN>i<CR>

@	= Dateneröffnungszeichen
<GN>	= Gerätenummer, Standard = 0
i	= Befehl CNC-Datenfeld speichern
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: @0i

Erläuterung: Die Steuerung wird durch „@0“ adressiert. „i“ gibt an, dass ein CNC-Datenfeld gespeichert werden soll. Der Befehl wird mit Carriage Return abgeschlossen. Die Steuerung akzeptiert anschließend bis zum Befehl „Datenfeld-Ende“ oder bis zum Auftreten eines Fehlers nur noch CNC-Befehle. Der Befehl wird mit einer entsprechenden Rückmeldung quittiert. Alle nachfolgenden speicherbaren Befehle werden im FlashPROM abgespeichert.

Beschränkung: Der Befehl kann nur verwendet werden, wenn die Steuerung vorher initialisiert wurde und keine Bewegung abgearbeitet wird. Ist bereits ein Programm auf den Speichern abgelegt, so führt dies zu einer Fehlermeldung.

Programmierbeispiel: siehe Anhang D

3.2.2 Referenzfahrt im CNC-Modus

Befehl: Referenzfahrt

Zweck: Die Steuerung speichert eine Bewegung aller angegebenen Achsen an ihre Nullpunkte (Referenzpunkte). Die Referenzpunkte der Achsen sind bei *isel*-Systemen immer in einer sinnvollen Defaultanordnung festgelegt, können aber durch entsprechende Initialisierungsbefehle angepasst werden.

Aufbau: 7<Achsen><CR>

7	= Befehlscode Referenzfahrt
<Achsen>	= Achsenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: 77, 78

Erläuterung: „7“ gibt an, dass eine Referenzfahrt ausgeführt werden soll. Der anschließende Zahlenwert definiert die Achsen, die eine Referenzfahrt durchführen sollen. Dabei wird intern jede Achse durch ein Bit eines Binärwertes repräsentiert und somit ergeben sich folgende Werte:

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

Die Reihenfolge der Ausführung ist dabei wie folgt festgelegt:
Z-Achse --> Y-Achse --> X-Achse --> A-Achse

Nach erfolgter Referenzfahrt wird der nächste CNC-Befehl aus dem Speicher gelesen und abgearbeitet.

Beschränkung: Der Befehl beschränkt sich auf die initialisierte Achsenkonfiguration. Bei falscher Achsangabe erfolgt die Fehlerrückmeldung „3“. Befindet sich die Steuerung im 3D-Modus, schaltet der Befehl wieder in die 2,5-dimensionale Ausführung zurück.

ACHTUNG: Die A-Achse muss immer separat referenziert werden.

ACHTUNG: Bei nicht angeschlossenem Referenzschalter, wird die entsprechende Achse permanent angesteuert. Durch Betätigen des Stop-Tasters besteht jedoch die Möglichkeit die Referenzfahrt zu beenden.

Programmierbeispiel: siehe Anhang D

3.2.3 Relative Bewegung im CNC-Modus

Befehl: Bewegung relativ

Zweck: Die Steuerung speichert entsprechend der übergebenen Schrittanzahl und Schrittgeschwindigkeit eine relative Bewegung.

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

0	= Befehlscode Bewegung relativ
<Sx>	= Schrittanzahl x
<Gx>	= Geschwindigkeit
<Sy>	= Schrittanzahl y
<Gy>	= Geschwindigkeit
<Sz>, <Sz1>	= Schrittanzahl z
<Sz2>	= Schrittanzahl z, 2. Bewegung bei 2.5d, 3 Achsen
<Gz>, <Gz1>	= Geschwindigkeit
<Gz2>	= Geschwindigkeit z, 2. Bewegung bei 2.5d, 3 Achsen
<Sa>	= Schrittanzahl a, bei 4 Achsen
<Ga>	= Geschwindigkeit, bei 4 Achsen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:

05000,900	(nur x-Achse)
050,900,20,9000	(x und y-Achse)
030,800,10,900,4,90,-4,30	(x,y und z-Achse, bei 3 Achsen)
030,800,10,900,4,90,-4,30	(x,y,z und a-Achse, bei 4 Achsen)

Erläuterung: „0“ gibt an, dass eine relative Bewegung erfolgen soll. Die Steuerung erwartet nun für jede Achse ein Zahlenpaar bestehend aus Schrittanzahl und Geschwindigkeit. Die Angabe der Entfernungen erfolgt im Relativmaß, d.h. bezogen auf die letzte Position. Die Anzahl der Angaben muss mit der Achsenzahl übereinstimmen, d.h. ein Parameterpaar bei x-Betrieb, zwei Parameterpaare bei xy-Betrieb, vier Parameterpaare für xyz-Betrieb und vier Parameterpaare für xyza-Betrieb. Die einzelnen Zahlen müssen durch Kommata getrennt werden. Für die z-Achse werden bei 3 Achsen und 2,5D zwei Zahlenpaare erwartet, da für Bearbeitungsanwendungen sehr häufig die Situation „Fahren, Werkzeug absenken und anschließend anheben“ vorkommt. Im 2.5D-Interpolationsbetrieb werden zuerst die Bewegungen der x- und y-Achse verfahren (linear interpoliert), anschließend wird die z-Achse zuerst um die in z1-angegebenen und dann um die in z2-angegebenen Werte verfahren. Besteht die Notwendigkeit nur eine Achse zu bewegen, so sind dennoch für alle initialisierten Achsen Werte zu übertragen. Dabei ist für die Schrittzahlen der nicht bewegten Achsen 0 anzugeben. Die Steuerung meldet sich nach erfolgter Speicherung mit dem Handshake-Charakter („0“).

Beschränkung: Die Steuerung prüft nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Im 2.5D-Interpolationsbetrieb wird die Geschwindigkeitsangabe der Achse mit dem längsten Weg als Bahngeschwindigkeit übernommen und die Geschwindigkeit der anderen Achsen entsprechend dem Wegeverhältnis angepasst. Dem gegenüber wird im 3D-Interpolationsbetrieb die Geschwindigkeitsangabe der x-Achse als Vorgabewert für die Bahngeschwindigkeit herangezogen.

Programmierbeispiel: siehe Anhang D

3.2.4 Absolute Bewegung im CNC-Modus

Befehl: Bewegung zur absoluten Position

Zweck: Die Steuerung speichert entsprechend den angegebenen Geschwindigkeiten und Positionen eine absolute Bewegung.

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

m	= Befehlscode Bewegung absolut
<Sx>	= Position x
<Gx>	= Geschwindigkeit
<Sy>	= Position y
<Gy>	= Geschwindigkeit
<Sz>, <Sz1>	= Position z
<Sz2>	= Position z, 2. Bewegung immer = 0
<Gz>, <Gz1>	= Geschwindigkeit
<Gz2>	= Geschwindigkeit
<Sa>	= Position a, bei 4 Achsen
<Ga>	= Geschwindigkeit, bei 4 Achsen
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:

m5000,900	(nur x-Achse)
m50,900,20,9000	(x und y-Achse)
m30,800,10,900,4,90,0,30	(x,y und z-Achse, bei 3 Achsen)
m30,800,10,900,4,90,4,30	(x,y,z und a-Achse, bei 4 Achsen)

Erläuterung: „m“ gibt an, dass eine Absolut-Position folgt. Aus Kompatibilitätsgründen zum relativen Positionierbefehl werden auch hier bei 3 Achsen für die z-Achse zwei Zahlenpaare erwartet. Die zweite Positionsangabe der z-Position muss dann jedoch Null sein und wird ignoriert. Die Steuerung meldet sich nach erfolgter Speicherung mit dem Handshake-Charakter.

Beschränkung: Die Steuerung prüft nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Programmierbeispiel: siehe Anhang D

3.2.5 Nullpunkt setzen im CNC-Modus

Befehl: Nullpunkt am aktuellen Punkt setzen

Zweck: Die Steuerung speichert einen Befehl, um die momentane Position während der Abarbeitung des CNC-Programms als virtuellen Nullpunkt für die angegebene(n) Achse(n) zu setzen. Die nachfolgenden „Verfahre absolut“-Anweisungen beziehen sich dann auf diesen virtuellen Nullpunkt.

Aufbau: n<Achsen><CR>

n	= Befehlscode Nullpunkt setzen
<Achsen>	= Achsenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: n7, n1, n8

Erläuterung: „n“ gibt an, dass eine Nullpunktverschiebung vorgenommen werden soll. Nach dem Befehlscode werden der Steuerung die Achsen mitgeteilt, für die eine Nullpunktverschiebung durchgeführt werden soll. Dabei wird intern jede Achse durch ein Bit eines Binärwertes repräsentiert und somit ergeben sich folgende Werte:

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

Die Steuerung meldet sich nach erfolgter Speicherung mit einer Rückmeldung.

Beschränkung: Der virtuelle Nullpunkt hat nur für den Befehl „Verfahre absolut“ eine Bedeutung. Relativpositionierung wird vom virtuellen Nullpunkt nicht beeinflusst, da hier ein relativer Verfahrvektor angegeben wird.

ACHTUNG: Die Nullpunktverschiebung für die A-Achse muss immer separat durchgeführt werden.

Programmierbeispiel: siehe Anhang D

3.2.6 3D-Interpolation Ein-/Ausschalten im CNC-Modus

Befehl: 3D-Linear-Interpolation Ein-/Ausschalten

Zweck: Die Steuerung speichert den Befehl, um die 2.5D-Interpolation des Betriebssystemes auf eine 3-dimensionale Interpolation zu erweitern. Durch Verwenden des Befehles kann diese Interpolation gezielt aus- und eingeschaltet werden.

Aufbau: z<Status><CR>

z	= Befehlscode 3D-Interpolation
<Status>	= 0 --> Ausschalten, 1 --> Einschalten
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: z1, z0

Erläuterung: „z1“ ändert die Interpolation von 2D- auf 3D-Betrieb. Die Anweisung wirkt modal, d. h. alle relativen und absoluten Bewegungen werden dreidimensional ausgeführt.
Die Angabe von z2-Parametern bei 3 Achsen in diesen Verfahrbewegungen wird ignoriert. Die Geschwindigkeitsangabe der Interpolation muss bei der x-Angabe erfolgen. Bei 4 Achsen wird die 4. Achse entsprechend nachgeführt.

Programmierbeispiel: siehe Anhang D

3.2.7 Ebenenwahl für Kreisinterpolation im CNC-Modus

Befehl: Ebenenwahl

Zweck: Speichern der Interpolationsebene für die Kreisinterpolation. Kreise sind nur innerhalb einer Ebene definiert. Die Defaultebene für die Kreisinterpolation ist die XY-Ebene. Durch den „Ebenenwahl“-Befehl besteht hier jedoch die Möglichkeit jede andere Ebenenkonfiguration als Kreisebene zu definieren.

Aufbau: e<Ebene><CR>

e	= Befehlscode Kreisebene setzen
<Ebene>	= Ebenenangabe, s. u.
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: e1, e0

Erläuterung: „e“ gibt an, dass die Ebene für die Kreisinterpolation eingestellt werden soll. Der anschließende Zahlenwert definiert die Ebene in folgender Weise:

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

Beschränkung: Dieser Befehl ist modal wirksam, d.h. eine Ebenenwahl für die Kreisinterpolation bleibt erhalten bis sie von einer erneuten Ebenenwahl überschrieben wird.

Programmierbeispiel: siehe Anhang D

3.2.8 Einstellen der Kreisrichtung für die Kreisinterpolation im CNC-Modus

Befehl: Kreisrichtung einstellen

Zweck: Speichern der Kreisrichtung für die Kreisinterpolation. Die Kreisinterpolation wird durch zwei aufeinanderfolgende Befehle programmiert. Der erste Befehl legt die Kreisrichtung fest, im Zweiten (s. 3.2.9.) werden die Interpolationsparameter übergeben.

Aufbau: f<Richtung><CR>

f	= Befehlscode Kreisrichtung einstellen
<Richtung>	= 0 --> CW (Uhrzeigersinn), -1 --> CCW (gegen Uhrzeigersinn)
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: f-1, f0

Erläuterung: „f“ gibt an, dass die Richtung für die Kreisinterpolation eingestellt werden soll. Der anschließende Zahlenwert definiert die Richtung in folgender Weise:

0 --> CW (Kreisinterpolation im Uhrzeigersinn)
-1 --> CCW (Kreisinterpolation entgegen dem Uhrzeigersinn)

Beschränkung: Die Richtung für die Kreisinterpolation ist prinzipiell vor jeder Kreisbewegung zu programmieren.

Programmierbeispiel: siehe Anhang D

3.2.9 Kreisinterpolation im CNC-Modus

Befehl: Kreisinterpolation

Zweck: Speichern von Bewegungsbefehlen für Kreise und Kreisbögen mit konstanter Bahngeschwindigkeit. Die Kreisinterpolation wird durch zwei aufeinanderfolgende Befehle ausgelöst. Der erste Befehl legt die Kreisrichtung fest (s. 3.2.8.), im Zweiten werden die Interpolationsparameter übergeben.

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

y	= Befehlscode Kreisinterpolation
<B>	= Bogenlänge in Schritten
<V>	= Geschwindigkeit
<D>	= Interpolationsparameter
<Xs>	= Startpunkt x
<Ys>	= Startpunkt y
<Rx>	= Richtung x
<Ry>	= Richtung y
<CR>	= Carriage Return als Befehlsabschluss

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

Erläuterung: „y“ gibt an, dass eine Kreisinterpolation gespeichert werden soll. Die Bogenlänge gibt die Länge des Bogens in Schritten zwischen dem Start- und Endpunkt der Kreisinterpolation an. Als Geschwindigkeitsangaben sind alle ganz zahligen Werte innerhalb des gültigen Wertebereiches für Geschwindigkeiten zulässig. Der Interpolationsparameter muss übergeben werden, da die Steuerung aufgrund ihrer Speicherkapazität nicht in der Lage ist diesen Parameter selbst zu berechnen. Die Parameter Xs und Ys geben den Startpunkt des Bogens relativ zum Kreismittelpunkt an.Rx und Ry geben an, in welchem Quadranten des Kreises die Interpolation startet. Die Steuerung meldet sich nach erfolgter Ausführung mit dem Handshake-Charakter („0“).

ACHTUNG: Zur Berechnung der Parameter lesen Sie bitte das Kapitel „Berechnung der Parameter für die Kreisinterpolation“.

Beschränkung: Die Steuerung prüft nicht, ob die Bewegung den zulässigen Bereich der angeschlossenen Mechanik verlässt.

Programmierbeispiel: siehe Anhang D

3.2.10 Schleifen, Verzweigungen im CNC-Modus

Befehl: Schleife, Verzweigung

Zweck: Speichern von Schleifen und Verzweigungen. Schleifen dienen dazu, gleichartige Bewegungsabläufe zusammenzufassen. Hierdurch wird der zur Verfügung stehende Speicherplatz der Steuerung besser genutzt. Durch Verzweigungen kann nach einer logischen Entscheidung zu einem bestimmen Satz innerhalb des Programms gesprungen werden.

Aufbau: 3<Anzahl>,<Offset><CR>

3	= Befehlscode Schleife, Verzweigung	
<Anzahl>	= Schleifenzahl	
	Schleife:	0 < Schleifenzahl < 32768
	Verzweigung:	immer 0
<Offset>	= Sprungziel	
	Schleife:	-1 >= Sprungziel >= -32768
	Verzweigung:	-32768 <= Sprungziel <= 32767
<CR>	= Carriage Return als Befehlsabschluss	

Anwendung:	3 25,-1	Wiederhole letzten Befehl 25 mal
	3 0,-5	Verzweige immer 5 Schritte zurück
	3 0,5	Überspringe die nächsten 4 Befehle
	3 6,-5	Wiederhole die letzten 5 Befehle 6 mal

Erläuterung: Trifft die Steuerung innerhalb des CNC-Programmablaufes auf den Befehl „Schleife/Verzweigung“ wird zunächst durch Prüfen der Schleifenzahl entschieden, ob es sich um einen Schleifen- oder um einen Verzweigungsbefehl handelt. Bei einem Schleifenbefehl, wird ein Schleifenzähler eingerichtet, vorbesetzt und der Befehlszähler um den angegebenen Offset korrigiert. Die Befehle bis zum Schleifenzähler werden nun jeweils wiederholt und der Schleifenzähler dekrementiert, bis dieser Null erreicht hat. Anschließend wird mit der Ausführung des ersten Befehls nach der Schleife fortgefahren. Schleifen können mit einer Schachtelungstiefe von 15 ineinander verschachtelt sein. Die notwendigen Zähler werden dann auf einem entsprechenden Schleifen-Stack verwaltet. Bei einer Verzweigung wird der Offset als relatives Sprungziel innerhalb des CNC-Programmes verstanden und der Befehlszähler entsprechend um den Offset korrigiert.

Beschränkung: Es darf nicht vor den Anfang oder hinter das Ende des Datenfeldes verzweigt werden. Vorwärtsschleifen sind unzulässig. Eine Schleife wiederholt immer die letzten n-Befehle. Es muss mindestens ein Befehl wiederholt werden. Schleifen dürfen geschachtelt sein, die maximale Schachtelungstiefe beträgt 15. Eine Schleife darf nicht durch eine Verzweigung verlassen werden.

Programmierbeispiel: siehe Anhang D

3.2.11 Zeitverzögerungen im CNC-Modus

Befehl: Zeitverzögerung

Zweck: Speichern von Zeitverzögerungen.

Aufbau: 5<Zeit><CR>

5	= Befehlscode Zeitverzögerung
<Zeit>	= Zeit in 1/10 sec
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: 3 50 Verzögerung 5 Sekunden

Erläuterung: Trifft die Steuerung innerhalb des CNC-Programmablaufes auf den Befehl „Zeitverzögerung“. So erfolgt die Ausführung des nächste Befehls im CNC-Programm erst nach Ablauf der Verzögerungszeit. Die Zeitangabe erfolgt dabei in 1/10 Sekunden.

Beschränkung: Eine Zeitverzögerung kann nicht durch Betätigen des Stop-Tasters der Steuerung abgebrochen werden.

Programmierbeispiel: siehe Anhang D

3.2.12 Port setzen im CNC-Modus

Befehl: Ausgangsport setzen

Zweck: Definiertes Ein- / Ausschalten von vorhandenen Ausgangsports.

Aufbau: p<Portnr>,<Bitnr>,<Wert><CR>

p	= Befehlscode Port setzen
<Portnr>	= Portnummer
<Bitnr>	= Bitnummer, 0 - 7 --> bitweise, 128 --> byteweise
<Wert>	= neuer Wert
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:

p2,128,1	Port 0, byteweise auf 1 setzen
p2,0,1	Port 0 , Bit 0 auf 1 setzen

Erläuterung: „p“ gibt an, dass der Wert eines Ausgabeports gesetzt werden soll. Anschließend wird die Portnummer, die Bitnummer und der neue Portwert getrennt durch Komma übermittelt und der Befehl mit Carriage Return abgeschlossen. Die Steuerung antwortet mit dem Software-Handshake „0“, falls die Speicherung erfolgreich war, oder mit einer Fehlermeldung, falls falsche Portnummern und/oder Werte übergeben wurden. Für die Steuerung IMC4 sind folgende Ports mit entsprechenden Funktionalitäten definiert:

Port	Bit	Wert	Funktion
0	0 - 7	0 - 255	User E/A (auch noch mit 65529 möglich)
1	0	0	Haube darf nicht geöffnet werden
	0	1	Haube darf geöffnet werden
2	0	0	Spindel ausschalten
	0	1	Spindel einschalten
3	0	0	Motorströme ausschalten
	0	1	Motorströme einschalten

Beschränkung: Das Setzen der Portausgänge wird innerhalb der Steuerung dem Programmablauf entsprechend durchgeführt. Somit ist ein Setzen bzw. Löschen von Ausgängen während einer Befehlsbearbeitung z.B. einer Positionierbewegung nicht möglich.

Programmierbeispiel: siehe Anhang D

3.2.13 Port lesen und verzweigen im CNC-Modus

Befehl: Eingangsport lesen

Zweck: Eingangsport lesen und im Programmablauf verzweigen. Durch die Verzweigung kann nach einem logischen Vergleich zu einem bestimmen Satz innerhalb des Programms gesprungen werden.

Aufbau: o<Portnr>,<Bitnr>,<Wert>,<Offset><CR>

o	= Befehlscode Port setzen
<Portnr>	= Portnummer
<Bitnr>	= Bitnummer, 0 - 7 --> bitweise, 128 --> byteweise
<Wert>	= Vergleichswert
<Offset>	= Sprungziel -32768 <= Sprungziel <= 32767
<CR>	= Carriage Return als Befehlsabschluss

Anwendung:	o2,128,1,-1	warten bis Port 0 <> 1
	o2,0,1,-1	warten bis Port0, Bit0 = 0
	o2,0,1,3	wenn Port0, Bit0 == 1, Befehlszähler += 3

Erläuterung: „o“ gibt an, dass der Wert eines Eingabeports gelesen und der Programmablauf entsprechend des Wertes angepasst werden soll. Anschließend wird die Portnummer, die Bitnummer ,der Vergleichswert und der Befehls-Offset getrennt durch Komma übermittelt und der Befehl mit Carriage Return abgeschlossen. Die Steuerung antwortet mit dem Software-Handshake „0“, falls die Speicherung erfolgreich war, oder mit einer Fehlermeldung, falls falsche Portnummern und/oder Werte übergeben wurden. Während des Programmablaufs wird das entsprechende Port abgefragt und bit- oder byteweise logisch mit dem vorgegebenen Wert verglichen. Ist der logische Vergleich wahr, wird um den Offset verzweigt, sonst wird der nächste Befehl im Programmablauf abgearbeitet. Für die Steuerung IMC4 sind folgende Ports mit entsprechenden Funktionalitäten definiert:

Port	Bit	Zustand	Funktion
0	0 - 7	00 - FF	User E/A (auch noch mit 65531 möglich)
1	0	00	Haube ist geöffnet
	0	01	Haube ist geschlossen
2	0	00	Spindel ist ausgeschaltet
	0	01	Spindel ist eingeschaltet
3	0	00	Motorströme sind ausgeschaltet
	0	01	Motorströme sind eingeschaltet

Beschränkung: Das Abfragen der Porteingänge wird innerhalb der Steuerung dem Programmablauf entsprechend durchgeführt. Somit ist eine Abfrage von Eingängen während einer Befehlsbearbeitung z.B. einer Positionierbewegung nicht möglich.

Programmierbeispiel: -

3.2.14 Datenfeld-Ende im CNC-Modus

Befehl: Datenfeld-Ende

Zweck: Der Befehl kennzeichnet das Ende eines CNC-Datenfeldes und dient dem Abschluss der Datenübertragung und Speicherung von speicherbaren Befehlen.

Aufbau: 9<CR>

9	= Befehlscode Datenfeld-Ende
<CR>	= Carriage Return als Befehlsabschluss

Anwendung: 9

Erläuterung: „9“ gibt an, dass das Ende des übertragenen CNC-Datenfeldes erreicht ist. Der Befehl wird mit Carriage Return abgeschlossen. Die Steuerung antwortet mit dem Software-Handshake „0“, falls die Speicherung erfolgreich war, oder mit einer Fehlermeldung. Neben der Kennzeichnung des Datenfeldes als gültiges CNC-Programm werden Status-Informtionen (z.B. die aktuelle Referenzgeschwindigkeit) im FlashProm abgelegt. Anschließend befindet sich die Steuerung wieder im DNC-Modus und akzeptiert die entsprechenden Befehle.

Beschränkung: Ein CNC-Datenfeld muss mit dem Befehl Datenfeld-Ende abgeschlossen werden, sonst ist das abgespeicherte CNC-Programm nicht gültig und kann nicht abgearbeitet werden.

Programmierbeispiel: siehe Anhang D

4 Die Fehlermeldungen der IMC4

Nach jedem übertragenen Befehl antwortet die Steuerung mit einer entsprechenden Rückmeldung. Diese Codes werden als ASCII-Zeichen übertragen und können somit einfach ausgewertet werden. Anhand des übermittelten Zeichens können Fehlerquellen und -ursachen erkannt werden.
Die einzelnen Fehlercodes sind im Folgenden beschreiben.

Code	Beschreibung
0	Handshake-Charakter - Kein Fehler, der Befehl wurde korrekt ausgeführt. - Der nächste Befehl kann übermittelt werden.
1	Fehler in übergebener Zahl - Die Steuerung hat eine Zahlenangabe empfangen, die nicht korrekt interpretiert werden konnte. - Der übergebene Zahlenwert ist außerhalb des zulässigen Bereiches oder der übergebene Zahlenwert enthält unzulässige Zeichen.
2	Endschalter-Fehler - Durch die Verfahrbewegung wurde ein Endschalter angesprochen. Die aktuelle Bewegung wurde abgebrochen. Dies geschieht durch ein stoppen der Bewegung ohne Bremsrampe. Dadurch sind die Ist-Positionen der Steuerung nicht mehr korrekt, wahrscheinlich sind Schrittverluste aufgetreten. - Die Referenzfahrt einer Schrittmotorachse wurde nicht korrekt oder noch nicht ausgeführt. **ACHTUNG:** Nach einem Endschalterfehler muss die Steuerung neu initialisiert und eine Referenzfahrt ausgeführt werden.
3	unzulässige Achsangabe - Der Steuerung wurde eine Achsangabe für einen auszuführenden Befehl übermittelt, die eine nicht definierte Achse enthält. - Verwenden Sie in den Befehlen, die Achsangaben enthalten nur Kombinationen aus Achsen, die auch initialisiert sind.
4	keine Achsen definiert - Bevor der Steuerung Bewegungen oder allgemein Befehle, die eine von der Achsanzahl abhängige Anzahl von Parametern haben, übergeben werden, muss der Befehl „Achsenanzahl setzten“ übergeben werden, um die internen Achsenparameter korrekt zu setzen.
5	Syntax-Fehler - Ein Befehl wurde fehlerhaft übermittelt. - Der verwendete Befehl existiert nicht oder kann von dieser Steuerung nicht abgearbeitet werden. - Überprüfen Sie, ob alle übertragenen Befehle korrekt sind.
6	Speicherende - Es wurde versucht mehr Befehle im CNC-Modus zu übertragen, als in der Steuerung gespeichert werden können.
7	unzulässige Parameterzahl - Die Steuerung hat mehr oder weniger Parameter für den Befehl erhalten, als benötigt werden. - Prüfen Sie, ob die Anzahl der Parameter für den Befehl in Verbindung mit der Anzahl der

	Achsen korrekt ist.
8	zu speichernder Befehl inkorrekt - Der Steuerung wurde ein Befehl übergeben, der als CNC-Befehl nicht verfügbar ist.
9	Anlagenfehler - Die Spannungsversorgung der Anlage ist noch nicht eingeschaltet. - Der Sicherheitskreis der Anlage ist nicht aktiv. - Die Endstufen und/oder der Sicherheitskreis konnten nicht eingeschaltet werden, weil die Haube noch geöffnet ist. - Eine Not-Aus-Situation ist eingetreten. **ACHTUNG:** Nach einer Not-Aus-Situation muss die Steuerung neu initialisiert und eine Referenzfahrt ausgeführt werden.
A	von dieser Steuerung nicht benutzt
B	von dieser Steuerung nicht benutzt
C	von dieser Steuerung nicht benutzt
D	unzulässige Geschwindigkeit - Die zulässigen Grenzen für Geschwindigkeitsangaben wurden nicht eingehalten. - Prüfen Sie, ob alle Geschwindigkeitsangaben korrekt sind.
E	von dieser Steuerung nicht benutzt
F	Benutzer-Stopp - Der Benutzer hat die Halttaste an der Steuerung betätigt, die aktuelle Bewegung wurde angehalten. Die Befehlsausführung kann mit der Starttaste oder dem Startbefehl „@0s“ wieder aufgenommen werden.
G	ungültiges Datenfeld - Der Steuerung wurde ein Startbefehl übermittelt, obwohl kein Bewegungsrest im Speicher vorhanden, d.h. obwohl zuvor keine Haltfunktion ausgeführt wurde. - Es wurde versucht, ein CNC-Programm zu übertragen, obwohl im Speicher noch ein Programm oder Teile eines Programmes vorhanden sind.
H	Haubenfehler - Es wurde versucht, einen Befehl auszuführen, der bei geöffneter Haube nicht zulässig ist.
=	von dieser Steuerung nicht benutzt

A1 Software-Routinen zur Berechnung der Parameter beim Kreisbefehl in Turbo-Pascal

Beispielprogramm zur Berechnung der Kreisparameter 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-Routinen zur Berechnung der Parameter beim Kreisbefehl in C

Beispielprogramm zur Berechnung der Kreisparameter 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 Programmierung IMC4 in Turbo Pascal

Diese Beispielprogramme sollen zeigen, wie z.B. mit der Programmiersprache Turbo Pascal eine einfache Programmierung der IMC4 für einfache Aufgaben erreichbar ist. Außerdem sollen sie zum besseren Verständnis der seriellen Ansteuerung der IMC4 und ihres Funktionsumfangs dienen und Anregung für eigene Applikationen, auch in anderen Programmiersprachen, sein.

B2 Unit für serielle Datenübertragung in Turbo Pascal

Zunächst möchten wir hier kurz eine Unit vorstellen, deren Funktionen in den folgenden Beispielprogrammen benutzt werden und die ein einfaches Handling der seriellen Schnittstelle ermöglicht.

```
{*********************************************}
{* 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 Beispielprogramme in Turbo Pascal

B3.1 Referenzfahrt, Achseninitialisierung

```
{*********************************************************************}
{* 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 Referenzgeschwindigkeit einstellen

```
{************************************************************************}
{* 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 Bewegung

```
{*********************************************************************}
{* 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 Bewegung

```
{*************************************************************************}
{* 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 Nullpunktverschiebung

```
{************************************************************************}
{* 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 Positionsabfrage

```
{*************************************************************************}
{* 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 Ebenenwahl

```
{*************************************************************************}
{* 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 Port lesen

```
{*************************************************************************}
{* 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 Port schreiben

```
{************************************************************************}
{* 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 Beispielprogramm zur Programmierung IMC4 in C

Dieses Beispielprogramm sollen zeigen, wie z.B. mit der Programmiersprache Microsoft C eine einfache Programmierung der IMC4 für einfache Aufgaben erreichbar ist. Außerdem soll dies zum besseren Verständnis der seriellen Ansteuerung der IMC4 und ihres Funktionsumfangs dienen und Anregung für eigene Applikationen, auch in anderen Programmiersprachen, sein.

```
/*********************************************************************/
/***                          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 Beispiel zur Programmierung IMC4 im CNC-Mode

Dieses Beispiel soll zeigen, dass auch mittels BASIC eine einfache Programmierung der IMC4 für einfache Aufgaben erreichbar ist. Die IMC4 kann ab Version 2.00 auf eine Übertragungsrate von 9600 Baud eingestellt werden. Damit ist mittels einfacher BASIC-Anweisungen eine Programmierung der Steuerung (wie bei der isel-Interfacekartenserie) möglich. Die Übertragung der CNC-Befehle kann natürlich auch in jeder anderen Programmiersprache mittels entsprechender Schnittstellenfunktionen erfolgen.

Unter „D2 Beispiel für ein CNC-Programm“ finden Sie zunächst ein kurzes Ablaufprogramm im Pseudecode und in der entsprechenden CNC-Syntax, welches an die Steuerung übertragen werden soll. Diese Programm kann einfach in einem ASCII-File abgelegt sein. Im Punkt D3 finden Sie dann ein kurzes BASIC-Programm, mit welchem ein solches File als CNC-Programm an die Steuerung übertragen werden kann. Das Programm ist bewußt sehr einfach gehalten, für einen normalen Betrieb sollten die Auswertung der Rückmeldungen und die Reaktionen darauf einer umfangreicheren Behandlung durch das Programm unterzogen werden. Die Fehlermeldungen und deren mögliche Ursachen finden Sie unter „4 Fehlermeldungen der IMC4“.

D2 Beispiel für ein CNC-Programm

D2.1 Pseudocode

Der hier verwendete Pseudocode soll lediglich der Veranschaulichung des Programmablaufes dienen und gleichzeitig selbsterklärend sein. Werte in () stellen jeweils Parameter für einen Befehl dar. Abschnitte in {} sind als Blöcke des Programms zu verstehen. Die Koordinaten wurden in Millimeter, die Geschwindigkeiten in Schritten/Sekunde angegeben. Bei linearen Bewegungen steht X für die Bewegung der X-Achse, Y für die Bewegung der Y-Achse. Für die Z-Achse werden bei Absolutbewegungen 1 Parameter und bei Relativbewegungen 2 Parameter angegeben. Bei zirkularen Bewegungen steht R für den Radius in Millimetern, A für den Anfangswinkel und E für den Endwinkel in Grad.

```
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

Aus dem unter D2.1 beschriebenen Pseudocode würde sich bei einer Süindelsteigung von 10 mm und einer Schrittauflösung von 400 Schritten/Umdrehung folgendes CNC-Programm ergeben (zur Umrechnung von mm in Schritte siehe „1.3 Umrechnungsfaktoren“, zu Berechnung der Parameter für Kreisbewegungen siehe „2.3 Die Berechnung der Kreisparameter“):

```
@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-Beispiel zur Übertragung eines CNC-Programms

Zur Übertragung des CNC-Programms könnte folgendes kurzes BASIC-Programm, bei welchem der Dateiname des CNC-Programms als Kommandozeilenparameter übergeben werden kann, dienen:

```
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 ------------------------------------------------------------------------
```

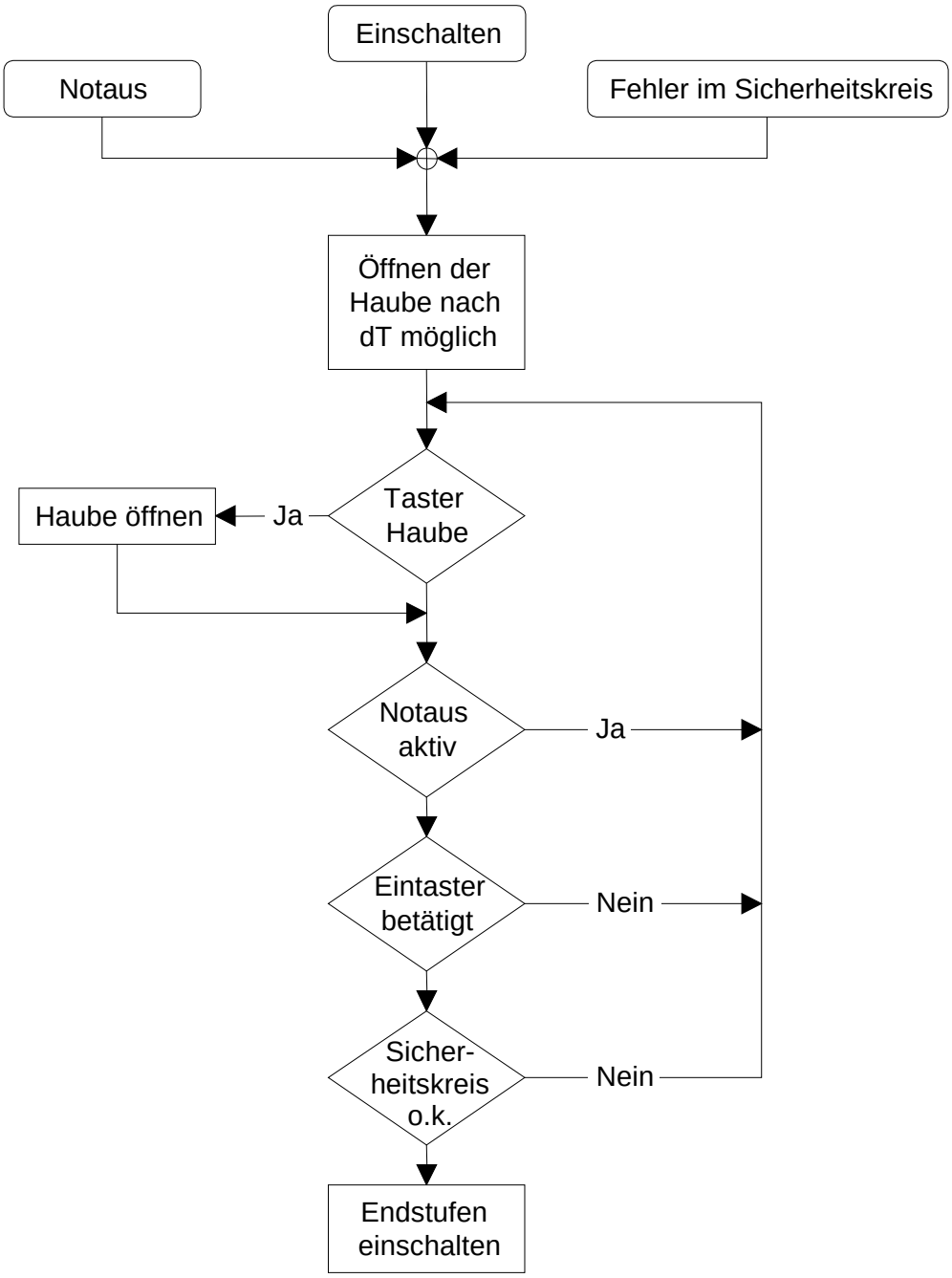
E1 Bedienelemente der IMC4

Die Bedienelemente der IMC und deren Funktionen seien der Vollständigkeit halber hier kurz beschrieben. Eine Veranschaulichung der Funktionen kann den nachfolgenden Flußdiagrammen entnommen werden.

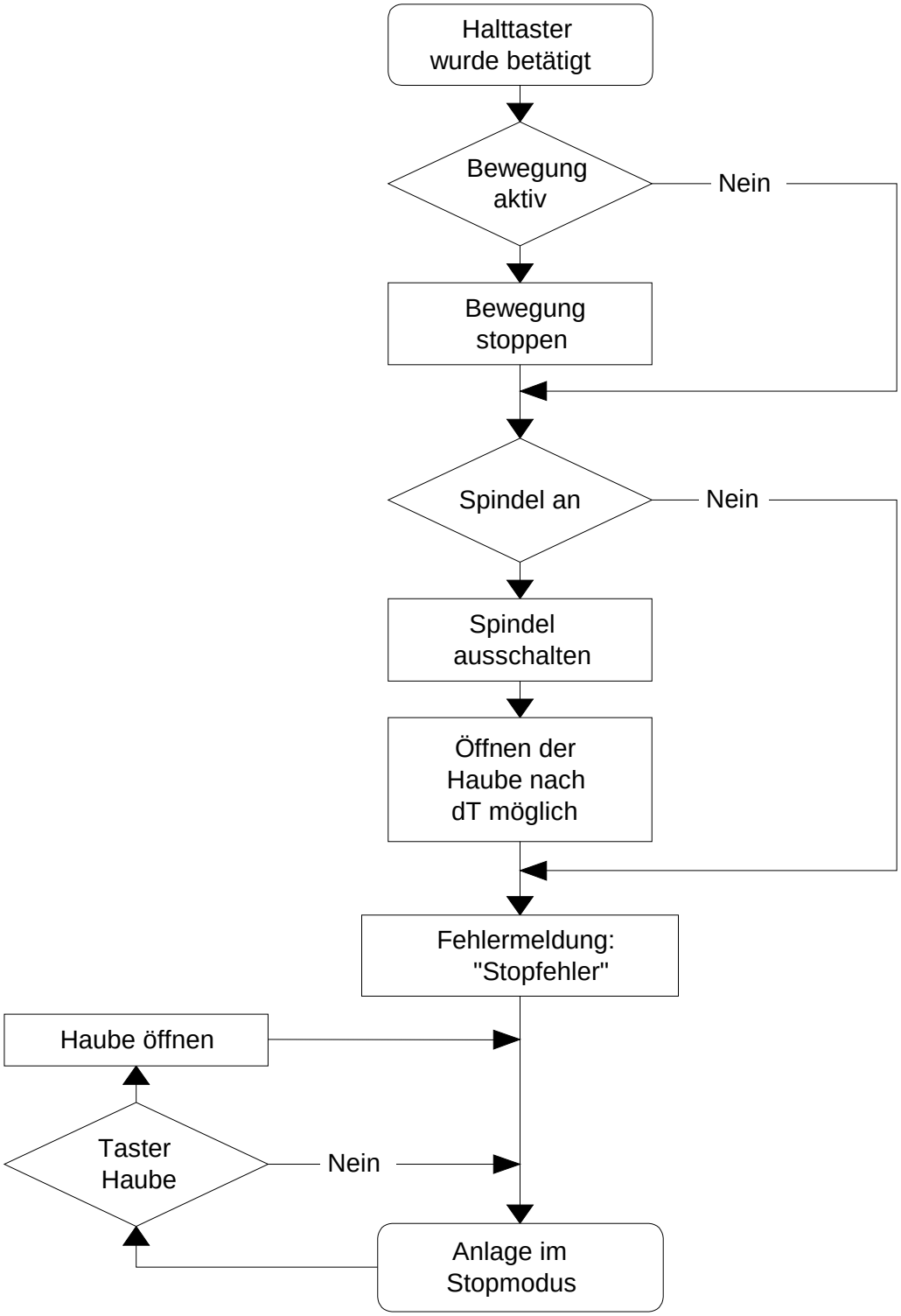
Bedienelement	**Funktion**
Not-Aus-Schalter	Not-Aus-Funktion, Sicherheitskreis - bei Betätigung Abschaltung der Endstufenversorgung, Abschalten aller Ausgänge, Software-Reset - bei aktivem Not-Aus können die Endstufen nicht zugeschaltet werden
EIN-Taster	Einschalten der Endstufen, Selbstest, Löschen der FlashProm's - dient im Grundzustand zum Einschalten der Endstufen - bei gleichzeitig betätigter Start-Taste wird Selbsttest ausglöst - bei gleichzeitig betätigter Stopp-Taste werden FlashProm's gelöscht
Schlüsselschalter	Umschaltung Test-Modus - Automatik, Löschen der FlashProm's - im Automatik-Modus kann bei geöffneter Haube nicht verfahren werden - im Test-Modus können die Achsen bei offener Haube verfahren werden - zum Löschen der FlashProm's muss der Schalter auf Test stehen
Start-Taster	Starten von CNC-Programmen, -bewegungen, Löschen der FlashProm's - Start von abgespeicherten CNC-Programmen - Start von angehaltenen Bewegungen im CNC-Modus - Löschen von FlashProm's im Grundzustand der Steuerung
Stopp-Taster	Anhalten von Bewegungen, Selbsttest - Anhalten von Bewegungen im CNC- und DNC-Modus, bei laufender Spindel wird diese anschließend abgeschaltet - Ausführen des Selbsttests wenn Stopp beim Einschalten der Endstufen betätigt ist
Hauben-Taster	Öffnen der Schutzhaube - bei beleuchtetem Taster ist das Öffnen der Schutzhaube möglich

E2 Funktionalitäten der Bedienelemente

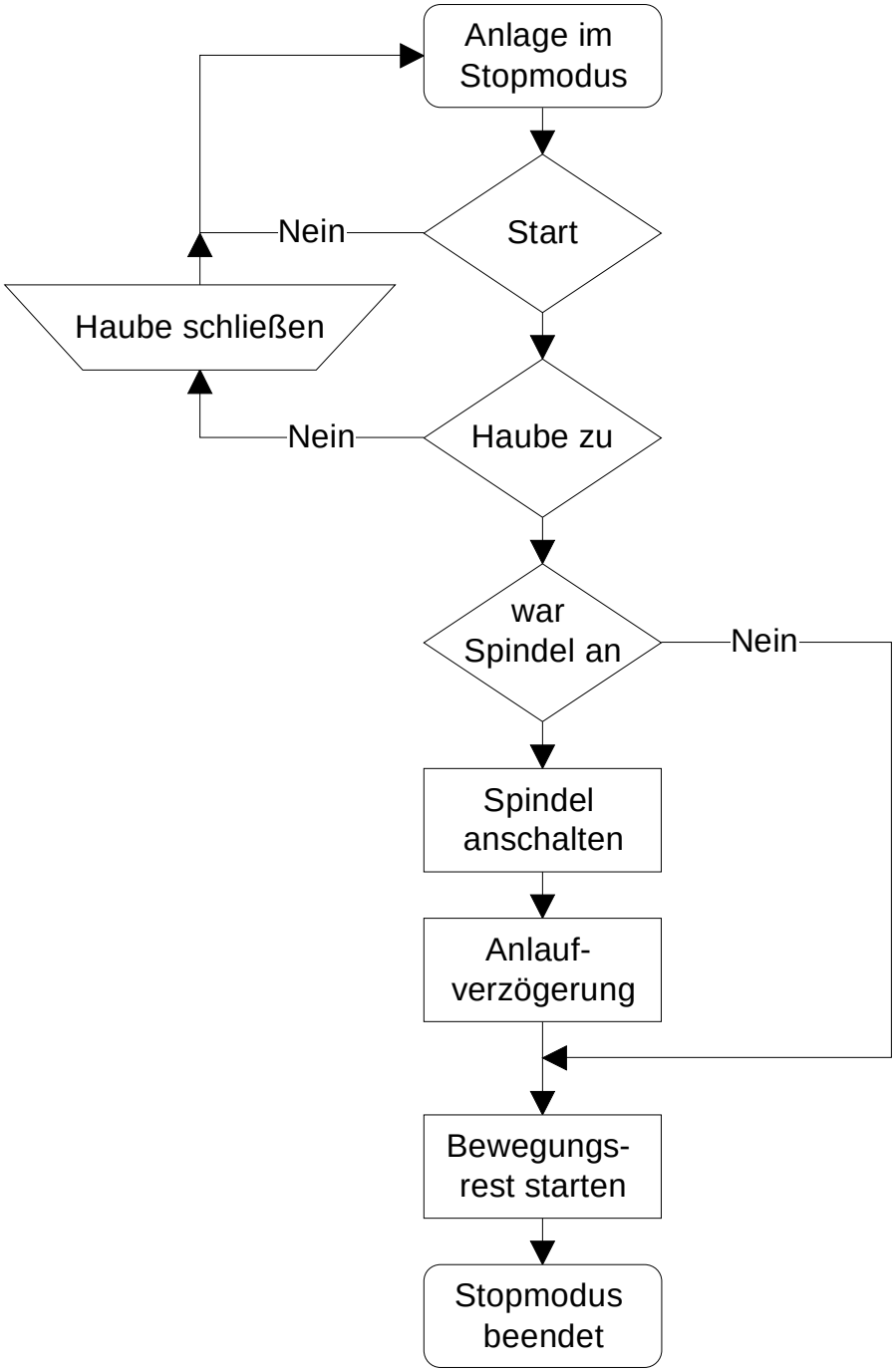
E2.1 Einschalten der Endstufen



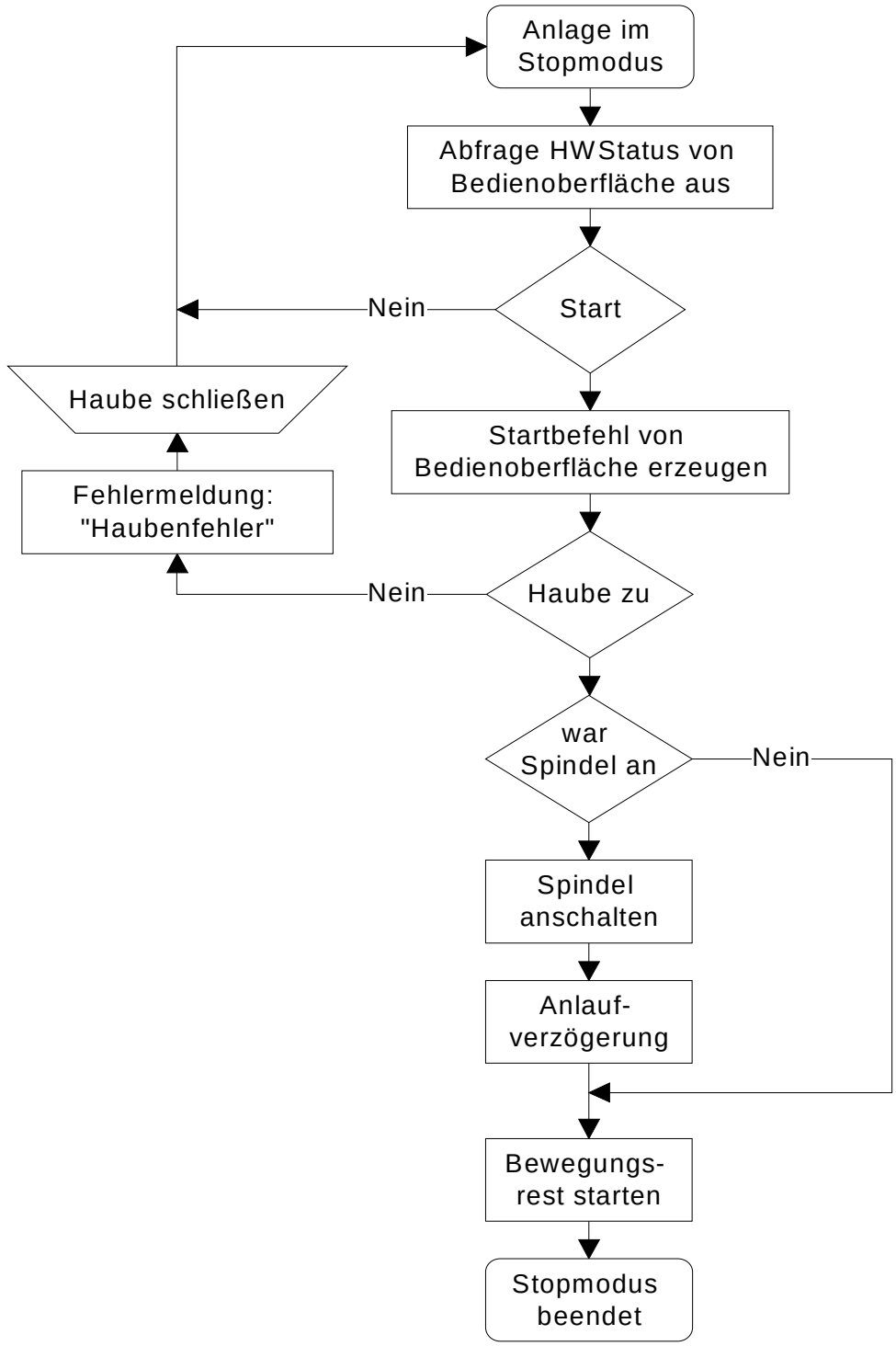
E2.2 Funktion des Halt-Tasters



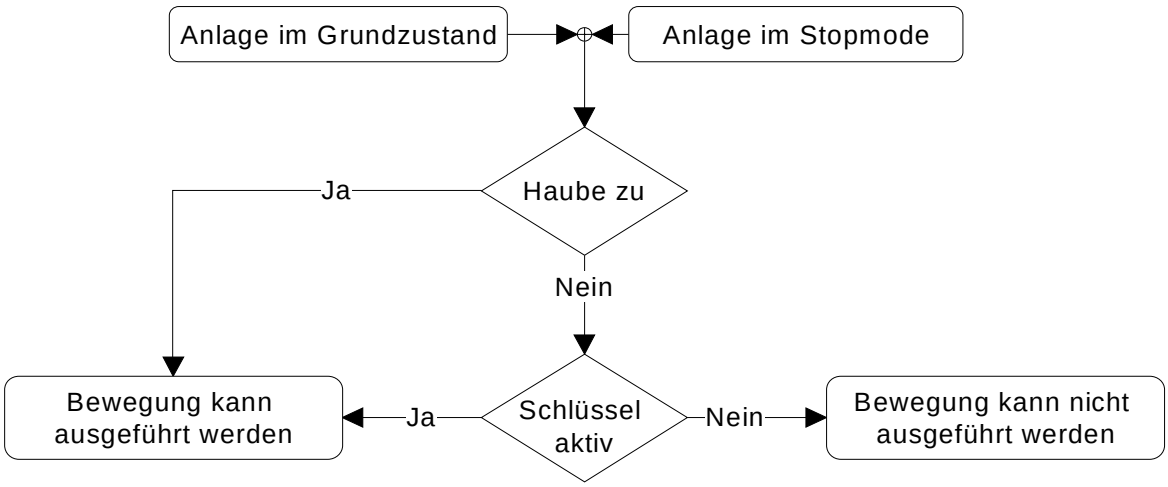
E2.3 Funktion des Start-Tasters im CNC-Modus



E2.4 Funktion des Start-Tasters im DNC-Modus



E2.5 Funktion des Schlüsselschalters



E2.6 Löschen der FlashProm's