CPC 464
CPC 664

SYBEX

# Mein Schneider CPC

Norbert Hesselmann
Christoph Hesselmann

# Mein Schneider CPC

# Mein Schneider CPC

Norbert Hesselmann
Christoph Hesselmann


DÜSSELDORF · BERKELEY · PARIS

Umschlagentwurf: Daniel Boucherie/tgr
Satz: tgr – typo-grafik-repro gmbh, Remscheid
Gesamtherstellung: Boss-Druck und Verlag, Kleve



ISBN 3-88745-602-5
1. Auflage 1985

# Inhaltsverzeichnis

## Anhang

# Vorwort

In dem vorliegenden Buch finden Sie eine Fülle nützlicher Informationen zu den beiden Schneider-Computern CPC 464 und CPC 664. Um einem möglichst großen Anwender- und Leserkreis gerecht zu werden, spannt das Werk einen weiten Bogen von den ersten Kontakten zu Ihrem Schneider CPC 464 bis hin zum Anschluß und den Betrieb von schnellen Massenspeichern unter dem Betriebssystem CP/M. Es weist Sie fundiert in den Umgang mit dem System ein und vermittelt Ihnen alle Kenntnisse, die Sie zur erfolgreichen Entwicklung eigener Programme oder zur Anwendung professioneller Software benötigen.

Verlag wie auch die Autoren haben sich bemüht, ein möglichst fehlerfreies Produkt herzustellen. Die Erfahrung zeigt jedoch, daß trotz intensiver Korrektur bei der Herstellung immer noch der eine oder andere Fehler durch die Maschen schlüpft. Sollten Sie bei Ihrer Lektüre auf so einen mißlichen Fehler stoßen, dann zögern Sie nicht, dem Verlag eine kleine Notiz zukommen zu lassen.

Den Mitarbeiterinnen und Mitarbeitern des SYBEX-Verlags möchten wir an dieser Stelle unseren Dank für die Hilfe bei der Planung und der Herstellung dieses Buches aussprechen. Unser besonderer Dank gilt auch Herrn H. Krott, der die vielen Fotos und Zeichnungen mit großer Sorgfalt und Liebe zum Detail hergestellt hat.

Aachen, im April 1985

Norbert Hesselmann
Christoph Hesselmann

# Einleitung

Ein Buch wie dieses sollte nach den Vorstellungen der Autoren viele praktische Aufgaben erfüllen. Zum einen sollte es dem Einsteiger, der sich anderweitig bereits mit dem Problemkreis der Mikrocomputertechnik vertraut gemacht hat, jene Informationen zur Verfügung stellen, die er für eine erste Auseinandersetzung mit dem System und seiner Peripherie benötigt. Es sollte zum anderen dem erfahrenen Systemanwender Informationen bieten, die im Handbuch nicht enthalten sind, und außerdem für lange Zeit als ein nützliches und unentbehrliches Nachschlage werk dienen.

Diesem Anspruch gerecht zu werden, ist natürlich nicht ganz einfach, da auf dem Wege zu diesem Ziel zwangsweise Kompromisse geschlossen werden müssen. In der Ihnen vorliegenden Form stellt dieses Buch das Ergebnis intensiver Überlegungen und einer eingehenden praktischen Auseinandersetzung mit diesem höchst leistungsfähigen Heimcomputer dar. Es enthält in komprimierter Form zu fast allen wichtigen Anwendungen ein Fülle von Fakten, die Sie unter Umständen erst durch langwieriges Studium vieler anderer Bücher und Artikel sammeln würden.

Das *Kapitel 1* dieses Buches stellt Ihnen zu Beginn die Systemhardware vor und beschreibt im wesentlichen die Verbindungen zur „Außenwelt", d. h. zu peripheren Geräten wie Drucker und Diskettenlaufwerk.

*Kapitel 2* beschäftigt sich mit der Schnittstelle Mensch–Computer, d. h., es erläutert Ihnen die drei Standardbetriebsarten des Systems für die Abbildung von Texten auf dem Bildschirm, macht Sie mit den Mechanismen der Farbwahl, der Definition und Anwendung von Textfenstern und mit Besonderheiten der Tastatur vertraut.

In *Kapitel 3* erhalten Sie eine fundierte Einführung in den Umgang mit der Programmiersprache BASIC. Sie lernen die Regeln für den Aufbau von BASIC-Programmen und für das Editieren und Korrigieren von Programmtexten auf dem CPC 464/664 kennen. Die meisten BASIC-Befehle

sind anhand von Beispielen erläutert, die zum Teil durch Abbildungen ergänzt werden. Ein Abschnitt über die Arbeitsweise des Interpreters und die speicherinterne Struktur von BASIC-Programmen schließt das Kapitel ab.

*Kapitel 4* macht Sie mit den außergewöhnlichen grafischen Fähigkeiten des CPC 464/664 bekannt. Es erläutert das Systemverhalten in den drei verschiedenen Betriebsarten und die Organsiation des Bildwiederholspeichers. Einige wichtige Grafikbefehle werden ausführlich erklärt. Außerdem lesen Sie auch etwas über die Definition von Grafikfenstern sowie die gemischte Darstellung von Grafik und Text.

Das *Kapitel 5* setzt sich eingehend mit der Erzeugung von Tönen und Geräuschen auseinander. Sie lernen anhand vieler Beispiele, die durch Abbildungen unterstützt werden, die Funktionsweise des Soundgenerators (PSG) und dessen Programmierung kennen. Die unter BASIC zur Verfügung stehenden Befehle SOUND, ENV, ENT, ON SQ, SQ und RELEASE werden ausführlich vorgestellt.

Für diejenigen von Ihnen, die sich ein wenig genauer für die technische Funktion des CPC 464/664 interessieren und sich in einem Schnellgang auf den Umgang mit Maschinenprogrammen vorbereiten wollen, gibt *Kapitel 6* einen Einblick in den internen Aufbau eines Mikrocomputers sowie in die Funktion des zentralen Bauelementes, des Mikroprozessors Z80. Sie lernen außerdem den wichtigen Umgang mit binären Datenworten und deren externe Darstellung in unterschiedlichen Zahlensystemen kennen.

Mit den in Kapitel 6 erworbenen Kenntnissen können Sie sich anschließend in *Kapitel 7* dem Thema Maschinensprache widmen. Sie erhalten in diesem Kapitel all jene Informationen, die Sie für einen fundierten Einstieg in dieses schwierige, aber dennoch sehr reizvolle Thema benötigen. Nach der Lektüre dieses Kapitels werden Sie mühelos weiterführende Literatur zu diesem speziellen Themenkreis mit Erfolg lesen und verarbeiten können. Im einzelnen werden Sie mit der Struktur von Maschinenbefehlen, deren Verarbeitung im Mikroprozessor, dessen Befehlssatz sowie der Erstellung einfacher Maschinenroutinen anhand von erläuternden Abbildungen und kleineren Programmbeispielen bekannt gemacht. Sie erhalten so ein Gefühl für den großen Gewinn an Ablaufgeschwindigkeit, den Sie im Vergleich zu den in BASIC erstellten Programmen erzielen können.

*Kapitel 8* erläutert den Gebrauch des Diskettenlaufwerks. Dieses ist beim CPC 664 integraler Systembestandteil. Für den CPC 464 ist es unter der

Bezeichnung DDI-1 als externes Zusatzgerät erhältlich. Mitgeliefert wird in beiden Fällen das Disketten-orientierte Betriebssystem AMSDOS, das weltweit verbreitete Betriebssystem CP/M des Softwarehauses Digital Research sowie die unter CP/M lauffähige Programmiersprache DR. LOGO. Sie erhalten Informationen über die Technik von schnellen Massenspeichern, die Datenorganisation auf Disketten sowie die Pflege von Laufwerk und Datenträgern. Die wesentlichen Unterschiede zwischen den beiden erwähnten Betriebssystemen werden Ihnen genauso erläutert wie die Anwendung der entsprechenden Befehlserweiterungen. Ergänzt wird das Kapitel durch eine kurze Vorstellung der wichtigsten Dienstprogramme unter dem Betriebssystem CP/M sowie eine kommentierte Zusammenstellung der in diesen Programmen zulässigen Befehle. Auf die Programmiersprache LOGO wird nicht eingegangen, da dies den Rahmen des Buches sprengen würde.

Das letzte *Kapitel 9* beschäftigt sich mit dem aktuellen Themenkreis der Datenfernübertragung über das Telefonnetz (DFÜ). Sie wird für die beiden CPC-Systeme durch den Zukauf einer seriellen Schnittstelle, eines Akustikmodems sowie des zugehörigen Kommunikationsprogramms ermöglicht. Sie erfahren in diesem Kapitel alles, was Sie für den Einstieg in die Datenkommunikation benötigen.

Der umfangreiche *Anhang* enthält von der Standard-ASCII-Tabelle über einige wichtige Einsprungadressen des im ROM enthaltenen Betriebssystems bis hin zur Übersicht über musikalische Notenbezeichnungen und deren Frequenz- und Parameterwerte all jene Informationen, die Sie vermutlich im Laufe Ihrer Arbeit mit dem CPC 464/664 immer wieder benötigen werden. Er erspart Ihnen in den weitaus meisten Fällen das Nachblättern in den unterschiedlichen Kapiteln, aber sicherlich auch den Griff zu manch anderem Buch.

Eine Übersicht über die bei Drucklegung dieses Buches verfügbare deutschsprachige Literatur und ein umfangreiches Stichwortverzeichnis runden das Buch ab.

# Kapitel 1
# Der Schneider CPC

## Übersicht

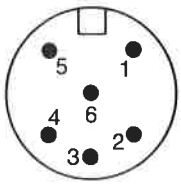
Natürlich werden Sie, wenn Sie bereits den Heimcomputer Schneider CPC 464/664 besitzen, im wesentlichen über Ihr System informiert sein. Sie können daher für den Fall, daß Sie das Gefühl haben, alles über den CPC zu wissen, ruhig dieses Kapitel überschlagen. Falls Sie sich dennoch zur Lektüre entschließen, lesen Sie das Wichtigste über den Aufbau Ihres Heimcomputers sowie die Bedeutung und Funktionsweise der äußeren Anschlüsse. Sie erhalten außerdem einige Informationen über das, was Sie nach dem Einschalten des Computers erwartet, einige nützliche Hintergrundinformationen zum System sowie Tips für den Anschluß der wichtigsten peripheren Komponenten.

## Die Komponenten des Grundsystems

Der Schneider CPC besteht in der Grundausstattung aus der Zentraleinheit sowie einem monochromen oder farbtüchtigen Sichtgerät mit einem Bildschirmdurchmesser von 12 Zoll ohne Ton- und Fernsehempfangsteil. Sichtgeräte dieses Typs werden in der Fachsprache auch als *Monitore* bezeichnet.

### *Der Videoanschluß*

Das zur Ansteuerung benötigte Schwarzweiß- oder Farbsignal wird an einer 6-poligen DIN-Buchse an der Gehäuserückseite zur Verfügung gestellt, dessen Belegung in Abb. 1.1 gezeigt ist. Die für die Farbansteuerung benötigten Signale treten getrennt als Rot- (R), Grün- (G) und Blausignale (B) an den Anschlußstiften 1, 2 und 3 auf. Das zusätzlich benötigte Synchronisationssignal kann an PIN 5 abgegriffen werden.




| | | | |
|---|---|---|---|
| Pin 1 | Rot | Pin 4 | Synchronisation |
| Pin 2 | Grün | Pin 5 | Masse |
| Pin 3 | Blau | Pin 6 | BAS-Signal |

*Abb. 1.1: Pin- und Signalbelegung des Videoanschlusses*

Aus den zuvor genannten Bildsignalen wird über eine elektronische Schaltung ein für die Ansteuerung des Schwarzweiß-Sichtgerätes verwendbares BAS-Signal (*B*ild-*A*ustast- und *S*ynchronisationssignal) gewonnen. Hierzu werden, wie auch dem Anhang V des Bedienungshandbuches zu entnehmen ist, die RGB-Signale zusammen mit dem Synchronisationssignal über einen Transistor ausgekoppelt. Die Vorwiderstandswerte sind seitens des Herstellers so gewählt, daß bei Betrieb eines monochromen Monitors eine vernünftige Grauskala erzielt werden kann. Das so erzeugte BAS-Signal liegt an Pin 6 des Videosteckers an. Beachten Sie bitte, daß dieses Signal auch bei Anschluß eines FBAS-Videomonitors immer nur ein monochromes Bild liefert.

Die Spannungsversorgung des Computers wird beim CPC 464 durch ein 5V-Netzteil vorgenommen. Es ist ein integraler Bestandteil des Monitors. Die benötigte Versorgungsspannung kann auch einem gesonderten Netzteil entnommen werden, das unter der Bezeichnung MP-1 im Handel erhältlich ist. Dieses Netzteil enthält zusätzlich einen Hochfrequenzmodulator, mit dessen Hilfe Sie anstelle des mitgelieferten Sichtgerätes ein normales Fernsehgerät an Ihren CPC 464 anschließen können. Der CPC 664 benötigt im Gegensatz zum CPC 464 noch zusätzlich 12 Volt Versorgungsspannung für das integrierte Diskettenlaufwerk. Diese wird von dem mitgelieferten Monitor zur Verfügung gestellt werden. Die Sichtgeräte sind somit nur bedingt zwischen dem CPC 464 und dem 664 austauschbar. Handelsübliche Fernsehgeräte benötigen im Gegensatz zu Monitoren ein Videosignalgemisch, das einem hochfrequenten Träger aufmoduliert wird. Dieses wird wie ein Antennensignal eines Senders über den Antenneneingang eingespeist. Diese Lösung ist dann interes-

sant, wenn Sie keinen Farbmonitor gekauft haben oder wenn Sie anstelle eines Farbmonitors einmal ein Schwarzweiß-Sichtgerät anschließen möchten. Für mehr kommerzielle Anwendungen in der durch MODE 2 gekennzeichneten Betriebsart ist dies sehr zu empfehlen, da die Abbildungsschärfe für Text- und Grafikdarstellungen bei monochromen Monitoren üblicherweise größer als bei dem Schneider-Farbmonitor 640 ist.

Viele Versuche haben gezeigt, daß die Verwendung guter Computermonitore am Videoausgang des CPC 464/664 ohne Probleme möglich ist. Der Schneider-Farbmonitor ist darüber hinaus auch für andere Computersysteme mit RGB-Ausgang hervorragend zu verwenden. Entsprechende Adapterkabel kann man sich leicht selbst herstellen oder aber in einer Fachwerkstatt preiswert anfertigen lassen.

### *Die Tastatur*

Die zusammen mit dem Massenspeicher in das Computergehäuse integrierte Tastatur besitzt eine Zeichenbelegung nach dem internationalen ASCII-Standard. ASCII ist eine Abkürzung für *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange. Sie finden genauere Informationen zu diesem Thema im nachfolgenden Kapitel 2. An dieser Stelle ist nur wichtig festzustellen, daß die hierzulande üblichen Umlaute Ä, Ö, Ü, ä, ö, ü und der Laut ß nicht vorhanden sind. Außerdem ist die Anordnung der Buchstaben Z und Y vertauscht. Dies ist nur dann von Bedeutung, wenn Sie einmal mit Hilfe Ihres Schneider-Computers Texte verarbeiten möchten. Wie an anderer Stelle noch erläutert wird, ist eine für die Textverarbeitung angemessenere Zeichenbelegung der Tasten ohne Schwierigkeiten möglich. Für die Programmerstellung und die unmittelbare Bedienung des CPC ist die vorliegende Ausführung der Tastenbelegung besser. Neben den auf einer Schreibmaschine üblichen Zeichen finden Sie auf der Tastatur Ihres Computers einige Sonderzeichen und Spezialtasten, auf die an gegebener Stelle noch eingegangen wird.

### Die Verbindungen zur Außenwelt

Computersysteme sind nur dann voll nutzbar, wenn sie durch Zusatzgeräte, wie beispielsweise Drucker oder Diskettenlaufwerke, ergänzt werden. Alle Geräte, die über Kabel mit dem zentralen Computer verbunden sind, werden als *periphere* Geräte oder einfach als *Peripherie* bezeichnet. Jedes gute Computersystem stellt für diesen Zweck Anschlüsse zur Verfügung, die üblicherweise rückseitig am Gehäuse des Computers zugänglich sind. Auf der Mutterplatine des Rechners befinden sich zur Versor-

gung dieser Anschlüsse elektronische Schaltungen, die als *Interfaces* oder als Schnittstellen bezeichnet werden. Ihre Aufgabe besteht darin, die vom CPC verwendeten Signale an die der externen Geräte anzupassen. Die weitaus meisten Interface-Elektroniken bestehen heute nur noch aus einem höchstintegrierten Bauelement, das in seinem Inneren viele tausend Transistorfunktionen vereinigt.

### *Der Druckeranschluß*

Für den Anschluß eines Druckers steht beim Schneider CPC 464 eine Schnittstelle nach dem Centronics-Standard zur Verfügung. Sie wird kurz Centronics-Schnittstelle genannt. Die Bezeichnung geht auf den Druckerhersteller Centronics Corporation zurück, der als erster diese Schnittstelle verwendet hat. Abb. 1.2 zeigt die Pinnumerierung und die Signalbelegung der entsprechenden Messerleiste.

Zu beachten ist, daß die Signalleitung mit der Bezeichnung DATA 7 auf Massepotential liegt. Für die Übermittlung von Standard-ASCII-Zeichen ist dieser Umstand nicht von Bedeutung, da deren Codes 127 nicht überschreiten. Das heißt, sie sind mit 7 Bits zu verschlüsseln. Die Ansteuerung von grafikfähigen Druckern dagegen, wie beispielsweise des Schneider NLQ 401 oder eines EPSON-Druckers, wird durch diesen Umstand allerdings etwas erschwert. In Zeitschriften wie auch einigen Büchern werden


| | |
|---|---|
| Pin 1: $\overline{\text{STROBE}}$ | Pin 6: DATA 4 |
| Pin 2: DATA 0 | Pin 7: DATA 5 |
| Pin 3: DATA 1 | Pin 8: DATA 6 |
| Pin 4: DATA 2 | Pin 9: DATA 7 |
| Pin 5: DATA 3 | Pin 11: BUSY |

Pin 14,16 und 19 bis 33: Masse

*Abb. 1.2: Numerierung und Signalbelegung des Druckeranschlusses*

praktikable Hardcopy-Routinen beschrieben, die diesen mißlichen Umstand mit einigen Tricks umgehen. Die für die Ansteuerung des Drukkers erforderlichen Signale erzeugt auf der Mutterplatine des CPC ein einziger Baustein mit der Bezeichnung Z 8255. Fachleute bezeichnen ihn als Programmierbaren Interface Adapter (PIA).

Die folgende Abb. 1.3 zeigt das Anschlußschema des Centronics-Steckers am Drucker NLQ 401 von Schneider. Dieser Drucker ist übrigens technisch baugleich mit dem Drucker M1009 von der Firma Brother. Der Zeichensatz im NLQ 401 ist speziell auf den Schneider CPC 464/664 zugeschnitten (siehe auch Anhang B).

*Hinweis:* Da Pin 14 auf Masse liegt, wird beim Schneider Drucker NLQ 401, der technisch baugleich mit dem Brother M1009 ist, bei jedem gesendeten Wagenrücklaufzeichen (CR = *C*arriage *R*eturn) automatisch ein Zeilenvorschub erzeugt. Da üblicherweise der Zeilenvorschub (LF = *L*ine *F*eed) mit ausgesendet wird, macht der Drucker immer *zwei* Zeilenvorschübe. Dies ist nur dadurch zu unterbinden, daß die Leitung am PIN 14 unterbrochen oder aber der Anschlußstift isoliert wird. Die nachfolgende Abb. 1.4 zeigt, welche der Leitungen des Flachbandkabels zu dieser Anschlußfahne führt, damit Sie sich beim Durchtrennen nicht vertun.

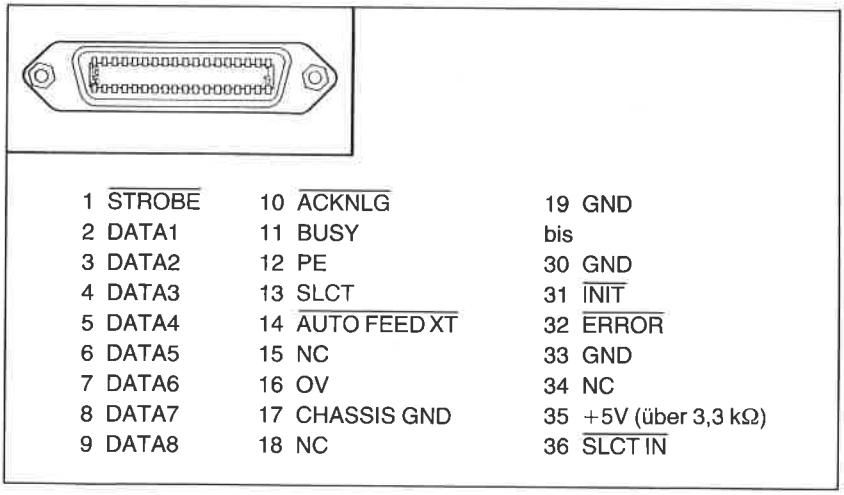



*Abb. 1.3: Anschlußbelegung des Centronics-Steckers am Matrixdrucker NLQ 401 der Firma Schneider*

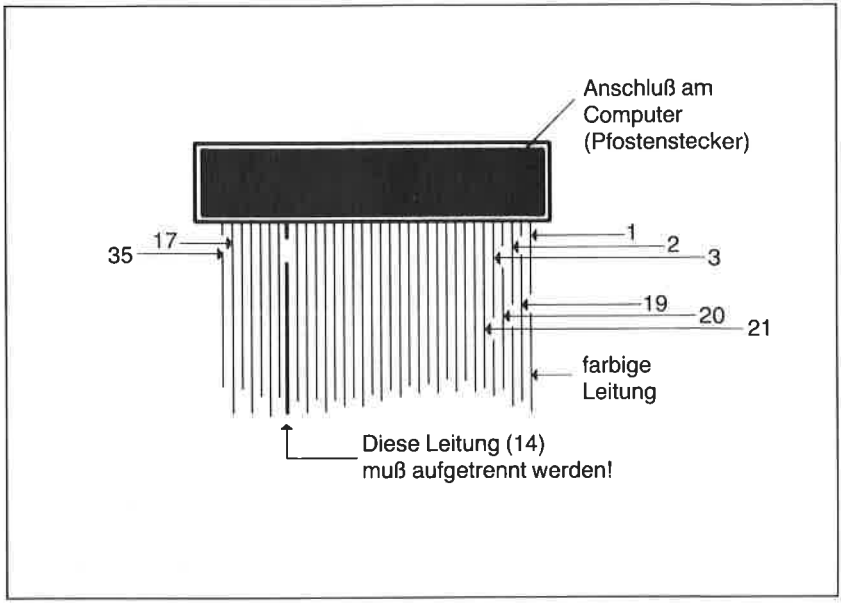



*Abb. 1.4: Die Flachbandkabelverbindung zum Pfostenstecker des Druckerkabels. Bitte achten Sie darauf, daß die Leitungen mit den Nummern 18 und 36 nicht existieren*

### *Der Erweiterungsanschluß*

Neben der erwähnten Centronics-Schnittstelle ist noch ein Anschluß für Diskettenlaufwerke oder andersgeartete Systemerweiterungen vorhanden. Die Anschlußfahnen sind, wie für den Drucker auch, Bestandteil der Mutterplatine (Abb. 1.5). Achten Sie bitte darauf, daß Sie Kontakte dieses Anschlusses möglichst nicht berühren. In trockener Umgebung können Sie sich nämlich sehr leicht mit statischer Elektrizität aufladen, die mehrere 10000 (für Sie selbst ungefährliche) Volt erreichen kann. Bei Berührung der Kontaktfahnen wird sich die elektrische Ladung über die hochempfindlichen Bauelemente zu entladen suchen. Es kann sein, daß diese dann einen nicht rückgängig zu machenden Schaden nehmen.

Für den CPC 464 wird ein externes Diskettenlaufwerk geliefert, dessen Interface mit dem Erweiterungsanschluß verbunden wird. Beim CPC 664 ist das Diskettenlaufwerk integraler Systembestandteil. Lesen Sie für den Fall, daß Sie sich über dieses Laufwerk und seinen Betrieb näher informieren möchten, bitte die Ausführungen in Kapitel 8, das sich speziell mit diesem Themenkreis befaßt.


| Pin-Nr. | Signal | Pin.-Nr. | Signal | Pin.-Nr. | Signal |
|---|---|---|---|---|---|
| 1 | SOUND | 18 | A0 | 35 | INT |
| 2 | GND | 19 | D7 | 36 | NMI |
| 3 | A15 | 20 | D6 | 37 | BUSRD |
| 4 | A14 | 21 | D5 | 38 | BUSAK |
| 5 | A13 | 22 | D4 | 39 | READY |
| 6 | A12 | 23 | D3 | 40 | BUS RESET |
| 7 | A11 | 24 | D2 | 41 | RESET |
| 8 | A10 | 25 | D1 | 42 | ROMEN |
| 9 | A9 | 26 | D0 | 43 | ROMDIS |
| 10 | A8 | 27 | +5V | 44 | RAMRD |
| 11 | A7 | 28 | MREQ | 45 | RAMDIS |
| 12 | A6 | 29 | M1 | 46 | CURSOR |
| 13 | A5 | 30 | RFSH | 47 | LIGHT PEN |
| 14 | A4 | 31 | IORQ | 48 | EXP |
| 15 | A3 | 32 | RD | 49 | GND |
| 16 | A2 | 33 | WR | 50 | Ø |
| 17 | A1 | 34 | HALT | | |

*Abb. 1.5: Belegung der Kontakte des Erweiterungssteckers*

### *Der Kassettenrecorder als externer Massenspeicher*

Ein Anschluß für einen gesonderten Kassettenrecorder, der im Heimcomputerbereich gerne als preiswerter Speicher für Programme und Daten verwendet wird, ist beim Schneider CPC 464 nicht vorgesehen und auch nicht notwendig, da der Recorder integraler Bestandteil des Gerätes ist. Die gerade im Heimcomputerbereich üblichen Probleme mit Fremdprodukten treten hier nicht auf, da Recorder und Interface optimal aufeinander abgestimmt sind. Der CPC 664 mit seinem integrierten Diskettenlaufwerk besitzt einen zusätzlichen Anschluß für einen externen Kassettenrecorder. Wie Sie in Kapitel 6 gegebenenfalls nachlesen können, werden vom Computer nur binäre Signale in Form von Spannungswerten mit zwei definierten Zuständen (Spannung – keine Spannung) verarbeitet. Diese Gleichspannungswerte müssen, bevor sie auf Magnetband mit einem sogenannten analog aufzeichnenden Recorder abgespeichert werden können, in Wechselspannungssignale umgesetzt werden.

Hierfür besitzt der CPC, wie andere Heimcomputer auch, eine spezielle elektronische Schaltung, die Datenwerte in eine Folge von Tönen unterschiedlicher Frequenz zerlegt. Erst danach können sie auf Band aufgezeichnet werden. Bei dem umgekehrten Vorgang wandelt dieselbe Schaltung die von Band eingelesene Tonfolge wieder in Gleichspannungswerte um, die vom Computer verarbeitet werden können. Alle Angaben zum Betrieb des Recorders finden Sie im Handbuch zum Schneider CPC. Achten Sie darauf, daß Sie den für die Aufzeichnung und die Wiedergabe verantwortlichen Tonkopf des Recorders stets sauberhalten. Tips für die Pflege des Gerätes sowie gegebenenfalls zur Justierung des Tonkopfs finden Sie u. a. in dem Buch „Mein Heimcomputer", das ebenfalls im SYBEX-Verlag erschienen ist. Als Bandmaterial verwenden Sie möglichst einfaches Low-Noise-Material eines Herstellers mit gutem Namen und kein Chromdioxid- oder Reineisenband.

### *Die Tonausgabe*

Die mit dem Soundgenerator (PSG) des Typs AY-3-8912 erzeugten Töne und Geräusche werden über einen im CPC eingebauten Verstärker mit nachgeschaltetem Lautsprecher wiedergegeben. Über eine Koaxialbuchse (für sogenannte Klinkenstecker) können Sie den Ausgang des Klangerzeugungsbausteins auch auf den Eingang einer Stereoanlage schalten. Dabei ist es wichtig zu wissen, daß der Kanal A des PSG auf dem linken, Kanal C auf dem rechten und Kanal B auf beiden Kanälen ausge-

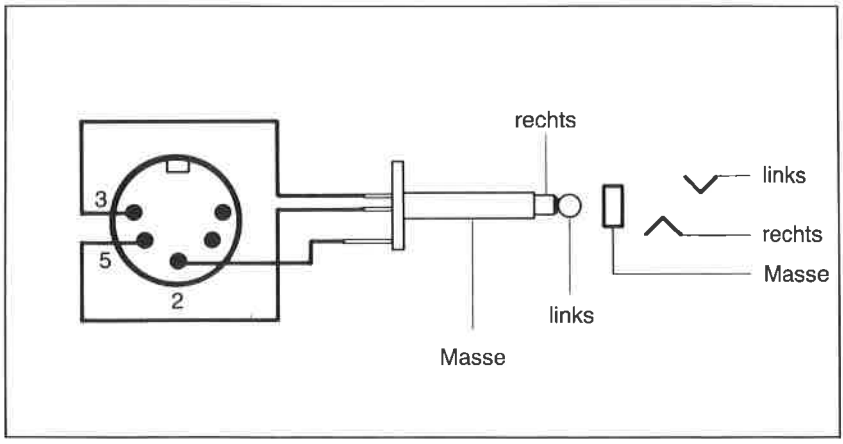



*Abb. 1.6: Anschluß des CPC an eine Stereoanlage mit DIN-Anschluß*

geben wird. Abb. 1.6 zeigt Ihnen, wie Sie einen – in jedem Kaufhaus erhältlichen – Koaxialstecker mit 3,5 mm Durchmesser verdrahten und mit den Anschlußpins eines Stereo-DIN-Steckers verbinden müssen, um in den vollen Genuß der musikalischen Eigenschaften Ihres Heimcomputers zu kommen.

Bevor Sie nachfolgend einen neugierigen Blick in das Innere des Gehäuses werfen können, noch ein Wort zum Anschluß eines Spielereglers, üblicherweise als Joystick bezeichnet. Anschließen können Sie neben den von Schneider lieferbaren ohne Probleme jene Spieleregler, die für ATARI-Computer und zu diesen kompatiblen Systemen hergestellt wurden. Die Anschlußbelegung der computerseitigen Buchse zeigt Abb. 1.7.

### Ein Blick ins Innere

Im allgemeinen ist für jene Zeit, innerhalb der die Garantie gilt, der Blick in das Innere des Computergehäuses riskant, da bei einem eventuell auftretenden Schaden die Gewährleistung des Herstellers erlischt. Sie sollen deshalb nachfolgend die Gelegenheit erhalten, einen (Garantie-) unschädlichen Blick in das Innere des Gehäuses zu werfen. Es beherbergt neben der Tastatur und dem Kassettenrecorder eine mit vielen mikroelektronischen Bauelementen bestückte Platine (Abb. 1.8). Die zur elektrischen Verbindung der Bauelemente dienenden Leitungen sind beidseitig in Form flacher Kupferbahnen auf einer Kunststoffplatine aufgebracht. Deren silbriges Aussehen kommt dadurch zustande, daß sie mit Lötzinn versehen sind. Die gesamte Platine wird in der Fachsprache als Mutterplatine (motherboard) bezeichnet.

Die wichtigsten Bauelemente sind als höchstintegrierte Schaltungen ausgeführt. Zu sehen sind nur deren Gehäuse mit den an den Seiten herausgeführten Anschlußstiften. Sie sind – wie alle anderen Bauelemente auch

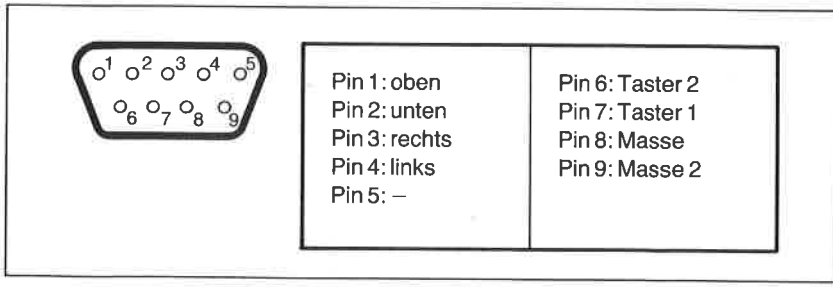



*Abb. 1.7: Der Joystick-Anschluß des Schneider CPC 464/664*

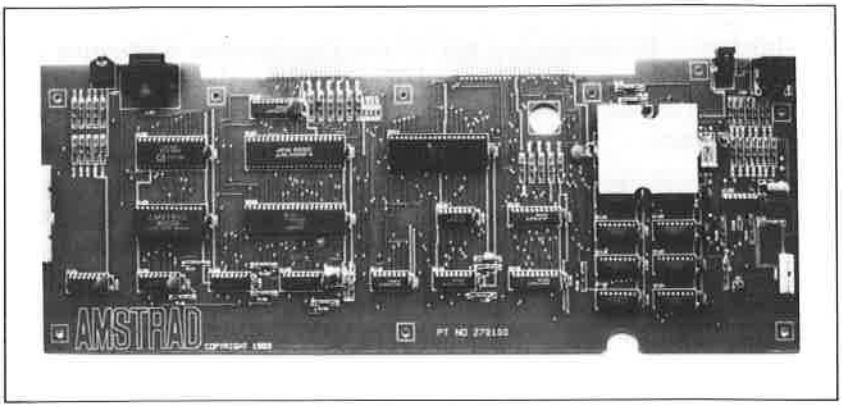



*Abb. 1.8: Die Mutterplatine des CPC 464*

– in die Mutterplatine eingesteckt und verlötet. Das wichtigste Bauelement ist der Mikroprozessor des Typs Z80A. Sie werden ihn und seine Funktion in Kapitel 6 noch eingehend kennenlernen.

Bei den 8 integrierten Schaltkreisen in der rechten unteren Ecke handelt es sich um das „Gedächtnis" des Computers in Form von Speicherbausteinen (RAM) mit einem Fassungsvermögen von über 65000 verschiedenen Datenwerten. Informationen können in diesen Speichern nur so lange abgelegt werden, wie der Computer eingeschaltet und mit Spannung versorgt wird. Die „Seele" Ihres Schneider CPC ist in der großen Schaltung links unten enthalten. Es handelt sich dabei um den sogenannten Festwertspeicher (ROM). In ihm sind unlösch- und jederzeit abrufbar jene Programme und Informationen abgelegt, die der CPC benötigt, um sofort nach dem Einschalten mit dem Benutzer Kontakt aufnehmen zu können. Er tut dies mittels des bereits erwähnten Sichtgerätes, auf dem er seine Meldungen in Form von Text abbildet. Sie selbst bedienen sich für die Befehlseingabe der Tastatur, über deren Funktion und Zeichenbelegung Sie im nachfolgenden Kapitel noch wichtige Informationen erhalten.

Die bisher noch nicht erwähnten anderen integrierten Bausteine auf der Mutterplatine (ein SOUND-Chip, ein Ein-/Ausgabechip, der Displaykontroller sowie das Gate-Array mit den anwenderspezifischen Logiken) erfüllen eine Menge höchst komplizierter Aufgaben, über die Sie sich als Anwender im allgemeinen jedoch nicht zu kümmern brauchen. Mehr über die Funktion des Computers und seines Prozessors finden Sie bei Interesse im Kapitel 6.

# Kapitel 2
# Die ersten Kontakte

## Übersicht

Mit einem Computer verhält es sich so ähnlich wie mit einem neu gekauften Kraftfahrzeug. Erst wenn man die wichtigsten Handgriffe in seinem Fahrzeug „im Schlaf" beherrscht, wird die Benutzung des Wagens zu einem Vergnügen. Das ist bei einem Computer ebenso, wenn man einmal davon absieht, daß die Übungsphase erheblich weniger riskant ist als bei einem Fahrzeug im fließenden Verkehr.

Jeder Computer bietet dem Benutzer eine Standardumgebung für die Arbeit mit dem System und dessen Peripherie. Die Auslegung dieser *Benutzerschnittstelle* wird weitgehend durch die von Hard- und Firmware festgelegten Arbeitsmöglichkeiten bestimmt. Gemeint sind hier Dinge wie die Art der Abbildung von Text und Grafik auf dem Sichtgerät, der Umgang mit der Tastatur als zentrales Eingabemedium oder die Möglichkeiten, Programme auf Datenträgern zu sichern bzw. von diesen abzurufen.

In diesem Kapitel erhalten Sie eine Übersicht über die verschiedenen Arten der Bildschirmdarstellung, der Definition von Vorder- und Hintergrundfarben sowie dem Aufruf von Sonderzeichen über die Tastatur.

## Die Organsiation der Bildschirmdarstellung

Der Schneider CPC unterscheidet sich im Hinblick auf die Abbildung von Informationen auf dem Bildschirm von den meisten seiner Konkurrenten auf dem Markt dadurch, daß ohne nennenswerte Probleme Text und Grafik gleichzeitig abgebildet werden können.

Eine gute Ausnutzung der systemgegebenen Möglichkeiten ist jedoch nur dann möglich, wenn die Mechanismen zur Darstellung von schriftlichen und grafischen Informationen sowie der Definition der verschiedenen Farben gut beherrscht werden. In diesem Abschnitt finden Sie vornehm-

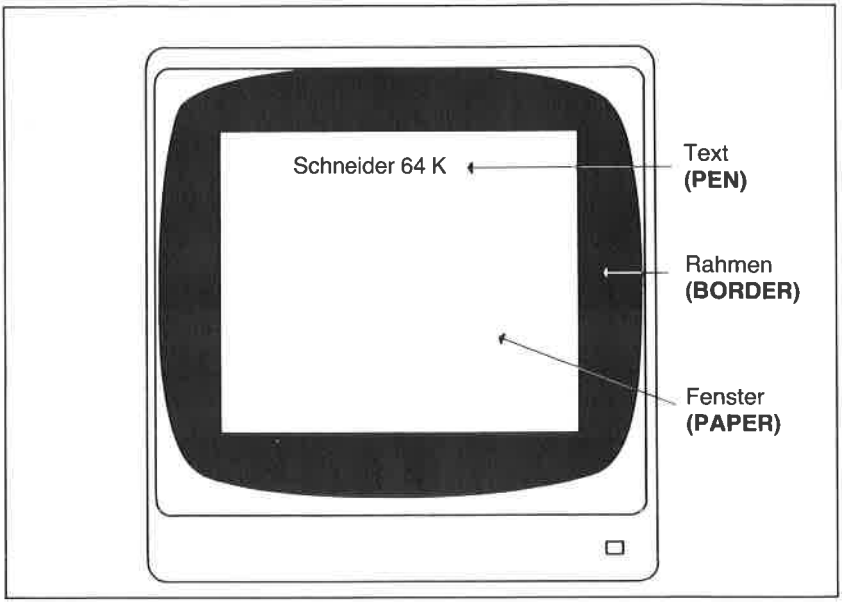



*Abb. 2.1: Die Aufteilung des Sichtfeldes auf dem Monitor*

lich Hinweise zur Textausgabe. Falls Sie sich speziell für das Themengebiet der grafischen Abbildung interessieren, können Sie sich in Kapitel 4 näher darüber informieren.

Der Bildschirm des Sichtgerätes ist in ein Informationsfeld, das im weiteren Verlauf einfach als *Fenster* (window) bezeichnet wird, und einen *Rahmen* (border) aufgeteilt (Abb. 2.1). Das Fenster wird sowohl für die Darstellung von Texten wie auch von Grafik verwendet. Die höchstmögliche Anzahl von Bildpunkten (Pixel) beträgt in horizontaler Richtung 640 und in vertikaler Richtung 200.

Je nach Betriebsart ist die Anzahl von Bildpunkten bzw. darstellbaren Textzeichen unterschiedlich. Definiert werden können vom Anwender die Betriebsarten MODE 0, MODE 1 und MODE 2, die nachfolgend näher erläutert werden.

*MODE 0:*

Die Eingabe von MODE 0 unmittelbar über die Tastatur oder innerhalb eines Programms (siehe auch Befehlszusammenstellung in Kapitel 3) schaltet die Darstellung auf 20 Zeichen pro Textzeile bei insgesamt 25

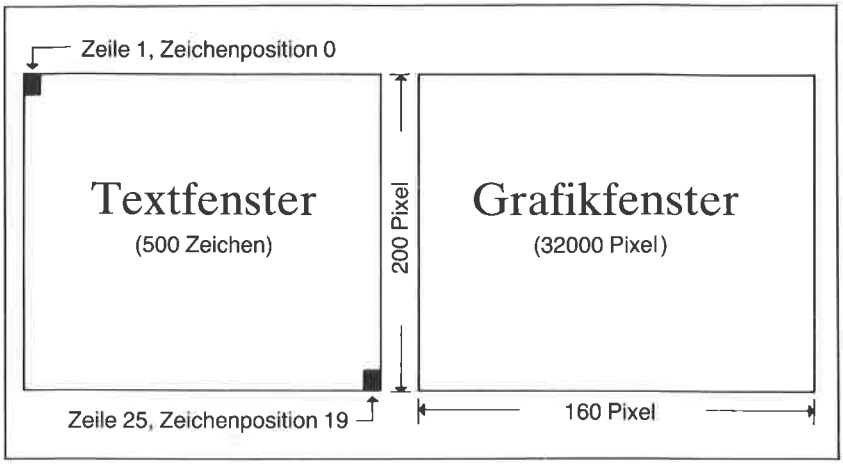



*Abb 2.2: Bildschirmorganisation in der Betriebsart MODE 0*

Zeilen und einer Grafikauflösung von 160 Pixel horizontal und 200 Pixel vertikal um (Abb. 2.2).

Insgesamt sind somit 500 Textzeichen bzw. 32000 Bildpunkte gleichzeitig darstellbar. Für die Abbildung stehen in dieser Betriebsart maximal 16 voneinander unterscheidbare aktuelle Farbwerte n zur Verfügung (n = 0 bis 15), die aus einer Palette von insgesamt 27 physikalischen Farbwerten f (f= 0 bis 26) ausgewählt werden können (Abb 2.3). Es sind zwar formal 32 physikalische Farbwerte vereinbar, die über den Wert 26 hinausgehenden führen jedoch zu Wiederholungen.

Mit dem Befehl BORDER f können alle möglichen Farbwerte f der Farbpalette ($0 \leq f \leq 31$) aufgerufen werden. Es ist so möglich, den Bildschirmrahmen direkt auf jede der in Abb. 2.3 angegebenen Farben zu setzen. Die Farbe der Schrift wird dagegen durch den Befehl PEN n festgelegt. Im Gegensatz zu BORDER können für n nur Werte zwischen 0 und 15, also 16 verschiedene Vordergrundfarben vereinbart werden. Beim Systemstart sind den für n erlaubten Werten Standardfarben aus der Farbpalette zugeordnet. Für die Betriebsart MODE 0 ist diese Zuordnung in der folgenden Abb. 2.4 angegeben.

Sie gilt auch für den Befehl PAPER n, mit dessen Hilfe die *Hintergrundfarbe* des Fensters gewählt werden kann. Die Vereinbarung von PEN 11 führt somit zu einer Abbildung von Schrift in der Farbe Rosa. PAPER 5 schaltet die Hintergrundfarbe des Fensters auf die Farbe Schwarz.

| Farbnr. | Farbe | Farbnr. | Farbe |
|---|---|---|---|
| 0 | Schwarz | 16 | Rosa |
| 1 | Blau | 17 | Pastellmagenta |
| 2 | Hellblau | 18 | Hellgrün |
| 3 | Rot | 19 | Seegrün |
| 4 | Magenta | 20 | helles Blaugrün |
| 5 | Hellviolett | 21 | Limonengrün |
| 6 | Hellrot | 22 | Pastellgrün |
| 7 | Purpur | 23 | Pastellblaugrün |
| 8 | helles Magenta | 24 | Hellgelb |
| 9 | Grün | 25 | Pastellgelb |
| 10 | Blaugrün | 26 | Leuchtendweiß |
| 11 | Himmelblau | 27 | Weiß (wie 13) |
| 12 | Gelb | 28 | Purpur (wie 7) |
| 13 | Weiß | 29 | Pastellgelb (wie 25) |
| 14 | Pastellblau | 30 | Blau (wie 1) |
| 15 | Orange | 31 | Seegrün (wie 19) |

*Abb. 2.3: Die 32 Farben des CPC*

Sofern Sie ein Farbsichtgerät haben, sollten Sie das nachfolgend angegebene Programm ausprobieren, um ein Gefühl für die Wirkung der Farbvereinbarung zu erhalten. Bei Sichtgeräten mit monochromer Informationsabbildung ergeben sich je nach Toleranz der für die Bildsynthese verwendeten Widerstände unterschiedliche Wirkungen. Das Programm bewirkt nichts anderes, als daß der Text „SCHNEIDER CPC“ vierundzwanzigmal untereinander im Textfenster ausgegeben wird. Jeder der Buchstaben wird dabei in einer anderen Farbe (Graustufe) dargestellt. Im Mode 0 werden Sie das S des Wortes SCHNEIDER allerdings vergeblich suchen. Da es dieselbe Farbe hat wie der Hintergrund des Fensters, ist es nicht sichtbar.

| PAPER#<br>PEN# | Farbcode | physikalische<br>Farbe aus der Palette |
|---|---|---|
| 0 | 1 | Blau |
| 1 | 24 | Hellgelb |
| 2 | 20 | helles Blaugrün |
| 3 | 6 | Hellrot |
| 4 | 26 | Leuchtendweiß |
| 5 | 0 | Schwarz |
| 6 | 2 | Hellblau |
| 7 | 8 | helles Magenta |
| 8 | 10 | Blaugrün |
| 9 | 12 | Gelb |
| 10 | 14 | Pastellblau |
| 11 | 16 | Rosa |
| 12 | 18 | Hellgrün |
| 13 | 22 | Pastellgrün |
| 14 | 1,24 | Blau/Hellgelb (blinkend) |
| 15 | 16,11 | Rosa/Himmelblau (blinkend) |

*Abb. 2.4: Standardfarbzuordnung in der Betriebsart MODE 0*

```
10 CLS
20 A$="SCHNEIDER CPC"
30 FOR J=1 TO 24
40 FOR I=1 TO LEN(A$)
50 PEN I-1:PRINT MID$(A$,I,1);
60 NEXT I
70 PRINT:NEXT J
80 END
```

Geben Sie nun zur Probe unmittelbar über die Tastatur PAPER 2 ein, und starten Sie das Programm erneut. Jetzt ist der dritte Buchstabe des Wortes SCHNEIDER verschwunden.

Die Vereinbarung von Farbwerten ist also recht durchsichtig, wenn das Zuordnungsprinzip nach Abb. 2.4 beachtet wird. Am Ende dieses Kapitels finden Sie eine allgemeine Übersicht, die den Sachverhalt noch einmal im einzelnen verdeutlicht.

*MODE 1:*

Die Betriebsart MODE 1 wird unmittelbar nach dem Kaltstart des Schneider CPC, d. h. nach dem Einschalten oder dem Auslösen eines RESET über die Tastenkombination CTRL SHIFT und ESC aktiviert. Entsprechend der nachfolgenden Abb. 2.5 lassen sich 25 Zeilen zu je 40 Zeichen bzw. 320 Pixel horizontal und 200 Pixel vertikal abbilden.

Die höhere Auflösung im Text- wie auch im Grafikbereich muß durch eine Einschränkung der Farbdarstellung erkauft werden. Aus der bereits erwähnten Palette von 32 Farben sind nur noch *vier* gleichzeitig verwendbar. Sofern keine spezielle Zuordnung über den INK-Befehl vorgenommen wurde, entspricht die Farbvereinbarung nach dem Systemstart den in Abb. 2.6 angegebenen Standardwerten. Auswahl des jeweils links stehenden Farbparameters im Zusammenhang mit einem PEN-Befehl führt jeweils zu der rechts außen angegebenen Farbe. Das heißt, daß im Nor-

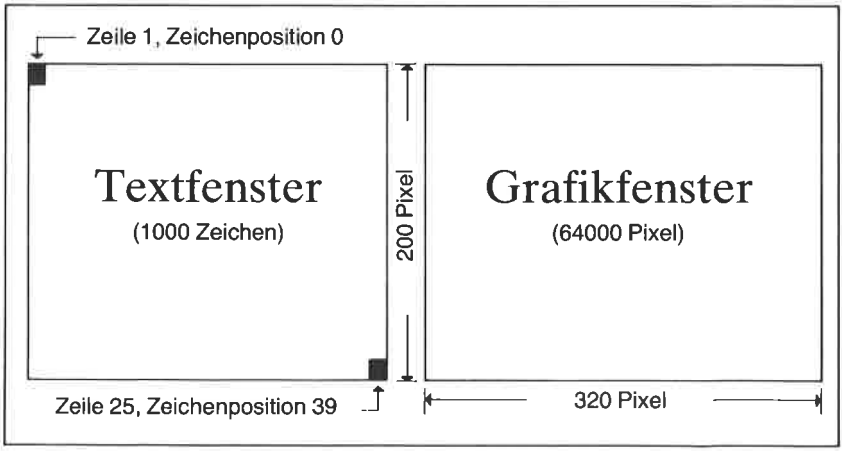



*Abb. 2.5: Bildschirmorganisation in der Betriebsart MODE 1*

| Farbparameter für PAPER# PEN# | Farbcode | physikalische Farbe aus der Palette |
|---|---|---|
| 0,4,8,12 | 1 | Blau |
| 1,5,9,13 | 24 | Hellgelb |
| 2,6,10,14 | 20 | Blaugrün (hell) |
| 3,7,11,15 | 6 | Hellrot |

*Abb. 2.6: Standardfarbzuordnung in der Betriebsart MODE 1*

malfall beispielsweise sowohl PEN 0 wic auch PEN 4, PEN 8 und PEN 12 zur einer Schriftabbildung in der Farbe blau (Farbcode 1 der Farbpalette) führen, usw.

Unter Kontrolle des BASIC-Interpreters steht zur Vereinbarung abweichender Farbzuordnungen der Befehl

INK <n>,<$f_1$>,<$f_2$>

zur Verfügung. Mit seiner Hilfe kann dem aktuellen Farbparameter n für PEN und PAPER jeder beliebige Wert $f_1$ aus der Farbpalette zugeordnet werden. Werden sowohl $f_1$ wie auch $f_2$ angegeben, wechselt n zwischen den beiden Farbwerten.

Eine nochmalige Anhebung der Text- und Grafikauflösung ist in der nachfolgend erläuterten Betriebsart MODE 2 möglich. Es sind dann jedoch nur noch *zweifarbige* Abbildungen darstellbar.

*MODE 2:*

In der Betriebsart MODE 2 können 25 Zeilen zu je *80 Zeichen* und Grafiken mit einer Auflösung von 640 Punkten in horizontaler sowie 200 Punkten in vertikaler Richtung abgebildet werden, wie dies in Abbildung 2.7 schematisch dargestellt ist.

### Mehr über die Definition und Zuordnung von Farben

Besonders für den Erstanwender eines Schneider CPC 464/664 bereitet die zuvor bereits erläuterte Definition und Vereinbarung von Farben sowie deren Zuordnung zu den aktuell darstellbaren Farbwerten für die Befehle PEN und PAPER immer wieder Schwierigkeiten. Schauen wir uns deshalb diesen Problemkreis noch einmal etwas ausführlicher an.

Leicht verständlich wird alles, wenn Sie sich merken, daß für die aktuell in den PEN- und PAPER-Befehlen zu vereinbarenden Farben immer nur eine beschränkte Auswahl aus einer Palette von 26 (32) Farben verfügbar ist. In der Betriebsart MODE 0 sind dabei *sechzehn* Farben, in der Betriebsart MODE 1 *vier* und in der Betriebsart MODE 2 nur *zwei* Farbwerte gleichzeitig darstellbar.

Die in der Farbpalette enthaltenen Farben sind mit Kennummern f zwischen f = 0 und f = 31 versehen. Die über die Nummer 26 hinausgehenden Werte führen nur zu Wiederholungen anderer Farbwerte. Sie spielen demnach für den praktischen Betrieb in aller Regel keine Rolle (Abb. 2.9).

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Abb. 2.10: Standardfarbbelegung in den Betriebsarten MODE 0 bis 2*

#### *Veränderung der Farbzuordnung durch den INK-Befehl*

Die übliche Zuordnung der Farben aus der Farbpalette zu den aktuellen Farbparametern nach der in Abb. 2.10 angegebenen Tabelle kann mittels des Befehls

INK <Farbnummer n>,$f_1$,$f_2$

jederzeit geändert werden. Hierbei ist zu beachten, daß in den entsprechenden Textmodi keinesfalls mehr als die bereits erwähnte maximale Anzahl gleichzeitig darstellbarer Farben verwendet werden kann. Der INK-Befehl ordnet dem mit n angegebenen aktuellen Farbwert die mit f vereinbarte Farbe aus der Farbpalette zu. So wird beispielsweise durch die Anweisung INK 1,21 dem aktuellen Farbparameter 1 die Farbe Limonengrün zugeordnet. Da im Normalfall zunächst der Hintergrund auf den aktuellen Farbwert 0 und die Zeichenabbildung über den aktuellen Farbparameter 1 erfolgt, bleibt als Folge dieses Befehls die Hintergrundfarbe unverändert, und die Schrift ändert sich von Gelb nach Limonengrün. Die Anweisung INK 0,7 führt folgerichtig dazu, daß das gesamte Fenster sofort Purpurrot eingefärbt wird.

Das nachfolgend angegebene Programm führt Ihnen alle Farben (Graustufen) mit Angaben über die Parameter vor.

```
5 REM *** Darstellung aller Farbkombinationen ****
10 MODE 1
20 WINDOW #0,1,40,1,12
30 WINDOW #1,5,35,13,13:PEN #1,2:PAPER #1,3:INK 2,0:INK
 3,13:CLS #1
40 WINDOW #2,1,40,14,25
50 WINDOW #3,30,40,1,1::PEN #3,2:PAPER #3,3:INK 2,0:INK
 3,13:CLS #3
60 LOCATE #0,5,5:PRINT #0,"Dies ist ein Beispiel !!!":L
OCATE #2,5,5:PRINT #2,"Dies ist ein Beispiel !!!"
70 PAPER #0,0:PAPER #2,0:PEN #0,1:PEN #2,1
80 PRINT #1,TAB(7);"PAPER: ";TAB(18);"PEN: "
90 FOR J=0 TO 26
100 FOR I=0 TO 26
110 INK 0,J
120 INK 1,I
130 BORDER J
140 LOCATE #1,14,1:PRINT #1,J:LOCATE #1,23,1:PRINT #1,I
150 LOCATE #3,3,1:PRINT #3,"Taste !!":CALL &BB18
160 PRINT#3
170 NEXT I
180 NEXT J
190 INK 0,1:PAPER 0
200 INK 1,24:PEN 1
210 BORDER 1
220 MODE 1
```

### *Der Transparentmodus*

Eine Ausnahme bei der zuvor erläuterten Art der Abbildung von Hinter- und Vordergrundfarben bildet die Farbdefinition im sogenannten Transparentmodus.

Dieser Modus wird durch die Anweisung

```
PRINT CHR$(22)+CHR$(1)
```

eingeschaltet und durch die Anweisung

```
PRINT CHR$(22)+CHR$(0)
```

wieder ausgeschaltet.

Im Normalfall erfolgt die Zeichenausgabe grundsätzlich in der Weise, daß an der Ausgabeposition das Zeichenfeld mit der definierten Hintergrundfarbe gefüllt und dann das Zeichen in der vereinbarten Vordergrundfarbe abgebildet wird. Bereits auf dem Bildschirm vorhandene Informationen werden auf diese Weise gelöscht. Beim Transparentmodus wird nur die Vordergrundinformation (das Zeichen) auf die bereits vorhandene Hintergrundinformation geschrieben. Das bedeutet, daß nur ein Teil der Bildschirminformation an der Ausgabestelle zerstört wird. Auch hier ist ein Beispiel besser als tausend Worte. Geben Sie deshalb zunächst einmal das nachfolgend angegebene Programm ein, und starten Sie es anschließend.

```
5 REM **** Transparent-Modus ****
10 MODE 1
20 LOCATE 12,12
30 PEN 3:PRINT STRING$(19,CHR$(140))
40 PRINT CHR$(22)
50 LOCATE 12,12
60 PEN 1:PRINT "Hallo, wie geht's ?"
70 PRINT CHR$(22);CHR$(0)
80 END
```

Ändern Sie nach eigenen Vorstellungen die wichtigen Vereinbarungen ab, und testen Sie auf diese Weise, ob Sie das Prinzip der Farbwahl wie auch des Transparentmodus verstanden haben.

### Fenstertechniken

Nach dem Systemstart wird der gesamte vom Rahmen umgebene Bereich des Bildschirms als Ausgabefenster für die Darstellung von Text und Grafik benutzt.

Über den BASIC-Befehl

```
WINDOW [#<Ausgabefenster>,] <linker Rand>,<rechter Rand>,
<oberer Rand>,<unterer Rand>
```

können bis zu 8 andere Fensterbereiche definiert werden, die durch Nummern mit einem vorangestellten Doppelkreuz (#) kenntlich gemacht werden können. Die Randeinstellungen beziehen sich dabei auf Zeichenfelder. Das bedeutet, daß die Vereinbarungen abhängig von der Betriebsart sind. Das beim Systemstart vereinbarte Fenster mit 25 Textzeilen zu je x Zeichen pro Zeile (x=20 im MODE 0, x= 40 im MODE 1, x= 80 im MODE 2) hat die Nummer #0. In der Betriebsart MODE 1 mit den maximal 40*25 darstellbaren Zeichen wird beispielsweise durch die Vereinbarung

```
WINDOW #1,8,22,6,13
```

ein Ausgabefenster mit der Kennummer #1 für die Textdarstellung definiert, wie dies in Abbildung 2.11 gezeigt ist.

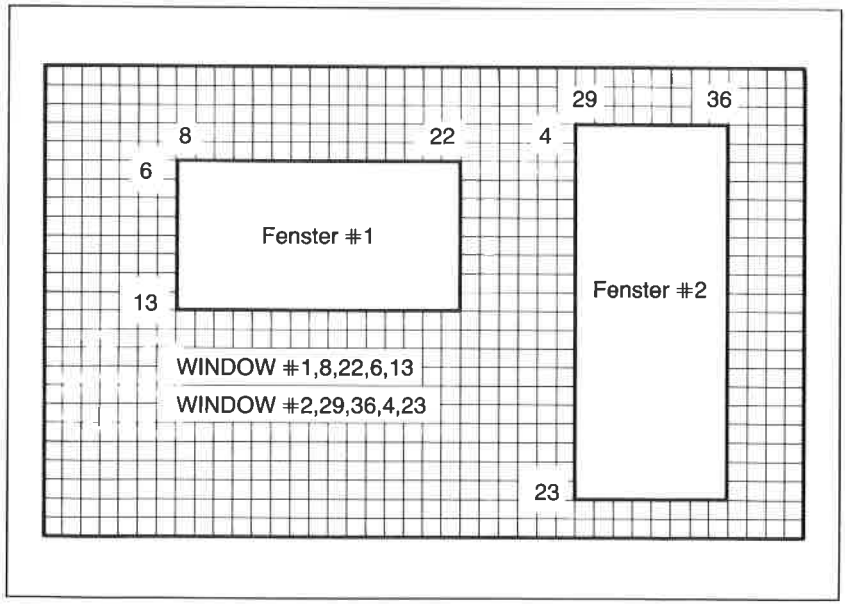



*Abb. 2.11: Definition von Textfenstern*

Textausgabebefehle wie beispielsweise WRITE oder PRINT bieten die Möglichkeit zur direkten Anwahl derart vereinbarter Fenster. So wird mittels des Befehls PRINT #1,"SCHNEIDER CPC 464" die angegebene Textkonstante nicht mehr im Fenster #0, sondern im zuvor vereinbarten Fenster #1 ausgegeben.

Zu den BASIC-Befehlen, die eine direkte Auswahl des Zielfensters erlauben, gehören: CLS, INPUT, LINE INPUT, LOCATE, PAPER, PEN, POS, TAG, TAGOFF, VPOS, WINDOW, WINDOW SWAP, WRITE PRINT und PRINT USING.

Wird ein vom Wert #0 abweichender Fensterwert in den genannten Befehlen nicht vereinbart, erfolgt die Ausgabe immer im aktiven Fenster. Dieses ist bei Systemstart dasjenige mit der Nummer #0.

Über den Befehl

WINDOW SWAP <#aktuelles Fenster>,<#Zielfenster>

kann das aktuelle Textausgabefenster mit einem vorher vereinbarten anderen Zielfenster ausgetauscht werden. So definiert beispielsweise der Befehl WINDOW 0,1 das unter der Nummer 1 angegebene Fenster als aktuelles Ausgabefenster. Logisch wird die Zuweisung also durch die symbolische Operation

Nummer des aktuellen Fensters → Nummer des neuen Zielfensters

dargestellt. Das Fenster mit der ursprünglichen Kennummer #0 besitzt nunmehr die Nummer #1 und umgekehrt. Es ist durch sorgfältige Wahl von Fensterbereichen und deren programmtechnische Ansteuerung beispielsweise möglich, unterschiedliche Programmausgaben in verschiedenen Fenstern zur Anzeige zu bringen.

Neben den 8 Textfenstern kann noch ein Grafikfenster gesetzt werden, das über den BASIC-Befehl ORIGIN vereinbart wird. Hierüber finden Sie nähere Informationen im Kapitel 4.

Zum Schluß dieses Kapitels sollen Sie noch einige Tips über die Definition von Tastenfunktionen und die Veränderung der Tastencodes erhalten.

### Tasten und Tastenfunktionen

Die insgesamt 57 Tasten der Haupttastatur, die 12 Tasten des numerischen Tastenblocks und die 5 Tasten für die Cursorsteuerung sind durch zwei wichtige Angaben, dem Tastencode und dem zugehörigen ASCII-Code, gekennzeichnet. In der folgenden Abb. 2.12 finden Sie eine schematische Darstellung der Tastatur des CPC 464. Sie enthält sowohl die Zeichenbelegung wie auch die Tastencodes. Im Anhang R dieses Buches finden Sie außerdem eine vollständige Zusammenstellung aller Tastencodes zusammen mit den zugehörigen ASCII-Codes der Zeichen im Normal-, SHIFT- und CTRL-Modus, die sich für Neudefinitionen von einzelnen Tasten als sehr nützlich erweisen wird.

Die Zeichenbelegung der einzelnen Tasten wird beim Systemstart in einen Teil des Schreib-/Lesespeichers geladen und von dort seitens des Videocontrollers ausgelesen. Die Zeichen sind damit durch den Anwender veränderbar.

Unter der Kontrolle des Interpreters steht hierfür der Befehl

```
KEY DEF <Tastencode>,<Wiederholfunktion>
[,<normal>[,<mit Shift>[,<mit Ctrl>]]]
```

zur Verfügung. Die neuzubelegende Taste wird durch ihren Tastencode (siehe Abb. 2.12) gekennzeichnet. Für den nachfolgenden Parameter „Wiederholfunktion" sind die Werte 0 (abgeschaltet) und 1 (eingeschaltet) erlaubt. Unter den letzten drei Parametern werden die ASCII-Codewerte für die drei möglichen Tastenbelegungsfälle

1. Taste einzeln
2. Taste zusammen mit Shift
3. Taste zusammen mit Ctrl

vereinbart.

Ein brauchbares Beispiel soll diesen Sachverhalt ein wenig erhellen. Der Übersicht in Abb. 2.13 entsprechend ist für die Zuordnung international unterschiedlicher ASCII-Zeichen aus Kompatibilitätsgründen nur eine festgelegte Auswahl von Codes üblich. Es ist sehr sinnvoll, sich an diese Vereinbarungen zu halten, da Drucker mit wählbaren Zeichensätzen die oben gezeigte Codebelegung berücksichtigen. Der Umlaut Ä ist nach dieser Übereinkunft dem ASCII-Code &5B (dezimal 91) zugeordnet. Im USASCII-Zeichensatz entspricht dies dem Zeichen für die eckige Klammer links ([). Das ä dagegen entspricht dem Code &7B (dezimal 123). Dies ist auf der Tastatur des Schneider CPC das Zeichen {, also die

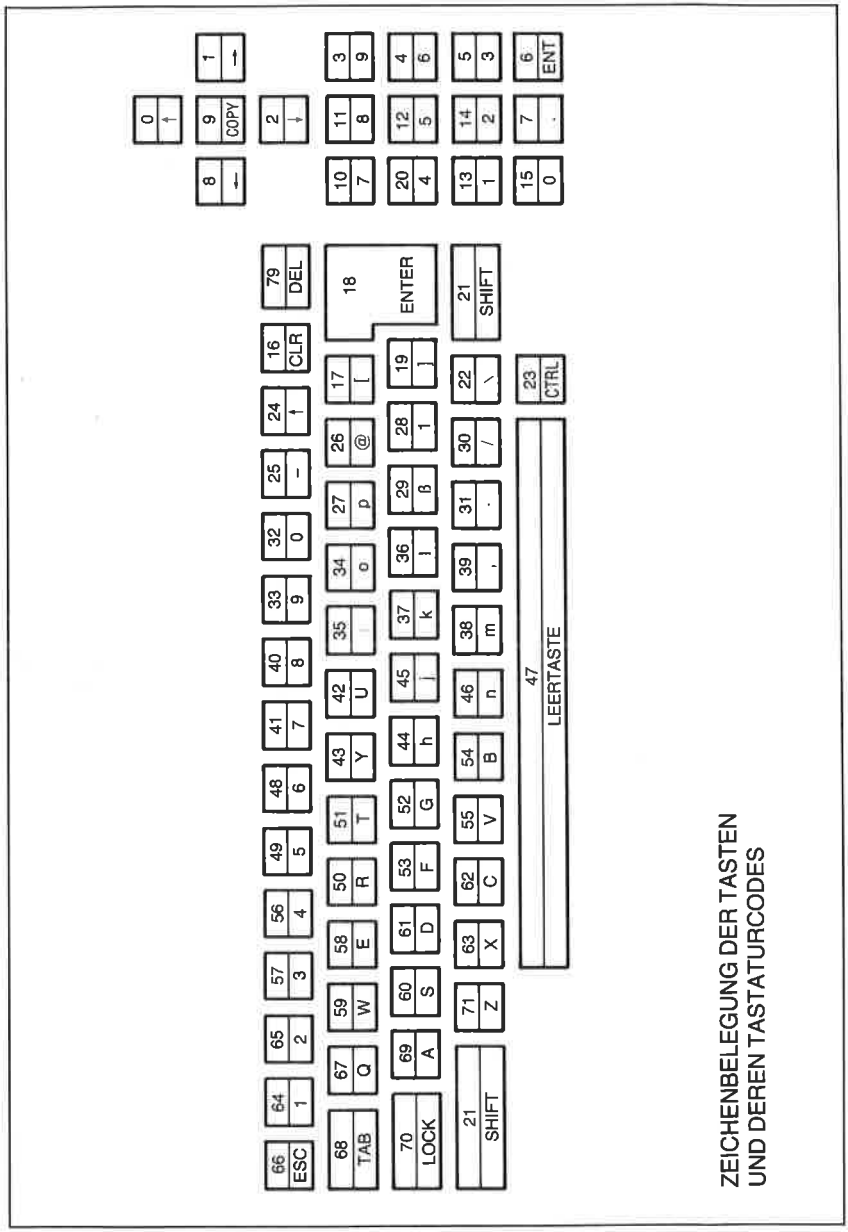



*Abb. 2.12: Zeichenbelegung und Tastencodes der Tastatur*

geschweifte Klammer links. Wenn Sie einfach durch den nachfolgend noch näher erläuterten Befehl SYMBOL den entsprechenden Codezeichen die Matrixdefinitionen für die Umlaute zuordnen, kommen Sie in die mißliche Lage, daß der Abruf von Groß- und Kleinschreibung genau umgekehrt funktioniert, als dies bei den anderen Zeichen üblich ist. Hier hilft auf einfache Weise der zuvor bereits vorgstellte KEY DEF-Befehl weiter. Wir müssen nur die Codebelegung der mit dem Zeichen [ belegten Taste umkehren. Da nach Abb. 2.12 die Taste den Positionscode 17 besitzt, lautet der entsprechende Befehl demnach

```
KEY DEF 17,1,&7B,&5B
```

Entsprechende Maßnahmen müssen Sie natürlich auch für die anderen Zeichen ergreifen. Bevor Sie dies tun (Sie finden etwas später ein Beispielprogramm hierzu), wollen wir uns noch einen weiteren Befehl ansehen, der es Ihnen ermöglicht, die USASCII-Zeichen gegen beliebige andere auszutauschen. Der Befehl lautet

```
SYMBOL <ASCII-Code>,<Liste von Matrixelementen>
```

Wir benutzen nachfolgend den Befehl dazu, um in einem Beispiel das USASCII-Zeichen | gegen den Umlaut ö auszutauschen. In Abbildung 2.14 sehen Sie anhand eines Schemas, wie die beiden Zeichen innerhalb einer Matrix von 8*8 Bildpunkten aussehen.

| hex. Code | dez. Code | USA | Frankreich | Deutschland | England |
|---|---|---|---|---|---|
| 23 | 35 | # | # | # | £ |
| 40 | 64 | @ | à | § | @ |
| 5B | 91 | [ | ° | Ä | [ |
| 5C | 92 | \ | ç | Ö | \ |
| 5D | 93 | ] | § | Ü | ] |
| 7B | 123 | { | é | ä | { |
| 7C | 124 | \| | ù | ö | \| |
| 7D | 125 | } | è | ü | } |
| 7E | 126 | ~ | " | ß | ~ |

*Abb. 2.13: Nationale Varianten des ASCII-Zeichensatzes und deren Codes*

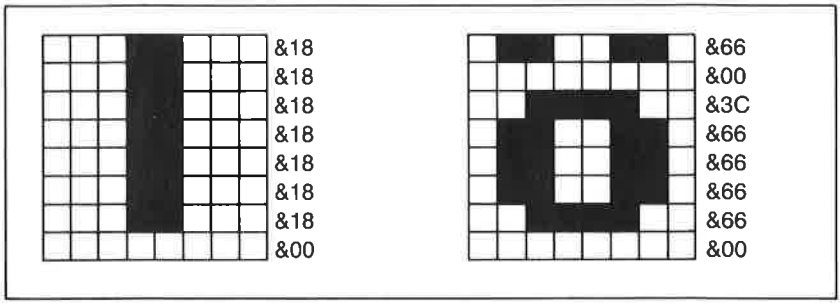



*Abb. 2.14: Duale Verschlüsselung der Zeichen* | *und ö*

Jeder horizontalen Matrixzeile ist im Speicher des CPC ein Byte zugeordnet. Einem gesetzten Bildpunkt entspricht dabei eine logische Eins. Um das Zeichen | im RAM abzulegen, brauchen wir somit 8 aufeinanderfolgende Bytes mit den dezimalen Werten 24, 24, 24, 24, 24, 24, 24, 0 bzw. hexadezimal 18, 18, 18, 18, 18, 18, 18, 0. Wenn wir dagegen das Zeichen | durch den Umlaut ö ersetzen, so lautet die Bytefolge in dezimaler Schreibweise 102, 0, 60, 102, 102, 102, 60, 0 bzw. 66, 0, 3C, 66, 66, 66, 3C, 0 in hexadezimaler Darstellung. Diese 8 Bytes bilden jeweils die im SYMBOL-Befehl anzugebenden Elemente der Matrixliste. Der Befehl zur Änderung des unter dem ASCII-Code &7C (dezimal 124) stehenden Zeichens lautet demnach:

```
SYMBOL 124,102,0,60,102,102,102,60,0
```

Im Zusammenhang mit der Änderung von Zeichen ist noch der Befehl

```
SYMBOL AFTER <ASCII-Code>
```

von Bedeutung, der dem System mitteilt, ab welchem Code eine Zeichenänderung vorgenommen werden soll. Sofern Sie nur die Umlaute einführen wollen, genügt der Befehl SYMBOL AFTER 91. Alle Codes ab 91 aufwärts sind so für eine Zeichenänderung zugänglich. Für jedes zur Änderung freigegebene Zeichen wird im Speicher ein Platz von 8 Bytes reserviert. Denken Sie daran, wenn es mit dem Speicherplatz einmal hapern sollte.

Das nachfolgend gezeigte Programm ändert die Tastaturbelegung derart um, daß alle USASCII-Zeichen mit den Codes 91 bis 93 und 123 bis 126 mit deutschen Umlauten sowie dem Zeichen ß belegt werden (letzteres

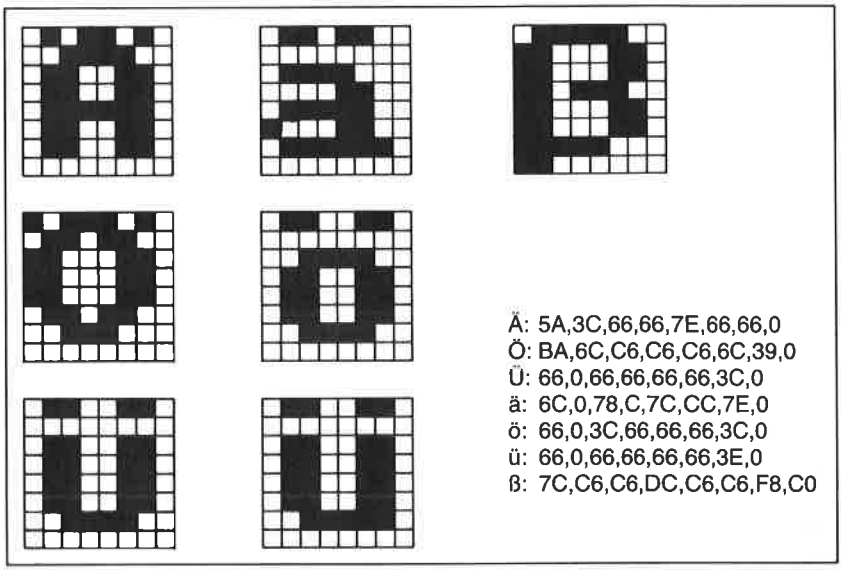



*Abb. 2.15: Mustervorlagen für die Zeichen ä, ö, ü, Ä, Ö, Ü und ß*

liegt dann auf der Taste mit dem Pfeil nach oben). Der korrekte Abruf von Klein- und Großbuchstaben ist durch die KEY DEF-Befehle berücksichtigt.

In der Abb. 2.15 sind die Matrixmuster für die Zeichen ä, ö, ü, Ä, Ö, Ü und ß zu Ihrer Orientierung zusammen mit den zugehörigen Bytes angegeben. Die Abbildung 2.16 erläutert die mit dem Programm erreichte neue Tastenbelegung.

```
5 REM **** Deutsche Umlaute ****
10 KEY DEF 24,1,&7E:    REM ß
20 KEY DEF 19,1,&7D,&5D:REM ü Ü
30 KEY DEF 17,1,&7B,&5B:REM ä Ä
40 KEY DEF 22,1,&7C,&5C:REM ö Ö
50 SYMBOL AFTER 91
60 SYMBOL 91,&5A,&3C,&66,&66,&7E,&66,&66,0
70 SYMBOL 92,&BA,&6C,&C6,&C6,&C6,&6C,&38,0
80 SYMBOL 93,&66,0,&66,&66,&66,&66,&3C,0
90 SYMBOL 123,&48,0,&78,&C,&7C,&CC,&76,0
100 SYMBOL 124,&24,0,&3C,&66,&66,&66,&3C,0
110 SYMBOL 125,&24,0,&66,&66,&66,&66,&3E,0
120 SYMBOL 126,&7C,&C6,&C6,&DC,&C6,&C6,&F8,&C0
```

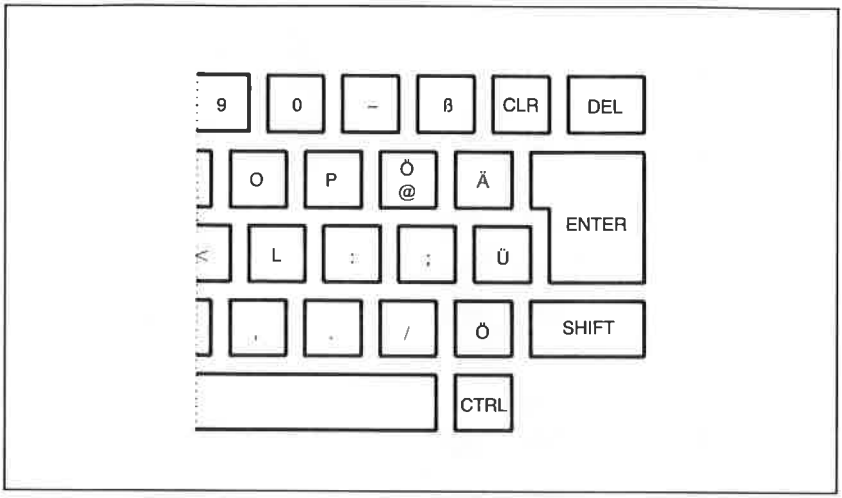



*Abb. 2.16: Die Tastenbelegung mit Umlauten*

Natürlich können Sie durch einfache Änderung des Programms die Schneider-Tastatur auch mit einer Belegung nach DIN versehen. Sie finden diese als Vorbild auf Ihrer Schreibmaschine. Vielleicht versuchen Sie es einmal?!

### Vereinbarung von Funktionstasten

Zum Schluß noch ein Hinweis zur Belegung einzelner Tasten mit Zeichenketten. Hierzu dient der Befehl

KEY <ASCII-Code>,[CHR$(n)+]<Zeichenkette>[+CHR$(n)]

Mit seiner Hilfe können Sie bis zu 32 ASCII-Codes im Codebereich zwischen 128 (&80) und 159 (&9F) recht komplexe Zeichenketten zuordnen. Die Tasten, über die diese ASCII-Codes angerufen werden können, zeigen somit die Wirkung von Funktionstasten. Gerade während der Programmentwicklungsphase sollten Sie von dieser Möglichkeit Gebrauch machen, um immer wieder benötigte Befehlssequenzen per Tastendruck abrufen zu können. Wenn Sie sich die ASCII-Belegung der Tastatur noch einmal genau ansehen, werden Sie feststellen, daß die genannte Möglichkeit ohne zusätzliche Neubelegung mit abweichenden ASCII-Codes zunächst nur für die Tasten des numerischen Tastenblocks zutreffen. Nur

diese erzeugen Codes in dem oben erwähnten Bereich. Mit Ausnahme der mit dem Dezimalpunkt bezeichneten Taste ist darüber hinaus die Codebelegung für alle Modi (also auch SHIFT und CTRL) dieselbe.

Wenn Sie andere Tasten als Funktionstasten benutzen wollen, müssen Sie über den zuvor bereits erläuterten KEY DEF-Befehl zunächst eine Belegung mit ASCII-Codes im oben erwähnten Bereich von 128 bis 159 durchführen. Am günstigsten ist es, wenn Sie nur den über CTRL erreichbaren Teil einer Taste belegen. Sie kommen dann nicht in Konflikt mit der normalen Tastenfunktion.

Die nachfolgend als einfaches Beispiel aufgeführten Definitionen belegen die drei unteren Tasten des numerischen Tastenblocks mit den Funktionen RUN, LIST und MODE 1. Die ASCII-Codes für die Tasten lauten:

```
Taste 0 im Nummernblock: 128
Taste .                : 138
Taste ENTER            : 139
```

Und dies sind die zugehörigen KEY-Befehle:

```
KEY 128,"RUN"+CHR$(13)
KEY 138,"LIST"+CHR$(13)
KEY 139,"MODE 1"+CHR$(13)
```

Die Ergänzung der Zeichenkette durch CHR$(13) führt dazu, daß der Befehl unmittelbar ausgeführt wird. (CHR$(13) entspricht der Wirkung der ENTER-Taste.) Sie können auf diese Art und Weise den gesamten numerischen Tastenblock mit für Sie nützlichen Befehlsmakros belegen (so nennt man so etwas in der Fachsprache). Und – falls Sie sich einmal so gründlich verrannt haben, daß nichts mehr funktioniert: Denken Sie daran, daß durch gleichzeitiges Betätigen der Tasten CTRL, SHIFT und ESC ein Kaltstart ausgelöst werden kann, der Ihren Computer wieder ins Leben zurückruft. Ein eventuell im Speicher stehendes Programm ist allerdings dann verloren.

# Kapitel 3
# Programmieren in BASIC

**Übersicht**

Computer arbeiten, wie Sie in Kapitel 6 nachlesen können, nur mit binären Datenworten, d.h. mit Datenworten, die aus einer Folge von Nullen und Einsen bestehen. Wie diese im einzelnen aussehen müssen, wird bereits bei der Herstellung dem wichtigsten Bauelement eines Computers, dem Mikroprozessor, mit in die Wiege gelegt. Auch die Befehle als Bestandteil eines Programms müssen binär verschlüsselt werden. Diese Art der Befehlsvereinbarung entspricht allerdings nicht dem, was wir als Benutzer eines Systems als angenehm empfinden. Denn wer drückt sich schon gerne in einer unübersehbaren Folge von Nullen und Einsen aus?

Damit Ihnen als Anwender die Zusammenarbeit mit Ihrem Computer so bequem wie eben möglich gemacht werden und ohne Einarbeitung in die Maschinensprache des Mikroprozessors vonstatten gehen kann, werden für alle auf dem Markt angebotenen Computersysteme sogenannte *Übersetzerprogramme* mitgeliefert, die im allgemeinen Bestandteil des Festwertspeichers sind und daher sofort nach dem Start eines Rechners zur Verfügung stehen. Diese Übersetzerprogramme werden je nach Funktionsweise als *Compiler* oder *Interpreter* bezeichnet. Mit ihrer Hilfe werden in einer höheren Sprache formulierte Befehle in die für den Prozessor verständlichen binären Datenfolgen umgesetzt. Die Sprache, in der Sie Ihre Befehle formulieren müssen, wird als *höhere Programmiersprache* bezeichnet. Der üblicherweise aus dem Englischen kommende Wortschatz einer solchen Sprache ist nicht sehr umfangreich und leicht zu erlernen.

Wie alle Systeme der Heimcomputerklasse besitzt auch der CPC 464/664 ein Übersetzerprogramm für die Programmiersprache BASIC. Es gehört zur Kategorie der Interpreter und ist Bestandteil des Festwertspeichers (ROM). BASIC ist eine Abkürzung für *B*eginners *A*ll *S*ymbolic *I*nstruction *C*ode, was soviel wie „Symbolischer Befehlscode für Anfänger"

bedeutet. Im Laufe seiner Entwicklung ist aus dieser zunächst sehr einfachen Programmiersprache ein Werkzeug zur Programmerstellung mit bemerkenswerter Leistungsfähigkeit und großer Befehlsvielfalt geworden. Insbesondere ist die Sprache für jene sehr geeignet, die sich nicht so systematisch und intensiv in die Wissenschaft der Systemprogrammierung einarbeiten, aber möglichst rasch die ersten Erfolge eigener Programmierarbeiten ernten wollen.

Damit Sie sich mit den Regeln und den Sprachelementen dieser Programmiersprache bekannt machen können, finden Sie in diesem recht umfangreichen Kapitel alles, was Sie für den Umgang mit BASIC benötigen. Da Sie nicht notwendigerweise mit der Programmierung und den Regeln der Programmerstellung vertraut sein müssen, erhalten Sie zu Beginn einige grundlegende Informationen zu diesem Thema. Sie können diesen Abschnitt überschlagen, wenn Sie bereits Erfahrungen im Umgang mit Computern besitzen. Ein gesonderter Abschnitt führt Sie in den Vorgang des Editierens und des Korrigierens von Programmtexten ein. Ein sehr umfangreicher Teil ist den über 159 BASIC-Kommandos gewidmet, die – sofern notwendig – anhand kleiner, aber aussagekräftiger Programmbeispiele erläutert werden. Sie finden außerdem einige interessante Informationen zur Frage, wo und in welcher Form BASIC-Programme im Speicher abgelegt werden.

### Was ist ein Programm?

Wenn Sie Ihren Schneider CPC einschalten, meldet er sich bekanntlich mit der Grußmeldung:

```
Schneider 64K Microcomputer (v1)

©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.0 (bzw. BASIC 1.1 beim 664)

Ready
```

Das englische Wort *ready* bedeutet *fertig*. Es wird damit zum Ausdruck gebracht, daß der Computer zur Entgegennahme von Befehlen bereit ist. Das System befindet sich nach der Ausgabe dieser Meldung immer im *Befehls-* oder *Editiermodus*. Das bedeutet nicht anderes, als daß Sie entweder einen Programmtext oder aber auch unmittelbar ausführbare

Kommandos mittels der Tastatur eingeben können. Eines der einfachsten direkten Kommandos ist beispielsweise CLS. Wenn Sie diese drei Zeichen hintereinander eintippen und anschließend die größte vorhandene (oder L auf dem Kopf) Taste mit der Bezeichnung ENTER betätigen, wird der Bildschirm gelöscht, und es wird am oberen linken Rand wiederum die Meldung Ready ausgegeben. CLS als leicht merkbare Abkürzung für *Cl*ear *S*creen gehört zu den Systemkommandos. Diese rufen sofort nach der Eingabe eine Wirkung hervor. Bitte merken Sie sich, daß jede Eingabe über die Tastatur durch Betätigen der mit ENTER beschrifteten Taste abgeschlossen werden muß! Im weiteren Verlauf dieses Buches wird auf diesen Sachverhalt nicht mehr gesondert eingegangen. Ein weiterer Systembefehl ist beispielsweise NEW (neu). Er führt dazu, daß ein im Speicher abgelegtes Programm nicht mehr erreichbar ist. (Es ist zwar nicht gelöscht, jedoch nicht mehr ohne Trick verfügbar.)

Neben diesen einfachen Systembefehlen, die zusammen mit weiteren in dem Abschnitt über die BASIC-Kommandos vorgestellt und erläutert werden, ist ebenfalls die unmittelbare Eingabe komplizierterer Befehle (auch Anweisungen genannt) möglich, wie Sie vermutlich schon im Handbuch gelesen haben. So führt beispielsweise die Eingabe von

```
PRINT (45 + 30)/ 5
```

zur Ausgabe von

```
15
Ready
```

Der Befehl PRINT (drucke) gehört zu den BASIC-Kommandos. Er bewirkt, daß etwas auf dem Bildschirm ausgegeben wird. Im vorliegenden Fall ist dies das Ergebnis der Berechnung von (45+30)/5. Wie Sie leicht nachrechnen können, lautet das Ergebnis 15.

Da immer unmittelbar nach Betätigen der ENTER-Taste eine Systemreaktion erfolgt, ist diese Art der Befehlseingabe für die Eingabe einer umfangreicheren Folge von Befehlen nicht sehr gut geeignet.

Eine zusammengehörende Folge von Anweisungen an den Computer, die im allgemeinen die Formulierung eines Algorithmus zur Lösung eines mathematisch oder logisch beschreibbaren Problems darstellen, wird als *Programm* bezeichnet. Das klingt abstrakt, ist es aber gar nicht. Am besten schauen wir uns das anhand eines einfachen Beispiels einmal an.

Angenommen, Sie möchten für den Entwurf eines kleinen Bildschirmspiels ein grafisches Objekt horizontal über den Bildschirm bewegen. Die aktuelle Position ist dabei durch den Wert der X-Koordinate bei konstantem Y-Wert in einer Ebene bestimmt. Die erste mögliche Position sei bei X=0 und die letztmögliche bei X=79. Aus dieser Aufgabenstellung ergibt sich die folgende Vorgehensweise: Damit Sie das Objekt über den Bildschirm bewegen können, müssen Sie es an einer bestimmten Stelle erscheinen lassen (man nennt so etwas setzen), die durch den Startwert von X und die konstante Y-Koordinate festgelegt wird. Anschließend erhöhen Sie den Wert von X um eine fest vorgegebene Schrittweite (beispielsweise 1) und fragen ab, ob das Ende der Bewegungsstrecke erreicht ist. Ist dies nicht der Fall, wird das Objekt an der ursprünglichen Position gelöscht und an die neue gesetzt. Ist dagegen das Ende der Strecke erreicht, wird das Programm beendet. Mit einfachen Worten ausgedrückt lautet der Algorithmus:

1. Anfangswerte von X und Y festlegen
2. Objektkoordinaten ($X_0$ und $Y_0$) auf X und Y setzen
3. Objekt abbilden
4. X um Schrittweite $\triangle x$ erhöhen
5. Abfragen, ob neuer Wert von X den Grenzwert 79 erreicht hat
6. Wenn ja: Programm beenden
7. Wenn nein: Objekt an der alten Position löschen
8. X-Koordinate des Objektes ($X_0$) auf neuen Wert setzen
9. Bei Schritt 3 fortfahren

In Form eines einfachen Flußdiagramms gebracht, sieht das dann so wie in Abbildung 3.1 aus. Natürlich können Sie die Befehle auf Ihrem CPC nicht in der soeben formulierten Form eingeben. Er würde Sie nicht verstehen, weil die Sprachelemente nicht den Regeln der Programmiersprache BASIC entsprechen. Wir wollen uns einige wichtige Regeln nachfolgend kurz anschauen.

### Regeln für die Erstellung von BASIC-Programmen

Im Gegensatz zur Eingabe von unmittelbar ausführbaren Kommandos werden Programmbefehle zunächst im Schreib-/Lesespeicher des Rechners zwischengespeichert. Dies ist immer dann der Fall, wenn die Eingabe einer Anweisung mit einer Zahl beginnt. Sie wird als *Zeilennummer* bezeichnet. Die Eingabe von

```
10 PRINT (45 + 30)/ 5
```

bewirkt somit (auch nach Bestätigung durch die ENTER-Taste) zunächst nichts. Durch Voranstellen der Zahl 10 ist die Anweisung als Bestandteil eines BASIC-Programms ausgewiesen, das als Programmtext in einem bestimmten Bereich des Arbeitsspeichers abgelegt und erst nach ausdrücklicher Aufforderung durch Eingabe des Systemkommandos RUN gestartet und ausgeführt wird.

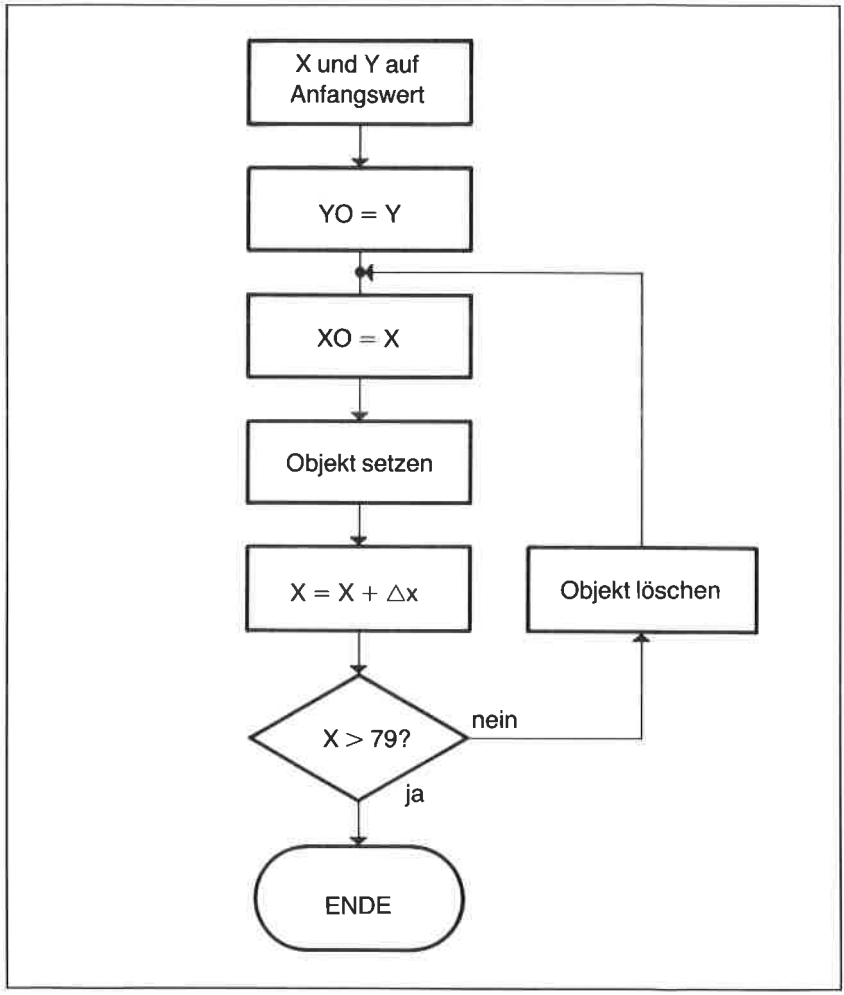



*Abb. 3.1: Beispiel für ein Flußdiagramm*

Als Zeilennummern sind Zahlen zwischen 0 und 65535 erlaubt. Sie werden entweder vom Programmierer selbst vereinbart oder nach Eingabe des Systemkommandos AUTO automatisch mit einer bestimmten Schrittweite erzeugt.

Das Übersetzerprogramm Ihres Schneider CPC gehört zur Kategorie der Interpreter. Es arbeitet nach dem Programmstart alle Programmzeilen in *zahlenmäßig aufsteigender* Reihenfolge ab, sofern nicht programminterne Anweisungen etwas anderes vorschreiben. Die zeitliche Reihenfolge bei der Eingabe ist dabei völlig gleichgültig, weil der Interpreter vor jeder Programmausführung alle Zeilen in aufsteigender Reihenfolge ordnet. Der Grund dafür, daß dieser Vorgang auch bei sehr langen Programmen und nach jeder beliebigen Programmänderung in fast unmeßbar kurzer Zeit abläuft, liegt in der internen Struktur der Organisation des Programmtextes im Speicher. Mehr Informationen zu diesem Themenkreis lesen Sie in einem gesonderten Abschnitt am Ende dieses Kapitels.

Überzeugen Sie sich von dieser bemerkenswerten Fähigkeit des Übersetzerprogramms einfach anhand eines Beispiels. Geben Sie zunächst die folgenden Programmzeilen in der angegebenen Reihenfolge ein:

```
20 PRINT "SCHNEIDER CPC 464"
10 CLS
30 END
```

Bedienen Sie sich dann bitte des Systemkommandos LIST. Es bewirkt, daß der im Speicher abgelegte Programmtext in geschlossener Form auf dem Bildschirm ausgegeben wird. Wie Sie sehen, sieht diese Ausgabe (im Fachjargon Listing genannt) wie folgt aus:

```
LIST

10 CLS
20 PRINT "SCHNEIDER CPC 464"
30 END
Ready
```

Offensichtlich wird entgegen der bei der Eingabe gewählten Reihenfolge der Programmtext immer korrekt mit wachsender Zeilennummer ausgegeben.

Wichtig ist, daß sich alle Anweisungen an die Sprachregeln (Syntax) der Programmiersprache BASIC halten. Verstöße gegen diese Regeln werden vom Interpreter nach dem Programmstart mit einer Fehlermeldung des Typs

```
Syntax error in <Zeilennummer>
```

unter Angabe der Zeilennummer angemahnt.

Falls Sie sich mit dem leicht erlernbaren Sprachschatz von BASIC und dessen Syntax vertraut machen, werden Sie schnell in der Lage sein, nützliche und lauffähige Programme nach Ihren eigenen Vorstellungen zu entwerfen und in Befehlsfolgen umzusetzen. Neben dem BASIC-Befehlssatz, den Sie im übernächsten Abschnitt kennenlernen werden, sollten Sie sich jedoch zuvor noch mit einigen grundsätzlichen Vereinbarungen befassen.

## Allgemeine Vereinbarungen

### *Multistatement-Zeilen*

Eine in sich abgeschlossene BASIC-Anweisung wird als *Statement* bezeichnet. Eine Programmzeile kann mehrere aufeinanderfolgende Statements enthalten. Sie müssen durch Doppelpunkte (:) voneinander getrennt werden.

Beispiel:

```
10 CLS:PRINT "SCHNEIDER CPC 464":END
```

Programmzeilen dieses Typs werden als *Multistatement-Zeilen* bezeichnet. Besonderheiten ergeben sich bei sogenannten Kommentarstatements, deren Beginn durch REM gekennzeichnet wird, und bei absoluten oder bedingten Sprüchen (beispielsweise GOTO, IF...THEN...ELSE) sowie bei dem Aufruf von Unterprogrammen (GOSUB).

### *Zeilenlänge*

Eine BASIC-Programmzeile darf nicht mehr als 255 Zeichen einschließlich Leerräumen und Sonderzeichen enthalten.

### *Variablen und Konstanten*

In der Programmiersprache BASIC können *numerische* und *nichtnumerische* Variablen und Konstanten definiert werden. Letztere werden als *Zeichenketten* oder *Strings* bezeichnet.

*Numerische Variablen und Konstanten*

Numerische Variablen wie auch Konstanten können ganzzahlig *(integer)* oder als Dezimalzahlen *(real)* vereinbart werden. Durch ein Ausrufungszeichen (!) oder nicht gesondert gekennzeichnete Variablen sind Real-Variablen. Ganzzahlige Variablen werden durch Anhängen des Zeichens % gekennzeichnet.

Beispiel: Ganzzahlige numerische Variablen: A1%, VG%, Z%, SUMME%
Real-Variablen : A1!, VG!
Numerische Konstanten: 2.345, 1E-6

*Nichtnumerische Variablen und Konstanten (Zeichenketten)*

Textvariablen unterliegen im Hinblick auf die Vereinbarung der Variablennamen den nachfolgend angegebenen Regeln. Sie werden durch ein nachgestelltes Dollarzeichen gekennzeichnet. Die Elemente von Textkonstanten müssen eindeutig in Anführungszeichen eingeschlossen sein. (Das hintere kann fehlen.)

Beispiel: Textvariablen: A$, NAME$, Z1$
Textkonstanten: "CPC 464", "Programm"

### *Namen von Variablen*

Variablennamen können aus bis zu 40 alphanumerischen Zeichen (Zahlen und Buchstaben) bestehen. Bedingung ist, daß das erste Zeichen ein *Buchstabe* ist. Reservierte BASIC-Wörter dürfen nicht Bestandteil eines Variablennamens sein.

### *Felder und Feldvariablen*

Für ein- oder mehrdimensionale Datenfelder muß zu Beginn eines Programms ein ausreichender Raum im Arbeitsspeicher mittels einer DIM-Anweisung (Dimension) reserviert werden, wenn der Variablenindex 10 überschreitet. Die Regeln für die Typ- und die Namensvereinbarung entsprechen den zuvor angegebenen Grundsätzen.

### *Rechenoperationen und logische Verknüpfungen*

Die Programmiersprache BASIC kennt neben den vier Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) sowie der Exponentiation ( ↑ ) als höhere Rechenoperation noch einige logische Verknüpfungen. Die höchste Priorität besitzt die Exponentiation. Es folgen die Multiplikation sowie die Division auf gleicher Prioritätsebene. Die niedrigste Priorität besitzen die Addition und die Subtraktion. Klammern können die Prioritätsfolge jedoch verändern.

Beispiele:

A = 4*5 − 12/4 weist der Variablen A den Wert 20 − 3, also 17 zu.
A= 4*(5−12/4) führt zu 4*2 , also zu dem Wert 8.
A= (4*5−12)/4 ergibt 8/4, also 2 als Ergebnis.
A= 2 ↑ 5/8 führt zum Ergebnis 32/8, also zu 4.
A= 2 ↑ (5/8) ergibt 1.54221083.

Zulässige logische Operatoren sind:

<kleiner als
<kleiner gleich
=gleich
>größer als
>größer gleich
<ungleich

Weiterhin sind als logische Verknüpfungen AND, OR, XOR und NOT erlaubt. Sie zählen zu den logischen Funktionen des BASIC-Interpreters und gehören daher eigentlich in den hierfür vorgesehenen folgenden Abschnitt dieses Kapitels. Wir wollen sie uns aus didaktischen Gründen vorab ansehen, damit ihre wechselseitige Verknüpfung deutlich wird:

*Die logische UND-Verknüpfung*

Die logische UND-Verknüpfung wird durch das in Abbildung 3.2 oben gezeigte Symbol gekennzeichnet. Sie verknüpft zwei Eingangsgrößen a und b zu einer Ausgangsgröße c nach der ebenfalls in Abb. 3.2 angegebenen Wahrheitstabelle.

Beispiel:

```
PRINT &FE AND &FF
254
```

Das Ergebnis wird verständlich, wenn wir uns dazu die duale Schreibweise der einzelnen Werte ansehen:

```
a: 11111110
b: 11111111
c: 11111110 = 254
```

*Die logische ODER-Verknüpfung*

Die logische ODER-Verknüpfung wird in BASIC durch die Operation OR durchgeführt. Symbol wie auch Wahrheitstabelle entnehmen Sie bitte ebenfalls Abbildung 3.2. Die ODER-Verknüpfung der durch &FE und &FF gegebenen binären Datenworte führt zu

```
PRINT &FE OR &FF

a: 11111110
b: 11111111
c: 11111111 = 255
```

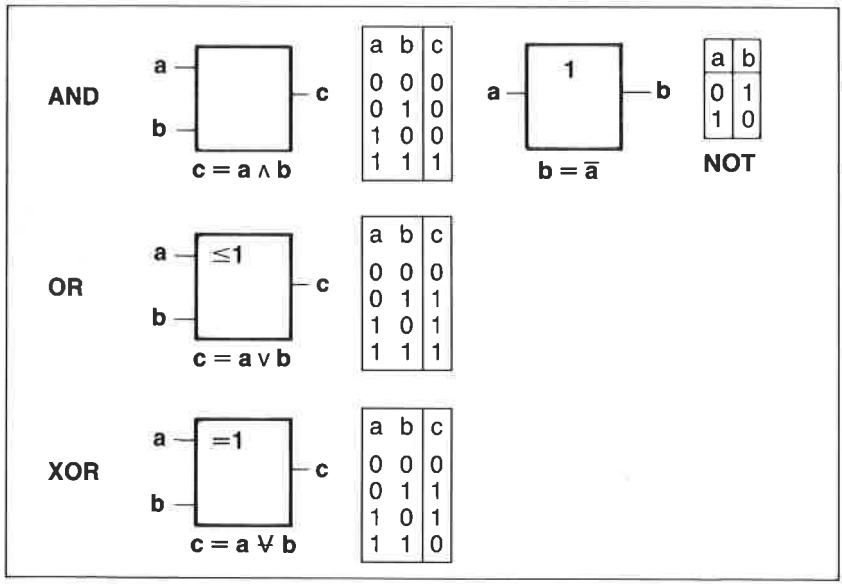



*Abb. 3.2: Logische Verknüpfungen von zwei binären Variablen a und b*

*Die logische Negation*

Das Kommando NOT repräsentiert eine logische Negation (Verneinung) (siehe ebenfalls Abb. 3.2).

```
PRINT NOT (&FE AND &FF)
```

```
a: 11111110
b: 11111111
c: 00000001
```

Nach diesem Schnellgang durch die Regeln und Vorschriften der Programmiersprache BASIC folgt eine kurze Übersicht über den Vorgang der Programmtexterstellung (Editieren) und dessen Korrektur. Im Anschluß daran finden Sie eine Referenzliste der einzelnen BASIC-Befehle mit Erläuterungen.

**Editieren und Korrigieren**

Es ist bereits eine alte Erfahrung, daß BASIC-Programme sehr gern einen recht unhandlichen Umfang annehmen. Das Austesten von Programmteilen, aber auch deren Korrektur im Fehlerfalle ist nicht immer eine einfache Sache. Gute Dienste erfüllt ein leistungsfähiger Texteditor mit möglichst komfortablen Korrekturmöglichkeiten. Mancher Heimcomputerfreak ist da sicherlich von anderen Systemen her recht verwöhnt.

Auch der Schneider CPC braucht sich da nicht zu verstecken, wenn man die systemgegebenen Möglichkeiten nach einiger Übung voll ausschöpfen kann. Der Editor des CPC ist ein Zwischending zwischen einem Bildschirm- und einem Zeileneditor. In der einfachsten Form wird eine Programmkorrektur dadurch eingeleitet, daß im Befehlsmodus von der Anweisung

```
EDIT <Zeilennummer>
```

Gebrauch gemacht wird. Wenn die Zeilennummer vorhanden ist, gibt der Editor den Text der gewünschten Programmzeile auf dem Bildschirm an der aktuellen Position des Textcursors aus und stellt diesen auf die erste Ziffer der Zeilennummer. Da der Textcursor im Transparentmodus arbeitet, ist das darunterliegende Zeichen immer gut sichtbar.

Korrekturen sind an jeder beliebigen Stelle einer Programmzeile möglich, die unter Umständen mehrere Bildschirmzeilen umfassen kann. Den

Ort der Korrektur können Sie durch Betätigen der mit einem Rechts- oder Linkspfeil versehenen Tasten des Cursorblocks ansteuern.

*Hinweis:* Bei gleichzeitiger Betätigung einer der vier Cursortasten und der CTRL-Taste werden folgende Sonderfunktionen abgerufen:

CTRL →Sprung auf das Ende der Bildschirmzeile
CTRL ←Sprung auf den Anfang der Bildschirmzeile
CTRL ↑ Sprung auf den Anfang der Programmzeile
CTRL ↓ Sprung auf das Ende der Programmzeile

### *Einfügen von Zeichen*

Nach der Ausgabe der für die Korrektur freigegebenen Programmzeile befindet sich der Editor im sogenannten *Einfüge*modus. Das bedeutet, daß unabhängig von der Position des Textcursors innerhalb der Zeile über die Tastatur eingetippte Zeichen *links* vom Cursor eingefügt werden. Der Cursor selbst rutscht bei dieser Operation zusammen mit dem Rest des Programmzeilentextes nach rechts. Es können nur so viele Zeichen eingefügt werden, bis der Zeilenpuffer gefüllt ist.

### *Löschen von Zeichen*

Zum Löschen von Zeichen stehen zwei Möglichkeiten zur Verfügung. Betätigen der mit CLR (*CLeaR*) bezeichneten Taste löscht das unter dem Cursor liegende Zeichen. Der Cursor ändert seine Ortslage dabei nicht. Rechts vom Cursor stehender Text rückt bündig nach links nach. Betätigen der mit DEL (*DEL*elete) bezeichneten Taste löscht jeweils das links vom Cursor liegende Zeichen. Dieser rückt dabei mit dem Rest der Zeile nach links.

### *Bestätigen der Korrektur und Übernahme*

Nach Beendigung einer Korrektur wird durch einfaches Betätigen der ENTER-Taste die Korrektur bestätigt und die korrigierte Programmzeile in den Speicher übernommen.

### *Kopieren von Programmtexten*

Eine Übernahme von Textelementen aus anderen auf dem Bildschirm abgebildeten Zeilen ist problemlos möglich. Zu diesem Zweck fahren Sie zunächst auf den Anfang des zu kopierenden Textes. Es steht ein eigener *Kopiercursor* zur Verfügung, der durch gleichzeitiges Betätigen einer der vier Cursortasten *und* der SHIFT-Taste bewegt wird. Kopieren Sie nach dem Aufsuchen des Textanfangs das gewünschte Textelement durch

Betätigen der COPY-Taste. Der von dem Kopiercursor überstrichene Text wird an derjenigen Stelle eingefügt, an der der Originalcursor steht. Beachten Sie bitte, daß der Kopiercursor nur bei *gleichzeitiger* Betätigung der SHIFT-Taste und einer entsprechenden Cursortaste bewegt werden kann. Durch geschicktes Kombinieren der unterschiedlichen Editiermöglichkeiten läßt sich jeder Programmtext mit den vorgestellten Editierwerkzeugen zeitsparend ändern.

### Die BASIC-Befehle

In diesem Abschnitt werden alle BASIC-Befehle in alphabetischer Reihenfolge aufgeführt und, soweit nötig, anhand von Beispielen erläutert. Neben dem jeweiligen Befehl und seiner Syntax werden noch die BASIC-Codes in dezimaler sowie hexadezimaler Schreibweise angegeben. Diese Codes werden im Englischen als *Tokens* bezeichnet. Sie ersetzen bei der Abspeicherung des Programmtextes im Speicher die aufwendigere direkte ASCII-Verschlüsselung. Näheres hierzu lesen Sie im letzten Abschnitt dieses Kapitels, der Sie über die Abspeicherung von BASIC-Programmtexten im Arbeitsspeicher informiert.

Die Befehle berücksichtigen sowohl die Syntax des BASIC 1.0 für den CPC 464 wie auch die neuen Befehle wie die Befehlserweiterungen des BASIC 1.1 für den CPC 664. Diese Erweiterungen sind jeweils in geschweiften Klammern angegeben.

Für Besitzer des 464 sieht die Welt im übrigen gar nicht so trostlos aus, wie dies auf den ersten Blick hin scheinen mag. Die Betriebssystemroutinen enthalten nämlich bereits alle notwendigen Module, um den BASIC-Wortschatz beträchtlich auszubauen. Genaue Informationen über diese Module finden Sie in dem Firmwarehandbuch der Firma Schneider.

Aufbauend auf diesen Routinen lassen sich mittels verhältnismäßig einfach zu erstellender Maschinenprogramme BASIC-Befehlserweiterungen (RSX-Kommandos) erzeugen. Voraussetzung ist natürlich, daß Sie sich ein wenig mit den Regeln der Programmierung in Maschinensprache auskennen. Empfohlen werden kann an dieser Stelle dem Einsteiger wie auch dem bereits erfahrenen Programmierer der CPC 464 Assemblerkurs, der im selben Verlag wie dieses Buch erschienen ist. Es macht Sie nicht nur mit der Programmierung in der Assemblersprache und der Ausnutzung der durch den CPC gegebenen Möglichkeiten vertraut, sondern zeigt auch direkt Wege und fertige Programmodule zur Erweiterung des BASIC-Befehlssatzes durch Kommandos wie CIRCLE, TRI und BOX sowie BOXF, mit deren Hilfe Sie Kreise, Dreiecke und Rechtecke auf den Bildschirm zaubern können.

## ABS

Syntax: ABS(<numerischer Ausdruck>)
BASIC-Code: 255 und 0 bzw. &FF und &00

Erläuterung:

ABS zählt zu den numerischen Funktionen. Mit dieser Funktion wird der Absolutwert der im Argument vereinbarten numerischen Variablen, numerischen Konstante oder eines numerischen Ausdrucks errechnet, d. h. negative Werte werden zu positiven Werten umgewandelt.

Beispiel:

```
PRINT ABS(-23.456)
    23.456

PRINT ABS(SIN(3*PI/2))
    1
```

## AFTER

Syntax: AFTER <Zeitparameter>, [<Nummer des Timers>] GOSUB <Zeilennummer>
BASIC-Code: 128 bzw. &80

Erläuterung:

Der Befehl AFTER führt einen zeitverzögerten Einsprung in ein Unterprogramm durch, dessen Zeilennummer hinter dem GOSUB-Befehl angegeben werden muß. Die Zeitverzögerung T selbst wird mittels des ganzzahligen Parameters n vereinbart. Es gilt

n = T/0.02 Sekunden

Bei der Verzögerung von einer Sekunde beträgt n also 50. Die Verzögerung kann durch jeweils einen von insgesamt vier Hardwaretimern kontrolliert werden. Der Timer wird mittels des Parameters m ausgewählt. Der ganzzahlige Wert von m darf zwischen 0 und 3 liegen. Wenn keine Vereinbarung bezüglich des Timers getroffen wird, wählt der Interpreter den Timer 0 aus. Werden im Programm mehrere Timer angesprochen, gilt die Prioritätsfolge 3, 2, 1, 0. Eine Wirkung besitzt der Befehl nur während des Programmlaufs.

Beispiel:

```
10 AFTER 50 GOSUB 100
20 GOTO 20
100 PRINT CHR$(7)
110 RETURN
```

Querverweis: EVERY, REMAIN

---

**ASC**

Syntax: ASC (<Zeichenkette>)
BASIC-Code: 255 und 1 bzw. &FF und &1

Erläuterung:

Die Funktion ASC ermittelt vom ersten Zeichen der im Argument vereinbarten Zeichenkette den dezimalen Wert des ASCII-Codes. Das Argument ist entweder eine *Textvariable*, eine *Textkonstante* oder ein *Textausdruck*.

Beispiel 1:

```
PRINT ASC("A")
    65
```

Beispiel 2:

```
X$="SCHNEIDER"
PRINT ASC(X$)
    83
```

Querverweis: CHR$

---

**ATN**

Syntax: ATN(<numerischer Ausdruck>)
BASIC-Code: 255 und 2 bzw. &FF und &2

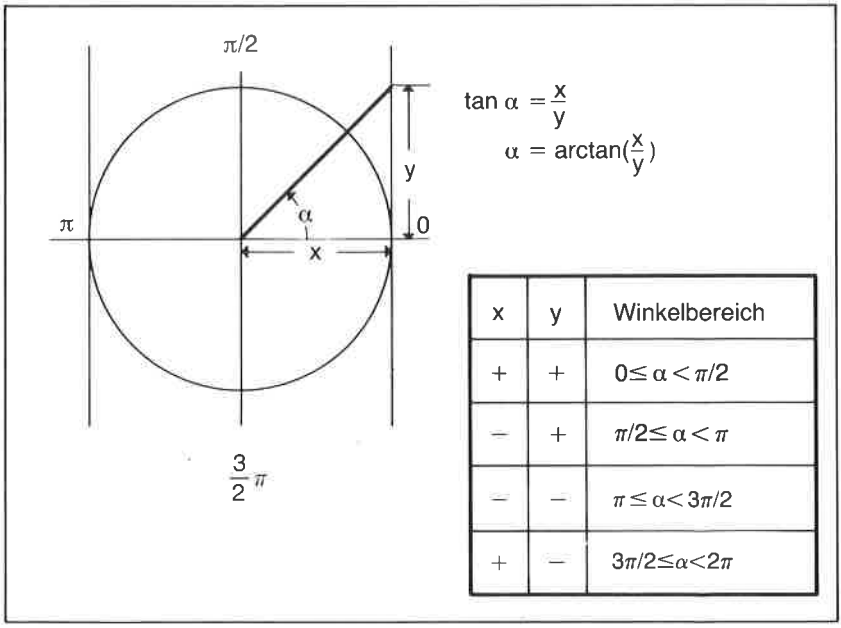
Erläuterung:

Die numerische Funktion ATN errechnet den Arcustangens des Arguments. Dieses ist entweder eine numerische Variable, eine numerische Konstante oder ein numerischer Ausdruck.

Mathematisch ist der Arcustangens entsprechend der Skizze in Abb. 3.3 definiert. Das Ergebnis der Berechnung liegt zwischen $-\pi/2$ ( entspricht $+3\pi/2$) und $+\pi/2$ und wird im Bogenmaß (siehe auch RAD) oder nach der Vereinbarung von DEG im Gradmaß angegeben. Wegen der Mehrdeutigkeit der Funktion muß die Bestimmung des korrekten Winkels im Bereich zwischen 0 und $2\pi$ durch den Programmierer vorgenommen werden.

Beispiel:

```
PRINT ATN(1)
 0.785398163
```




| x | y | Winkelbereich |
|---|---|---|
| + | + | $0 \leq \alpha < \pi/2$ |
| − | + | $\pi/2 \leq \alpha < \pi$ |
| − | − | $\pi \leq \alpha < 3\pi/2$ |
| + | − | $3\pi/2 \leq \alpha < 2\pi$ |

*Abb. 3.3: Zur Definition des Arcustangens*

Diese Zahl entspricht im Rahmen der Stellengenauigkeit dem Wert von π/4. Der Ausdruck 4*ATN(1) ergibt also den Wert von π, der jedoch über die Konstante mit dem Namen PI beim Schneider CPC direkt abgerufen werden kann.

Beispiel:

```
PRINT 4*ATN(1)
 3.14159265

PRINT PI
 3.14159265
```

Algorithmen zur Umrechung in andere trigonometrische Funktionen finden Sie im Anhang N dieses Buches.

Querverweis: TAN

---

## AUTO

Syntax: AUTO [<Zeilennummer>][,<Schrittweite>]
BASIC-Code: 129 bzw. &81

Erläuterung:

Der Systembefehl AUTO erzeugt bei der Erstellung von Programmtexten automatisch Zeilennummern, die bei der im Parameter *Zeilennummer* angegebenen Zahl beginnen und mit *<Schrittweite>* fortgesetzt werden. Wird die Schrittweite nicht angegeben, setzt der Editor sie auf 10. Ist die so erzeugte Zeilennummer bereits vergeben worden, warnt der Editor durch Ausgabe eine Sterns (*) hinter der Zeilennummer, um unbeabsichtigtes Überschreiben zu verhindern. Beim CPC 664 geht der Interpreter im vorgenannten Fall in den Editiermodus über. Das heißt, daß die bereits vorhandene Programmzeile auf dem Bildschirm ausgegeben wird und gegebenenfalls direkt geändert werden kann.

*Hinweis:* Stimmen die durch die Schrittweite gegebenen neuen Folgezeilennummern mit bereits bestehenden überein, kann durch fortlaufendes Betätigen der ENTER-Taste eine Daueredtierfunktion erreicht werden.

Beispiel:

```
AUTO 10,2
```

führt zu einer automatischen Zeilennummer-Erzeugung beginnend bei 10 mit einer Schrittweite von 2, d. h. 10,12,14,16,18, usw.

---

**BIN$**

Syntax: BIN$(<numerischer Ausdruck n> [,<numerischer Ausdruck m>])
BASIC-Code: 255 und 113 bzw. &FF und &71

Erläuterung:

BIN$ zählt zu den Textfunktionen (auch Stringfunktionen genannt). Mit Hilfe von BIN$ wird eine Zeichenkette erzeugt, die dem dualen Wert des positiven ganzzahligen Arguments n entspricht. Der Parameter n muß im Wertebereich zwischen 0 und 65535 liegen. Die Länge der Zeichenkette entspricht grundsätzlich mindestens der für die duale Verschlüsselung benötigten Wortlänge w. Der Parameter m bestimmt die Länge der Zeichenkette nur dann, wenn diese die mindestbenötigte Wortlänge w überschreitet. Es gilt die Beziehung

$$w \leq m \leq 16$$

Ist $m < w$, wird die Parameterangabe für m nicht berücksichtigt.

Beispiel 1:

```
PRINT BIN$(255)
  11111111
```

Beispiel 2:

```
PRINT BIN$(255,3)
  11111111
```

Beispiel 3:

```
PRINT BIN$(16,12)
000000010000
```

Querverweis: HEX$

---

## BORDER

Syntax: BORDER <Farbe $f_1$>[,<Farbe $f_2$>]
BASIC-Code: 130 oder &82

Erläuterung:

Die Bildschirmausgabe von Text- und Grafikinformationen erfolgt beim Schneider CPC innerhalb eines rechteckförmigen Fensters, das von einem bis zu den Sichtgrenzen des Bildschirms reichenden Rahmen *(border)* umgeben ist (siehe hierzu auch Kapitel 2 und die folgende Abb. 3.4).

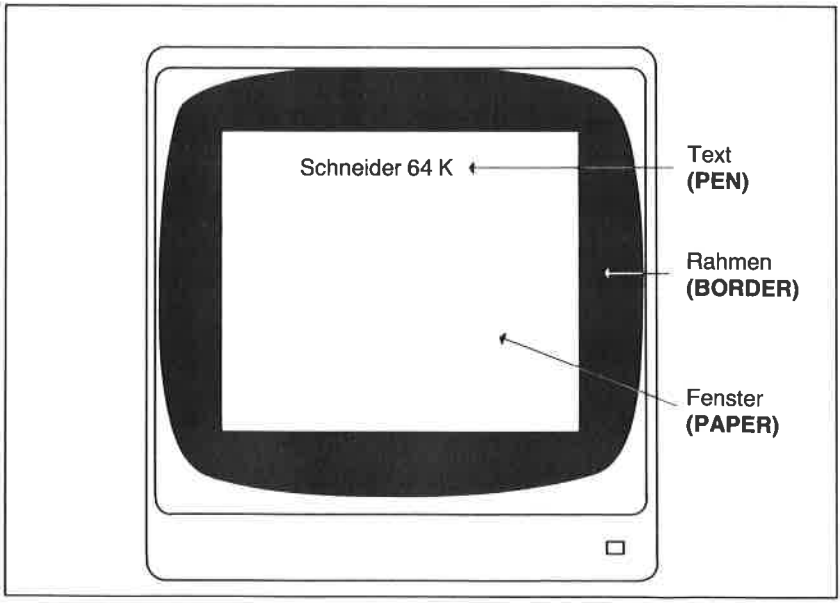



*Abb. 3.4: Erläuterung der Bildschirmausgabe*

Die Farbe dieses Rahmens wird im Kommando BORDER mittels des Parameters $f_1$ vereinbart. Er kann Werte zwischen 0 und 31 entsprechend der in Abb. 3.5 angegebenen Tabelle annehmen. Nur im Bereich zwischen 0 und 26 ergeben sich allerdings voneinander abweichende Farbwerte. Wird der zweite Parameter $f_2$ angegeben, dann wechselt die Farbe rhythmisch zwischen $f_1$ und $f_2$ (siehe auch SPEED INK).

Beispiel 1:

```
BORDER 6
```

setzt den Rahmen auf die Farbe Rot.

Beispiel 2:

```
BORDER 6,9
```

| Farbnr. | Farbe | Farbnr. | Farbe |
|---|---|---|---|
| 0 | Schwarz | 16 | Rosa |
| 1 | Blau | 17 | Pastellmagenta |
| 2 | Hellblau | 18 | Hellgrün |
| 3 | Rot | 19 | Seegrün |
| 4 | Magenta | 20 | helles Blaugrün |
| 5 | Hellviolett | 21 | Limonengrün |
| 6 | Hellrot | 22 | Pastellgrün |
| 7 | Purpur | 23 | Pastellblaugrün |
| 8 | helles Magenta | 24 | Hellgelb |
| 9 | Grün | 25 | Pastellgelb |
| 10 | Blaugrün | 26 | Leuchtendweiß |
| 11 | Himmelblau | 27 | Weiß (wie 13) |
| 12 | Gelb | 28 | Purpur (wie 7) |
| 13 | Weiß | 29 | Pastellgelb (wie 25) |
| 14 | Pastellblau | 30 | Blau (wie 1) |
| 15 | Orange | 31 | Seegrün (wie 19) |

*Abb. 3.5: Die Standardfarbtabelle des Schneider CPC*

führt zu einem Wechsel der Farben zwischen Rot und Grün. Die Geschwindigkeit des Farbwechsels kann über SPEED INK gewählt werden. Zur Wahl der Fenstergröße wird der BASIC-Befehl WINDOW verwendet.

Querverweis: PAPER, INK, WINDOW und SPEED INK

---

## CALL

Syntax: CALL <Startadresse> [,<Variablenliste>]
BASIC-Code: 131 oder &83

Erläuterung:

Mit dem Befehl CALL erfolgt der Einsprung in ein Maschinenprogramm, dessen Startadresse in dem entsprechend bezeichneten Parameter vereinbart werden muß. Wie bei allen Zahlenangaben kann die Adresse sowohl in dezimaler als auch in hexadezimaler Form angegeben werden. Über die Variablenliste können Datenwerte zwischen dem BASIC-Programm und der Maschinenroutine ausgetauscht werden. Für den BASIC-Programmierer interessant sind die Maschinenroutinen, die Bestandteil des im ROM abgelegten Betriebssystems sind. Sie erleichtern vielfach das Arbeiten mit dem System erheblich. Wie eventuelle Variablen zu vereinbaren sind, ist in der Dokumentation derartiger Programme in der Regel erläutert (siehe auch das Firmware Handbuch der Firma Schneider).

Beispiele:

| | |
|---|---|
| CALL &BB9C | vertauscht die Farben von PEN und PAPER |
| CALL &BB18 | unterbricht den Programmlauf bis zum Betätigen einer Taste der Tastatur |
| CALL &BB6C | löscht das aktuelle Fenster |
| CALL &BBDB | löscht das Grafikfenster |
| CALL &BD19 | synchronisiert die Grafikausgabe mit dem Strahlrücklauf (entspricht dem BASIC-Befehl FRAME) |

Im Anhang O dieses Buches finden Sie eine große Auswahl von Einsprungsadressen in die ROM-Routinen des Betriebssystems mit Kurzerläuterungen.

## CAT

Syntax: CAT
Eingabe: Unmittelbar
BASIC-Code: 132 oder &84

Erläuterung:

Der Befehl CAT führt zur Ausgabe des Inhaltsverzeichnisses eines angeschlossenen Massenspeichers. Beim Grundsystem wirkt dieser Befehl auf die im Kassettenrecorder eingelegte Magnetbandkassette. Das Inhaltsverzeichnis von *Kassette* wird in der Form

Name der Datei Blocknummer Format Fehlermeldung

ausgegeben.

Beispiel 1:

```
AMFOOT      block 1     %     OK
AMFOOT      block 2     %     OK
```

Die Attribute für die Aufzeichnungsformate lauten:

$ BASIC-Programm (ungeschützt)
% BASIC-Programm (geschützt)
* ASCII-Datei
& Binäre Datei

Unter AMSDOS wirkt der Befehl auch auf das aktivierte angeschlossene Diskettenlaufwerk. Das Inhaltsverzeichnis einer *Diskette* besitzt die Form

Name der Datei Dateigröße in KByte

Beispiel 2:

```
Drive A: user 0

ADRESSEN.BAS  1K     TEST.BAS  1K
BASTEST  .BAS  1K     TEST.BAK 1K

175K free
```

Querverweis: CHAIN, CHAIN MERGE, LOAD, MERGE, RUN, SAVE

Nähere Informationen über den Betrieb Ihres Schneider CPC mit Diskettenlaufwerken finden Sie in Kapitel 9.

---

**CHAIN**
**CHAIN MERGE**

Syntax: CHAIN "Programmname" [, <Zeilennummer>]
CHAIN MERGE "Programmname" [,<Zeilennummer>]
[,DELETE {<Zeilennummer1> – <Zeilennummer2>}]
BASIC-Code: 133 bzw. &85 (CHAIN)
133 und 171 bzw. &85 und &AB (CHAIN MERGE)

Erläuterung:

Der CHAIN-Befehl ruft aus einem Programm heraus ein unter „Programmname" vereinbartes anderes Programm vom angeschlossenen Massenspeicher auf und liest es in den Speicher ein. Das aufrufende Programm wird überschrieben. Alle Variablen des aufrufenden Programms werden an das neue Programm übergeben.

Es ist möglich, beim Aufruf zusätzlich eine Zeilennummer zu vereinbaren, ab der das neue Programm gestartet werden soll.

In der zweiten Form CHAIN MERGE bleibt beim Aufruf des Programms vom Massenspeicher das ursprüngliche Programm erhalten, sofern es nicht durch Programmzeilen mit derselben Zeilennummer überschrieben wird. Über die Befehlserweiterung DELETE kann festgelegt werden, daß ein Programmteil (beginnend bei <Zeilennummer1> und endend bei <Zeilennummer 2>) vorher gelöscht werden kann.

*Achtung:* Beim CHAIN MERGE-Befehl werden alle im aufrufenden Programm vereinbarten Definitionen für Variablentypen (beispielsweise DEFINT, DEFREAL) und Funktionen (DEFFN) unwirksam. Sie müssen gegebenenfalls neu gesetzt werden!

*Hinweis:* Beim CPC 464 kommt es im Zusammenhang mit dem Betrieb eins externen Diskettenlaufwerkes beim CHAIN MERGE-Befehl zu Problemen wegen eines Fehlers im Betriebssystem. Das folgende kleine Programm (Betriebssystem-Patch) behebt diesen Fehler. Es wird mit freundlicher Genehmigung der Firma ESCON, 8050 Freising, wiedergegeben.

```
100 MEMORY HIMEM-41
110 DEF FNMSB(A)=&FF AND INT(A/256)
120 DEF FNLSB(A)=&FF AND UNT(A)
130 FOR I=HIMEM+1 TO HIMEM+38
140 READ BYTE
150 POKE I,BYTE
160 NEXT I
170 POKE HIMEM  +3,FNLSB(HIMEM+39)
180 POKE HIMEM  +4,FNMSB(HIMEM+39)
190 POKE HIMEM  +9,FNLSB(HIMEM+41)
200 POKE HIMEM +10,FNMSB(HIMEM+41)
210 POKE HIMEM +18,FNLSB(HIMEM+1)
220 POKE HIMEM +19,FNMSB(HIMEM+1)
230 POKE HIMEM +39,PEEK(&BC80+0)
240 POKE HIMEM +40,PEEK(&BC80+1)
250 POKE HIMEM +41,PEEK(&BC80+2)
260 POKE &BC80  +0,&C3
270 POKE &BC80  +1,FNLSB(HIMEM+1)
280 POKE &BC80  +2,FNMSB(HIMEM+1)
290 DATA &E5,&2A,&00,&00,&22,&80,&BC
300 DATA &3A,&00,&00,&32,&82,&BC
310 DATA &CD,&80,&BC,&21,&00,&00
320 DATA &22,&81,&BC,&21,&80,&BC
330 DATA &36,&C3,&E1,&D8,&C8,&FE,&1A
340 DATA &37,&3F,&C0,&B7,&37,&C9
```

Querverweis: GOTO, MERGE, LOAD, RUN und SAVE

---

**CHR$**

Syntax: CHR$(<ASCII-Code>)
BASIC-Code: 255 und 3 oder &FF und &3

Erläuterung:

Mittels CHR$ wird ein Zeichen abgerufen, dessen ASCII-Code im Argument vereinbart werden muß. Erlaubt sind im Argument alle Codes zwischen 0 und 255 bzw &00 und &FF. Die ASCII-Codes zwischen 0 und 31 gehören zu den sogenannten Steuercodes, führen also nicht zur Ausgabe eines alphanumerischen oder grafischen Zeichens. In den Anhängen A und B sind alle ASCII-Codes einschließlich der für den CPC 464 gültigen grafischen Sonderzeichen angegeben.

Beispiel 1:

```
5 REM **** Ausgabe der ASCII-Zeichen ****
10 MODE 1
20 FOR I%=32 TO 255
30 PRINT USING "\ \";CHR$(I%);
40 NEXT I%
```

*Hinweis:* Um die unter Zuhilfenahme der Controltaste zu erzeugenden speziellen Grafikzeichen abzurufen, benutzen Sie bitte die Kombination

```
PRINT CHR$(1);CHR$(N)
```

N muß einen Wert zwischen 0 und 31 annehmen.

Beispiel 2:

```
5 REM **** Ausgabe der Sonderzeichen
10 MODE 1
20 FOR I%=0 TO 31
30 PRINT CHR$(1);CHR$(I%);" ";
40 NEXT I%
```

Beispiel 3:

```
PRINT #8,CHR$(27);CHR$(14);
```

schaltet einen am Druckerport angeschlossenen Matrixdrucker NLQ 401 auf Breitschrift um. Der Befehl entspricht der Ausgabe der Sequenz ESC 14. Durch ESC DC4 (27 und 20) wird auf Normalschrift zurückgesetzt. Dieselbe Codefolge führt auch bei vielen anderen Matrixdruckern (beispielsweise EPSON MX-, RX- oder FX-Serie) zu demselben Ergebnis.

**CINT**

Syntax: CINT(<numerischer Ausdruck>)
BASIC-Code: 255 und 4 bzw. &FF und &4

Erläuterung:

Der Befehl CINT bildet vom numerischen Argument den ganzzahligen gerundeten Wert. Das Argument X ist entweder eine numerische Variable, eine numerische Konstante oder ein numerischer Ausdruck.

Beispiel 1:

```
PRINT CINT(-23.4)
 -23
```

Beispiel 2:

```
PRINT CINT(-23.5)
 -24
```

Querverweis: INT, FIX, CREAL und ROUND

---

**CLEAR**

Syntax: CLEAR
BASIC-Code: 134 bzw. &86

Erläuterung:

Der Befehl CLEAR setzt alle numerischen Variablen auf den Wert Null und löscht sämtliche Textvariablen.

Querverweis: NEW

---

**CLEAR INPUT** (nur CPC 664)

Syntax: CLEAR INPUT
BASIC-Code 134 32 163 bzw. &86 &20 &A3

Erläuterung:

Der Schneider CPC nimmt Tastatureingaben vor der Übernahme in das System in einem Zwischenpuffer auf, der als Tastaturpuffer bezeichnet wird. Mit dem Befehl CLEAR INPUT wird der Inhalt des Tastaturpuffers geleert.

Querverweis: INKEY,INKEY$,JOY

---

## CLG

Syntax: CLG [<Farbwert n>]
BASIC-Code: 135 bzw.&87

Erläuterung:

Der Befehl CLG setzt das Grafikfenster auf die aktuelle Hintergrundfarbe und die Position des Grafikcursors auf die Koordinaten (0,0). Die Hintergrundfarbe kann durch Angabe des Parameters n beim Löschvorgang verändert werden. Für n können Werte zwischen 0 und 15 vereinbart werden. Die zugehörigen Farben hängen von der Betriebsart (MODE) ab. Sie können durch den INK-Befehl geändert werden. Die Standardvereinbarungen für n sind der folgenden Tabelle zu entnehmen.

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Abb. 3.6: Zuordnung der Farbwerte n zum Parameter f bei unterschiedlichen Betriebsarten*

Das aktuelle Grafikfenster kann durch den ORIGIN-Befehl definiert werden.

Beispiel:

```
10 MODE 1
20 ORIGIN 0,0,100,500,200,100
30 CLG 2
```

Das Programm setzt ein Grafikfenster mit den Eckkoordinaten $x_u$=100, $y_u$=200, $x_o$=500 und $y_o$= 100 in der Hintergrundfarbe Hellblau.

Querverweis: ORIGIN

---

**CLOSEIN**

Syntax: CLOSEIN
BASIC-Code: 136 bzw. &88

Erläuterung:

CLOSEIN schließt eine Eingabedatei, die zuvor mit OPENIN geöffnet wurde. Da immer nur eine Datei geöffnet sein kann, muß immer vor dem Einlesen und Öffnen einer neuen Datei über den Befehl CLOSEIN die alte geschlossen werden. Falls nicht sichergestellt ist, daß die zuletzt benutzte Datei geschlossen wurde, sollte der Befehl vor dem Programmlauf unmittelbar eingegeben werden.

Querverweis: CLOSEOUT

---

**CLOSEOUT**

Syntax: CLOSEOUT
BASIC-Code: 137 bzw. &89

Erläuterung: Wirkt auf Ausgabedateien. Siehe auch CLOSEIN.

Querverweis: CLOSEIN

**CLS**

Syntax: CLS [# <Ausgabefenster>]
BASIC-Code: 138 oder &8A

Erläuterung:

Der Befehl CLS (CLear Screen) löscht den Bildschirm, d. h. er setzt die Bildschirmfläche auf die vereinbarte Hintergrundfarbe. Über das WINDOW-Kommando definierte Textfenster (#0 bis #7) können gezielt und unabhängig voneinander durch einen erweiterten CLS-Befehl unter Angabe der Nummer des Zielfensters (#0 bis #7) gelöscht werden.

Beispiel 1:

```
CLS oder CLS #0
```

löschen das Textfenster #0.

Beispiel 2:

```
CLS #6
```

löscht das Textfenster #6.

Querverweis: CLG, NEW

---

**CONT**

Syntax: CONT
BASIC-Code: 139 bzw. &8B

Erläuterung:

Ein durch BREAK (zweimaliges Betätigen der ESC-Taste) oder das BASIC-Kommando STOP abgebrochener Programmlauf kann durch die CONT-Anweisung wieder gestartet werden. Veränderungen des Programmtextes führen zu einer Zerstörung der Rücksprungbedingung.

Querverweis: STOP

**COPYCHR$**(nur CPC 664)

Syntax: COPYCHR$ (#<Ein-/Ausgabekanal>)
BASIC-Code: 225 und 126 bzw. &FF und &7E

Erläuterung:

COPYCHR$ kopiert ein Textzeichen von einer zuvor vereinbarten Position in den Textpuffer.

Beispiel:

```
10 WINDOW #2, 10,10,50,50
20 PRINT "SCHNEIDER"
20 LOCATE 2,1
30 A$=COPYCHR$(#2)
40 PRINT A$

RUN
C
```

Querverweis: LOCATE

---

**COS**

Syntax: COS(<numerischer Ausdruck>)
BASIC-Code: 255 und 5 bzw &FF und &5

Erläuterung:

Durch COS wird die Kosinusfunktion abgerufen. Sie gehört wie die Funktionen Sinus (SIN) und Tangens (TAN) zu den *trigonometrischen Funktionen*.

Das Argument ist entweder eine numerische Variable, eine numerische Konstante oder ein numerischer Ausdruck. Die Kosinusfunktion ist periodisch in $2\pi$ (siehe Abb. 3.7). Das Argument ist ein Winkel, der ohne gesonderte Vereinbarung im Bogenmaß (siehe auch: RAD) anzugeben ist. Über das BASIC-Kommando DEG können auch Argumente im Gradmaß vereinbart werden. Das nachfolgend angegebene kleine Programm zeichnet im MODE 2 eine Sinuskurve im Bereich zwischen 0 und $2\pi$ bzw. zwischen 0 und 360 Grad.

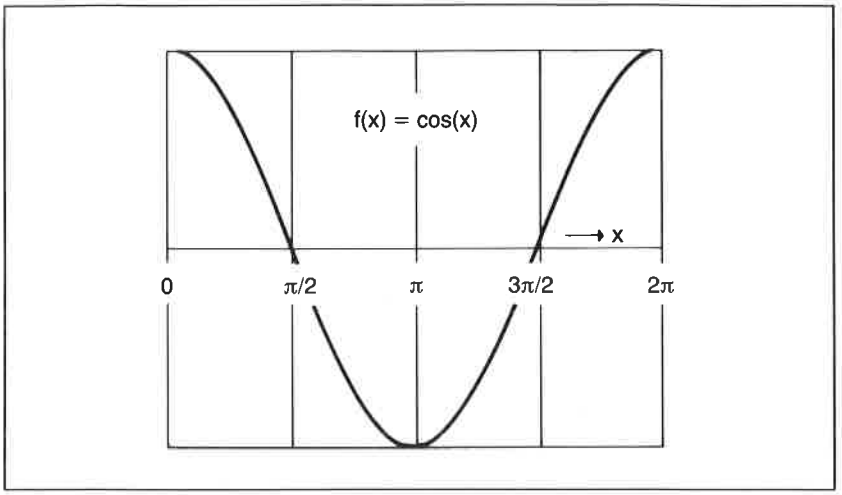



*Abb. 3.7: Kurvenverlauf der Kosinusfunktion für Argumentwerte x zwischen 0 und 2π bzw. 0 und 360 Grad*

Beispiel:

```
5 REM **** Sinuskurve zeichnen ****
10 MODE 2
20 DEG
30 PLOT 150,200:DRAWR 360,0
40 PLOT 150,90:DRAWR 0,220
50 FOR I=0 TO 360
60 PLOT I+150,200+100*COS(I)
70 NEXT I
80 END
```

Querverweis: DEG, RAD, SIN, TAN

---

## CREAL

Syntax: CREAL (<numerischer Ausdruck>)
BASIC-Code: 255 und 6 bzw. &FF und &6

Erläuterung:

CREAL wandelt das numerische ganzzahlige Argument in eine gebrochene Zahl (Realzahl) um.

**CURSOR** (nur CPC 664)

Syntax: CURSOR [<Systemstatus des Cursors>][,<Anwenderstatus des Cursors>]
BASIC-Code: 225 bzw. &E1

Erläuterung:

Der Cursor wird vom Betriebssystem abhängig von der Art der Ausgabeoperation ein- oder auch ausgeschaltet. Bei eingeschalteten Cursor entspricht der Systemstatus einer logischen Eins (1), bei ausgeschalteten Cursor einer logischen Null (0). Mit Hilfe des CURSOR-Befehls kann der Systemstatus vom Anwender geändert werden. CURSOR 1 schaltet beispielsweise die Ausgabe des vom System im Normalfall unterdrückten Cursors ein. Der Anwenderstatus ist logisch Eins, wenn er nicht ausdrücklich geändert wird. Da die beiden Parameter durch eine UND-Verknüpfung voneinander abhängig sind, genügt die Angabe eines einzigen Parameters.

Querverweis: LOCATE

---

**DATA**

Syntax: DATA <Liste von Konstanten>
BASIC-Code: 140 bzw. &8C

Erläuterung:

DATA dient zur Vereinbarung von numerischen Konstanten oder Zeichenketten in Form einer Liste, auf die durch READ-Anweisungen zugegriffen werden kann. Der Typ der Zuweisungsvariablen muß mit dem Typ des jeweiligen Listenelements übereinstimmen. Die einzelnen Elemente der Liste müssen durch Kommata voneinander getrennt werden. Leerzeichen werden nicht berücksichtigt. DATA-Anweisungen können an jeder Stelle des Programms stehen. Auf bereits durch eine READ-Anweisung ausgelesene Listenelemente kann nur dann erneut zugegriffen werden, wenn sie durch eine RESTORE-Anweisung wieder aktiviert werden. Zeichenketten als Bestandteil der DATA-Liste brauchen nicht durch Anführungszeichen kenntlich gemacht zu werden. Sollen jedoch Leerzeichen zu Beginn oder am Ende eines Elements als Bestandteile von Zeichenketten berücksichtigt werden, müssen die entsprechenden Elemente in Anführungszeichen gesetzt werden.

Beispiel:

```
5 REM **** Beispiel fuer den Gebrauch von DATA ****
10 DATA 10,20
20 DATA SCHNELLZUG, LOKOMOTIVE
30 DATA "SCHNELLZUG"," LOKOMOTIVE"
40 READ A,B
50 READ A$,B$
60 READ C$,D$
70 PRINT A,B
80 PRINT A$;B$
90 PRINT C$;D$

RUN
 10            20
SCHNELLZUGLOKOMOTIVE
SCHNELLZUG LOKOMOTIVE
```

## DEC$

Syntax: DEC$(<numerischer Ausdruck>,<Format>)
BASIC-Code: 225 und 114 bzw. &FF und &72

Erläuterung:

Der Befehl DEC$ formatiert den unter dem Parameter <numerischer Ausdruck> vereinbarten Wert nach dem angegebenen Format. Die Formatangaben entsprechen denen des PRINT USING-Befehls.

Folgende Formatangaben sind erlaubt:

| Formatzeichen zeichen | Erläuterung |
|---|---|
| # | Platzhalter für Ziffern. |
| . | Position des Dezimalpunktes. |
| , | Die Lesbarkeit großer Zahlen wird dadurch erhöht, daß nach jeder dritten Ziffer (bezogen auf den Dezimalpunkt) automatisch ein Komma gesetzt wird. |
| – | führt zur Ausgabe eines negativen Vorzeichens am Ende der Zahl. |

| Formatzeichen zeichen | Erläuterung: |
|---|---|
| ** | sorgt dafür, daß führende Leerstellen bei der Zahlenausgabe durch Sternchen aufgefüllt werden. |
| $ | setzt ein Dollarzeichen an die erste durch die Formatangabe definierte Position. |
| $$ | setzt das Dollarzeichen unmittelbar vor die erste auszugebende Ziffer. |
| **$ | Kombination aus ** und $$. |
| + | gibt je nach Formatposition ein positives Vorzeichen zu Beginn oder am Ende der ausgegebenen Zahl aus. |
| – | gibt je nach Formatposition ein negatives Vorzeichen zu Beginn (zusätzlich) oder am Ende der auszugebenden Zahl aus. |
| ↑ ↑ ↑ ↑ | Ausgabe im Exponentialformat. |
| ! | gibt bei Zeichenketten nur das erste Zeichen aus. |
| \<Leerzeichen>\ | es werden so viele Zeichen einer Zeichenkette ausgegeben, wie Leerzeichen vereinbart worden sind. |
| & | Ausgabe der gesamten Zeichenkette. |

Beispiel:

```
10 INPUT B
20 A$ = DEC$(B,"###.##")
30 PRINT A$
40 GOTO 10
```

Nach dem Programmstart werden alle eingegebenen Zahlen mit zwei Stellen hinter und maximal 3 Stellen vor dem Dezimalpunkt ausgegeben.

Querverweis: BIN$, HEX$, PRINT USING, STR$

---

**DEF FN**

Syntax: DEF FN Name [(<unabhängige Variable>)] = <Ausdruck>
BASIC-Code: 141 bzw. &8D

Erläuterung:

DEF FN dient zur Definition von Funktionen, die im Programm dann einfach unter Nennung des vereinbarten Namens sowie der Variablenzuweisung aufgerufen werden können. Die Funktionsdefinition wird möglichst zu Beginn des Programms festgelegt.

Beispiel:

Die Funktion Arcuscosinus ist als Umkehrfunktion der Kosinusfunktion nicht standardmäßig vorhanden. Sie kann über vorhandene Funktionen mittels der mathematischen Beziehung

$$\arccos(x) = \arctan\left(\frac{x}{\sqrt{-x^2+1}}\right) + \pi/2$$

errechnet werden.

Beispiel:

```
5 REM **** Darstellung einer Funktion ****
10 MODE 1
20 DEG
30 DEF FN ACS(X)=ATN(X/SQR(-X*X+1))+PI/2
40 FOR X=0 TO 0.5 STEP 0.1
50 PRINT X,FN ACS(X)
60 NEXT X
70 END

RUN
 0            1.57079633
 0.1          7.30996681
 0.2          13.1077554
 0.3          19.0283994
 0.4          25.1489748
 0.5          31.5707963
```

Eine weitere interessante Funktion ist die Arcussinusfunktion:

ASN = ATN(X/SQR(−X∗X+1))

Querverweis: DEFINT, DEFSTR und DEFREAL

**DEFINT**
**DEFSTR**
**DEFREAL**

Syntax: DEFtyp <Buchstabenbereich>
BASIC-Code: DEFINT: 142 bzw. &8E
DEFSTR: 143 bzw. &8F
DEFREAL: 144 bzw. &90

Erläuterung:

Alle drei Vereinbarungen definieren die Variablen mit den nach dem Befehl angegebenen Anfangsbuchstaben als ganzzahlig (DEFINT), als Zeichenkette (DEFSTR) oder als reell (DEFREAL). Bereiche werden dabei durch einen Bindestrich angezeigt.

Beispiele:

```
DEFINT I-N
DEFSTR A,W-Z
DEFREAL
```

Querverweis: DEF FN

---

**DEG**

Syntax: DEG
BASIC-Code: 145 bzw. 91

Erläuterung:

Mittels der Anweisung DEG wird von Bogenmaß auf Winkelgradmaß umgeschaltet. Trigonometrische Funktionen wie Sinus, Cosinus, Tangens oder Cotangens sind periodische Funktionen in $2\pi$. Anschaulicher ist die Angabe im Gradmaß. Dem Wert $2\pi$ entspricht dann der Winkel 360 Grad, also eine vollständige Umdrehung im Einheitskreis. Die übliche Berechnung

Winkel im Gradmaß = Winkel im Bogenmaß $*180/\pi$

kann beim Schneider CPC wegen der DEG-Anweisung entfallen.

Querverweis: RAD

## DELETE

Syntax: DELETE [<Zeilennummer1>] – [<Zeilennummer2>]
BASIC-Code: 146 bzw. &93

Erläuterung:

DELETE löscht je nach Zusatzvereinbarung alle, nur einzelne oder einen Bereich von Zeilen innerhalb der angegebenen Grenzen zwischen <Zeilennummer1> und <Zeilennummer2>. Im Gegensatz zu NEW werden die bereits vereinbarten Variablen nicht gelöscht.

Beispiele:

DELETE löscht alle Zeilen,
DELETE 1200 löscht die Zeile mit der Nummer 1200
DELETE 1200 – 2400 löscht alle Zeilen im Bereich zwischen 1200 und 2400.

Querverweis: NEW

---

## DERR

SYNTAX: DERR
BASIC-Code: 255 und 73 bzw. &FF und &49

Erläuterung:

DERR ist ein Mnemonic für *D*isk *ERR*or. Die beim Diskettenbetrieb auftretenden Fehler werden in Form von Fehlermeldungen ausgegeben, die sogenannten Fehlercodes zugeordnet sind. Diese können in speziellen Fehlerbehandlungsroutinen bearbeitet werden. Die Fehlercodes werden entweder vom Betriebssystem AMSDOS oder vom Diskettencontroller erzeugt. Für die vom AMSDOS generierten Fehlercodes gelten die folgenden DERR-Werte:

| | |
|---|---|
| 0 oder 22 | ESC-Taste wurde betätigt |
| 142 | Kanalstatus unbekannt |
| 143 | Dateiende erreicht |
| 144 | Fehlerhaftes DOS-Kommando |
| 145 | Datei existiert bereits |
| 146 | Datei existiert noch nicht |

| | |
|---|---|
| 147 | Kein Platz mehr für einen Eintrag im Inhaltsverzeichnis frei |
| 148 | Diskette voll |
| 149 | Diskette entnommen bei noch geöffneten Dateien |
| 150 | Datei nur les- aber nicht überschreibbar |
| 154 | Dateiende |

Die vom Controller erzeugten Fehlercodes werden bitsignifikant ausgegeben. Die Bits des entsprechenden Statuswortes haben folgende Bedeutung:

| | |
|---|---|
| Bit 0: | Adressenbezeichnung fehlt |
| Bit 1: | Diskette nicht beschreibbar (Schreibschutz aktiv) |
| Bit 2: | Unbekannter Sektor |
| Bit 3: | Laufwerk nicht bereit/keine Diskette im Laufwerkschacht |
| Bit 4: | Datenüberlauf |
| Bit 5: | Datenfehler/CRC-Fehler (cyclic redundancy check) |
| Bit 6: | Eins, wenn Fehlercode vom Controller erzeugt wird |
| Bit 7: | Eins, wenn Fehlercode von AMSDOS erzeugt und ausgegeben wird. |

Querverweis: ERL, ERR, ERROR, ON ERROR..., RESUME

---

**DI**

Syntax: DI
BASIC-Code: 219 bzw. &DB

Erläuterung:

DI bedeutet „*D*isable *I*nterrupt". Der Befehl verhindert, daß ein Programmlauf durch externe Unterbrechungen beeinflußt wird. (Eine Ausnahme bildet der Programmabbruch durch BREAK.) Aufgehoben wird dieser Betriebszustand durch den entgegengesetzten Befehl EI (*E*nable *I*nterrupt). Interessant ist der Befehl im Zusammenhang mit programmierten Einsprüngen in Unterprogramme mittels EVERY, AFTER oder REMAIN. Um deren Wirkung vorübergehend auszuschalten, wird DI und später EI vereinbart. Die ausgesetzten Einsprünge werden unmittelbar nach EI nachgeholt.

Beispiel:

```
5 REM **** Demonstration von DI und EI ****
10 CLS
20 EVERY 50 GOSUB 100
30 FOR I=1 TO 5000:NEXT I
40 DI
50 PRINT "Jetzt wird der Einsprung unterdrueckt,"
60 FOR I=1 TO 5000:NEXT I
70 PRINT "Und jetzt wieder freigegeben."
80 EI
90 END
100 PRINT CHR$(7);
110 RETURN
```

Querverweis: EI, AFTER, EVERY, REMAIN

---

**DIM**

Syntax: DIM <Liste von indizierten Variablen>
BASIC-Code: 147 bzw. &93

Erläuterung:

Der Befehl DIM dient zur Reservierung von Speicherbereichen für Datenfelder. Für diese ist der aus der Mathematik stammende Begriff Matrix bzw. Matrizen geläufig. Jedes Datenelement eines solchen Feldes besitzt eine konstante Nummer, die als Index bezeichnet wird. Die Nummer legt die Lage innerhalb des Feldes genau fest. Die Argumente der in der Variablenliste angegebenen Elemente (numerische Variablen oder Textvariablen) enthalten die maximale Anzahl der Feldelemente. Werden keine Felder ausdrücklich vereinbart, gestattet BASIC die Zuordnung von maximal 10 Feldelementen ohne besondere Angaben.

Feldvariablen können unterschiedliche *Dimensionen* besitzen. Abb. 3.8 zeigt, wie Felder unterschiedlicher Dimension vereinbart werden und wie man sie sich bis zur dritten Dimension anschaulich vorstellen kann.

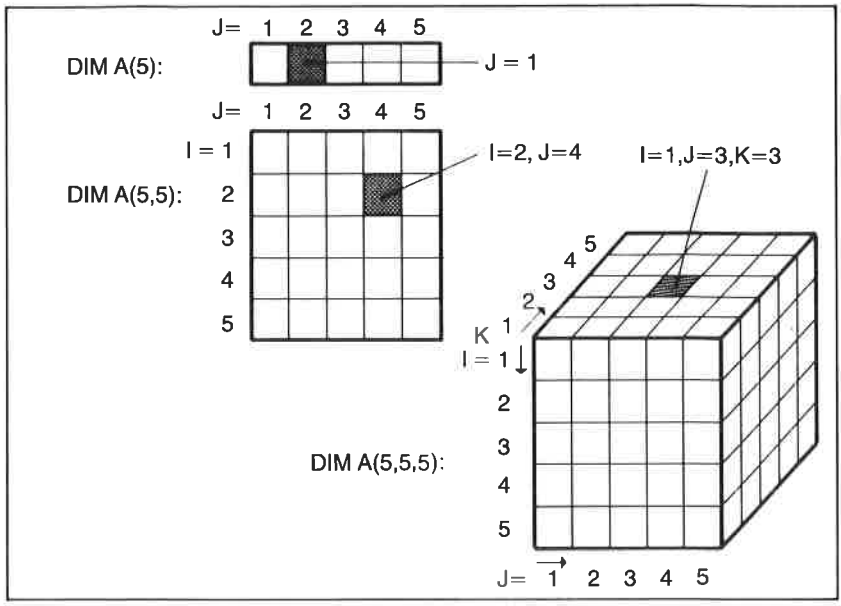



*Abb. 3.8: Geometrische Deutung ein- und mehrdimensionaler Felder*

Die Laufindizes sind in den gezeigten Beispielen mit I, J und K angenommen. Beachten Sie, daß der Umfang von Feldern mit wachsender Dimension rasch zunimmt.

Querverweis: ERASE

---

## DRAW

Syntax: DRAW <x-Koordinate>, <y-Koordinate> [,<Farbe>] {,<Abbildungsmodus>}
BASIC-Code: 148 bzw. &94

Erläuterung:

Der Befehl DRAW zählt zu den Grafikbefehlen des Schneider CPC. Mit seiner Hilfe wird von der aktuellen Position des Grafikcursors zu dem durch das Koordinatenpaar (x,y) vereinbarten Punkt auf dem Bildschirm eine Verbindungslinie gezogen.

Sofern keine besonderen Vereinbarungen getroffen werden, liegt die Koordinate (x,y) =(0,0) in der linken unteren Ecke des Bildschirmfen-

sters. Der rechte obere Eckpunkt des Fensters besitzt unabhängig vom Grafikmodus die Koordinaten x=639, y=399. Beachten Sie jedoch, daß die Pixelauflösung in den drei Betriebsmodi unterschiedlich groß ist. Die Anzahl darstellbarer Farben hängt ebenfalls von der gewählten Betriebsart ab.

Beim CPC 664 kann als zusätzlicher Parameter ein Abbildungsmodus vereinbart werden. Er ist in der Syntaxbeschreibung in geschweifte Klammern gesetzt. Der Modus legt fest, welche logische Verknüpfung die im Farbparameter vereinbarte Vordergrundfarbe mit der aktuellen Farbe an der Ausgabeposition einnimmt.

Es gilt:

0: normal
1: XOR (Exklusiv-ODER-Verknüpfung)
2: UND-Verknüpfung
3: ODER-Verknüpfung

Beispiel:

```
5 REM **** Das ist das Haus ****
6 REM **** vom Nikolaus     ****
10 MODE 1
20 PLOT 639,0
30 DRAW 639,300
40 DRAW 320,399
50 DRAW 0,300
60 DRAW 639,300
70 DRAW 0,0
80 DRAW 639,0
90 DRAW 0,300
100 DRAW 0,0
```

*Hinweis:* Die im Argument angegebenen Koordinaten können in allen drei Darstellungsmodi unverändert gelassen werden. Es ändert sich lediglich die Pixelauflösung.

Querverweis: DRAWR, PLOT, PLOTR

## DRAWR

Syntax: DRAWR <x-Offset>,<y-Offset>,[<Farbwert n>] {,<Abbildungsmodus>}
BASIC-Code: 149 bzw &95

Erläuterung:

Mit dem DRAWR-Befehl wird eine Gerade zwischen der aktuellen Position des Grafikcursors zu einer Zielkoordinate gezeichnet, die nicht absolut, sondern in Form eines Koordinatenoffsets angegeben wird. Zusätzlich kann eine für die entsprechende Betriebsart zulässige Farbe angegeben werden.

Beim CPC 664 kann als zusätzlicher Parameter ein Abbildungsmodus vereinbart werden. Er ist in der Syntaxbeschreibung in geschweifte Klammern gesetzt. Der Modus legt fest, welche logische Verknüpfung die im Farbparameter vereinbarte Vordergrundfarbe mit der aktuellen Farbe an der Ausgabeposition einnimmt.

Es gilt:
0: normal
1: XOR (Exklusiv-ODER-Verknüpfung)
2: UND-Verknüpfung
3: ODER-Verknüpfung

Die Wirkung des Befehls wird in der Abb. 3.9 anschaulich erläutert.

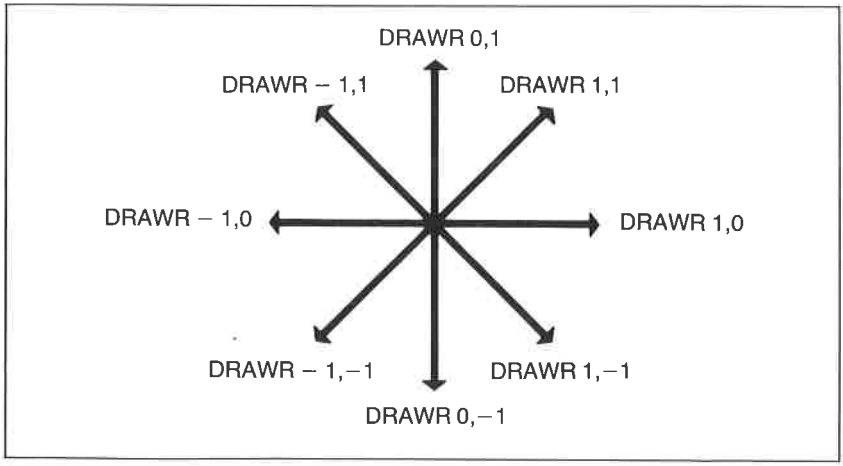



*Abb. 3.9: Die Wirkung des Koordinatenoffsets im DRAWR-Befehl*

Der Grafikcursor kann unsichtbar mit dem Befehl MOVE oder MOVER an eine definierte Ausgangsposition gesetzt werden. Dies ist ebenfalls mittels der Befehle DRAW,PLOT und PLOTR möglich, wenn die Vordergrundfarbe auf die vereinbarte Hintergrundfarbe gesetzt wird.

Programmbeispiel:

```
5 REM **** Box mit DRAWR zeichnen ****
10 MODE 2
20 PLOT 250,150
30 DRAWR 150,0
40 DRAWR 0,150
50 DRAWR -150,0
60 DRAWR 0,-150
70 END
```

Querverweis: DRAW, PLOT, PLOTR, MOVE, MOVER, TEST, TESTR, XPOS, YPOS, ORIGIN

---

## EDIT

Syntax: EDIT <Zeilennummer>
BASIC-Code: 150 bzw. &96

Erläuterung:

Der EDIT-Befehl dient zum gezielten Aufruf zu korrigierender Zeilen des BASIC-Programmtextes. Die Eingabe erfolgt im Befehlsmodus des Interpreters direkt von der Tastatur aus.

Beispiel:

Aus dem Programmtext

```
10 REM **** Testprogramm ****
20 PRINT "TESTPROGRAMM"
30 END
```

kann mittels EDIT 20 die Programmzeile 20 auf den Bildschirm abgerufen und für eine Korrektur vorbereitet werden.

```
EDIT 20
20 PRINT "TESTPROGRAMM"
```

Der Cursor steht dem nach Aufruf auf dem ersten Zeichen, im vorliegenden Fall also auf der 2 der Zeilennummer 20. Da der Cursor grundsätzlich im Transparentmodus abgebildet wird, ist das darunterliegende Zeichen immer sichtbar. Mittels der mit einem Rechts- bzw. einem Linkspfeil gekennzeichneten Cursortasten kann jede Zeichenposition zwischen dem Zeilenanfang und dem physikalischen Zeilenende aufgesucht werden. Änderungen des Programmtextes werden einfach durch Drücken der ENTER-Taste bestätigt und bleibend abgespeichert.

*Hinweis:* Bei gleichzeitiger Betätigung einer der vier Cursortasten mit der CTRL-Taste werden folgende Sonderfunktionen abgerufen:

| | |
|---|---|
| CTRL → | Sprung auf das Ende der Bildschirmzeile |
| CTRL ← | Sprung auf den Anfang der Bildschirmzeile |
| CTRL ↑ | Sprung auf den Anfang der Programmzeile |
| CTRL ↓ | Sprung auf das Ende der Programmzeile |

Folgende Korrekturmodi sind zulässig:

*Einfügen von neuem Text:*

Der Interpreter befindet sich bei dem Korrigiervorgang immer im sogenannten Einfügemodus (auch Insertmodus genannt). Über die Tastatur eingegebene Zeichen werden unmittelbar *vor* der aktuellen Cursorposition in den Programmtext eingefügt.

*Löschen von Text:*

Betätigen der DELETE-Taste löscht *vor* dem Cursor stehende Zeichen, Betätigen der CLR-Taste dagegen immer das jeweils *unter* dem Cursor liegende Zeichen. Rechts vom Cursor liegender Text rückt in beiden Fällen automatisch nach links nach. Sowohl die DELETE- als auch die CLR-Taste besitzen Wiederholfunktion.

*Kopieren von Text:*

Einzelne Zeichen wie auch ganze Textpassagen können von jeder beliebigen Position des Bildschirms in eine für die Korrektur aufgerufene Pro-

grammzeile kopiert werden. Hierzu wird zunächst die Einfügeposition innerhalb der Programmzeile aufgesucht und anschließend mittels der vier mit Pfeilen gekennzeichneten Cursortasten *bei gleichzeitigem Betätigen der SHIFT-Taste* der Kopiercursor auf den Anfang des zu kopierenden Textes gesetzt. Die zu kopierenden Zeichen werden dann durch fortlaufend wiederholtes oder dauerhaftes Drücken der mit COPY bezeichneten Taste unmittelbar in die zu korrigierende Programmzeile übernommen. Auch hier wird der Korrekturvorgang durch die ENTER-Taste bestätigt und beendet.

---

**EI**

Syntax: EI
BASIC-Code: 220 bzw. &DC

Erläuterung:

EI hebt die Sperre externer Unterbrechungen durch den DI-Befehl wieder auf.

Querverweis: DI

---

**END**

Syntax: END

Erläuterung:

Eine END-Anweisung kennzeichnet das logische Ende eines Programms. Wird END nicht angegeben, ergänzt der Interpreter intern dieses Kommando als letzte Programmanweisung.

*Hinweis:* Achten Sie darauf, daß das Hauptprogramm von eventuell angehängten Unterprogrammen entweder durch eine Endlosschleife (Beispiel: 999 GOTO 999) oder durch ein END-Statement abgeschlossen wird. Der Interpreter läuft sonst bei Programmende linear in das als erstes ans Programm angehängte Unterprogramm und bricht den Lauf mit der Fehlermeldung „Unexpected RETURN in ...“ ab.

Eine END-Anweisung schließt alle Kassetten- und Diskettendateien und kehrt in den Befehlsmodus des Interpreters mit der Ready-Meldung

zurück. Achtung: Stationäre SOUND-Anweisungen werden durch END nicht unterbrochen!

Querverweis: STOP

---

**ENT**

Syntax:

ENT <Hüllkurvennummer> [,<N1>,<dN1>,<dT1>] [,....][,....][,....] [,<N5>,<dN5>,<dT5>]

oder

ENT <Hüllkurvennummer> = <Signalperiode>, <Pausenzeit>
BASIC-Code: 153 bzw. &99

Erläuterung:

Die Tonerzeugung mittels des Sound-Generators AY-3-8912 wird von der BASIC-Ebene aus über die Befehle SOUND, ENV und ENT gesteuert. Während mit der ENV-Anweisung (*EN*velope of *V*olume) die Art der *Amplituden*modulation definiert werden kann, wird über die ENT-Anweisung (*EN*velope of *T*one) eine *Frequenz*modulation, d. h. eine programmierte Veränderung der Tonhöhe definiert. Für den Befehl existieren zwei unterschiedliche Vereinbarungen.

In der ersten Form sind die Befehlsparameter wie folgt definiert:

*Hüllkurvennummer:* Unter Hüllkurvennummer wird ein Wert zwischen 1 und 15 angegeben, der bei der Angabe des SOUND-Komandos zum Aufruf der unter ENT vereinbarten Hüllkurvenform dient. Wird die Hüllkurvennummer negativ vereinbart, erfolgt eine periodische Wiederholung für die im SOUND-Befehl vereinbarte Gesamtdauer.

Als Hüllkurvenparameter können die *Schrittzahl N*, die *Schrittweite dN* und die *Schrittdauer dT* angegeben werden. Für die Parameter gelten folgende Vereinbarungen:

*Schrittzahl N:* Die Anzahl von Schritten muß jeweils zwischen 0 und 127 liegen.

*Schrittweite dN:* Die Schrittweite dN darf Werte zwischen −128 und +127 annehmen. Eine positive Schrittweite führt zu einer schrittweisen Ernied-

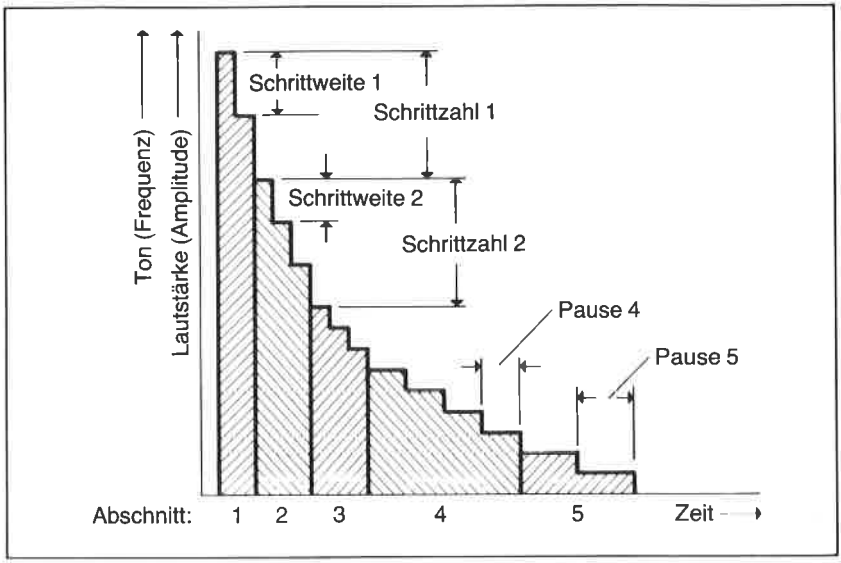



*Abb. 3.10: Konstruktion eines nichtlinearen Verlaufs der Hüllkurve für Lautstärke- und Tonhöhenvariationen*

rigung des durch den SOUND-Befehl vereinbarten Anfangswertes der Frequenz (schrittweise Erniedrigung der Tonhöhe um dN). Negative Werte führen zu einer Erhöhung der Tonhöhe in Schritten von dN.

*Schrittdauer dT:* Die Schrittdauer dT bestimmt die Zeitdauer eines Intervallschrittes. dT kann zwischen 0 und 255 liegen. Das absolute Zeitintervall für einen Intervallschritt der Hüllkurve beträgt T = dT∗0.01 Sekunden. Die Vereinbarung des Wertes 0 entspricht 256.

Bis zu fünf unterschiedliche Parametergruppen (Abschnitte) für die soeben angegebenen Größen können mit einem ENT-Befehl vereinbart werden. Hierdurch ist es möglich, einen nichtlinearen Hüllkurvenverlauf zu erzielen, wie dies in Abb. 3.10 schematisch gezeigt ist.

In der zweiten Form gilt:

*Signalperiode:* Es sind Tonhöhenwerte zwischen 0 und 4096 möglich. Beachten Sie bitte, daß die Tonhöhe umgekehrt proportional zur Periodendauer ist:

$$\text{Tonhöhe} = \frac{1}{\text{Periodendauer}}$$

Große Zahlenangaben führen also zu tiefen, kleinen Zahlenangaben zu hohen Tönen. Im Anhang P finden Sie eine Tabelle, in der den Standardwerten der Periodendauer die entsprechenden Frequenzwerte sowie die Notenbezeichnungen zugeordnet sind.

Querverweis: ENV, SOUND, Kapitel 8

---

**ENV**

Syntax:
ENV<Hüllkurvennummer>[,<N1>,<dN1>,<dT1>][,....][,....][,....][,<N5>,<dN5>,<dT5>]

oder

ENV <Hüllkurvennummer> = <Hüllkurventyp>, <Wiederholperiode>
BASIC-Code: 164 bzw. &9A

Erläuterung:

Mittels des Befehls ENV wird der Hüllkurvenverlauf der Signal*amplitude* vereinbart. Diese bestimmt die zeitliche Änderung der über SOUND festgelegten Lautstärke eines Tons oder Geräusches. Die folgende Abb. 3.11 zeigt die allgemeinen Vereinbarungen für den Verlauf der Einhüllenden, deren wesentliche Eigenschaften durch den Toneinsatz (engl. attack) sowie den Ausklang (engl. decay) bestimmt werden (siehe auch Kapitel 5).

Wie beim ENT-Befehl existieren zwei verschiedene Befehlsformen. In der ersten angegebenen Form sind die einzelnen Parameter wie folgt definiert:

*Hüllkurvennummer:* Unter Hüllkurvennummer wird ein Wert zwischen 1 und 15 angegeben, der bei der Angabe des SOUND-Komandos zum Aufruf der unter ENT vereinbarten Hüllkurvenform dient.

Als Hüllkurvenparameter können die *Schrittzahl N*, die *Schrittweite dN* und die *Schrittdauer dT* angegeben werden. Für die Parameter gelten folgende Vereinbarungen:

*Schrittzahl N:* Die Anzahl von Schritten muß jeweils zwischen 0 und 127 liegen.

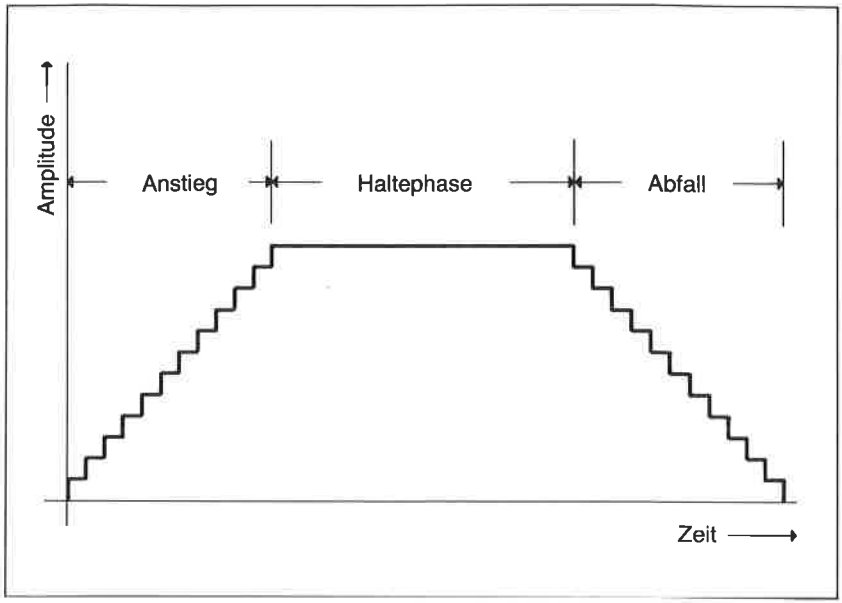



*Abb. 3.11: Erläuterung der Einhüllenden eines Tones*

*Schrittweite dN:* Die Schrittweite dN darf Werte zwischen −128 und +127 annehmen. Eine positive Schrittweite führt zu einer schrittweisen Erniedrigung des durch den SOUND-Befehl vereinbarten Anfangswertes der Frequenz (schrittweise Erniedrigung der Tonhöhe um dN). Negative Werte führen zu einer Erhöhung der Tonhöhe in Schritten von dN.

*Schrittdauer dT:* Die Schrittdauer dT bestimmt die Zeitdauer eines Intervallschrittes. dT kann zwischen 1 und 256 liegen. Das absolute Zeitintervall für einen Intervallschritt der Hüllkurve beträgt T = dT∗0.01 Sekunden. Die Vereinbarung des Wertes 0 entspricht 256.

Bis zu fünf unterschiedliche Parametergruppen (Abschnitte) für die soeben angegebenen Größen können mit einem ENT-Befehl vereinbart werden. Hierdurch ist es möglich, einen nichtlinearen Hüllkurvenverlauf zu erzielen, wie dies in Abb. 3.11 schematisch gezeigt ist.

In der zweiten zu Beginn angegebenen Form gilt:

*Hüllkurventyp:* Der Sound-Chip des Typs AY-3-8912 erzeugt bei entsprechendem Abruf Hüllkurvenverläufe, die bereits hardwaremäßig auf dem

Chip programmiert sind. Diese Hüllkurvenverläufe können dadurch aufgerufen werden, daß in das Register &15 des Sound-Chips die entsprechenden Codeworte geladen werden. Die Abb. 3.12 zeigt schematisch die so abrufbaren Typen von Einhüllenden und die zugehörigen Codes.

*Wiederholperiode:* Der über den Parameter Hüllkurventyp abgerufende Verlauf der Einhüllenden kann hinsichtlich seiner zeitlichen Wiederholung durch diesen Parameter festgelegt werden. Der vereinbarte Wert wird in die Register &13 und &14 des Sound-Chips eingeschrieben.

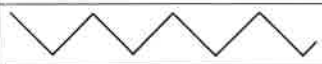
| | | Fortführung | Anstieg | Wechsel | Halten | Hüllkurvenverlauf |
|---|---|---|---|---|---|---|
| dezimal | hex. | B3 | B2 | B1 | B0 | |
| 0 bis 3 | 0 bis 3 | 0 | 0 | x | x |  |
| 4 bis 7 | 4 bis 7 | 0 | 1 | x | x |  |
| 8 | 8 | 1 | 0 | 0 | 0 |  |
| 9 | 9 | 1 | 0 | 0 | 0 |  |
| 10 | A | 1 | 0 | 1 | 0 |  |
| 11 | B | 1 | 0 | 1 | 1 |  |
| 12 | C | 1 | 1 | 0 | 0 |  |
| 13 | D | 1 | 1 | 0 | 1 |  |
| 14 | E | 1 | 1 | 1 | 0 |  |
| 15 | F | 1 | 1 | 1 | 1 |  |

*Abb. 3.12: Die Codierung des Hüllkurvenregisters über den ENV-Befehl*

Beispiele für die einzelnen Befehlsformen finden Sie in Kapitel 5.

**EOF**

Syntax: EOF
BASIC-Code: 255 und 64 bzw. &FF und &40

Erläuterung:

Mit der Anweisung EOF (End Of File) kann das Ende einer von einem Datenträger (Kassette oder Diskette) einzulesenden Datei abgefragt werden. Als Ergebnis der Abfrage wird bei Erkennung des Dateiendes die Zahl −1 und andernfalls die Zahl 0 an das Programm übergeben.

Querverweis: OPENIN

---

**ERASE**

Syntax: ERASE <Liste von Feldvariablen>
BASIC-Code: 155 bzw. &9B

Erläuterung:

Der Befehl ERASE dient dazu, nicht mehr benötigte Feldvariablen zu löschen. Im Speicher wird dadurch Platz für eine anderweitige Verwendung des für diese Variablen ursprünglich reservierten Arbeitsspeichers geschaffen.

Querverweis: DIM

---

**ERR**
**ERL**

Syntax: ERR oder ERL
BASIC-Codes: ERR: 255 und 65 bzw. &FF und &41
ERL: 156 bzw. &9C

Erläuterung:

Vom Interpreter nach dem Programmstart entdeckten Fehlern sind Fehlernummern zugeordnet. Diese können im Fehlerfall durch ERR abgefragt werden, während ERL diejenige Zeilennummer an das Programm übergibt, in der der Fehler entdeckt wurde.

Achtung: Denken Sie bitte daran, daß die Fehlerursache nicht unbedingt in jener Zeile zu suchen ist, die von ERL angezeigt wird!

Üblicherweise sind ERR und ERL Bestandteile von Fehlerbehandlungsroutinen, die als Unterprogramme über einen zu Programmbeginn vereinbarten ON ERROR GOSUB-Befehl angesprungen werden.

Beispiel:

```
10 CLS
20 ON ERROR GOTO 100
30 GOTO 40
50 END
100 IF ERR=8 THEN PRINT "Es gibt keine Zeile 40 !!"
110 RESUME NEXT
```

Querverweis: ON ERROR, ERROR

---

**ERROR**

Syntax: ERROR <numerischer Ausdruck>
BASIC-Code: 156 bzw. &9C

Erläuterung:

Der Interpreter erlaubt neben den im Handbuch unter den Nummern 1 bis 30 verzeichneten Fehlermeldungen die Definition zusätzlicher Fehlernummern, deren Behandlung vom Programmierer in eigens geschriebenen Unterprogrammen abgehandelt wird. Bereits vorhandene Fehlernummern führen zu den für den Interpreter vereinbarten Systemreaktionen.

Beispiel:

```
10 CLS:PRINT:PRINT
20 CD$=CHR$(67)+CHR$(80)+CHR$(67)+CHR$(32)+CHR$(52)+CHR
$(54)+CHR$(52)
30 INPUT "Codewort eingeben: ",CO$
40 IF CO$<>CD$ THEN ERROR 28 ELSE PRINT:PRINT "OK."
50 END
```

Querverweis: ON ERROR, ERR, ERL

## EVERY

Syntax: EVERY <Zeitangabe n>[,<Nummer des Taktgebers m>] GOSUB <Zeilennummer>
BASIC-Code: 157 bzw &9D

Erläuterung:

Der CPC 464/664 enthält Hardwaretaktgeber, mit deren Hilfe bis zu vier unterschiedliche, gleichzeitige Zeitnahmen möglich sind. Über die EVERY-Anweisung können periodische Systemreaktionen erzeugt werden, deren Wiederholperiodenzeit T über den Parameter n vereinbart wird. Im zweiten Parameter m wird durch Angabe einer ganzen Zahl zwischen 0 und 3 definiert, welcher der vier vorhandenen Taktgeber die Zeitkontrolle übernehmen soll. Jeder der Taktgeber erzeugt Programmunterbrechungen, die zu einem Einsprung in ein vom Programmierer zu erstellendes Unterprogramm führen. Dessen Zeilennummer wird am Ende des EVERY-Befehls angegeben.

Wie beim Befehl AFTER gilt

n = T/0.02 Sekunden

Wird unter n die Zahl 50 vereinbart, erfolgt der Sprung in das Unterprogramm jede Sekunde. Wird der Parameter m nicht vereinbart, bedient sich der Interpreter des Taktgebers 0. Eine Wirkung besitzt der Befehl nur während der Programmablaufphase.

Beispiel:

```
5 REM **** Sekunden zaehlen ****
10 CLS
20 EVERY 50 GOSUB 40
30 GOTO 30
40 LOCATE 10,11:PRINT"Sekunden vergangen:";T
50 T=T+1
60 RETURN
```

Querverweis: AFTER, REMAIN

---

## EXP

Syntax: EXP(<numerischer Ausdruck>)
BASIC-Code: 255 und 7 bzw. &FF und &7

Erläuterung:

Über EXP(X) wird die Exponentialfunktion

$$y = e^x$$

aufgerufen. Die Basis e dieser Funktion ist die Eulersche Zahl. Sie besitzt näherungsweise den Wert 2.7182818. Die folgende Abb. 3.13 zeigt den Verlauf dieser Funktion für Argumente zwischen −2.5 und +2.5. Sie erhalten den Basiswert e mit der für den Schneider CPC möglichen maximalen Stellengenauigkeit durch unmittelbaren Aufruf von

```
PRINT EXP(1)
  2.71828183
Ready
```

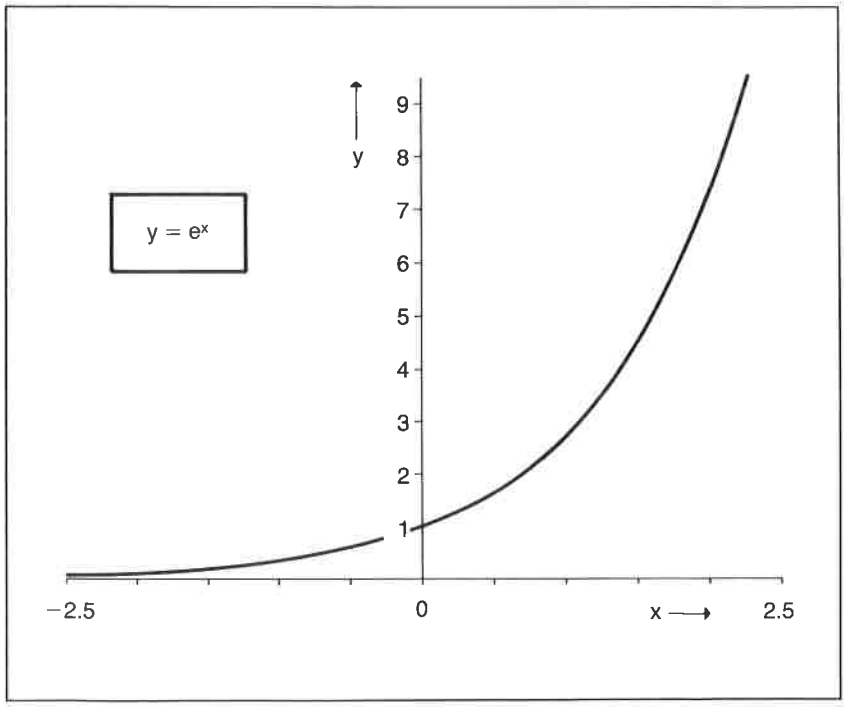



*Abb. 3.13: Die Exponentialfunktion $y = e^x$ für $x = -2{,}5$ bis $x = +2{,}5$*

**FILL** (NUR CPC 664)

Syntax: FILL <Farbe>
BASIC-Code: 221 bzw. &DD

Erläuterung:

Mit dem FILL-Befehl kann beim CPC 664 ein beliebiger Bereich des grafischen Bildschirms mit einer im Parameter <Farbe> vereinbarten Farbe gefüllt werden. Ausgangspunkt für den Füllvorgang ist die aktuelle Position des Grafikcursors. Durch Linien oder Randkurven vollständig geschlossene Bereiche bilden beim Füllvorgang natürliche Grenzen.

Querverweis: GRAPHICS PEN

---

**FIX**

Syntax: FIX (<numerischer Ausdruck>)
BASIC-Code: 255 und 8 bzw. &FF und &8

Erläuterung:

Die Anweisung FIX ermittelt vom numerischen Argument den ganzzahligen, nicht gerundeten (!) Wert. Die Stellen hinter dem Dezimalpunkt werden also einfach abgeschnitten.

Beispiel:

```
PRINT FIX(3.4567)
        3
Ready

PRINT FIX(-3.9876)
        -3
Ready
```

Querverweis: CINT, INT, ROUND

---

**FOR**

Syntax: FOR <Laufvariable> = <numerischer Anfangswert>
TO <numerischer Endwert> [STEP <numerische Schrittweite>]
BASIC-Code: 158 bzw. &9E

Erläuterung:

Der FOR-Befehl eröffnet eine Programmschleife. Die zwischen dem FOR-Teil und dem zugehörigen NEXT-Statement eingeschlossenen Programmzeilen werden so lange wiederholt ausgeführt, bis die Laufvariable den Endwert ihres Definitionsbereichs überschritten hat. Die Laufvariable kann sowohl ganzzahlig als auch gebrochen sein. Bei einfachen Zählvorgängen sollte sie als ganzzahlig vereinbart werden, um die Ablaufgeschwindigkeit zu erhöhen. Der Wert der Variablen wird beginnend bei dem Anfangswert in der gegebenen Schrittweite bei jedem erneuten Programmdurchlauf so lange erhöht, bis die durch den Endwert angegebene Grenze überschritten ist. In diesem Fall wird das nächstfolgende Programmstatement angesprungen. Ist der Endwert kleiner als der Anfangswert, muß im STEP-Teil eine negative Schrittweite angegeben werden.

Beispiel:

```
10 FOR I%=1 TO 1000
20 PRINT I%
30 NEXT I%
```

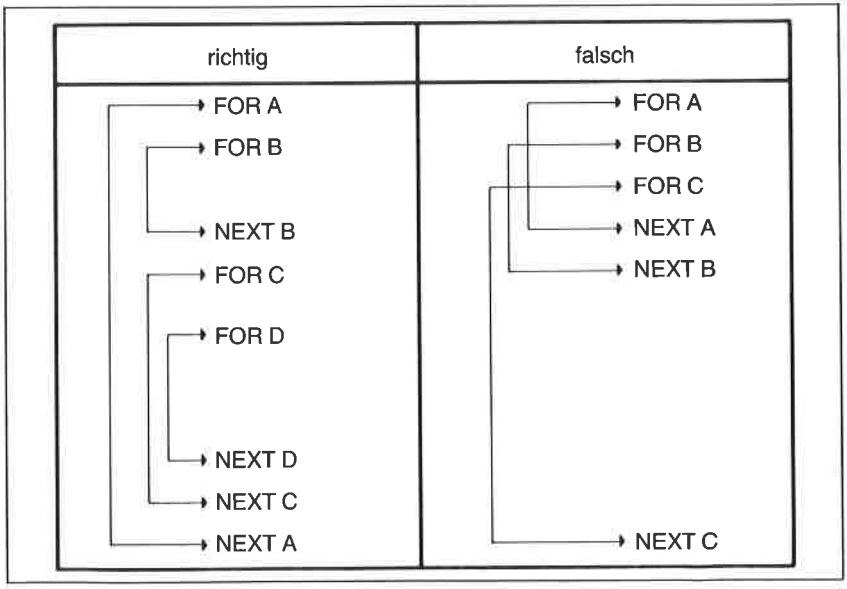



*Abb. 3.14: Erlaubte und unerlaubte Schachtelungen von FOR-NEXT-Schleifen*

Mehrere Schleifen können ineinandergeschachtelt werden, wenn die logische Reihenfolge der Abarbeitung nicht durch unerlaubte Sprünge oder übergreifende Schachtelungen zerstört wird. Abb. 3.14 zeigt erlaubte wie auch unerlaubte Verschachtelungen von Schleifen.

Achtung: Bei der Vereinbarung der Variablengrenzen in hexadezimaler Schreibweise kann es zu Problemen kommen, weil Zahlenwerte oberhalb von &7FFF (also ab 32767) wegen der Interpretation im Zweierkomplement als negativ angesehen werden. Bei der Ausführung der Schleife

```
10 FOR I%=&BCF TO &8000
20 PRINT I%
30 NEXT I%
```

ist das Programm bereits nach dem ersten Schleifendurchlauf beendet, da die Zahl &8000 (in Zweierkomplementnotation = – 32768) kleiner ist als &BCF (entspricht 3023).

Ist die logische Zuordnung der NEXT-Teile eindeutig, kann die Angabe der Variablen im NEXT-Teil weggelassen werden.

---

**FRAME** (nur CPC 664)

Syntax: FRAME
BASIC-Code: 224 bzw. &EO

Erläuterung:

Bewegungen von grafischen Objekten oder Textelementen im TAG-Modus geraten im allgemeinen mit der Bildschirmsynchronisation in Konflikt. Dies führt zu störenden optischen Interferenzerscheinungen (Flackern). Mit dem FRAME-Befehl wird die Ausgabe mit dem Strahlrücklauf aufeinanderfolgender Bilder synchronisiert. Die erwähnten Nebeneffekte werden dadurch beseitigt. Beim CPC 464 wird derselbe Effekt durch den Aufruf der Betriebssystemroutine CALL &BD19 erzielt.

Beispiel:

```
10 CLS
20 TAG
30 FOR I= 1 TO 639
40 FRAME
50 MOVE I,200: PRINT CHR$(224);
60 MOVE I,200:PRINT " ";
70 NEXT I
```

Lassen Sie das Programm einmal mit und einmal ohne den FRAME-Befehl laufen, um die Wirkung besser abschätzen zu können.

Querverweis: TAG, TAGOFF

---

**FRE**

Syntax: FRE(<numerischer Ausdruck>) oder FRE(<Zeichenkette>)
BASIC-Code: 255 und 9 bzw. &FF und &9

Erläuterung:

Über die Anweisung FRE ist der noch verfügbare Speicherumfang abrufbar. In der Form FRE("") wird vor der Werteausgabe noch eine Bereinigung des für Zeichenketten in Anspruch genommenen Speichers durchgeführt. Gerade in der letzten Form ist der FRE-Befehl sehr wertvoll, weil er – regelmäßig innerhalb eines Programms angewendet – ein Überlaufen des für Zeichenketten benutzten Bereichs des Arbeitsspeichers verhindert. Der BASIC-Interpreter benutzt nämlich auch bei mehrfacher Zuweisung von Strings zu ein und derselben Zeichenkettenvariablen nicht denselben Speicherplatz. Er belegt vielmehr immer wieder einen neuen Bereich.

Querverweis: HIMEM, MEMORY

---

**GOSUB**

Syntax: GOSUB <Zeilennummer>
BASIC-Code: 159 bzw. &9F

Erläuterung:

Über GOSUB wird ein als Unterprogramm geschriebener Teil eines Programms angesprungen. Der Beginn dieses Programmteils muß unter „Zeilennummer" angegeben werden. Die Rücksprungadresse nach der Ausführung des Unterprogramms wird vom System auf den Stapel abgelegt. Unterprogramme müssen mit einem RETURN-Befehl beendet werden, der zu einem Rücksprung auf die nächste nach dem Sprungbefehl folgende Zeilennummer führt. Es ist darauf zu achten, daß bei verschachtelten Unterprogrammaufrufen die Rücksprünge in derselben Reihenfolge ausgeführt werden wie die Einsprünge, da ansonsten die Rangfolge des Stapels zerstört wird.

Querverweis: RETURN

---

## GOTO

Syntax: GOTO <Zeilennummer>
BASIC-Code: 160 bzw. &A0

Erläuterung:

Mittels GOTO wird ein unmittelbarer Sprung zu einer Programmzeile ausgeführt, deren Zeilennummer im GOTO-Befehl vereinbart worden ist. Ist diese Zeilennummer im Programm nicht vorhanden, erfolgt eine Fehlermeldung.

---

## GRAPHICS PAPER

Syntax: GRAPHICS PAPER <Farbnummer>
BASIC-Code: 222 32 186 bzw. &DE &20 &BA

Erläuterung:

Mit dem GRAPHICS PAPER-Befehl kann die Hintergrundfarbe bei der Ausgabe grafischer Informationen festgelegt werden. Dies gilt beispielsweise für die Darstellung der Hintergrundflächen von Zeichen bei Verwendung des TAG-Befehls sowie die Zwischenräume maskierter Linien (siehe auch MASK-Befehl).

Querverweis: CLG, GRAPHICS PEN, INK, MASK, TAG, TAGOFF

**GRAPHICS PEN** (nur CPC 664)

Syntax: GRAPHICS PEN (<Farbe>,[<Hintergrund>]
BASIC-Code: 222 32 187 bzw. &DE &20 &BB

Erläuterung:

GRAPHICS PEN bestimmt die Farbe für die Ausgabe grafischer Informationen. Als Farbe sind die Werte 0 bis 15 unter Beachtung der für die unterschiedlichen Betriebsarten geltenden Regeln erlaubt. Für den Hintergrund gilt:

0: nicht transparent
1: transparent

Querverweis: GRAPHICS PAPER, INK, MASK, TAG, TAGOFF

---

**HEX$**

Syntax: HEX$(<numerischer Ausdruck>)
BASIC-Code: 255 und 115 bzw. &FF und &73

Erläuterung:

Die Zeichenkettenfunktion überführt den in Form eines numerischen Ausdrucks oder einer Konstante im Argument angegebenen Wert in die hexadezimale Schreibweise.

Beispiel:

```
PRINT HEX$(65533)
 FFFD
```

Querverweis: BIN$, STR$

---

**HIMEM**

Syntax: HIMEM
BASIC-Code: 255 und 66 bzw. &FF und &42

Erläuterung:

HIMEM ist eine Variable, die die höchste vom BASIC-Interpreter verwendete Speicheradresse in dezimaler Form angibt. Unmittelbar nach

dem Systemstart gibt der CPC 464 ohne angeschlossenes und aktiviertes Diskettenlaufwerk als höchste verfügbare RAM-Adresse den Wert 43903 (&AB7F), der CPC 664 und der CPC 464 mit aktivem Laufwerk dagegen die Adresse 42619 (&A67B) aus. Die 1285 nicht mehr für den Interpreter verfügbaren Bytes nehmen das Diskettenbetriebssystem AMSDOS auf.

Querverweis: FRE, MEMORY

---

**IF ... THEN ... ELSE**

Syntax:
IF <logischer Ausdruck> THEN <Anweisung1> [ELSE <Anweisung2>]
BASIC-Code für IF: 161 bzw. &A1

Erläuterung:

Der IF-Befehl leitet immer einen bedingten Sprung ein. Dieser wird entweder direkt durch Eingabe einer Zeilennummer oder aber indirekt über im Befehl vereinbarte Statements ausgeführt. Der ELSE-Teil des Befehls läßt eine alternative Lösung zu. Bei der Ausführung des Befehls wird zunächst der logische Ausdruck daraufhin untersucht, ob er wahr oder unwahr ist. Ist die durch den logischen Ausdruck formulierte Bedingung erfüllt, wird die im THEN-Teil vereinbarte Anweisung1 ausgeführt. Sie kann aus einem oder mehreren Statements, aber auch nur aus einer Zeilennummer bestehen. Wird der ELSE-Teil weggelassen, springt das Programm bei Nichterfüllung der Bedingung auf das nächstfolgende Statement. Andernfalls wird die im ELSE-Teil vereinbarte Anweisung ausgeführt. Sie kann ebenfalls einfach nur aus einer Zeilennummer bestehen.

Beispiel 1:

```
5 REM **** Normale Loesung durch IF .. THEN ****
10 FOR I%=1 TO 4
20 IF I%>2 THEN 50
30 PRINT "Die Variable ist kleiner als 3"
40 NEXT I%
50 PRINT "Die Grenze ist ueberschritten"
60 END
RUN
Die Variable ist kleiner als 3
Die Variable ist kleiner als 3
Die Grenze ist ueberschritten
Ready
```

Beispiel 2:

```
5 REM **** Loesung durch IF .. THEN .. ELSE ****
10 FOR I%=1 TO 4
20 IF I%>2 THEN PRINT "Die Grenze ist ueberschritten" E
LSE PRINT "Die Variable ist kleiner als 3"
30 NEXT I%
40 END
RUN
Die Variable ist kleiner als 3
Die Variable ist kleiner als 3
Die Grenze ist ueberschritten
Ready
```

Querverweis: WEND, WHILE

## INK

Syntax: INK <Farbwert n>,<Farbe $f_1$>[,<Farbe $f_2$>]
BASIC-Code: 162 bzw. &A2

Erläuterung:

Der INK-Befehl ordnet den für eine bestimmte Betriebsart zulässigen Farbwerten n ($0 \leq n \leq 15$) eine Auswahl von Farben f aus der Farbpalette des CPC 464/664 zu. Die Anzahl gleichzeitig darstellbarer Farben hängt

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Abb. 3.15: Standardfarbzuordnungen beim Systemstart für die Betriebsarten MODE 0,1 und 2*

von der Betriebsart ab. Im MODE 0 sind 15, im MODE 1 vier und im MODE 2 zwei Farben zugleich auf dem Bildschirm darstellbar. Beim Einschalten des Systems entsprechen die Farbzuordnungen denen in Abb. 3.15.

Nach Abb. 3.15 sind in jeder Betriebsart 15 Farbwerte vereinbar. Sie führen jedoch nur im MODE 0 zu 15 unterscheidbaren, gleichzeitig darstellbaren Farben aus der Farbtabelle (siehe auch Abb. 3.5). Die Standardzuordnung kann über den INK-Befehl verändert werden.

Beispiel:

```
INK 6, 22
```

ordnet dem aktuellen Farbparameter n=6 den Farbwert f=22 (Pastellgrün) aus der Farbpalette von insgesamt 26 verschiedenen Farben zu. In der Betriebsart MODE 1 werden hierdurch auch gleichzeitig die Farbparameter 2, 10 und 14 und in der Betriebsart MODE 2 alle geradzahligen Farbparameter auf Pastellgrün umgeschaltet.

Querverweis: PEN und PAPER

---

**INKEY**

Syntax: INKEY(<numerischer Ausdruck>)
BASIC-Code: 255 und 10 bzw. &FF und &A

Erläuterung:

Die Funktion INKEY fragt die Tastatur für eine Zeitdauer von 20 Millisekunden nach der Befehlsausführung daraufhin ab, ob eine durch das Argument vereinbarte Taste betätigt wurde. Das Argument muß den Tastenpositionscode (nicht den ASCII-Code) einer der Tasten des CPC 464/664 festlegen. Die folgende Abb. 3.16 zeigt ein Schema der Tastatur einschließlich der Standard-Tastencodes.

Der von der Funktion an das BASIC-Programm übergebene Wert hängt vom Zustand der angewählten Taste wie der Zusatztasten SHIFT und CONTROL ab, wie Sie der kleinen Tabelle in Abb. 3.17 entnehmen können. Beispiel: Für den Fall, daß die im Argument vereinbarte Taste zum Zeitpunkt der kurzzeitigen Abfrage durch die Hardware nicht betätigt ist, wird der Wert −1 übergeben.

nen Adressen angesprochen werden. Mit dem INP-Befehl werden an den Eingangsleitungen anliegende 8-bit-Daten ins Programm übernommen. Der Interpreter überführt die dualen Datenwerte für die Weiterverarbeitung in Dezimalzahlen.

Querverweis: OUT, WAIT

---

**INPUT**

Syntax:

INPUT [#<Eingabekanal>,][;][<Fragetext>;]<Liste von Variablen>

oder

INPUT [#<Eingabekanal>,][;][<Fragetext>,]<Liste von Variablen>
BASIC-Code: 163 bz. &A3

Erläuterung:

Die INPUT-Anweisung stellt die klassische Möglichkeit in BASIC dar, Werte über unterschiedliche Eingangskanäle an das Programm zu übergeben. In der einfachsten Form INPUT <Variable> wird ein über die Tastatur eingegebener numerischer Wert oder eine Textkonstante an eine entsprechend vereinbarte Variable übergeben.

Die einzelnen Elemente des INPUT-Befehls lauten:

*#Eingangskanal:*

Folgende Vereinbarungen für die Eingabekanäle sind zulässig:

#0–#7: Bildschirmfenster #0 bis #7
#9: externer Massenspeicher

*Fragetext:*

Soll bei der Abarbeitung des INPUT-Befehls eine Meldung auf dem Bildschirm ausgegeben werden, kann diese durch eine Textkonstante festgelegt werden:

```
INPUT "Bitte geben Sie die naechste Ziffer ein";A
```

Der Fragetext wird von der nachfolgenden Variablen durch ein Semikolon oder ein Komma getrennt. Im Falle eines Semikolons wird nach dem

Text ein Fragezeichen ausgegeben. Bei Angabe eines Kommas wird das Fragezeichen unterdrückt.

*Liste von Variablen:*

Mindestens eine numerische oder nichtnumerische Variable ist innerhalb eines INPUT-Befehls zu vereinbaren. Wird mehr als eine Variable angegeben, müssen die Variablen untereinander durch Kommata getrennt werden. Die eingelesenen Konstanten müssen mit dem jeweils vereinbarten Variablen*typ* übereinstimmen. Da alle Eingaben über die Tastatur mit ENTER abgeschlossen werden müssen, springt der Cursor normalerweise in die nächstfolgende Zeile. Angabe eines Semikolons unmittelbar hinter dem INPUT bzw. der Vereinbarung des Eingabekanals unterdrückt diesen Sprung.

Beispiele:

| | |
|---|---|
| INPUT A | erlaubt die Eingabe einer Zahl über die Tastatur. Der Wert wird an die Variable A übergeben. |
| INPUT "Wert: ";A | gibt zusätzlich den Text Wert:? auf dem Bildschirm aus. Der Cursor springt nach der Eingabe in die nächstfolgende Textzeile des Bildschirms. |
| INPUT "Wert: ",A | unterdrückt bei der Ausgabe das Fragezeichen am Ende der Textmeldung. |
| INPUT;"Wert";A | unterdrückt den Sprung des Cursors in die nächstfolgende Textzeile nach der Werteeingabe. |

Querverweis: LINE INPUT, READ

---

**INSTR**

Syntax: INSTR ([<numerischer Ausdruck n>,]<Zeichenkette>, <Suchwort>
BASIC-Code: 255 und 116 bzw. &FF und &74

Erläuterung:

Mit dem Befehl INSTR wird die angegebene Zeichenkette daraufhin gestestet, ob sie den unter „Suchwort" vereinbarten Begriff enthält. <Zeichenkette> wie auch <Suchwort> können entweder aus einer Textkonstanten oder einer Textvariablen bestehen. Die Suche beginnt bei dem im Parameter n vereinbarten Zeichen. Wird n weggelassen, star-

tet die Suche beim ersten Zeichen. Als Ergbnis der Suche wird eine Positionsangabe an das Programm übergeben.

Beispiel:

```
PRINT INSTR "SCHNEIDER","NEIDER"
      4
```

Querverweis: MID$, LEFT$, RIGHT$

---

**INT**

Syntax: INT(<numerischer Ausdruck>)
BASIC-Code: 255 und 12 bzw. &FF und &C

Erläuterung:

Die BASIC-Funktion INT bildet vom Argument den nächstniedrigeren ganzzahligen Wert.

Beispiele:

```
PRINT INT(-3.678)
      -4
PRINT INT(3.678)
      3
```

Querverweis: CINT, FIX , ROUND

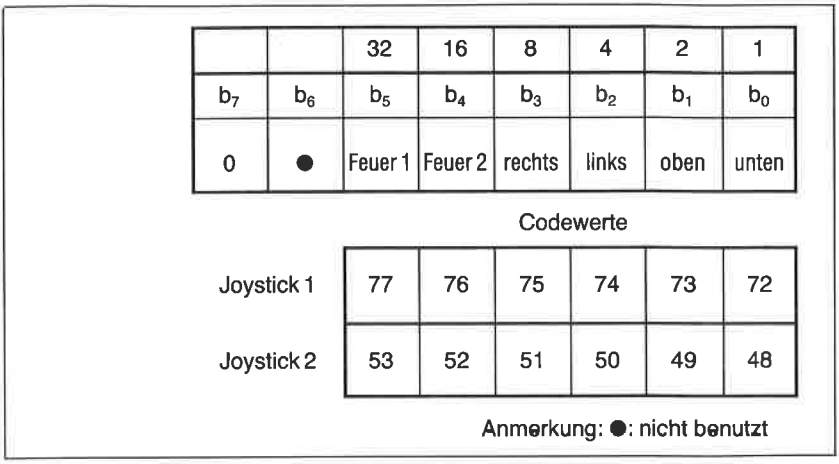
---

**JOY**

Syntax: JOY(<numerischer Ausdruck>)
BASIC-Code: 255 und 13 bzw. &FF und &D

Erläuterung:

Über die BASIC-Funktion JOY wird ein Eingangsregister des Joystickinterfaces auf seinen logischen Inhalt hin abgefragt. Die Bedeutungen der einzelnen Bits sind in der Abb. 3.18 angegeben.



| | | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |
| 0 | ● | Feuer 1 | Feuer 2 | rechts | links | oben | unten |

Codewerte

| | | | | | | |
|---|---|---|---|---|---|---|
| Joystick 1 | 77 | 76 | 75 | 74 | 73 | 72 |
| Joystick 2 | 53 | 52 | 51 | 50 | 49 | 48 |

Anmerkung: ●: nicht benutzt

*Abb. 3.18: Die Statusbits des Joystickregisters*

Der dem binären Datenwort entsprechende Dezimalwert wird an das Programm übergeben.

Programmbeispiel:

```
5 REM ******* JOYSTICK-ORGEL *******
10 TON=200:LAENGE=10
20 IF JOY(0)=0 OR JOY(0)>2 THEN SOUND 1,TON,LAENGE
30 IF JOY(0)=1 AND TON<4000 THEN TON=TON+1:SOUND 1,TON,
LAENGE
40 IF JOY(0)=2 AND TON>10 THEN TON=TON-1:SOUND 1,TON,LA
ENGE
50 PRINT TON:GOTO 20
```

Querverweis: INKEY

---

**KEY**

Syntax:
KEY <ASCII-Code>,[CHR$(m)+]<Zeichenkette>[+CHR$(n)]
BASIC-Code: 164 bzw. &A4

Erläuterung:

Die 32 ASCII-Codes zwischen dezimal 128 (&80) und dezimal 159 (&9F) besitzen einen eigenen Puffer für Zeichenketten bis zu 120 Zeichen

Länge. Sie können über den KEY-Befehl als Funktionstasten für den Schnellaufruf von Befehlen oder Befehlsketten vereinbart werden. Diese werden in der Zeichenkette vereinbart. Die Erweiterungen CHR$(m) und CHR$(n) dienen zur Einführung spezieller Zeichen, die nicht als Bestandteil von Zeichenketten über die Tastatur abgerufen werden können.

Beispiel:

```
KEY 140,"LIST"+ CHR$(13)
```

Der ASCII-Code 140 wird entsprechend der Tastenbelegungstabelle dadurch abgerufen, daß die ENTER-Taste des numerischen Tastenblocks gleichzeitig mit der CTRL-Taste betätigt wird. Entsprechend der Vereinbarung wird unmittelbar danach ein im Speicher abgelegtes Programm auf dem Bildschirm ausgegeben. Die Angabe von CHR$(13) entspricht einem CR (Carriage Return, beim Schneider ENTER). Nähere Informationen zu diesem Themenkreis finden Sie in Kapitel 2.

Querverweis: KEY DEF, Kapitel 2

---

**KEY DEF**

Syntax: KEY DEF <Tastenpositionscode>,<Wiederholung> [,<$ASCII_n$>[,<$ASCII_s$>[,<$ASCII_c$>]]]
BASIC-Code: 164 und 141 bzw. &A4 und &8D

Erläuterung:

Jede Taste ist durch ihren Tastenpositionscode gekennzeichnet, der in Anhang R in vollständiger Form angegeben ist (siehe auch Anhang III des Handbuchs). Bei den weitaus meisten Tasten können durch Einzelbetätigung oder aber in Verbindung mit den Tasten SHIFT und CTRL Zeichen mit unterschiedlichen ASCII-Verschlüsselungen abgerufen werden.

Jede beliebige, durch den Tastenpositionscode näher bezeichnete Taste kann durch den KEY DEF-Befehl mit abweichenden Codes belegt werden. Der Parameter „Wiederholung" wird für den Fall, daß die automatische Zeichenwiederholfunktion aktiv bleiben soll, mit dem Wert 1 versehen. Vereinbarung von 0 schaltet die Wiederholfunktion ab. Für die drei letzten Parameter gilt:

$ASCII_n$: ASCII-Code bei Einzelbetätigung
$ASCII_s$: ASCII-Code bei Betätigung zusammen mit SHIFT
$ASCII_c$: ASCII-Code bei Betätigung zusammen mit CTRL

Beispiel:

Die folgende Tastendefinition legt den ASCII-Code 140 auf die Taste mit dem Dezimalpunkt, wenn diese zusammen mit CTRL betätigt wird. Die anderen Codes werden nicht verändert, müssen aber zur Beachtung der Syntax explizit mit angegeben werden. Die Wiederholfunktion wird abgeschaltet.

```
KEY DEF 7,0,138,138,140
```

Querverweis: KEY, Kapitel 2

---

**LEFT$**

Syntax: LEFT$(<Zeichenkette>,<numerischer Ausdruck n>)
BASIC-Code: 255 und 117 bzw. &FF und &75

Erläuterung:

Die Zeichenkettenfunktion LEFT$ übergibt bei Aufruf die ersten n Zeichen der Zeichenkette an das Programm.

Beispiel:

```
10 CLS
20 FOR I%=1 TO 9
30 PRINT LEFT$("SCHNEIDER CPC 464",I%)
40 NEXT I%
50 END

RUN
S
SC
SCH
SCHN
SCHNE
SCHNEI
SCHNEID
SCHNEIDE
SCHNEIDER
Ready
```

Querverweis: MID$,RIGHT$

## LEN

Syntax: LEN(<Zeichenkette>)
BASIC-Code: 255 und 14 bzw. &FF und &OE

Erläuterung:

Die Function LEN ermittelt die Anzahl der Zeichen, aus denen die im Argument angegebene Zeichenkette besteht. Das Ergebnis ist eine ganze Zahl.

Beispiel:

```
A$="":PRINT LEN("")
        0
```

---

## LET

Syntax: LET <logischer Ausdruck>
BASIC-Code: 165 bzw. &A5

Erläuterung:

Mittels der LET-Anweisung wurden in den frühen Versionen der Programmiersprache BASIC Variablenzuweisungen eingeleitet. In der vorliegenden Version ist diese Form der Zuweisung entbehrlich, da Variablenzuweisungen direkt erfolgen.

Beispiel:

| | |
|---|---|
| Alte Form: | `10 LET A=123` |
| Neue Form: | `10 A=123` |

---

## LINE INPUT

Syntax: LINE INPUT [<#Eingabekanal>,][;][<Textkonstante>;]
<Textvariable>
BASIC-Code: 166 und 163 bzw. &A6 und &A3

Erläuterung:

Mittels der LINE INPUT-Anweisung wird eine Textzeile über den definierten Eingabekanal eingelesen. Ein Semikolon unmittelbar nach dem

INPUT-Teil unterdrückt eine Wiederholung der durch die ENTER-Taste bewirkten ASCII-Kombination (CR)(LF). Wird kein Eingabekanal angegeben, wird als Standardeingabegerät die Tastatur angenommen. Im Gegensatz zum INPUT-Befehl kann die LINE INPUT-Anweisung Satzzeichen wie Kommata, Semikola, Anführungszeichen usw. problemlos verarbeiten. Achten Sie darauf, daß nur *eine* Textvariable angegeben werden darf!

Beispiel:

```
5 REM **** LINE INPUT-Demo ****
10 LINE INPUT "Bitte Text eingeben: ",TEX$
20 PRINT TEX$
RUN
Bitte Text eingeben: Sehr geehrte Damen, sehr geehrte H
erren.
Sehr geehrte Damen, sehr geehrte Herren.
Ready
```

Querverweis: READ, INPUT, INKEY$

---

**LIST**

Syntax: LIST [<Zeilennummernbereich>][,#<Ausgabekanal>]
BASIC-Code: 167 bzw. &A7

Erläuterung:

Mittels der LIST-Anweisung wird ein Programmtext auf dem angegebenen Ausgabekanal ausgegeben. Wird kein Ausgabekanal vereinbart, erfolgt die Ausgabe auf dem Bildschirm. Folgende Formen des LIST-Befehls sind zulässig:

| | |
|---|---|
| LIST | gibt das ganze Programm auf dem Bildschirm des Computers aus. |
| LIST <Zeilennummer1> – <Zeilennummer2> | listet das Programm, beginnend mit der Zeilennummer1 und endend mit der Zeilennummer2. |
| LIST–<Zeilennummer> | gibt das Programm von Beginn an bis zur angegebenen Zeilennummer aus. |

List<Zeilennummer>– führt zur Ausgabe des Programmtextes ab der vereinbarten Zeilennummer bis zum Ende.

Als Ausgabekanäle sind zulässig:

#0 bis #7: Bildschirmfenster #0 bis #7
#8: Drucker

---

## LOAD

Syntax: LOAD <Dateiname>[,<Adresse>]
BASIC-Code: 168 bzw. &A8

Erläuterung:

Mittels LOAD wird eine Datei vom externen Massenspeicher (Kassette oder Diskette) in den Speicher eingelesen. Wird der Parameter <Dateiname> bei Kassettenbetrieb nicht vereinbart, liest das System das nächste erreichbare Programm ein. Bei einer Binärdatei kann eine Zieladresse für die Binärdaten angegeben werden. Wird sie weggelassen, erfolgt die Abspeicherung unter der bei der Programmsicherung vereinbarten Adresse. Bei einem aktivierten Kassettenlaufwerk wird nach Eingabe des Befehls die Meldung

Press PLAY then any key

(zu deutsch: Betätige die PLAY-Taste am Recorder und dann irgendeine Taste) ausgegeben. Ist ein Diskettenlaufwerk angeschlossen und aktiv, erfolgt der Ladevorgang direkt nach der Befehlseingabe.

Achtung: Die mit SHIFT, CAPS LOCK, CTRL und ESC bezeichneten Tasten lösen den Ladevorgang nicht aus!

Der Ladevorgang von Kassette wird durch Kontrollmeldungen auf dem Bildschirm unterstützt. Die Meldungen können unterdrückt werden, wenn als erstes Zeichen des Dateinamens ein Rufzeichen (!) angegeben wird.

Querverweis: SAVE, CAT

## LOCATE

Syntax:
LOCATE [#<Ausgabefenster>,]<x-Koordinate>,<y-Koordinate>
BASIC-Code: 169 bzw. &A9

Erläuterungen:

LOCATE setzt den Textcursor in dem angegebenen Fenster an eine durch die Koordinaten x und y vereinbarte Position. Wird das Ausgabefenster nicht vereinbart, nimmt der Interpreter #0 als gegeben an. Beachten Sie, daß die Position sich immer auf den durch den Fensterrand festgelegten Startpunkt 1,1 in der linken oberen Ecke bezieht (Abb. 3.19).

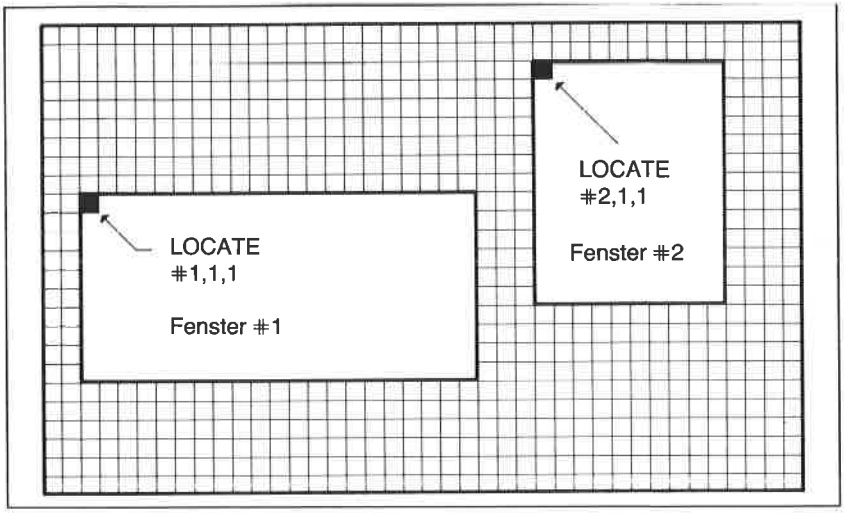



*Abb. 3.19: Erläuterung der Koordinatenangaben im LOCATE-Befehl*

Programmbeispiel:

```
5 REM **** LOCATE-Zufallssteuerung ****
10 MODE 2
20 X=INT((RND(1)*56)+1)
30 Y=INT((RND(1)*25)+1)
40 LOCATE X,Y:PRINT " Hallo, hier bin ich !!!";
50 FOR WARTESCHLEIFE=1 TO 500:NEXT
60 LOCATE X,Y:PRINT "                          ";
70 GOTO 20
```

Querverweis: WINDOW

**LOG**
**LOG10**

Syntax: LOG(X) und LOG10(X)
BASIC-Code: 255 und 15 bzw. &FF und &F (LOG)
255 und 16 bzw. &FF und &10 (LOG10)

Erläuterung:

LOG und LOG10 zählen zu den mathematischen Funktionen. LOG bildet den Logarithmus zur Basis e (natürlicher Logarithmus genannt), LOG10 bildet den Logarithmus zur Basis 10. In der Daten- und Computertechnik ist außerdem noch der Logarithmus zur Basis 2 (Logarithmus Dualis) gebräuchlich. Er kann in einem BASIC-Programm als Funktion über die Beziehung

```
DEF FNLD(X) = LOG(X)/LOG(2)
```

bzw.

```
DEF FNLD(X) = LOG10(X)/LOG10(2)
```

errechnet werden.

Querverweis: EXP

---

**LOWER$**

Syntax: LOWER$(<Zeichenkette>)
BASIC-Code: 255 und 17 bzw. &FF und &11

Erläuterung:

LOWER$ übergibt die im Argument vereinbarte Zeichenkette in Kleinbuchstaben an das Programm. Das Ergebnis der Operation ist eine Textkonstante.

Beispiel:

```
PRINT LOWER$("SCHNEIDER")
      schneider
```

Querverweis: UPPER$

**MASK** (nur CPC 664)

Syntax: MASK [<numerischer Ausdruck>][,<Status des MSB>]
BASIC-Code: 223 bzw. &DF

Erläuterungen:

Mit dem MASK-Befehl können auf einfache Weise gestrichelte Linien oder gerasterte Flächen erzeugt werden. Die Maske entspricht dem Binärmuster der im numerischen Ausdruck vereinbarten Zahl. So erzeugt beispielsweise der Wert &AA bzw. dezimal 170 eine gepunktete Linie. Der zweite Parameter legt fest, ob das höchstwertige Bit gesetzt (1) oder nicht gesetzt (0) sein soll. Wird keine gesonderte Vereinbarung getroffen, gelten die logischen Angaben der binären Maske. Das Programm

```
10 SGP-1
20 FOR I=100 TO 200 STEP 2
30 SG=SG/2: IF SG - INT(SG)< 0.1 THEN SG -0
40 MASK 170,SG
50 MOVE 100,I:DRAW 100,I
60 NEXT I
```

gibt beispielsweise eine gerasterte Fläche auf dem Bildschirm aus. Durch die Abfrage in Zeile 20 wird das höchstwertige Bit der Maske im Wechsel als gesetzt und als nicht gesetzt vereinbart. Falls Sie sich alle möglichen Maskierungsstrukturen ansehen möchten, versuchen Sie es einmal mit dem nachfolgenden kleinen Programm.

```
10 MODE 1
20 FOR I% = 0 TO 255
30 MASK I%,1
40 MOVE 0,I%:DRAWR 639,0
50 NEXT I%
60 END
```

Ändern Sie bitte bei weiteren Programmläufen den Darstellungsmodus, um dic Wirkung in allen 3 Abbildungsarten kennenzulernen.

Querverweis: DRAW, DRAWR, GRAPHICS PAPER, GRAPHICS PEN

## MAX

Syntax: MAX(<Liste numerischer Ausdrücke>)
BASIC-Code: 255 und 118 bzw. &FF und &76

Erläuterung:

Die Funktion MAX errechnet aus den im Argument angegebenen Elementen der numerischen Liste den maximalen Wert.

Beispiele:

```
PRINT MAX(-2,-5,-6,-1)
      -1

PRINT MAX(-2,-3,-1,3)
      3
```

Querverweis: MIN

---

## MEMORY

Syntax: MEMORY (<Adresse>)
BASIC-Code: 170 bzw. &AA

Erläuterung:

Der Interpreter geht von einer Standardaufteilung des ihm zugänglichen Adreßbereichs aus. Die Speicherzelle mit der höchstmöglichen, ihm zugänglichen Adresse kann vom Anwender durch den MEMORY-Befehl geändert werden. Dies ist dann interessant, wenn ein vor dem Zugriff des BASIC-Interpreters geschützter Bereich für die Abspeicherung von Maschinenroutinen oder speziellen Daten geschaffen werden soll. Die aktuelle obere Speichergrenze kann durch HIMEM abgefragt werden.

Querverweis: HIMEM, FRE

---

## MERGE

Syntax: MERGE [<Dateiname>]
BASIC-Code: 171 bzw. &AB

Erläuterung:

Mittels des MERGE-Befehls kann ein Programm von einem externen Datenträger eingelesen und mit dem bereits im Speicher befindlichen Programm verkettet werden. Der Befehl ist sowohl für Kassetten- als auch für Diskettenbetrieb gültig. Ist das erste Zeichen des Dateinamens ein Rufzeichen (!), dann werden die ansonsten beim Laden üblichen Kontrollmeldungen auf dem Bildschirm unterdrückt. Wird kein Dateiname angegeben, dann wird das nächstfolgende, auf Kassette erreichbare Programm geladen. Programmzeilen mit identischen Zeilennummern in beiden Programmen werden durch das nachgeladene überschrieben. Alle Variablen und anwenderdefinierten Funktionen werden wirkungslos. Offene Dateien werden geschlossen.

*Hinweis:* Beim CPC 464 kommt es im Zusammenhang mit dem Betrieb eines externen Diskettenlaufwerkes beim MERGE Befehl zu Problemen wegen eines Fehlers im Betriebssystem. Das folgende kleine Programm (Betriebssystem-Patch) behebt diesen Fehler. Es wird mit freundlicher Genehmigung der Firma ESCON, 8050 Freising, wiedergegeben.

Programmbeispiel:

```
100 MEMORY HIMEM-41
110 DEF FNMSB(A)=&FF AND INT(A/256)
120 DEF FNLSB(A)=&FF AND UNT(A)
130 FOR I=HIMEM+1 TO HIMEM+38
140 READ BYTE
150 POKE I,BYTE
160 NEXT I
170 POKE HIMEM  +3,FNLSB(HIMEM+39)
180 POKE HIMEM  +4,FNMSB(HIMEM+39)
190 POKE HIMEM  +9,FNLSB(HIMEM+41)
200 POKE HIMEM +10,FNMSB(HIMEM+41)
210 POKE HIMEM +18,FNLSB(HIMEM+1)
220 POKE HIMEM +19,FNMSB(HIMEM+1)
230 POKE HIMEM +39,PEEK(&BC80+0)
240 POKE HIMEM +40,PEEK(&BC80+1)
250 POKE HIMEM +41,PEEK(&BC80+2)
260 POKE &BC80  +0,&C3
270 POKE &BC80  +1,FNLSB(HIMEM+1)
280 POKE &BC80  +2,FNMSB(HIMEM+1)
290 DATA &E5,&2A,&00,&00,&22,&80,&BC
300 DATA &3A,&00,&00,&32,&82,&BC
310 DATA &CD,&80,&BC,&21,&00,&00
320 DATA &22,&81,&BC,&21,&80,&BC
330 DATA &36,&C3,&E1,&D8,&C8,&FE,&1A
340 DATA &37,&3F,&C0,&B7,&37,&C9
```

Querverweis: CHAIN, CHAIN MERGE, LOAD

**MID$**

Syntax: MID$(<Zeichenkette>,<numerischer Ausdruck $n_1$>
[,<numerischer Ausdruck $n_2$>])
BASIC-Code: 172 bzw. &AC

Erläuterung:

Mit der Zeichenkettenfunktion (Stringfunktion) MID$ kann aus der vereinbarten Zeichenkette ein Textbestandteil zwischen $n_1$ und $n_1+n_2$ isoliert werden.

Beispiel:

```
A$ = " DONAUDAMPFSCHIFFAHRT"
B$=MID$(A$,6,5)
PRINT LEFT$(B$,1)+LOWER$(MID$(B$,2,4))
      Dampf
```

Querverweis: LEFT$, RIGHT$

---

**MIN**

Syntax: MIN(<Liste numerischer Ausdrücke>)
BASIC-Code: 255 und 119 bzw. &FF und &77

Erläuterung:

Die Funktion MIN ermittelt aus den im Argument angegebenen Elementen einer Liste numerischer Ausdrücke den minimalen Wert.

Beispiel:

```
MIN( -3,-2, 5, 7)
     -3
```

Querverweis: MAX

---

**MOD**

Syntax: <Argument1> MOD <Argument2>
BASIC-Code: 251 bzw &FB

Erläuterung:
Die MOD(ulo)-Funktion ermittelt den ganzzahligen Rest nach Berechnung des Quotienten Argument1/Argument2

---

**MODE**

Syntax: MODE <n>
BASIC-Code: 173 bzw. &AD

Erläuterung:

Mittels der MODE-Anweisung wird die Betriebsart für die Darstellung von Text und Grafik festgelegt. Der Parameter n kann die Werte 0, 1 und 2 annehmen. MODE 1 ist die Standardbetriebsart nach dem Systemstart oder einem durch gleichzeitiges Betätigen der Tasten CTRL, SHIFT und ESC ausgelösten Kaltstart.

Für die einzelnen Betriebsarten gelten folgende Randbedingungen:

| | | | |
|---|---|---|---|
| MODE 0 | 25 Zeilen/20 Zeichen | 160*200 Pixel | 16 Farben |
| MODE 1 | 25 Zeilen/40 Zeichen | 320*200 Pixel | 4 Farben |
| MODE 2 | 25 Zeilen/80 Zeichen | 640*200 Pixel | 2 Farben |

Näheres zu diesem Themenkreis finden Sie in Kapitel 2.

Querverweis: WINDOW, ORIGIN

---

**MOVE**

Syntax:
CPC 464: MOVE <x-Koordinate>,<y-Koordinate>
CPC 664: MOVE <x-Koordinate>,<y-Koordinate>[,[<Farbnummer>][,<Farbmodus>]]
BASIC-Code: 174 bzw. &AE

Erläuterung:

Der MOVE-Befehl setzt den Grafikcursor (unsichtbar) auf einen durch die Koordinatenangaben definierten Punkt. Die absolute Lage von x und y auf dem Bildschirm hängt von der Definition des Koordinatennullpunktes ab. Dieser liegt nach dem Systemstart in der unteren linken Ecke des

Grafikfensters. Er kann durch den ORIGIN-Befehl vom Anwender geändert werden. Die Angabe von Koordinatenwerten außerhalb des Grafikfensters führt zu keiner Fehlermeldung. Beim CPC 664 kann zusätzlich eine neue Farbe und ein Farbmodus vereinbart werden. Der Farbmodus legt fest, welche logische Verknüpfung die neue Farbe mit der bereits am Zielpunkt vereinbarten Farbe aufweisen soll. Die Farbmodi lauten:

0: normal
1: XOR (Exklusiv-ODER-Verknüpfung)
2: UND-Verknüpfung
3: ODER-Verknüpfung

Programmbeispiel:

```
5 REM **** MOVE-Quadrate ****
10 MODE 2
20 PLOT 0,0
30 FOR QUADRATE=0 TO 620 STEP 10
40 DRAWR 20,0:DRAWR 0,20:DRAWR -20,0:DRAWR 0,-20
50 MOVE QUADRATE,P+QUADRATE:P=P-4
60 NEXT
```

Querverweis: DRAW, DRAWR ,MOVER, ORIGIN, PLOT, PLOTR, TEST, TESTR, XPOS, YPOS

---

**MOVER**

Syntax:
CPC 464: MOVER <x-Koordinatenoffset>,<y-Koordinatenoffset>
CPC 664: MOVER <x-Koordinatenoffset>,<y-Koordinatenoffset>
[,[<Farbnummer>][,<Farbmodus>]]
BASIC-Code: 175 bzw. &AF

Erläuterung:

MOVER bewegt den Grafikcursor relativ zur aktuellen Position um den im Argument vereinbarten Offset in x- und y-Richtung. Ansonsten gelten dieselben Randbedingungen wie für den MOVE-Befehl.

Programmbeispiel:

```
5 REM **** MOVER-Quadrate ****
10 MODE 2
20 PLOT 0,0
30 FOR QUADRATE=0 TO 120 STEP 10
40 DRAWR 20,0:DRAWR 0,20:DRAWR -20,0:DRAWR 0,-20
50 MOVER QUADRATE,P+QUADRATE:P=P-4
60 NEXT
```

Querverweis: DRAW, DRAWR, MOVE, ORIGIN, PLOT, PLOTR, TEST, TESTR, XPOS, YPOS

---

**NEW**

Syntax: NEW
BASIC-Code: 177 bzw. &B1

Erläuterung:

NEW löscht das im Speicher liegende aktuelle Programm und alle Variablen dadurch, daß der Zeiger für das Programmende auf den Anfang des BASIC-Speicherbereichs gesetzt wird. Das Programm ist also unmittelbar nach Eingabe von NEW noch vorhanden, aber nicht mehr für den Interpreter verfügbar.

---

**ON BREAK CONT**

Syntax: ON BREAK CONT
BASIC-Code: 179 32 138 bzw. &B3 &20 &8B

Erläuterung:

Normalerweise kann jedes BASIC-Programm durch Betätigen der ESC-Taste im Lauf unterbrochen werden. Der Befehl ON BREAK CONT setzt diese Taste außer Kraft. Es kann nur noch ein Systemreset ausgeführt werden.

Querverweis: ON BREAK GOSUB, ON BREAK STOP

## ON GOSUB
## ON GOTO

Syntax:
ON <numerischer Ausdruck> GOSUB <Liste von Zeilennummern>
ON <numerischer Ausdruck> GOTO <Liste von Zeilennummern>
BASIC-Code: 178 und 159 bzw. &B2 und &9F
oder 178 und 160 bzw. &B2 und &A0

Erläuterung:

Der ON GOSUB- oder auch ON GOTO-Befehl errechnet zunächst das ganzzahlige Ergebnis n (gerundet) des numerischen Ausdrucks und springt dann in die im n-ten Listenelement angegebene Unterroutine oder Zeilennummer.

Programmbeispiel:

```
5 REM **** ON variable GOSUB ***
10 INPUT "Zahl: ";i:IF i<1 OR i>3 THEN 10
20 ON i GOSUB 100,200,300
30 GOTO 10
100 PRINT "Sie haben 1 eingegeben.":RETURN
200 PRINT "Sie haben 2 eingegeben.":RETURN
300 PRINT "Sie haben 3 eingegeben.":RETURN
```

Querverweis: GOTO, GOSUB

---

## ON BREAK GOSUB

Syntax: ON BREAK GOSUB <Zeilennummer>
BASIC-Code: 179 und 159 bzw. &B3 und &9F

Erläuterung:

Ein BASIC-Programm kann durch zweimaliges Betätigen der mit ESC bezeichneten Taste unterbrochen werden. Der Interpreter kehrt sofort in den Befehlsmodus zurück. Durch den Befehl ON BREAK GOSUB wird dieser normale Unterbrechungsablauf unterbunden und ein Sprung in ein

Unterprogramm ausgelöst, dessen Zeilennummer im GOSUB-Teil der Anweisung vereinbart werden muß. Damit der Befehl vom Programmstart an eine Wirkung besitzt, sollte er in der ersten Zeile des Programmtextes vereinbart werden.

Programmbeispiel:

```
5 REM **** List- und Abbruchschutz ****
10 CLS
20 PRINT "Abbruch durch zweimal <ESC> !!"
30 ON BREAK GOSUB 65000
40 GOTO 40
65000 PRINT CHR$(7);
65010 CLS
65020 PRINT "Dieses Programm kann nicht abgebrochen wer
den !!"
65030 PRINT:PRINT "Bitte eine Taste druecken !!":CALL &
DD10
65040 CLS
65050 PRINT "Abbruch durch zweimal <ESC> !!"
65060 RETURN
```

Querverweis: ON BREAK STOP

---

## ON BREAK STOP

Syntax: ON BREAK STOP
BASIC-Code: 179 und 206 bzw. &B3 und &CE

Erläuterung:

Durch den Befehl ON BREAK STOP wird in Programmen, die zu Beginn einen ON BREAK GOSUB-Befehl enthalten, die übliche Unterbrechungsbehandlung des Interpreters durch Rücksprung in den Befehlsmodus wiederhergestellt. Die Anweisung kann beispielsweise Teil von Unterprogrammen sein, um einen nochmaligen Ansprung nach einem BREAK zu verhindern.

Querverweis: ON BREAK GOSUB

---

## ON ERROR GOTO

Syntax: ON ERROR GOTO <Zeilennummer>
BASIC-Code: 180 und 160 bzw. &B4 und &A0

Erläuterung:

Jeder im Programm auftretende Fehler führt zu einem Sprung in eine Fehlerbehandlungsroutine, der Ausgabe einer Fehlermeldung auf dem Bildschirm und zu einem Rücksprung in den Befehlsmodus. Diese normale Abfolge kann durch den ON ERROR GOTO-Befehl unterbunden werden. Die Vereinbarung sollte Bestandteil der ersten ausführbaren Programmzeile sein. Im Fehlerfall springt der Interpreter unter Kontrolle des Betriebssystems zu der im GOTO-Teil der Anweisung vereinbarten Zeilennummer.

Programmbeispiel:

```
5 REM **** Fehlerueberpruefung durch ON ERROR ****
10 MODE 2
20 ON ERROR GOTO 100
30 FOR I=1 TO 200
40 I$=I:REM Fehler !!!
50 NEXT
100 PRINT "Im Programm ist ein Fehler aufgetreten.
110 PRINT "Bitte ueberpruefen Sie es.
120 PRINT:PRINT "<< Taste druecken >>":CALL &BB18
130 CLS:LIST
```

Querverweis: ERR, ERL, RESUME

---

**ON SQ GOSUB**

Syntax: ON SQ(<Kanalnummer n>) GOSUB <Zeilennummer>
BASIC-Code: 181 und 159 bzw. &B5 und &9F

Erläuterung:

Jedem der Tonausgabekanäle ist ein Pufferspeicher zugeordnet, der bis zu vier SOUND-Befehle in einer Warteschlange aufnehmen kann. Nach dem Ende einer aktuellen Tonausgabe wird die nächste im Puffer enthaltene SOUND-Vereinbarung ausgelesen und vom Soundgenerator bearbeitet. Im Puffer rücken die nachfolgend gespeicherten Informationen um einen Platz weiter. Der freiwerdende Platz kann dann wieder mit einer neuen SOUND-Anweisung gefüllt werden. Wird im Programm der Befehl ON SQ GOSUB vereinbart, erzeugt das System in dem zuvor

genannten Fall ein Unterbrechungssignal, das ein laufendes Programm veranlaßt, in ein Unterprogramm zu springen. Dessen Zeilennummer wird im GOSUB-Teil der Anweisung vereinbart. In der Regel wird dieses Unterprogramm einen neuen SOUND-Befehl enthalten, der den freigewordenen Platz in der Warteschlange wieder füllt. Nach dieser Operation kehrt das System ins aufrufende Programm zurück. Im Gegensatz zum ON BREAK- oder ON ERROR-Befehl muß der ON SQ-Befehl nach einmaliger Ausführung wieder aktiviert werden. Es ist deshalb üblich, den Befehl im Unterprogramm vor dem notwendigen RETURN-Befehl erneut aufzurufen. Ein Beispiel hierzu finden Sie im Kapitel 5. Für den Parameter <Kanalnummer n> gilt:

Kanal A: n=1
Kanal B: n=2
Kanal C: n=4

Querverweis: SOUND, SQ, ENT, ENV, RELEASE

---

**OPENIN**

Syntax: OPENIN <Dateiname>
BASIC-Code: 182 bzw. &B6

Erläuterung:

Mittels OPENIN wird eine unter <Dateiname> vereinbarte Datei von einem externen Massenspeicher für einen Lesevorgang geöffnet. Enthält der Dateiname als erstes ein Rufzeichen (!), dann werden bei Kassettenbetrieb die üblichen Kontrollmeldungen auf dem Bildschirm unterdrückt.

Querverweise: CLOSEIN, CLOSEOUT, OPENOUT

---

**OPENOUT**

Syntax: OPENOUT <Dateiname>
BASIC-Code: 183 bzw. &B7

Erläuterung:

Wie OPENIN, nur umgekehrt; d. h. Öffnen einer Datei für die Ausgabe auf einen externen Massenspeicher.

Querverweise: CLOSEIN, CLOSEOUT, OPENIN

## ORIGIN

Syntax: ORIGIN <x-Koordinate>,<y-Koordinate>[,<linker Rand>,<rechter Rand>,<oberer Rand>,<unterer Rand>]
BASIC-Code: 184 bzw. &B8

Erläuterung:

Der ORIGIN-Befehl erfüllt zwei verschiedene Aufgaben: In seiner einfachsten Form definiert er die Lage des Bildpunktes mit den Koordinaten x=0 und y=0. Nach dem Systemstart liegt dieser Punkt im Gegensatz zu den meisten anderen Computersystemen in der linken unteren Ecke des Grafikausgabefensters. Dieses ist mit dem Standard-Textausgabefenster identisch (Abb. 3.20).

In der erweiterten Form kann neben dem Nullpunkt des Koordinatensystems noch ein durch Angabe der Koordinaten der Eckpunkte aktivierter Grafikbereich als Ausschnitt des maximal möglichen Fensters definiert werden, wie dies ebenfalls in Abb. 3.20 erläutert ist. Die mit Grafikbefehlen wie beispielsweise PLOT oder DRAW angesteuerten Bildpunkte führen dann nur in dem durch ORIGIN definierten Grafikfenster zu sichtbaren Ergebnissen.

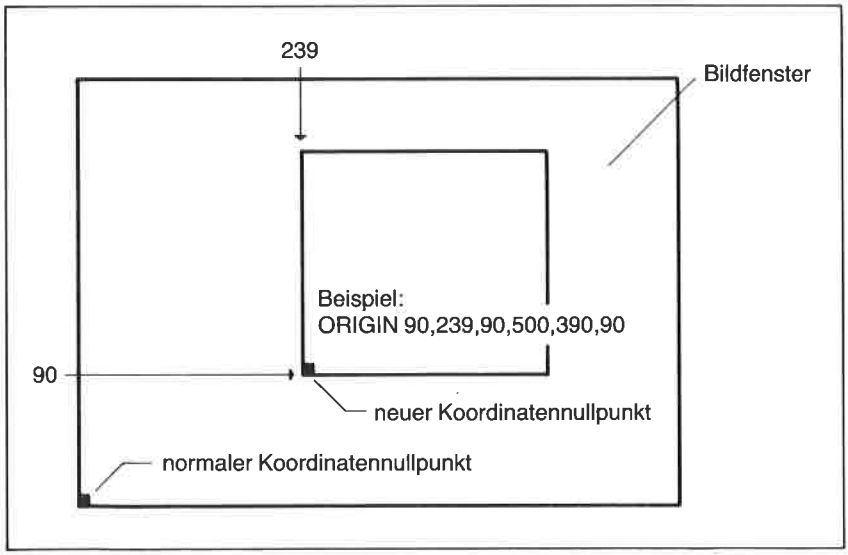



*Abb. 3.20: Zur Definition des Koordinatennullpunktes sowie eines Grafikfensters*

Nähere Einzelheiten zu diesem Thema finden Sie in Kapitel 4 dieses Buchs.

Programmbeispiel:

```
5 REM **** Grafikfenster-Demo ****
10 MODE 2
20 ORIGIN 0,0,0,640,0,399
30 LOCATE 2,3:PRINT"Ohne Grafik-Fenster:":GOSUB 100
40 PRINT:PRINT "Bitte eine Taste druecken !!":CALL &BB1
8:MODE 2
50 ORIGIN 0,0,100,300,100,300
60 LOCATE 2,3:PRINT "Mit Grafik Fenster:":GOSUB 100
70 END
100 PLOT 150,250:DRAWR 200,0:DRAWR 0,-200:DRAWR -200,0:
DRAWR 0,200
110 RETURN
```

Querverweis: WINDOW

---

## OUT

Syntax: OUT <Adresse des Ausgangstors>,<numerischer Ausdruck>
BASIC-Code:185 bzw. &B9

Erläuterung:

Die Hardware des Schneider CPC besitzt eine Reihe von Ausgangsregistern von 8 bit Wortbreite als Bestandteile von Schnittstellenbausteinen, die durch konkrete Adreßangaben angesprochen werden können. Über den Befehl OUT ist es möglich, 8-bit-Datenwerte (im Bereich zwischen 0 und 255) an die Ausgänge zu legen. Eine detaillierte Erläuterung geht über den Rahmen dieses Buchs hinaus.

Querverweis: WAIT, INP

---

## PAPER

Syntax: PAPER [#<Ausgabefenster>,]<Farbe n>
BASIC-Code: 186 bzw. &BA

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Abb. 3.21: Die Standardfarbvereinbarungen für den PAPER-Befehl in den Betriebsarten MODE 0 bis MODE 2*

Erläuterung:

PAPER definiert für ein vereinbartes Ausgabefenster im Bereich zwischen #0 und #7 eine Hintergrundfarbe. Der Parameter n kann grundsätzlich Werte zwischen 0 und 15 annehmen. Nicht jeder dieser 16 Parameterwerte führt jedoch in den Betriebsarten MODE 1 und MODE 2 zu unterschiedlichen Farben (siehe Abb. 3.21).

Die Farbzuordnung kann über den Befehl INK verändert werden. Nähere Hinweise finden Sie in Kapitel 2 dieses Buchs.

Querverweis: INK, PEN, WINDOW

---

**PEEK**

Syntax: PEEK(<Adresse>)
BASIC-Code: 255 und 18 bzw &FF und &12

Erläuterung:

Von der BASIC-Ebene aus ist es möglich, mit dem PEEK-Befehl den Inhalt jeder einzelnen Speicherzelle im Adreßbereich zwischen 0 und 65535 auszulesen.

Mit dem nachfolgend angegebenen kleinen Programm können Sie sehr komfortabel einen sogenannten Memorydump durchführen. Es zeigt

nach Eingabe der Startadresse im MODE 2 immer Blöcke des Schreib-/Lesespeichers von 64 Werten in übersichtlich geordneter Form an. Ein Zugriff auf den Inhalt des Festwertspeichers ist ohne Tricks nicht direkt möglich.

Programm zur Ausgabe von Speicherinhalten

```
5 REM *** Memory-Dump ***
10 MODE 2
20 DIM IN(8),IN$(8)
30 LOCATE 27,12:INPUT "ANFANGSADRESSE (Dez.): ",A1
40 CLS
50 WINDOW #0,1,80,4,19:WINDOW #1,1,80,1,3:WINDOW #2,22,
70,23,23:WINDOW #4,1,80,20,20
60 PRINT #4,STRING$(71,"-")
70 PRINT #1:PRINT #1,USING"\    \";" ADR.";:FOR J%=0 TO
7:PRINT #1,USING "####";J%;:NEXT J%:PRINT #1,"
    ASCII - TEXT"
80 PRINT #1,STRING$(71,"-")
90 B=0
100 FOR I%=1 TO 8
110 IN(I%)=PEEK(A1+I%-1):IF IN(I%)> 31 AND IN(I%)<=127
THEN IN$(I%)=CHR$(IN(I%)) ELSE IN$(I%)="."
120 NEXT I%
130 PRINT USING "#####";A1;:FOR I%=1 TO 8:PRINT USING "
####";IN(I%);:NEXT I%
140 PRINT "      ";
150 FOR I%=1 TO 8: PRINT USING "\  \";IN$(I%);:NEXT I%
160 PRINT
170 A1=A1+8
180 B=B+1:IF B>15 THEN B=0:PRINT #2,"<<< Bitte eine Tas
te druecken >>>":CALL &BB18:CLS:CLS #2:GOTO 100
190 GOTO 100
200 PRINT #0,STRING$(71,"-")
```

Querverweis: POKE

---

## PEN

Syntax:
CPC 464: PEN [#<Ausgabefenster>,]<Farbe n>
CPC 664: PEN [#<Ausgabefenster>,]<Farbe n>[,<Hintergrundmodus>]
BASIC-Code: 188 bzw. &BC

Erläuterung:

Textausgabe- wie auch Grafikanweisungen benutzen zur Darstellung von Informationen Vordergrundfarben, die mit der PEN-Anweisung verein-

bart werden. Wie beim PAPER-Befehl auch, kann der Farbparameter n Werte zwischen 0 und 15 annehmen. Im Hinblick auf die Farbdefinitionen orientieren Sie sich bitte an der vorangegangenen Abb. 3.21. Wird kein Ausgabefenster explizit angegeben, gilt die Vereinbarung für das Fenster mit der Nummer 0. Beim CPC 664 kann neben der Vordergrundfarbe noch ein Hintergrundmodus vereinbart werden. Es gilt:

0: nicht transparent
1: transparent

Querverweis: INK, PAPER, WINDOW

---

**PI**

Syntax: PI

Erläuterung:

PI entspricht der mathematischen Konstanten $\pi$ mit dem Wert 3.14159265.

Querverweis: DEG, RAD, SIN, COS, TAN, ATN

---

**PLOT**

Syntax:
CPC 464: PLOT <x-Koordinate>,<y-Koordinate>[,<Farbe n>]
CPC 664: PLOT <x-Koordinate>,<y-Koordinate>[,<Farbe n>]
[,<Farbmodus>]]
BASIC-Code: 188 bzw. &BC

Erläuterung:

Mittels des Befehls PLOT wird ein Bildpunkt an der Stelle x,y auf dem Bildschirm gesetzt. Wird die Farbe nicht gesondert angegeben, findet die zuletzt benutzte Farbe Anwendung. Der Parameter n darf zwischen 0 und 15 liegen. In MODE 1 und MODE 2 sind die Parameter mehrfach mit derselben Farbe belegt (siehe auch PAPER). Die Koordinatenangaben dürfen unabhängig von der Betriebsart im Bereich zwischen (0,0) und (639,399) liegen (Abb. 3.22). Der Grafikprozessor nimmt die Anpassung an die jeweils gültige Pixelauflösung automatisch vor. Beim CPC 664 kann zusätzlich ein Farbmodus vereinbart werden. Der Farbmodus legt

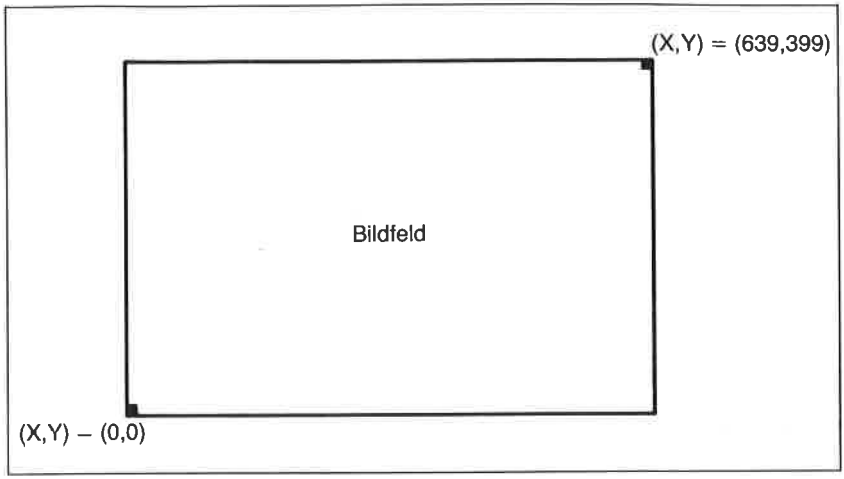



*Abb. 3.22: Zur Definition von Bildpunktkoordinaten in den Betriebsarten MODE 0 bis MODE 2*

fest, welche logische Verknüpfung die Farbe n mit der bereits am Zielpunkt vereinbarten Farbe aufweisen soll. Die Farbmodi lauten:

0: normal
1: XOR (Exklusiv-ODER-Verknüpfung)
2: UND-Verknüpfung
3: ODER-Verknüpfung

Das nachfolgend angegebene kleine Programm zeichnet mit hoher Geschwindigkeit einen fensterfüllenden Kreis im MODE 1.

Programmbeispiel:

```
5 REM **** Kreis plotten ****
10 MODE 1:DEG
20 FOR I%=0 TO 45
30 X=199*COS(I%):Y=199*SIN(I%)
40 PLOT 320+X,199+Y
50 PLOT 320+X,199-Y
60 PLOT 320-X,199+Y
70 PLOT 320-X,199-Y
80 PLOT 320+X,199+Y
90 PLOT 320-Y,199+X
100 PLOT 320+Y,199-X
110 PLOT 320-Y,199-X
120 NEXT I%
130 END
```

Probieren Sie das Programm aus. Es lohnt sich.

Querverweis: DRAW, DRAWR, MOVE, MOVER, ORIGIN, PLOTR, TEST, TESTR, XPOS, YPOS

---

**PLOTR**

Syntax:
CPC 464: PLOTR <x-Koordinate>,<y-Koordinate>[,<Farbe n>]
CPC 664: PLOTR <x-Koordinatenoffset>,<y-Koordinatenoffset>
[,[<Farbnummer>][,<Farbmodus>]]
BASIC-Code: 189 bzw. &BD

Erläuterung:

Der Befehl PLOTR verhält sich wie der Befehl PLOT. Die Koordinatenangaben werden jedoch *relativ* zur aktuellen Position des Grafikcursors vorgenommen. Beim CPC 664 kann zusätzlich ein Farbmodus vereinbart werden. Der Farbmodus legt fest, welche logische Verknüpfung die Farbe mit der bereits am Zielpunkt vereinbarten Farbe aufweisen soll. Die Farbmodi lauten:

0: normal
1: XOR (Exklusiv-ODER-Verknüpfung)
2: UND-Verknüpfung
3: ODER-Verknüpfung

Querverweis: DRAW, DRAWR, MOVE, MOVER, ORIGIN, PLOT, TEST, TESTR, XPOS, YPOS

---

**POKE**

Syntax: POKE (<Adresse>), <numerischer Ausdruck>
BASIC-Code: 190 bzw. &BE

Erläuterung:

Über den Befehl POKE können gezielt Bytes des Schreib-/Lesespeichers mit Daten gefüllt werden, die unter der <Adresse> angesprochen wurden. Der <numerische Ausdruck> darf zwischen 0 und 255 liegen, da in einem Byte nicht mehr als 8 Bits gesetzt werden können. Beachten Sie bei Experimenten, daß Teile des Schreib-/Lesespeichers vom Betriebssystem

verwendet werden. Änderungen der Speicherinhalte können dazu führen, daß sich das System „aufhängt" und nicht mehr von außen ansprechbar ist.

Querverweis: PEEK

---

**POS**

Syntax: POS(#<Ausgabekanal>)
BASIC-Code: 255 und 120 bzw. &FF und &78

Erläuterung:

Mittels POS läßt sich die horizontale Position des Textcursors im angesprochenen Ausgabefenster oder aber die aktuelle Position des Druckkopfes eines angeschlossenen Druckers (Ausgabekanal #8) ermitteln.

Beispiel: Steht der Textcursor im Fenster #5 in der dritten Zeile und in der fünften Spalte, dann führt der Befehl

```
PRINT POS(#5)
```

zur Ausgabe des Wertes 5.

Querverweis: VPOS

---

**PRINT**

Syntax: PRINT [#<Ausgabekanal>,][Ausgabeliste1][USING <Formatangabe>;] [Ausgabeliste2][;]
BASIC-Code: 191 bzw. &BF

Erläuterung:

Der PRINT-Befehl ist der Standardausgabebefehl des BASIC-Interpreters. Je nach Vereinbarung des Ausgabekanals wirkt er auf den Bildschirm, ein angeschlossenes dokumentierendes Peripheriegerät wie einen Drucker bzw. einen Plotter oder auf einen externen Massenspeicher (Kassette, Diskette). Die Ausgabe erfolgt dabei entweder unformatiert oder nach vorgegebenen Formatangaben über den Zusatz USING.

Die Ausgabekanäle lauten:

#0–#7: Textfenster 0 bis 7
#8: Drucker
#9: externer Massenspeicher

PRINT [#<Ausgabekanal>] ohne weitere Parameterangaben erzeugt eine Leerzeile.

Die Elemente der Ausgabeliste müssen durch Semikola (;), durch Kommata (,) oder durch Leerzeichen voneinander getrennt werden. Kommata und Leerzeichen führen zu einem Sprung an die nächstfolgende Tabulatorposition. Diese beginnen normalerweise bei ganzzahligen Vielfachen von 13, sofern nicht durch den Befehl ZONE eine abweichende Regelung getroffen wird.

Achtung! Alle positiven numerischen Werte werden mit einem vorangestellten und einem nachfolgenden Leerzeichen ausgegeben. Das vorangestellte Leerzeichen wird bei negativen Werten zur Ausgabe des Vorzeichens verwendet.

Bei Verwendung der USING-Option sind folgende Formatangaben erlaubt:

| Formatzeichen | Erläuterung |
|---|---|
| # | Platzhalter für Ziffern. |
| . | Position des Dezimalpunktes. |
| , | Die Lesbarkeit großer Zahlen wird dadurch erhöht, daß nach jeder dritten Ziffer (bezogen auf den Dezimalpunkt) automatisch ein Komma gesetzt wird. |
| – | führt zur Ausgabe eines negativen Vorzeichens am Ende der Zahl. |
| ** | sorgt dafür, daß führende Leerstellen bei der Zahlenausgabe durch Sternchen aufgefüllt werden. |
| $ | setzt ein Dollarzeichen an die erste durch die Formatangabe definierte Position. |
| $$ | setzt das Dollarzeichen unmittelbar vor die erste auszugebende Ziffer. |
| **$ | Kombination aus ** und $$ |

| Formatzeichen | Erläuterung |
|---|---|
| + | gibt je nach Formatposition ein positives Vorzeichen zu Beginn oder am Ende der ausgegebenen Zahl aus. |
| – | gibt je nach Formatposition ein negatives Vorzeichen zu Beginn (zusätzlich) oder am Ende der auszugebenden Zahl aus. |
| ↑ ↑ ↑ ↑ | Ausgabe im Exponentialformat. |
| ! | gibt bei Zeichenketten nur das erste Zeichen aus. |
| \<Leerzeichen>\ | es werden so viele Zeichen einer Zeichenkette ausgegeben, wie Leerzeichen vereinbart worden sind. |
| & | Ausgabe der gesamten Zeichenkette. |

Beispiele:

```
A=123:B=234.456
PRINT A;B
123 234.456
PRINT A,B
123          234
PRINT USING"#####";B
234
PRINT USING"###.#";B
234.5
PRINT USING"#.##↑↑↑↑";B
0.23E+03
```

Ein Semikolon am Ende des Printbefehls unterdrückt nach der Ausgabe den Sprung des Cursors in die nächstfolgende Zeile.

Querverweis: TAB, SPC

---

**RAD**

Syntax: RAD
BASIC-Code: 193 bzw. &C1

Erläuterung:

Mittels RAD werden alle Zahlenangaben in trigonometrischen Funktionen im Bogenmaß aufgefaßt. Der Zusammenhang zwischen Gradmaß und Bogenmaß lautet:

Bogenmaß = Gradmaß *PI/180

Querverweis: DEG, SIN, COS, TAN, ATN

---

**RANDOMIZE**

Syntax: RANDOMIZE [<numerischer Ausdruck>]
BASIC-Code: 194 bzw. &C2

Erläuterung:

Der Interpreter enthält einen Zufallszahlengenerator, aus dem über die Funktion RND Zufallszahlen im Bereich zwischen 0 und 1 abgerufen werden können. RANDOMIZE setzt den internen Startpunkt für den RND-Befehl fest.

Beispiel 1:

```
10 FOR I%=1 TO 5
20 PRINT RND(1)
30 NEXT I%
40 END

1.Lauf:
RUN
 0.653729687
 0.601031218
 0.934601486
 0.545995176
 6.34897E-02

2.Lauf:
RUN
 0.989510193
 0.140522114
 3.67999E-02
 0.182864718
 0.352758596
```

Beispiel 2:

```
5 RANDOMIZE 12
10 FOR I%=1 TO 5
20 PRINT RND(1)
30 NEXT I%
40 END
1.Lauf:
RUN
 0.271940658
 0.528612386
 0.021330127
 0.175138616
 0.657773343

2.Lauf:
RUN
 0.271940658
 0.528612386
 0.021330127
 0.175138616
 0.657773343
```

Querverweis: RND

---

**READ**

Syntax: READ <Liste von Variablen>
BASIC-Code: 195 bzw. &C3

Erläuterung:

Die Anweisung READ dient zum Einlesen von Daten aus einem DATA-Statement. Es werden immer so viele Werte eingelesen, wie Variablen in der Liste vereinbart wurden. Bereits eingelesene DATA-Elemente können nur dann erneut verwendet werden, wenn sie durch RESTORE wieder verfügbar gemacht werden.

Programmbeispiel:

```
10 FOR I=1 TO 2
20 READ A$
30 PRINT A$;
40 NEXT I
50 END
60 DATA "GUTEN ",TAG

RUN
GUTEN TAG
```

Querverweis: DATA, RESTORE

---

**RELEASE**

Syntax: RELEASE <Kanal>
BASIC-Code: 196 bzw. &C4

Erläuterung:

Jeder der drei Tonkanäle A, B und C besitzt einen Pufferspeicher, in den SOUND-Befehle „auf Vorrat" abgelegt werden. Wird im SOUND-Befehl zur Kanalnummer noch die Zahl 64 addiert, geht das so programmierte Ereignis so lange in einen Halte- bzw. Wartezustand über, bis durch den RELEASE-Befehl die Sperre aufgehoben wird. Für die Kanalbezeichnung im RELEASE-Befehl gilt A = 1, B = 2, C = 4.

Im nachfolgenden Beispiel wird zunächst der Puffer mit zwei Tonausgabebefehlen geladen. Da der erste mit einem Haltebit markiert ist, geht er in einen Wartezustand über und verhindert gleichzeitig die Ausführung des nachfolgenden Soundelements. Nach Ablauf der Schleife durch den RELEASE-Befehl wird der Puffer normal abgearbeitet.

```
10 SOUND 65,145,10
20 SOUND 1,290,20
30 FOR I%=1 TO 1000:NEXT I%
40 PRINT "Jetzt wird die Ausgabe freigegeben."
50 RELEASE 1
```

Querverweis: ON SQ, SOUND, SQ

**REM**

Syntax: REM <Text>
BASIC-Code: 197 bzw. &C5

Erläuterung:

REM leitet einen Kommentartext ein. Dieser Text wird vom Interpreter bei der Programmausführung übersprungen. Anstelle der Anweisung REM ist auch ein Apostrophzeichen (') zulässig. Es macht in Multistatementzeilen das Trennungszeichen (:) vor dem Kommentartext überflüssig.

Beachten Sie bitte, daß Kommentarstatements in DATA-Zeilen auch am Zeilenende nicht vereinbart werden dürfen.

Beispiele für zulässige und nichtzulässige Kommentare:

Zulässig:

```
10 L=1'***Anfangswert
```

oder

```
10 L=1:REM***Anfangswert
```

Nicht zulässig:

```
10 DATA 10,20,45,67:REM**Maschinencode
```

*Hinweis:* Das Apostrophzeichen besitzt den BASIC-Code 192 bzw. &CO.

---

**RENUM**

Syntax: RENUM [<Anfangszeilennummer>],[<alte Zeilennummer>] [,<Schrittweite>]
BASIC-Code: 198 bzw. &C6

Erläuterung:

Mittels der RENUM-Anweisung können BASIC-Programmtexte mit neuen Zeilennummern versehen werden. RENUM ohne weitere Parameterangaben führt zu einer Organisation beginnend bei 10 in Schritten von 10. Dies entspricht RENUM 10,,10.

## RESTORE

Syntax: RESTORE [<Zeilennummer>]
BASIC-Code 199 bzw. &C7

Erläuterung:

Durch den READ-Befehl aus einer DATA-Zeile bereits eingelesene Datenelemente sind einem erneuten Zugriff des Interpreters entzogen, da der interne Tabellenzeiger immer hinter dem zuletzt ausgelesenen DATA-Element steht. Durch den Befehl RESTORE können die in der angegebenen Programmzeile stehenden Elemente erneut aktiviert werden.

1. Programmbeispiel:

```
10 FOR I%=1 TO 10
20 READ A
30 PRINT A;
40 NEXT I%
50 DATA 1,2,3,4,5

RUN
1  2  3  4  5
DATA exhausted in 20
```

2. Programmbeispiel:

```
10 FOR I%=1 TO 10
20 READ A
30 PRINT A;
40 IF I%=5 GOTO 70
50 NEXT I%
60 END
70 RESTORE:GOTO 50
80 DATA 1,2,3,4,5

RUN
 1  2  3  4  5  1  2  3  4  5
Ready
```

Querverweis: READ, DATA

**RESUME**

Syntax:RESUME [<Zeilennummer>]
oder: RESUME NEXT
BASIC-Code: 200 bzw. &C8

Erläuterung:

RESUME führt zu einer Weiterführung eines durch einen ON ERROR GOTO-Befehl unterbrochenen Programms. Bei Angabe einer Zeilennummer wird der Programmlauf an der vereinbarten Stelle fortgesetzt.

Querverweis: ON ERROR GOTO

---

**RETURN**

Syntax: RETURN
BASIC-Code: 201 bzw. &C9

Erläuterung:

Der Rücksprung aus einem Unterprogramm ins aufrufende Hauptprogramm muß immer mit einem RETURN-Befehl erfolgen. Hierdurch wird gewährleistet, daß der für die Programmablaufkontrolle vom Interpreter verwendete Adressenstapel nicht durcheinandergerät.

Querverweis: AFTER, EVERY, GOSUB, ON...GOSUB, ON SQ, ON BREAK

---

**RIGHT$**

Syntax: (<Zeichenkette>,<numerischer Ausdruck n>)
BASIC-Code: 255 und 121 bzw. &FF und &79

Erläuterung:

Die Funktion RIGHT$ übergibt an das Programm die letzten n Zeichen der im Argument angegebenen Zeichenkette. Diese kann in Form einer Variablen oder als Textkonstante vereinbart werden.

Beispiel:

```
PRINT RIGHT$("SCHNEIDER",3)
      DER
```

Querverweis: MID$,LEFT$

---

**RND**

Syntax: RND [<numerischer Ausdruck>]
BASIC-Code: 255 und 69 bzw &FF und &45

Erläuterung:

RND erzeugt annähernd gleichverteilte Zufallszahlen im Bereich zwischen 0 und 1. Das Argument braucht nicht angegeben zu werden. Besitzt das Argument den Wert 0, wird immer dieselbe Zahl erzeugt. Positive Argumente führen zu nicht reproduzierbaren Zufallsfolgen. Negative Argumente führen zu wertabhängigen, aber reproduzierbaren Zahlenfolgen.

Querverweis: RANDOMIZE

---

**ROUND**

Syntax:ROUND(<numerischerAusdruck>[,<numerischerAusdruck>]
BASIC-Code: 255 und 122 bzw. &FF und 7A

Erläuterung:

Die Funktion ROUND rundet das Ergebnis des im Argument angegebenen ersten numerischen Ausdrucks ab oder auf. Die Stellenzahl für die Ausgabe kann im zweiten numerischen Ausdruck vereinbart werden.

Querverweis: INT, FIX, CINT, ABS

---

**RUN**

Syntax: RUN <Zeichenkette>
oder RUN [<Zeilennummer>]
BASIC-Code: 202 bzw. &CA

Erläuterung:

Der Befehl RUN dient zum Start eines von einem externen Datenträger zu ladenden oder bereits im Speicher liegenden Programms.

In der zuerst angegebenen Form wird das in der Zeichenkette angegebene Programm vom Datenträger (Kassette oder Diskette) geladen und unmittelbar nach Beendigung des Ladevorgangs gestartet. Ein bereits im Speicher befindliches Programm wird vorher gelöscht (siehe auch NEW). Wird eine leere Zeichenkette ("") angegeben, wird bei Kassettenbetrieb die nächsterreichbare Datei geladen. Bei aktivem Diskettenlaufwerk erfolgt in diesem Fall eine Fehlermeldung „Bad Command". Wird zu Beginn der Zeichenkette ein Ausrufungszeichen angegeben (!), werden die üblichen Kontrollmeldungen beim Laden von Kassette unterdrückt. RUN ohne Angabe von Parametern startet ein im Speicher abgelegtes Programm beginnend bei der ersten Zeilennummer. Soll der Programmlauf ab einer vorgegebenen Zeilennummer beginnen, muß diese explizit angegeben werden.

Beispiele:

| | |
|---|---|
| RUN "Test" | lädt das Programm mit dem Namen Test von Kassette oder Diskette. |
| RUN "" | lädt das nächsterreichbare Programm von Kassette. |
| RUN "!Test" | lädt das Programm Test ohne Kontrollmeldungen von Kassette. |
| RUN | startet ein im Speicher abgelegtes Programm. |
| RUN 200 | startet ein im Programm im Speicher ab der Zeile 200. |

Querverweis: LOAD

---

**SAVE**

Syntax: SAVE <Dateiname>[,<Dateityp>][,<Adreßbereich>]
BASIC-Code: 203 bzw. &CB

Erläuterung:

Mit dem Befehl SAVE wird ein im Speicher abgelegtes Programm oder der Inhalt eines näher vereinbarten Speicherbereichs unter dem angege-

benen Dateinamen auf einen externen Datenträger gesichert. Für den Dateityp sind folgende Angaben möglich:

A speichert das Programm im ASCII-Format.
P schützt das Programm vor unbefugtem Zugriff (Listschutz).
B speichert einen im Adreßbereich angegebenen Speicherbereich in binärer Form.

Beispiel:

```
SAVE "Bild",B,&C000,16384
```

Querverweis: LOAD, RUN

---

**SGN**

Syntax: SGN(<numerischer Ausdruck>)
BASIC-Code: 255 und 20 bzw. &FF und &14

Erläuterung:

Die Funktion SGN ermittelt das Vorzeichen des im Argument angegebenen numerischen Ausdrucks.

Querverweis: ABS

---

**SIN**

Syntax: SIN(<numerischer Ausdruck>)
BASIC-Code: 255 und 21 bzw. &FF und &15

Erläuterung:

Die durch die Funktion SIN abrufbare Sinusfunktion gehört wie Tangens und Cosinus zu den trigonometrischen Funktionen. Sie ist periodisch in $2\pi$ bzw. in 360 Grad (Abb. 3.23).

Querverweis: COS, TAN, ATN, DEG, RAD

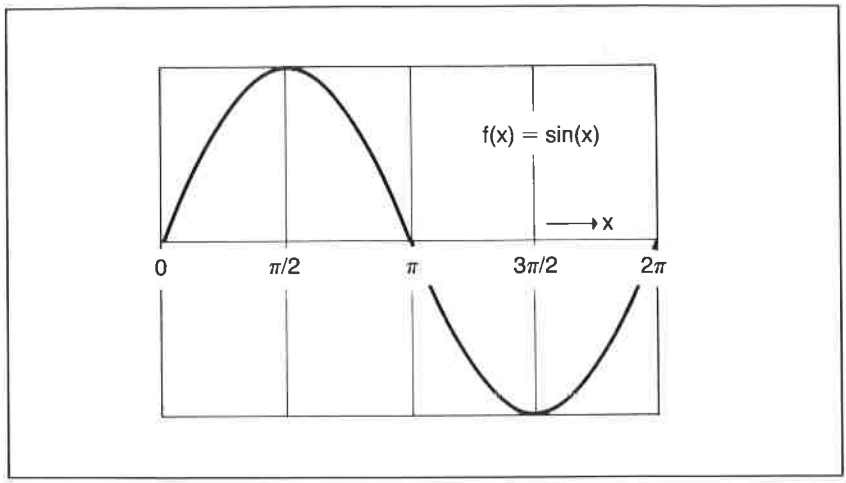



*Abb. 3.23: Die Funktion SIN(X)*

## SOUND

Syntax: SOUND <Kanalkombination>, <Periodendauer> [,<Zeitdauer> [,<Lautstärke> [,<Nummer der Lautstärkeeinhüllenden>[,<Nummer der Frequenzmodulierenden> [,<Rauschperiode>]]]]]
BASIC-Code: 204 bzw. &CC

Kurzerläuterung:

Der SOUND-Befehl dient zur Programmierung des Soundgenerators AY-3-3912. Er wird zusammen mit anderen Soundbefehlen in Kapitel 5 ausführlich behandelt.

Bedeutung der Parameter:

*Kanalkombination*

Der Parameter „Kanalkombination" bezieht sich auf die Wirkung des Registers R7 des Soundchips. Die einzelnen Bits haben die nachfolgend angegebene Bedeutung:

| | | |
|---|---|---|
| B0: | Kanal A | entspricht dezimal 1 |
| B1: | Kanal B | entspricht dezimal 2 |

| | | |
|---|---|---|
| B2: | Kanal C | entspricht dezimal 4 |
| B3: | Synchronisation mit Kanal A | entspricht dezimal 8 |
| B4: | Synchronisation mit Kanal B | entspricht dezimal 16 |
| B5: | Synchronisation mit Kanal C | entspricht dezimal 32 |
| B6: | Wert halten (HOLD) | entspricht dezimal 64 |
| B7: | Puffer leeren und Ausgabe abbrechen | entspricht dezimal 128 |

*Periodendauer*

Der Wert für die Periodendauer d wird für eine vorgegebene Frequenz f eines Tons nach der Beziehung

$$d = 1.000.000/(16*f)$$

berechnet.

*Zeitdauer*

Die Zeitdauer wird in Vielfachen von 1/100 Sekunden angegeben.

*Lautstärke*

Als Lautstärkewerte sind Zahlen zwischen 0 und 7 ohne Vereinbarung einer Lautstärkeeinhüllenden bzw. zwischen 0 und 15 mit deren Vereinbarung zulässig.

*Nummer der Lautstärkeeinhüllenden*

Wird für die Festlegung des zeitabhängigen Lautstärkeverlaufs die ENV-Anweisung benutzt, muß deren Kennummer im SOUND-Befehl an dieser Stelle angegeben werden. Zulässig sind Werte zwischen 1 und 15.

*Nummer der Frequenzmodulierenden*

Eine zeitliche Veränderung der Tonhöhe (Frequenzmodulation) ist über den ENT-Befehl möglich. Dessen Kennummer ist an dieser Stelle anzugeben. Erlaubt sind Nummern zwischen 1 und 15.

*Rauschperiode*

Unter dem Parameter „Rauschperiode" ist die Wiederholperiode des für die Rauscherzeugung verwendeten rückgekoppelten Schieberegisters einstellbar. Erlaubt sind Werte zwischen 1 und 31. Erreicht wird hierdurch eine Änderung des Klanges.

Querverweis: ENT, ENV, ON SQ, SQ, RELEASE, Kapitel 5

**SPACE$**

Syntax: SPACE$(<numerischer Ausdruck>)
BASIC-Code: 255 und 22 bzw &FF und &16

Erläuterung:

Mit SPACE$ wird eine aus Leerzeichen bestehende Zeichenkette erzeugt. Ihre Länge muß im Argument angegeben werden.

Querverweis: PRINT, SPC, TAB

---

**SPEED INK**

Syntax: SPEED INK <Dauer1>,<Dauer2>
BASIC-Code: 205 und 162 bzw. &CD und &3E

Erläuterung:

SPEED INK legt die Zeitdauer für die Farbwechsel fest, die durch Doppelparameter in BORDER-Befehlen vereinbart wurden. Angegeben werden die Parameter Dauer1 und Dauer2 in Vielfachen von 0.02 Sekunden.

Querverweis: INK, BORDER

---

**SPEED KEY**

Syntax: SPEED KEY <Ansprechverzögerung>,<Wiederholperiode>
BASIC-Code: 205 und 164 bzw. &CD und &A4

Erläuterung:

Die meisten Tasten der Tastatur besitzen eine automatische Wiederholfunktion, die 200 Millisekunden nach der Tastenbetätigung (Ansprechverzögerung) einsetzt und die Zeichen mit einer Wiederholrate von 5 Hz ausgibt. Über den SPEED KEY-Befehl sind diese Werte in Vielfachen von 0.02 Sekunden veränderbar. Achtung: Zu kurze Zeiten kommen in Konflikt mit den Routinen zur Tastenentprellung! Über den KEY DEF-Befehl kann die Wiederholfunktion einzelner Tasten abgeschaltet werden. Die Standardeinstellung lautet SPEED KEY 10,10.

Querverweis: KEY DEF

## SPEED WRITE

Syntax: SPEED WRITE <numerischer Ausdruck>
BASIC-Code: 205 und 217 bzw. &CD und &D9

Erläuterung:

Der eingebaute Kassettenrecorder kann Programme und Daten mit Geschwindigkeiten von 2000 Baud (bit/s) oder 1000 Baud laden und speichern. Beim Ladevorgang wird vom Betriebssystem automatisch die korrekte Geschwindigkeit eingestellt. Der Parameter im SPEED WRITE-Befehl ist 1 für 2000 Baud und 0 für 1000 Baud.

Der Interpreter macht von einer Routine im Betriebssystem Gebrauch, mit deren Hilfe auch von den zuvor angegebenen Standardwerten abweichende Geschwindigkeiten erzeugt werden können. Das nachfolgende Programm stellt die Baudrate auf einen vom Anwender anzugebenden Wert ein:

Programmbeispiel:

```
5 REM **** Neue Baud-Rate setzen ****
10 CLS:GOSUB 80
20 INPUT "Gewuenschte Baud-Rate (750-2500): ",B
30 BA=INT(333333/B)
40 IF BA>256 THEN YY=BA-256:XX=1 ELSE YY=BA:XX=0
50 IF BA<130 OR BA>480 THEN 20
60 POKE &7002,XX:POKE &7001,YY
70 CALL &7000:END
80 FOR DAT=0 TO 8
90 READ DA:POKE &7000+DAT,DA
100 NEXT DAT
110 RETURN
120 DATA &21,&00,&00
130 DATA &3E,&19
140 DATA &CD,&68,&BC
150 DATA &C9
```

Querverweis: SAVE

## SQ

Syntax: SQ (<Kanal>)
BASIC-Code: 255 und 23 bzw. &FF und &17

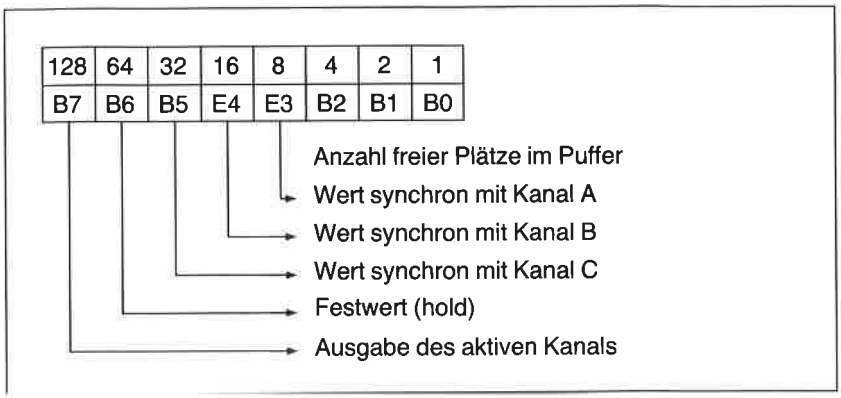



*Abb. 3.24: Bedeutung der Bits im Statusregister des Soundpuffers*

Erläuterung:

Über die SQ-Funktion kann eine Information über den aktuellen Zustand (Statusinformation) eines Soundpuffers abgerufen werden. Die einzelnen Bits des entsprechenden Statusregisters haben die in der Abb. 3.24 gezeigte Bedeutung.

Beispiel:

```
PRINT SQ(1)
      130
```

bedeutet, daß zur Zeit ein Ton auf Kanal 1 ausgegeben wird und noch zwei Plätze im Puffer frei sind.

Querverweis: SOUND, ON SQ

---

**SQR**

Syntax: SQR(<positiver numerischer Ausdruck>)
BASIC-Code: 255 und 24 bzw. &FF und &18

Erläuterung:

Die Funktion SQR ermittelt die Quadratwurzel des im Argument vereinbarten numerischen Ausdrucks.

Beispiel:

```
PRINR SQR(25)
      5
Ready
```

---

**STOP**

Syntax: STOP
BASIC-Code: 205 bzw. &CE

Erläuterung:

Die Anweisung STOP in einer Programmzeile unterbricht den Lauf des Programms mit einer Kontrollmeldung auf dem Bildschirm. Das Programm kann unmittelbar danach durch die CONT-Anweisung fortgesetzt werden.

Querverweis: CONT, END

---

**STR$**

Syntax: STR$(<numerischer Ausdruck>)
BASIC-Code: 255 und 25 bzw. &FF und &19

Erläuterung:

Die Funktion STR$ überführt den im Argument angegebenen numerischen Ausdruck in eine Zeichenkette.

Beispiel:

```
A$=STR$(&9A)
PRINT A$
      154
```

Querverweis: VAL, PRINT, HEX$, BIN$

---

**STRING$**

Syntax: STRING$ (<numerischer Ausdruck>,<Zeichen>)
BASIC-Code: 255 und 123 bzw. &FF und &7B

Erläuterung:

Die Funktion STRING$ erzeugt von dem angegebenen Zeichen eine Zeichenkette eines Umfangs, der als numerischer Ausdruck definiert werden muß.

Beispiel:

```
PRINT STRING$(5,CHR$(65))
AAAAA
```

Querverweis: SPACE$

---

**SYMBOL**

Syntax: SYMBOL <Zeichencode>,<Matrixdaten>
BASIC-Code: 207 bzw. &CF

Erläuterung:

SYMBOL erlaubt die individuelle Definition von Zeichen. Interessant ist dies im Zusammenhang mit dem Entwurf von länderspezifischen Zeichen

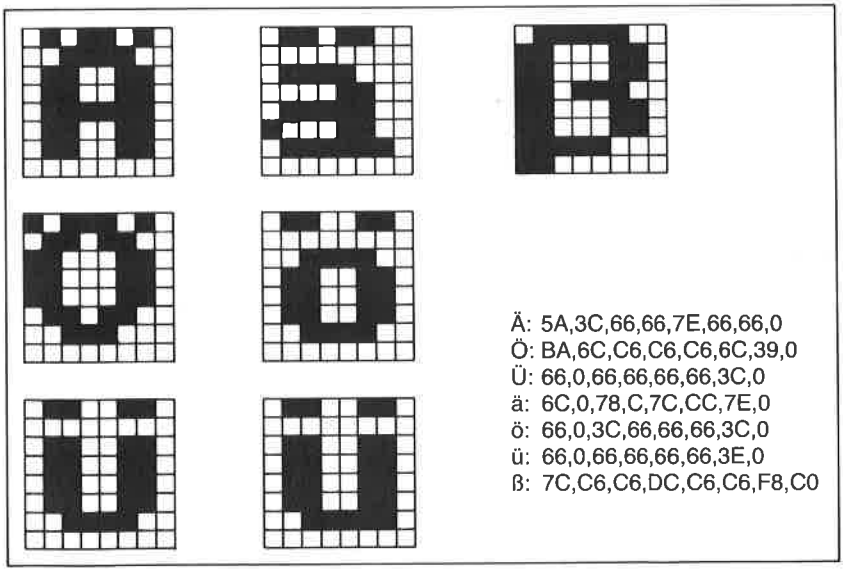



*Abb. 3.25: Konstruktion und Verschlüsselung der Umlaute und des Zeichens ß*

und Symbolen. Für den deutschen Sprachraum sind dies beispielsweise die Umlaute ä, ö, ü, Ä, Ö, Ü und ß. Auf diesen Problemkreis geht das Kapitel 2 ausführlich ein. Der SYMBOL-Befehl muß zum einen den ASCII-Code des zu ändernden Zeichens und zum anderen die Bitelemente der 8*8-Matrix in dezimaler oder auch hexadezimaler Notation enthalten. Für die Konstruktion der deutschen Umlaute sowie des ß sehen Sie Beispiele in der Abb. 3.25.

Alle in einer Matrixzeile gesetzten Bits führen zu einem Bildpunkt in der vereinbarten Vordergrundfarbe (PEN), alle nichtgesetzten Bits zu einem Punkt in der Hintergrundfarbe (PAPER).

Querverweis: SYMBOL AFTER

---

**SYMBOL AFTER**

Syntax: SYMBOL AFTER <Zeichencode>
BASIC-Code: 207 und 128 bzw. &CF und &80

Erläuterung:

SYMBOL AFTER legt diejenige Grenze fest, ab der Zeichen vom Anwender neu definiert werden können.

Beispiel:

```
SYMBOL AFTER 32
```

gibt alle ASCII-Zeichen, beginnend vom ASCII-Code 32, für eine Änderung frei. Das folgende Programm definiert als Beispiel die Tasten mit den Zeichen ([), (]),(\), (I) und ( ↑ ) um und belegt diese mit den Umlauten sowie dem ß. Hierbei wurde die für Drucker geltende Codebelegung berücksichtigt.

Querverweis: SYMBOL

---

**TAG**

Syntax: TAG[#<Ausgabefenster>]
BASIC-Code: 208 bzw. &D0

Erläuterung:

Die Textausgabe innerhalb eines Fensters erfolgt normalerweise an den üblichen Zeichenpositionen, die durch die eingestellte Betriebsart bestimmt werden. Über den Befehl TAG (Text And Graphic) ist eine Positionierung im Bildpunktraster möglich. Besonders wertvoll ist dies bei der Beschriftung von Grafiken u. ä. Wird das Ausgabefenster nicht angegeben, nimmt der Interpreter #0 an. Durch den TAG-Befehl wird der linke untere Matrixpunkt des ersten auszugebenden Zeichenfeldes auf den Punkt gesetzt, der durch die Lage des Grafikcursors bestimmt ist. Achten Sie bitte darauf, daß die Schriftfarbe durch die des Bildpunktes bestimmt wird! Alle normalerweise nicht sichtbaren Kontrollzeichen werden dann angezeigt, wenn der zur Ausgabe dienende PRINT-Befehl nicht mit einem Semikolon abgeschlossen wird. Versuchen Sie es einmal mit dem nachfolgend angegebenen kleinen Programm:

Programmbeispiel:

```
5 REM **** TAG-Demo ****
10 MODE 0
20 CLS
30 DEG
40 FOR F%=1 TO 15
50 FOR I%= 0 TO 360 STEP 10
60 x=250+190*COS(I%)
70 y=210+190*SIN(I%)
80 PLOT X,Y,F%
90 TAG
100 PRINT "SYBEX";
110 NEXT i%
120 X=X-4:MOVE X,Y
130 TAG
140 PRINT "SYBEX";
150 IF X>250 GOTO 120
160 NEXT F%
170 GOTO 10
```

Querverweis: TAGOFF

---

**TAGOFF**

Syntax:TAGOFF [#<Ausgabefenster>]
BASIC-Code: 209 bzw. &D1

Erläuterung:

TAGOFF schaltet die Wirkung des TAG-Befehls ab. Text wird an der ursprünglichen Textcursorposition (vor Ausführung des TAG-Befehls) ausgegeben.

Querverweis: TAG

---

**TAN**

Syntax: TAN(<numerischer Ausdruck>)
BASIC-Code: 255 und 26 bzw. &FF und &1A

Erläuterung:

Durch TAN wird die Tangensfunktion abgerufen. Sie gehört wie die Funktionen Sinus und Kosinus zu den trigonometrischen Funktionen. Das Argument ist entweder eine numerische Variable, eine numerische Konstante oder ein allgemeiner numerischer Ausdruck. Die Tangensfunktion ist periodisch in π. Das Argument ist ein Winkel, der ohne gesonderte Vereinbarung im Bogenmaß (RAD) anzugeben ist. Über das BASIC-Kommando DEG können Argumentwerte auch im Gradmaß vereinbart werden.

Querverweis: COS, SIN, ATN, RAD, DEG

---

**TEST**

Syntax: TEST (<x-Koordinate>,<y-Koordinate>)
BASIC-Code: 255 und 124 bzw. &FF und &7C

Erläuterung:

TEST ermittelt für die im Argument angegebene Bildschirmposition den Wert der Vordergrundfarbe.

Querverweis: TESTR

---

**TESTR**

Syntax: TESTR (<x-Koordinatenoffset>,<y-Koordinatenoffset>)
BASIC-Code: 255 und 125 bzw. &FF und &7D

Erläuterung:

TESTR ermittelt für die im Argument angegebene Bildschirmposition relative zur aktuellen Position des Grafikcursors den Wert der Vordergrundfarbe.

Querverweis: TEST

---

**TIME**

Syntax: TIME
BASIC-Code: 255 und 70 bzw. &FF und &46

Erläuterung:

Über die BASIC-Funktion TIME ist die nach dem Einschalten des Systems vergangene Zeit zu ermitteln. Abgezogen werden müssen jene Zeitspannen, die für Schreib- und Leseoperationen mit externen Massenspeichern benötigt wurden. Der an das Programm übergebene Wert ist ein ganzzahliges Vielfaches von 1/300 Sekunde.

---

**TRON**
**TROFF**

Syntax: TRON bzw. TROFF
BASIC-Code: 211 bzw. &D3 für TRON und 210 bzw. &D2 für TROFF

Erläuterung:

Mittels TRON bzw. TROFF wird eine sogenannte Programmablaufverfolgung ein- bzw. abgeschaltet. Sie besteht darin, daß jede vom Interpreter ausgeführte Programmzeile durch Angabe der Zeilennummer auf dem Bildschirm signalisiert wird. Brauchbar ist diese Art der Programmablaufkontrolle nur, wenn keine Konflikte mit anderen Ausgaben auf dem Bildschirm auftreten.

---

**UNT**

Syntax: UNT(<numerischer Ausdruck>)
BASIC-Code: 255 und 27 bzw. &FF und &1B

Erläuterung:

Die Funktion UNT wandelt eine ganze Zahl im Bereich zwischen 0 und 65535 in die entsprechende Zweierkomplementzahl um. Entsprechend den Regeln der Zweierkomplementbildung (siehe auch Kapitel 6) bleiben alle Zahlen, die kleiner als 32768 sind, unverändert. Die übrigen werden nach dem Gesetz

Zweierkomplement = (Zahlenwert>32767) − 65536

umgewandelt.

Beispiel:

```
PRINT UNT(65535)
−1
```

---

**UPPER$**

Syntax: UPPER$(<Zeichenkette>)
BASIC-Code: 255 und 28 bzw. &FF und &1C

Erläuterung:

Die Zeichenkettenfunktion UPPER$ überführt die kleingeschriebenen Zeichen des im Argument angegebenen Textes in die Großschreibung.

Beispiel:

```
PRINT UPPER$("Schneider CPC 464")
SCHNEIDER CPC 464
```

Querverweis: LOWER$

---

**VAL**

Syntax: VAL (<Zeichenkette>)
BASIC-Code: 255 und 29 bzw. &FF und &1D

Erläuterung:

Die Funktion VAL überführt das erste Zeichen der im Argument stehenden Zeichenkette in einen numerischen Wert, falls es sich um eines der Zeichen 0 bis 9 handelt.

Querverweis: STR$

**VPOS**

Syntax: VPOS(#<Ausgabefenster>)
BASIC-Code: 255 und 127 bzw. &FF und &7F

Erläuterung:

VPOS ermittelt die vertikale Position des Textcursors in dem angegebenen Fenster.

Querverweis: POS

---

**WAIT**

Syntax: WAIT <Datenregister>,<Binärmaske>[,<Referenzwert>]
BASIC-Code: 212 bzw. &D4

Erläuterung:

Der WAIT-Befehl dient im Zusammenhang mit der Programmierung (in der Regel Maschinenprogrammierung) von Ein-/Ausgabebausteinen zur Detektion von Bitmustern. Unter dem Parameter <Datenregister> wird die Adresse eines 8-bit-Ein-/Ausgaberegisters angegeben. Zwischen dem Registerwert und der Binärmaske wird eine Exklusiv-ODER-Verknüpfung vorgenommen und gegebenenfalls noch eine UND-Verknüpfung mit dem Referenzwert.

Solange seitens des Herstellers keine zuverlässigen Angaben über die Adressierung der Ein-/Ausgabeports verfügbar sind, ist dieser Befehl für den BASIC-Programmierer wertlos.

Querverweis: INP, OUT

---

**WHILE .... WEND**

Syntax: WHILE .....WEND
BASIC-Code: 213 bzw. &D5 (WEND) und 214 bzw. &D6 für WHILE

Erläuterung:

Die WHILE...WEND-Kombination dient zur wiederholten Ausführung eines durch die beiden Befehlselemente eingeschlossenen Programmteils. Die Wiederholung ist abhängig von einer Bedingung, die Bestandteil des WHILE-Teils ist.

Ein gutes Beispiel ist im Handbuch zum Schneider CPC enthalten.

## WIDTH

Syntax: WIDTH <numerischer Ausdruck>
BASIC-Code: 215 bzw. &D7

Erläuterung:

Der WIDTH-Befehl dient zur Festlegung der maximalen Anzahl von Zeichen für die Ausgabe auf einem angeschlossenen Drucker. WIDTH 60 beschränkt somit die Druckerausgabe auf 60 Zeichen pro Zeile.

Querverweis: PRINT, POS

---

## WINDOW

Syntax: WINDOW[#<Ausgabefenster>,]<linker Rand>,<rechter Rand>,<oberer Rand>,<unterer Rand>
BASIC-Code: 216 bzw. &D8

Erläuterung:

BASIC erlaubt die Definition von bis zu 8 unterschiedlichen Textausgabefenstern, die mit #0 bis #7 bezeichnet werden können. Für jedes individuelle Fenster sind die Abmessungen durch die nachfolgenden Parameter festlegbar (Abb. 3.26). Eine ausführliche Erläuterung finden Sie in Kapitel 2 dieses Buchs.

Programmbeispiel:

```
5 REM **** Text-Fenster ****
10 MODE 1
20 INK 0,0:INK 1,24:INK 2,6:INK 3,9
30 WINDOW #0,1,10,1,12:PEN #0,1:PAPER #0,0:CLS #0:GOSUB
 150
40 WINDOW #1,11,20,1,12:PEN #1,3:PAPER #1,2:CLS #1:GOSU
B 150
50 WINDOW #2,21,30,1,12:PEN #2,0:PAPER #2,3:CLS #2:GOSU
B 150
60 WINDOW #3,31,40,1,12:PEN #3,3:PAPER #3,0:CLS #3:GOSU
B 150
70 WINDOW #4,1,10,13,25:PEN #4,2:PAPER #4,3:CLS #4:GOSU
B 150
80 WINDOW #5,11,20,13,25:PEN #5,0:PAPER #5,1:CLS #5:GOS
UB 150
90 WINDOW #6,21,30,13,25:PEN #6,2:PAPER #6,1:CLS #6:GOS
UB 150
100 WINDOW #7,31,40,13,25:PEN #7,0:PAPER #7,2:CLS #7:GO
SUB 150
110 PRINT:PRINT:END
150 PRINT #I:PRINT #I,"Hallo !!!";:I=I+1:RETURN
```

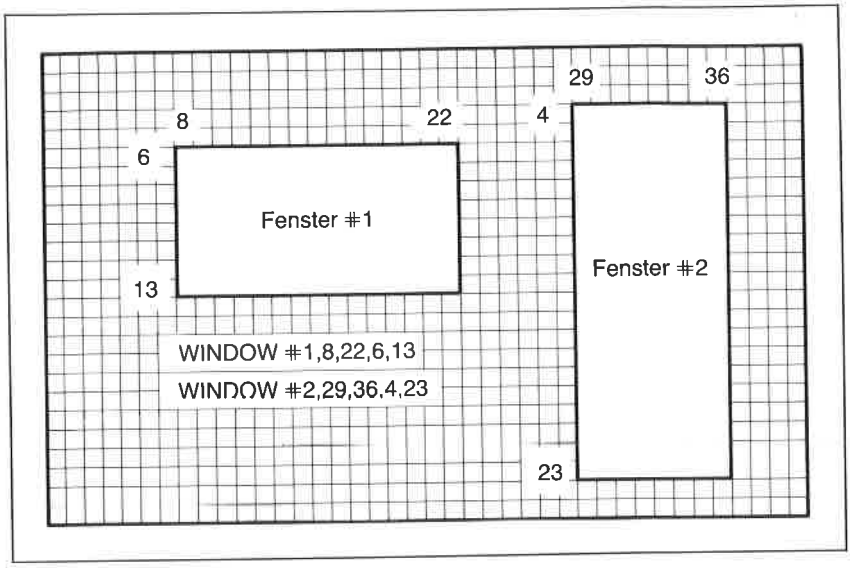



*Abb. 3.26: Beispiele für die Definition von Textfenstern*

Querverweis: ORIGIN, WINDOW SWAP

**WINDOW SWAP**

Syntax: WINDOW SWAP <Fensternummer1>,<Fensternummer2>
BASIC-Code: 216 und 231 bzw. &D8 und &E7

Erläuterung:

Der WINDOW SWAP-Befehl tauscht die Kennummern der in WINDOW-Befehlen definierten Fenster aus. Dieser Befehl ist dann besonders wertvoll, wenn er im Zusammenhang mit dem Fenster #0 angewandt wird. In diesem erfolgen alle Ausgaben, wenn nicht explizit ein anderes Datenfenster vereinbart wurde. Siehe auch Kapitel 2.

Querverweis: WINDOW

**WRITE**

Syntax: WRITE [#<Ausgabekanal>][,<Liste>]
BASIC-Code: 217 bzw. &D9

Erläuterung:

Der WRITE-Befehl unterscheidet sich vom PRINT-Befehl dadurch, daß er bei der Ausgabe Zeichenketten in Anführungszeichen setzt und Zahlen durch Kommata trennt.

Als Ausgabekanäle gelten:

#0 bis #7 Bildschirmfenster 0 bis 7
#8 Drucker
#9 Datenrecorder oder Diskettenlaufwerk

Querverweis: PRINT

---

**XPOS**

Syntax: XPOS
BASIC-Code: 255 und 71 bzw. &FF und &47

Erläuterung:

XPOS liefert die horizontale Position des Grafikcursors.

Beispiel:

```
MOVE 200,300
PRINT XPOS
200
```

Querverweis: YPOS

## YPOS

Syntax: YPOS
BASIC-Code: 255 und 72 bzw. &FF und &48

Erläuterung:

YPOS liefert die vertikale Position des Grafikcursors.

Beispiel:

```
MOVE 200,300
PRINT YPOS
300
```

Querverweis: XPOS

---

## ZONE

Syntax: ZONE <numerischer Ausdruck>
BASIC-Code: 218 bzw. &DA

Erläuterung:

Der Tabulatorsprung bei der Ausgabe von Werten im PRINT-Befehl, die durch Kommata voneinander getrennt sind, beträgt standardmäßig 13 Zeichenfelder. Mittels ZONE können Tabulatorzonen zwischen 1 und 255 gesetzt werden.

Querverweis: PRINT, WIDTH, WRITE

## BASIC intern

In diesem Abschnitt werden Sie abschließend zum Thema BASIC noch einige interessante Informationen über die Arbeitsweise des Interpreters und die Speicherung von BASIC-Programmen im RAM Ihres CPC finden. Sofern Ihnen an dieser Stelle Begriffe wie Speicherorganisation, binäre Daten oder hexadezimale Verschlüsselung von BASIC-Codes noch nicht viel sagen, sollten Sie sich zunächst das benötigte Grundwissen in Kapitel 6 aneignen.

### *Der Programmspeicher*

Wie Sie bereits wissen, werden BASIC-Programmzeilen von den unmittelbar eingegebenen Befehlen dadurch unterschieden, daß sie mit einer Zeilennummer beginnen. Deren Reihenfolge können Sie entweder selbst vereinbaren oder aber automatisch durch den AUTO-Befehl festlegen (siehe AUTO). Mit einer Zeilennummer versehene Anweisungen werden nicht unmittelbar nach Beendigung der Eingabe mittels der ENTER-Taste ausgeführt, sondern zunächst im Schreib-/Lesespeicher abrufbar zwischengespeichert. Sofern keine besonderen Maßnahmen getroffen werden, beginnt der für die Speicherung des Programmtextes reservierte Bereich im RAM bei der Adresse 368 dezimal oder &170 in hexadezimaler Schreibweise. Wir wollen uns anhand eines einfachen Beispiels einmal ansehen, in welcher Form der Programmtext im Speicher abgelegt wird. Dazu bedienen wir uns des nachfolgend angegebenen kleinen Programms, das nach dem Start nichts anderes tut, als den Text SCHNEIDER CPC 464 auf dem Bildschirm auszugeben. Dazu sollten Sie zunächst die Kommandos NEW und MODE 2 und dann die Programmzeilen ein-

```
10 PRINT "SCHNEIDER CPC 464"
20 END
30 CLS
40 A1=368:N%=1
50 PRINT"Adresse            Inhalt
60 PRINT STRING$(36,"-")
70 PRINT"       0   1   2   3   4   5   6   7"
80 PRINT STRING$(36,"-")
90 PRINT USING "####";A1;
100 FOR I%=A1 TO A1+7
110 PRINT USING "####";PEEK(I%);
120 NEXT I%
130 PRINT
140 A1=A1+8:N%=N%+1
150 IF N%>8 THEN N%=1:CALL &BB18:PRINT:PRINT:GOTO 50
160 GOTO 90
```

geben. Anschließend schauen Sie sich den Inhalt der Speicherzellen, beginnend bei der Adresse 368 an. Hierzu benutzen wir in Zeile 70 den bereits in der Liste der BASIC-Befehle vorgestellten Befehl PEEK(adr). Starten Sie das Programm durch Eingabe von RUN 30. Auf dem Bildschirm wird abgesehen von den Kopfzeilen folgende Ausgabe erfolgen:

```
Adresse                Inhalt
-------------------------------------------
        0   1   2   3   4   5   6   7
-------------------------------------------
 368   26   0  10   0 191  32  34  83
 376   67  72  78  69  73  68  69  82
 384   32  67  80  67  32  52  54  52
 392   34   0   6   0  20   0 152   0
 400    6   0  30   0 138   0  21   0
 408   40   0  13   6   0  65 177 239
 416   26 112   1   1   2  15   0 206
 424  239  15   0  35   0  50   0 191
```

*Abb. 3.27: Die Speicherbelegung des BASIC-Programms*

Nach Ausgabe dieser 8 Zeilen hält der Rechner an. Betätigen einer beliebigen Taste führt zur Ausgabe der nächsten 8 Zeilen und so fort. Die Ausgabe ist so organisiert, daß als erstes eine Speicheradresse (hier beginnend mit 368 dezimal) und anschließend deren Inhalt sowie die Inhalte der folgenden 7 Speicherzellen in dezimaler Notation innerhalb einer Zeile ausgegeben werden. Verantwortlich dafür, daß die Ausgabe nach jeweils acht Zeilen bis zum Betätigen einer beliebigen Taste unterbrochen wird, ist der Aufruf der Maschinenroutine &BB18 in Zeile 110. Die Formatierung erfolgt mittels der bereits vorgestellten PRINT USING-Befehle.

Bevor wir nun den Versuch unternehmen wollen, die Zahlenkolonnen zu entschlüsseln, sollten Sie sich einmal die nachfolgende Abbildung genauer ansehen. Sie zeigt anhand eines Schemas den grundsätzlichen Aufbau von BASIC-Programmtexten im Speicher des Computers in Form einer Liste. Hierunter versteht man in der Datentechnik eine Folge von Daten, auf deren Elemente gezielt über Verweise (Zeiger) zugegriffen werden kann.

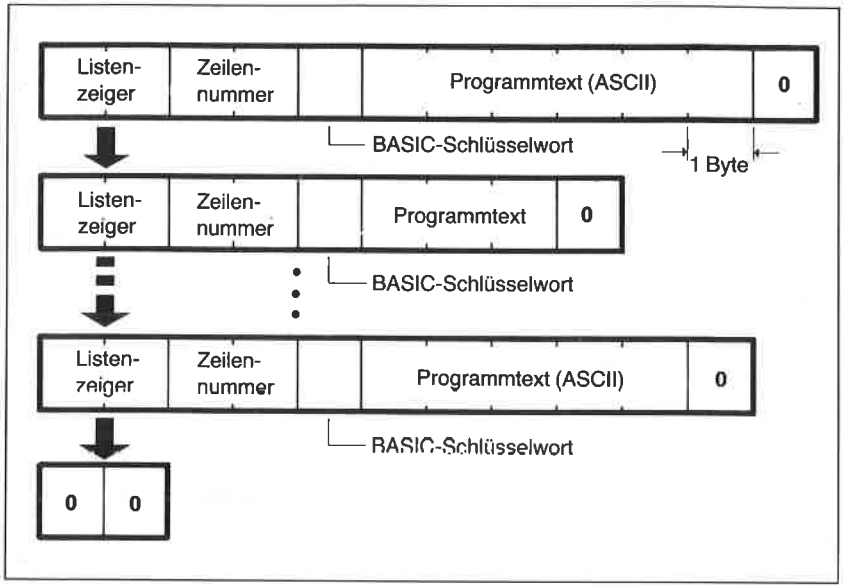



*Abb. 3.28: Listenstruktur von BASIC-Zeilen im Speicher*

Jede BASIC-Programmzeile beginnt im Speicher mit einem derartigen Listenzeiger. Dieser verweist immer auf jene Speicheradresse, bei der die nächstfolgende Programmzeile beginnt. Da der Listenzeiger nicht gesondert in einer Tabelle, sondern als Bestandteil der Liste selbst vereinbart wird, spricht man von einer *verschachtelten* Liste.

Der Wert des Listenzeigers wird in die ersten beiden Bytes eingeschrieben. Er hat somit eine Wortlänge von 16 bit und kann deshalb bis zu 65536 verschiedene Werte im Bereich zwischen 0 und 65535 annehmen. In den beiden unmittelbar folgenden Bytes wird die Information über die vom Anwender vereinbarte Programmzeilennummer abgelegt. Sowohl der Zeiger wie auch die Zeilennummer werden in der Reihenfolge Lowbyte (niederwertiges Byte), Highbyte (höchstwertiges Byte) abgespeichert. Anschließend folgt der ASCII-codierte Programmtext. Dieser weist jedoch einige Besonderheiten auf.

### *Die BASIC-Codes*

BASIC-Schlüsselwörter (BASIC-Tokens genannt) wie PRINT, TAB, FOR usw. werden nämlich nicht Zeichen für Zeichen im ASCII-Code, sondern in Form eines Codewortes von 1 byte (oder in Sonderfällen auch

zwei byte) Wortlänge abgespeichert. Sie haben die entsprechenden BASIC-Codes bereits in dem Abschnitt über die BASIC-Kommandos kennengelernt. Für den Befehl PRINT bedeutet das beispielsweise, daß nur der Codewert 191 dezimal bzw. &BF hexadezimal abzuspeichern ist. Es ist so nur ein Byte erforderlich und nicht die ansonsten benötigten 5 Bytes.

Die Codewörter sind in einer Tabelle abgelegt, in der das Übersetzerprogramm bei der Interpretation von Programmzeilen nachschaut. Auf diese Art und Weise wird beträchtlich Speicherplatz gespart. Für die Vereinbarung von Variablen und Konstanten gelten ähnliche, im einzelnen aber etwas komplizierter anmutende Regeln, auf die etwas später noch kurz eingegangen wird. Ergänzend sei an dieser Stelle noch bemerkt, daß jede BASIC-Programmzeile im Speicher durch ein Nullbyte abgeschlossen wird. Das Programm*ende* wird durch einen Listenzeigerwert von 00 gekennzeichnet, also durch einen Verweis auf eine nicht zulässige Speicheradresse für Programme.

Auf unser konkretes Beispiel angewendet bedeutet das, daß Sie unter den Adreßwerten 368 und 369 den Verweis auf den Beginn der nächstfolgenden Programmzeile finden werden. Nach den zuvor erörterten Regeln ist dies in der ersten Zeile der Matrix die Zahl 26 dezimal. Überzeugen Sie sich davon, daß dieser Sachverhalt richtig ist. Sie müßten daher bei der Adresse 368 + 26 = 394 auf den nächstfolgenden Listenzeiger stoßen. In der Tat steht dort, unmittelbar nach dem Nullbyte als Kennzeichnung des

| Adresse | Inhalt | Bedeutung | Adresse | Inhalt | Bedeutung |
|---|---|---|---|---|---|
| 368 | 26 | } Zeiger auf | 381 | 68 | D |
| 369 | 0 | } nächste Programmzeile | 382 | 69 | E |
| 370 | 10 | } Zeilennummer | 383 | 82 | R |
| 371 | 0 | } Zeilennummer | 384 | 32 | Leerzeichen |
| 372 | 191 | BASIC-Token für PRINT | 385 | 67 | C |
| 373 | 32 | Leerzeichen | 386 | 80 | P |
| 374 | 34 | " | 387 | 67 | C |
| 375 | 83 | S | 388 | 32 | Leerzeichen |
| 376 | 67 | C | 389 | 52 | 4 |
| 377 | 72 | H | 390 | 54 | 6 |
| 378 | 78 | N | 391 | 52 | 4 |
| 379 | 69 | E | 392 | 34 | " |
| 380 | 73 | I | 393 | 0 | Zeilenende |

*Abb. 3.29: Analyse einer BASIC-Zeile im Speicher*

Endes der ersten Programmzeile, die Zahl 6 als Zeiger auf die Programmzeile 30. Das höchstwertige Byte des Listenzeigers ist in beiden Fällen Null, da die Verweise einen unter 256 liegenden Wert haben.

Dem Zeigerwert 26 folgt als nächstes die Zahl 10. Dies ist die Nummer der ersten vereinbarten Programmzeile. Auch hier ist das höchstwertige Byte Null. Es folgt die Zahl 191. Sie entspricht dem BASIC-Code (BASIC-Token) für den Befehl PRINT, wie Sie unter dem entsprechenden Befehl im vorangegangenen Abschnitt dieses Kapitels bzw. im Anhang M nachsehen können. Es folgt anschließend der ASCII-verschlüsselte Programmtext, dessen Ende durch ein Nullbyte (unter der Adresse 393) gekennzeichnet wird. Anschließend folgt der Listenzeiger zu Beginn der Zeile 20, der nach dem zuvor Gesagten auf die Zeile 30 verweisen muß. Zeile 20 enthält nichts anderes als den BASIC-Befehl END, dessen Code dezimal 152 beträgt. Er steht in der Speicherzelle mit der Adresse 398.

Überzeugen Sie sich durch ein einfaches Beispiel von der Richtigkeit des zuvor Gesagten, indem Sie unmittelbar über die Tastatur das Kommando

```
POKE 398,191
```

eingeben. Sie schreiben auf diese Weise in die Speicherzelle mit der Adresse 398 den Wert 191 ein und ersetzen damit den Befehl END durch PRINT. Lassen Sie sich zur Kontrolle durch LIST 10,30 die ersten drei Programmzeilen auf dem Bildschirm ausgeben. Sie sehen nun wie folgt aus:

```
10 PRINT "SCHNEIDER CPC 464"
20 PRINT
30 CLS
```

In Zeile 20 ist END durch PRINT ersetzt worden.

Die zuvor erläuterte einfache Struktur der Listenelemente gilt nur für Statements des eben erwähnten Typs. Spezielle BASIC-Schlüsselwörter, die größtenteils zur Kategorie der Funktionen gehören, benötigen ein Byte mehr. Sie werden durch ein Byte mit dem Wert 255 eingeleitet, dem dann ein weiteres Byte zur Befehlsverschlüsselung folgt. Dies ist beispielsweise bei der mathematischen Funktion ATN der Fall, die über die beiden Codewerte 255 und 2 verschlüsselt wird (siehe auch Anhang M).

Bei der Vereinbarung von Variablen oder Konstanten sieht es noch ein wenig anders aus. Schauen Sie sich in diesem Zusammenhang in Abb. 3.27 einmal diejenige Codefolge an, die zur Programmzeile 40 gehört.

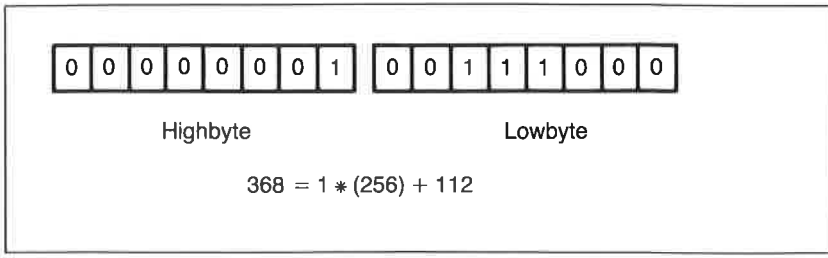



*Abb. 3.30: Zur Erläuterung der Verschlüsselung des Variablenwertes 368*

Nach dem Listenzeiger (14, 0) und der Zeilennummer (40,0) stoßen Sie auf die Zahlen 13 und 6 bei den Adressen 410 und 411. Sie sind von der nachfolgenden Codefolge durch ein Nullbyte getrennt. Der Dezimalcode 13 kennzeichnet den Typ der verwendeten Variablen (numerisch), die Zahl 6 die Länge des Variablennamens + 4. Im vorliegenden Fall haben wir es mit der numerischen Variablen A1 zu tun, die aus zwei alphanumerischen Zeichen besteht. Nach dem Nullbyte folgt der Variablenname, der – abgesehen von dem letzten alphanumerischen Zeichen – einfach ASCII-verschlüsselt ist. Das letzte Zeichen wird dadurch gekennzeichnet, daß das höchstwertige Bit gesetzt wird. Dies bedeutet, daß zum ASCII-Wert die Zahl 128 addiert wird. Die Variable A1 wird daher nicht als Codefolge 65 (ASCII-Code für A) und 49 (ASCII-Code für 1), sondern in der Folge 65 und 177 (49+128) verschlüsselt. Das Gleichheitszeichen (=) entspricht dem BASIC-Code 239. Die nachfolgende Zahl 26 besagt, daß der ganzzahlige Wert der Variablen in den nachfolgenden zwei Bytes enthalten ist. Das ist immer dann der Fall, wenn der Absolutwert den Betrag 255 überschreitet, aber andererseits kleiner als $2^{15}-1$ (= 65535) ist. Für ganzzahlige Werte, die nur ein Byte zur Verschlüsselung benötigen, wird statt der 26 eine 25 angegeben. Die beiden folgenden Bytes enthalten in der Reihenfolge Lowbyte, Highbyte den Zahlenwert der Variablen nach dem in Abb. 3.30 gezeigten Schema.

Zuerst wird somit die Zahl 112 und dann die Zahl 1 im Speicher abgelegt. Zum Schluß folgt dann noch das Nullbyte als Kennzeichen für das Ende der Zeile.

Zum Schluß noch eine kleine Auswahl interessanter Vektoradressen, die sich im Zusammenhang mit der Arbeit in BASIC als nützlich erweisen. So finden Sie beispielsweise bei einem Ladevorgang von einem externen Massenspeicher die Startadresse der geladenen Datei in den beiden Bytes mit den Adressen &AE3F sowie &AE40, den Dateityp unter der Adresse &AE42, die Länge der Datei unter den Adressen &AE43 sowie &AE44.

| BASIC-Vektoren | |
|---|---|
| Adresse | Erläuterung |
| &AC23 | Position des Druckkopfes |
| &AC24 | Druckbreite in Zeichen |
| &AE3F–&AE40 | Startadresse bei LOAD |
| &AE42 | Dateityp |
| &AE43–&AE44 | Dateilänge |
| &AE7B–&AE7C | HIMEM |
| &AE7D–&AE7E | Ende freies RAM |
| &AE7F–&AE80 | Start freies RAM |
| &AE81–&AE82 | Start BASIC-Programm |
| &AE83–&AE84 | Ende BASIC-Programm |
| &AE85–&AE86 | Start der Variablen |
| &AE87–&AE88 | Start Feldvariablen |
| &AE89–&AE8A | Ende Feldvariablen |
| &B08D–&B08E | Start Stringpool |
| &B08F–&B090 | Ende Stringpool |
| Hinweis: bei 16-bit-Zeigern: 1. Adresse enthält das Lowbyte, die 2. Adresse das Highbyte | |

*Abb. 3.31: Auswahl wichtiger Vektoradressen für den BASIC-Interpreter*

Im Anhang M dieses Buchs finden Sie eine vollständige Übersicht über alle BASIC-Kommandos und -Funktionen sowie deren BASIC-Codes. Informationen über die Speicherbelegung von BASIC-Programmen sind dann interessant, wenn Programme mit selbstmodifizierenden Programmzeilen erstellt oder Maßnahmen zum Schutz vor unerwünschtem Zugriff Dritter getroffen werden müssen.

# Kapitel 4
# Grafik und Grafikfunktionen

## Einführung

Der Schneider CPC kann je nach Betriebsart Grafiken mit einer Auflösung von bis zu 640 Punkten horizontal und 200 Punkten vertikal darstellen. Zur Erzeugung farbiger Grafiken stellt der BASIC-Interpreter eine leistungsfähige und leicht zu handhabende Palette von Befehlen zur Verfügung, die Ihnen in diesem Kapitel anhand von Beispielen vorgestellt und erläutert wird.

Der überwiegende Teil des Kapitels beschäftigt sich mit den Grafikbefehlen, die sowohl auf dem CPC 464 als auch auf dem CPC 664 gleichermaßen verwendet werden können. Der letzte Abschnitt dagegen ist jenen Befehlen und Befehlserweiterungen gewidmet, die ausschließlich auf dem CPC 664 zu benutzen sind. Es handelt sich dabei um die neuen Befehle FILL, FRAME, GRAPHICS PAPER, GRAPHICS PEN, MASK sowie die Befehlserweiterungen für die Befehle DRAW, DRAWR, MOVE, MOVER, PEN sowie PLOT und PLOTR.

## Die Grafikmodi des Schneider CPC

Wie Sie bereits in Kapitel 2 gelesen haben, können beim Schneider CPC die drei verschiedenen Betriebsarten MODE 0, MODE 1 und MODE 2 vereinbart werden, die sowohl zu einer unterschiedlichen Text- wie auch Grafikauflösung führen. Zu Ihrer Erinnerung sind in der nachfolgenden Tabelle nochmals die wichtigsten Daten zusammengestellt:

| Betriebsart | Textauflösung | Grafikauflösung | Anzahl Farben |
|---|---|---|---|
| MODE 0 | 20*25 | 160*200 | 16 aus 27 (32) |
| MODE 1 | 40*25 | 320*200 | 4 aus 27 (32) |
| MODE 2 | 80*25 | 640*200 | 2 aus 27 (32) |

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Abb. 4.1: Die Standardfarbzuordnungen in den Betriebsarten MODE 0, MODE 1 und MODE 2*

Für die aktuellen Farben und deren Auswahl gelten die bereits in Kapitel 2 geschilderten Regeln. In der Abb. 4.1 sind nochmals zu Ihrer Information die Standardfarbzuordnungen angegeben.

Wie auch im Textmodus üblich, wird ohne abweichende Vereinbarungen der Hintergrund mit dem Farbwert n = 0 dargestellt. In allen drei Modi ist dies die Farbe Tiefblau. Die gesetzten Bildpunkte werden mit dem Farbwert n=1, im Normalfall also Hellgelb, abgebildet. Andere Farbdefinitionen sind als Parameterbestandteile der Grafikbefehle des BASIC-Interpreters möglich. Hierauf wird noch gesondert eingegangen.

### *Grundlegende Vereinbarungen für grafische Abbildungen*

Unabhängig vom Abbildungsmodus liegt der Koordinatennullpunkt (X=0, Y=0) im Gegensatz zu vielen anderen Computersystemen nach dem Systemstart in der linken *unteren* Ecke des Bildschirmfensters. Ebenfalls unabhängig von der gewählten Betriebsweise und damit von der Auflösung besitzt der rechte obere Eckpunkt die Koordinatenwerte (X=639, Y=399) (Abb. 4.2).

Sofern keine anderslautenden Angaben im ORIGIN-Befehl gemacht werden, füllt das Grafikfenster den innerhalb des Rahmens liegenden Bildschirmbereich aus.

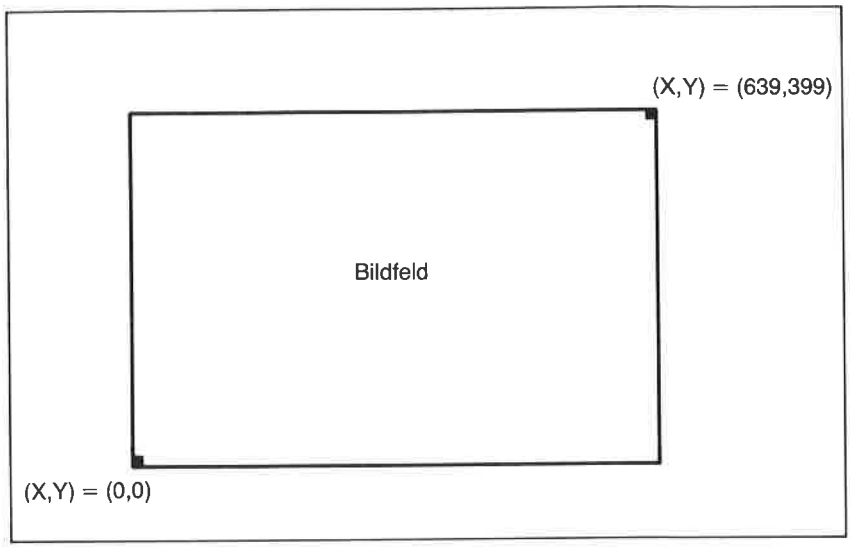



*Abb. 4.2: Die Koordinatenvereinbarungen für grafische Abbildungen*

### Die Organisation des Bildwiederholspeichers

Derjenige Bereich des Schreib-/Lesespeichers, der vom Videocontroller für die Abbildung grafischer Darstellungen verwendet wird, wird üblicherweise als Bildwiederholspeicher bezeichnet. Beim Schneider CPC beginnt dieser Speicherbereich bei der Adresse &C000 (dezimal −16384). In den Bytes dieses Bildwiederholspeichers ist neben der Bildinformation selbst auch ein Teil der für einzelne Bildpunkte gültigen Farbzuordnung verschlüsselt. Für einen Nichteingeweihten wird die Organisation des Speicherinhalts somit nur schwer durchschaubar sein. Hinzu kommt, daß die logische Zuordnung der Speicherinformation zur Bildstruktur beim CPC ebenfalls verhältnimäßig kompliziert ist.

Schauen wir uns in diesem Zusammenhang zunächst einmal in Abb. 4.3 an, wie die Verknüpfung des Speicherinhalts mit dem Bildschirminhalt aussieht.

Wie Sie leicht erkennen können, wird der Inhalt der niedrigsten Adresse &C000 in der oberen linken Ecke des Bildschirms abgebildet. Für die Abbildung der ersten horizontalen Bildpunktzeile stehen nach Abb. 4.3 insgesamt 80 Bytes bis zur Adresse &C04F zur Verfügung. Das nächstfolgende Byte im Bildwiederholspeicher mit der Adresse &C050 enthält

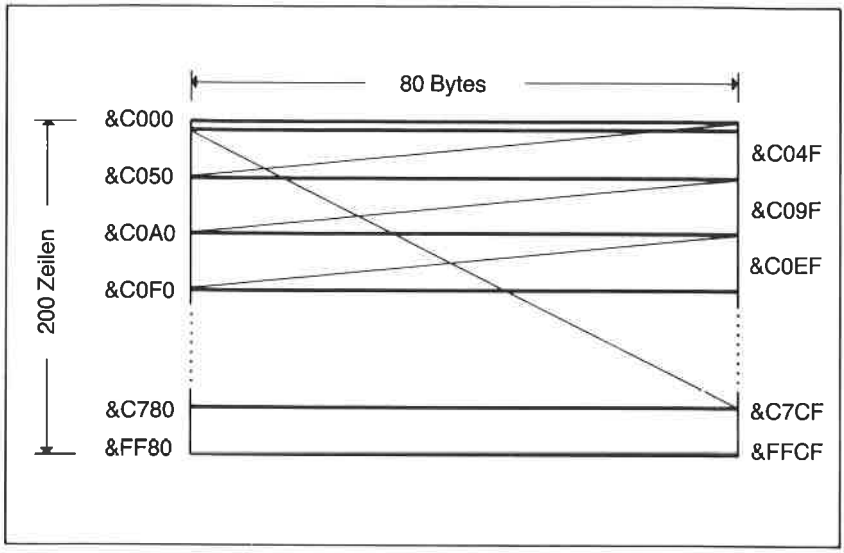



*Abb. 4.3: Zuordnung der Speicheradressen zum Bildschirminhalt bei der grafischen Darstellung*

allerdings nicht eine Information über den ersten Bildpunkt der nächstfolgenden Zeile, sondern vielmehr über den ersten Bildpunkt (oder auch die ersten Bildpunkte) der neunten Zeile. Dieses Schema setzt sich entsprechend fort, d. h. es werden immer 8 Bildpunktzeilen übersprungen. Das sieht recht unübersichtlich aus, und ist es auch. Im Anhang R finden Sie eine komplette Liste aller Anfangs- und Endadressen für die 200 darstellbaren Pixelzeilen.

Die Inhalte der einzelnen Bildspeicherbytes werden vom Controller in Abhängigkeit von der Betriebsart unterschiedlich interpretiert.

Am einfachsten gestalten sich die Verhältnisse im höchstauflösenden Modus, also im MODE 2. Jedes Bit eines Bildspeicherbytes enthält in diesem speziellen Fall die Information über einen einzelnen von insgesamt 640 horizontalen Bildpunkten (gesetzt oder nicht gesetzt, bzw. hell (Vordergrundfarbe) oder dunkel (Hintergrundfarbe)). Siehe hierzu auch die Abb. 4.4. Die Informationen über die jeweils aktuell gültigen Farben aus der Farbpalette sind nicht im Speicher explizit abgelegt. Sie werden vielmehr direkt an den Videocontroller übergeben.

In der Betriebsart MODE 1 mit 320 Bildpunkten horizontal wird die aktuelle Farbinformation für die Vordergrundfarbe mitverschlüsselt. Da –

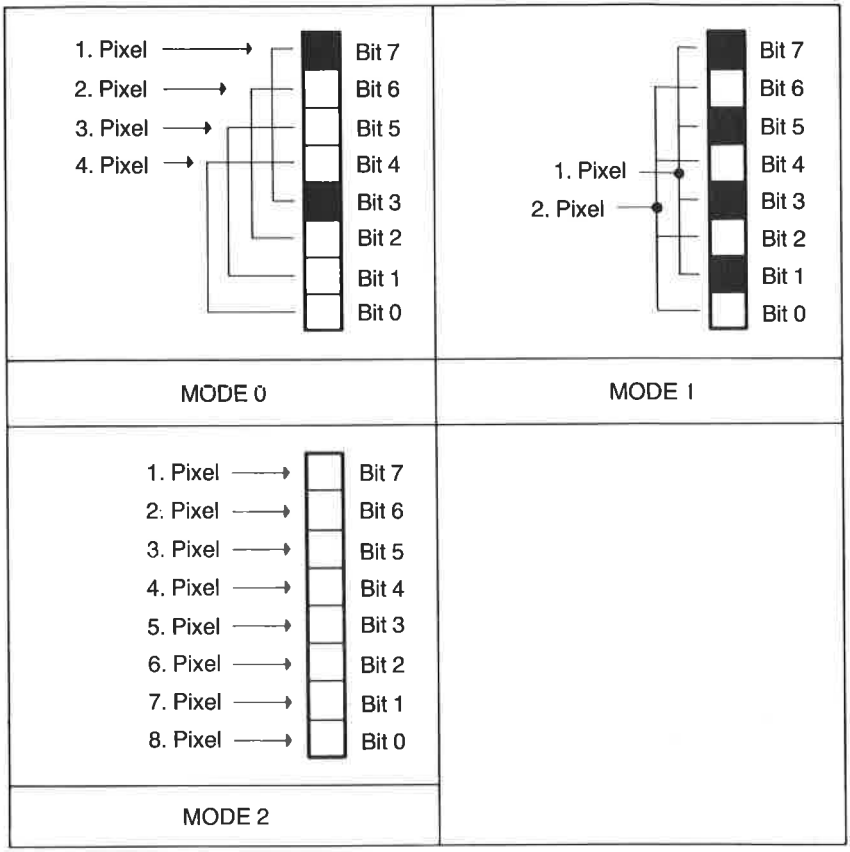



*Abb. 4.4: Erläuterung der Bildpunktverschlüsselung in den drei Darstellungsmodi MODE 0, MODE 1 und MODE 2*

wie Sie bereits wissen – im MODE 1 vier verschiedene Vordergrundfarben gleichzeitig dargestellt werden können, benötigen wir für jeden Bildpunkt zwei Bits zur Verschlüsselung. Abb. 4.4 zeigt, welche Bits jeweils für einen Bildpunkt zusammengefaßt werden. Die nachfolgende tabellarische Übersicht in Abb. 4.5 erläutert den Zusammenhang zwischen logischen Werten der einzelnen Bits und der Farbabbildung.

Da jeweils zwei Bits für einen Bildpunkt zur Verfügung gestellt werden müssen, sinkt die Auflösung bei konstanter Anzahl von Bytes pro Pixelzeile natürlich von 640 auf 320 Punkte horizontal.

| MODE 1 | | | MODE 0 | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 3 | Farbe | Bit 7 | Bit 5 | Bit 3 | Bit 1 | Pixel |
| 0 | 0 | Schwarz | 0 | 0 | 0 | 0 | Schwarz |
| 0 | 1 | Farbe 1 | 0 | 0 | 0 | 1 | Farbe 1 |
| 1 | 0 | Farbe 2 | 0 | 0 | 1 | 0 | Farbe 2 |
| 1 | 1 | Farbe 3 | 0 | 0 | 1 | 1 | Farbe 3 |
| Bit 6 | Bit 2 | Farbe | 0 | 1 | 0 | 0 | Farbe 4 |
| 0 | 0 | Schwarz | 0 | 1 | 0 | 1 | Farbe 5 |
| 0 | 1 | Farbe 1 | 0 | 1 | 1 | 0 | Farbe 6 |
| 1 | 0 | Farbe 2 | 0 | 1 | 1 | 1 | Farbe 7 |
| 1 | 1 | Farbe 3 | 1 | 0 | 0 | 0 | Farbe 8 |
| | | | 1 | 0 | 0 | 1 | Farbe 9 |
| Die Bitpaare 5 und 1 bzw 4 und 0 werden entsprechend verschlüsselt | | | 1 | 0 | 1 | 0 | Farbe 10 |
| | | | 1 | 0 | 1 | 1 | Farbe 11 |
| | | | 1 | 1 | 0 | 0 | Farbe 12 |
| | | | 1 | 1 | 0 | 1 | Farbe 13 |
| | | | 1 | 1 | 1 | 0 | Farbe 14 |
| | | | 1 | 1 | 1 | 1 | Farbe 15 |

*Abb. 4.5: Die Verschlüsselung der Farbwerte im MODE 0 und im MODE 1*

In der Betriebsart MODE 0 sind wegen der 16 verschiedenen, gleichzeitig darstellbaren Farben nur noch 160 Bildpunkte horizontal möglich. Wir benötigen hier nämlich nunmehr für jeden Bildpunkt 4 Bits, wie Sie sich anhand der Abb. 4.5 und 4.4 überzeugen können.

Natürlich brauchen Sie sich als BASIC-Programmierer oder als Anwender vorgefertigter Programme um die zuvor erläuterten Sachverhalte gar nicht zu kümmern. Interpreter wie auch Videocontroller machen als Folge der von Ihnen geschriebenen BASIC-Befehle automatisch alles richtig. Falls Sie jedoch einmal die Neigung verspüren, auf der Maschinenebene Grafikroutinen zu entwerfen, sollten Sie sich an das zuvor Gesagte erinnern.

## Die Grafikbefehle des BASIC-Interpreters

Der BASIC-Interpreter stellt Ihnen für die Erzeugung attraktiver farbiger Grafiken einige sehr leistungsfähige Befehle zur Verfügung. Der einfachste und am häufigsten verwendete Befehl lautet

PLOT <x-Koordinate>,<y-Koordinate>[,<Farbwert n>]

Er bildet an dem durch die Koordinatenwerte x,y vereinbarten Ort auf dem Bildschirm einen Bildpunkt ab. Wird die Farbangabe weggelassen, verwendet der Interpreter die aktuelle Vordergrundfarbe. Natürlich gelten auch hier die bereits an anderer Stelle erläuterten Regeln für die Farbabbildung in den unterschiedlichen Betriebsarten. Wird als Farbwert die Hintergrundfarbe vereinbart, ist der Bildpunkt nicht sichtbar.

Ein sehr ähnlicher Befehl lautet

```
PLOTR<X-Koordinatenoffset>,<Y-Koordinatenoffset>
[, <Farbwert n>]
```

Er unterscheidet sich von dem zuvor genannten Befehl nur dadurch, daß der angegebene Bildpunkt *relativ* zur aktuellen Position des *Grafikcursors* angesteuert und gesetzt wird. Verbunden mit dem Setzen eines Bildpunktes ist nämlich immer die Bewegung eines nicht sichtbaren Grafikcursors. Er steht beim Systemstart oder unmittelbar nach dem Aufruf eines der Abbildungsmodi MODE 0, MODE 1 oder MODE 2 auf x=0 und y=0. Nach einem PLOT-Befehl befindet er sich bei der im Befehl vereinbarten Ortskoordinate.

Der Grafikcursor läßt sich ohne Beeinflussung der Bildschirminformation durch die beiden Befehle

```
MOVE <x-Koordinate>,<y-Koordinate>[,<Farbwert n>]
(Bewegung absolut)
```

bzw.

```
MOVER <x-Koordinatenoffset>,<y-Koordinatenoffset>
[,<Farbwert n>]
(Bewegung relativ)
```

an jede beliebige Stelle des Bildschirmfensters setzen. Die beiden letzten, zur gleichen Kategorie von Grafikbefehlen zählenden Anweisungen lauten:

```
DRAW <x-Koordinate>,<y-Koordinate>[,<Farbwert n>]
(Bewegung absolut)
```

bzw.

```
DRAWR <x-Koordinatenoffset>,<y-Koordinatenoffset>
[,<Farbwert n>]
(Bewegung relativ)
```

Mit Hilfe dieser Befehle wird zwischen der aktuellen Ortskoordinate des Grafikcursors und dem durch die Koordinatenangabe oder den Koordinatenoffset vereinbarten Zielpunkt eine gerade Linie gezeichnet. Wegen der endlichen Auflösung kann es je nach Orientierung dieser Linie auf dem Bildschirm zu einer groben Stufendarstellung kommen.

Wir wollen uns die Wirkung aller drei Befehle einmal im Zusammenhang anhand eines kleinen Programms anschauen, das in einer Schritt für Schritt ablaufenden Grafik die Verknüpfung zwischen den Winkelabschnitten in einem Kreis und der Sinusfunktion demonstriert.

Lassen Sie das Programm bitte nach der Eingabe probeweise in allen drei Betriebsarten nacheinander ablaufen, damit Sie sich davon überzeugen können, daß die Koordinatenangaben bei unterschiedlicher Vereinba-

Programm: Kreisfunktion mit Sinus

```
5 REM **** Demo einer Sinusfunktion ****
10 MODE 2
20 BORDER 0:INK 0,0:INK 1,26
30 CLS
40 GOSUB 270
50 r0=90
60 FOR i%=0 TO 44
70 x=r0*COS(i%*PI/180)
80 y=r0*SIN(i%*PI/180)
90 PLOT X+100,Y+200
100 PLOT -X++100,Y+200
110 PLOT -X+100,-Y+200
120 PLOT X+100,-Y+200
130 PLOT Y+100,X+200
140 PLOT -Y+100,X+200
150 PLOT -Y+100,-X+200
160 PLOT Y+100,-X+200
170 NEXT i%
180 FOR i%=0 TO 360 STEP 10
190 x=r0*COS(i%*PI/180)
200 y=r0*SIN(i%*PI/180)
210 PLOT 100,200:DRAW x+100,y+200:PEN 2:DRAWR 0,-y
220 PEN 1
230 PLOT 300+i%*300/360,200:DRAWR 0,y
240 FOR k%=1 TO 500:NEXT k%
250 NEXT i%
260 END
270 PLOT 100,100:DRAWR 0,200
280 PLOT 0,200:DRAWR 200,0
290 PLOT 300,200:DRAWR 300,0
300 LOCATE 40,3:PRINT "Demo einer Sinusfunktion"
310 RETURN
```

rung der Modi nicht geändert zu werden brauchen. Sie brauchen dazu nur die Zeile 10 zu ändern.

Neben den reinen Zeichenbefehlen enthält der Interpreter noch einige Zusatzbefehle, die im Zusammenhang mit raffinierten Programmabläufen sehr wertvoll sind. So sind beispielsweise die Abmessungen des Grafikfensters auf dem Bildschirm wie auch die Lage des Koordinatennullpunktes über den Befehl

```
ORIGIN <x-Koordinate>>,<y-Koordinate>[,<linker Rand>,
<rechter Rand>,<oberer Rand>,<unterer Rand>]
```

in weiten Grenzen veranderbar.

Unter einem Grafikfenster wird jener Bereich des Bildfensters verstanden, innerhalb dessen die zuvor erläuterten Grafikbefehle PLOT, PLOTR, DRAW und DRAWR zu einer sichtbaren Abbildung führen.

Die Änderung der Lage des Koordinaten*nullpunktes* wird über die ersten beiden Parameter im ORIGIN-Befehl erreicht. Dieser braucht – wie alle anderen Koordinatenangaben auch – nicht unbedingt in den Grafikfensterbereich zu fallen. Die Fenstergrenzen werden dagegen über die letzten vier Parameter vereinbart.

Ergänzen Sie in diesem Zusammenhang einmal das zuvor abgedruckte Programm durch die folgende Programmzeile:

```
15 ORIGIN 0,0,100,300,200,300
```

und starten Sie das Programm erneut. Wie Sie sehen, sind jetzt nur noch jene Teile der Grafik sichtbar, die in das neu definierte Grafikfenster fallen.

Das nachfolgend angegebene Beispielprogramm zeichnet nullpunktsymmetrische Kreise auf den Bildschirm, deren Durchmesser und Ortslage dem Zufall unterworfen sind. Beachten Sie, daß wegen der Nullpunktsymmetrie der Programmteil zur Kreisabbildung erheblich einfacher ausfallen kann, als dies im Programm zuvor der Fall ist.

Es ist im übrigen, wie bei allen anderen Koordinatenangaben in Grafikbefehlen auch, gestattet, den Nullpunkt außerhalb des Bildschirmfeldes zu vereinbaren.

Programmbeispiel: Nullsymmetrische Kreise mit dem Zufall unterworfener Ortslage

```
10 MODE 1
20 BORDER 0:INK 0,0:INK 1,26
30 ORIGIN 550*RND(1),310*RND(1)
40 R0=90*RND(1)
50 FOR i%=0 TO 44 STEP 4
60 x=r0*COS(i%*PI/180)
70 y=r0*SIN(i%*PI/180)
80 PLOT X,Y
90 PLOT -X,Y
100 PLOT -X,-Y
110 PLOT X,-Y
120 PLOT Y,X
130 PLOT -Y,X
140 PLOT -Y,-X
150 PLOT Y,-X
160 NEXT i%
170 GOTO 30
```

Für ein Grafikfenster gibt es, wie bei den Textfenstern, einen Befehl, der den angesprochenen Bildschirmbereich löscht und gegebenenfalls auf eine andere Hintergrundfarbe setzt. Er lautet

CLG [<Farbwert n>]

Wird die Farbangabe weggelassen, verwendet das System entweder n=0 oder die beim letzten CLG-Aufruf vereinbarte Farbe.

Das bei Systemen der Heim- und sogar der Personalcomputerklasse nur schwer lösbare Problem der gleichzeitigen Abbildung von Text *und* Grafik existiert beim Schneider CPC nicht. Außergewöhnlich ist in diesem Zusammenhang, daß Textausgaben sich dabei nicht an das übliche Zeichenraster halten müssen, sondern vielmehr bildpunktgenau positioniert werden können. Dem Programmierer steht hierfür der Befehl TAG (*T*ext *A*nd *G*raphics) zur Verfügung.

Die Wirkung dieses Befehls besteht darin, daß das erste Zeichen eines Textes mit der linken unteren Ecke des Matrixfeldes genau auf den aktuellen Ort des Grafikcursors positioniert wird.

Am besten sehen Sie sich die Wirkung anhand eines kleinen Programmbeispiels an, das Sie auch zu anderen Experimenten anregen sollte: Es zeichnet Punkt für Punkt eine Sinuskurve auf dem Bildschirm. Die Koordinaten des gerade ausgegebenen Punktes werden unmittelbar daneben

als Zahlenwerte ausgegeben. Damit Sie den Vorgang in Ruhe verfolgen können, enthält das Programm eine Verzögerungsschleife. Wenn Sie diese entfernen, führt der Text eine fließende, wellenförmige Bewegung auf dem Bildschirm aus. Ein weiteres Beispiel finden Sie in der Liste der BASIC-Befehle in Kapitel 3.

Programmbeispiel für den TAG-Befehl:

```
5 REM **** TAG-Demo ****
10 CLS:MODE 1
20 DEG
30 N=0
40 FOR I%= 0 TO 360
50 Y =200+190*SIN(2*I%)
60 PLOT N,Y
70 TAG
80 PLOT N+5,Y:PRINT I%;
90 N=N+2
100 FOR J=1 TO 200:NEXT J
110 PLOT N+5,Y:PRINT "      ";
120 NEXT I%
130 TAGOFF
```

Die Wirkung des TAG-Befehls wird durch die Anweisung TAGOFF aufgehoben. Nachfolgender Text erscheint wieder an der durch den Textcursor festgelegten Stelle. Beachten Sie bitte, daß der Text unter Kontrolle der TAG-Anweisung in jener Farbe auf dem Bildschirm ausgegeben wird, die zuletzt für einen *Zeichenbefehl* verwendet wurde. Falls Sie sicherstellen wollen, daß die Text- oder auch die Grafikinformationen an einem bestimmten Punkt des Grafikfensters in einem vom Hintergrund abweichenden Farbton ausgegegeben werden, sollten Sie den Befehl

TEST (<x-Koordinate>,<y-Koordinate>)

oder

TESTR (<x-Koordinatenoffset>,<y-Koordinatenoffset>)

anwenden, der für die vereinbarten Koordinaten den Farbwert des Grafikcursors ermittelt. Sie können so jeden auszugebenden Punkt in Abhängigkeit von der Bildpunktabfrage durch TEST oder TESTR individuell in der Farbgebung steuern. Das folgende Programm demonstriert dies anhand eines Beispiels.

Demoprogramm

```
5 REM **** TEST-Demo ****
10 MODE 1:PEN 0
20 WINDOW #0,1,19,1,25
30 PAPER 3:CLS
40 PRINT "FARBSTIFT:";TEST(100,200)
50 WINDOW #0,20,40,1,25
60 PAPER 1:CLS
70 PRINT "FARBSTIFT:";TEST(400,200)
80 WINDOW #0,1,40,1,25:PAPER 3
90 FOR I=2 TO 629 STEP 17
100 IF TEST (I,200)=3 THEN X=1:Y=3 ELSE X=3:Y=1
110 PLOT I,200,Y
120 DRAWR 10,0,X:DRAWR 0,-10,X:DRAWR -10,0,X:DRAWR 0,10
,X
130 FOR WART=1 TO 150:NEXT WART
140 PLOT I,200,Y:DRAWR 10,0,Y:DRAWR 0,-10,Y:DRAWR -10,0
,Y:DRAWR 0,10,Y
150 NEXT I
160 PRINT:PRINT
```

Sind Sie sich im übrigen unsicher, welche aktuelle Position der Grafikcursor gerade einnimmt, können Sie seine horizontale wie auch seine vertikale Koordinate durch Anwendung der Funktionen XPOS und YPOS an das Programm weitergeben. Dies ist insbesondere dann sehr nützlich, wenn Sie sich mit dem Grafikcursor außerhalb des sichtbaren Bereichs des Grafikfensters bewegen oder wenn Sie sicherstellen wollen, daß der Grafikcursor den Definitionsbereich ihres Grafikfensters nicht verläßt.

## Die Grafikerweiterung des CPC 664

Alle im vorangegangenen Abschnitt vorgestellten Grafikbefehle sind sowohl auf dem CPC 464 als auch auf dem CPC 664 einsetzbar. Die Kommandos DRAW, DRAWR, MOVE, MOVER, PEN, PLOT UND PLOTR können auf dem CPC 664 jedoch um einen oder auch zwei Parameter ergänzt werden, die ihre Leistungsfähigkeit beträchtlich steigern. Bei den Befehlen DRAW, DRAWR, PLOT und PLOTR besteht die Möglichkeit, durch einen Abbildungsmodus die logische Verknüpfung zwischen der aktuell gewählten Abbildungsfarbe und der bereits bestehenden Farbe der betroffenen Pixels festzulegen. Wie Sie der Übersicht über die entsprechenden Befehle in Kapitel 3 entnehmen können, dürfen die entsprechenden Zusatzparameter Werte zwischen 0 und 3 annehmen.

Dabei gilt:

0: normaler Abbildungsmodus
1: Exklusiv-ODER-Verknüpfung zwischen der aktuellen Abbildungsfarbe und dem bereits bestehenden Farbwert der Pixel (XOR)
2: UND-Verknüpfung zwischen der aktuellen Abbildungsfarbe und dem bereits bestehenden Farbwert der Pixel (AND)
3: ODER-Verknüpfung zwischen der aktuellen Abbildungsfarbe und dem bereits bestehenden Farbwert der Pixel (OR)

Zu Ihrer Information ist in der Abb. 4.6 nochmals die bereits in Kapitel 3 gezeigte logische Verknüpfung zweier Werte a und b angegeben.

Für das Verständnis ist es wichtig, daß unabhängig von der Farbe nur der Zustand „Bildpunkt gesetzt" und „Bildpunkt nicht gesetzt" berücksichtigt wird. Auch hier ist ein Beispiel besser als viele Worte. Der Befehl

```
PLOT 100,100,3,2
```

setzt im MODE 0 einen Bildpunkt in der Farbe Leuchtendrot dann und nur dann, wenn zuvor der betreffende Punkt bereits gesetzt war. Maßge-

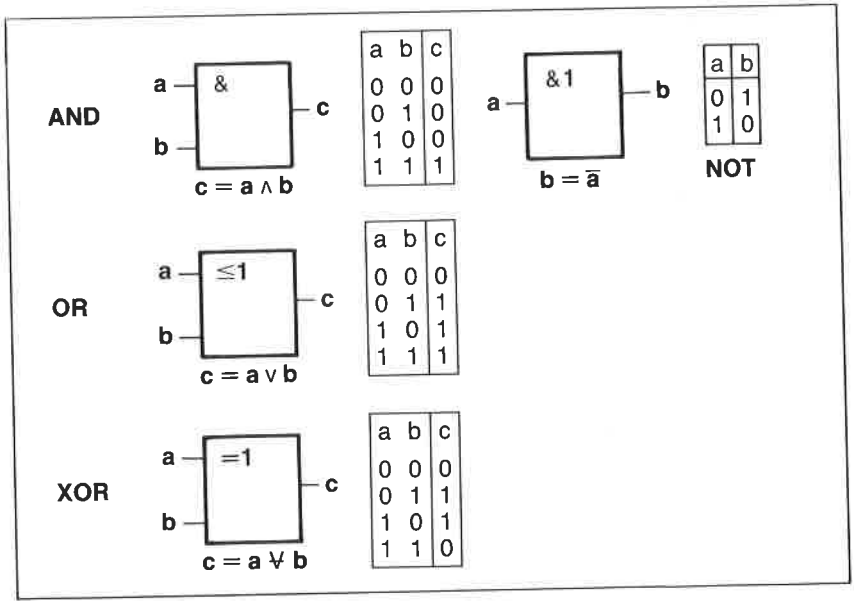



*Abb. 4.6: Logische Verknüpfungen von zwei binären Variablen a und b*

bend für die ausgegebene Farbe ist die zuletzt (also im PLOT-Befehl) gegebene Vereinbarung. Für den Fall, daß der Befehl

```
PLOT 100,100,3,1
```

benutzt wird, führt das System eine Exklusiv-ODER-Verknüpfung aus. Das bedeutet, daß der im PLOT-Befehl vereinbarte Bildpunkt nur dann gesetzt wird, wenn er zuvor nicht gesetzt war und jetzt wird, oder wenn er jetzt nicht gesetzt wird und zuvor bereits gesetzt war.

Dieselben Vereinbarungen gelten für den PLOTR-, den DRAW- und DRAWR-Befehl.

Besonders angenehm ist, daß der Interpreter des CPC 664 bei den üblicherweise für die Bildpunktpositionierung verwendeten Befehlen MOVE und MOVER bereits eine Vorvereinbarung der Farbe für die nachfolgende Grafik sowie zusätzlich die Angabe eines Farbmodus ermöglicht. Das spart zusätzliche Befehle und damit kostbaren Speicherplatz. Ergänzt werden diese Befehle durch die globalen Vereinbarungen GRAPHICS PAPER und GRAPHICS PEN, die zur Wahl der Hintergrundfarbe im Grafikfenster sowie zur Festlegung der Abbildungs- (Vordergrund-)Farbe dienen. In dem Beispielprogramm mit dem Namen BALKENDIAGRAMM werden einige der zuvor erwähnten Befehle in ihrer

```
10 REM *** Balkendiagramm ***
20 CLS:MODE 1
30 ORIGIN 50,100,50,550,100,350
40 CLG 2
50 FOR i=0 TO 490 STEP 50
60 y=250*RND(1)
70 MOVE i,0,3
80 DRAWR 0,y:DRAWR 40,0
90 DRAWR 0,-y:DRAWR -40,0
100 MOVER 2,2:FILL 3
110 NEXT i
120 MASK 255*RND(1)
130 FOR i=0 TO 250 STEP 2
140 MOVE 0,i,1:DRAWR 500,0,,1
150 NEXT i
160 FOR i=250 TO 0 STEP -2
170 MOVE 0,i:DRAWR 500,0,1,1
180 NEXT i
190 GOTO 130
```

erweiterten Form angewendet. Bitte beachten Sie, daß dieses Programm auf einem CPC 464 ohne Änderungen nicht lauffähig ist, da die benutzte Syntax von dessen Interpreter nicht verstanden wird. Das Programm macht im übrigen auch von dem MASK-Befehl zur gerasterten Darstellung der Balken Gebrauch.

Kapitel 5

# Musik- und Geräuscherzeugung

## Übersicht

Neben den grafischen Leistungen des Schneider CPC beeindrucken vornehmlich die Möglichkeiten der Klang- und Geräuscherzeugung. Der CPC stellt für den genannten Zweck einen höchstintegrierten Baustein des Typs AY-3-3912 zur Verfügung. Er wird abkürzend als PSG bezeichnet (*P*rogrammable *S*ound *G*enerator). Besonders erfreulich für den BASIC-Programmierer ist, daß der Interpreter eine Reihe von höchst leistungsfähigen Kommandos zur Programmierung dieses Bausteins zur Verfügung stellt, die auch dem Einsteiger die Erzeugung von Musik und Geräuschen erleichtern. In diesem Kapitel finden Sie neben einigen grundsätzlichen Erläuterungen zur Physik der Erzeugung musikalischer wie auch nichtmusikalischer Klangereignisse einige Hintergrundinformationen über die Funktionsweise des PSG sowie Hinweise zu dessen Programmierung. Eine Reihe erläuternder Programmbeispiele vertieft die erworbenen Kenntnisse und zeigt Ihnen einen Weg zur Entwicklung eigener Programme zu dem angeschnittenen Themenkreis.

## Zur Physik der Tonerzeugung

Alle vom Menschen über das Ohr wahrnehmbare Schallereignisse beruhen auf der Auswertung von sehr schnellen Luftdruckschwankungen. Sie regen das Trommelfell zu Schwingungen an, die über ein kompliziertes und ausgeklügeltes Übertragungssystem an das Gehirn weitergeleitet werden. Der zeitliche Verlauf dieser Druckschwankungen bestimmt weitgehend den Charakter unserer Schallempfindung.

Einen sehr einfachen und mathematisch eindeutig beschreibbaren Schwingungsverlauf besitzt ein reiner Ton, der nach einem hier nicht näher interessierenden physikalisch-mathematischen Gesetz als Aufbauelement für höchst verschieden klingende musikalische und auch

nichtmusikalische Schallereignisse dienen kann. Die einen reinen Ton erzeugende Schwingung wird als harmonische Schwingung und der durch sie erzeugte Ton als Sinuston bezeichnet. Abb. 5.1 zeigt, wie so ein Schwingungsverlauf in Abhängigkeit von der Zeit aussieht.

Eine harmonische Schwingung nach der in Abb. 5.1 gezeigten Art verläuft *periodisch*, d. h., der innerhalb einer Zeitspanne T ersichtliche Kurvenverlauf wird fortlaufend wiederholt. Diese Zeitspanne T wird *Periodendauer* genannt. Sie sollten sich diesen Begriff merken, da er Ihnen bei der Programmierung des PSG in vielfältiger Form wieder begegnen wird. Der Kehrwert der Periodendauer T wird als *Frequenz* bezeichnet und im allgemeinen mit dem kleinen Buchstaben f benannt. Da die Periodendauer in Vielfachen von Sekunden gemessen wird, also die Dimension einer Zeit besitzt, wird die Frequenz mit der Dimension 1/Sekunde angegeben. Hierfür ist die Bezeichnung Hertz (Hz) gebräuchlich. Es gilt somit:

$$f = \frac{1}{T}$$

Beachten Sie bitte: Je größer die Periodendauer T, desto niedriger ist die Frequenz. Die Frequenz bzw. die Periodendauer ist insofern für uns so wichtig, weil Sie die *Tonhöhe* bestimmt.

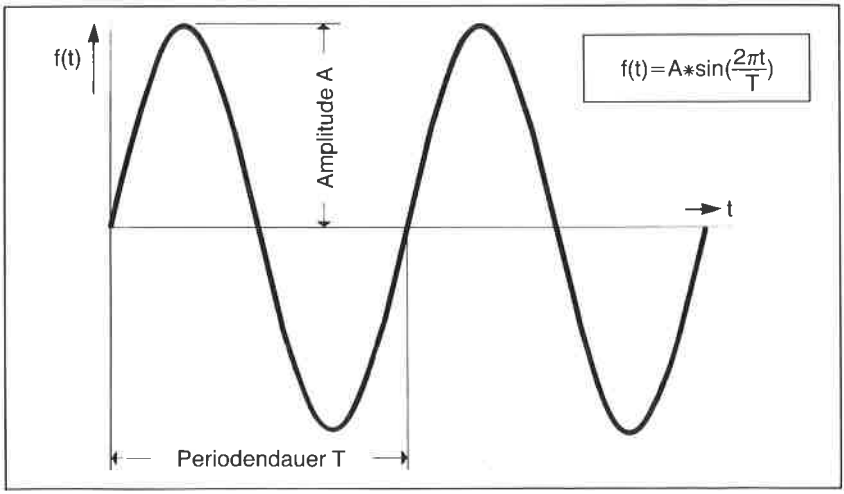



*Abb. 5.1: Zeitlicher Verlauf einer Sinusschwingung*

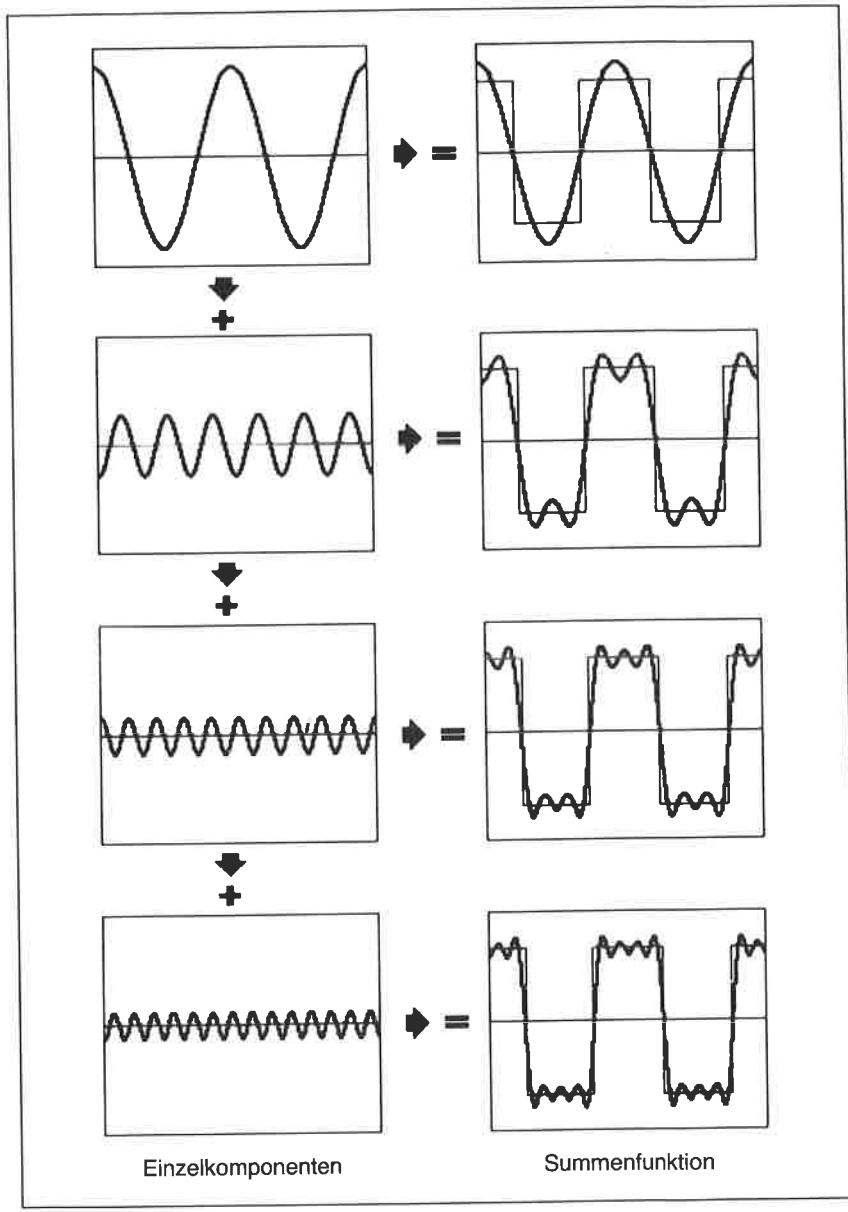



*Abb. 5.2: Beispiel für die Erzeugung eines nahezu rechteckförmigen periodischen Amplitudenverlaufs durch Addition verschiedener harmonischer Schwingungen*

Ebenso wichtig wie die Angabe der Periodendauer ist die der Schwingungsamplitude. Sie bestimmt nämlich bei einem Schallereignis, wie laut dieses von uns empfunden wird. In Abb. 5.1 ist die Amplitude mit A bezeichnet.

Es ist möglich, durch Veränderung der Frequenz eines reinen Tones Musik zu erzeugen. Besonders gut klingt das Ergebnis allerdings nicht. Das liegt daran, daß ein reiner Ton sehr rund und langweilig klingt. Musikinstrumente erzeugen Schwingungen, die zwar (etwas vereinfacht gesehen) auch periodisch sind, aber einen wesentlich komplizierteren Verlauf der Amplitude in Abhängigkeit von der Zeit aufweisen. Im Prinzip ist es möglich, durch Addition einer Vielzahl harmonischer Schwingungen unterschiedlicher Frequenz, Amplitude und Zeit*verschiebung* den Klangcharakter praktisch jedes Instrumentes künstlich zu erzeugen. Abb. 5.2 zeigt dies anhand der künstlichen Synthese einer periodischen Schwingung, die bei Wiedergabe über einen Lautsprecher etwa den Klang einer Klarinette vortäuscht.

Von dieser Möglichkeit machen beispielsweise Klangsynthesizer Gebrauch, die heute nichts anderes mehr sind als Computer mit speziel-

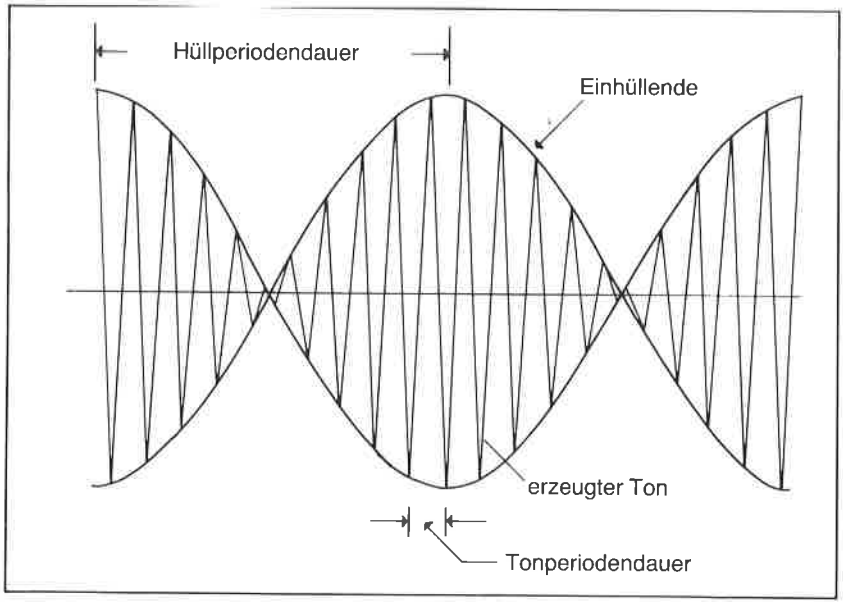



*Abb. 5.3: Anschauliche Erläuterung der Begriffe „Einhüllende" und „Hüllperiodendauer"*

len, auf die Musik- und Geräuscherzeugung zugeschnittenen Möglichkeiten. Auch Sie können mit dem PSG Ihres Schneider CPC 464/664 ähnliche Dinge erreichen, wie Sie noch sehen werden.

Neben dem reinen Zeitverlauf der klangerzeugenden Schwingung selbst ist für den *Klangcharakter* noch die zeitliche Veränderung der Schwingungs*amplitude*, d. h. die Lautstärkeveränderung, sowie unter Umständen noch die zeitliche Veränderung der *Frequenz*, d. h. die Tonhöhenveränderung, bestimmend. Die Funktion, die für die entsprechenden Veränderungen der Parameter verantwortlich ist, wird im allgemeinen kurz als *Einhüllende* bezeichnet.

Besonders anschaulich wird dieser Begriff bei der zeitabhängigen Veränderung der Lautstärke. Die verändernde Funktion hüllt die eigentliche periodische Schwingung ein, wie die Abb. 5.3 anhand einer schematischen Darstellung zeigt und erläutert. Sie ist bestimmt durch die Hüllfrequenz $f_H$ bzw. deren Kehrwert, der Hüllperiodendauer $T_H$.

Neben den musikalischen Schallereignissen lassen sich auch nichtmusikalische Klänge vom Kanonendonner, dem Geräusch eines Besens bis hin zum Zischen einer Dampflokomotive künstlich erzeugen. Wesentlich ist dabei, daß die erzeugenden Funktionen keinen periodischen Charakter besitzen und zum Teil sehr schnellen Amplitudenwechseln unterworfen sind. Eine Vielzahl bemerkenswerter Geräusche lassen sich durch Veränderungen von sogenanntem Rauschen erzielen, das im allgemeinen durch spezielle Generatoren erzeugt wird. Es ist dadurch gekennzeichnet, daß der Schwingungsverlauf völlig regellos ist. Er setzt sich physikalisch aus einer unendlichen Vielzahl von Schwingungen statistisch schwankender Amplitude zusammen.

Nach diesen einleitenden Bemerkungen zur Physik der Klangerzeugung wollen wir uns im nächsten Abschnitt den für die Musik- und Geräuscherzeugung dienenden PSG des Typs AY-3-3912 einmal näher ansehen, um seine Fähigkeiten richtig einschätzen und programmtechnisch umsetzen zu können.

## Der programmierbare Soundgenerator

Beim PSG AY-3-8912 sind eine Vielzahl von speziellen Funktionseinheiten, wie beispielsweise drei Tongeneratoren, ein Rauschgenerator und Elemente zur Mischung und Beeinflussung der Signale auf dem Chip integriert. Die digital erzeugten Signale können über drei voneinander unab-

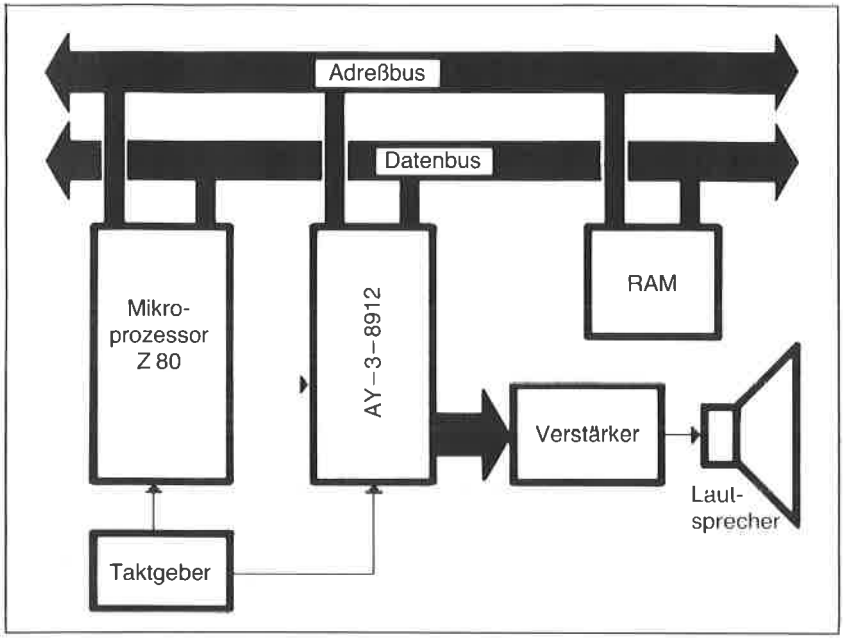



*Abb. 5.4: Schema für die Zusammenarbeit zwischen PSG und anderen Funktionseinheiten des CPC 464/664*

hängige Kanäle A, B und C als analoge Signale ausgegeben werden. Die Verknüpfung dieses hochkomplexen Bauelementes mit anderen Funktionseinheiten des Computers zeigt die Abb. 5.4.

Der PSG-Baustein kann von der BASIC-Ebene aus mittels der Befehle SOUND, ENT und ENV angesprochen und gezielt programmiert werden. Die Adressierung der internen Register und der Datenverkehr werden unter Interpreterkontrolle automatisch vorgenommen. Die in den BASIC-Befehlen zu vereinbarenden Parameter werden verständlicher, wenn Sie sich die Registerstruktur des AY-3-3912 in der folgenden Abb. 5.5 etwas näher ansehen. Programmiert werden können unter anderem die Tonhöhen sowie die Lautstärken der Kanäle A, B und C, die Wiederholfrequenz des binären Rauschgenerators und die Wiederholperioden der Einhüllenden sowie deren Form.

Die Programmierung geschieht durch Einschreiben binärer Datenworte in 8-bit-Register bzw. zu 16-bit-Registern kombinierte Registerpaare.

| Register \ Bit | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|
| R0 | Kanal A | 8-bit-Feineinstellung Kanal A | | | | | | | |
| R1 | Tonhöhe | | | | | 4-bit-Grobton | | | |
| R2 | Kanal B | 8-bit-Feineinstellung Kanal B | | | | | | | |
| R3 | Tonhöhe | | | | | 4-bit-Grobton | | | |
| R4 | Kanal C | 8-bit-Feineinstellung Kanal C | | | | | | | |
| R5 | Tonhöhe | | | | | 4-bit-Grobton | | | |
| R6 | Rauschfrequenz | | | | | 5-bit-Zeitkontrolle | | | |
| R7 | Freigabe | Ein/Aus | | Rauschen | | | Ton | | |
| | | IOA | IOB | C | B | A | C | B | A |
| R10 | Lautstärke A | | | | M | L3 | L2 | L1 | L0 |
| R11 | Lautstärke B | | | | M | L3 | L2 | L1 | L0 |
| R12 | Lautstärke 10 | | | | M | L3 | L2 | L1 | L0 |
| R13 | Hüllkurvendauer | 8-bit-Feineinstellung | | | | | | | |
| R14 | | 8-bit-Grobeinstellung | | | | | | | |
| R15 | Hüllkurvenform | | | | | CONT | ATT | ALT | HLD |
| R16 | Ein-/Ausgabe A | 8-bit-parallel-Ein-/Ausgang Tor A | | | | | | | |

*Abb. 5.5: Die Registerstruktur des PSG AY-3-8912*

### Die Tonhöhenwahl

Die Höhe des am Kanal A anliegenden Tons wird durch das in das Registerpaar R0 und R1 eingeschriebene Datenwort und die Frequenz des steuernden Taktgenerators bestimmt. Die 8 Bit des niederwertigen Bytes dienen zur Feineinstellung, die ersten 4 Bit des höherwertigen Bytes zur Grobeinstellung. Mit den insgesamt 12 Bits lassen sich bis zu 4096 verschiedene Tonhöhenwerte einstellen. (Die höchstwertigen Bits des Registers R1 werden nicht benutzt. Die gleiche Vereinbarung gilt für die Registerpaare R2/R3 und R4/R5. Der in eines der genannten Register einzuschreibende Datenwert d errechnet sich für eine gewünschte Frequenz f in der Maßeinheit Hz zu

$$d = \text{Taktfrequenz in Hz}/(16 * f)$$

| Notenbe-zeichnung | Frequenz in Hz | Register-wert |
|---|---|---|
| c | 261,6 | 239 |
| cis | 277,2 | 225 |
| d | 293,7 | 213 |
| dis | 311,1 | 201 |
| e | 329,6 | 190 |
| f | 349,2 | 179 |
| fis | 379,0 | 165 |
| g | 392,0 | 159 |
| gis | 415,3 | 150 |
| a | 440,0 | 142 |
| b | 466,2 | 134 |
| h | 493,9 | 127 |

*Abb. 5.6: Frequenzen und Registerwerte für die Noten $a^1$ bis $h^1$*

Bei der für den CPC 464/664 gültigen Taktfrequenz von 1 MHz (1 Million Hertz) muß somit für den Kammerton a (440 Hz) der Wert d = 1000000/(16*440) = 142 geladen werden. Nach der Frequenz aufgelöst, lautet die Beziehung:

f in Hz = Taktfrequenz in Hz/(16*d)

Da von Null verschiedene Datenwerte zwischen d=1 und d=4095 in die Register eingeschrieben werden können, beträgt der vom PSG überstrichene Frequenzbereich 15,25 Hz bis 62,5 kHz.

In der Zusammenstellung in Abb. 5.6 finden Sie für jene Oktave, die den Kammerton a mit einer Frequenz von 440 Hz enthält, die Frequenzen und zugehörigen Registerdatenwerte. Eine für den SOUND-Befehl gültige vollständige Tabelle finden Sie im Anhang O. Beachten Sie bitte, daß die Angaben im Anhang VII des Handbuchs zum Schneider CPC 464 fälschlicherweise um eine Oktave verschoben sind. Bei Verwendung der zuvor angegebenen Beziehungen erhalten Sie die korrekten Zahlenwerte.

### *Die Wiederholrate des binären Rauschsignals*

Der Rauschgenerator erzeugt ein binäres Pseudorauschen mit einer einstellbaren Wiederholrate. Diese wird über die niederwertigen 5 Bits des

Registers R6 eingestellt. Das Rauschen besitzt einen tonalen Anteil und somit eine von R6 abhängige Klangfarbe, deren Fundamentalfrequenz sich zwischen 62,5 kHz (Registerwert 1) und 2 kHz (Registerwert 15 bzw. &1F) bewegt.

### *Die Lautstärkewahl*

Die Grundlautstärke kann für die einzelnen Kanäle durch einen Schreibvorgang in die Register R10 bis R12 vereinbart werden. Über die mit L0 bis L3 gekennzeichneten vier niederwertigen Bits kann ein zeitunabhängiger Lautstärkewert zwischen 0 und 15 angesteuert werden. Über das fünfte, mit M bezeichnete Bit wird die Lautstärkekontrolle an das Register R15 abgegeben. Dieses ruft eine der im Chip vorprogrammierten Lautstärkeeinhüllenden ab und aktiviert so einen zeitabhängigen Verlauf der Lautstärke (siehe auch Abb. 5.8).

### *Die Betriebsarten*

Über das 8-bit-Register R7 ist die Klangkombination sowie die Betriebsweise des Ein-/Ausgangstors einstellbar. Die niederwertigen drei Bits B0 bis B2 beziehen sich auf den Ton-, die nächstfolgenden drei Bits B3 bis B5 auf den Rauschgenerator.

Es sind für die Ton- wie die Geräuscherzeugung jeweils 8 Kanalkombinationen möglich. Betrachten wir nur die unteren drei Bits B0 bis B2, die

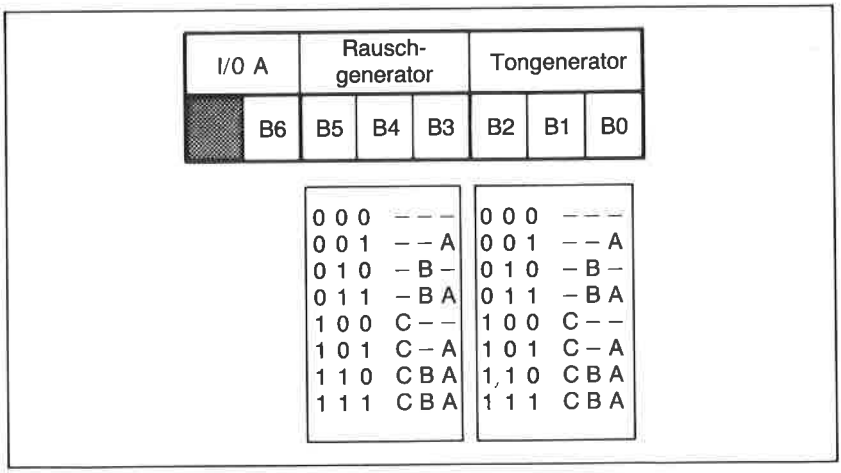



*Abb. 5.7: Interne Organisation des Registers R7*

sich nach Abb. 5.6 auf die Tongeneratoren beziehen, dann ergeben sich die in Abb. 5.7 gezeigten Kombinationsmöglichkeiten. Für B3 bis B5 gelten entsprechende Vereinbarungen. Bit B6 schaltet das einzige belegte Ein-/Ausgangstor A des AY-3-3912 ein bzw. aus. Bit B7 ist beim AY-3-8912 im CPC 464/664 nicht belegt.

Wie Sie sehen, existiert eine Fülle von Kombinationsmöglichkeiten zur Mischung der Ton- und Geräuschausgänge. Unter der Kontrolle des BASIC-Interpreters weichen die Registerfunktion und die Bedeutung der einzelnen Bits etwas von der herstellerseitig vorgesehenen Art ab.

### *Die Hüllkurvenform und Frequenzmodulation*

Das Registerpaar R13/R14 steuert zusammen mit R15 die Parameter der Hüllkurvenfunktionen. R13/R14 ist hierbei für die *Frequenz- bzw. Tonhöhenmodulation* zuständig. Da das Registerpaar Datenworte von 16 bit Wortbreite (&0001 bis &FFFF) speichern kann, sind insgesamt 65535 von Null verschiedene Werte für die Hüllfrequenz definierbar. Der in das Registerpaar einzuschreibende Wert $d_H$ kann bei gegebener Hüllfrequenz $f_H$ mittels der Beziehung

$d_H$= Taktrate in Hz/(256∗fH)

errechnet werden. Auch hier ist für die Taktrate der Wert 1 MHz anzusetzen.

Vorprogrammierte *Hüllkurvenverläufe* lassen sich über das Register R15 abrufen, das einen Digital-Analog-Umsetzer mit einer Auflösung von 4 bit steuert. Die vier höchstwertigen Bits des Registers werden nicht verwendet. Jedes der vier niederwertigen Bits besitzt eine definierte Bedeutung. Im einzelnen gilt für B0 bis B3:

B0: *Halten* — Falls B0 logisch 1 ist, wird nach Ablauf eines Zyklus der Einhüllenden der Endwert festgehalten.

B1: *Wechsel* — Wenn B1 gesetzt ist, wird die Richtung des Amplitudenverlaufs der Einhüllenden nach jedem Zyklus umgekehrt.

B2: *Anstieg* — Der Hüllkurvenverlauf wird über Digital-Analog-Umsetzer (DAU) in kleinen Schritten erzeugt, die durch 4-bit-Zähler kontrolliert werden. Ist B2 logisch Null, läuft der entsprechende Zähler von dual 1111 auf 0000 herunter; ist B2 gesetzt, wird von 0000 auf 1111 aufwärtsgezählt.

B3: *Fortführung* Ist B3 logisch Null, wird der zuvor erwähnte Zähler nach einem Zählzyklus auf Null gesetzt. Für den Fall, daß B3 logisch Eins ist, wird der Zyklus in Abhängigkeit der anderen Bits fortgeführt.

Abb. 5.8 zeigt ergänzend die über verschiedene Bitkombinationen in R15 abrufbaren Hüllkurvenverläufe des AY-3-8912 zusammen mit den entsprechenden binären, dezimalen und hexadezimalen Codes.

Das Register 15 kann direkt von der BASIC-Ebene aus über eine Sonderform des ENV-Befehls mit den gewünschten Daten geladen werden

| | | Fortführung | Anstieg | Wechsel | Halten | Hüllkurvenverlauf |
|---|---|---|---|---|---|---|
| dezimal | hex. | B3 | B2 | B1 | B0 | |
| 0 bis 3 | 0 bis 3 | 0 | 0 | x | x |  |
| 4 bis 7 | 4 bis 7 | 0 | 1 | x | x |  |
| 8 | 8 | 1 | 0 | 0 | 0 |  |
| 9 | 9 | 1 | 0 | 0 | 0 |  |
| 10 | A | 1 | 0 | 1 | 0 |  |
| 11 | B | 1 | 0 | 1 | 1 |  |
| 12 | C | 1 | 1 | 0 | 0 |  |
| 13 | D | 1 | 1 | 0 | 1 |  |
| 14 | E | 1 | 1 | 1 | 0 |  |
| 15 | F | 1 | 1 | 1 | 1 |  |

*Abb. 5.8: Vorprogrammierte Hüllkurven und deren Verschlüsselung*

(siehe auch den folgenden Abschnitt über die Programmierung des PSG-Bausteins). Im Zusammenhang mit dem SOUND-Befehl ergibt sich damit eine Vielzahl von Klangformen und Klangfarben.

**Programmierung des PSG**

Die Programmierung des Soundgenerators erfolgt von der BASIC-Ebene aus über die Befehle SOUND, ENT, ENV, SQ, ON SQ(x) GOSUB und RELEASE, die je nach Befehlstyp und Parameterfolge die für die Klang- und Geräuscherzeugung richtigen Werte in die entsprechenden Register schreiben.

Wir wollen uns nachfolgend die unter BASIC verfügbaren Befehle ansehen und anhand kleiner Programme die Wirkungsweise einzelner Befehlsparameter kennenlernen.

Der wichtigste Befehl lautet

SOUND <Kanalkennung>,<Periodendauer>[,<Zeitdauer> [,<Lautstärke>[,<Nummer der Lautstärkeeinhüllenden> [,<Nummer der Frequenzmodulierenden> [,<Rauschperiode>]]]]]

Dieser zunächst recht kompliziert aussehende Befehl wird am besten in einzelnen Phasen erläutert.

Zunächst wollen wir einige nicht unbedingt notwendige Befehlsergänzungen weggelassen. Wir gelangen so zur Befehlsform

SOUND <Kanalkennung>,<Periodendauer>, <Zeitdauer>

Der Parameter „Kanalkennung" bezieht sich auf die Wirkung des zuvor bereits erläuterten Registers R7. Die Wirkung ist jedoch ein wenig anders, als Sie dies nach den Ausführungen zuvor erwarten dürfen. Die 6 niederwertigen Bits des Registers haben nämlich unter Interpreterkontrolle die nachfolgende Bedeutung:

| | | |
|---|---|---|
| B0: | Kanal A | entspricht dezimal 1 |
| B1: | Kanal B | entspricht dezimal 2 |
| B2: | Kanal C | entspricht dezimal 4 |
| B3: | Synchronisation mit Kanal A | entspricht dezimal 8 |
| B4: | Synchronisation mit Kanal B | entspricht dezimal 16 |
| B5: | Synchronisation mit Kanal C | entspricht dezimal 32 |
| B6: | Wert halten (HOLD) | entspricht dezimal 64 |
| B7: | Puffer leeren (FLUSH) | entspricht dezimal 128 |

Da gerade in diesem Zusammenhang leicht Mißverständnisse entstehen können, wollen wir den Befehl Schritt für Schritt untersuchen.

Zur Ausgabe eines Tons auf Kanal A mit einer Frequenz von 440 Hz (Kammerton a) und einer Dauer von 10 Sekunden lautet der SOUND-Befehl:

```
SOUND 1,142,100
```

Für eine Ausgabe auf Kanal 2 muß der Parameter „Kanalkennung" zu 2, für eine Ausgabe auf Kanal C zu 4 vereinbart werden. Bei einer gleichzeitigen Ausgabe auf mehr als einem Kanal ist aus den entsprechenden Parametern jeweils die Summe zu bilden, also A+B entspricht dezimal 1+2=3, A | C entspricht dezimal 1+4=5 usw.

```
SOUND 7,142,100
```

führt somit zu einer Ausgabe des Tons auf allen drei Kanälen für die vereinbarte Darbietungsdauer.

Was aber bedeuten die durch die Bits B3 bis B5 vereinbarten Synchronisationswerte? Um deren Wirkungsweise verstehen zu können, muß bekannt sein, daß die Hardware des Systems einen Soundpufferspeicher bereithält, der so ähnlich wie ein Stapel funktioniert. In diesen Puffer können „markierte" Anweisungen eingeschrieben werden, die nicht unmittelbar bei der Interpretation durch BASIC ausgeführt werden.

Ein kleines, aber recht anschauliches Beispiel möge diesen Sachverhalt etwas erhellen. Dazu ändern wir den zuvor als ersten angegebenen Befehl ein wenig ab:

```
SOUND 17,142,100
```

Geben Sie ihn zur Probe einmal über die Tastatur ein. Es wird nichts passieren. Die Kanalkombination 17 bedeutet: Ausgabe auf Kanal A (Bit B0) und Synchronisation mit B (Bit B4). Diese Information wird in den Soundpuffer geschrieben, aber nicht ausgeführt, da keine Anweisung für den Kanal B getroffen wurde. Erst wenn Sie beispielsweise zusätzlich den Befehl

```
SOUND 10, 500,100
```

eingeben (gib den durch den Parameter 500 gegebenen Ton auf Kanal B aus und synchronisiere mit Kanal A), werden die für A und B vereinbar-

ten Töne gemeinsam über den eingebauten Lautsprecher ausgegeben. Beachten Sie bitte, daß immer *beide* zu synchronisierenden Kanäle markiert werden müssen!

Die Bits B3 bis B5 markieren offensichtlich die einzelnen Kanäle, d.h. sie sorgen dafür, daß die Informationen bis zum Vorliegen der Synchronisationsbedingung abrufbar im Pufferspeicher abgelegt werden.

Sie sollten noch ein weiteres Beispiel ausprobieren:

```
SOUND 33,478,2000    Anm.: Ausgabe auf Kanal A und Synchronisation mit C
SOUND 34,319,2000    Anm.: Ausgabe auf Kanal B und Synchronisation mit C
SOUND 28,284,2000    Anm.: Ausgabe auf Kanal C und Synchronisation mit A und B
```

Erst bei Eingabe des letzten der drei SOUND-Befehle wird die Tonausgabe freigegeben.

Um die Bits B6 und B7 werden wir uns noch am Ende dieses Kapitels kümmern. Zunächst schauen wir uns noch die übrigen Parameter des SOUND-Befehls an.

Der Parameter *Periodendauer* wird nach der Beziehung

$$d = 1000000/(16*f)$$

bzw. die erzielte Frequenz durch Umkehrung dieser Beziehung

$$f = 1000000/(16*d)$$

berechnet.

Mit f ist die Frequenz des Tons gemeint. Im Anhang O finden Sie eine vollständige Tabelle für die vom PSG erzeugbaren 8 Oktaven. Aus technischen Gründen werden keine harmonischen Schwingungen, sondern Rechteckschwingungen erzeugt, die ein erheblich umfangreicheres Obertonspektrum besitzen. Der Parameter kann formal Werte zwischen d = 0 und d= 4095 annehmen. Das führt allerdings im unteren wie auch im äußerst oberen Wertebereich zu keinen hörbaren Tönen mehr. Der Wert d = 0 ist nur dann sinnvoll, wenn im letzten Parameter der Rauschgenerator aktiviert wird.

Beispiel:

```
SOUND 1,0,1000,7,0,0,2
```

führt zu einem hellgefärbten Rauschen von 10 Sekunden Dauer auf Kanal A.

Der Parameter *Zeitdauer* wird in Vielfachen von 1/100 Sekunden angegeben. Für eine Sekunde Tonausgabe lautet der Parameter somit 100, da 100*1/100 = 1 ist. Wird der Parameter nicht vereinbart, nimmt der Interpreter 20 als Standardwert (also 0,2 s) an.

Als nächster Parameter im SOUND-Befehl folgt die *Lautstärke*. Der Parameterwert wird in eines der Register R10 bis R12 eingeschrieben, in denen – wie Sie bereits wissen – nur die unteren Bits eine Bedeutung haben. Unter Kontrolle von BASIC können Sie ohne Vereinbarung einer Einhüllenden konstante Lautstärkewerte im Bereich zwischen 0 und 7 bzw. mit Einhüllender zwischen 8 und 15 angeben. Die Angabe von Werten zwischen 8 und 15 ohne zusätzliche Freigabe einer Einhüllenden mittels des ENV Befehls, führt zu einer Wiederholung der Werte des unteren Bereichs.

Beispiel:

```
10 FOR L%=0 TO 15
20 SOUND 1,100,50,L%
30 NEXT L%
```

### *Nummer der Einhüllenden*

Über die beiden folgenden Parameter *„Nummer der Lautstärkeeinhüllenden"* und *„Nummer der Frequenzmodulierenden"* werden Lautstärke- bzw. Frequenzveränderungen (Amplituden- und Frequenzmodulationen) abgerufen, die über die beiden Befehle ENV und ENT programmiert werden. Diese beiden Befehle enthalten als ersten Parameter eine Identitätsnummer, die jeweils im SOUND-Befehl angegeben werden muß. Zulässig sind abgesehen vom Wert 0 (Einhüllende ausgeschaltet) die Werte 1 bis 15.

Einhüllende Funktionen der Lautstärke sind von ihrem Zeitverhalten her gesehen in unterschiedliche Bereiche einteilbar, die jedem Freund elektronischer Musik geläufig sind. Der Toneinsatz wird beispielsweise durch die Anstiegsgeschwindigkeit (attack) bestimmt. Es kann dann eine Phase konstanter Lautstärke oder nur leicht abnehmender Lautststärke folgen, die als stationäre Phase oder auch Haltephase (sustain bzw. hold)

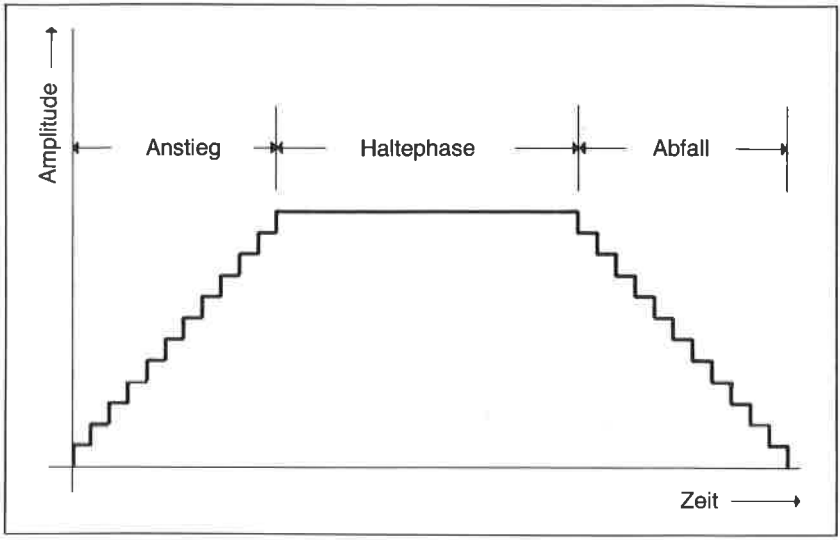



*Abb. 5.9: Typische Phasen einer einhüllenden Funktion*

bezeichnet wird. Irgendwann folgt dann ein Ausklingvorgang (decay), der mehr oder minder rasch ablaufen kann (Abb. 5.9). Die relativen Zeitverhältnisse sowie die Art des Funktionsverlaufs in den für die musikalische Empfindung wichtigen Ein- und Ausklingbereichen der Hüllkurve können die Qualität der Tonerzeugung wesentlich beeinflussen. Durch einen kurzen und steilen Anstieg sowie ein anschließendes exponentiell abklingendes Hüllkurvenverhalten wird beispielsweise sehr gut der typische Klangcharakter eines Cembalos bzw. eines Spinetts simuliert.

Obgleich die für die Hüllkurvenfestlegung wichtige ENV-Anweisung erst im nächsten Abschnitt behandelt wird, sollten Sie das nachfolgende kleine Programm eingeben und starten. Die Eingabe erfolgt über die Zifferntastatur (Ziffern 1 bis 9). Auch hier wird durch die Kanalsynchronisation ein präziser Einsatz erzielt.

```
5 REM **** Testprogramm ****
10 A$=INKEY$:IF A$="" GOTO 10
20 D%=VAL(A$)
30 ENV 1,=0,2000
40 SOUND 33,D%*20,30,15,1
50 SOUND 34,D%*10,30,15,1
60 SOUND 28,D%*5,30,15,1
70 GOTO 10
```

### *Die Wiederholperiode des Rauschvorgangs*

Der letzte Parameter bestimmt die Wiederholperiode des binären Pseudorauschens. Physikalisch gesehen ist ein idealer Rauschvorgang völlig regellos. Er besitzt keinerlei periodische Anteile. In einem digitalen Rauschgenerator wird im allgemeinen ein „regelloses" Signal dadurch erzeugt, daß über ein rückgekoppeltes Schieberegister eine fast willkürliche Folge von Impulsen abgerufen wird. Aus technischen Gründen ist dies auf einfache Weise nicht ideal möglich. Jede Impulsfolge kehrt irgendwann einmal wieder. Diese Periodendauer kann über den letzten Parameter mit der Bezeichnung *Rauschperiode* eingestellt werden. Akustisch macht sich die veränderte Wiederholperiode durch eine Klangverfärbung des Rauschens bemerkbar. Versuchen Sie es einmal mit dem nachfolgenden Beispiel:

Programmbeispiel:

```
5 REM **** elektronische Dampflok ****
10 ENV 1,=4,400
20 FOR Z%=30 TO 15 STEP -1
30 GOSUB 90
40 NEXT Z%
50 GOSUB 60:GOTO 50
60 SOUND 2,0,Z%,15,1,0,8
70 SOUND 2,0,Z%,15,1,0,1
80 SOUND 2,0,Z%,15,1,0,1
90 SOUND 2,0,Z%,15,1,0,1
100 RETURN
```

Sie sollten mit dem SOUND-Befehl ein wenig experimentieren, um ein Gefühl für seine Arbeitsweise zu erhalten. Das zuvor angegebene Programmbeispiel zeigt jedoch, daß offensichtlich erst durch Hinzunahme der Befehle ENV bzw. ENT eine faszinierendere Klangwelt eröffnet wird.

Wir wollen uns die beiden zuletzt erwähnten Anweisungen einmal etwas näher ansehen und einige Experimente damit durchführen.

### Die Lautstärkevariation mittels der ENV-Anweisung

Der BASIC-Befehl ENV (*EN*velope *V*olume) dient zur Beeinflussung des zeitlichen Verlaufs der Laustärke eines durch SOUND definierten Tons. Er ist auf zweierlei Art vereinbar:

In der ersten Form lautet der ENV-Befehl:

```
ENV <Hüllkurvennummer>,<Schrittzahl1>,<Schritthöhe1>,
<Schrittweite1>[, <Schrittzahl2>,<Schritthöhe2>,
<Schrittweite2>] [,....][,....][,....]
```

Unter dem Parameter *Hüllkurvennummer* wird eine Zahl zwischen 1 und 15 vereinbart, die als reine Kennung dient. Sie wird im SOUND-Befehl an der entsprechenden Stelle aufgerufen. Es sind folglich bis zu 15 verschiedene Hüllkurvenvereinbarungen möglich, die unter Angabe der jeweiligen Kennummer von verschiedenen SOUND-Befehlen aufgerufen werden können.

Jede Hüllkurve kann aus bis zu fünf voneinander verschiedenen Hullkurvenabschnitten bestehen, innerhalb derer jeweils die nachfolgend noch erläuterten Parameter Schrittzahl, Schritthöhe und Schrittweite individuell vereinbart werden können. Sie sollten sich zu diesem Zweck die Abb. 5.10 ansehen, in der als Beispiel die Konstruktion einer Hüllkurve aus fünf Abschnitten A bis E gezeigt ist. Bei der Vereinbarung der einzelnen

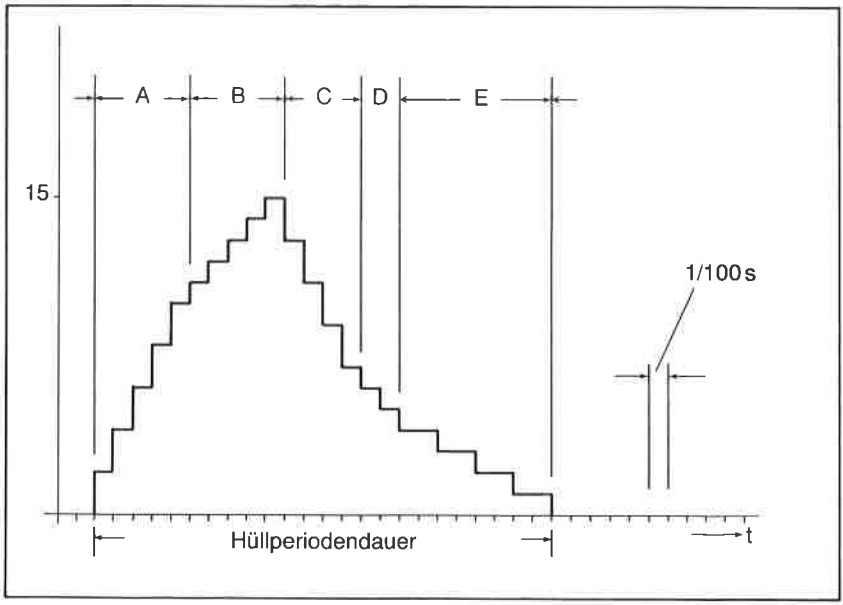



*Abb. 5.10: Konstruktion einer Einhüllenden aus fünf verschiedenen Abschnitten A bis E mit unterschiedlicher Schrittzahl N, Schritthöhe dN und Schrittweite dT*

Hüllkurvenparameter sind einige grundsätzliche Dinge zu beachten, die wir uns nachfolgend anhand einiger erläuternder Beispiele ansehen werden.

Die *Schrittweite dT* wird immer in Vielfachen von 1/100 Sekunden angegeben. Erlaubt sind laut Handbuch für diesen Parameter Werte zwischen 0 und 255, entsprechend 0 bis 2,55 Sekunden. Die Gesamtzahl aller zu einer Hüllkurvenvereinbarung gehörenden Schritte, multipliziert mit der jeweils gültigen Schrittweite führt zur Gesamtdauer der Hüllkurvenperiode oder – falls keine Wiederholung der Funktion vereinbart wird – zur Gesamtdauer der einhüllenden Funktion. Die in Abb. 5.11 definierte Hüllkurve besteht aus 24 Intervallen zu je 0,01 s. Zwischen dem Einsatz und dem Abklingen liegen folglich 0,24 Sekunden. Beachten Sie bitte, daß die im SOUND-Befehl angegebene Dauer für den Ton oder das Geräusch bestimmend ist. Ist die Hüllperiodendauer größer, wird nach Maßgabe des SOUND-Befehls die Ausgabe abgebrochen. Im umgekehrten Fall wird der nach Beendigung der Hüllkurvenfunktion erreichte Lautstärkewert bis zu der im SOUND-Befehl vereinbarten Gesamtdauer weitergeführt.

Innerhalb eines Abschnitts ist ausschließlich ein linear ansteigender oder abfallender Amplitudenverlauf möglich, der nur in kleinen Amplitudenintervallschritten dA erfolgen kann. Diese Intervalle sind über den Parameter *Schritthöhe dA* zu vereinbaren. Sie darf laut Handbuch im Bereich zwischen −128 und + 127 liegen. Das Vorzeichen bestimmt hierbei die Richtung der Schritte (ansteigende oder abnehmende Amplitude). Nichtlineare Hüllkurvenverläufe sind nur über unterschiedliche Vereinbarungen in den einzelnen Abschnitten zu erzielen.

Die Anzahl von Amplitudenintervallschritten je Hüllkurvenabschnitt kann über den Parameter *Schrittzahl N* festgesetzt werden. Formal möglich sind Werte zwischen 0 und 127. Zu beachten ist aber, daß die Lautstärkevariation von Digital-Analog-Umsetzern gesteuert wird, die eine Auflösung von nur 4 Bit aufweisen. Das bedeutet, daß Amplitudenintervallschritte nur in 16er-Schritten möglich sind ( 0 bis 15). Ausgangswert für die Lautstärke ist dabei jener Wert, der im SOUND-Befehl festgesetzt wurde. Im Falle der Einhüllenden nach Abb. 5.11 ist demnach die Vereinbarung der Anfangslautstärke 0 in der zugehörigen SOUND-Anweisung vernünftig.

Achtung! Überschreitet der durch den Anfangswert und durch das Produkt aus N*dA gegebene maximale Wert in einem der Hüllkurvenabschnitte den Wert 15, ergeben sich recht überraschende und wenig kon-

| | Schritt-zahl N | Schritt-höhe dA | Schritt-weite dT | Gesamt-dauer T | Endwert Lautst. |
|---|---|---|---|---|---|
| Abschnitt A | 5 | 2 | 1 | 0,05 | 10 |
| Abschnitt B | 5 | 1 | 1 | 0,05 | 15 |
| Abschnitt C | 4 | −2 | 1 | 0,04 | 7 |
| Abschnitt D | 3 | −1 | 1 | 0,03 | 4 |
| Abschnitt E | 4 | −1 | 2 | 0,08 | 0 |

*Abb. 5.11: Lautstärke- und Zeitbilanz der Einhüllenden nach Abb. 5.10 bei Annahme, daß im SOUND-Befehl der Lautstärkeparameter zu Null vereinbart wird*

trollierbare Lautstärkesprünge. Dieselbe Aussage gilt, wenn bei negativen Intervallschritten der Lautstärkewert 0 unterschritten wird. Für die in Abb. 5.11 gezeigte Hüllkurve gilt die ENV-Vereinbarung:

```
ENV 1, 5,2,1,5,1,1,4,−2,1,3,−1,1,4,−1,2
```

Für die einzelnen Abschnitte ergeben sich Lautstärke- und Zeitbilanzen, wie in Abb. 5.11 dargestellt.

Die angegebenen Gesamtzeiten T für die einzelnen Abschnitte errechnen sich aus dem Produkt der Schrittzahl und der Schrittweite. Der Lautstärke-Endwert innerhalb eines Abschnitts kann aus der Anfangslautstärke und dem Produkt von Schrittzahl und Schritthöhe ermittelt werden. Da im SOUND-Befehl die Anfangslautstärke zu Null vereinbart ist, klettert sie im ersten Intervall A auf den Wert 0 + 5∗2 =10, im zweiten Intervall B auf 10 + 5∗1 =15. Im dritten Abschnitt C sinkt sie auf den Wert 15 + 4∗(−2)=7 usw.

In der zweiten Form lautet der ENV-Befehl:

```
ENV <Hüllkurvennummer>,=<Hüllkurventyp>,
<Hüllperiodendauer>
```

Mit seiner Hilfe können die bereits in der Abb. 5.8 angegebenen Hüllkurvenverläufe abgerufen werden. Jede der Hüllkurven setzt sich aus Grundzyklen zusammen, deren Zeitdauer über den Parameter „Hüllperiodendauer" in Vielfachen von 0,01 Sekunden einstellbar ist (Abb. 5.12). Schauen Sie sich in diesem Zusammenhang noch einmal das Beispiel TESTPROGRAMM an.

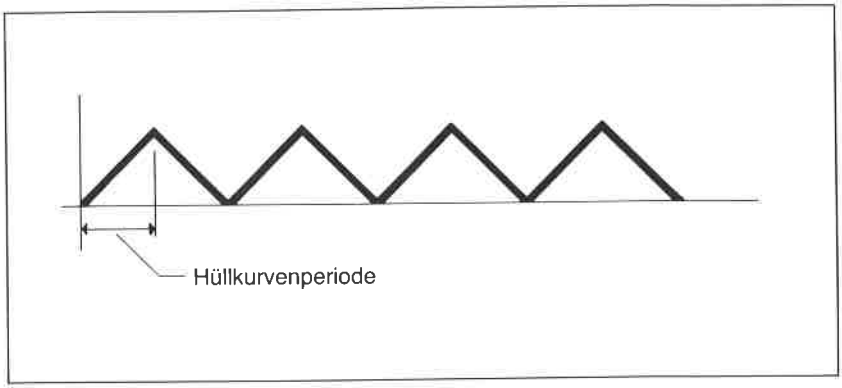



*Abb. 5.12: Definition der Hüllkurvenperiode*

### Die Tonhöhenvariation mittels der ENT-Anweisung

Über den Befehl ENT (*EN*velope *T*one) kann die Tonhöhe in den drei Kanälen A, B und C einer zeitlichen Variation unterworfen werden. In der Musik sind leichte Tonhöhenvariationen unter dem Begriff *Vibrato* geläufig. Physikalisch gesehen handelt es sich um eine Frequenzmodulation, deren Verlauf durch die im ENT-Befehl festgesetzte einhüllende Funktion (die modulierende Funktion) bestimmt wird. Wie die ENV-Anweisung kann auch die ENT-Anweisung in zwei unterschiedlichen Formen vereinbart werden. In der ersten Form lautet sie:

```
ENT <Hüllkurvennummer>,<Schrittzahl N1>,<Schritthöhe dA1>,
<Schrittweite dT1>[,<Schrittzahl N2>,<Schritthöhe dA2>,
<Schrittweite dT2>][,...][,....][,....]
```

Unter dem Parameter *Hüllkurvennummer* wird eine Kennummer zwischen −15 und +15 vereinbart, die durch den 6. Parameter in einem SOUND-Befehl aufgerufen wird. Angabe eines negativen Vorzeichens führt zu einer fortwährenden Wiederholung der in der ENT-Anweisung definierten Tonhöhenvariation bis zum Ende der im SOUND-Befehl angegebenen Darbietungszeit (Parameter Nr. 3). Beachten Sie, daß die entsprechende Kennummer im SOUND-Befehl kein negatives Vorzeichen haben darf. Die *Schrittzahl* kann zwischen 0 und 239 und die *Schritthöhe* zwischen −128 und +127 (je nach Laufrichtung) und die *Schrittweite* im Bereich zwischen 0 und 255 * 0,01 Sekunden liegen.

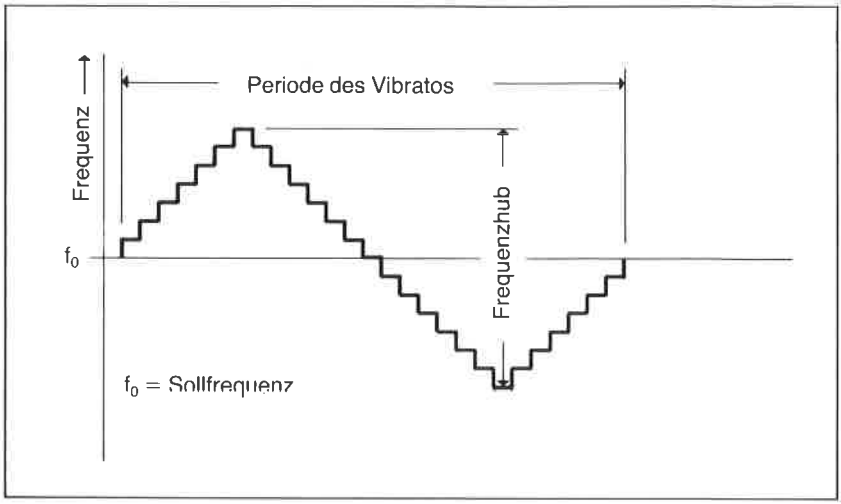



*Abb. 5.13: Konstruktion eines schnellen Vibratos*

Beispiel:

```
10 ENT -12,2,3,1,4,-3,1,2,3,1
20 SOUND 1,145,1000,15,0,12
```

In diesem Beispiel läuft die Signalperiode zwischen den Grenzwerten 145+2*3=151, entsprechend einer Frequenz von 413,9, und 145−2*3=139, entsprechend einer Frequenz von 449,6. Der Frequenzhub (so nennt man die Differenz zwischen maximaler und minimaler Tonhöhe) beträgt somit etwa 35 Hz.

Auch hier ist darauf zu achten, daß die durch das Produkt aus Schrittzahl und Schritthöhe gegebene Signalperiode nicht die zulässigen Tonhöhenbereiche überschreitet.

### Unterbrechungsgesteuerte Tonerzeugung

Vermutlich wird es Ihnen schon häufiger bei Ihren Klangexperimenten aufgefallen sein, daß Ihr Schneider CPC nach der Eingabe von Ton- und

Geräuschbefehlen bereits vor Beendigung der Ausgabe über den Lautsprecher wieder in den Befehlsmodus zurückkehrt. Das liegt daran, daß der Computer die für die Ansteuerung des Sound-Chips benötigten Informationen unmittelbar an die Hardware weiterleitet und sich selbst anschließend für andere Aufgaben freischaltet. Sie können so beispielsweise bereits Programmzeilen durch Editieren verändern, während der Generator noch mit der Ausgabe der vereinbarten Befehle beschäftigt ist. Sämtliche Anweisungen an den PSG werden vom System in einem Pufferspeicher abgelegt und Schritt für Schritt abgearbeitet. Da bis zu vier Anweisungen im Puffer abgelegt werden können, kann das System sofort andere Aufgaben wahrnehmen, wenn die letzten vier Anweisungen zwischengepuffert sind. Das Ganze funktioniert nach einem sogenannten „Warteschlangenprinzip". Sobald der Generator mit der Ausführung einer aktuellen Anweisung fertig ist, holt er sich die nächstfolgende Anweisung aus dem Puffer. Nur während dieser kurzen Zeit beansprucht er die Aufmerksamkeit des Prozessors für sich. In der übrigen Zeit kann das System vielerlei Dinge tun. Beispielsweise könnte auch ein Programm ablaufen.

Falls Sie also den dringenden Wunsch verspüren, ein laufendes Programm mit Musik zu hinterlegen: Nichts leichter als das! Sie bedienen sich dazu eines speziellen Unterbrechungsbefehls, der so ähnlich funktioniert wie die Anweisungen EVERY , AFTER oder REMAIN. Dieser Befehl lautet

ON SQ(<Kanalnummer>) GOSUB <Zeilennummer>

Die Funktionsweise ist einfach: Schreiben Sie ein (Unter-)Programm, das die Informationen für den PSG enthält, also die gewünschte Melodie oder das gewünschte Geräusch. Vereinbaren Sie in Ihrem Hauptprogramm, gleichgültig, was es auch tut, den zuvor vorgestellten Befehl. Nach dem Start wird dann, wenn ein Platz im Soundpuffer eines Kanals für das Nachladen einer neuen Anweisung an den PSG frei wird, ein Unterbrechungssignal erzeugt. Dieses veranlaßt den Computer, das laufende Hauptprogramm kurz zu verlassen, um die Warteschlange wieder aufzufüllen. Das geht so schnell, daß Sie in der Regel die kurze Unterbrechung gar nicht bemerken werden. Weil der Befehl nicht automatisch wiederholt wird, müssen Sie unmittelbar vor dem RETURN-Statement den ON SQ-Befehl wiederholen, falls Sie den Puffer erneut nachladen möchten. Die Struktur eines Programms mit der ON SQ-Anweisung sieht dann wie folgt aus:

```
5 REM **** Hauptprogramm ****
10 ON SQ(1) GOSUB 1000
20 ...
30 ...
999 END
1000 REM **** Unterprogramm zur Tonerzeugung ****
1010 ...
1020 ...
1099 ON SQ(1) GOSUB 1000
1100 RETURN
```

Im Argument von SQ muß der Kanal angegeben werden, dessen Puffer nachgeladen werden soll. Für Kanal A ist dies der Wert 1, für B der Wert 2 und für C der Wert 4. Nachfolgend ist ein Beispiel aufgeführt, das die Wirkung dieses Unterbrechungsbefehls zeigt.

Beispielprogramm:

```
5 REM **** Musik simultan mit Programm ****
10 ON SQ(1) GOSUB 70
20 FOR I=1 TO 50000
30 PRINT I;" ";
40 NEXT I
50 END
60 DATA 239,225,213,201,190,179,169,159,150,142,134,127
,00
70 READ I: IF I=0 THEN RESTORE
80 SOUND 1,I,10
90 ON SQ(1) GOSUB 70
100 RETURN
```

Unmittelbar mit dem ON SQ-Befehl verknüpft ist die Funktion

SQ(<Kanal>)

Mit ihrer Hilfe läßt sich eine Information über den aktuellen Zustand des Soundpuffers gewinnen. Die einzelnen Bits dieser Statusinformation haben die in der Abb. 5.14 gezeigte Bedeutung. An das Programm übergeben wird die dezimale Verschlüsselung des dualen Datenwortes.

Geben Sie unmittelbar über die Tastatur die nachfolgenden Befehle ein:

```
SOUND 1,142,5000
SOUND 1,300,5000
SOUND 1,400,5000
PRINT SQ(1)
```

Auf dem Bildschirm wird die Zahl 130 ausgegeben, die sich aus 128 + 2 zusammensetzt. Das heißt: Zur Zeit wird ein Ton auf Kanal A (128) ausgegeben. Es sind noch 2 Plätze im Puffer für A frei.

### Spezialbefehle

Zum Schluß dieses Kapitels wollen wir uns noch mit den bisher etwas vernachlässigten Werten für den Parameter „Kanalkombination" im SOUND-Befehl beschäftigen, die durch die Bits B6 (HOLD = Haltezustand) und B7 (FLUSH = Zwangsabbruch) bestimmt werden. Den bisherigen Ausführungen zufolge wird offensichtlich bei der Ausgabe von Musik und Geräuschen über den PSG immer der Puffer in serieller Form abgearbeitet und gegebenenfalls aus dem Programm nachgeladen. Wird dagegen im Parameter Kanalkombination des SOUND-Befehls zusätzlich noch das Bit 7 (mit dem Gewicht 128) gesetzt, wird der angesprochene Puffer unmittelbar geleert und das aktuelle Tonereignis ausgegeben. Für eine Zwangsunterbrechung der Hintergrundmusik bei Bildschirmspielen kann dies recht nützlich sein.

Auch in diesem Zusammenhang ist ein Beispiel besser als viele Worte. Geben Sie zunächst einfach den Befehl

```
SOUND 1,125,5000
```

ein. Wie Sie wissen wird dann ein Dauerton von 50 Sekunden Länge ausgegeben. Wenn Sie nun anschließend

```
SOUND 1,250,5000
```

eintippen, wird erwartungsgemäß unmittelbar nichts passieren, weil der zweite Befehl sich in die Warteschlange des Puffers einordnet und erst nach Beendigung des ersten ausgeführt wird. Geben Sie statt dessen jedoch

```
SOUND 129,250,5000
```

ein, dann wird die aktuelle Tonausgabe sofort abgebrochen, der Puffer wird geleert und der zuletzt eingegebene Befehl ausgeführt.

Bit 6 mit dem Wert 64 wird dagegen als Haltebit bezeichnet. Ein entsprechend markierter Befehl führt dazu, daß das Tonereignis so lange im Puffer gehalten wird, bis es ausdrücklich freigegeben wird. Hierzu dient der BASIC-Befehl

```
RELEASE <Kanalnummer>
```

Soviel zum Thema Ton- und Geräuscherzeugung. Im nächsten Kapitel wollen wir uns ein wenig mit der Technik des CPC befassen, damit die in Kapitel 7 folgende Einführung in die Maschinenprogrammierung etwas leichter fällt.

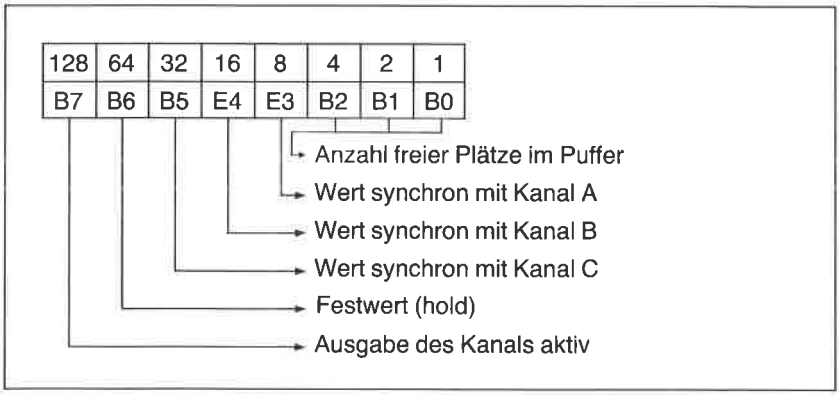



*Abb. 5.14: Bedeutung der Statusbits des Soundpuffers*

Kapitel 6

# Einführung in die Technik des CPC 464/664

## Übersicht

Wenn Ihr Interesse über das Arbeiten mit der Programmiersprache BASIC hinausgeht, sei es, weil Sie die Geschwindigkeit durch Maschinenroutinen erhöhen oder die von der Hardwareseite gegebenen Möglichkeiten voll ausnutzen wollen, dann benötigen Sie fundierte Kenntnisse über den Aufbau Ihres Computersystems und die Funktionsweise der mikroelektronischen Komponenten. Natürlich handelt es sich dabei um einen im einzelnen recht komplizierten Themenkreis, der in einem Buch wie dem vorliegenden nicht erschöpfend zu behandeln ist. In diesem Kapitel erhalten Sie all jene Informationen, die Ihnen den Einstieg in die Welt der Mikrocomputertechnik erleichtern und ein erfolgreiches Weiterarbeiten mit spezieller Literatur ermöglichen.

Sie werden zunächst ein wenig über die Hardwarearchitektur des Computers und seines wichtigsten Baulementes, des Mikroprozessors Z80, lesen. Sie lernen, wie der Prozessor Daten verarbeitet und mit welchen peripheren Bauelementen er zusammenarbeitet. Damit Sie mit binären Daten in den unterschiedlichen Darstellungsarten umzugehen lernen, werden Sie außerdem einiges über den Aufbau von Datenworten sowie den Umgang mit ihnen lesen. Ein Abschnitt über die Speicherorganisation des CPC 464/664 sowie die Funktionsweise einiger wichtiger Ein-/Ausgabebausteine rundet das Kapitel ab.

## Der Aufbau eines Mikrocomputersystems

Wir wollen uns zunächst mit dem internen Aufbau eines Mikrocomputers anhand eines vereinfachten grafischen Schemas vertraut machen. Schauen Sie sich hierzu einmal die Abb. 6.1 an, in der die wichtigsten Funktionselemente eines Mikrocomputers skizziert sind.

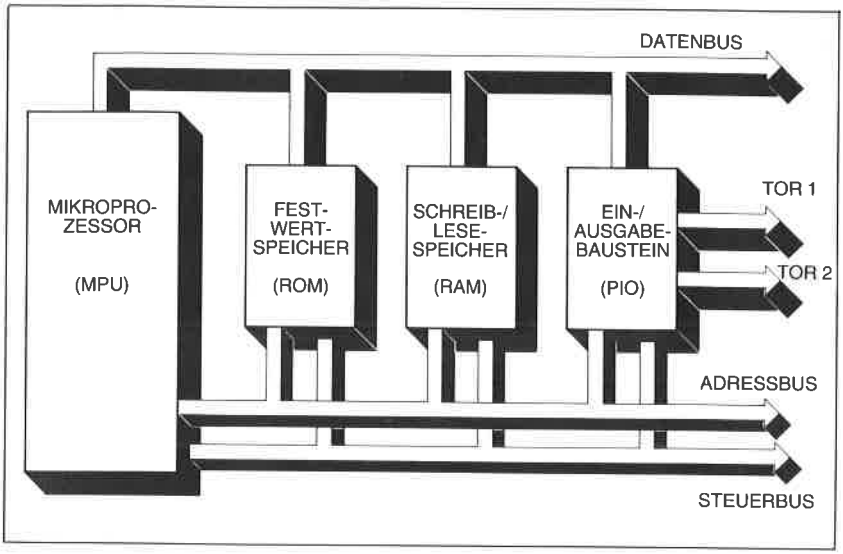



*Abb. 6.1: Blockschaltbild eines Mikrocomputers*

Das wichtigste Bauelement ist der Mikroprozessor. Er besteht aus vielen Tausend Halbleiterbauelementen. Diese sind alle zusammen auf einer winzig kleinen Halbleiterfläche untergebracht oder – wie man in der Fachsprache sagt – integriert. Eine derartig komplexe elektronische Struktur wird als integriertes Bauelement bezeichnet. Den inneren Aufbau des Mikroprozessors werden Sie gleich noch kennenlernen.

Der Mikroprozessor, abkürzend auch MPU (engl.: *M*icro *P*rocessing *U*nit) genannt, führt logische und mathematische Operationen mit binären Datenwerten durch. Der Schneider CPC ist um einen Mikroprozessor des Typs Z80 herum aufgebaut. Dieser Prozessor wurde von der Firma Zilog entwickelt und ist als Nachfolger des 8080 der Firma INTEL in viele Millionen von Computersystemen eingebaut worden. Für seine Arbeit benötigt der Z80, wie alle anderen Mikroprozessoren auch, eine Reihe externer Baugruppen und Funktionselemente, die ebenfalls in Abb. 6.1 angegeben sind. Neben den Speicherbausteinen (RAM und ROM) zählen dazu noch Ein-/Ausgabekomponenten, die den Datenverkehr des Computers mit peripheren, d. h. außen an das System angeschlossenen Geräten ermöglichen. Jeder Computer besitzt zwei unterschiedliche Arten von Speicherbausteinen. Der sogenannte *Festwertspeicher* enthält jene Daten und Programme, die bereits beim Einschalten des Computers

zur Verfügung stehen müssen. Im Bereich der Computertechnik sind folgende Arten von Festwertspeichern gebräuchlich:

ROM (*R*ead *O*nly *M*emory): Festwertspeicher, der bereits bei der Herstellung maskenprogrammiert wurde. Die eingeschriebenen Informationen sind durch den Anwender nicht mehr veränderbar.

PROM (*P*rogrammable *R*ead *O*nly *M*emory): Festwertspeicher, in den vom Anwender einmal mittels eines speziellen Programmiergerätes (Prommer genannt) Informationen eingeschrieben werden können. Der Schreibvorgang kann je nach Speicherumfang bis zu mehreren Minuten dauern.

EPROM (*E*rasable *P*rogrammable *R*ead *O*nly *M*emory): PROM, dessen Inhalt durch langzeitiges Bestrahlen mit ultraviolettem Licht wieder gelöscht werden kann. Im Gehäuse ist zu diesem Zweck ein Acrylglasfenster eingelassen, das nach dem Löschvorgang möglichst durch eine metallisierte selbstklebende Folie verdeckt werden sollte, um versehentliches Löschen von Informationen durch harte Strahlanteile im Licht zu verhindern.

Der CPC besitzt in der Grundausstattung einen Festwertspeicher von 32 Kilobyte Größe. Er enthält die Programme des Betriebssystems, die Zeichensätze für die Tastatur sowie das Übersetzerprogramm für die Programmiersprache BASIC (BASIC-Interpreter). Da die Informationen durch den Prozessor nur ausgelesen werden können, ist diese Art von Speichern für die Zwischenabspeicherung von Anwenderprogrammen und Daten nicht geeignet. Diese werden in Speicherbausteinen abgelegt, die als Schreib-/Lesespeicher bezeichnet werden. In der Computerfachsprache ist die Abkürzung RAM (engl.: *R*andom *A*ccess *M*emory) gebräuchlich. Das bedeutet: Speicher mit wahlfreiem Zugriff. Gemeint ist hier die Art des Zugriffs durch einen Lese- oder Schreibvorgang. Schreib-/Lesespeicher verlieren ihre Information im Gegensatz zu den Festwertspeichern, wenn die Spannungsversorgung abgeschaltet wird. Denken Sie bitte daran, wenn Sie Programme entwickeln oder mittels Anwendersoftware eigene Daten verarbeiten. Ein Stromausfall zerstört alle im RAM des Rechners gespeicherten Informationen. Programme wie auch Daten sollten daher in regelmäßigen Abständen auf einem externen Massenspeicher (Kassette oder Diskette) gesichert werden.

Alle am Computersystem angeschlossenen elektronischen Komponenten (Sichtgerät, Kassettenrecorder, Floppy-Disk-Laufwerke, Drucker usw.) können aus technischen Gründen nicht direkt vom Prozessor versorgt

werden. Jedes Computersystem besitzt daher sogenannte *Schnittstellen* (auch *Interfaces* genannt), die die Signale im Computer an die Erfordernisse des peripheren Gerätes anpassen. Die Versorgung derartiger Schnittstellen wird im allgemeinen durch spezielle, hochintegrierte Bausteine sichergestellt, die je nach Funktionsweise als PIO (*P*eripheral *I*nput/*O*utput), VIA (*V*ersatile *I*nterface *A*dapter) oder ähnlich bezeichnet werden. In der Skizze in Abb. 6.1 sind sie unter der Bezeichnung Ein-/Ausgabe zusammengefaßt.

Alle elektronischen Komponenten sind mit dem Mikroprozessor und/oder untereinander über elektrische Leitungsbündel verbunden. Je nach Aufgabenbereich besitzen sie unterschiedliche Bezeichnungen. Jene Leitungen, auf denen die elektrischen Signale für die Speicheradressen anliegen, werden als *Adreßbus* bezeichnet. Mit ihrer Hilfe werden gezielt Speicherzellen aus der Gesamtzahl aller Speicherelemente ausgewählt und angesprochen. Der Datenaustausch zwischen den Speicherelementen oder peripheren Bausteinen dagegen findet über ein anderes Leitungsbündel statt, das als *Datenbus* bezeichnet wird. Zusätzlich wird noch eine weitere Gruppe von Leitungen benötigt, über die die Synchronisations- und Steuersignale übertragen werden. Sie gehören zum Steuer- oder auch *Kontrollbus*.

Da alle Vorgänge in einem Computersystem unter Kontrolle einer inneren Uhr ablaufen, besitzt der Prozessor noch einen „Herzschrittmacher" in Form einer Quarzuhr (engl. Clock). Sie erzeugt die zum Betrieb benötigten Impulse mit einer Taktrate von 4 MHz. Dies entspricht 4 Millionen Schwingungen pro Sekunde.

### Ein Blick in das Innere des Z80

Um die Funktionsweise eines Mikrocomputersystems verstehen zu können, müssen Sie wissen, wie der Mikroprozessor arbeitet. Schauen Sie sich zu diesem Zweck einmal das Schema in Abb. 6.2 an, das die wichtigsten Elemente des Mikroprozessors Z80 zeigt. Er besitzt als auffälligstes Funktionselement ein Rechenwerk, das als ALU (engl. *A*rithmetic and *L*ogical *U*nit) bezeichnet wird. Seine Aufgabe besteht dem Namen entsprechend darin, arithmetische und logische Operationen mit den Daten durchzuführen. Die elementarste *arithmetische* Operation ist die Addition von Zahlen. Alle weiteren Grundrechenarten wie auch Rechenoperationen höherer Ordnung werden auf diese Grundoperation zurückgeführt. Zu den *logischen* Operationen zählen Datenvergleiche sowie Verknüpfungen mittels ODER-, EXKLUSIV ODER-, UND- oder ähnlichen Funktionen.

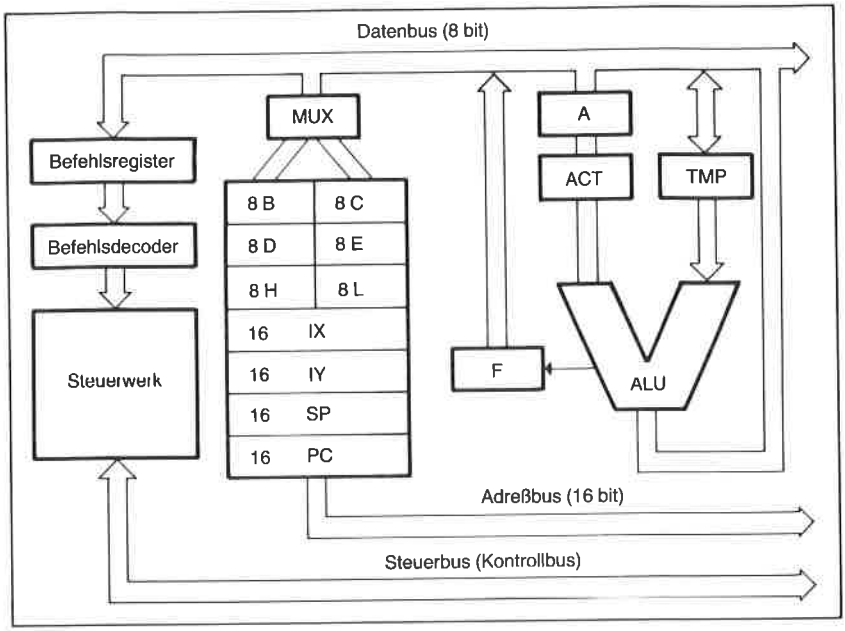



*Abb. 6.2: Blockschaltbild des Mikroprozessors Z 80*

Die ALU bedient sich zur Durchführung derartiger logischer oder arithmetischer Vorgänge unmittelbar- oder auch nur mittelbar sehr schnell arbeitender Speicher, die als *Register* bezeichnet werden. Wie Sie der Abbildung 6.3 entnehmen können, die eine Übersicht über die Registerstruktur des Z80 zeigt, stehen insgesamt 22 Register zur Verfügung.

Die gestrichelt gekennzeichneten Register wollen wir aus unserer Betrachtung ausklammern, weil sie dieselben Aufgaben erfüllen und dieselben Eigenschaften besitzen wie die nicht gekennzeichneten Speicher, und weil der Prozessor immer nur mit einer dieser Gruppen gleichzeitig arbeiten kann.

Der ALU zugeordnet sind direkt nur die mit A und mit F bezeichneten Register. Das A-Register wird als *Akkumulatorregister* (manchmal auch kurz nur als Akku) bezeichnet (akkumulieren heißt anhäufen). Diese Bezeichnung ist darauf zurückzuführen, daß dieses Register als zentrales Rechenregister nicht nur die für eine Operation benötigten Daten speichert, sondern auch das Ergebnis einer Berechnung oder eines logischen Vergleichs aufnimmt.

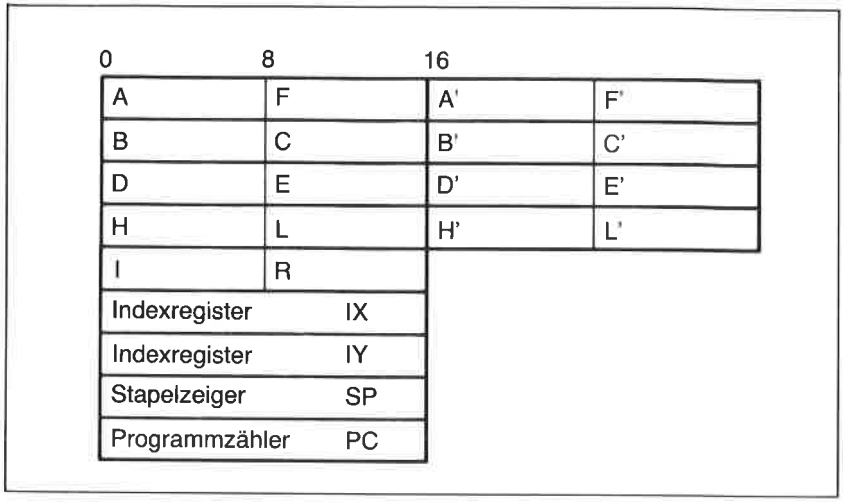



*Abb. 6.3: Die Registerstruktur des Z80*

Das mit F (F = Flag, „Kennzeichen") bezeichnete Register (Abb. 6.4) ist das Statusregister. Es enthält zu jedem Zeitpunkt wichtige logische Informationen über die von der ALU zuletzt ausgeführte Operation. Der Inhalt kann durch spezielle Befehle abgefragt werden. In Abhängigkeit davon sind dann bestimmte Aktionen des Prozessors möglich. Um die Arbeitsweise von Registern wie auch anderer Speichereinheiten eines Computers richtig verstehen zu können, müssen wir uns mit einigen

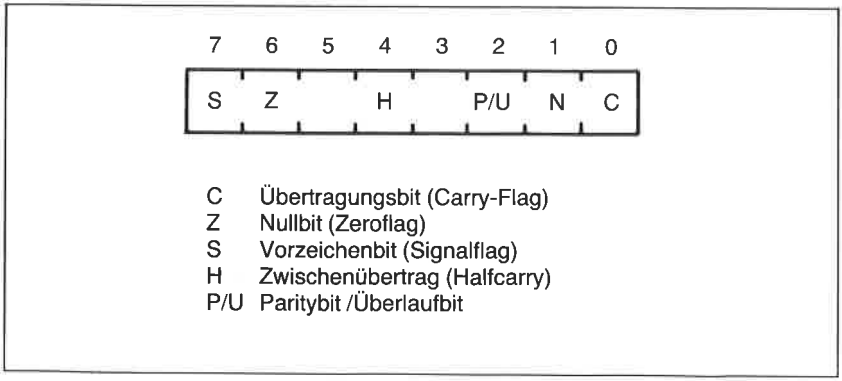



*Abb. 6.4: Organisation des Statusregisters F. Die leeren Positionen 3 und 5 enthalten immer Zufallswerte*

Begriffen und Vereinbarungen der Datentechnik etwas mehr vertraut machen. Bevor wir uns also um die bereits erwähnten wie auch die anderen Register etwas näher kümmern, wollen wir daher einen kurzen Ausflug in die Welt der Datentechnik machen.

### *Begriffe und Vereinbarungen der Datentechnik*

Jedes der insgesamt 8 Register besteht aus elementaren Speicherzellen, von denen jede nur zwei logische Zustände annehmen kann. Diese logischen Zustände werden üblicherweise durch die Symbole 0 und 1 dargestellt. Die in einer Speicherzelle des Registers abgelegte Information ist demnach *zweiwertig*. Hierfür ist der Begriff *binär* gebräuchlich.

Die durch eine zweiwertige, d. h. binäre Größe darstellbare Information wird Bit genannt. Ein Register setzt sich aus einer Folge von binären Speicherzellen zusammen, die jeweils für sich ein Bit speichern können (Abb. 6.5).

Alle Bits zusammen bilden ein *Datenwort* der Wortlänge w. Für die Wortlänge (oder auch Wortbreite) ist die Maßbezeichnung *bit* (also *klein* bit) gebräuchlich. Ein Datenwort der Wortlänge 8 bit wird als *Byte* bezeichnet. Wird der Begriff Byte im Zusammenhang mit einer Datenwortlänge verwendet, besitzt er den Status einer Maßbezeichnung und wird ebenfalls im allgemeinen klein geschrieben. Ein Byte hat also die Wortlänge 1 byte. Das rechtsäußere Bit besitzt aus nachfolgend noch näher erläuterten Gründen den niedrigsten Stellenwert (LSB = *l*east *s*ignificant *b*it), das linksäußere den höchsten Wert (MSB = *m*ost *s*ignificant *b*it).

Ein Byte setzt sich aus zwei Vierergruppen von Bits zusammen, die jeweils *Nibble* genannt werden. Die maximale Anzahl N von binären

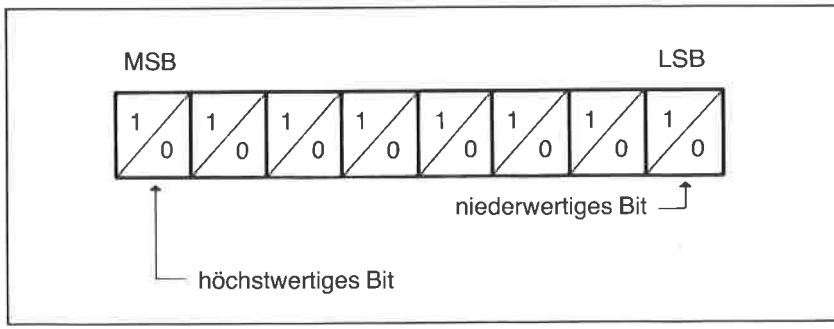



*Abb. 6.5: Organisation eines Registers aus einzelnen binären Speicherzellen*

Datenworten unterschiedlichen Inhalts wird durch die Wortlänge w eindeutig festgelegt. Es gilt die einfache Beziehung

$$N = 2^w$$

Das bedeutet, daß die Zahl N immer eine *Potenz von Zwei* ist. In der nachfolgenden Tabelle finden Sie eine Übersicht für Datenworte bis zu der in der Heimcomputertechnik üblichen Wortlänge von 16 bit.

| Wortlänge | Anzahl darstellbarer Datenworte | Wortlänge | Anzahl darstellbarer Datenworte |
|---|---|---|---|
| 1 | 2 | 9 | 512 |
| 2 | 4 | 10 | 1024 |
| 3 | 8 | 11 | 2048 |
| 4 | 16 | 12 | 4096 |
| 5 | 32 | 13 | 8192 |
| 6 | 64 | 14 | 16384 |
| 7 | 128 | 15 | 32768 |
| 8 | 256 | 16 | 65536 |

*Abb. 6.6: Zusammenhang zwischen der Wortlänge und der Anzahl darstellbarer Datenworte*

Für die Zahl $2^{10} = 1024$ ist in der Datentechnik die abkürzende Bezeichnung *Kilo* üblich. Beachten Sie bitte, daß diese Vereinbarung von der uns vertrauten Zuordnung im Zehnersystem abweicht (1 Kilo entspricht ja üblicherweise der Zahl 1000 und nicht 1024!).

### Interne Darstellung von Informationen

Grundsätzlich kann man einem binären Datenwort nicht ohne weiteres ansehen, welche Bedeutung es im aktuellen Fall hat. Letztendlich stellt es immer eine verschlüsselte Information dar, die beispielsweise einem alphanumerischen Zeichen (Textzeichen), einer binären Zahl oder auch einem Befehl im Maschinencode entsprechen kann. Gleichgültig, welche Bedeutung ein Byte auch im einzelnen haben mag: Es ist üblich, binäre Datenworte als sogenannte *Dualzahlen* zu interpretieren.

### *Duale Zahlendarstellung*

Dualzahlen werden nach einem ähnlichen Bildungsgesetz wie unsere Dezimalzahlen gebildet, die einem sogenannten *Stellenwertsystem* angehören. Sie erinnern sich? Die am weitesten rechts stehende Ziffer einer

ganzen Dezimalzahl gehört immer zu den Einern, die nächstfolgende links davon zu den Zehnern. Dann kommen die Hunderter, die Tausender usw. Je weiter links eine Ziffer innerhalb einer Zahl steht, desto höher ist offensichtlich ihr Wert. Den höchsten Wert hat immer die am weitesten links stehende Ziffer. Die stellenabhängigen Gewichte einer Dezimalzahl sind immer eine Potenz zur Basis 10 ($10^0=1$, $10^1=10$, $10^2=100$ usw.). Die zulässigen Ziffern selbst sind zehnwertig, d.h. sie können die Werte 0 bis 9 annehmen. Die Dezimalzahl 1234 läßt sich somit auch in der Form

$$1234 = 1*10^3 + 2*10^2 + 3*10^1 + 4*10^0$$

schreiben. Das Prinzip des Zahlenaufbaus ist bei den Dualzahlen genauso. Die einzelnen Ziffern haben jedoch jetzt keinen Umfang von 10 Zeichen ( 0 bis 9), sondern nur einen Umfang von zwei Zeichen ( 0 oder 1). Auch ist der Stellenwert (das Gewicht) einer Dualziffer anders als bei einer Dezimalzahl. Dualzahlen bauen auf Gewichten auf, die eine Potenz zur Basis 2 sind ($2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$ usw). Abb. 6.7 erläutert das Prinzip anhand eines einfachen Schemas für eine Dualzahl mit maximal 8 Ziffern.

Da die binären Ziffern 0 und 1 auch als Ziffern im dezimalen System erlaubt sind, entspricht der dezimale Wert einer Dualzahl ganz einfach immer der Quersumme der stellengewichteten Dualziffern, wie Sie Abb. 6.7 ebenfalls entnehmen können.

Für eine fünfstellige Dualzahl sind alle möglichen Kombinationen in der Abb. 6.8 als kleines Beispiel zusammengestellt. Zusätzlich angegeben sind noch die Verschlüsselungen in hexadezimaler Schreibweise. Sie werden diese im folgenden Abschnitt dieses Kapitels noch kennenlernen.

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |

$0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^2 + 1*2^0 = 55$ dez.

*Abb. 6.7: Schema für den Aufbau einer Dualzahl*

| dual | dez. | hex. | dual | dez. | hex. |
|---|---|---|---|---|---|
| 00000 | 0 | 00 | 10000 | 16 | 10 |
| 00001 | 1 | 01 | 10001 | 17 | 11 |
| 00010 | 2 | 02 | 10010 | 18 | 12 |
| 00011 | 3 | 03 | 10011 | 19 | 13 |
| 00100 | 4 | 04 | 10100 | 20 | 14 |
| 00101 | 5 | 05 | 10101 | 21 | 15 |
| 00110 | 6 | 06 | 10110 | 22 | 16 |
| 00111 | 7 | 07 | 10111 | 23 | 17 |
| 01000 | 8 | 08 | 11000 | 24 | 18 |
| 01001 | 9 | 09 | 11001 | 25 | 19 |
| 01010 | 10 | 0A | 11010 | 26 | 1A |
| 01011 | 11 | 0B | 11011 | 27 | 1B |
| 01100 | 12 | 0C | 11100 | 28 | 1C |
| 01101 | 13 | 0D | 11101 | 29 | 1D |
| 01110 | 14 | 0E | 11110 | 30 | 1E |
| 01111 | 15 | 0F | 11111 | 31 | 1F |

*Abb. 6.8: Die Dualzahlen von 00000 bis 11111 einschließlich deren dezimaler wie auch hexadezimaler Notation*

Eine Zusammenstellung aller mit einem Byte, also mit Datenworten der Wortlänge 8 bit darstellbaren Dualzahlen zwischen 00000000 und 11111111 finden Sie im Anhang C dieses Buchs.

Da es im Zusammenhang mit der Entwicklung von Programmen recht häufig vorkommt, daß dezimale Werte in duale umgewandelt werden müssen, finden Sie in Abb. 6.9 ein Beispiel für ein formales Schema, mit dem diese Aufgabe für Zahlen beliebigen Werts zu lösen ist. Die angegebene Methode beruht auf der fortlaufenden ganzzahligen Divison der Dezimalzahl durch zwei.

Da Sie nicht nur für die Auseinandersetzung mit der Programmierung Ihres Schneider CPC in Maschinensprache, sondern auch für einige Operationen in der Programmiersprache BASIC etwas über die Zweierkomplementdarstellung ganzer binärer Zahlen wissen müssen, können Sie sich im übernächsten Abschnitt dieses Kapitels die notwendigen Kenntnisse aneignen. Zuvor jedoch sollten Sie sich mit der vereinfachten Darstellung binärer Werte in Form von Hexadezimalzahlen vertraut machen.

| Dezimalzahl | | Quotient | Rest | |
|---|---|---|---|---|
| 14567 | :2 = | 7283 | 1 | MSB |
| 7283 | :2 = | 3641 | 1 | |
| 3641 | :2 = | 1820 | 1 | |
| 1820 | :2 = | 910 | 0 | |
| 910 | :2 = | 455 | 0 | |
| 455 | :2 = | 227 | 1 | |
| 227 | :2 = | 113 | 0 | |
| 113 | :2 = | 56 | 1 | |
| 56 | :2 = | 28 | 0 | |
| 28 | :2 = | 14 | 0 | |
| 14 | :2 = | 7 | 0 | |
| 7 | :2 = | 3 | 1 | |
| 3 | :2 = | 1 | 1 | |
| 1 | :2 = | 0 | 0 | LSB |

1456710 = 10110010001102

*Abb. 6.9: Beispiel für die Umrechnung einer Dezimal- in eine Dualzahl*

### Die hexadezimale Darstellung binärer Datenworte

Bei langen Datenworten ist der Umgang mit den Binärmustern recht unbequem und die Umrechnung in dezimale Zahlenwerte oder auch umgekehrt eine recht mühsame Angelegenheit. Um den Ziffernumfang zu verringern und andererseits noch einen gut durchschaubaren Bezug zur dualen Schreibweise sicherzustellen, bedient man sich in der Computertechnik gerne der *hexadezimalen* Zahlendarstellung. Zahlen dieses Typs bilden ein Stellenwertsystem im *sedezimalen* Zahlensystem. Hexadezimalziffern besitzen einen Zeichenumfang von 16 Symbolen zwischen 0 und F. Für die über die Zahl 9 hinausgehenden Werte werden hier die Buchstaben A bis F verwendet, um Verwechslungen mit Zahlen anderer Stellenwertsysteme zu vermeiden. Die Umwandlung von Dualzahlen in Hexadezimalzahlen ist denkbar einfach, wenn man sich vergegenwärtigt, daß jeweils 4 Bits (also ein Nibble) ausreichen, um 16 verschiedene Zustände darstellen zu können. Die Zuordnung sieht dann so aus, wie bereits in Abb. 6.8 dargestellt.

Jede beliebige Dualzahl läßt sich also einfach in eine Hexadezimalzahl umwandeln, indem man jeweils, von rechts beginnend, die Bits in Vierer-

gruppen zusammenfaßt und diesen nach der zuvor gezeigten Tabelle die entsprechenden Hexadezimalziffern zuordnet. Nach dem Schema in Abb. 6.8 entspricht beispielsweise der Dualzahl

```
1000 0101 1110 0111
  8    5    E    7
```

die Hexadezimalzahl

85E7

Der CPC versteht diesen Zahlentyp, wenn eine Zahlenangabe durch ein vorangestelltes & oder &H gekennzeichnet ist (Beispiel: &85E7 oder &H85E7). Hexadezimalzahlen sind nichts anderes als eine freundlichere und im allgemeinen leichter merkbare Darstellung binärer Datenwerte. Rechnen läßt sich üblicherweise dagegen besser unmittelbar mit den Dualzahlen.

Bevor wir nach diesem kurzen Exkurs wieder zu unserem Mikroprozessor Z80 und seinen Registern zurückkehren, sollten Sie sich den nachfolgenden Abschnitt über die Darstellung von Dualzahlen in der Zweierkomplementform ansehen. Sie werden nämlich bei Ihrem Umgang mit dem CPC noch häufig von ihnen Gebrauch machen müssen.

### Zweierkomplementdarstellung ganzer Zahlen

Jede Dualzahl gestattet nach den vorab erläuterten Regeln die Darstellung von $2^w$ verschiedenen Zahlenwerten, wenn w die Wortlänge des binären Datenwortes ist. Bisher waren wir davon ausgegangen, daß nur positive ganze Zahlen zu verschlüsseln sind. Wir müssen aber auch negative ganzzahlige wie auch gebrochene Zahlen zulassen. Einer alten internationalen Übereinkunft zufolge, die ursprünglich nicht unter dem Gesichtspunkt der digitalen Datenverarbeitung mit Computern getroffen wurde, werden negative Zahlen dadurch gekennzeichnet, daß das höchstwertige Bit (MSB) zu Eins gesetzt wird. Bei gleichem Werteumfang brauchen wir also ein Bit mehr, oder anders gesagt: Wenn wir ein Bit für die Kennzeichnung des Vorzeichens spendieren müssen, wird der Wertebereich bei gleicher Wortlänge verringert. Mit drei Bit können wir also entweder die positiven ganzzahligen Werte 0 bis 7 oder nur die Werte von −4 bis +3, mit 4 Bit diejenigen von −16 bis +15 usw. verschlüsseln.

Grundsätzlich sind unter der zuvor genannten Annahme bezüglich des höchstwertigen Bits zwei voneinander abweichende Darstellungsarten

gebräuchlich. Die eine wird in der englischen Sprache als „binary offset" (Offset binär), die andere als „two's complement" (Zweierkomplement) bezeichnet. In der Abb. 6.10 sind beide Möglichkeiten anhand der Zahlenverschlüsselung mit 5 bit langen Codeworten der positivwertigen dualen Verschlüsselung gegenübergestellt.

Digitale Rechenanlagen führen alle Rechenoperationen, also auch die Subtraktion, auf die Addition zurück, weil dies genau die Operation ist, die eine elektronische Datenverarbeitungsanlage beherrscht.

Eine besonders einfache Art der Zahlenverarbeitung ergibt sich dann, wenn negative Zahlen in der Zweierkomplementform vereinbart werden. Da Sie auf diese Form der Zahlendarstellung bei Ihrem CPC nicht nur im Zusammenhang mit der Programmierung in Maschinensprache, sondern auch bei Benutzung von BASIC als Programmiersprache stoßen, sollen Sie nachfolgend eine einfache Methode zur Bildung von dualen Zahlen in der Zweierkomplementform kennenlernen, die darauf beruht, daß Sie bei einem binären Datenwort alle links von der niederwertigsten Eins lie-

| positiv dual | dual mit Offset | Zweierkomplement |
|---|---|---|
| 0 0000 | 7 0000 | 7 0111 |
| 1 0001 | 6 0001 | 6 0110 |
| 2 0010 | 5 0010 | 5 0101 |
| 3 0011 | 4 0011 | 4 0100 |
| 4 0100 | 3 0100 | 3 0011 |
| 5 0101 | 2 0101 | 2 0010 |
| 6 0110 | 1 0110 | 1 0001 |
| 7 0111 | 0 0111 | 0 0000 |
| 8 1000 | −1 1000 | −1 1111 |
| 9 1001 | −2 1001 | −2 1110 |
| 10 1010 | −3 1010 | −3 1101 |
| 11 1011 | −4 1011 | −4 1100 |
| 12 1100 | −5 1100 | −5 1011 |
| 13 1101 | −6 1101 | −6 1010 |
| 14 1110 | −7 1110 | −7 1001 |
| 15 1111 | −8 1111 | −8 1000 |

*Abb. 6.10: Vergleich der Werteverschlüsselung bei rein positiver wie auch positiver und negativer Zahlendarstellung nach der binären Offsetmethode (Mitte) sowie im Zweierkomplement (rechts) für Codeworte mit 4 bit Wortlänge*

genden Bits in ihrem logischen Wert umkehren. Die Eins mit dem niedrigsten Stellenwert ist immer die am weitesten rechts liegende. Bei der Zahl 001001000*1*00 finden Sie diese an der dritten Stelle von rechts (kursiv gekennzeichnet). Die Zweierkomplementform dieses Datenwortes lautet nach der zuvor angegebenen Regel 110110111100.

Weitere Beispiele:

110001111: 001110001
001000000: 111000000

Überprüfen Sie bitte anhand der in Abb. 6.10 gezeigten Zahlenwerte, ob Sie das Prinzip gut verstanden haben.

Daß der BASIC-Interpreter Ihres Schneider CPC in gleicher Weise verfährt, können Sie an einem einfachen Beispiel testen. Der unmittelbar eingegebene Befehl

```
PRINT &CFFF
```

führt zur Ausgabe der Zahl −12289 auf dem Bildschirm. Die Erklärung hierfür ist denkbar einfach. Hexadezimal &CFFF entspricht dem dualen Datenwort 1100 1111 1111 1111. Da das höchstwertige Bit gleich Eins ist, faßt der Rechner die Zahl als Zweierkomplement auf und interpretiert sie als −12289.

Zur Kontrolle sollten Sie einmal das Zweierkomplement dieser Zahl bilden. Sie erhalten dann die positive Dualzahl, die Sie durch Bildung der gewichteten Quersumme einfach in die entsprechende Dezimalzahl umwandeln können: Das Zweierkomplement von 1100 1111 1111 1111 lautet: 0011 0000 0000 0001. Die Quersumme ergibt dann $1*2^{13} + 1*2^{12} + 1*2^{0} = 8192 + 4096 + 1 = 12289$. Die Bildung des Zweierkomplements von einer bereits im Zweierkomplement vereinbarten negativen Zahl liefert offensichtlich den positiven korrekten Zahlenwert. Der Schneider CPC stellt somit als 16-bit-Worte verschlüsselte ganzzahlige Werte in Zweierkomplementform im Bereich von −32768 bis + 32767 dar.

Nach diesem kurzen Ausflug in die Welt binärer Zahlen kehren wir nun noch einmal zu unserem Mikroprozessor Z80 zurück und schauen uns abschließend die noch nicht erwähnten Register, den Vorgang der Befehlsverarbeitung und die Speicherbelegung des CPC an.

## Die Register des Z80

Die Register des Z80 weisen mit Ausnahme der Indexregister IX und IY, des Stapelzeigers SP sowie des Programmzählers PC eine Wortbreite von 8 bit auf. Neben den bereits erwähnten 8-bit-Registern A (Akkumulator) und F (Statusregister) erfüllen auch die mit I und R bezeichneten Register spezielle Aufgaben. Das I-Register dient zur Auswahl einer Speicherseite bei Unterbrechungsvorgängen (Interrupts genannt), während das Register R als Zählregister im Zusammenhang mit dem Wiederauffrischungsvorgang dynamischer Speicherbausteine eingesetzt wird. Eine genaue Erläuterung deren Funktionsweise muß vertiefender Spezialliteratur vorbehalten bleiben.

Die übrigen 8-bit-Register B, C, D, E, H und L übernehmen im Zusammenhang mit einer sehr leistungsfähigen Palette von Maschinenbefehlen allgemeine Aufgaben. Von Bedeutung ist, daß jeweils aus B und C, D und E bzw. H und L Registerpaare gebildet werden können, die 16 bit lange Datenworte oder Adreßangaben aufnehmen können.

Die Indexregister IX und IY können Datenworte mit einer Wortlänge von 16 bit speichern. Sie sind demnach in der Lage, Zahlenwerte im Bereich zwischen 0 und 65535 darzustellen. Sie werden vornehmlich dazu verwendet, um eine indizierte Adressierung durchzuführen, d. h. im Speicher abgelegte Datentabellen zu adressieren. Das funktioniert so ähnlich wie die Indizierung von Feldern unter BASIC. Unterstützt wird diese Art der Adressierung durch sehr leistungsfähige Maschinenbefehle. Auch das mit SP bezeichnete Register ist mit seiner Wortbreite von 16 bit zur Aufnahme von Speicheradressen vorgesehen. SP ist eine Abkürzung für „stack pointer", also Stapelzeiger. Unter einem Stapel versteht man einen nach einem speziellen Schema verwalteten Speicherbereich. Die Datenablage ist dadurch gekennzeichnet, daß immer das sogenannte LIFO-Prinzip (last in first out) angewendet wird. Das zuletzt in den Speicher eingeschriebene Datenelement muß immer als erstes wieder abgerufen werden. Angewendet wird dieses Prinzip zur Adressenzwischenspeicherung bei Unterprogrammaufrufen wie auch bei der Abarbeitung geschachtelter Programmschleifen.

Das mit PC bezeichnete Register enthält immer die aktuelle Adresse der als nächstes auszulesenden Speicherzelle. In dieser ist entweder ein Befehl oder ein Datenwert abgelegt. Vereinfacht gesagt wird beim Programmstart immer zunächst die Startadresse in den PC geladen, dessen Inhalt bei der Befehlsabarbeitung automatisch so lange erhöht wird, bis das Programmende erreicht ist.

Die in der Abb. 6.3 ebenfalls noch angegebenen Register A' bis L' können zusätzlich verwendet werden. Man kann jedoch immer nur mit einer der beiden Gruppen gleichzeitig arbeiten. Interessant ist der zweite Registersatz im Zusammenhang mit der Vorder- und Hintergrundverarbeitung von Programmen. Auch hierauf kann in einem einführenden Buch wie diesem nicht eingegangen werden.

### Die Befehlsverarbeitung des Z80

Die Verarbeitung von Befehlen und Daten durch den Mikroprozessor geschieht unter Kontrolle einer inneren Uhr (einem Quarzoszillator mit einer Taktfrequenz von 4 MHz beim Schneider CPC 464) in mehreren aufeinanderfolgenden Schritten. Wesentlich beteiligt sind dabei ein weiteres, in der Übersicht der Abb. 6.3 nicht genanntes Register, das Befehlsregister genannt wird, sowie zwei ebenfalls auf dem Mikroprozessorchip integrierte Funktionseinheiten, die Befehlsdecoder und Steuerwerk genannt werden. Das Befehlsregister kann nicht durch den Programmierer direkt geladen werden.

Jedes durch den Prozessor abzuarbeitende Programm ist irgendwo im Arbeitsspeicher (ROM oder RAM), beginnend bei einer fest vorgegebenen Startadresse, abgelegt. Nach dem Laden der Startadresse in das PC-Register wird diese auf den Adreßbus geschaltet und sorgt somit für die Auswahl einer konkreten Speicherzelle, die den ersten Programmbefehl enthält. Vom Steuerwerk wird gleichzeitig ein Lesesignal über den Kontrollbus ausgegeben. Der Inhalt des ersten adressierten Speichers wird anschließend auf den Datenbus geschaltet und in das Befehlsregister geladen.

Diese Phase wird *Holphase* genannt (Fetchzyklus). Der Befehlsdecoder entschlüsselt den Befehl und veranlaßt, daß unter der Kontrolle des Steuerwerks die dem Befehl entsprechenden Operationen ausgeführt werden *(Ausführungsphase)*. Der Programmzähler wurde zwischenzeitlich erhöht, und er verweist bereits auf die nächstfolgende Speicheradresse. Die vollständige Verarbeitung des Befehls kann mehrere Hol- und Ausführungszyklen erforderlich machen, je nachdem, ob noch weitere Bytes als Bestandteile des Befehls benötigt werden oder nicht.

Die Abb. 6.11 erläutert anhand einer schematischen Darstellung den zuvor geschilderten Vorgang unter der Annahme, daß der erste zu bearbeitende Befehl unter der Adresse &1000 zu finden ist.

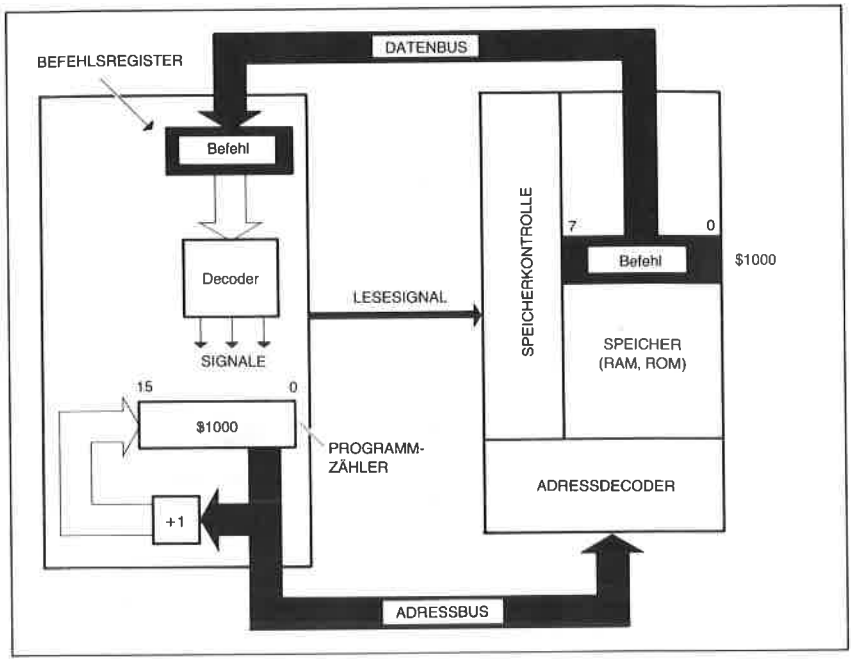



*Abb. 6.11: Schema der Befehlsverarbeitung durch den Mikroprozessor*

## Die Speicherbelegung (Memory Map) des CPC

Befehle wie auch Daten für den Prozessor werden – wie bereits erwähnt – aus dem Arbeitsspeicher des Systems, also entweder aus dem ROM oder dem RAM abgerufen. Im RAM abgespeicherte Informationen können allerdings nicht jede beliebige Speicherbelegung einnehmen, weil das jeweils aktive Betriebssystem eine definierte Standardorganisation festlegt. Die folgende Abb. 6.12 zeigt anhand eines Übersichtsschemas diese Standard-Speicherbelegung des CPC. Unter Kontrolle des Diskettenorientierten Betriebssystems CP/M 2.2 sieht die Belegung im einzelnen etwas anders aus.

Da der Z80 einen Programmzähler mit einer Wortlänge von 16 bit besitzt, können nur maximal 64 K verschiedene Adressenwerte zwischen &0000 und &FFFF erzeugt werden (das entspricht den dezimalen Adreßwerten 0 bis 65535). Neben einem Schreib-/Lesespeicher dieses Umfangs besitzt der CPC noch zusätzlich einen Festwertspeicher von insgesamt 32 kilo-

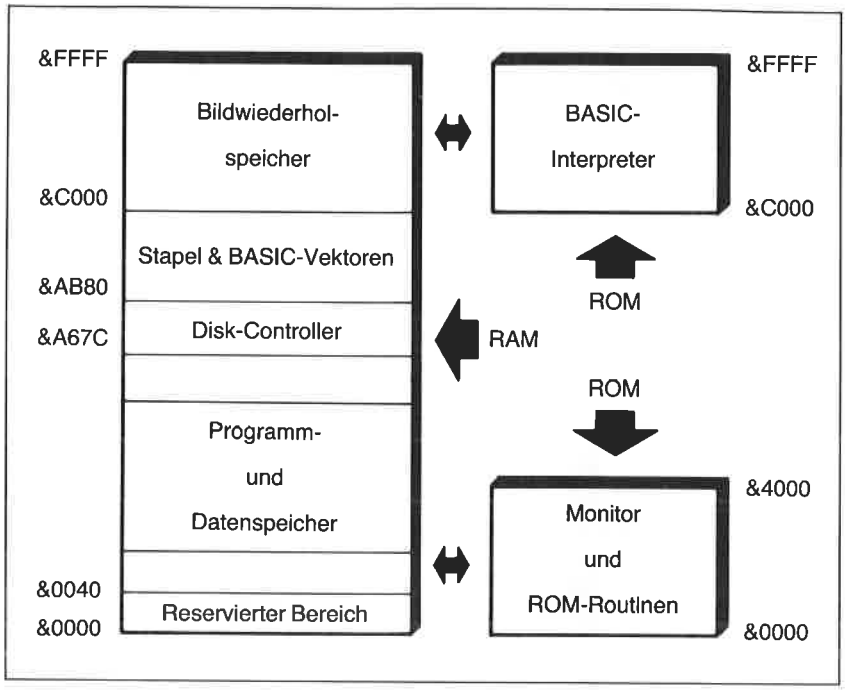



*Abb. 6.12: Die Standard-Speicherbelegung des Schneider CPC 464/664*

byte Umfang, der in zwei Blöcke zu je 16 kbyte aufgeteilt ist. Das Monitorprogramm sowie die Hilfsroutinen des Betriebssystems liegen im niederwertigen Adreßteil des Speichers (zwischen den Adressen &0000 und &4000), während der BASIC-Interpreter im höherwertigen Adreßteil (zwischen &C000 und &FFFF) angesiedelt ist. Natürlich kann der Prozessor mit einer der Adressen in diesen Bereichen jeweils nur auf das RAM *oder* das ROM zugreifen, nicht auf beide gleichzeitig. Es muß somit auf den jeweils physikalisch korrekten Speicherort umgeschaltet werden. Ohne spezielle Maßnahmen wird beispielsweise der BASIC-Befehl PEEK (adr) immer nur auf Inhalte im RAM-Bereich zugreifen.

Der CPC ist so organisiert, daß er auf insgesamt bis zu vier parallel liegende Zusatz-ROMs umschalten kann. Falls Sie mit dem im Kapitel 4 angegebenen kleinen Programm zur Darstellung von Speicherinhalten auf dem Bildschirm einmal einen Blick in die ROM-Routinen werfen wollen, müssen Sie dazu entweder durch CALL &B900 das obere ROM ab

&C000 bis &FFFF oder durch CALL &B906 das untere ROM von &0000 bis &3FFF freischalten. Leseoperationen in diesen Adreßbereichen wirken dann nicht mehr auf den Schreib-/Lesespeicher. Wirklich sinnvoll ist dies nur, wenn Sie sich eines leistungsfähigen *Disassemblers* bedienen. Er funktioniert genau umgekehrt wie ein *Assembler* (siehe Kapitel 7). Von einem Disassembler gelesene binäre Codes werden, soweit dies möglich ist, in symbolische Elemente der *Assemblersprache* übersetzt.

Kapitel 7

# Einführung in die Maschinenprogrammierung

## Übersicht

Sie haben im vorangegangenen Kapitel gelesen, daß der Schneider CPC mit einem Mikroprozessor des Typs Z80 ausgerüstet ist. Wie andere Prozessoren auch, kann er nur Befehlsworte und Daten im binären Format, d. h. als Muster von Nullen und Einsen verarbeiten. Die Schöpfer des Mikroprozessors Z80 haben ihrem „Kind" einen binären Befehlssatz gewaltigen Umfangs mit in die Wiege gelegt. Einen Befehlssatz, den Sie sich zu eigen machen müssen, wenn Sie erfolgreich Programme in der „Muttersprache" dieses Prozessors schreiben wollen. Sie müssen dies dann tun, wenn Sie die höchstmögliche Geschwindigkeit des Systems ausnutzen wollen. Interessant ist dies beispielsweise für die Entwicklung von Bildschirmspielen mit bewegten grafischen Objekten oder den Anschluß zeitkritischer peripherer elektronischer Komponenten.

Im einzelnen lesen Sie etwas über den inneren Aufbau von binär codierten Maschinenbefehlen. Sie können sich außerdem über den Befehlssatz sowie die wichtigsten Adressierungsarten des Z80 informieren. Darüber hinaus lernen Sie die Arbeitsweise eines Übersetzerprogramms zur komfortablen Erstellung von Maschinenprogrammen kennen.

Da den angeschnittenen Themen in diesem Buch nur ein Kapitel begrenzten Umfangs gewidmet sein kann, muß die Information zwangsläufig unvollständig bleiben. Sie sollten jedoch nach der Lektüre ohne Probleme weiterführende Literatur lesen und verstehen und vielleicht einige einfache Probleme selbst lösen können.

## Die Struktur von Maschinenbefehlen

Programme in Maschinensprache werden im Computer in Form einer Folge von 8-bit-Worten im Speicher abgelegt. Sie bestehen aus Befehlsworten, die nach Abruf aus dem Speicher in das Befehlsregister und der

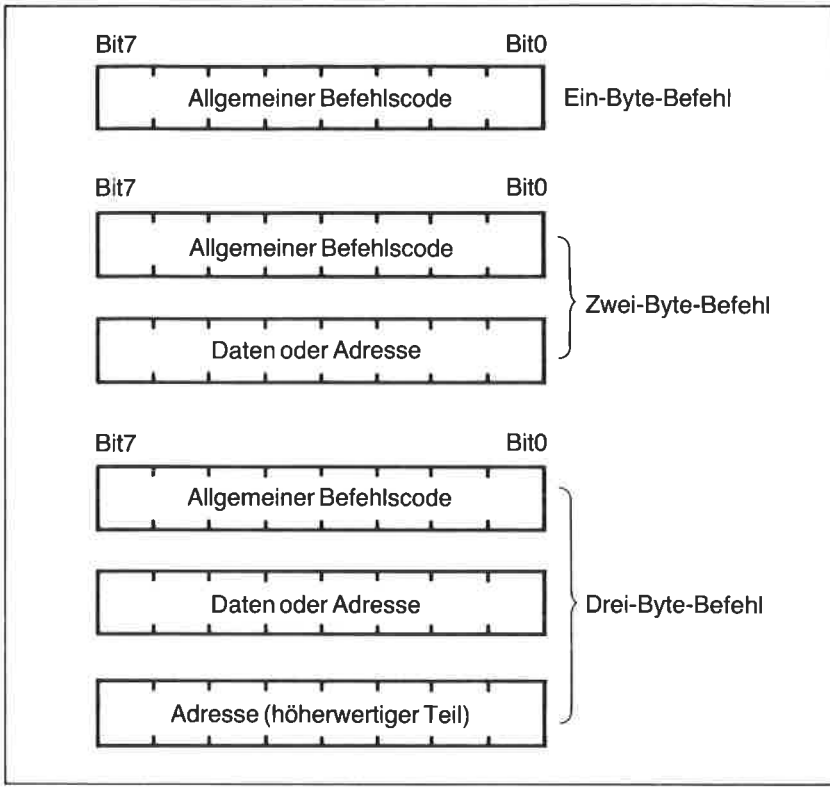



*Abb. 7.1: Verschiedene Formate von Maschinenbefehlen*

Entschlüsselung durch den Befehlsdecoder Operationen des Prozessors bewirken, sowie aus Adressen und/oder Daten als ergänzende Bestandteile.

Abb. 7.1 zeigt zur Erläuterung dieses Sachverhalts anhand eines einfachen Schemas den Aufbau eines allgemeinen Maschinenbefehls. Er besteht immer aus einem Befehlswort von mindestens 8 bit Wortlänge, das den verallgemeinerten Befehlscode enthält. Je nach Befehlstyp und Adressierungsart wird dieses Befehlswort noch durch ein oder zwei (bei einigen speziellen Befehlen auch drei) weitere Bytes ergänzt, die entweder Daten oder Adreßangaben enthalten. Maschinenbefehle, die nur aus einem einzigen Byte bestehen, werden Ein-Byte-Befehle, die anderen Mehr-Byte-Befehle (Zwei-Byte-, Drei- oder Vier-Byte-Befehle)

genannt. Am besten schauen wir uns das anhand einiger praktischer Beispiele an:

### *Ein-Byte-Befehle*

Der Befehl LD r,r' ist ein typisches Beispiel für einen Ein-Byte-Befehl. Er bedeutet: Übertrage (*LOAD*) den Inhalt des unter r' vereinbarten Registers in das unter r näher bezeichnete Register. Die Parameter r und r' sind in dieser symbolischen Schreibweise nur Platzhalter für Registerkennungen, die vom Programmierer anzugeben sind. In binärer Schreibweise ist der Befehl nach dem Schema in Abb. 7.2 aufgebaut.

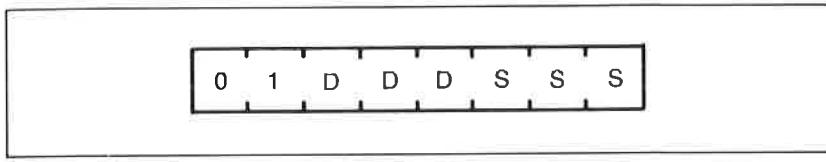



*Abb. 7.2: Allgemeine Form des Maschinenbefehls LDA r,r'*

Unter DDD ist der Code für das Zielregister (*d*estination), unter SSS der des Quellregisters (*s*ource) zu vereinbaren. Um den Befehl vollständig angeben zu können, müssen wir noch die Registercodes einsetzen. Diese sind für den Z80 in Abb. 7.3 gezeigt.

| Code | Register |
|---|---|
| 000 | B |
| 001 | C |
| 010 | D |
| 011 | E |
| 100 | H |
| 101 | L |
| 110 | – (Speicher) |
| 111 | A |

*Abb 7.3: Die Registercodes des Z80*

Konkret wird damit beispielsweise der Befehl LD A,B (Überführe den Inhalt des Registers B in den Akkumulator A) so aussehen:

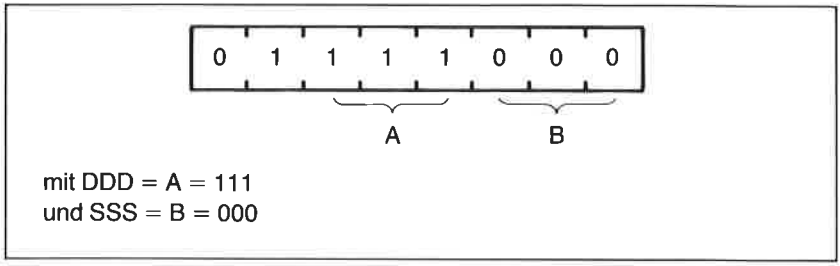



*Abb. 7.4: Binäre Verschlüsselung des Befehls LD A,B*

Mit diesem Befehlscode kann der Prozessor unmittelbar etwas anfangen. In hexadezimaler Schreibweise lautet er einfach &78.

### *Zwei-Byte-Befehle*

Ein typischer Zwei-Byte-Befehl ist ADD A,n. Er bedeutet: Addiere den unter n definierten Wert zum Inhalt des Registers A. Das erste Byte dieses Maschinenbefehls enthält den Befehlscode, das nachfolgende den Zahlenwert von n. Nehmen wir an, daß der Wert 5 zum aktuellen Inhalt des Akkumulators addiert werden soll. Dann lautet die Folge der beiden hierzu benötigten 8 bit langen Datenworte:

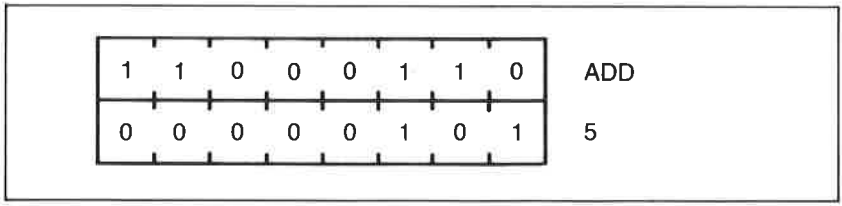



*Abb. 7.5: Verschlüsselung des Zwei-Byte-Befehls ADD A,5*

In hexadezimaler Notation können die beiden Bytes zu &C6, &05 angegeben werden.

Wird das erste Byte des Befehls decodiert, faßt der Prozessor das zweite Byte immer als eine Konstante auf, die – gleichgültig, welche aktuelle Bedeutung sie auch haben mag – als Dualzahl behandelt und zum Inhalt

des Registers A binär addiert wird. Das für die Ausführung des Befehls verantwortliche Steuerwerk ruft bei Mehr-Byte-Befehlen immer eine dem Befehl entsprechende Anzahl von Bytes aus dem Speicher ab und gibt diese zur Weiterverarbeitung an die ALU frei. Sie haben hierzu bereits im Kapitel 6 einiges gelesen.

### *Drei-Byte-Befehle*

Der Bezeichnung entsprechend bestehen Drei-Byte-Befehle aus einer Folge von drei aufeinanderfolgenden Datenworten zu je 8 bit Wortlänge. Ein typischer Befehl dieser Kategorie ist LD A,(nn). Er bedeutet: Überführe den Inhalt derjenigen Speicherzelle, deren Adresse unter nn vereinbart ist, in das A-Register. Da für eine vollständige Adreßangabe ein Datenwort von 16 bit Wortlänge erforderlich ist, müssen zwei Bytes zur Kennzeichnung der auszulesenden Speicherstelle vorgesehen werden. Der niederwertige Teil der Adresse (das *Lowbyte*) wird in dem unmittelbar auf das Befehlswort folgenden Byte, der höherwertige Teil (das *Highbyte*) in dem dritten Byte abgelegt. Für die Adresse &910F, die dem Binärmuster 1001 0001 0000 1111 entspricht, sieht dann die Folge der zum obengenannten Befehl gehörenden Bytes wie folgt aus:

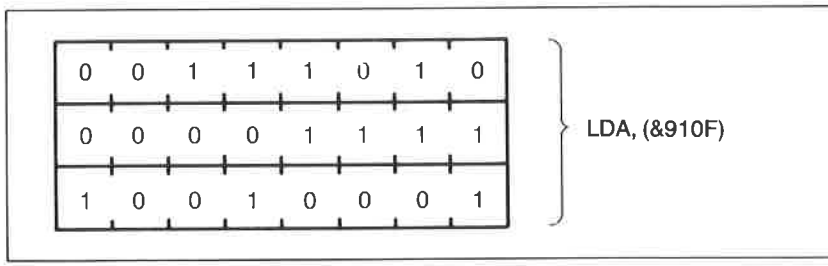



*Abb. 7.6: Verschlüsselung des Drei-Byte-Befehls LD A,(&910F)*

Die Befehlsfolge in hexadezimaler Schreibweise lautet: &3A, &0F, &91.

### *Vier-Byte-Befehle*

Neben den zuvor bereits anhand kleiner Beispiele vorgestellten Befehlstypen kennt der Z80 auch noch einige Befehle, die ein zusätzliches Byte erforderlich machen. Zu diesen gehört beispielsweise der Befehl BIT b,(IX+d), dessen Struktur in Abb. 7.7 angegeben ist. Der Befehl führt dazu, daß das b-te Bit derjenigen Speicherzelle auf seinen Zustand hin (0 oder 1) abgefragt wird, dessen Adresse in der Klammer vereinbart wurde.

Diese Adresse setzt sich im aktuellen Fall aus jenem 16-bit-Wert zusammen, der im Indexregister IX steht, wenn der Adreßoffset d hinzuaddiert wird. Byte 1 und Byte 2 enthalten demnach den Befehlscode, in Byte 3 wird der Adreßoffset, in Byte 4 die duale Kennung des abzufragenden Bits vereinbart.

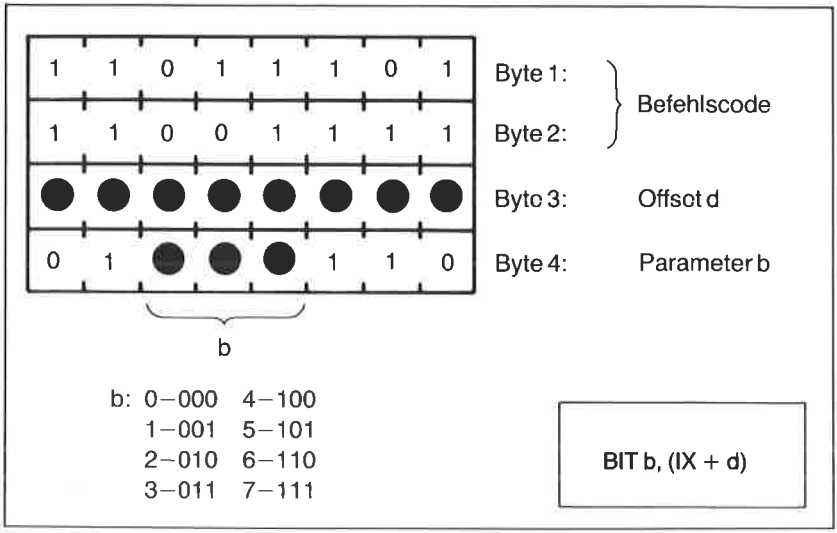



*Abb. 7.7: Allgemeines Format des Vier-Byte-Befehls BIT b,(IX+d)*

Da dieser Maschinenbefehl recht kompliziert aussieht, wollen wir uns auch hierzu ein konkretes Beispiel ansehen.

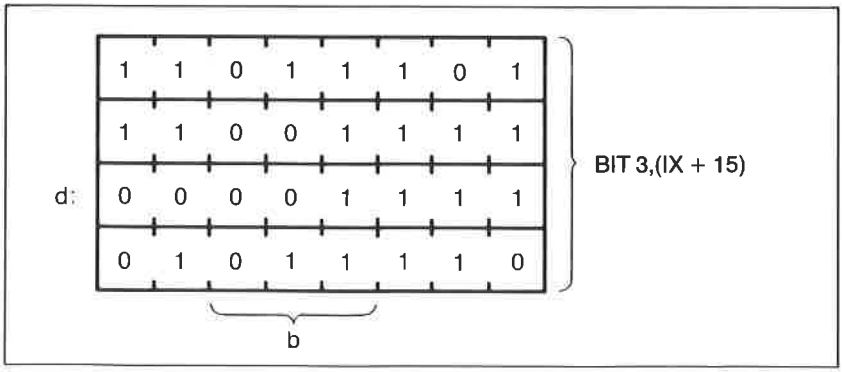



*Abb. 7.8: Verschlüsselung des Befehls BIT b,(IX+d) für b=3 und d=15*

Steht im Indexregister die Adresse &9100, dann wird mit dem Befehl BIT 3,(IX+15) das Bit Nr. 3 (das ist das vierte (!) Bit von rechts) des Speichers mit der Adresse &910F auf seinen logischen Zustand hin abgefragt. Abb. 7.8 zeigt die entsprechende Bytefolge.

Enthält der Speicher mit der Adresse &910F beispielsweise die Bitfolge 10010001, dann führt das Ergebnis dieser Abfrage dazu, daß das Z-Bit im Statusregister gesetzt wird. Dies liegt daran, daß das Bit Nr. 3 (das vierte Bit von rechts) in unserem Beispiel den Wert Null hat. Der Befehl wirkt sich also unmittelbar auf den Zustand des Z-Bits im Statuswort aus.

Es sind gerade derartige, auf den ersten Blick recht kompliziert erscheinende Befehle, die dem erfahrenen Entwickler von Maschinenprogrammen als besonders willkommene Hilfsmittel für die Erstellung besonders leistungsfähiger, codesparender und damit schnell ablaufender Programme dienen.

### Die Adressierungsarten des Z80

Ein einführendes Kapitel in die Programmierung in Maschinensprache ist nicht vollständig, wenn nicht noch einige Worte zum Thema Adressierungsarten gesagt werden. Bereits im vorangegangenen Kapitel 6, das Ihnen die interne Arbeitsweise von Computern und deren Bauelementen erläutert hat, haben Sie gelesen, daß zum Abruf eines Operanden die entsprechende Speicherzelle, in der dieser abgespeichert ist, angesprochen, d. h. adressiert werden muß. Die Art, wie dies geschieht, spielt im allgemeinen bei dem Entwurf eines Maschinenprogramms eine entscheidende Rolle für dessen Effektivität und Ablaufgeschwindigkeit. Voraussetzung für ein gutes Programm ist somit, daß der Programmierer die zur Verfügung stehenden Adressierungstechniken perfekt beherrscht.

Im einfachsten Fall steht der Operand in einem Register. Befehle, die ausschließlich mit Registern arbeiten, verwenden die *implizite* Adressierung. Es handelt sich um Ein-Byte-Befehle, bei denen das (oder auch die) Register in Form dualer Schlüsselworte Bestandteil des Operationscodes ist. Ein typisches Beispiel ist der bereits erwähnte Befehl LD r,r' (siehe auch Abb. 7.2). Dieser Befehlstyp ist deshalb sehr nützlich, weil er im Gegensatz zu Mehr-Byte-Befehlen eine sehr kurze Ausführungsdauer benötigt.

Steht der Operand nicht in einem Register, sondern in einer beliebigen Zelle des Arbeitsspeichers (ROM oder RAM), dann ist im allgemeinen Fall die Angabe einer 16-bit-Adresse nötig, um die Lage des Operanden zu kennzeichnen. In diesem Fall folgen dem Befehlscode *zwei* weitere

Bytes. Diese Art der Adressierung wird als *absolut* bezeichnet. Ein typischer Befehl mit absoluter Adressierung ist LD HL,(nn). In diesem Fall ist (nn) die Adresse, unter der der 8-bit-Operand für die Ladeoperation zu finden ist. Die Klammer um nn besagt also, daß nicht der Wert nn selbst, sondern der unter der Adresse nn stehende Wert in das Registerpaar HL zu laden ist.

Bedingte wie auch unbedingte Sprünge im Programm werden über eine *relative* Adressierung ausgeführt. Hierbei handelt es sich typischerweise um Zwei-Byte-Befehle, in denen das erste Byte den Befehlscode und das zweite den Adreßoffset (Sprungweite) enthält. Sie werden diesen Adressierungstyp noch im Rahmen eines Beispiels kennenlernen.

Die *indizierte* Adressierung benötigt einen Befehlscode von zwei byte Länge, dem dann noch ein oder zwei weitere Bytes folgen können. Sie macht von den beiden 16 bit breiten Indexregistern IX unbd IY Gebrauch. Sie werden vornehmlich zur Indizierung von Datentabellen verwendet, die so ähnlich wie Felder in der Programmiersprache BASIC behandelt werden können. Zu diesem Adressierungstyp gehört beispielsweise der Befehl LD (IX + d),n. Er besagt, daß unter jener Adresse, die durch die Addition des Inhaltes des IX-Registers und einer Zahl d gebildet wird, der Datenwert n abzulegen ist. Der Befehl verknüpft im übrigen zwei unterschiedliche Adressierungsarten miteinander, nämlich die indizierte und die *unmittelbare* Adressierung. Unter der unmittelbaren Adressierung versteht man den direkten Zugriff auf konstante Datenwerte, die nach dem 8-bit-Befehlswort unmittelbar vereinbart werden. Da der Z80 sowohl Register mit 8 wie auch mit 16 bit Wortlänge besitzt, führt die unmittelbare Adressierung allein entweder zu Zwei- oder Drei-Byte-Befehlen. Typische Befehle dieser Kategorie sind LD r,n oder LD dd,nn.

Neben den zuvor erwähnten Adressierungsarten kennt der Z80 noch eine *indirekte* Adressierung, die insbesondere im Zusammenhang mit Unterprogrammtechniken Verwendung findet. Typisch für diese Art der Adressierung ist, daß der Operand unter der Adresse steht, deren Adresse wiederum Bestandteil des Befehls ist.

Falls Sie sich eingehender mit dem Thema Maschinensprache befassen wollen oder müssen, sollten Sie sich anhand weiterführender Bücher sehr intensiv mit den Möglichkeiten der Adressierung sowie dem Befehlssatz des Z80 im Detail auseinandersetzen.

Wegen des einführenden Charakters dieses Kapitels ist es hier aus naheliegenden Gründen nicht möglich, alle Befehlsvarianten des Z80 sowie

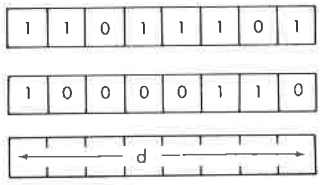
die Wirkungsweise unterschiedlicher Adressierungsarten genau vorzustellen. Erschöpfende Informationen über den Befehlssatz des Z80, die Funktion und Wirkung einzelner Befehle sowie auch über die Beeinflussung von Statusbits finden Sie entweder in den technischen Unterlagen und den Handbüchern des Herstellers oder aber in mehr oder weniger umfangreichen Lehrbüchern, die von verschiedenen Verlagen auf dem Markt angeboten werden. Beachten Sie in diesem Zusammenhang auch die SYBEX-Bibliothek zum Schluß dieses Buches, in der Sie ein Fülle weiterführender Literatur finden werden. Die Abbildung 7.10 zeigt als Beispiel einen modifizierten Ausschnitt aus dem Buch „Programmierung des Z80", das ebenfalls im SYBEX-Verlag erschienen ist. Sämtliche Varianten des Befehls sind in schematischer Form angegeben. Zusätzlich finden Sie noch Informationen über die hexadezimalen Befehlscodes, über die benötigten Maschinen- und Taktzyklen sowie die Ausführungszeiten unter der Annahme, daß der Prozessor mit 2 MHz getaktet wird. Das durch den Befehl beeinflußte Statusbit ist durch • gekennzeichnet.

**ADD A, (IX + d)**

Addiere den Akkumulator mit der indiziert adressierten Speicherstelle (IX + d)

*Funktion:* A ← A + (IX + d)

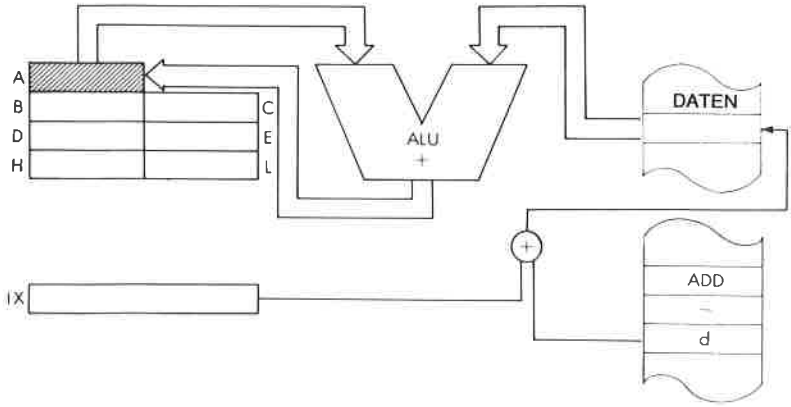
*Format:*




*Beschreibung:* Der Inhalt des Akkumulators und der Inhalt der Speicherstelle werden addiert, die durch den Inhalt des Registers IX plus den unmittelbaren Offset adressiert wird. Das Ergebnis wird im Akkumulator gespeichert.

*Abb. 7.9: Ausschnitt aus der Z80-Befehlsliste für den Befehl ADD A, (IX+d)*

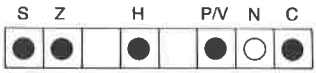
*Datenfluß:*




*Befehlsablauf:* 5 M Zyklen; 19 T Zustände: 9,5 µsek @ 2 MHz

*Adressierungsart:* Indiziert.

*Flags:*




*Beispiel:* ADD A, (IX + 3)

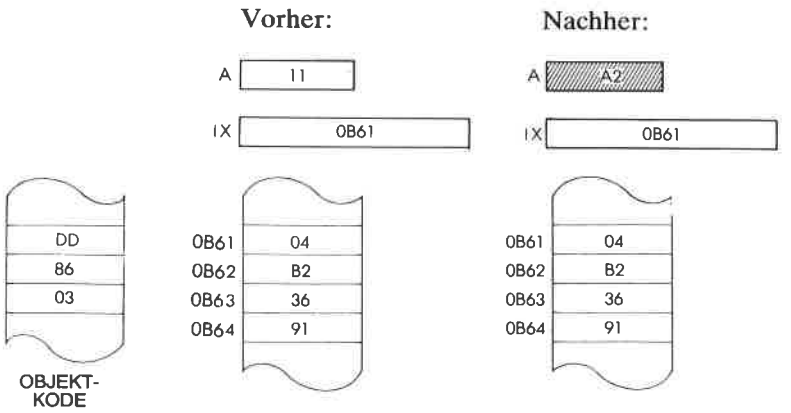



*Abb. 7.9: Ausschnitt aus der Z80-Befehlsliste für den Befehl ADD A, (IX+d) (Forts.)*

Für den Programmierer sind im allgemeinen Detailkenntnisse über die binäre Struktur der Maschinenbefehle nicht unbedingt erforderlich, da er sich für die Programmerstellung üblicherweise eines speziellen Übersetzerprogramms bedienen wird, das ihm die Formulierung des Programmtextes in einer einfachen symbolischen Sprache erlaubt. Übersetzerprogramme, die diese Aufgabe übernehmen, werden als *Assembler* bezeichnet. Sie sind für den Schneider CPC sowohl auf Kassette als auch auf Diskette im Handel erhältlich. Wir wollen im nachfolgenden Abschnitt dieses Kapitels kurz die Arbeitsweise eines Assemblers kennenlernen, damit Ihnen das Lesen der manchmal nicht besonders informativ geschriebenen Dokumentationen der Softwarehersteller etwas leichter fällt.

### Von der Arbeitsweise eines Assemblers

Da die Erstellung von Programmen, deren Elemente nur aus einer Folge binär codierter Daten- und Befehlsworte bestehen, eine sehr mühsame und darüber hinaus auch fehlerträchtige Angelegenheit ist, gibt es für die alltägliche Programmierarbeit – wie Sie bereits in Kapitel 3 erfahren haben – Übersetzerprogramme in Form von Interpretern oder Compilern, die es Ihnen als Anwender gestatten, Programme in einer *höheren* Programmiersprache zu erstellen. Derartige Übersetzerprogramme überführen die nach vorgeschriebenen Sprachregeln formulierten Anweisungen in für den Prozessor verständliche Folgen von Binärcodes um.

Kein noch so gutes Übersetzerprogramm für eine höhere Programmiersprache kann allerdings mit vernünftigem Maß an Aufwand ein Maschinenprogramm mit einer kürzestmöglichen Anzahl von Befehlsworten erzeugen. Die mittels Übersetzerprogrammen erzeugten Maschinenroutinen (sie werden häufig als *Objektprogramme* bezeichnet) weisen daher immer ein gewisses Maß an *Redundanz* auf, d. h. es werden mehr Maschinenbefehle erzeugt, als bei korrekter Analyse des Programms für den Prozessor zur Durchführung einer Operation notwendig sind. Das ist so ähnlich, als wenn man bei einer Wanderung erst über Umwege an das gewünschte Ziel gelangt, anstatt den direkten und kürzesten Weg zu wählen. Von erfahrenen Programmierern direkt in Maschinensprache geschriebene Programme oder Programmteile sind bei sorgfältiger Analyse der Problemstellung und Umsetzung im allgemeinen kürzer und auch um ein Vielfaches schneller als diejenigen, die von einem Übersetzerprogramm für höhere Programmiersprachen erzeugt werden.

Nun wird sich trotz der Tatsache, daß ein Mikroprozessor nur binäre Befehlsworte versteht, kein Programmierer einer direkten Codierung von Maschinenbefehlen bedienen. Zumindest in der Regel nicht. Für die

Erstellung von Maschinenroutinen stehen nämlich auch geeignete Übersetzerprogramme zur Verfügung, die – wie bereits zuvor erwähnt – als Assembler bezeichnet werden (to assemble = engl. zusammensetzen). Übersetzerprogramme dieses Typs arbeiten mit symbolischen Befehlen, deren wesentliche Teile aus leicht merkbaren Kürzeln bestehen. Sie werden *Mnemonics* genannt. Einige haben Sie bereits im vorangegangenen Abschnitt über die grundsätzliche Form von Maschinenbefehlen kennengelernt.

Wenn beispielsweise eine einfache Ladeoperation durchzuführen ist, enthält der Befehl das Mnemonic LD als Abkürzung für das englische Wort *Load*. Ein Vertauschen von Daten dagegen wird mittels eines Assemblerbefehls durchgeführt, der als Befehlselement das Mnemonic EX (vom engl. Begriff *Ex*change) enthält. Wie sie sehen, sind offensichtlich alle symbolischen Abkürzungen so gewählt, daß sie leicht in Erinnerung zu rufen sind. Die nachfolgende kleine Zusammenstellung von Befehlskürzeln des Z80 verdeutlicht diesen Sachverhalt nochmals anhand einiger aussagekräftiger Beispiele, die Ihnen zum Teil bereits bekannt vorkommen werden (Abb. 7.10).

Symbolische Befehlselemente der nachstehend gezeigten Art lassen sich ohne Zweifel viel einfacher handhaben als die direkten, binär oder hexadezimal angegebenen Codes. Sie sind Bestandteile von Assemblerbefehlen, die vom Übersetzerprogramm beim Übersetzungsvorgang (beim Assemblieren) direkt in maschinenlesbare Codes umgesetzt werden.

| Assembler-Mnemonic | engl. Bedeutung | deutsche Bedeutung |
|---|---|---|
| ADC | *AD*D with *C*arry | Addiere mit Übertrag |
| ADD | *ADD* | Addiere |
| AND | *AND* | UND-Operation |
| CPD | *C*ompare and *d*ecrement | Vergleiche und erniedrige um Eins |
| CPI | *C*ompare and *i*ncrement | Vergleiche und erhöhe um Eins |

*Abb. 7.10: Auszug symbolischer Assemblerbefehle für den Z80*

Leider reicht die Kenntnis der Befehlssymbole allein nicht aus, um erfolgreich mit Hilfe dieses Befehlstyps umgehen zu können. Ein vollständiger Assemblerbefehl besteht im allgemeinen nämlich aus mehreren Elementen, wie Sie aus Abb. 7.11 entnehmen können.

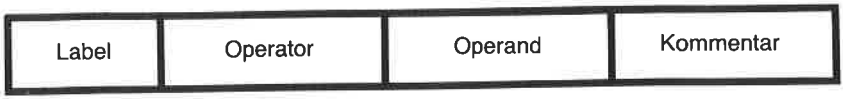



*Abb. 7.11: Prinzipielle Organisation eines Assemblerbefehls*

Dem symbolischen Befehl folgen nämlich bei Mehr-Byte-Befehlen noch *Operanden*, deren Vereinbarungen in Einzelfällen recht kompliziert aussehen können. Sie können, wie Sie bereits aus dem vorangegangenen Abschnitt wissen, Daten, Registercodes, Adressen oder auch Adreßoffsets enthalten.

Im Programmtext selbst kommen unter Umständen noch sogenannte *Marken* hinzu, die im Fachjargon auch *Labels* genannt werden. Darüber hinaus werden gegebenenfalls zusätzlich noch Kommentare vereinbart, die nichts anderes darstellen als Kurzerläuterungen von Befehlen. Sie dienen dem Programmierer als Gedächtnisstütze bei der Programmpflege, der Fehlerkorrektur oder der Dokumentation für andere Anwender.

Die bereits erwähnten Marken oder Labels erleichtern den Ansprung von Programmteilen sowie die Vereinbarung von bedingten Sprüngen und Datentabellen. Mit Hilfe eines Assemblers und der durch seine Hilfe möglichen einfachen Symbolik kann der Programmierer sein Programm in einer verhältnismäßig leicht lesbaren und gut korrigierbaren Form formulieren.

Die Phase der Erstellung des Programmtextes (das *Editieren* des *Quellprogramms*) ist bei den weitaus meisten Assemblern von der eigentlichen Übersetzungsphase getrennt. Sie wird unter Kontrolle eines Texterstellungsprogramms durchgeführt, das als *Editor* bezeichnet wird. Im Gegensatz zu Editierprogrammen für umfangreiche Fließtexte arbeitet ein Editor zur Erstellung von Assemblerprogrammen in der Regel zeilenorientiert. Korrektur- und Suchfunktionen erstrecken sich in diesem Fall nur über die Länge einer Programmzeile. Typisch ist auch, daß das Ausgabefeld für Programmzeilen im allgemeinen durch Tabulatorfunktionen strukturiert ist, die dafür Sorge tragen, daß unmittelbar bei der Eingabe des Programmtextes gut lesbare Befehlstexte entstehen (Abb. 7.12).

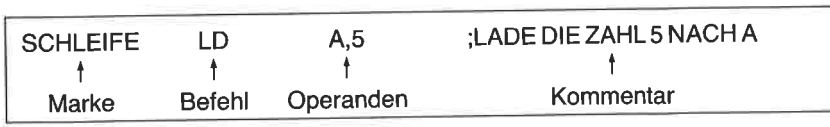



*Abb. 7.12: Beispiel für einen vollständigen Assemblerbefehl*

Die für die Programmerstellung zulässigen symbolischen Anweisungen sind als Bestandteile des Befehlssatzes vom Hersteller für jeden Prozessor eindeutig festgelegt. Beachten Sie bitte, daß von der Logik her ähnliche Befehle bei unterschiedlichen Prozessoren zu recht verschiedenen Mnemonics führen können. Wenn Sie also beispielsweise mit einem 6502, nicht jedoch mit dem Prozessor Z80 des Schneider CPC vertraut sind, müssen Sie sich wohl oder übel erst in die neue Befehlsstruktur und die abweichenden Adressierungsarten dieses Mikroprozessors einarbeiten.

Zur Erleichterung der Programmierarbeit stellt jeder gute Assembler zusätzliche Befehle, sogenannte Pseudobefehle, zur Speicherorganisation beim Übersetzungsvorgang oder bei der Vereinbarung von Datenstrukturen zur Verfügung. Auch diese Befehlstypen können je nach Assemblertyp anders aussehen.

Für den Schneider CPC sind einige recht leistungsfähige Assembler auf Kassette und auf Diskette erhältlich, deren Dokumentation sich allerdings leider in recht engen Grenzen bewegt. Sie sollten sich in jedem Falle noch vertiefende Literatur zulegen, wie beispielsweise das bereits an anderer Stelle genannte, sehr umfangreiche Buch „Programmierung des Z80" von Rodnay Zaks, erschienen ebenfalls bei SYBEX. Es erläutert auf mehr als 500 Seiten alle wichtigen Befehle des Z80 und beschreibt eine Fülle von Programmiertricks, die Sie auch bei Ihrer Arbeit mit dem Schneider CPC sehr gut verwenden können.

Um die in Abb. 7.12 schematisch gezeigte Befehlsorganisation etwas anschaulicher zu gestalten, wollen wir einen Sprung ins kalte Wasser wagen und versuchen, mit den bisher erworbenen allgemeinen Kenntnissen ein kleines Maschinenprogramm zu verstehen, das Sie in Abb. 7.13 als Ausdruck eines Quellprogramms vorfinden. Es besteht nur aus drei Programmzeilen.

```
SCHLEIFE  LD    A,5          ;LADE DIE ZAHL 5 NACH A
WEITER    DEC   A            ;ERNIEDRIGE A UM EINS
          JR    NZ,WEITER    ;STATUSBIT Z ABFRAGEN
                             ;UND RUECKSPRUNG
          RTS                ;ZURUECK INS
                             ;HAUPTPROGRAMM
```

*Abb. 7.13: Assemblerprogramm für eine Zeitverzögerung*

Das Programm stellt nichts weiter dar als eine Verzögerungsschleife. Zu Beginn wird die konstante Zahl 5 in den Akkumulator geladen. Anschlie-

ßend wird der Inhalt des Akkumulators um Eins erniedrigt. Das Programm testet nach dieser Operation, ob der Inhalt des Akkumulators den Wert Null angenommen hat. Hierzu wird nicht der Inhalt von A selbst, sondern der Status des Z-Bits (Zerobit = Nullbit) im Statusregister F abgefragt. Dieses Bit wird nämlich unter anderem immer dann gesetzt, wenn das Ergebnis einer DEC-Operation den Wert Null ergibt. Ist die Bedingung (Z=1) nicht erfüllt (not zero = NZ), springt das Programm an diejenige Stelle zurück, die mit der Marke WEITER versehen ist. Das Label SCHLEIFE dient hier nur als Name für diese Routine. Die Schleife zwischen dem Befehl DEC A und JR NZ,WEITER wird genau fünfmal durchlaufen, weil erst dann der Inhalt des Akkumulators den Wert Null angenommen hat. In diesem Fall wird der zuletzt angegebene Befehl RTS (*Re*turn from *S*ubroutine) ausgeführt. Es wird hier also angenommen, daß dieses kleine Programmodul aus einem aufrufenden Programm heraus als Unterprogramm angesprungen wird, wie dies in vielen Fällen üblich ist.

Natürlich kann der Z80 Ihres Computers mit Anweisungen in der obigen Form noch nichts anfangen. Der Programmtext muß erst durch den Assembler in den Maschinencode übersetzt werden.

Wenn Sie im Anhang L dieses Buchs nachschlagen, werden Sie sehen, daß der Befehlscode für LD A,n als hexadezimal &3E•• angegeben ist. Der Code &3E ist der Befehlscode, •• steht als Platzhalter für den aktuell zu ladenden 8-bit-Wert, hier also die Zahl 5 (dual 000 0101). Der Befehlscode für DEC A lautet entsprechend &3D, der für JR NZ, Adreßoffset lautet &30••. Der im Folgebyte anzugebende Adreßoffset gibt die Sprungweite an. Sie wird aus den aktuellen Daten der Labels vom Assembler bei der Übersetzung automatisch richtig berechnet. Der Vorteil der symbolischen Schreibweise gegenüber der direkten Codierung im Binärcode wird hier ganz besonders deutlich, weil der Programmierer sich um die recht mühsame Berechnung der Sprungweite nicht zu kümmern braucht. Bei der unmittelbaren Verschlüsselung müßte der Adreßoffset nämlich durch den Programmierer selbst errechnet werden. Das ist im einzelnen gar nicht so einfach, weil der Prozessor eine Vereinbarung der durch den Adreßoffset gegebenen Sprungweite im *Zweierkomplement* verlangt. Hierdurch wird es möglich, Sprünge mit Weiten zwischen −128 und + 127 durchzuführen, d. h. vorwärts und rückwärts zu springen. Falls Sie unsicher bezüglich der Darstellung von Dualzahlen im Zweierkomplement sind, dann schlagen Sie bitte nochmals in Kapitel 6 nach. Dort wurde die Zweierkomplementdarstellung von Dualzahlen bereits eingehend erläutert. Wie dies im Zusammenhang mit dem JR NZ-Befehl konkret funktioniert, werden Sie etwas später noch an einem Beispiel sehen.

Für den letzten Maschinenbefehl RTS finden Sie in der Befehlstabelle zum Schluß noch den Code &C9.

Die Zusammenstellung der aktuellen Codefolge reicht immer noch nicht, um ein ablauffähiges Programm zu erhalten. Der Computer benötigt natürlich neben den Befehlen und Daten noch genaue Angaben darüber, *wohin* er das übersetzte Programm im RAM abspeichern soll. Im Gegensatz zur Programmentwicklung in BASIC werden bei einem Assembler keine Standardannahmen über die Lage des Programmcodes im Speicher vereinbart. Hier muß immer der Programmierer exakt angeben, wo die Befehlsfolge für das Maschinenprogramm abgelegt werden soll.

Voraussetzung dafür, daß das Programm dann nach der Übersetzung und der Abspeicherung im RAM nicht durch das Betriebssystem oder durch Programmteile von BASIC überschrieben wird, ist, daß Sie sich als Programmentwickler sehr genau über die Speicherbelegung des CPC in den unterschiedlichen Betriebsarten auskennen. Grundsätzliche Informationen darüber haben Sie bereits in Kapitel 7 gelesen.

Die Vereinbarung der aktuellen Startadresse eines Maschinenprogramms und damit die Festlegung der Ortslage im Speicher geschieht bei einem Assembler immer durch sogenannte Pseudobefehle. Hierbei handelt es sich um assemblertypische symbolische Vereinbarungen, die wichtige, für die Übersetzung benötigte Randinformationen enthalten. Ein typischer (Pseudo-)Befehl zur Vereinbarung der Startadresse eines Programms ist beispielsweise

```
ORG adr
```

Als Parameter adr wird jene Speicheradresse angegeben, bei der das Maschinenprogramm beginnen soll.

Sie können, wie Sie bereits in der Übersicht über die BASIC-Befehle erfahren haben, Bereiche im Arbeitsspeicher vor einem Zugriff des Interpreters durch den MEMORY-Befehl schützen. Wie Sie aus Kapitel 4 bzw. 6 wissen, beginnt bei der Adresse &B100 jener Teil des Schreib-/Lesespeichers, der die Grafik- und Textinformationen sowie wichtige Daten des Betriebssystems enthält. Sie sollten diese nicht durch Maschinenprogramme überschreiben. Damit im Speicher abgelegte BASIC-Programme andererseits Ihre Maschinenroutinen nicht überschreiben, sollten Sie sich für Experimente einen kleinen Speicherbereich durch den MEMORY-Befehl schützen. Vereinbaren Sie beispielsweise MEMORY &9FF, dann können Sie ab &A000 selbstgeschriebene kurze Maschinen-

programme ablegen und zur Probe ablaufen lassen. Vor der Übersetzung wird diese Adresse dem Assembler mit der Programmzeile

```
.ORG &A000
```

mitgeteilt. Falls wir uns darauf einigen, daß das kleine Programm beginnend bei dieser Adresse im Speicher abgelegt werden soll, wird der Assembler nach dem Aufruf eine Speicherbelegung errechnen, wie sie in Abb. 7.14 als sogenanntes Assemblerlisting angegeben ist.

| Adresse | Inhalt | symbolischer Befehl |
|---|---|---|
| A0 00 | 3E 05 | LD A,5 |
| A0 02 | 3D | DEC A |
| A0 03 | 20 FD | JR NZ,−3 |
| A0 05 | C9 | RET |

*Abb. 7.14: Assemblerlisting mit Objektcode für das Programm nach Abb. 7.13*

Die Abb. 7.15 zeigt schematisch, wie das Programm als Folge von insgesamt 7 Bytes im Speicher abgelegt wird.

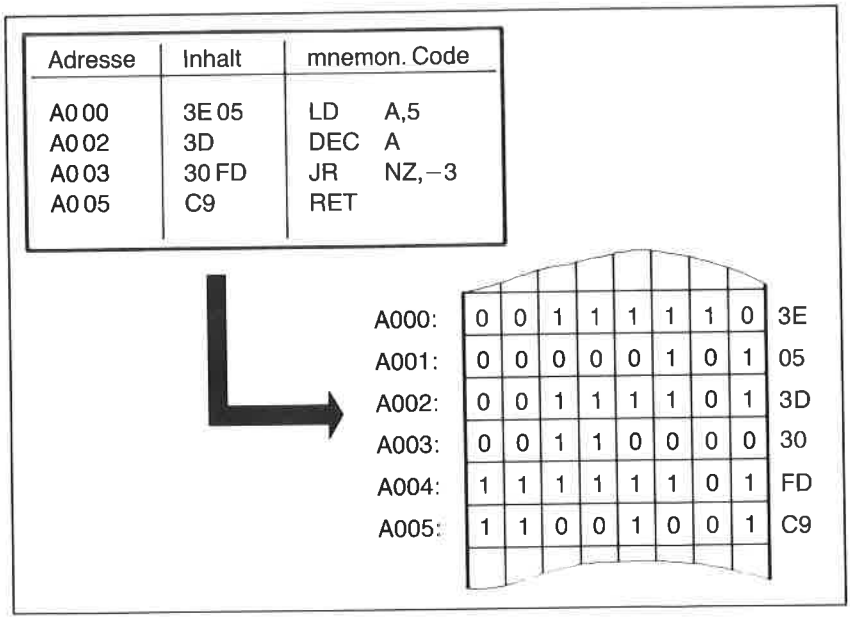



*Abb. 7.15: Speicherorganisation des Programmes nach Abb. 7.14*

Bevor wir uns ein für den Schneider CPC interessanteres Beispiel für den Einsatz von Maschinenspracheprogrammen ansehen, sollten Sie Ihr Augenmerk noch einmal auf den Befehl JR NZ,−3 richten, der noch einer etwas eingehenderen Erläuterung bedarf. Von der Adresse &A0 03 zur Adresse &A0 02 zurück beträgt der Sprung augenscheinlich nur −1 und nicht, wie oben angegeben, −3. Die Sprungweite orientiert sich jedoch immer an dem aktuellen Wert des PC (Programmzählers). Dieser verweist aber zum Zeitpunkt der Decodierung des bedingten Sprungs bereits auf den nächstfolgend abzurufenden Befehl. Das bedeutet, daß der Zähler bereits um zwei Schritte weiter ist.

Von der Sprungweite ist demnach immer der Wert 2 abzuziehen. Bei positiven Sprüngen ist daher der effektive Adreßoffset absolut um den Wert 2 kleiner, bei negativen natürlich absolut um den Wert 2 größer, denn −1−2 ergibt ja −3. Da das duale 8-bit-Datenwort für eine 3 der Bitfolge 0000 0011 entspricht, errechnet sich dessen Zweierkomplement zu 1111 1101. Dies entspricht der hexadezimalen Zahl &FD, die Sie im Listing als Adreßoffset wiederfinden (Abb. 7.16).

Wohlgemerkt, der Assembler errechnet diese Größen automatisch, d. h. Sie brauchen sich weder um die Sprungweiten, noch um die Bildung von Zweierkomplementen zu kümmern. Bei der Durchsicht eines Assemblerlistings ist es allerdings in vielen Fällen sehr nützlich, wenn einem der

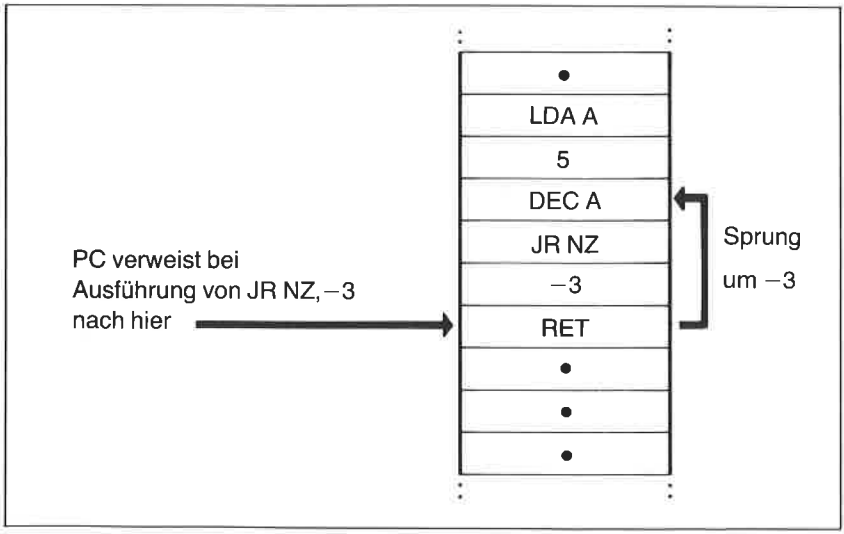



*Abb. 7.16: Zur Erläuterung des negativen Adreßoffsets im Verzögerungsprogramm*

zuvor erläuterte Sachverhalt vertraut ist. Wenn Sie also in einem Assemblerlisting einen Adreßoffset von &F3 entdecken, dann können Sie leicht durch Bildung des Zweierkomplements und Addition der Zahl 2 die wahre Sprungweite errechnen. Die Hexadezimal &F3 entspricht einer dualen 1111 0011 in der Zweierkomplementform. Der zugehörige absolute duale Wert ist folglich 0000 1101 , d. h. hex. &D bzw. dezimal 13. Der angegebene Adreßoffset von &F3 entspricht somit einem Sprung von der Adresse des Sprungbefehls aus zurück um 13 + 2 = 15 Speicherzellen.

### Der Zeitbedarf von Maschinenprogrammen

Sofern das kleine, bewußt einfach gewählte Programm einen unmittelbaren praktischen Nutzen für Sie hätte, könnten Sie es übrigens auch dann eingeben und starten, wenn Sie gar nicht im Besitz eines geeigneten Assemblerprogramms sein sollten. Von der BASIC-Ebene aus geht es nämlich auch mittels des POKE-Befehls. Sie können die Speicherzellen entweder unmittelbar über die Tastatur oder über ein kleines BASIC-Programm mit den entsprechenden Werten füllen. In Programmform sähe das Ganze dann so aus:

```
10 MEMORY &9FF
20 FOR I% = &A000 TO &AB05
30 READ CW
40 NEXT I%
50 DATA &3E,&05,&3D,&30,$FD,&C9
60 CALL &AB00
70 END
```

*Abb. 7.17: BASIC-Ladeprogramm für die Maschinenroutine nach Abb. 7.14*

Sie sollten es nicht eingeben. Denn Sie werden nach dem Start des Programms keine merkbare Reaktion verspüren. Die Frage nach dem „Warum" ist leicht beantwortet: Es läuft so schnell ab, daß Sie die erreichte Verzögerung nicht wahrnehmen können. Bleibt immer noch die interessante Frage, wie groß denn nun eigentlich die durch das kleine Programm erzielte Zeitverzögerung ist. Auch die Antwort auf diese Frage ist für einen erfahrenen Programmierer exakt zu beantworten. Jede gute Befehlsliste enthält nämlich neben den Angaben über die Syntax und die Funktion von Befehlen auch noch genaue Angaben über deren Ausführungsdauer. Schlagen Sie zur Kontrolle nochmals die Abb. 7.9 auf, in der die Befehlsausführungszeiten für den Befehl ADD A, (IX+d) angegeben sind.

| Befehl | Masch.-Zyklen | Takt Zustände | Dauer in µs bei 2 MHz | Bemerkung |
|---|---|---|---|---|
| LD A | 2 | 7 | 3.5 | |
| DEC A | 1 | 4 | 2.0 | |
| JR NZ,e | 2 | 7 | 3.5 | (Bedingung nicht erfüllt) |
| JR NZ,e | 3 | 12 | 6.0 | (Bedingung erfüllt) |
| RET | 3 | 10 | 5.0 | |

*Abb. 7.18: Zeitbedarf für die Befehlsausführung*

Für die in unserem kleinen Programm verwendeten Befehle sieht die Bilanz aus, wie in Abb. 7.18 angegeben. (Alle Werte wurden der Befehlsübersicht in dem Buch „Programmierung des Z80" von R. Zaks entnommen.)

Um die Gesamtablaufzeit des Verzögerungsprogramms zu berechnen, muß berücksichtigt werden, daß die Schleife insgesamt viermal durchlaufen wird, weil die durch JR NZ,e getestete Bedingung erfüllt ist. Erst nach dem letzten (fünften) Durchlauf führt die Nichterfüllung der Bedingung zu dem Rücksprung über RET.

Die Gesamtzeit beträgt also

$$T_{\text{Verzögerung}} = \underset{\text{LDA}}{3.5} + 4*\underset{\text{Schleife}}{(2.0+6.0)} + 1*\underset{\text{Schleife}}{(2.0+3.5)} + \underset{\text{RET}}{5.0} = 46\ \mu s.$$

Das sind bei einer Taktrate von 2 MHz nur 46 millionstel Sekunden. Da der CPC eine effektive Taktrate von 3,3 MHz besitzt, erreichen wir sogar einen Wert von nur 27,9 µs. Für uns Menschen ist das eine unvorstellbar kleine und daher nicht wahrnehmbare Zeitspanne. Es lohnt sich deshalb nicht, die Maschinenroutine auszuprobieren. Auch eine Erhöhung des Anfangswertes von 5 auf den für 8-bit-Worte maximalen Wert von 255 bringt da nichts.

Um in einem so raschen Durcheilen der Welt der Programmierung ein besseres Gefühl für die Effektivität von Maschinenprogrammen zu gewinnen, wollen wir uns deshalb nachfolgend noch ein anderes Programm ansehen, mit dessen Hilfe wir den Bildschirm mit einer einheitlichen Farbe füllen können.

Damit Sie einen Zeitvergleich durchführen können, wollen wir uns zunächst ein einfaches BASIC-Programm schreiben, das sich im Prinzip

desselben Algorithmus bedient wie das dann erläuterte Maschinenprogramm. Beide Programme füllen den für die Bildschirmdarstellung benutzten Speicherbereich zwischen den Adressen &C000 und &FFCF (siehe auch Kapitel 2 und Kapitel 4) direkt mit dem Wert 255 (&FF). Im Grafikmodus 1 führt dies zur Ausgabe roter Striche. Das BASIC-Programm lautet:

```
10 MODE 1
20 FOR ADR% = &C000 TO &FFCF
30 POKE ADR%,DT%
40 NEXT ADR%
50 END
```

*Abb. 7.19: Einfärben des Bildschirmfensters (BASIC-Routine)*

Geben Sie das Programm ein, starten Sie es und stoppen Sie die Zeit, bis die Meldung READY erscheint. Sie werden feststellen, daß das Bildschirmfenster zeilenweise rot eingefärbt wird. Ein vollständiger Durchlauf benötigt etwa 25 Sekunden.

Wir wollen uns nun davon überzeugen, daß ein Maschinenprogramm, das sich des gleichen Algorithmus bedient, um ein Beträchtliches schneller abläuft. Ein einfaches, wenn auch in den Augen eines erfahrenen Programmierers sicherlich nicht ganz optimiertes Programm in Maschinensprache ist in Abb. 7.20 angegeben.

```
4000  01 CF FF START  LD    BC,FFCF ;ENDADRESSE
4003  21 00 C0        LD    HL,C000 ;ANFANGSADRESSE
4006  36 FF    NEXT  LD    (HL),FF ;GRAFIKBYTE &FF
4008  23              INC   HL      ;ADRESSE ERHOEHEN
4009  E5              PUSH  HL      ;WERT RETTEN
400A  ED 42           SBC   HL,BC   ;ADRESSOFFSET?
400B  E1              POP   HL      ;ADRESSE ABRUFEN
400C  20 F7           JR    NZ,NEXT ;SCHLEIFENABFRAGE
400E  C9              RET           ;ZURUECK ZU BASIC
```

*Abb. 7.20: Assemblerlisting*

Mit etwas Konzentration ist das Programm mit den bisher von Ihnen erworbenen Kenntnissen leicht zu durchschauen. Es bewirkt nämlich nichts anderes, als daß die einzelnen Zellen des Bildwiederholspeichers zwischen den Adressen &C000 und &FFCF mit dem Hexadezimalwert

FF gefüllt werden. Zunächst wird die höchste Bildschirmadresse &FFCF in dem Registerpaar BC abgelegt. Entsprechend den Regeln wird nach dem Befehlscode (&01) zunächst das niederwertige Byte dieser Adresse (&CF) und dann das höchstwertige Byte (&FF) angegeben. Die Anfangsadresse des Bildwiederholspeichers ist normalerweise &C000. Sie wird in dem Registerpaar HL abgespeichert. Mittels des nachfolgenden Befehls LD (HL),FF wird das Bitmuster 1111 1111 (&FF) in die Speicherzelle eingeschrieben, deren Adresse in HL enthalten ist. Dieser *indirekte* Zugriff auf eine Speicherzelle wird in der Assemblersymbolik dadurch gekennzeichnet, daß HL in Klammern gesetzt wird. Zunächst wird also in &C000 der Wert &FF eingeschrieben. Diese Operation führt genauso wie in dem vorangegangenen BASIC-Programm dazu, daß ein roter Strich von acht Bildpunkten Breite in der oberen linken Ecke des Bildschirms abgebildet wird. Über INC HL wird die Zieladresse dann um den Wert Eins erhöht, so daß nach dessen Ausführung in HL der Wert &C001 steht.

Um festzustellen, ob nach dieser Adreßerhöhung bereits das Ende des Bildwiederholspeichers erreicht ist, wird über den Befehl SBC HL,BC der in BC stehende Endadreßwert (&FFCF) des Speichers von der neuen aktuellen Ausgabeadresse abgezogen. Das Ergebnis dieser Subtraktionsoperation wird in HL abgelegt. Das bedeutet, daß der ursprüngliche, für uns wichtige Inhalt von HL verloren geht. Aus diesem Grunde retten wir ihn vor der Subtraktionsoperation auf den Stapel. Erreicht wird dies durch den Befehl PUSH HL.

Das Ergebnis der Subtraktion beeinflußt die Statusbits S, Z, H, P/V und C. Wegen des späteren bedingten Sprungs interessieren wir uns an dieser Stelle nur für das Verhalten des Z-Bits. Da das Ergebnis der Subtraktion so lange nicht Null ist, wie die aktuelle Adresse von der Endadresse in BC abweicht, wird das Z-Bit erst beim letzten Programmschritt gesetzt (&FFCF = &FFCF). Bevor der bedingte Sprung ausgeführt werden kann, muß natürlich der ursprüngliche Inhalt von HL mit der aktuell zu füllenden Adresse wiederhergestellt werden. Dies erreichen wir durch den Befehl POP HL, der die auf den Stapel gerettete Adresse wieder nach HL lädt. Der nachfolgende relative Sprungbefehl JR NZ,NEXT (Springe zurück zu NEXT, wenn das Ergebnis [der Subtraktion] verschieden von Null ist) führt so lange zu einem erneuten Schleifendurchlauf, bis die Endadresse des Bildwiederholspeichers erreicht ist. (Diese wird also nicht mehr berücksichtigt.) Das Programm kehrt dann wegen des Befehls RET (Return from Subroutine) wieder in das aufrufende Programm zurück.

Es ist sehr einfach, das Programm unter Kontrolle des Interpreters in den Speicher zu laden und es anschließend von der Interpreterebene aus zu

starten. Ein BASIC-Programm dieses Typs wird häufig auch als BASIC-Lader bezeichnet. Das entsprechende Programm sieht so aus:

```
10 MODE 1
20 MEMORY &3FFF
30 ADR%=&4000
40 FOR I% = 0 TO 15
50 READ DT%
60 POKE ADR%+I%,DT%
70 NEXT I%
80 END
90 DATA &01,&CF,&FF,&21,&00,&C0,&36,
&FF,&23,&E5,&ED,&42,&EL,&20,&F7,&C9
```

*Abb. 7.21: Ladeprogramm in BASIC*

Der Befehl MODE 1 stellt sicher, daß &C000 als erste legale Adresse des Bildwiederholspeichers gültig ist. Mittels des Befehls MEMORY &3FFF wird dem Interpreter der Zugriff auf die Speicherstellen oberhalb von &3FFF verwehrt und damit das Maschinenprogramm vor Überschreiben durch Daten von BASIC-Programmen geschützt. Wir können somit unser Programm beginnend ab &4000 im Speicher ablegen.

Starten Sie nach der Eingabe das Programm und rufen Sie anschließend durch den unmittelbar über die Tastatur eingegebenen Befehl CALL &4000 die Maschinenroutine auf. Haben Sie die Zeit gestoppt? Vermutlich nicht, denn der Bildschirmaufbau dauert nunmehr weniger als eine Sekunde. Offensichtlich läuft der Algorithmus in Maschinensprache um ein Vielfaches schneller als unter Interpreterkontrolle. Ganz genau berechnet benötigt das Programm in der vorliegenden Form 1.045.465 Taktzyklen (sofern sich die Autoren dieses Buchs nicht verrechnet haben). Geht man von einer effektiven Taktrate von 3,3 MHz beim Schneider CPC aus, dann benötigt das Programm eine Zeit von etwa 0,3 Sekunden. Es ist somit etwa 78mal schneller als das zunächst verwendete BASIC-Programm.

Falls Sie nach diesen Ausführungen auf den Geschmack gekommen sind und sich dazu entschließen sollten, eigene Programme im Maschinencode zu entwerfen, so sollten Sie sich zunächst anhand weiterführender Literatur eingehend in den vollständigen Befehlssatz des Z80 einarbeiten. Viele nützliche Programme, die Sie sich für die Arbeit mit Ihrem Computer mühsam erarbeiten müssen, stehen Ihnen im übrigen abrufbereit zur Ver-

fügung, wenn Sie sich eingehender mit der *Firmware* des Systems beschäftigen. Unter Firmware versteht man die im Festwertspeicher (ROM) abgelegten Programme, also das Betriebssystem wie auch den Interpreter selbst. Die Firma Schneider bietet eine vollständige, aber leider nicht in allen Teilen korrekt ausgeführte Übersetzung des originalen englischen Firmwaremanuals, in dem Sie alle interessanten Informationen über nützliche Maschinenroutinen sowie deren Einsprungadressen finden. Einige davon finden Sie im Anhang O dieses Buchs mit kleinen Hinweisen für deren Anwendung.

Bis zu diesem Punkt haben Sie einige interessante Informationen über die Grundlagen der Programmierung Ihres Schneider CPC in Maschinensprache erhalten. Natürlich werden Sie nach der Lektüre nicht zu den Profis gehören (falls Sie es nicht schon vorher waren). Dennoch sollten Sie nunmehr ohne größere Probleme weiterführende Fachliteratur zu diesem speziellen Themenkreis lesen und sich so das benötigte Wissen aneignen können, um mit Erfolg auch komplizierte und umfangreiche Maschinenprogramme schreiben zu können. Anhand der vollständigen Tabelle des Befehlssatzes mit seinen unterschiedlichen Adressierungsvarianten im Anhang L sollten Sie einfachere Probleme (Programme mit einem Umfang von 10 bis 20 Bytes) vermutlich bereits jetzt „per Hand", d. h. mit der zuvor erläuterten Methode der Formulierung in BASIC lösen können. Falls Sie Ihre Kenntnisse systematisch aufbauen und trainieren möchten, sollten Sie sich eines Assemblerkurses für den CPC bedienen. Empfohlen werden kann für Einsteiger in die Thematik der in der Reihe Mister Micro bei SYBEX erschienene Kurs, als dessen Bestandteil ein einfacher, aber recht leistungsfähiger Assembler/Editor mitgeliefert wird. Große Programme sind komfortabel nur mit Hilfe eines professionellen und im allgemeinen recht teuren *Editors/Assemblers* sowie eines guten *Debuggers* (Programm zur Fehlersuche auf Maschinenebene) vernünftig zu entwickeln. Informieren Sie sich vor dem Kauf eines entsprechenden Programmpaketes sehr sorgfältig über dessen Leistungsfähigkeit. Wie in vielen anderen Bereichen der Mikrocomputertechnik gilt auch hier: Nicht alles, was teuer ist, ist notwendigerweise auch gut!

Kapitel 8

# Arbeiten mit dem Floppydisk-Laufwerk

## Übersicht

Laden und Abspeichern von Programmen und Daten mit dem Kassettenrecorder ist vom Standpunkt der Preiswürdigkeit her gesehen optimal für den schmalen Geldbeutel und beim Schneider CPC eine recht sichere Angelegenheit. Wesentlich schneller und komfortabler ist die externe Speicherung mittels eines 3-Zoll- oder eines 5,25-Zoll-Laufwerks für Disketten. Von der Firma Schneider ist für den CPC 464 ein entsprechendes Laufwerk unter der Bezeichnung DDI-1 für 3-Zoll-Disketten erhältlich. Beim CPC 664 ist das Diskettenlaufwerk integraler Bestandteil des Systems. Zum Anschluß eines Kassettenrecorders ist deshalb ein gesonderter Ein-/Ausgang auf der Rückseite des Gehäuses vorhanden (siehe auch Kapitel 7, Seite 37 des CPC 664 Benutzerhandbuches).

Mitgeliefert wird neben dem diskettenorientierten Betriebssystem AMSDOS als Bestandteil des im Interface enthaltenen ROM noch auf Diskette das für den CPC angepaßte Betriebssystem CP/M 2.2B der Softwarefirma Digital Research sowie die Programmiersprache DR. LOGO. (DR. ist hier kein akademischer Titel, sondern eine Abkürzung für die Herstellerfirma Digital Research.) In dem vorliegenden Kapitel lesen Sie alles Wissenswerte für Ihre Arbeit mit dem Diskettenlaufwerk. Außerdem erhalten Sie eine Fülle interessanter Informationen zu den erwähnten Betriebssystemen und einige Tips für den Umgang mit speziellen Hilfsprogrammen. Die zuvor erwähnte Programmiersprache DR. LOGO wird in diesem Buch nicht behandelt. Hierfür gibt es am Markt spezielle Lehrbücher, die Sie sich gegebenenfalls für eine Auseinandersetzung mit LOGO zulegen sollten.

## Die Technik von Massenspeichern

Im Gegensatz zu der Aufzeichnung binärer Daten auf Magnetband mittels eines analogen Kassettenrecorders werden die Signale auf eine Dis-

kette digital aufgezeichnet. Bei einer Diskette handelt es sich um einen Magnetdatenträger in Form einer kleinen flexiblen Folie, die in eine schlagfeste Hülle aus Hartkunststoff eingebettet ist. Der Schneider CPC macht von sogenannten 3-Zoll-Disketten Gebrauch. Sie werden als Compact-Disk bezeichnet.

Diese Datenträger können digitale Informationen beidseitig speichern. Im Laufwerk werden sie bei Bedarf gewendet. Jede der magnetbeschichteten Seiten der Folie hat eine Speicherkapazität von bis zu 180 kbyte. Das sind 180*1024 = 184.320 Bytes.

Während eines Schreib- oder Lesevorgangs wird die Magnetfolie in schnelle Drehungen versetzt (300 U/Min). Der Schreib-/Lesekopf des Laufwerks greift radial durch den dafür vorgesehenen (und im Betrieb durch eine Mechanik geöffneten) Schlitz auf die Informationen zu. Der Kopf liegt während des Betriebs bündig auf der Oberfläche auf. Das bedeutet, daß die Schutzschicht der Magnetfolie einem andauernden Verschleiß ausgesetzt ist. Obgleich wegen der außergewöhnlich glatten Kopfoberfläche der Abrieb minimal zu sein scheint, ist die Lebensdauer einer Diskette nicht unendlich groß. Sie sollten daher grundsätzlich von allen wichtigen Disketten sogenannte Sicherungskopien (auch Backup-Kopien genannt) anfertigen. Bitte beachten Sie, daß das Laufwerk immer *vor* dem Computer einzuschalten ist. Das Betriebssystem testet nämlich beim Einschalten des Computers, ob ein aktives Laufwerk am Erweiterungsstecker angeschlossen ist. Nur in diesem Fall ergänzt es seinen Befehlsvorrat um einige massenspeicherorientierte Kommandos, die Sie später noch näher kennenlernen werden.

### *Datenorganisation von Disketten*

Damit der Computer gezielt auf die abgespeicherten Informationen einer Diskette zugreifen kann, muß die Magnetschicht des Datenträgers eine Positionsinformation für den Schreib-/Lese-Mechanismus enthalten. Disketten werden zu diesem Zweck mit einer magnetischen Kennung versehen, die aus 40 parallel verlaufenden Spuren (0 bis 39) und 9 Sektoren pro Spur besteht. Jeder Sektor hat eine physikalische Speicherkapazität von 512 byte. Die Anzahl der Einträge in ein Inhaltsverzeichnis ist auf 64 beschränkt.

### *Formatieren von Disketten*

Im Geschäft erworbene *Leerdisketten* sind nicht ohne Vorbereitung vom Computer lesbar. Vor dem ersten Gebrauch müssen die zuvor erwähnten

magnetischen Marken aufgeschrieben werden, die später während des Betriebs als Kennung für die aktuelle Ortslage der Daten- und Programminformationen dienen. Ein Programm, das diesen Vorgang automatisch ausführt, nennt man *Formatierungsprogramm*. Die Betriebssystemerweiterung im ROM enthält keine unmittelbare Möglichkeit zur Formatierung einer Diskette. Sie müssen hierzu zunächst den mitgelieferten Datenträger mit dem Betriebssystem CPM (Seite A der dem Laufwerk beiliegenden Systemdiskette) in das Laufwerk einlegen und über das Kommando

```
|CPM
```

das Betriebssystem CP/M aufrufen. Es meldet sich nach wenigen Sekunden mit der Mitteilung:

```
CPM 2.2 – Amstrad Consumer Electronics plc.
A>
```

auf dem Bildschirm. Die Umschaltung auf 80 Zeichen pro Zeile und eine Darstellung der Schrift auf weißem Hintergrund erfolgt automatisch. Überzeugen Sie sich zunächst davon, daß sich das für den Formatierungsvorgang benötigte Hilfsprogramm mit dem Namen FORMAT.COM auf der Diskette befindet. Hierzu lassen Sie sich einfach ein Inhaltsverzeichnis ausgeben, das Sie mit dem Kommando

```
DIR <ENTER>
```

abrufen können. Es sollte wie folgt aussehen:

```
A>dir
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB      COM
A: ED       COM : ASM      COM : DDT      COM : LOAD      COM
A: STAT     COM : DUMP     COM : DUMP     ASM : AMSDOS    COM
A: FILECOPY COM : SYSGEN   COM : BOOTGEN  COM : COPYDISC  COM
A: CHKDISC  COM : DISCCOPY COM : DISCCHK  COM : SETUP     COM
A: FORMAT   COM : CSAVE    COM : CLOAD    COM : EX1       BAS
A: EX2      BAS : ROINTIME DEM : PARAM    DAT : FLAGDUMP  SRN
A: DUMP     PRN : DUMP     HEX
```

Unter dem Betriebssystem CP/M, das Ihnen in einem eigenen Abschnitt noch vorgestellt wird, können Sie alle Programme des Typs <Dateiname>.COM direkt durch Eingabe des Namens aufrufen und ablaufen lassen.

Der Formatierungsbefehl lautet

```
FORMAT [<Datenformat>]
```

Im Parameter <Datenformat> können Sie vereinbaren, für welchen Zweck die zu formatierende Diskette verwendet werden soll. Wenn Sie nicht speziell mit dem Betriebssystem CP/M zu arbeiten gedenken, sollten Sie eine reine Datendiskette herstellen. Sie verhält sich für Sie als Benutzer wie eine Programmkassette, wenn man einmal davon absieht, daß auf jede Datei direkt zugegriffen werden kann und daß dieser Zugriff um ein Vielfaches schneller vonstatten geht als bei dem eingebauten Datenrecorder.

Der Formatierungsbefehl lautet in diesem Fall

```
FORMAT D
```

Nach dem Formatieren steht Ihnen eine Speicherkapazität von 173 kbyte zur Verfügung. Beachten Sie bitte die Kontrollmeldungen und Anweisungen auf dem Bildschirm, damit Sie nicht versehentlich die Systemdiskette löschen. Sie können diese im übrigen dadurch vor einem versehentlichen Überschreiben schützen, daß Sie den kleinen Plastikschieber am Diskettenrand nach außen schieben. Der Formatierungsablauf wird durch Kontrollmeldungen auf dem Bildschirm unterstützt.

Bereiten Sie am besten sofort so viele Disketten vor, wie Sie für Ihre Arbeit mit dem Schneider CPC benötigen. Denken Sie bitte daran, daß Sie die Disketten *beidseitig* mit Informationen versehen können. Sie sollten sie daher auch beidseitig formatieren.

### Das Diskettenbetriebssystem AMSDOS

Wie das Computersystem selbst, benötigt auch die Diskettenstation eine Reihe von Hilfsroutinen zur Unterstützung des Programm- und Datenverkehrs. Diese Sammlung von Hilfs- und Verwaltungsprogrammen wird *Diskettenbetriebssystem* genannt. Der englischsprachige Ausdruck dafür lautet *D*isk *O*perating *S*ystem, dessen Anfangsbuchstaben zu dem Kürzel DOS geführt haben. Da der englische Hersteller des Schneider CPC die Firma Amstrad ist, wird das spezielle Betriebssystem Ihres CPC als AMSDOS bezeichnet.

Dieses Betriebssystem steht Ihnen dann sofort nach dem Einschalten des Computers zur Verfügung, wenn Sie *vorher* das am Erweiterungsstecker

angeschlossene Laufwerk eingeschaltet haben. Für Sie bereits bekannte BASIC-Kommandos wie beispielsweise CAT, SAVE oder LOAD beziehen sich nun unmittelbar auf das Diskettenlaufwerk. Dennoch geht Ihnen für Ihre alten Programme der Kassettenrecorder nicht gänzlich verloren. Sie müssen jetzt vielmehr immer genau angeben, ob Sie Ihren Datenverkehr mit dem Recorder oder dem Diskettenlaufwerk abwickeln wollen.

### *Die internen AMSDOS-Kommandos*

Folgende Befehle wirken in gleicher Weise wie auch beim Kassettenbetrieb. Sie finden diese in der Liste der BASIC-Kommandos im Kapitel 3:

*Dateneingabe von Diskette:*

CHAIN, CHAIN MERGE, CLOSEIN, EOF, INPUT #9, LINE INPUT #9, LOAD, OPENIN, RUN

*Datenausgabe auf Diskette:*

CLOSEOUT, LIST #9, OPENOUT, PRINT #9, WRITE #9, SAVE

*Inhaltsverzeichnis der Diskette:*

CAT

Neben diesen bereits vertrauten Befehlen stellt AMSDOS noch eine Reihe weiterer Kommandos zur Verfügung. Sie werden alle mit dem Zeichen | eingeleitet.

### *Die externen AMSDOS-Kommandos*

**|A und |B:** An den Diskettencontroller des Schneider CPC können bis zu zwei Laufwerke angeschlossen werden. Diese haben die logischen Bezeichnungen A und B. Beim Einschalten des Systems wird immer zuerst das Laufwerk A aktiviert (Standardlaufwerk). Über |B kann das Laufwerk B als Standardlaufwerk vereinbart werden. Der Befehl |A setzt wieder auf A zurück.

**|CPM** Mit diesem Befehl wird auf das diskettenorientierte Betriebssystem CP/M umgeschaltet. Voraussetzung ist, daß sich beim Aufruf eine Systemdiskette im Laufwerk A befindet.

|DIR

Mit diesem Befehl (*DIR*ectory) wird ein Inhaltsverzeichnis vom aktiven Laufwerk abgerufen. Im Gegensatz zum CAT-Befehl erlaubt der |DIR-Befehl eine Auswahl der Dateitypen und oder -namen. Allgemein lautet der Befehl:

```
|DIR [,<Laufwerk>:]
```

Beispiel für den CPC 464:

```
D$="*.TXT"
|DIR,@D$
```

Beispiel für den CPC 664:

```
|DIR,"*.TXT"
```

Die Angabe von *EN.* führt damit zur Ausgabe aller Elemente des Inhaltsverzeichnisses, deren Namen auf EN enden.

|DISC

Dieses Kommando schaltet auf Diskettenein-/ausgabe, wenn zuvor durch |TAPE der Recorderbetrieb angewählt wurde.

|DISC.IN

Nach diesem Kommando wirken alle *Eingabe*befehle auf das aktivierte Diskettenlaufwerk.

|DIS.OUT

Nach diesem Kommando wirken alle *Ausgabe*befehle auf das aktivierte Diskettenlaufwerk.

|DRIVE

Dieses Kommando definiert bei zwei angeschlossenen Laufwerken das (aktive) Standardlaufwerk. Die allgemeine Form dieses Befehls lautet:

```
|DRIVE ,<Laufwerk>
```

Der Parameter „Laufwerk" muß eine Zeichenkette sein.

Beispiel für den CPC 464:

```
D$="A"
|DRIVE,@D$
```

Beispiel für den CPC 664:

```
|DRIVE,"A"
```

|ERA

Mit dem Kommando |ERA (*ERA*se) können gezielt Dateien auf der im aktiven Laufwerk befindlichen Diskette gelöscht werden. Die allgemeine Form dieses Kommandos lautet:

```
|ERA,<Dateiname>
```

Beispiel für den CPC 464:

```
DT$="TEST.BAS"
|ERA,@DT$
```

Beispiel für den CPC 664:

```
|ERA, "TEST.BAS"
```

**|REN** Mittels |REN werden auf dem aktiven Datenträger abgespeicherte Dateien umbenannt. Die allgemeine Form dieses Befehls lautet:

```
|REN <Neuer Dateiname>,<Alter Dateiname>
```

Beispiel für den CPC 464:

```
AN$="MONTAG.TXT"
NN$="DIENSTAG.TXT"
|REN, @NN$,@AN$
```

Beispiel für den CPC 664:

```
|REN,"DIENSTAG.TXT","MONTAG.TXT"
```

**|TAPE** Dieses Kommando schaltet auf Kassettenein-/ausgabe, wenn zuvor durch |DISC der Diskettenbetrieb angewählt wurde.

**|TAPE.IN** Nach diesem Kommando wirken alle *Eingabe*befehle auf das aktivierte Kassettenlaufwerk.

**|TAPE.OUT** Nach diesem Kommando wirken alle *Ausgabe*befehle auf das aktivierte Kassettenlaufwerk.

**|USER** Benutzerdefinition. Die allgemeine Form dieses Kommandos lautet:

```
|USER <Nummer>
```

Beispiel:

```
|USER,1
```

## Einführung in das Betriebssystem CP/M

Das Betriebssystem CP/M wurde von der Firma Digital Research ursprünglich für Prozessoren des Typs 8080 entwickelt, die sich nach der Markteinführung rasch als Industriestandard durchzusetzen begannen. Das Kürzel CP/M bedeutet *C*ontrol *P*rogram for *M*icroprocessors. Obgleich der Wert dieses Betriebssystems heutzutage umstritten ist, fand es weltweit Verbreitung, weil zum Zeitpunkt der Markteinführung kein ernstzunehmendes Konkurrenzprodukt verfügbar war.

Unter CP/M, das in einer für den Schneider modifizierten Version 2.2 zur Verfügung steht, ist eine Vielzahl professioneller und daher auch im allgemeinen recht teurer Softwareprodukte lauffähig. Normalerweise sind diese nicht auf 3-Zoll-Disketten, sondern im 5,25-Zoll-Format verfügbar. Es ist somit für diejenigen unter den Lesern, die sich des Betriebssystems CP/M für fertige Anwenderprogramme bedienen wollen, empfehlenswert, als Zweitlaufwerk ein Minidiskettenlaufwerk zu wählen. Derartige Laufwerke sind von verschiedenen Herstellern lieferbar. Eine Gewähr dafür, daß die käuflich erworbene Software dann auf dem Schneider CPC auch lauffähig ist, kann leider nicht gegeben werden. Unter Umständen sind hierzu noch spezielle Anpassungen im Betriebssystem notwendig. Lassen Sie sich auf jeden Fall beim Kauf vorführen, daß Sie die Software auf Ihrer Systemkonfiguration auch betreiben können.

Vorteilhaft ist, daß für CP/M einige sehr leistungsfähige Programmiersprachen verfügbar sind. Hierzu zählen beispielsweise die BASIC-Interpreter MBASIC, BASIC80 und GBASIC, für die es auch geeignete Compiler gibt, oder die Sprachen Fortran und Pascal, die beide Programme im Objektcode mit verhältnismäßig hoher Ablaufgeschwindigkeit erzeugen.

Im Gegensatz zum AMSDOS ist CP/M nicht Bestandteil des Festwertspeichers im Controller. Es muß vielmehr, wie bereits im Zusammenhang mit der Diskettenformatierung beschrieben, von der Startdiskette aufgerufen werden. Unter CP/M lauffähige Disketten besitzen außerdem ein anderes Datenformat. Es wird beim Formatiervorgang durch die Befehle FORMAT oder FORMAT V erzeugt. Im erstgenannten Fall wird sofort beim Formatierungsvorgang eine startfähige CP/M-Diskette angelegt. Im zweiten Fall wird eine reine CP/M-Datendiskette angelegt, die eine etwas höhere Speicherkapazität besitzt.

Eine startfähige Diskette meldet sich nach einem Kaltstart unabhängig von ihrem aktuellen Inhalt immer durch die Meldung

```
CP/M 2.2 – Amstrad Consumer Electronics plc.
A>
```

Die Kennung A> zeigt an, daß das aktive Laufwerk das Laufwerk A ist.

Für das Arbeiten mit CP/M sollten Sie sich folgenden Umstand gut merken: Das Betriebssystem führt Schreibvorgänge nur mit jenen Disketten aus, deren Kennung es zuvor eingelesen hat. Dieser Vorgang wird „log in" genannt. Er wird für das aktivierte Laufwerk immer nach dem Systemstart oder nach einem Programmabbruch durch die Tastenkombination CTRL-C durchgeführt. Für ein eventuell angeschlossenes Laufwerk B:

erfolgt das Einlesen der Diskettenkennung immer nach dem ersten Zugriff, sofern A: bereits bekannt ist. Nach einem „login" meldet sich das System immer mit A> oder B>.

Falls Sie nur ein Laufwerk besitzen, sollten Sie aus dem genannten Grund nach jedem Diskettenwechsel ein CTRL-C eingeben. Andernfalls bricht das Betriebssystem einen eventuell eingeleiteten Schreibzugriff mit der Meldung

```
Bdos Error on <Laufwerk>: R/O
```

CTRL-C zählt zu den Steuerbefehlen, die Ihnen im übernächsten Abschnitt noch vorgestellt werden.

### *Die CP/M-Konsolbefehle*

Wie unter AMSDOS auch können Sie einige Systemkommandos direkt, d. h. ohne Hilfsprogramm auf einem Massenspeicher abrufen. Die wichtigsten Kommandos lauten:

A: Umschalten auf Laufwerk A: als Standardlaufwerk.

B: Umschalten auf Laufwerk B: als Standardlaufwerk.

DIR Aufrufen des Inhaltsverzeichnisses. Als Parameter kann ein Laufwerksname angegeben werden. Die allgemeine Form dieses Kommandos lautet

```
DIR <Laufwerk>:
```

Beispiel:

```
DIR A: Inhaltsverzeichnis von A:
DIR B: Inhaltsverzeichnis von B:
```

ERA ERA löscht Dateien auf dem angegebenen Laufwerk. Die allgemeine Form dieses Kommandos lautet:

```
ERA [<Laufwerk>: ]<Dateiname>
```

Wird kein Laufwerk angegeben, wirkt der Befehl immer auf das aktive Laufwerk.

Beispiel:

```
ERA B: *.*
```

löscht alle Dateien auf der im Laufwerk B: eingelegten Diskette.

```
ERA *.BAS
```

löscht alle Dateien mit der Namenserweiterung BAS auf dem aktiven Laufwerk.

REN

Mit dem Kommando REN werden Dateien auf dem angesprochenen Laufwerk umbenannt. Die allgemeine Form dieses Kommandos lautet:

```
REN [<Laufwerk>:]<Dateiname>
```

Wird das Laufwerk nicht explizit angegeben, wirkt der Befehl auf das aktive Laufwerk.

Beispiel:

```
B:
REN TEST.*
```

löscht alle Dateien mit dem Namen TEST auf dem Laufwerk B.

TYPE

Der Befehl TYPE liest den Inhalt der angegebenen Datei auf dem Bildschirm aus. Sinnvoll ist dies nur bei ASCII-Dateien. Die allgemeine Form dieses Kommandos lautet:

```
TYPE [<Laufwerk>:]<Dateiname>
```

Beispiel:

```
TYPE B:BRIEF:TXT
```

gibt den Inhalt der Textdatei BRIEF.TXT auf dem Bildschirm aus.

### *Control-Codes*

Im Zusammenhang mit direkten Eingaben von der Tastatur sind folgende Control-Codes von Interesse:

CTRL-S

stoppt die Ausgabe auf dem Bildschirm. Fortsetzung durch Betätigen einer beliebigen Taste.

CTRL-C

führt einen Warmstart des Systems durch und liest die Diskettenkennung ein. Während eines Programmlaufs eingegeben, wird ein Programmabbruch ausgelöst.

CTRL-P Alle Ausgaben werden auf einen angeschlossenen Drukker geleitet. Rückschaltung erfolgt durch nochmaliges Betätigen von CTRL-P.

Beispiel:

```
CTRL-P
DIR
CTRL-P
```

gibt den Inhalt der Diskette im aktiven Laufwerk auf den Drucker aus. Anschließend wird wieder auf Bildschirmausgabe zurückgeschaltet.

CTRL-X löscht eine gerade durchgeführte Tastatureingabe, sofern noch nicht die ENTER-Taste betätigt wurde.

### *Die CP/M-Dienstprogramme*

Zum Betriebssystem CP/M gehören einige sehr nützliche Dienstprogramme, die auf der Systemdiskette mitgeliefert werden. Sie besitzen, wie alle unmittelbar ablauffähigen CP/M-Programme, die Namenserweiterung .COM.

Gestartet werden sie einfach durch Eingabe des Dateinamens (ohne oder mit Erweiterung) und Bestätigen durch die ENTER-Taste. Im einzelnen handelt es sich um die folgenden Programme:

PIP.COM PIP ist ein Kürzel für *P*eripheral *I*nterchange *Pro*gram. Das Programm wird in seiner vollständigen Form durch

```
PIP <Ziel> = <Quelle>
```

angegeben. Es dient zum Austausch von Daten zwischen dem Rechner und peripheren Komponenten wie beispielsweise Tastatur, Bildschirm, Diskettenlaufwerke oder Drucker. Die Parameter Ziel oder Quelle enthalten unter Umständen gezielte Angaben über das angesprochene Peripheriegerät sowie Hinweise über die betroffenen Dateien.

Beispiel:

```
PIP B: = A:*.TXT
```

überspielt bei zwei angeschlossenen Laufwerken A und B alle Dateien mit der Namenserweiterung TXT

auf die im Laufwerk B enthaltene Diskette. Das Zeichen * wird als „wild card character" bezeichnet. Es dient als Platzhalter für Textelemente von Namen und Erweiterungen.

DDT.COM — DDT (*D*ynamic *D*ebugging *T*ool) ist ein komfortables Programm zum Auslesen und Ändern von Speicherinhalten auf Maschinenebene sowie zur Ausgabe von Speicherinhalten in symbolischer Form (dieser Vorgang wird Disassemblieren genannt). Die DDT-Kommandos finden Sie im Anhang dieses Buchs.

ED.COM — Das Dienstprogramm ED ist ein zeilenorientierter Texteditor, der vornehmlich zur Erstellung von Assemblerprogrammen entwickelt wurde. Seine Kommandos sind ebenfalls im Anhang zusammengestellt.

DUMP.COM — Das Programm liefert beim Aufruf einen geordneten Auszug einer Binärdatei in hexadezimaler Notation. Der Aufruf lautet vollständig:

```
DUMP <Dateiname>
```

ASM.COM — ASM ist der Name eines Assemblers. Falls Sie mit ihm arbeiten wollen, benötigen Sie detaillierte Informationen, die die Grenzen dieses Buchs sprengen würden.

STAT.COM — STAT dient zur Verwaltung des Inhaltsverzeichnisses sowie der Peripherie. Nähere Informationen finden Sie im Anhang.

FILECOPY.COM — FILECOPY ermöglicht die Übertragung einzelner Dateien von einer Diskette auf eine andere, wenn Sie nur ein Laufwerk angeschlossen haben. Die vollständige Angabe lautet:

```
FILECOPY <Dateiname>
```

Für den Kopiervorgang ist ein kontrollierter Diskettenwechsel erforderlich.

DISCCOPY.COM — Das Dienstprogramm DISCCOPY erlaubt das Anfertigen einer Sicherungskopie mit nur einem Laufwerk. Der erforderliche mehrfache Disketten-

wechsel wird durch Kontrollmeldungen unterstützt. Nicht formatierte Zieldisketten werden vor Beginn des Kopiervorgangs mit den notwendigen Kenninformationen versehen.

COPYDISC.COM — COPYDISC fertigt eine Sicherungskopie mit zwei Laufwerken an. Sonst: siehe DISCCOPY.

DISCCHK.COM — Das Dienstprogramm DISCCHK überprüft zwei Disketten auf Datengleichheit mit einem Laufwerk.

CHKDISC.COM — Das Dienstprogramm CHKDISC überprüft zwei Disketten auf Datengleichheit mit zwei Laufwerken.

CLOAD.COM — CLOAD überträgt eine Datei von einer Kassette auf eine Diskette. Die allgemeine Form des Programmaufrufs lautet:

CLOAD <Dateiname auf Kassette>,<Dateiname>

Der Kassettendateiname muß in Anführungsstriche gesetzt werden. Wird der Dateiname für die Diskette weggelassen, nimmt das System den Namen des geladenen Programms. Wie üblich werden die Kontrollmeldungen beim Laden unterdrückt, wenn als erstes Zeichen des Dateinamens ein Rufzeichen vereinbart wird.

CSAVE.COM — Wie CSAVE, nur in umgekehrter Richtung.

FORMAT.COM — Dienstprogramm zur Formatierung von Disketten im Laufwerk A. Folgende Vereinbarungen sind möglich: FORMAT D legt eine Datendiskette für AMSDOS an. FORMAT I legt eine Diskette mit IBM-kompatiblem Datenformat an. FORMAT legt eine startfähige CP/M-Diskette an. FORMAT V erzeugt eine nicht startfähige CP/M-Diskette (CP/M-Datendiskette). Die Systemspuren enthalten keine Information. Die unter dem Standard CP/M 2.2 mögliche Angabe eines Laufwerks für den Formatiervorgang ist bei der Version für den Schneider CPC nicht erlaubt. Es wird immer nur mit dem aktiven Laufwerk formatiert.

SETUP.COM — Dieses Dienstprogramm dient zur Konfiguration der Systemparameter wie auch der Peripherie. Es enthält

eine Fülle von Möglichkeiten, die Tastatur oder die Schnittstellen für konkrete Anwendungen zu definieren.

Nach dem Aufruf des Dienstprogrammes meldet es sich mit der Ausgabe:

```
SETUP V2.0

**Initial command buffer empty

Is this correct (Y/N):_
```

Erläuterung: Es ist möglich eine CP/M-Diskette als Autostartdiskette zu erklären. Das unmittelbar nach dem CP/M-Start aufzurufende Programm kann in den Kommandopuffer geschrieben werden. Soll beispielsweise automatisch das Programm SETUP nach dem Systemstart aufgerufen werden, so ist die oben gezeigte Meldung durch Betätigen der Taste N zu verneinen. Das SETUP-Programm fordert dann den neuen Inhalt des Kommandopuffers an.

Beispiel:

```
Enter new initial command buffer:_SETUP.COM
```

Nach der Betätigung gibt das System die Standard-Startmeldung:

```
Sign-on string:

^Ö§ww^Öa§§^ÜwwCP/M2.2+Amstrad Consumer
Electronics plc^J^M
```

aus, die Sie nach Wunsch ändern können. Auch für den Drucker läßt sich Ähnliches vereinbaren, wenn Sie nach der Ausgabe der Meldung

```
** Printer power-up string empty

Is this correct (Y/N)
```

die Frage verneinen und eine eigene Meldung eingeben.

Wie unter Kontrolle des BASIC-Interpreters auch, kann mit dem SETUP-Programm die Tastenbelegung geändert werden. Sie gelangen in den entsprechenden Änderungsmodus, wenn Sie nach der Ausgabe

No keybord translations set

Is this correct (Y/N):_

die Frage mit N beantworten. Das System gibt dann die möglichen Optionen aus:

Enter required command from:–

A – Add key translation (key number, normal shift, control)
B – Delete key translation (key number)
C – Clear all translation
F – Finish translation

Key number ist der Tastenpositionscode. Unter *normal, shift* und *control* sind dann die gewünschten ASCII-Codes zu vereinbaren, wie Sie dies vom KEY-Befehl unter BASIC kennen.

Ähnliches gilt für die Definition von Funktionstasten, die als *keybord expansions set* bezeichnet werden.

Auch hier wird der Tastenpositionscode und der zugeordnete Stringausdruck vereinbart wie beim KEY DEF-Befehl in BASIC.

Die Standardvereinbarungen für die Ein-/Ausgabegeräte können ebenfalls vom Anwender geändert werden. Sie sind im SETUP-Programm als *10 byte settings* bezeichnet. Standardmäßig gilt:

CON: entspricht CRT: (Tastatur und Bildschirm)
RDR: entspricht TTY:
PUN: entspricht TTY:
LST: entspricht LPT: (Centronics-Printer)

Weitere SETUP-Optionen betreffen die Massenspeicher sowie die Datenprotokolle der Ein-/Ausgabebausteine. Sie sollten ohne Zuhilfenahme von speziellen Informationen keine Änderungen dieser Parameter vornehmen.

Falls Sie die von Ihnen vorgenommenen Änderungen dauerhaft auf ihrer Diskette sichern wollen, müssen Sie die Frage

```
Do you want to update your system disc (Y/N):_
```

mit Y beantworten. Andernfalls bleiben die Vereinbarungen nur bis zum erneuten Booten des CP/M-Systems erhalten.

Eine eingehende Erläuterung der zuvor aufgeführten Programme mit all ihren Optionen muß spezieller Literatur vorbehalten bleiben. Eine gute, kommentierte Zusammenstellung, die auch für den Schneider CPC in weiten Teilen gültig ist, finden Sie in dem CP/M-Handbuch von Rodnay Zaks. Dort ist die hier interessierende Version unter der Bezeichnung CP/M-80 aufgeführt. In den Anhängen dieses Buches finden Sie darüber hinaus für Ihre ersten Schritte unter CP/M eine kommentierte Zusammenstellung der Kommandos und Optionen für die Programme ED.COM, DDT.COM und STAT.COM und eine Liste der CTRL-Kommandos.

# Kapitel 9
# Datenkommunikation

## Einleitung

Sowohl der CPC 464 wie auch der CPC 664 ist von Hause aus nur mit einer parallelen Schnittstelle zum Anschluß externer Geräte wie Drucker oder Plotter ausgerüstet. Die für eine Datenfernübertragung (DFÜ) über Modems oder Akustikkoppler benötigte serielle Schnittstelle nach dem RS-232-Standard muß daher gesondert beschafft werden. Auf dem Markt sind verschiedene Versionen verfügbar, die zum Teil sogar mit der für eine Datenkommunikation erforderlichen Software und einem Akustikkoppler geliefert werden. Dieses Kapitel informiert Sie über die wichtigsten Grundlagen der Datenkommunikation und informiert Sie über sogenannte elektronische Briefkästen, die als Mailboxen bezeichnet werden.

## Grundlagen der Datenkommunikation

Alle in einem Computer abgespeicherten Daten liegen in paralleler Form als Binärmuster vor. Diese können in Form von Gleichspannungswerten angegeben werden (beispielsweise logisch Eins gleich 5 Volt, logisch Null gleich 0 Volt). Die Übertragung derartiger Daten von einem Ort A zu einem Ort B über eine handelsübliche Telefonleitung setzt voraus, daß die parallel vorliegenden Daten in eine serielle Folge einzelner Bits zerlegt und die Gleichspannungs- in Wechselspannungswerte umgesetzt werden. Telefonleitungen können nämlich nur Wechselspannungen im Bereich zwischen 300 Hz und 3500 Hz übertragen. Die Nullen und Einsen werden aus diesem Grunde in diskrete Töne unterschiedlicher Frequenz umgesetzt, die empfängerseitig wieder in entsprechende Gleichspannungsmuster zurückverwandelt werden.

Diese Aufgabe übernehmen spezielle elektronische Schaltungen, die je nach Arbeitsprinzip hierzulande als Modems oder Akustikkoppler bezeichnet werden. Diese sprachliche Unterscheidung ist eigentlich nicht ganz korrekt, da ein Akustikkoppler ebenfalls ein Modem ist. Er besitzt

nur zusätzlich eine Einrichtung, mit deren Hilfe die Einspeisung der Signale akustisch üer den Telefonhörer ermöglicht wird. Unter einem Modem versteht die Post (sie und nur sie ist für die Genehmigung und den Anschluß zuständig) dagegen eine Übertragungseinheit, die elektrisch mit dem Telefonnetz verbunden ist. Auf dem Markt werden Universalmodems angeboten, die den Datenverkehr mit Baudraten zwischen 75 und 19200 Baud und einer Vielfalt unterschiedlicher Übertragungsstandards ermöglichen. Sie dürfen ein derartiges Modem zur Zeit nur innerhalb hausinterner Anlagen einsetzen. Natürlich bietet auch die Post Modems an. Sie dürfen diese allerdings nicht selbst anschließen. Außerdem müssen die mit einem Modem verbundenen Geräte ebenfalls eine FTZ-Zulassung der Post besitzen.

Bei einem Akustikkoppler sieht das alles weniger schwierig aus. Er enthält ein Mikrophon zum Empfang und einen kleinen Lautsprecher zum Senden der Töne. Diese beiden Wandler werden über Gummimanschetten mit der Sprech- bzw. Hörkapsel eines Telefons akustisch gekoppelt. Auf dem Markt wird eine Vielzahl von Akustikkopplern unterschiedlicher Preisklassen (ab etwa DM 150) angeboten, die zum großen Teil eine Zulassungsnummer der Post besitzen. Da keine direkte elektrische Verbindung zum Postnetz erforderlich ist, können Sie die erforderliche Installation selbst vornehmen.

### Übertragungsfrequenzen und -geschwindigkeiten

Die Auswahl der für die Übertagung verwendeten Frequenzen ist einheitlich geregelt. Die folgende kleine Tabelle zeigt die in Europa sowie in Übersee üblichen Frequenzvereinbarungen für den Sende- und den Empfangsbetrieb.

| Modemtyp | Übertragungs-rate in Baud | Sende-frequenzen | | Empfangs-frequenzen | |
|---|---|---|---|---|---|
| | | 0 | 1 | 0 | 1 |
| CCITT V.21 Orig. | 300 | 1180 | 980 | 1850 | 1650 |
| CCITT V.21 Answ. | 300 | 1850 | 1650 | 1180 | 980 |
| Bell 103 Orig. | 300 | 1070 | 1270 | 2025 | 2225 |
| Bell 103 Answ. | 300 | 2025 | 2225 | 1070 | 1270 |

*Abb. 9.1: Übertragungsraten und Sende-/Empfangsfrequenzen nach der CCITT-Empfehlung V.21 bzw. Bell 103 (USA)*

Das Zeitraster, mit dem die einzelnen Bits über die Telefonleitung geschickt werden, hängt von der Übertragungsgeschwindigkeit ab. Diese Geschwindigkeit wird in Bit pro Sekunde angegeben. Hierfür ist auch die Maßeinheit Baud gebräuchlich. 300 Bit pro Sekunde entspricht also einer Übertragungsgeschwindigkeit von 300 Baud. Die üblichen Übertragungsgeschwindigkeiten liegen in der Regel zwischen 75 Bit pro Sekunde und 1200 Bit pro Sekunde. Der weitaus am häufigsten benutzte Wert beträgt 300 Baud. Auch die nachfolgend noch erwähnten Mailboxen arbeiten alle mit dieser Geschwindigkeit.

Das Wort Kommunikation bedeutet so viel wie Wechselgespräch. Damit so ein „Gespräch" zwischen Ihrem Computer und einer Mailbox oder einer Datenbank ordnungsgemäß ablaufen kann, müssen Datenkommu-

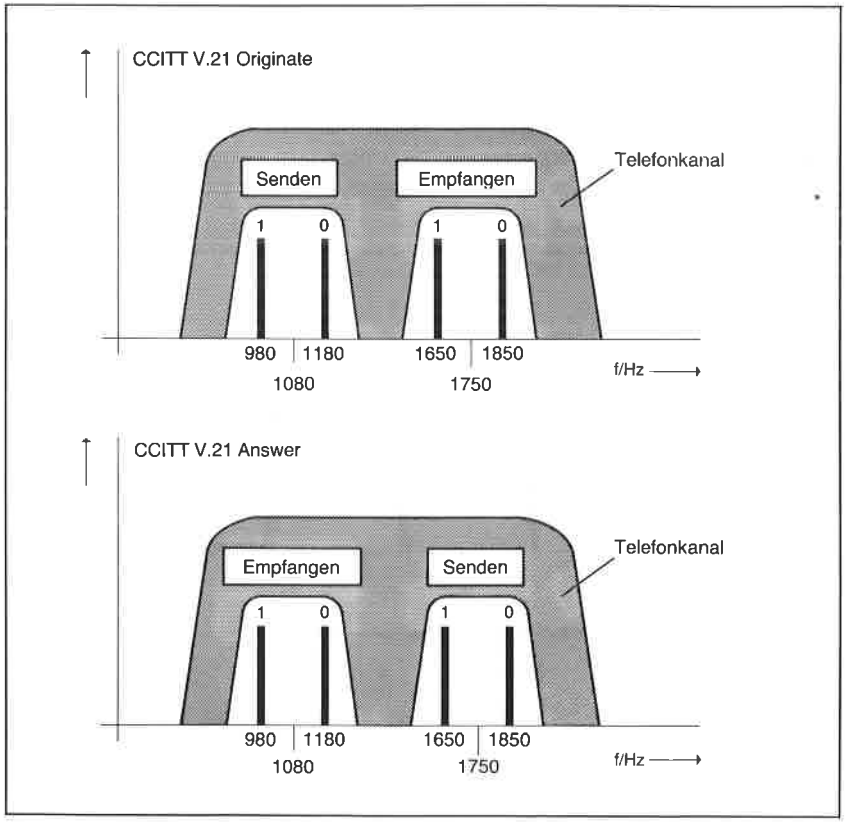



*Abb. 9.2: Übertragungsschema für den Originate- (oben) und Answermodus (unten)*

nikationsgeräte senden wie auch empfangen können. Damit es dabei nicht zu Konflikten kommt, gibt es ein standardisiertes Frequenzschema. Abb. 9.2 erläutert dies anhand einer Skizze. Im oberen Teil sind die Sende- und Empfangsfrequenzen für den sogenannten Originate-Modus dargestellt. Modems oder Akustikkoppler, die in dieser Betriebsart arbeiten, senden logische Nullen mit einer Frequenz von 1180 Hz und logische Einsen mit einer Frequenz von 1180 Hz und logische Einsen mit einer Frequenz von 980 Hz. Die Frequenzen 1650 und 1850 dagegen werden beim Empfang von Daten als Null und Eins interpretiert.

Der Kommunikationspartner am anderen Ende der Übertragungsstrecke muß natürlich genau umgekehrt verfahren, sonst kommt es zu keinem Nachrichtenaustausch. Er arbeitet im sogenannten Answermodus. Auch das von der Bundespost favorisierte Bildschirmtextsystem (Btx) macht von dieser Art der Übertragung Gebrauch. Die vom Postrechner an den Endbenutzer übertragenen Daten werden allerdings mit 1200 Baud, die Antworten des Endbenutzers nur mit 75 Baud übertragen. Bei einem Datenverkehr über den „großen Teich" müssen Sie ein Modem oder einen Koppler verwenden, der die dort üblichen Frequenzen auswerten kann (beispielsweise den in Abb. 9.1 angegebenen Bell-Standard).

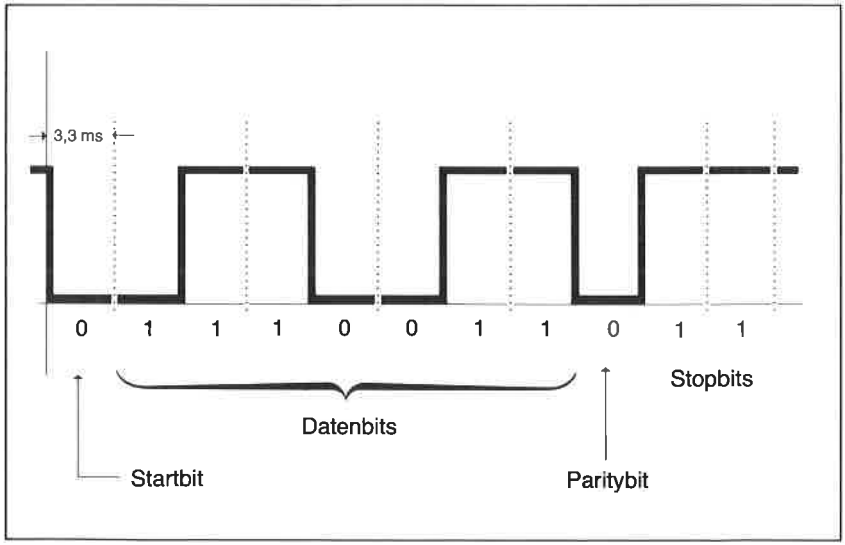



*Abb. 9.3: Serielle Übertragung eines binären Datenwortes mit 7 bit Wortlänge, einem Paritybit, einem Start- und zwei Stopbits*

Im einfachsten Fall ist für die Übertragung nur eine Verbindung mit wenigen Leitungen notwendig, über die neben den Nutzdaten selbst nur noch wenige Zusatzinformationen zur Synchronisation des Datenverkehrs übertragen werden. Jede Übertragung beginnt üblicherweise mit einem sogenannten Startbit, das einer logischen Null entspricht (Abb. 9.3).

Es folgen anschließend die Datenbits, eventuell ein Prüfbit zur Fehlerüberprüfung sowie ein oder zwei Stopbits, die einer logischen Eins entsprechen. Im einfachsten Fall benötigt das Datenwort 7 Bits für die Übertragung eines ASCII-Zeichens. Ohne zusätzliches Prüfbit, das auch als Paritybit bezeichnet wird, sind daher bei einem Start- und zwei Stopbits insgesamt 10 serielle Bits für die Übertragung eines Codezeichens erforderlich. Bei einer Datenübertragungsgeschwindigkeit von 300 Baud können pro Sekunde 300/10 = 30 Zeichen übertragen werden. Eine eng beschriebene DIN A4-Seite mit 2000 Zeichen benötigt somit für die Übertragung 67 Sekunden. Sie sollten an diesen Umstand einmal denken, wenn Sie mit Ihrem Schneider CPC eine Datenverbindung über größere Entfernungen herstellen. Von dem als Prüfbit dienenden Paritybit wird nur in Sonderfällen Gebrauch gemacht, weil die Erzeugung beim Sendevorgang und die Auswertung beim Empfang für einfache Datenkommunikationsumgebungen verhältnismäßig aufwendig ist.

Bei einer Übertragungsgeschwindigkeit von 300 Baud folgen die einzelnen Bits im Zeitraster von 3,3 ms (Millisekunden) aufeinander, wie dies in der Abb. 9.3 angedeutet ist.

### Die RS-232-Schnittstelle

Die für die Übertragung eingesetzte RS-232-Schnittstelle (serielle Schnittstelle) enthält als wichtigste Bauelemente einen sogenannten Parallel-Serien-Wandler und einen Baudratengenerator. Sind den logischen Werten Eins und Null die Spannungswerte +12 Volt und −12 Volt zugeordnet, spricht man auch von einer V24-Schnittstelle. Sie erzeugt eine Vielzahl von Nutz- und Steursignalen. Die wichtigsten sind in der folgenden Übersicht in Abb. 9.4 zusammengefaßt. Computer wie auch Modems bzw. Akustikkoppler stellen die Signale an speziellen Steckern bzw. Buchsen zur Verfügung, die als DB-25-Anschlüsse bezeichnet werden, weil sie maximal 25 Leitungen nach außen führen.

Die Datenflußrichtung ist für die beiden unterschiedlichen Gerätetypen jeweils angegeben. DTE ist dabei eine Abkürzung für Data Terminal Equipment, DCE dagegen für Data Communication Equipment.

| DB-25-Pin | Datenrichtung DCE | Datenrichtung DTE | Signal | Bedeutung | Erläuterung |
|---|---|---|---|---|---|
| 1 | • | • | GND | Protective Ground | Schutzerde |
| 2 | <– | –> | TxD | Transmitted Data | Sendedaten |
| 3 | –> | <– | RxD | Receidved Data | Empfangsdaten |
| 4 | <– | –> | RTS | Request To Send | Sendeteil einschalten |
| 5 | –> | <– | CTS | Clear To Send | Sendebereitschaft |
| 6 | –> | <– | DSR | Data Set Ready | Betriebsbereitschaft |
| 7 | • | • | GND | Signal GrouND | Signalmasse |
| 8 | –> | <– | DCD | Data Carrier Detector | Empfangssignalpegel |
| 20 | <– | –> | DTR | Data terminal ready | Terminal betriebsber. |

*Abb. 9.4: Anschlußbelegung der für den Modembetrieb wichtigen DB-25-Pins einer seriellen Schnittstelle nach dem RS-232-Standard*

Für eine einfache Verbindung werden nicht alle der in Abb. 9.4 genannten Verbindungen benötigt. Es reichen in den weitaus meisten Fällen die Signale TxD (Transmit Data = Daten senden) an Pin 2, RxD (Receive Data = Daten empfangen) an Pin 3 sowie die Signalmasse an Pin 7 (Abb. 9.5). Eine eventuell vorhandene Leitungsabschirmung wird mit Pin 1 verbunden.

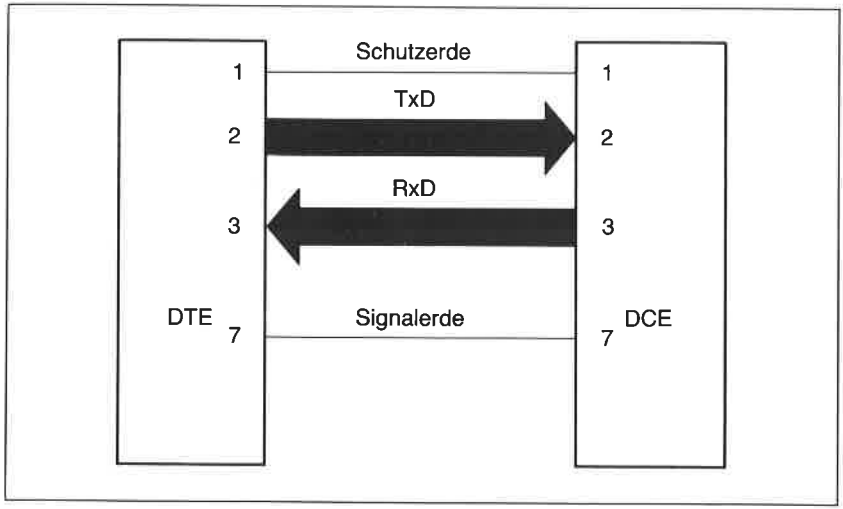



*Abb. 9.5: Die Signal- und Masseverbindungen zwischen einem DTE- und einem DCE-Gerät*

Bitte achten Sie darauf, daß die Bezeichnungen TxD und RxD sowohl bei DTE- als auch für DCE-Geräte an den Pins 2 und 3 gleich sind, obgleich die damit verbundenen Signalflußrichtungen nur für DTE-Geräte logisch stimmen. TxD ist also an einem DTE-Gerät ein Ausgangs- und RxD ein Eingangssignal. Beim DCE-Gerät ist es genau umgekehrt. Sie dürfen daher die entsprechenden Pins der Anschlußstecker oder -fahnen nur dann einfach miteinander verbinden, wenn Geräte verschiedenen Typs miteinander kommunizieren. Wenn zwei DTE-Geräte oder zwei DCE-Geräte, also Geräte des gleichen Typs, miteinander kommunizieren sollen, sind an einer Seite die Anschlußleitungen 2 und 3 zu vertauschen. Im Handel gibt es fertig konfektionierte Zwischenstücke, die diese Vertauschung vornehmen. Sie werden Nullmodems genannt.

Unglücklicherweise sagen nicht alle Hersteller von Computersystemen (und leider auch nicht die Hersteller von RS-232-Schnittstellen für den CPC), ob sie nach dem DTE- oder DCE-Standard geschaltet sind. Ein Akustikkoppler oder ein Modem arbeitet immer als DCE-Gerät. Bei ihnen ist somit der mit TxD bezeichnete Pin 2 ein Eingang und der mit TxD bezeichnete Pin ein Ausgang. Lassen Sie sich im Zweifelsfall von dem Lieferanten Ihres RS-232-Interfaces sagen, in welchem Modus er arbeitet.

### Kommunikationsprogramme

Zur Abwicklung einer Datenkommunikation zwischen Ihrem Computer und einer Mailbox, einer Datenbank oder einem Computer eines Bekannten genügt die RS-232-Schnittstelle allein noch nicht ganz. Sie benötigen zusätzlich noch ein Programm, das es Ihnen erlaubt, die Schnittstelle entsprechend dem geforderten Datenprotokoll oder der Übertragungsgeschwindigkeit zu programmieren und anschließend den Datenverkehr abzuwickeln. Besonders komfortabel und nützlich sind dabei jene Softwarepakete, die den gesamten Text einer derartigen Datenverbindung kontinuierlich auf einem externen Datenträger sichern. Dies ist besonders zu Beginn einer Reise in die Welt der Mailboxen recht nützlich, da Telefonverbindungen über große Entfernungen sehr teuer werden können.

Jede angewählte Datenbank oder Mailbox hält sofort nach der Grußmeldung per Menü die wichtigsten Informationen zur Benutzung abrufbar bereit. Sie können diese Information zunächst dauerhaft speichern, die Verbindung nach deren Empfang abbrechen und sich in Ruhe „auf dem Trockenen" informieren und mit den Eigenheiten Ihres Kommunikationspartners vertraut machen. Die auf dem deutschen Markt verfügba-

ren RS-232-Module für den Schneider CPC werden gegen Aufpreis mit mehr oder weniger komfortablen Kommunikationsprogrammen geliefert.

### Mailboxen

Die Frage, was ein Heimcomputeranwender mit einem Akustikkoppler sowie der zugehörigen Treibersoftware alles anfangen kann, hätte sicherlich hierzulande noch vor nur etwas mehr als einem Jahr die weitaus größte Zahl von Computerfreunden etwas in Verlegenheit gebracht. Heute allerdings weiß fast jeder, daß er mit den genannten Hilfsmitteln mindestens Kontakt zu den vielen elektronischen Briefkästen (Mailboxen) aufnehmen kann, die von privaten wie auch kommerziellen Betreibern zur (oftmals kostenlosen) Nutzung zur Verfügung gestellt werden. Sie müssen nur die Telefonnummer des Teilnehmers kennen. Diese finden Sie entweder in Mikrocomputerzeitschriften, speziellen Büchern zu diesem Thema oder aber auch in den Briefkästen selbst, wie die Abb. 9.6 zeigt, die über eine von der Firma SATURN in Köln eingerichtete und unterhaltene Mailbox abgerufen (und um einige Informationen ergänzt) wurde. Die darin aufgeführten Telefonnummern werden an dieser Stelle ohne Gewähr für deren Richtigkeit wiedergegeben.

<*> MAILBOXEN <*>

| | |
|---|---|
| W.M.S (20–06) | 0202-448204 |
| TOELLETURM (18–19) | 0211-556136 |
| EDV | 0211-328249 |
| SOFTWARE EXPRESS | 0211-414579 |
| EPSON | 0211-593453 |
| COMPUTER CENTER | 02202-50033 |
| SATURN | 0221-1616284 |
| WDR | 0221-371076 |
| COMMODORE MAILBOX | 069-6638191 |
| FRANZIS (TEDAS) | 089-598423 |
| | 089-596422 |

*Abb. 9.6: Auswahl von Mailboxen*

Wenn Sie beispielsweise die in der Übersicht angegebene Nummer des WDR Computerclubs anwählen und der Anschluß gerade einmal frei ist, meldet sich die Mailbox mit dem nachfolgend gezeigten Anfangsdialog (Stand 16. 5. 1985):

```
Der W D R - COMPUTERCLUB
begruesst Sie mit einem
SIEMENS 9753 Transdata-Computer / 1
>
*****************************************************
*  W D R - Computerclub Kommunikationssystem        *
*  KOMCOM - 1  Tel.-Nr.: 02 21 - 37 10 76           *
*  Version 1.0 von Guenter Eisbach (C) 1984         *
*  Timeout: 30 Sekunden                             *
*---------------------------------------------------*
*  Bedienungsanleitung:                             *
*  - RETURN -  schliesst jede Eingabe ab            *
*  - Ctrl/S -  haelt die Ausgabe an                 *
*  - Ctrl/Q -  setzt die Ausgabe fort               *
*  - Ctrl/X -  bricht die Ausgabe ab                *
*  - Ctrl/C -  fuehrt zum naechsten Eintrag         *
*  In der Auswahl bewirkt die Eingabe von:          *
*   ' ? '  die erneute Ausgabe des Menus            *
*   ' '  die Ausgabe des vorhergehenden             *
*        Menus                                       *
*   ' . '  die Ausgabe des Hauptmenus               *
*  ' ENDE ' die Ausloesung der Verbindung           *
*****************************************************

 Sie sind der  41832. Anrufer !

 ==>> Aktualisiert am: 15.05.1985

  *** H A U P T M E N U ***

 1  Informationen
 2  Programm - Boerse
 3  Briefkasten
 4  Computerclub

 ( 1281) Auswahl:
```

Eingabe der Zahl 4 an dieser Stelle führt dann zu folgender Bildschirmausgabe:

```
   Willkommen beim

*** COMPUTERCLUB ***

1  Club - Informationen
2  Club - Briefkasten
3  Basicode - Zeitung
4  BASICODE II - Informationen
5  BASICODE II Progr. DOWNLOAD
6  BASICODE II Progr.  UPLOAD
7  Begleitmaterial zu den SFB-CC
   Einfuehrungssendungen
```

Das übliche Übertragungsprotokoll der Mailboxen lautet:

Übertragungsgeschwindigkeit: 300 Baud
1 Startbit, 7 Datenbits, 2 Stopbits, kein Paritybit

Mit diesen Angaben und einem der verfügbaren RS-232-Bausteine sollten Sie in der Lage sein, Ihre ersten Schritte in die Welt der Datenkommunikation zu unternehmen.

Ein Hinweis noch zum Schluß: Manche der Mailboxen bzw. fast alle Datenbanken gewähren den Zugang nur für Mitglieder, die sich durch ein Schlüsselwort (Password) identifizieren müssen. Versuchen Sie nicht, das Codewort zu knacken. Sie belegen nur unnütz den Anschluß.

Anhang A

# Der Standard-ASCII-Zeichensatz und dessen Verschlüsselung

## ASCII-Symbole in Dezimal-, Oktal- und Hexadezimaldarstellung

| *Dez* | *Oktal* | *Hex* | *CHR* | *Dez* | *Oktal* | *Hex* | *CHR* | *Dez* | *Oktal* | *Hex* | *CHR* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 000 | 00 | NUL | 043 | 053 | 2B | + | 086 | 126 | 56 | V |
| 001 | 001 | 01 | SOH | 044 | 054 | 2C | , | 087 | 127 | 57 | W |
| 002 | 002 | 02 | STX | 045 | 055 | 2D | – | 088 | 130 | 58 | X |
| 003 | 003 | 03 | ETX | 046 | 056 | 2E | . | 089 | 131 | 59 | Y |
| 004 | 004 | 04 | EOT | 047 | 057 | 2F | / | 090 | 132 | 5A | Z |
| 005 | 005 | 05 | ENQ | 048 | 060 | 30 | 0 | 091 | 133 | 5B | [ |
| 006 | 006 | 06 | ACK | 049 | 061 | 31 | 1 | 092 | 134 | 5C | \ |
| 007 | 007 | 07 | BEL | 050 | 062 | 32 | 2 | 093 | 135 | 5D | ] |
| 008 | 010 | 08 | BS | 051 | 063 | 33 | 3 | 094 | 136 | 5E | ^ |
| 009 | 011 | 09 | HT | 052 | 064 | 34 | 4 | 095 | 137 | 5F | _ |
| 010 | 012 | 0A | LF | 053 | 065 | 35 | 5 | 096 | 140 | 60 | ' |
| 011 | 013 | 0B | VT | 054 | 066 | 36 | 6 | 097 | 141 | 61 | a |
| 012 | 014 | 0C | FF | 055 | 067 | 37 | 7 | 098 | 142 | 62 | b |
| 013 | 015 | 0D | CR | 056 | 070 | 38 | 8 | 099 | 143 | 63 | c |
| 014 | 016 | 0E | SO | 057 | 071 | 39 | 9 | 100 | 144 | 64 | d |
| 015 | 017 | 0F | SI | 058 | 072 | 3A | : | 101 | 145 | 65 | e |
| 016 | 020 | 10 | DLE | 059 | 073 | 3B | ; | 102 | 146 | 66 | f |
| 017 | 021 | 11 | DC1 | 060 | 074 | 3C | < | 103 | 147 | 67 | g |
| 018 | 022 | 12 | DC2 | 061 | 075 | 3D | = | 104 | 150 | 68 | h |
| 019 | 023 | 13 | DC3 | 062 | 076 | 3E | > | 105 | 151 | 69 | i |
| 020 | 024 | 14 | DC4 | 063 | 077 | 3F | ? | 106 | 152 | 6A | j |
| 021 | 025 | 15 | NAK | 064 | 100 | 40 | @ | 107 | 153 | 6B | k |
| 022 | 026 | 16 | SYN | 065 | 101 | 41 | A | 108 | 154 | 6C | l |
| 023 | 027 | 17 | ETB | 066 | 102 | 42 | B | 109 | 155 | 6D | m |
| 024 | 030 | 18 | CAN | 067 | 103 | 43 | C | 110 | 156 | 6E | n |
| 025 | 031 | 19 | EM | 068 | 104 | 44 | D | 111 | 157 | 6F | o |
| 026 | 032 | 1A | SUB | 069 | 105 | 45 | E | 112 | 160 | 70 | p |
| 027 | 033 | 1B | ESCAPE | 106 | 070 | 46 | F | 113 | 161 | 71 | q |
| 028 | 034 | 1C | FS | 071 | 107 | 47 | G | 114 | 162 | 72 | r |
| 029 | 035 | 1D | GS | 072 | 110 | 48 | H | 115 | 163 | 73 | s |
| 030 | 036 | 1E | RS | 073 | 111 | 49 | I | 116 | 164 | 74 | t |
| 031 | 037 | 1F | US | 074 | 112 | 4A | J | 117 | 165 | 75 | u |
| 032 | 040 | 20 | SPACE | 075 | 113 | 4B | K | 118 | 166 | 76 | v |
| 033 | 041 | 21 | ! | 076 | 114 | 4C | L | 119 | 167 | 77 | w |
| 034 | 042 | 22 | " | 077 | 115 | 4D | M | 120 | 170 | 78 | x |
| 035 | 043 | 23 | # | 078 | 116 | 4E | N | 121 | 171 | 79 | y |
| 036 | 044 | 24 | $ | 079 | 117 | 4F | O | 122 | 172 | 7A | z |
| 037 | 045 | 25 | % | 080 | 120 | 50 | P | 123 | 173 | 7B | { |
| 038 | 046 | 26 | & | 081 | 121 | 51 | Q | 124 | 174 | 7C | \| |
| 039 | 047 | 27 | ' | 082 | 122 | 52 | R | 125 | 175 | 7D | } |
| 040 | 050 | 28 | ( | 083 | 123 | 53 | S | 126 | 176 | 7E | ~ |
| 041 | 051 | 29 | ) | 084 | 124 | 54 | T | 127 | 177 | 7F | DEL |
| 042 | 052 | 2A | * | 085 | 125 | 55 | U | | | | |

*(Anmerkung: Der ASCII-Code verwendet nur 7 Bits eines Bytes. Das höchstwertige Bit (Bit 7) ist in dieser Tabelle auf Null gesetzt. Es kann in anderen Fällen auch den Wert 1 haben. Dann ist zum dezimalen Codewert 128, zum oktalen 200 und zum hexadezimalen 80 zu addieren.)*

Anhang B

# Der erweiterte Zeichensatz des Schneider CPC

| Dez | Hex | Chr | Dez | Hex | Chr |
|---|---|---|---|---|---|
| 0 | &00 | | 31 | &1F | |
| 1 | &01 | | 32 | &20 | |
| 2 | &02 | | 33 | &21 | ! |
| 3 | &03 | | 34 | &22 | " |
| 4 | &04 | | 35 | &23 | # |
| 5 | &05 | | 36 | &24 | $ |
| 6 | &06 | | 37 | &25 | % |
| 7 | &07 | | 38 | &26 | & |
| 8 | &08 | | 39 | &27 | ' |
| 9 | &09 | | 40 | &28 | ( |
| 10 | &0A | | 41 | &29 | ) |
| 11 | &0B | | 42 | &2A | * |
| 12 | &0C | | 43 | &2B | + |
| 13 | &0D | | 44 | &2C | , |
| 14 | &0E | | 45 | &2D | - |
| 15 | &0F | | 46 | &2E | . |
| 16 | &10 | | 47 | &2F | / |
| 17 | &11 | | 48 | &30 | 0 |
| 18 | &12 | | 49 | &31 | 1 |
| 19 | &13 | | 50 | &32 | 2 |
| 20 | &14 | | 51 | &33 | 3 |
| 21 | &15 | | 52 | &34 | 4 |
| 22 | &16 | | 53 | &35 | 5 |
| 23 | &17 | | 54 | &36 | 6 |
| 24 | &18 | | 55 | &37 | 7 |
| 25 | &19 | | 56 | &38 | 8 |
| 26 | &1A | | 57 | &39 | 9 |
| 27 | &1B | | 58 | &3A | : |
| 28 | &1C | | 59 | &3B | ; |
| 29 | &1D | | 60 | &3C | < |
| 30 | &1E | | 61 | &3D | = |

| Dez | Hex | Chr | Dez | Hex | Chr |
|---|---|---|---|---|---|
| 62 | &3E | > | 105 | &69 | i |
| 63 | &3F | ? | 106 | &6A | j |
| 64 | &40 | § | 107 | &6B | k |
| 65 | &41 | A | 108 | &6C | l |
| 66 | &42 | B | 109 | &6D | m |
| 67 | &43 | C | 110 | &6E | n |
| 68 | &44 | D | 111 | &6F | o |
| 69 | &45 | E | 112 | &70 | p |
| 70 | &46 | F | 113 | &71 | q |
| 71 | &47 | G | 114 | &72 | r |
| 72 | &48 | H | 115 | &73 | s |
| 73 | &49 | I | 116 | &74 | t |
| 74 | &4A | J | 117 | &75 | u |
| 75 | &4B | K | 118 | &76 | v |
| 76 | &4C | L | 119 | &77 | w |
| 77 | &4D | M | 120 | &78 | x |
| 78 | &4E | N | 121 | &79 | y |
| 79 | &4F | O | 122 | &7A | z |
| 80 | &50 | P | 123 | &7B | ä |
| 81 | &51 | Q | 124 | &7C | ö |
| 82 | &52 | R | 125 | &7D | ü |
| 83 | &53 | S | 126 | &7E | ß |
| 84 | &54 | T | 127 | &7F | ✚ |
| 85 | &55 | U | 128 | &80 | ^ |
| 86 | &56 | V | 129 | &81 | ´ |
| 87 | &57 | W | 130 | &82 | ¨ |
| 88 | &58 | X | 131 | &83 | £ |
| 89 | &59 | Y | 132 | &84 | © |
| 90 | &5A | Z | 133 | &85 | ¶ |
| 91 | &5B | Ä | 134 | &86 | $ |
| 92 | &5C | Ö | 135 | &87 | ' |
| 93 | &5D | Ü | 136 | &88 | ¼ |
| 94 | &5E | ^ | 137 | &89 | ½ |
| 95 | &5F | _ | 138 | &8A | ¾ |
| 96 | &60 | ` | 139 | &8B | ± |
| 97 | &61 | a | 140 | &8C | ÷ |
| 98 | &62 | b | 141 | &8D | ¬ |
| 99 | &63 | c | 142 | &8E | ¿ |
| 100 | &64 | d | 143 | &8F | ¡ |
| 101 | &65 | e | 144 | &90 | α |
| 102 | &66 | f | 145 | &91 | ß |
| 103 | &67 | g | 146 | &92 | Θ |
| 104 | &68 | h | 147 | &93 | δ |

| Dez | Hex | Chr | Dez | Hex | Chr |
|---|---|---|---|---|---|
| 148 | &94 | | 191 | &BF | |
| 149 | &95 | | 192 | &C0 | |
| 150 | &96 | | 193 | &C1 | |
| 151 | &97 | | 194 | &C2 | |
| 152 | &98 | | 195 | &C3 | |
| 153 | &99 | | 196 | &C4 | |
| 154 | &9A | | 197 | &C5 | |
| 155 | &9B | | 198 | &C6 | |
| 156 | &9C | | 199 | &C7 | |
| 157 | &9D | | 200 | &C8 | |
| 158 | &9E | | 201 | &C9 | |
| 159 | &9F | | 202 | &CA | |
| 160 | &A0 | | 203 | &CB | |
| 161 | &A1 | | 204 | &CC | |
| 162 | &A2 | | 205 | &CD | |
| 163 | &A3 | | 206 | &CE | |
| 164 | &A4 | | 207 | &CF | |
| 165 | &A5 | | 208 | &D0 | |
| 166 | &A6 | | 209 | &D1 | |
| 167 | &A7 | | 210 | &D2 | |
| 168 | &A8 | | 211 | &D3 | |
| 169 | &A9 | | 212 | &D4 | |
| 170 | &AA | | 213 | &D5 | |
| 171 | &AB | | 214 | &D6 | |
| 172 | &AC | | 215 | &D7 | |
| 173 | &AD | | 216 | &D8 | |
| 174 | &AE | | 217 | &D9 | |
| 175 | &AF | | 218 | &DA | |
| 176 | &B0 | | 219 | &DB | |
| 177 | &B1 | | 220 | &DC | |
| 178 | &B2 | | 221 | &DD | |
| 179 | &B3 | | 222 | &DE | |
| 180 | &B4 | | 223 | &DF | |
| 181 | &B5 | | 224 | &E0 | |
| 182 | &B6 | | 225 | &E1 | |
| 183 | &B7 | | 226 | &E2 | |
| 184 | &B8 | | 227 | &E3 | |
| 185 | &B9 | | 228 | &E4 | |
| 186 | &BA | | 229 | &E5 | |
| 187 | &BB | | 230 | &E6 | |
| 188 | &BC | | 231 | &E7 | |
| 189 | &BD | | 232 | &E8 | |
| 190 | &BE | | 233 | &E9 | |

| Dez | Hex | Chr | Dez | Hex | Chr |
|---|---|---|---|---|---|
| 234 | &EA | | 245 | &F5 | |
| 236 | &EC | | 246 | &F6 | |
| 237 | &ED | | 247 | &F7 | |
| 238 | &EE | | 248 | &F8 | |
| 239 | &EF | | 249 | &F9 | |
| 240 | &F0 | | 250 | &FA | |
| 241 | &F1 | | 251 | &FB | |
| 242 | &F2 | | 252 | &FC | |
| 243 | &F3 | | 253 | &FD | |
| 244 | &F4 | | 254 | &FE | |

Anhang C

# Tabelle zur Umrechnung von 8-bit-Dualzahlen in Hexadezimal- und Dezimalzahlen

| Dez | Hex | Binär | Dez | Hex | Binär |
|---|---|---|---|---|---|
| 0 | &00 | 00000000 | 27 | &1B | 00011011 |
| 1 | &01 | 00000001 | 28 | &1C | 00011100 |
| 2 | &02 | 00000010 | 29 | &1D | 00011101 |
| 3 | &03 | 00000011 | 30 | &1E | 00011110 |
| 4 | &04 | 00000100 | 31 | &1F | 00011111 |
| 5 | &05 | 00000101 | 32 | &20 | 00100000 |
| 6 | &06 | 00000110 | 33 | &21 | 00100001 |
| 7 | &07 | 00000111 | 34 | &22 | 00100010 |
| 8 | &08 | 00001000 | 35 | &23 | 00100011 |
| 9 | &09 | 00001001 | 36 | &24 | 00100100 |
| 10 | &0A | 00001010 | 37 | &25 | 00100101 |
| 11 | &0B | 00001011 | 38 | &26 | 00100110 |
| 12 | &0C | 00001100 | 39 | &27 | 00100111 |
| 13 | &0D | 00001101 | 40 | &28 | 00101000 |
| 14 | &0E | 00001110 | 41 | &29 | 00101001 |
| 15 | &0F | 00001111 | 42 | &2A | 00101010 |
| 16 | &10 | 00010000 | 43 | &2B | 00101011 |
| 17 | &11 | 00010001 | 44 | &2C | 00101100 |
| 18 | &12 | 00010010 | 45 | &2D | 00101101 |
| 19 | &13 | 00010011 | 46 | &2E | 00101110 |
| 20 | &14 | 00010100 | 47 | &2F | 00101111 |
| 21 | &15 | 00010101 | 48 | &30 | 00110000 |
| 22 | &16 | 00010110 | 49 | &31 | 00110001 |
| 23 | &17 | 00010111 | 50 | &32 | 00110010 |
| 24 | &18 | 00011000 | 51 | &33 | 00110011 |
| 25 | &19 | 00011001 | 52 | &34 | 00110100 |
| 26 | &1A | 00011010 | 53 | &35 | 00110101 |

| Dez | Hex | Binär | Dez | Hex | Binär |
|---|---|---|---|---|---|
| 54 | &36 | 00110110 | 97 | &61 | 01100001 |
| 55 | &37 | 00110111 | 98 | &62 | 01100010 |
| 56 | &38 | 00111000 | 99 | &63 | 01100011 |
| 57 | &39 | 00111001 | 100 | &64 | 01100100 |
| 58 | &3A | 00111010 | 101 | &65 | 01100101 |
| 59 | &3B | 00111011 | 102 | &66 | 01100110 |
| 60 | &3C | 00111100 | 103 | &67 | 01100111 |
| 61 | &3D | 00111101 | 104 | &68 | 01101000 |
| 62 | &3E | 00111110 | 105 | &69 | 01101001 |
| 63 | &3F | 00111111 | 106 | &6A | 01101010 |
| 64 | &40 | 01000000 | 107 | &6B | 01101011 |
| 65 | &41 | 01000001 | 108 | &6C | 01101100 |
| 66 | &42 | 01000010 | 109 | &6D | 01101101 |
| 67 | &43 | 01000011 | 110 | &6E | 01101110 |
| 68 | &44 | 01000100 | 111 | &6F | 01101111 |
| 69 | &45 | 01000101 | 112 | &70 | 01110000 |
| 70 | &46 | 01000110 | 113 | &71 | 01110001 |
| 71 | &47 | 01000111 | 114 | &72 | 01110010 |
| 72 | &48 | 01001000 | 115 | &73 | 01110011 |
| 73 | &49 | 01001001 | 116 | &74 | 01110100 |
| 74 | &4A | 01001010 | 117 | &75 | 01110101 |
| 75 | &4B | 01001011 | 118 | &76 | 01110110 |
| 76 | &4C | 01001100 | 119 | &77 | 01110111 |
| 77 | &4D | 01001101 | 120 | &78 | 01111000 |
| 78 | &4E | 01001110 | 121 | &79 | 01111001 |
| 79 | &4F | 01001111 | 122 | &7A | 01111010 |
| 80 | &50 | 01010000 | 123 | &7B | 01111011 |
| 81 | &51 | 01010001 | 124 | &7C | 01111100 |
| 82 | &52 | 01010010 | 125 | &7D | 01111101 |
| 83 | &53 | 01010011 | 126 | &7E | 01111110 |
| 84 | &54 | 01010100 | 127 | &7F | 01111111 |
| 85 | &55 | 01010101 | 128 | &80 | 10000000 |
| 86 | &56 | 01010110 | 129 | &81 | 10000001 |
| 87 | &57 | 01010111 | 130 | &82 | 10000010 |
| 88 | &58 | 01011000 | 131 | &83 | 10000011 |
| 89 | &59 | 01011001 | 132 | &84 | 10000100 |
| 90 | &5A | 01011010 | 133 | &85 | 10000101 |
| 91 | &5B | 01011011 | 134 | &86 | 10000110 |
| 92 | &5C | 01011100 | 135 | &87 | 10000111 |
| 93 | &5D | 01011101 | 136 | &88 | 10001000 |
| 94 | &5E | 01011110 | 137 | &89 | 10001001 |
| 95 | &5F | 01011111 | 138 | &8A | 10001010 |
| 96 | &60 | 01100000 | 139 | &8B | 10001011 |

| Dez | Hex | Binär | Dez | Hex | Binär |
|---|---|---|---|---|---|
| 140 | &8C | 10001100 | 183 | &B7 | 10110111 |
| 141 | &8D | 10001101 | 184 | &B8 | 10111000 |
| 142 | &8E | 10001110 | 185 | &B9 | 10111001 |
| 143 | &8F | 10001111 | 186 | &BA | 10111010 |
| 144 | &90 | 10010000 | 187 | &BB | 10111011 |
| 145 | &91 | 10010001 | 188 | &BC | 10111100 |
| 146 | &92 | 10010010 | 189 | &BD | 10111101 |
| 147 | &93 | 10010011 | 190 | &BE | 10111110 |
| 148 | &94 | 10010100 | 191 | &BF | 10111111 |
| 149 | &95 | 10010101 | 192 | &C0 | 11000000 |
| 150 | &96 | 10010110 | 193 | &C1 | 11000001 |
| 151 | &97 | 10010111 | 194 | &C2 | 11000010 |
| 152 | &98 | 10011000 | 195 | &C3 | 11000011 |
| 153 | &99 | 10011001 | 196 | &C4 | 11000100 |
| 154 | &9A | 10011010 | 197 | &C5 | 11000101 |
| 155 | &9B | 10011011 | 198 | &C6 | 11000110 |
| 156 | &9C | 10011100 | 199 | &C7 | 11000111 |
| 157 | &9D | 10011101 | 200 | &C8 | 11001000 |
| 158 | &9E | 10011110 | 201 | &C9 | 11001001 |
| 159 | &9F | 10011111 | 202 | &CA | 11001010 |
| 160 | &A0 | 10100000 | 203 | &CB | 11001011 |
| 161 | &A1 | 10100001 | 204 | &CC | 11001100 |
| 162 | &A2 | 10100010 | 205 | &CD | 11001101 |
| 163 | &A3 | 10100011 | 206 | &CE | 11001110 |
| 164 | &A4 | 10100100 | 207 | &CF | 11001111 |
| 165 | &A5 | 10100101 | 208 | &D0 | 11010000 |
| 166 | &A6 | 10100110 | 209 | &D1 | 11010001 |
| 167 | &A7 | 10100111 | 210 | &D2 | 11010010 |
| 168 | &A8 | 10101000 | 211 | &D3 | 11010011 |
| 169 | &A9 | 10101001 | 212 | &D4 | 11010100 |
| 170 | &AA | 10101010 | 213 | &D5 | 11010101 |
| 171 | &AB | 10101011 | 214 | &D6 | 11010110 |
| 172 | &AC | 10101100 | 215 | &D7 | 11010111 |
| 173 | &AD | 10101101 | 216 | &D8 | 11011000 |
| 174 | &AE | 10101110 | 217 | &D9 | 11011001 |
| 175 | &AF | 10101111 | 218 | &DA | 11011010 |
| 176 | &B0 | 10110000 | 219 | &DB | 11011011 |
| 177 | &B1 | 10110001 | 220 | &DC | 11011100 |
| 178 | &B2 | 10110010 | 221 | &DD | 11011101 |
| 179 | &B3 | 10110011 | 222 | &DE | 11011110 |
| 180 | &B4 | 10110100 | 223 | &DF | 11011111 |
| 181 | &B5 | 10110101 | 224 | &E0 | 11100000 |
| 182 | &B6 | 10110110 | 225 | &E1 | 11100001 |

| Dez | Hex | Binär | Dez | Hex | Binär |
|---|---|---|---|---|---|
| 226 | &E2 | 11100010 | 241 | &F1 | 11110001 |
| 227 | &E3 | 11100011 | 242 | &F2 | 11110010 |
| 228 | &E4 | 11100100 | 243 | &F3 | 11110011 |
| 229 | &E5 | 11100101 | 244 | &F4 | 11110100 |
| 230 | &E6 | 11100110 | 245 | &F5 | 11110101 |
| 231 | &E7 | 11100111 | 246 | &F6 | 11110110 |
| 232 | &E8 | 11101000 | 247 | &F7 | 11110111 |
| 233 | &E9 | 11101001 | 248 | &F8 | 11111000 |
| 234 | &EA | 11101010 | 249 | &F9 | 11111001 |
| 235 | &EB | 11101011 | 250 | &FA | 11111010 |
| 236 | &EC | 11101100 | 251 | &FB | 11111011 |
| 237 | &ED | 11101101 | 252 | &FC | 11111100 |
| 238 | &EE | 11101110 | 253 | &FD | 11111101 |
| 239 | &EF | 11101111 | 254 | &FE | 11111110 |
| 240 | &F0 | 11110000 | 255 | &FF | 11111111 |

# Anhang D
# Übersicht über die BASIC-Tokens

| BASIC-Befehl | BASIC-Token dez. | | BASIC-Token hex. | | |
|---|---|---|---|---|---|
| ABS | 255 | 0 | FF | 0 | |
| AFTER | 128 | | 80 | | |
| ASC | 255 | 1 | FF | 1 | |
| ATN | 255 | 2 | FF | 2 | |
| AUTO | 129 | | 81 | | |
| BIN$ | 255 | 113 | FF | 71 | |
| BORDER | 130 | | 82 | | |
| CALL | 131 | | 83 | | |
| CAT | 132 | | 84 | | |
| CHAIN | 133 | | 85 | | |
| CHR$ | 255 | 3 | FF | 3 | |
| CINT | 255 | 4 | FF | 4 | |
| CLEAR | 134 | | 86 | | |
| CLG | 135 | | 87 | | |
| CLOSEIN | 136 | | 88 | | |
| CLOSEOUT | 137 | | 89 | | |
| CLS | 138 | | 8A | | |
| CONT | 139 | | 8B | | |
| COPYCHR$ | 255 | 126 | FF | 7E | (nur CPC 664) |
| COS | 255 | 5 | FF | 5 | |
| CREAL | 255 | 6 | FF | 6 | |
| CURSOR | 225 | | E1 | | (nur CPC 664) |
| DATA | 140 | | 8C | | |
| DEC$ | 255 | 114 | FF | 72 | (auch CPC 464!) |
| DEF FN | 141 | | 8D | | |
| DEFINT | 142 | | 8E | | |
| DEFREAL | 143 | | 8F | | |
| DEFSTR | 144 | | 90 | | |

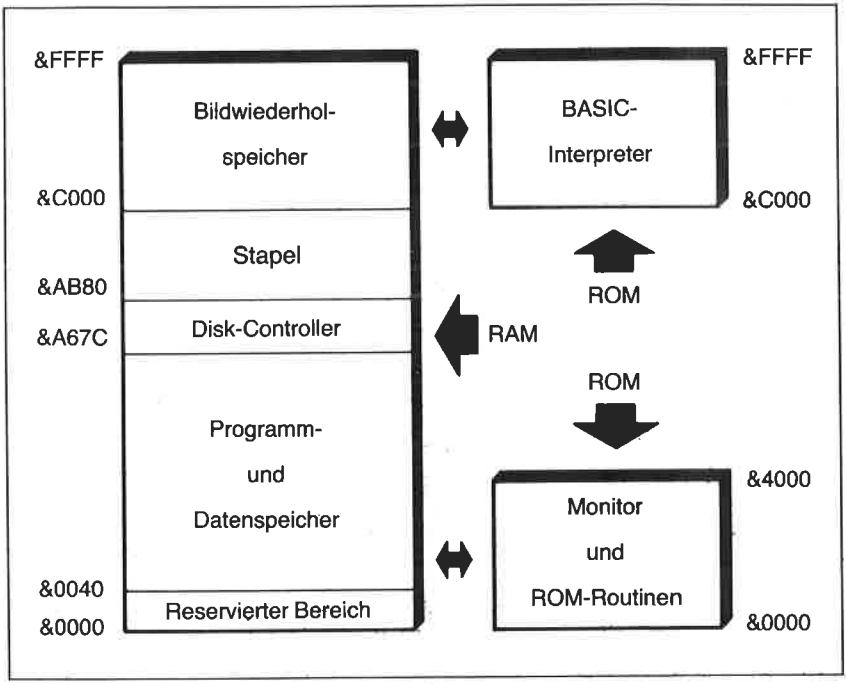
| BASIC-Befehl | BASIC-Token dez. | | BASIC-Token hex. | | |
|---|---|---|---|---|---|
| DEG | 145 | | 91 | | |
| DELETE | 146 | | 92 | | |
| DERR | 255 | 73 | FF | 49 | (auch CPC 464!) |
| DIM | 147 | | 93 | | |
| DRAW | 148 | | 94 | | |
| DRAWR | 149 | | 95 | | |
| EDIT | 150 | | 96 | | |
| EI | 220 | | DC | | |
| ELSE | 151 | | 97 | | |
| END | 152 | | 98 | | |
| ENT | 153 | | 99 | | |
| ENV | 154 | | 9A | | |
| EOF | 255 | 64 | FF | 40 | |
| ERASE | 155 | | 9B | | |
| ERL | 227 | | E3 | | |
| ERROR | 156 | | 9C | | |
| EVERY | 157 | | 9D | | |
| EXP | 255 | 7 | FF | 7 | |
| FILL | 221 | | DF | | (nur CPC 664) |
| FIX | 255 | 8 | FF | 8 | |
| FOR | 158 | | 9E | | |
| FRAME | 224 | | E0 | | (nur CPC 664) |
| FRE | 255 | 9 | FF | 9 | |
| GOSUB | 159 | | 9F | | |
| GOTO | 160 | | A0 | | |
| GRAPHICS.. | 222 | | DE | | (nur CPC 664) |
| HEX$ | 255 | 115 | FF | 73 | |
| HIMEM | 255 | 16 | FF | 42 | |
| IF | 161 | | A1 | | |
| INK | 162 | | A2 | | |
| INKEY | 255 | 10 | FF | 0A | |
| INKEY$ | 255 | 67 | FF | 43 | |
| INP | 255 | 11 | FF | 0B | |
| INPUT | 163 | | A3 | | |
| INSTR | 255 | 116 | FF | 74 | |
| INT | 255 | 12 | FF | 0C | |
| JOY | 255 | 13 | FF | 0D | |
| KEY | 164 | | A4 | | |

| BASIC-Befehl | BASIC-Token dez. | | BASIC-Token hex. | | |
|---|---|---|---|---|---|
| KEY DEF | 164 | 141 | A4 | 8D | |
| LEFT$ | 255 | 117 | FF | 75 | |
| LEN | 255 | 14 | FF | 0E | |
| LET | 165 | | A5 | | |
| LINE INPUT | 166 | 163 | A6 | A3 | |
| LIST | 167 | | A7 | | |
| LOAD | 168 | | A8 | | |
| LOCATE | 169 | | A9 | | |
| LOG | 255 | 15 | FF | 0F | |
| LOG10 | 255 | 16 | FF | 10 | |
| LOWER$ | 255 | 17 | FF | 11 | |
| MASK | 223 | | DF | | (nur CPC 664) |
| MAX | 255 | 118 | FF | 76 | |
| MEMORY | 170 | | AA | | |
| MERGE | 171 | | AB | | |
| MID$ | 172 | | AC | | |
| MIN | 255 | 119 | FF | 77 | |
| MOD | 251 | | FB | | (nur CPC 664) |
| MODE | 173 | | AD | | |
| MOVE | 174 | | AE | | |
| MOVER | 175 | | AF | | |
| NEXT | 176 | | B0 | | |
| NEW | 177 | | B1 | | |
| ON | 178 | | B2 | | |
| ON BREAK | 179 | | B3 | | |
| ON ERROR | 180 | | B4 | | |
| ON SQ | 181 | | B5 | | |
| OPENIN | 182 | | B6 | | |
| OPENOUT | 183 | | B7 | | |
| ORIGIN | 184 | | B8 | | |
| OUT | 185 | | B9 | | |
| PAPER | 186 | | BA | | |
| PEEK | 255 | 18 | FF | 12 | |
| PEN | 187 | | BB | | |
| PLOT | 188 | | BC | | |
| PLOTR | 189 | | BD | | |
| PI | 255 | 68 | FF | 44 | |
| POKE | 190 | | BE | | |

| BASIC-Befehl | BASIC-Token dez. | | BASIC-Token hex. | |
|---|---|---|---|---|
| POS | 255 | 120 | FF | 78 |
| PRINT | 191 | | BF | |
| ' | 192 | | C0 | |
| RAD | 193 | | C1 | |
| RANDOMIZE | 194 | | C2 | |
| READ | 195 | | C3 | |
| RELEASE | 196 | | C4 | |
| REM | 197 | | C5 | |
| REMAIN | 255 | 19 | FF | 13 |
| RENUM | 198 | | C6 | |
| RESTORE | 199 | | C7 | |
| RESUME | 200 | | C8 | |
| RETURN | 201 | | C9 | |
| RIGHT$ | 255 | 121 | FF | 79 |
| RND | 255 | 69 | FF | 45 |
| ROUND | 255 | 122 | FF | 7A |
| RUN | 202 | | CA | |
| SAVE | 203 | | CB | |
| SGN | 255 | 20 | FF | 14 |
| SIN | 255 | 21 | FF | 15 |
| SOUND | 204 | | CC | |
| SPACE$ | 255 | 22 | FF | 16 |
| SPEED INK | 205 | 162 | CD | A2 |
| SPEED KEY | 205 | 164 | CD | A4 |
| SPEED WRITE | 205 | 217 | CD | DA |
| SQ | 255 | 23 | FF | 17 |
| SQR | 255 | 24 | FF | 18 |
| STOP | 206 | | CE | |
| STR$ | 255 | 25 | FF | 19 |
| STRING$ | 255 | 123 | FF | 7B |
| SYMBOL | 207 | | CF | |
| SYMBOL AFT | 2071 | 28 | CF | 80 |
| TAG | 208 | | D0 | |
| TAGOFF | 209 | | D1 | |
| TAN | 255 | 26 | FF | 1A |
| TEST | 255 | 124 | FF | 7C |
| TESTR | 255 | 125 | FF | 7D |
| TIME | 255 | 70 | FF | 46 |

| BASIC-Befehl | BASIC-Token dez. | | BASIC-Token hex. | |
|---|---|---|---|---|
| TROFF | 210 | | D2 | |
| TRON | 211 | | D3 | |
| UNT | 255 | 27 | FF | 1B |
| UPPER$ | 255 | 28 | FF | 1C |
| VAL | 255 | 29 | FF | 1D |
| VPOS | 255 | 127 | F | 7F |
| WAIT | 212 | | D4 | |
| WEND | 213 | | D5 | |
| WHILE | 214 | | D6 | |
| WIDTH | 215 | | D7 | |
| WINDOW | 216 | | D8 | |
| WRITE | 217 | | D9 | |
| XPOS | 255 | 71 | FF | 47 |
| YPOS | 255 | 72 | FF | 48 |
| ZONE | 218 | | DA | |

# Anhang E
# Die Standard-Speicherorganisation des CPC 464 und des CPC 664




Hinweis: Beim Schneider CPC 464 ohne angeschlossenes Laufwerk beträgt die höchste nutzbare Speicheradresse &AB7F.

# Anhang F
# Adressen wichtiger BASIC-Vektoren

| BASIC-Vektoren | |
|---|---|
| Adresse | Erläuterung |
| &AC23 | Position des Druckkopfes |
| &AC24 | Druckbreite in Zeichen |
| &AE3F–&AE40 | Startadresse bei LOAD |
| &AE42 | Dateityp |
| &AE43–&AE44 | Dateilänge |
| &AE7B–&AE7C | HIMEM |
| &AE7D–&AE7E | Ende freies RAM |
| &AE7F–&AE80 | Start freies RAM |
| &AE81–&AE82 | Start BASIC-Programm |
| &AE83–&AE84 | Ende BASIC-Programm |
| &AE85–&AE86 | Start der Variablen |
| &AE87–&AE88 | Start Feldvariablen |
| &AE89–&AE8A | Ende Feldvariablen |
| &B08D–&B08E | Start Stringpool |
| &B08F–&B090 | Ende Stringpool |
| Hinweis: bei 16-bit-Zeigern: 1. Adresse enthält das Lowbyte, die 2. Adresse das Highbyte | |

# Anhang G
# Farbdefinitionen, Farbzuordnungen und Grauwerttabelle

| Farbnr. | Farbe | Farbnr. | Farbe |
|---|---|---|---|
| 0 | Schwarz | 16 | Rosa |
| 1 | Blau | 17 | Pastellmagenta |
| 2 | Hellblau | 18 | Hellgrün |
| 3 | Rot | 19 | Seegrün |
| 4 | Magenta | 20 | helles Blaugrün |
| 5 | Hellviolett | 21 | Limonengrün |
| 6 | Hellrot | 22 | Pastellgrün |
| 7 | Purpur | 23 | Pastellblaugrün |
| 8 | Helles Magenta | 24 | Hellgelb |
| 9 | Grün | 25 | Pastellgelb |
| 10 | Blaugrün | 26 | Leuchtendweiß |
| 11 | Himmelblau | 27 | Weiß (wie 13) |
| 12 | Gelb | 28 | Purpur (wie 7) |
| 13 | Weiß | 29 | Pastellgelb (wie 25) |
| 14 | Pastellblau | 30 | Blau (wie 1) |
| 15 | Orange | 31 | Seegrün (wie 19) |

*Die 32 (27) Farben des Schneider CPC*

| Farbnr. | MODE 0 | MODE 1 | MODE 2 | Farbnr. | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 8 | 10 | 1 | 1 |
| 1 | 24 | 24 | 24 | 9 | 12 | 24 | 24 |
| 2 | 20 | 20 | 1 | 10 | 14 | 20 | 1 |
| 3 | 6 | 6 | 24 | 11 | 16 | 6 | 24 |
| 4 | 26 | 1 | 1 | 12 | 18 | 1 | 1 |
| 5 | 0 | 24 | 24 | 13 | 22 | 24 | 24 |
| 6 | 2 | 20 | 1 | 14 | 1/24 | 20 | 1 |
| 7 | 8 | 6 | 24 | 15 | 16/11 | 6 | 24 |

*Farbzuordnungen für die Betriebsarten MODE 0, MODE 1 und MODE 2*

| Grauwert 1 | Grauwert 2 | Grauwert 3 | Grauwert 4 | Grauwert 5 | Grauwert 6 |
|---|---|---|---|---|---|
| (0) Schwarz | (1) Blau<br>(2) Hellblau | (3) Rot<br>(4) Magenta<br>(5) Hellviolett<br>(6) Hellrot<br>(7) Purpur<br>(8) helles Magenta | (9) Grün<br>(10) Blaugrün<br>(11) Himmelblau<br>(18) Hellgrün<br>(19) Seegrün<br>(20) helles Blaugrün | (12) Gelb<br>(15) Orange<br>(21) Limonengrün<br>(24) Hellgelb | (13) Weiß<br>(14) Pastellblau<br>(16) Rosa<br>(17) Pastellmagenta<br>(22) Pastellgrün<br>(23) Pastellblaugrün<br>(24) Pastellgelb<br>(26) Leuchtendweiß |

*Grauwerttabelle*

# Anhang H
# CP/M-Control-Codes

Im Zusammenhang mit direkten Eingaben von der Tastatur sind folgende Control-Codes von Interesse:

| | |
|---|---|
| CTRL-S | stoppt die Ausgabe auf dem Bildschirm. Fortsetzung durch Betätigen einer beliebigen anderen Taste. |
| CTRL-C | führt einen Warmstart des Systems durch und liest die Diskettenkennung ein. Während eines Programmlaufs eingegeben, wird ein Programmabbruch ausgelöst. |
| CTRL-P | Alle Ausgaben werden auf einen angeschlossenen Drucker ausgegeben. Rückschaltung erfolgt durch nochmaliges Betätigen von CTRL-P.<br><br>Beispiel:<br><br>CTRL-P<br>DIR<br>CTRL-P<br><br>gibt den Inhalt der Diskette im aktiven Laufwerk auf den Drucker aus. Anschließend wird wieder auf Bildschirmausgabe zurückgeschaltet. |
| CTRL-X | löscht eine gerade durchgeführte Tastatureingabe, sofern noch nicht die ENTER-Taste betätigt wurde. |

# Anhang I
# ED-Kommandos

Der Editor wird mit ED <Dateiname> aufgerufen. Der Dateiname muß angegeben werden, da andernfalls die Fehlermeldung <DISK OR DIRECTORY FULL> ausgegeben wird. Das Betriebssystem überprüft, ob der angegebene Dateiname neu ist oder einer bereits bestehenden Datei entspricht, und gibt eine entsprechende Meldung aus.

Nach dem Laden meldet sich der Editor mit einem Doppelpunkt, einem Leerzeichen und einem Stern: : *

Die Kommandos zur Texterstellung und Korrektur lauten:

| | |
|---|---|
| nA | Laden bzw. Anhängen von *n* Zeilen aus der im ED-Aufruf angegebenen Datei in den Speicher. Wird für n kein Wert angegeben, liest der Editor nur eine Zeile ein. Für *n* kann das Zeichen # angegeben werden. Dadurch werden 65535 Zeilen geladen. Das Kommando #A muß am Anfang vereinbart werden, wenn eine bereits bestehende Datei geöffnet und eingelesen werden soll. |
| (−)B | setzt den Textcursor an den Anfang (B) oder an das Ende (−B) des eingelesenen Textes. |
| (−)nC | setzt den Textcursor um *n* Zeichen vorwärts (nC) oder rückwärts (−nC). Ein ENTER wird als zwei Zeichen (Carriage Return und Line Feed) angesehen! Ist das Ende einer Textzeile erreicht, springt der Textcursor automatisch eine Zeile weiter. |

| | |
|---|---|
| E | beendet das Programm und sichert den Text in eine Datei mit dem alten Namen. Die zuvor eingelesene Datei wird in eine Sicherungsdatei umbenannt (Namenserweiterung .BAK). Das System kehrt in den CPM-Modus zurück. |
| nF <Zeichenkette> (ENTER/^Z) | sucht den hinter dem Textcursor stehenden Text auf das n-te Auftreten der angegebenen Zeichenkette im Speicher ab. Der Textcursor steht anschließend *hinter* dem gefundenen Textelement. Die Eingabe von ^Z (CTRL-Z) anstelle der ENTER-Taste erlaubt die Verkettung mehrerer Befehle. Sie werden einfach aneinandergehängt. |
| H | speichert den aktuellen Text im Speicher unter dem Namen der alten Datei ab und benennt letztere in eine Backup-Datei um. Das System verweilt im Editor. Die Datei muß gegebenenfalls jedoch neu geladen werden. |
| I | überführt den Editor in den Einfügemodus (Insertmodus). |
| I<Zeichenkette>^Z | fügt die angegebene Zeichenkette an der aktuellen Textcursorposition ein. Der Cursor steht anschließend hinter dem letzten Zeichen der Zeichenkette. |
| J<Zeichenkette1>^Z <Zeichenkette2>^Z <Zeichenkette3> (ENTER/^Z) | Die Zeichenkette1 wird aufgesucht, die Zeichenkette2 darangehängt, und alle Zeichen vom Ende der zweiten bis zum Anfang der Zeichenkette3 werden gelöscht. Der Cursor steht nach dieser Operation unmittelbar *vor* der Zeichenkette3. |
| (−)nK | löscht die *n* Zeichen nach oder vor dem Textcursor. Dieser steht anschließend direkt hinter dem letzten gelöschten Zeichen. |

| | |
|---|---|
| (−)n[L] | setzt den Textcursor *n* Zeilen vorwärts oder zurück. Ist n=0, dann wird der Cursor an den Anfang der aktuellen Zeile gesetzt. Das Zeichen L braucht nicht unbedingt angegeben zu werden. Beispiel: −23. |
| nM<Befehl> (ENTER/^Z) | führt das ED-Kommando <Befehl> n-mal aus. Ist n=0 oder 1, dann wird der Befehl so lange ausgeführt, bis das Ende des Textes erreicht ist. |
| nN<Zeichenkette> (ENTER/^Z) | sucht den im Speicher befindlichen Text der Datei auf das *n*-te Vorkommen der Zeichenkette ab. Wird diese im Speicher nicht gefunden, lädt ED so lange Zeilen aus der Hauptdatei nach, bis der Suchvorgang beendet ist. Der Cursor wird an das Ende des gefundenen Textes gesetzt. |
| O | löscht den im Speicher abgelegten Text und stellt die alte Datei wieder her. Nach dem Befehl ist das System noch im Editiermodus. |
| (−)nP | listet *n* Seiten (zu je 24 Textzeilen) vorwärts oder rückwärts. Ist n=0, werden von der aktuellen Cursorposition aus 23 Zeilen vorwärts ausgegeben. Nach der Operation steht der Textcursor am Anfang der angezeigten Seite. |
| Q | verläßt den Editor, ohne daß etwas verändert wird, und kehrt zu CP/M zurück. Eine vor dem Editieren existierende und aus dem Editor heraus angelegte Backup-Datei (.BAK) wird gelöscht. |

# Anhang J
# DDT-Kommandos

Programmaufruf:

DDT

bzw.

DDT <Dateiname.Dateityp>

Es spielt dabei keine Rolle, welchen Typs die Datei im aktuellen Fall ist. DDT arbeitet also auch mit BASIC-Dateien zusammen.

A <Startadresse> ruft den eingebauten Assembler auf. Beginnend mit der <Startadresse> können Maschinenbefehle in der Z80-Assemblersymbolik eingegeben werden. Rückkehr in die Kommandoebene durch Betätigen von ENTER.

D [<Startadresse>, [<Endadresse>]] gibt den Inhalt von 16*12 = 192 Bytes in hexadezimaler Form auf dem Bildschirm aus. Soweit möglich, werden die den jeweiligen Hexadezimalwerten entsprechenden ASCII-Zeichen am rechten Rand angegeben. Ohne Angabe einer Start- und Endadresse beginnt die Ausgabe bei der Adresse &100.

Beispiel:

```
A>ddt
DDT VERS 2.2
-d
0100 01 BC 0F C3 8B 01 43 4F 50 59 52 49 47 48 54 20 ......COPYRIGHT
0110 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA
0120 4C 20 52 45 53 45 41 52 43 48 20 20 20 20 20 20 L RESEARCH
0130 44 44 54 20 56 45 52 53 20 32 2E 32 24 31 00 02 DDT VERS 2.2$1..
0140 C5 C5 11 30 01 0E 09 CD 05 00 C1 21 07 00 7E 3D ...0.......!..~=
0150 90 57 1E 00 D5 21 00 02 78 B1 CA 65 01 0B 7E 12 .W...!..x..e..~.
0160 13 23 C3 58 01 D1 C1 E5 62 78 B1 CA 87 01 0B 7B .#.X....bx.....{
0170 E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01 ...z..~#.o}.o...
0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 2A EC 01 22 E7 .....i.....*..".
0190 08 23 22 ED 08 3A EB 01 32 D5 0A 32 EA 0F 32 F4 .#"..:..2..2..2.
01A0 10 C3 3D 01 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ..=._..........2
01B0 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
```

F <Startadresse>, <Endadresse>, <Code>

füllt die durch die Start- und die Endadresse angegebenen Speicherzellen mit dem vereinbarten Code.

G [<Startadresse>, [,<Haltepunkt1>, [<Haltepunkt2>]]]

startet ein Programm bei der <Startadresse>. Werden Haltepunkte vereinbart, unterbricht der CPC den Programmlauf bei den angegebenen Adressen und meldet sich mit * <Haltepunkt-Adresse>. Der zweite Haltepunkt wird im allgemeinen als Sicherung vorgesehen, falls der erste übersprungen wird.

I <Dateiname>, [<.Dateityp>]

wird im Zusammenhang mit dem R-Kommando verwendet, um eine zu ladende Datei näher zu kennzeichnen. Als Dateikennung kann wahlweise <.HEX> oder <.COM> angegeben werden.

L [<Startadresse> [,<Endadresse>]]

disassembliert den Speicherinhalt ab der <Startadresse>. Wird keine <Endadresse> vereinbart, werden immer nur 11 aufeinanderfolgende Speicherinhalte disassembliert.

Beispiel:

```
-^C
-l
  0175  XTHL
  0176  MOV  A,M
  0177  INX  H
  0178  XTHL
  0179  MOV  L,A
  017A  MOV  A,L
  017B  RAL
  017C  MOV  L,A
  017D  JNC  0183
  0180  LDAX D
  0181  ADD  H
```

M <Startadresse>, <Endadresse>, <neue Startadresse>

kopiert den Speicherinhalt zwischen <Startadresse> und <Endadresse> in einen neuen Speicherbereich, beginnend mit <neue Startadresse>.

R[<Anfangsadresse>] liest eine Datei von Diskette, deren Name zuvor mit dem I-Kommando festgelegt wurde. Für Dateien des Typs <.HEX> muß die Anfangsadresse angegeben werden. <.COM>-Dateien starten unter CP/M immer bei &100. Bei Start des Dienstprogramms DDT lassen sich die beiden Befehle einfach anhängen.

S <Startadresse> ermöglicht den direkten Abruf eines unter <Startadresse> vereinbarten Speicherinhaltes. Eine Bestätigung durch ENTER läßt den Inhalt unverändert. Wird ein neuer Wert angegeben, wird der alte überschrieben. Dieser Modus wird durch Eingabe eines Dezimalpunktes oder bei Auftreten eines Fehlers beendet.

T[<Step>] schaltet den Trace-Modus ein, d.h. eine Programmablaufverfolgung unter Angabe der Registerinhalte und der Statusbits (siehe auch X-Kommando). Wird der Parameter <Step> nicht angegeben, springt DDT sofort wieder in die Kommandoebene zurück. Dabei wird die Aussprungadresse angegeben. Der Parameter <Step> legt die Anzahl von Programmschritten fest. Bei jedem Schritt werden die oben erwähnten Angaben gemacht.

Hinweis: Oft springen Programme Unterroutinen im ROM an, die den Computer zum Absturz bringen können. Sehen Sie sich deshalb das Programm vorher an (Kommando L), und entfernen Sie gegebenenfalls Unterprogrammaufrufe mit Adressen oberhalb von &A200.

U[<Step>] entspricht in seiner Wirkung dem Befehl T. Zwischenschritte beim U-Kommando werden nicht angegeben.

X <Register(-Paar)>

zeigt den Inhalt des im Parameter angegebenen Registers an. Der Cursor verbleibt bei der Ausgabe in derselben Zeile. Eingabe eines neuen Wertes verändert den Registerinhalt. Ohne Angabe eines Registers werden die Inhalte von F, A, BE, DE, HL, S und P in der Form

CxZxMxExTx A=xx B=xxxx D=xxxx
H=xxxx S=xxxx P=xxxx

ausgegeben. Es bedeutet:

C = Übertragsbit (0/1)
Z = Nullbit (0/1)
M = Vorzeichenbit (0/1)
E = Paritybit (0/1)
I = Zwischenübertrag (0/1)
A = Akkumulator (0 – FFFF)
B = Registerpaar BC (0 – FFFF)
D = Registerpaar DE (0 – FFFF)
H = Registerpaar HL (0 – FFFF)
S = Stapelzeiger (0 – FFFF)
P = Programmzähler (0 – FFFF)

Alle Registerwerte lassen sich durch Angabe des Registernamens und einem nachfolgenden Wert direkt ändern.

# Anhang K
# STAT-Kommandos

Das transiente Kommando STAT wird von der Diskette über den Aufruf

```
STAT
```

oder

```
STAT <Kommandozeile>
```

geladen. Mit seiner Hilfe sind statistische Informationen über Dateien und periphere Geräte abrufbar.

Im einzelnen gelten folgende Vereinbarungen:

**STAT** berechnet den Restspeicher *aller* aktiven Massenspeicher und gibt eine Meldung des Typs

d: R/W, SPACE: xxxK

oder

d: R/O, SPACE: xxxK

aus. Im einzelnen bedeuten:

R/W: Es kann gelesen und geschrieben werden.
R/O: Es kann nur gelesen werden.
d: Laufwerkbezeichnung (A:,B:,C: usw.)

Beispiel:

```
A>stat
A: R/W, Space: 28k
```

**STAT d:** Ausgabe der restlichen Speicherkapazität des Laufwerks d:

Beispiel:

```
A>stat a:
Bytes Remaining On A: 28k
```

STAT afn — führt zur Ausgabe einer Dateistatistik für alle Dateien, die die Kennung afn enthalten. Die Ausgabe erfolgt mit der Kopfzeile

Recs Bytes ext Acc

Beispiel:

```
A>stat a:*.com

 Recs  Bytes  Ext Acc
    2     1k    1 R/W A:AMSDOS.COM
   64     8k    1 R/W A:ASM.COM
   10     2k    1 R/W A:BOOTGEN.COM
   19     3k    1 R/W A:CHKDISC.COM
   15     2k    1 R/W A:CLOAD.COM
   21     3k    1 R/W A:COPYDISC.COM
   14     2k    1 R/W A:CSAVE.COM
   38     5k    1 R/W A:DDT.COM
   19     3k    1 R/W A:DISCCHK.COM
   21     3k    1 R/W A:DISCCOPY.COM
    4     1k    1 R/W A:DUMP.COM
   52     7k    1 R/W A:ED.COM
   22     3k    1 R/W A:FILECOPY.COM
   21     3k    1 R/W A:FORMAT.COM
   14     2k    1 R/W A:LOAD.COM
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
   61     8k    1 R/W A:SETUP.COM
   41     6k    1 R/W A:STAT.COM
   10     2k    1 R/W A:SUBMIT.COM
   12     2k    1 R/W A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A: 28k
```

STAT d: afn — wie zuvor, jedoch mit Angabe des Ziellaufwerks

STAT <Dateiname>$S — führt zu Ausgabe einer Dateistatistik mit Angabe der Dateigrößen. Das Ausgabeformat lautet:

Size Recs Bytes Ext Acc

Beispiel:

```
A>
  stat a:*.bas $S

 Size  Recs  Bytes  Ext Acc
    9     9     2k    1 R/W A:EX1.BAS
    3     3     1k    1 R/W A:EX2.BAS
Bytes Remaining On A: 28k
```

STAT d:<Dateiname> wie zuvor, jedoch unter Angabe des Ziellaufwerks

STAT d:<Dateiname> &R/O versetzt die angegebenen Dateien in einen Nur-Lese-Status. Bei der Ausgabe des Disketteninhaltsverzeichnisses wird dieser Status mitausgegeben. Ein Versuch, eine so geschützte Datei zu löschen oder zu überschreiben, führt zur Fehlermeldung Bdos ERR on d: File R/O

Beispiel:

```
A>stat *.dat $R/O
PARAM.DAT set to R/O
```

STAT d:<Dateiname> &R/O versetzt die angegebenen Dateien in einen Schreib-/Lese-Status.

STAT d:<Dateiname> $SYS versieht die angegebene Datei mit der Kennung .sys. Bei der Ausgabe des Inhaltsverzeichnisses sind so gekennzeichnete Dateien nicht mehr sichtbar.

STAT d:<Dateiname> $DIR hebt die zuvor genannte Dateikennung wieder auf.

STAT d:DSK: führt zur Ausgabe der Diskettenparameter des Laufwerks d:

Beispiel:

```
A>STAT DSK:

    A: Drive Characteristics
 1368: 128 Byte Record Capacity
  171: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
```

```
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   36: Sectors/ Track
    2: Reserved Tracks
```

STAT DSK: führt zur Ausgabe der Diskettenparameter für alle am System angeschlossenen Laufwerke.

STAT USR: gibt eine Liste der USER-Nummern für das aktive Laufwerk aus.

Beispiel:

```
A>stat usr:
Active User: 0
Active Files: 0
```

STAT VAL: führt zur Ausgabe der verfügbaren Status-Kommandos.

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

STAT DEV: führt zur Ausgabe der Zuordnung zwischen logischer (links) und physikalischer (rechts) Gerätebezeichnung.

Beispiel:

```
STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
```

# Anhang L
# Befehlssatz des Z80

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| 8E | ADC | A,(HL) |
| DD8E.. | ADC | A,(IX+d) |
| FD8E.. | ADC | A,(IY+d) |
| 8F | ADC | A,A |
| 88 | ADC | A,B |
| 89 | ADC | A,C |
| 8A | ADC | A,D |
| 8B | ADC | A,E |
| 8C | ADC | A,H |
| 8D | ADC | A,L |
| CE.. | ADC | A,n |
| ED4A | ADC | HL,BC |
| ED5A | ADC | HL,DE |
| ED6A | ADC | HL,HL |
| ED7A | ADC | HL,SP |
| 86 | ADD | A,(HL) |
| DD86.. | ADD | A,(IX+d) |
| FD86.. | ADD | A,(IY+d) |
| 87 | ADD | A,A |
| 80 | ADD | A,B |
| 81 | ADD | A,C |
| 82 | ADD | A,D |
| 83 | ADD | A,E |
| 84 | ADD | A,H |
| 85 | ADD | A,L |
| C6.. | ADD | A,n |
| 09 | ADD | HL,BC |
| 19 | ADD | HL,DE |
| 29 | ADD | HL,HL |
| 39 | ADD | HL,SP |
| DD09 | ADD | IX,BC |
| DD19 | ADD | IX,DE |
| DD29 | ADD | IX,IX |
| DD39 | ADD | IX,SP |
| FD09 | ADD | IY,BC |
| FD19 | ADD | IY,DE |
| FD29 | ADD | IY,IY |
| FD39 | ADD | IY,SP |
| A6 | AND | (HL) |
| DDA6.. | AND | (IX+d) |
| FDA6.. | AND | (IY+d) |
| A7 | AND | A |
| A0 | AND | B |
| A1 | AND | C |
| A2 | AND | D |
| A3 | AND | E |
| A4 | AND | H |
| A5 | AND | L |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| E6.. | AND | n |
| CB46 | BIT | 0,(HL) |
| DDCB..46 | BIT | 0,(IX+d) |
| FDCB..46 | BIT | 0,(IY+d) |
| CB47 | BIT | 0,A |
| CB40 | BIT | 0,B |
| CB41 | BIT | 0,C |
| CB42 | BIT | 0,D |
| CB43 | BIT | 0,E |
| CB44 | BIT | 0,H |
| CB45 | BIT | 0,L |
| CB4E | BIT | 1 (HL) |
| DDCB..4E | BIT | 1,(IX+d) |
| FDCB..4E | BIT | 1,(IY+d) |
| CB4F | BIT | 1,A |
| CB48 | BIT | 1,B |
| CB49 | BIT | 1,C |
| CB4A | BIT | 1,D |
| CB4B | BIT | 1,E |
| CB4C | BIT | 1,H |
| CB4D | BIT | 1,L |
| CB56 | BIT | 2,(HL) |
| DDCB..56 | BIT | 2,(IX+d) |
| FDCB..56 | BIT | 2,(IY+d) |
| CB57 | BIT | 2,A |
| CB50 | BIT | 2,B |
| CB51 | BIT | 2,C |
| CB52 | BIT | 2,D |
| CB53 | BIT | 2,E |
| CB54 | BIT | 2,H |
| CB55 | BIT | 2,L |
| CB5E | BIT | 3,(HL) |
| DDCB..5E | BIT | 3,(IX+d) |
| FDCB..5E | BIT | 3,(IY+d) |
| CB5F | BIT | 3,A |
| CB58 | BIT | 3,B |
| CB59 | BIT | 3,C |
| CB5A | BIT | 3,D |
| CB5B | BIT | 3,E |
| CB5C | BIT | 3,H |
| CB5D | BIT | 3,L |
| CB66 | BIT | 4,(HL) |
| DDCB..66 | BIT | 4,(IX+d) |
| FDCB..66 | BIT | 4,(IY+d) |
| CB67 | BIT | 4,A |
| CB60 | BIT | 4,B |
| CB61 | BIT | 4,C |
| CB62 | BIT | 4,D |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| CB63 | BIT | 4,E |
| CB64 | BIT | 4,H |
| CB65 | BIT | 4,L |
| CB6E | BIT | 5,(HL) |
| DDCB..6E | BIT | 5,(IX+d) |
| FDCB..6E | BIT | 5,(IY+d) |
| CB6F | BIT | 5,A |
| CB68 | BIT | 5,B |
| CB69 | BIT | 5,C |
| CB6A | BIT | 5,D |
| CB6B | BIT | 5,E |
| CB6C | BIT | 5,H |
| CB6D | BIT | 5,L |
| CB76 | BIT | 6,(HL) |
| DDCB..76 | BIT | 6,(IX+d) |
| FDCB..76 | BIT | 6,(IY+d) |
| CB77 | BIT | 6,A |
| CB70 | BIT | 6,B |
| CB71 | BIT | 6,C |
| CB72 | BIT | 6,D |
| CB73 | BIT | 6,E |
| CB74 | BIT | 6,H |
| CB75 | BIT | 6,L |
| CB7E | BIT | 7,(HL) |
| DDCB..7E | BIT | 7,(IX+d) |
| FDCB..7E | BIT | 7,(IY+d) |
| CB7F | BIT | 7,A |
| CB78 | BIT | 7,B |
| CB79 | BIT | 7,C |
| CB7A | BIT | 7,D |
| CB7B | BIT | 7,E |
| CB7C | BIT | 7,H |
| CB7D | BIT | 7,L |
| DC.... | CALL | C,nn |
| FC.... | CALL | M,nn |
| D4.... | CALL | NC,nn |
| C4.... | CALL | NZ,nn |
| F4.... | CALL | P,nn |
| EC.... | CALL | PE,nn |
| E4.... | CALL | PO,nn |
| CC.... | CALL | Z,nn |
| CD.... | CALL | nn |
| 3F | CCF | |
| BE | CP | (HL) |
| DDBE.. | CP | (IX+d) |
| FDBE.. | CP | (IY+d) |
| BF | CP | A |
| B8 | CP | B |
| B9 | CP | C |
| BA | CP | D |
| BB | CP | E |
| BC | CP | H |
| BD | CP | L |
| FE.. | CP | n |
| EDA9 | CPD | |
| EDB9 | CPDR | |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| EDB1 | CPIR | |
| EDA1 | CPI | |
| 2F | CPL | |
| 27 | DAA | |
| 35 | DEC | (HL) |
| DD35.. | DEC | (IX+d) |
| FD35.. | DEC | (IY+d) |
| 3D | DEC | A |
| 05 | DEC | B |
| 0B | DEC | BC |
| 0D | DEC | C |
| 15 | DEC | D |
| 1B | DEC | DE |
| 1D | DEC | E |
| 25 | DEC | H |
| 2B | DEC | HL |
| DD2B | DEC | IX |
| FD2B | DEC | IY |
| 2D | DEC | L |
| 3B | DEC | SP |
| F3 | DI | |
| 10.. | DJNZ | e |
| FB | EI | |
| E3 | EX | (SP),HL |
| DDE3 | EX | (SP),IX |
| FDE3 | EX | (SP),IY |
| 08 | EX | AF,AF' |
| EB | EX | DE,HL |
| D9 | EXX | |
| 76 | HALT | |
| ED46 | IM | 0 |
| ED56 | IM | 1 |
| ED5E | IM | 2 |
| ED78 | IN | A,(C) |
| ED40 | IN | B,(C) |
| ED48 | IN | C,(C) |
| ED50 | IN | D,(C) |
| ED58 | IN | E,(C) |
| ED60 | IN | H,(C) |
| ED68 | IN | L,(C) |
| 34 | INC | (HL) |
| DD34.. | INC | (IX+d) |
| FD34.. | INC | (IY+d) |
| 3C | INC | A |
| 04 | INC | B |
| 03 | INC | BC |
| 0C | INC | C |
| 14 | INC | D |
| 13 | INC | DE |
| 1C | INC | E |
| 24 | INC | H |
| 23 | INC | HL |
| DD23 | INC | IX |
| FD23 | INC | IY |
| 2C | INC | L |
| 33 | INC | SP |
| DB.. | IN | A,(n) |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| EDAA | IND | |
| EDBA | INDR | |
| EDA2 | INI | |
| EDB2 | INIR | |
| C3.... | JP | nn |
| E9 | JP | (HL) |
| DDE9 | JP | (IX) |
| FDE9 | JP | (IY) |
| DA.... | JP | C,nn |
| FA.... | JP | M,nn |
| D2.... | JP | NC,nn |
| C2.... | JP | NZ,nn |
| F2.... | JP | P,nn |
| EA.... | JP | PE,nn |
| E2.... | JP | PO,nn |
| CA.... | JP | Z,nn |
| 38.. | JR | C,e |
| 30.. | JR | NC,e |
| 20.. | JR | NZ,e |
| 28.. | JR | Z,e |
| 18.. | JR | e |
| 02 | LD | (BC),A |
| 12 | LD | (DE),A |
| 77 | LD | (HL),A |
| 70 | LD | (HL),B |
| 71 | LD | (HL),C |
| 72 | LD | (HL),D |
| 73 | LD | (HL),E |
| 74 | LD | (HL),H |
| 75 | LD | (HL),L |
| 36.. | LD | (HL),n |
| DD77.. | LD | (IX+d),A |
| DD70.. | LD | (IX+d),B |
| DD71.. | LD | (IX+d),C |
| DD72.. | LD | (IX+d),D |
| DD73.. | LD | (IX+d),E |
| DD74.. | LD | (IX+d),H |
| DD75.. | LD | (IX+d),L |
| DD36..20 | LD | (IX+d),n |
| FD77.. | LD | (IY+d),A |
| FD70.. | LD | (IY+d),B |
| FD71.. | LD | (IY+d),C |
| FD72.. | LD | (IY+d),D |
| FD73.. | LD | (IY+d),E |
| FD74.. | LD | (IY+d),H |
| FD75.. | LD | (IY+d),L |
| FD36..20 | LD | (IY+d),n |
| 32.... | LD | (nn),A |
| ED43.... | LD | (nn),BC |
| ED53.... | LD | (nn),DE |
| 22.... | LD | (nn),HL |
| DD22.... | LD | (nn),IX |
| FD22.... | LD | (nn),IY |
| ED73.... | LD | (nn),SP |
| 0A | LD | A,(BC) |
| 1A | LD | A,(DE) |
| 7E | LD | A,(HL) |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| DD7E.. | LD | A,(IX+d) |
| FD7E.. | LD | A,(IY+d) |
| 3A.... | LD | A,(nn) |
| 7F | LD | A,A |
| 78 | LD | A,B |
| 79 | LD | A,C |
| 7A | LD | A,D |
| 7B | LD | A,E |
| 7C | LD | A,H |
| ED57 | LD | A,I |
| 7D | LD | A,L |
| 3E.. | LD | A,n |
| ED5F | LD | A,R |
| 46 | LD | B,(HL) |
| DD46.. | LD | B,(IX+d) |
| FD46.. | LD | B,(IY+d) |
| 47 | LD | B,A |
| 40 | LD | B,B |
| 41 | LD | B,C |
| 42 | LD | B,D |
| 43 | LD | B,E |
| 44 | LD | B,H |
| 45 | LD | B,L |
| 06.. | LD | B,n |
| ED4B.... | LD | BC,(nn) |
| 01.... | LD | BC,nn |
| 4E | LD | C,(HL) |
| DD4E.. | LD | C,(IX+d) |
| FD4E.. | LD | C,(IY+d) |
| 4F | LD | C,A |
| 48 | LD | C,B |
| 49 | LD | C,C |
| 4A | LD | C,D |
| 4B | LD | C,E |
| 4C | LD | C,H |
| 4D | LD | C,L |
| 0E.. | LD | C,n |
| 56 | LD | D,(HL) |
| DD56.. | LD | D,(IX+d) |
| FD56.. | LD | D,(IY+d) |
| 57 | LD | D,A |
| 50 | LD | D,B |
| 51 | LD | D,C |
| 52 | LD | D,D |
| 53 | LD | D,E |
| 54 | LD | D,H |
| 55 | LD | D,L |
| 16.. | LD | D,n |
| ED5B.... | LD | DE,(nn) |
| 11.... | LD | DE,nn |
| 5E | LD | E,(HL) |
| DD5E.. | LD | E,(IX+d) |
| FD5E.. | LD | E,(IY+d) |
| 5F | LD | E,A |
| 58 | LD | E,B |
| 59 | LD | E,C |
| 5A | LD | E,D |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| 5B | LD | E,E |
| 5C | LD | E,H |
| 5D | LD | E,L |
| 1E.. | LD | E,n |
| 66 | LD | H,(HL) |
| DD66.. | LD | H,(IX+d) |
| FD66.. | LD | H,(IY+d) |
| 67 | LD | H,A |
| 60 | LD | H,B |
| 61 | LD | H,C |
| 62 | LD | H,D |
| 63 | LD | H,E |
| 64 | LD | H,H |
| 65 | LD | H,L |
| 26.. | LD | H,n |
| 2A.... | LD | HL,(nn) |
| 21.... | LD | HL,nn |
| ED47 | LD | I,A |
| DD2A.... | LD | IX,(nn) |
| DD21.... | LD | IX,nn |
| FD2A.... | LD | IY,(nn) |
| FD21.... | LD | IY,nn |
| 6E | LD | L,(HL) |
| DD6E.. | LD | L,(IX+d) |
| FD6E.. | LD | L,(IY+d) |
| 6F | LD | L,A |
| 68 | LD | L,B |
| 69 | LD | L,C |
| 6A | LD | L,D |
| 6B | LD | L,E |
| 6C | LD | L,H |
| 6D | LD | L,L |
| 2E.. | LD | L,n |
| ED4F | LD | R,A |
| ED7B.... | LD | SP,(nn) |
| F9 | LD | SP,HL |
| DDF9 | LD | SP,IX |
| FDF9 | LD | SP,IY |
| 31.... | LD | SP,nn |
| EDA8 | LDD | |
| EDB8 | LDDR | |
| EDA0 | LDI | |
| EDB0 | LDIR | |
| ED44 | NEG | |
| 00 | NOP | |
| B6 | OR | (HL) |
| DDB6.. | OR | (IX+d) |
| FDB6.. | OR | (IY+d) |
| B7 | OR | A |
| B0 | OR | B |
| B1 | OR | C |
| B2 | OR | D |
| B3 | OR | E |
| B4 | OR | H |
| B5 | OR | L |
| F6.. | OR | n |
| EDBB | OTDR | |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| EDB3 | OTIR | |
| ED79 | OUT | (C),A |
| ED41 | OUT | (C),B |
| ED49 | OUT | (C),C |
| ED51 | OUT | (C),D |
| ED59 | OUT | (C),E |
| ED61 | OUT | (C),H |
| ED69 | OUT | (C),L |
| D3.. | OUT | (n),A |
| EDAB | OUTD | |
| EDA3 | OUTI | |
| F1 | POP | AF |
| C1 | POP | BC |
| D1 | POP | DE |
| E1 | POP | HL |
| DDE1 | POP | IX |
| FDE1 | POP | IY |
| F5 | PUSH | AF |
| C5 | PUSH | BC |
| D5 | PUSH | DE |
| E5 | PUSH | HL |
| DDE5 | PUSH | IX |
| FDE5 | PUSH | IY |
| CB86 | RES | 0,(HL) |
| DDCB.. 86 | RES | 0,(IX+d) |
| FDCB.. 86 | RES | 0,(IY+d) |
| CB87 | RES | 0,A |
| CB80 | RES | 0,B |
| CB81 | RES | 0,C |
| CB82 | RES | 0,D |
| CB83 | RES | 0,E |
| CB84 | RES | 0,H |
| CB85 | RES | 0,L |
| CB8E | RES | 1,(HL) |
| DDCB.. 8E | RES | 1,(IX+d) |
| FDCB.. 8E | RES | 1,(IY+d) |
| CB8F | RES | 1,A |
| CB88 | RES | 1,B |
| CB89 | RES | 1,C |
| CB8A | RES | 1,D |
| CB8B | RES | 1,E |
| CB8C | RES | 1,H |
| CB8D | RES | 1,L |
| CB96 | RES | 2,(HL) |
| DDCB.. 96 | RES | 2,(IX+d) |
| FDCB.. 96 | RES | 2,(IY+d) |
| CB97 | RES | 2,A |
| CB90 | RES | 2,B |
| CB91 | RES | 2,C |
| CB92 | RES | 2,D |
| CB93 | RES | 2,E |
| CB94 | RES | 2,H |
| CB95 | RES | 2,L |
| CB9E | RES | 3,(HL) |
| DDCB.. 9E | RES | 3,(IX+d) |
| FDCB.. 9E | RES | 3,(IY+d) |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| CB9F | RES | 3,A |
| CB98 | RES | 3,B |
| CB99 | RES | 3,C |
| CB9A | RES | 3,D |
| CB9B | RES | 3,E |
| CB9C | RES | 3,H |
| CB9D | RES | 3,L |
| CBA6 | RES | 4,(HL) |
| DDCB..A6 | RES | 4,(IX+d) |
| FDCB..A6 | RES | 4,(IY+d) |
| CBA7 | RES | 4,A |
| CBA0 | RES | 4,B |
| CBA1 | RES | 4,C |
| CBA2 | RES | 4,D |
| DBA3 | RES | 4,E |
| CBA4 | RES | 4,H |
| CBA5 | RES | 4,L |
| CBAE | RES | 5,(HL) |
| DDCB..AE | RES | 5,(IX+d) |
| FDCB..AE | RES | 5,(IY+d) |
| CBAF | RES | 5,A |
| CBA8 | RES | 5,B |
| CBA9 | RES | 5,C |
| CBAA | RES | 5,D |
| CBAB | RES | 5,E |
| CBAC | RES | 5,H |
| CBAD | RES | 5,L |
| CBB6 | RES | 6,(HL) |
| DDCB..B6 | RES | 6,(IX+d) |
| FDCB..B6 | RES | 6,(IY+d) |
| CBB7 | RES | 6,A |
| CBB0 | RES | 6,B |
| CBB1 | RES | 6,C |
| CBB2 | RES | 6,D |
| CBB3 | RES | 6,E |
| CBB4 | RES | 6,H |
| CBB5 | RES | 6,L |
| CBBE | RES | 7,(HL) |
| DDCB..BE | RES | 7,(IX+d) |
| FDCB..BE | RES | 7,(IY+d) |
| CBBF | RES | 7,A |
| CBB8 | RES | 7,B |
| CBB9 | RES | 7,C |
| CBBA | RES | 7,D |
| CBBB | RES | 7,E |
| CBBC | RES | 7,H |
| CBBD | RES | 7,L |
| C9 | RET | |
| D8 | RET | C |
| F8 | RET | M |
| D0 | RET | NC |
| C0 | RET | NZ |
| F0 | RET | P |
| E8 | RET | PE |
| E0 | RET | PO |
| C8 | RET | Z |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| ED4D | RETI | |
| ED45 | RETN | |
| CB16 | RL | (HL) |
| DDCB..16 | RL | (IX+d) |
| FDCB..16 | RL | (IY+d) |
| CB17 | RL | A |
| CB10 | RL | B |
| CB11 | RL | C |
| CB12 | RL | D |
| CB13 | RL | E |
| CB14 | RL | H |
| CB15 | RL | L |
| 17 | RLA | |
| CB06 | RLC | (HL) |
| DDCB..06 | RLC | (IX+d) |
| FDCB..06 | RLC | (IY+d) |
| CB07 | RLC | A |
| CB00 | RLC | B |
| CB01 | RLC | C |
| CB02 | RLC | D |
| CB03 | RLC | E |
| CB04 | RLC | H |
| CB05 | RLC | L |
| 07 | RLCA | |
| ED6F | RLD | |
| CB1E | RR | (HL) |
| DDCB..1E | RR | (IX+d) |
| FDCB..1E | RR | (IY+d) |
| CB1F | RR | A |
| CB18 | RR | B |
| CB19 | RR | C |
| CB1A | RR | D |
| CB1B | RR | E |
| CB1C | RR | H |
| CB1D | RR | L |
| 1F | RRA | |
| CB0E | RRC | (HL) |
| DDCB..0E | RRC | (IX+d) |
| FDCB..0E | RRC | (IY+d) |
| CB0F | RRC | A |
| CB08 | RRC | B |
| CB09 | RRC | C |
| CB0A | RRC | D |
| CB0B | RRC | E |
| CB0C | RRC | H |
| CB0D | RRC | L |
| OF | RRCA | |
| ED67 | RRD | |
| C7 | RST | 00H |
| CF | RST | 08H |
| D7 | RST | 10H |
| DF | RST | 18H |
| E7 | RST | 20H |
| EF | RST | 28H |
| F7 | RST | 30H |
| FF | RST | 38H |
| DE.. | SBC | A,n |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| 9E | SBC | A,(HL) |
| DD9E.. | SBC | A,(IX+d) |
| FD9E.. | SBC | A,(IY+d) |
| 9F | SBC | A,A |
| 98 | SBC | A,B |
| 99 | SBC | A,C |
| 9A | SBC | A,D |
| 9B | SBC | A,E |
| 9C | SBC | A,H |
| 9D | SBC | A,L |
| ED42 | SBC | HL,BC |
| ED52 | SBC | HL,DE |
| ED62 | SBC | HL,HL |
| ED72 | SBC | HL,SP |
| 37 | SCF | |
| CBC6 | SET | 0,(HL) |
| DDCB..C6 | SET | 0,(IX+d) |
| FDCB..C6 | SET | 0,(IY+d) |
| CBC7 | SET | 0,A |
| CBC0 | SET | 0,B |
| CBC1 | SET | 0,C |
| CBC2 | SET | 0,D |
| CBC3 | SET | 0,E |
| CBC4 | SET | 0,H |
| CBC5 | SET | 0,L |
| CBCE | SET | 1,(HL) |
| DDCB..CE | SET | 1,(IX+d) |
| FDCB..CE | SET | 1,(IY+d) |
| CBCF | SET | 1,A |
| CBC8 | SET | 1,B |
| CBC9 | SET | 1,C |
| CBCA | SET | 1,D |
| CBCB | SET | 1,E |
| CBCC | SET | 1,H |
| CBCD | SET | 1,L |
| CBD6 | SET | 2,(HL) |
| DDCB..D6 | SET | 2,(IX+d) |
| FDCB..D6 | SET | 2,(IY+d) |
| CBD7 | SET | 2,A |
| CBD0 | SET | 2,B |
| CBD1 | SET | 2,C |
| CBD2 | SET | 2,D |
| CBD3 | SET | 2,E |
| CBD4 | SET | 2,H |
| CBD5 | SET | 2,L |
| CBD8 | SET | 3,B |
| CBDE | SET | 3,(HL) |
| DDCB..DE | SET | 3,(IX+d) |
| FDCB..DE | SET | 3,(IY+d) |
| CBDF | SET | 3,A |
| CBD9 | SET | 3,C |
| CBDA | SET | 3,D |
| CBDB | SET | 3,E |
| CBDC | SET | 3,H |
| CBDD | SET | 3,L |
| CBE6 | SET | 4,(HL) |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| DDCB..E6 | SET | 4,(IX+d) |
| FDCB..E6 | SET | 4,(IY+d) |
| CBE7 | SET | 4,A |
| CBE0 | SET | 4,B |
| CBE1 | SET | 4,C |
| CBE2 | SET | 4,D |
| CBE3 | SET | 4,E |
| CBE4 | SET | 4,H |
| CBE5 | SET | 4,L |
| CBEE | SET | 5,(HL) |
| DDCB..EE | SET | 5,(IX+d) |
| FDCB..EE | SET | 5,(IY+d) |
| CBEF | SET | 5,A |
| CBE8 | SET | 5,B |
| CBE9 | SET | 5,C |
| CBEA | SET | 5,D |
| CBEB | SET | 5,E |
| CBEC | SET | 5,H |
| CBED | SET | 5,L |
| CBF6 | SET | 6,(HL) |
| DDCB..F6 | SET | 6,(IX+d) |
| FDCB..F6 | SET | 6,(IY+d) |
| CBF7 | SET | 6,A |
| CBF0 | SET | 6,B |
| CBF1 | SET | 6,C |
| CBF2 | SET | 6,D |
| CBF3 | SET | 6,E |
| CBF4 | SET | 6,H |
| CBF5 | SET | 6,L |
| CBFE | SET | 7,(HL) |
| DDCB..FE | SET | 7,(IX+d) |
| FDCB..FE | SET | 7,(IY+d) |
| CBFF | SET | 7,A |
| CBF8 | SET | 7,B |
| CBF9 | SET | 7,C |
| CBFA | SET | 7,D |
| CBFB | SET | 7,E |
| CBFC | SET | 7,H |
| CBFD | SET | 7,L |
| CB26 | SLA | (HL) |
| DDCB..26 | SLA | (IX+d) |
| FDCB..26 | SLA | (IY+d) |
| CB27 | SLA | A |
| CB20 | SLA | B |
| CB21 | SLA | C |
| CB22 | SLA | D |
| CB23 | SLA | E |
| CB24 | SLA | H |
| CB25 | SLA | L |
| CB2E | SRA | (HL) |
| DDCB..2E | SRA | (IX+d) |
| FDCB..2E | SRA | (IY+d) |
| CB2F | SRA | A |
| CB28 | SRA | B |
| CB29 | SRA | C |
| CB2A | SRA | D |

| OBJ CODE | QUELLBEFEHL | |
|---|---|---|
| CB2B | SRA | E |
| CB2C | SRA | H |
| CB2D | SRA | L |
| CB3E | SRL | (HL) |
| DDCB.. 3E | SRL | (IX+d) |
| FDCB.. 3E | SRL | (IY+d) |
| CB3F | SRL | A |
| CB38 | SRL | B |
| CB39 | SRL | C |
| CB3A | SRL | D |
| CB3B | SRL | E |
| CB3C | SRL | H |
| CB3D | SRL | L |
| 96 | SUB | (HL) |
| DD96.. | SUB | (IX+d) |
| FD96.. | SUB | (IY+d) |
| 97 | SUB | A |
| 90 | SUB | B |
| 91 | SUB | C |
| 92 | SUB | D |
| 93 | SUB | E |
| 94 | SUB | H |
| 95 | SUB | L |
| D6.. | SUB | n |
| AE | XOR | (HL) |
| DDAE.. | XOR | (IX+d) |
| FDAE.. | XOR | (IY+d) |
| AF | XOR | A |
| A8 | XOR | B |
| A9 | XOR | C |
| AA | XOR | D |
| AB | XOR | E |
| AC | XOR | H |
| AD | XOR | L |
| EE.. | XOR | n |

*(Courtesy of Zilog Inc.)*

Anhang M

# Anschlußbelegungen von Drucker-, Erweiterungsstecker und Joystickanschluß sowie des Anschlusses für 2. Diskettenlaufwerk

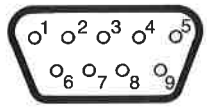

| | |
|---|---|
| Pin 1: $\overline{\text{STROBE}}$ | Pin 6: DATA4 |
| Pin 2: DATA0 | Pin 7: DATA5 |
| Pin 3: DATA1 | Pin 8: DATA6 |
| Pin 4: DATA2 | Pin 9: DATA7 |
| Pin 5: DATA3 | Pin 11: BUSY |

Pin 14,16 und 19 bis 33: Masse

*Abb. 1.2: Numerierung und Signalbelegung des Druckeranschlusses*

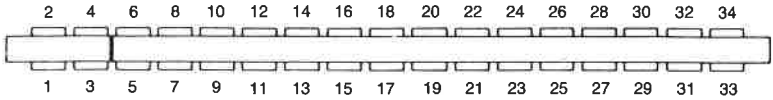

| Pin-Nr. | Signal | Pin.-Nr. | Signal | Pin.-Nr. | Signal |
|---|---|---|---|---|---|
| 1 | SOUND | 18 | A0 | 35 | INT |
| 2 | GND | 19 | D7 | 36 | NMI |
| 3 | A15 | 20 | D6 | 37 | BUSRD |
| 4 | A14 | 21 | D5 | 38 | BUSAK |
| 5 | A13 | 22 | D4 | 39 | READY |
| 6 | A12 | 23 | D3 | 40 | BUS RESET |
| 7 | A11 | 24 | D2 | 41 | RESET |
| 8 | A10 | 25 | D1 | 42 | ROMEN |
| 9 | A9 | 26 | D0 | 43 | ROMDIS |
| 10 | A8 | 27 | +5V | 44 | RAMRD |
| 11 | A7 | 28 | MREQ | 45 | RAMDIS |
| 12 | A6 | 29 | M1 | 46 | CURSOR |
| 13 | A5 | 30 | RFSH | 47 | LIGHT PEN |
| 14 | A4 | 31 | IORQ | 48 | EXP |
| 15 | A3 | 32 | RD | 49 | GND |
| 16 | A2 | 33 | WR | 50 | ∅ |
| 17 | A1 | 34 | HALT | | |

*Abb. 1.5: Belegung der Kontakte des Erweiterungssteckers*




| | |
|---|---|
| Pin 1: oben | Pin 6: Taster 2 |
| Pin 2: unten | Pin 7: Taster 1 |
| Pin 3: rechts | Pin 8: Masse |
| Pin 4: links | Pin 9: Masse 2 |
| Pin 5: – | |

*Abb. 1.7: Der Joystick-Anschluß des Schneider CPC 464/664*




ANSICHT VON HINTEN

| | | | |
|---|---|---|---|
| PIN 1 | READY | PIN 18 | GND |
| PIN 2 | GND | PIN 19 | MOTOR ON |
| PIN 3 | SIDE 1 SELECT | PIN 20 | GND |
| PIN 4 | GND | PIN 21 | N/C |
| PIN 5 | READ DATA | PIN 22 | GND |
| PIN 6 | GND | PIN 23 | DRIVE SELECT 1 |
| PIN 7 | WRITE PROTECT | PIN 24 | GND |
| PIN 8 | GND | PIN 25 | N/C |
| PIN 9 | TRACK 0 | PIN 26 | GND |
| PIN 10 | GND | PIN 27 | INDEX |
| PIN 11 | WRITE GATE | PIN 28 | GND |
| PIN 12 | GND | PIN 29 | N/C |
| PIN 13 | WRITE DATA | PIN 30 | GND |
| PIN 14 | GND | PIN 31 | N/C |
| PIN 15 | STEP | PIN 32 | GND |
| PIN 16 | GND | PIN 33 | N/C |
| PIN 17 | DIRECTION SELECT | PIN 34 | GND |

Anhang N

# Ausgewählte Einsprungadressen des Betriebssystems

| | |
|---|---|
| **&BB00** | Tastatur-Manager initialisieren (KM INITIALISE) |
| Aufgabe/Wirkung: | Löscht den Tastatur-Puffer. Setzt die Wiederholrate der Tasten auf den Standardwert. SHIFT- und CAPS LOCK werden ausgeschaltet. Löscht Tastaturdefinitionen. Unterbrechungen durch BREAK sind nicht möglich. |
| Einsprungbedingungen: | keine |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Die Registerinhalte von AF, BC, DE, HL werden zerstört. Alle anderen Register bleiben unverändert. |
| **&BB06** | Tastaturpufferabfrage mit Wartefunktion (KM RESET) |
| Aufgabe/Wirkung: | Fragt den Tastaturpuffer ab und wartet, bis eine Eingabe erfolgt. |
| Einsprungbedingungen: | keine |
| Übergabeparameter beim Aussprung: | A enthält den ASCII-Code des Zeichens. |
| Anmerkung: | Das Übertragsbit C wird gesetzt, alle anderen Statusbits werden zerstört. |
| **&BB09** | Tastaturpufferabfrage ohne Wartefunktion (KM READ CHARACTER) |
| Aufgabe/Wirkung: | Liest den Inhalt des Tastaturpuffers. |

Einsprungbedingungen: keine
Übergabeparameter beim Aussprung: A enthält das Zeichen oder ist zerstört.
Anmerkung: Übertragsbit ist nur gesetzt, wenn der Puffer ein Zeichen enthielt. Alle anderen Statusbits sind zerstört.

**&BB18** Tastaturabfrage mit Wartefunktion (KM WAIT KEY)

Aufgabe/Wirkung: Tastaturabfrage mit Wartefunktion.

Einsprungbedingungen: keine
Übergabeparameter beim Aussprung: A enthält das eingegebene Zeichen.
Anmerkung: Das Übertragsbit wird gesetzt, alle anderen Statusbits sind zerstört.

**&BB1E** Tasten- oder Tasterabfrage (KM TEST KEY)

Aufgabe/Wirkung: Die Routine ermittelt, ob eine vorgegebene Taste oder ein Taster eines Joysticks betätigt wurde.

Einsprungbedingungen: Der ASCII-Code muß nach A geladen werden.
Übergabeparameter beim Aussprung: C: 128 = CTRL wurde betätigt;
32 = SHIFT wurde betätigt;
160 = SHIFT & CTRL wurden betätigt.
Anmerkung: Übertragsbit wird zurückgesetzt.
Nullbit wird zurückgesetzt, wenn die angegebene Taste betätigt wurde. Andernfalls wird das Nullbit gesetzt. Alle anderen Statusbits sowie A und HL werden zerstört.

**&BB21** Tastaturstatus (KM GET STATE)

Aufgabe/Wirkung: Stellt fest, ob die SHIFT LOCK- oder die CAPS LOCK-Taste betätigt ist.

Einsprungbedingungen: keine
Übergabeparameter beim Aussprung: L enthält eine 0, wenn SHIFT LOCK ausgeschaltet, eine 255, wenn SHIFT LOCK eingeschaltet ist.
H enthält eine 0, wenn CAPS LOCK ausgeschaltet, eine 255, wenn CAPS LOCK eingeschaltet ist.
Anmerkung: A und alle Statusbits werden zerstört.

**&BB24** Joystick-Status
(KM GET JOYSTICK)

Aufgabe/Wirkung: Fragt den Status der Joysticks ab.

Aufrufparameter: keine
Übergabeparameter beim Aussprung: A und H enthalten den Status des Joysticks 0.
L enthält den Status des Joysticks 1.
Anmerkung: Bedeutung der Statusbits:

b7 b6 b5 b4 b3 b2 b1 b0
0 • F1 F2 R L U O

• = nicht benutzt
F1 = Feuerknopf 1
F2 = Feuerknopf 2
R = rechts
L = links
U = unten
O = oben

**&BB39** Wiederholfunktion
(KM SET REPEAT)

Aufgabe/Wirkung: Setzt/unterdrückt die Wiederholfunktion einer Taste.

Einsprungbedingungen: A muß mit dem Tastencode geladen werden.
B = 0, wenn Wiederholfunktion ausgeschaltet werden soll.
B = 255, wenn Wiederholfunktion aktiv werden soll.

Übergabeparameter beim Aussprung: keine
Anmerkung: A, B, C, H, L und die Statusbits werden zerstört.

**&BB3C** Test auf Wiederholfunktion (KM GET REPEAT)

Aufgabe/Wirkung: Abfrage des Wiederholstatus einer Taste.

Einsprungbedingungen: A muß mit dem Tastencode geladen werden.

Übergabeparameter beim Aussprung: Das Nullbit ist 0, wenn die Wiederholfunktion aktiv ist, andernfalls ist das Bit gesetzt. Das Übertragsbit wird zurückgesetzt.

Anmerkung: A und HL werden zerstört.

**&BB3F** Setzen von Verzögerung und Wiederholrate (KM SET DELAY)

Aufgabe/Wirkung: Die Routine setzt die Anfangsverzögerung und die Zeichenwiederholrate einer Taste.

Einsprungbedingungen: H wird mit dem Wert der Ansprechverzögerung geladen. L wird mit der Wiederholrate geladen. (Beide Werte entsprechen Vielfachen von 1/50 Sekunde, 0 = 255.)

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L und die Werte der Statusbits werden zerstört.

**&BB42** Abfrage von Ansprechverzögerung und Zeichenwiederholrate (KM GET DELAY)

Aufgabe/Wirkung: Die Routine fragt die aktuellen Werte der Ansprechverzögerung und der Zeichenwiederholrate ab.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: H enthält den Wert der Ansprechverzögerung. L enthält den Wert der Wiederholrate.

Anmerkung: A und F werden zerstört, alle anderen Register bleiben unverändert. Die in H und L stehenden Werte entsprechen ganzzahligen Vielfachen von 20 ms.

**&BB4E** Text VDU initialisieren (TXT INITIALISE)

Aufgabe/Wirkung: Initialisieren der Text-VDU. Alle Textparameter (INK, PAPER, WINDOW usw.) werden auf die Standardwerte zurückgesetzt. Anwenderdefinierte Zeichen gehen verloren.

Aufrufparameter: keine

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L sowie die Werte der Statusbits gehen verloren.

**&BB51** Text VDU zurücksetzen (TXT RESET)

Aufgabe/Wirkung: Die Routine setzt alle Control-Codes und Textparameter auf die Standardwerte.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte von A, B, C, D, E, H und L sowie die Werte der Statusbits gehen verloren.

**&BB54** Text-VDU aktivieren (TXT VDU ENABLE)

Aufgabe/Wirkung: Freigabe der Zeichenausgabe auf dem Bildschirm.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: keine

| | |
|---|---|
| Anmerkung: | Der Inhalt von A geht verloren. Die Werte der Statusbits werden zerstört. Der von TXT OUTPUT benutzte Control-Code-Puffer wird geleert. |
| **&BB57** | Text-VDU abschalten<br>(TXT VDU DISABLE) |
| Aufgabe/Wirkung: | Die Routine schaltet die TEXT-VDU ab. |
| Einsprungbedingungen: | keine |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Die Inhalte der Register A und F gehen verloren. Alle anderen Registerinhalte bleiben unverändert. Der Control-Code-Puffer wird geleert. |
| **&BB5A** | Zeichenübergabe an die VDU<br>(TXT OUTPUT) |
| Aufgabe/Wirkung: | Die Routine gibt ein Zeichen oder einen Steuercode auf dem aktiven Ausgabekanal aus. Nach dem Systemstart ist dies der Bildschirm. |
| Einsprungbedingungen: | A muß mit dem auszugebenden Zeichen oder Steuercode geladen werden. |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Alle Registerinhalte bleiben unverändert. |
| **&BB5D** | Zeichenausgabe<br>(TXT WR CHAR) |
| Aufgabe/Wirkung: | Die Routine gibt ein Zeichen auf dem Bildschirm aus. Control-Codes werden nicht beachtet. Der Cursor wird um eine Position weitergesetzt. |
| Einsprungbedingungen: | A muß das auszugebende Zeichen oder den Control-Code enthalten. |
| Übergabeparameter beim Aussprung: | keine |

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L sowie die Werte der Statusbits gehen verloren.

**&BB60** Zeichen lesen
(TXT RD CHAR)

Aufgabe/Wirkung: Es wird ein Zeichen an der aktuellen Position des Cursors eingelesen.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: Wenn an der Cursorposition ein Zeichen vorhanden ist, wird A mit dem Code geladen, das Übertragsbit wird gesetzt. Andernfalls sind Übertragsbit und der Inhalt von A gleich Null.

Anmerkung: Die Werte der übrigen Statusbits gehen verloren, die Inhalte der übrigen Register bleiben erhalten.

**&BB63** Textausgabe im Grafikmodus
(TXT SET GRAPHIC)

Aufgabe/Wirkung: Textausgabe im Grafikmodus ein- bzw. ausschalten.

Einsprungbedingungen: A <> 0 für Einschalten,
A = 0 für Ausschalten.

Übergabeparameter beim Aussprung: Der Inhalt des Registers A und die Werte der Statusbits gehen verloren.

Anmerkung: Control-Codes werden ausgegeben, aber nicht ausgeführt.

**&BB66** Textfenster setzen
(TXT WIN ENABLE)

Aufgabe/Wirkung: Die Routine setzt die Größe des aktuellen Textfensters. Der Cursor wird in die linke obere Ecke gesetzt.

Einsprungbedingungen: H: Spalteninformation für den linken Rand
D: Spalteninformation für den rechten Rand
L: Zeileninformation für den oberen Rand
E: Zeileninformation für den unteren Rand

Übergabeparameter
beim Aussprung: keine
Anmerkung: Die Registerinhalte A, B, C, D, E, H, L und die Werte der Statusbits gehen verloren.

**&BB6C** Fenster löschen
(TXT CLEAR WINDOW)

Aufgabe/Wirkung: Das aktuelle Fenster wird auf die Hintergrundfarbe und der Cursor in die linke obere Ecke gesetzt.

Einsprungbedingungen: keine
Übergabeparameter
beim Aussprung: keine
Anmerkung: Die Registerinhalte A, B, C, D, E, H, L und die Werte der Statusbits gehen verloren.

**&BB6F** Textspalte
(TXT SET COLUMN)

Aufgabe/Wirkung: Setzen der horizontalen Cursorkoordinate.

Einsprungbedingungen: A muß mit dem Spaltenwert geladen werden.
Übergabeparameter
beim Aussprung: keine
Anmerkung: Die Registerinhalte A, H und L sowie die Werte der Statusbits gehen verloren.

**&BB72** Textzeile
(TXT SET ROW)

Aufgabe/Wirkung: Setzt die vertikale Cursorkoordinate.

Einsprungbedingungen: A muß den Zeilenwert enthalten.
Übergabeparameter
beim Aussprung: keine
Anmerkung: Die Registerinhalte A, H und L sowie die Werte der Statusbits gehen verloren.

**&BB75** Cursor setzen
(TXT SET CURSOR)

Aufgabe/Wirkung: Positioniert den Cursor.

Einsprungbedingungen: H muß mit dem Spaltenwert und L mit dem Zeilenwert geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Spalten- wie auch Zeilenwert sind auf das Textfenster bezogen.
Die Registerinhalte A, H und L sowie die Werte der Statusbits gehen verloren.

**&BB78** Cursorposition lesen
(TXT GET CURSOR)

Aufgabe/Wirkung: Abfrage der aktuellen Cursorposition.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: H enthält die Spalten- und L die Zeileninformation.
A enthält einen Wert, der aussagt, wie oft das das aktuelle Fenster „gerollt" wurde.

Anmerkung: Die Werte der Statusbits gehen verloren, alle anderen Registerinhalte bleiben unverändert.

**&BB87** Text Cursorposition
(TXT VALIDATE)

Aufgabe/Wirkung: Die Routine testet, ob der Cursor sich an einer vorgegebenen Position innerhalb eines Fensters befindet. Ist dies nicht der Fall, wird jene Ausgabekoordinate ermittelt, bei der ein Zeichen bei der Ausgabe erscheinen würde.

Einsprungbedingungen: H muß mit dem Spalten-, L mit dem Zeilenwert der zu testenden Koordinate geladen werden.

Übergabeparameter beim Aussprung: H enthält die Spalten- und L die Zeileninformation.
Fall 1: Die in H und L enthaltenen Werte führen zu keinem Rollvorgang des Textes: Übertragsbit = 1, B geht verloren.
Fall 2: Die in H und L enthaltenen Werte führen bei der Zeichenausgabe zu einem Aufwärtsrollen:

Übertragsbit = 0, B = &FF.
Fall 3: Die in H und L enthaltenen Werte führen bei der Zeichenausgabe zu einem Abwärtsrollen:
Übertragsbit = 0, B = &00.

**&BB90** Vordergrundfarbe
(TXT SET PEN)

Aufgabe/Wirkung: Setzen der Stiftfarbe.

Einsprungbedingungen: A muß mit der Stiftfarbe geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Der Farbwert in A wird mit einer Maske logisch verknüpft, um in jedem Bildschirmmodus eine sichtbare Stiftfarbe zu gewährleisten.
Die Maskenwerte lauten:

MODE 0 = &0F
MODE 1 = &03
MODE 2 = &01

Die Inhalte der Register A, H, L und F gehen verloren.

**&BB96** Hintergrundfarbe
(TXT SET PAPER)

Aufgabe/Wirkung: Setzen der Hintergrundfarbe (PAPER).

Einsprungbedingungen: A muß mit dem Farbwert geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Der in A vereinbarte Farbwert wird mit einer Maske logisch verknüpft, um die für den Bildschirmmodus gültigen Farben zu gewährleisten.
Die Maskenwerte lauten:

MODE 0 = &0F
MODE 1 = &03
MODE 2 = &01

Die Inhalte der Register A, H, L und F gehen verloren.

**&BB9C** Farben invertieren (TXT INVERSE)

Aufgabe/Wirkung: Austauschen von Vorder- und Hintergrundfarbe.

Einsprungbedingungen: keine
Übergabeparameter beim Aussprung: keine
Anmerkung: Die Inhalte der Register A, H, L und F gehen verloren.

**&BBBA** Grafik-VDU initialisieren (GRA INITIALISE)

Aufgabe/Wirkung: Die Grafik-VDU wird auf die Standardwerte gesetzt.

Einsprungbedingungen: keine
Übergabeparameter beim Aussprung: keine
Anmerkung: Die Registerinhalte A, B, C, D, E, H, L und die Werte der Statusbits gehen verloren.

**&BBC0** Absolute Bewegung des Grafikcursors (GRA MOVE ABSOLUTE)

Aufgabe/Wirkung: Der Grafikcursor wird auf eine definierte Pixelposition gesetzt.

Einsprungbedingungen: DE muß mit der X-Koordinate geladen werden.
HL muß mit der Y-Koordinate geladen werden.
Übergabeparameter beim Aussprung: keine
Anmerkung: Die Registerinhalte A, B, C, D, E, H, L und die Werte der Statusbits gehen verloren.

**&BBC3** Relative Bewegung des Grafikcursors (GRA MOVE RELATIVE)

Aufgabe/Wirkung: Der Grafikcursor wird auf eine definierte Pixelposition relativ zur vorhergehenden gesetzt.

Einsprungbedingungen: DE muß mit dem X-Koordinatenoffset geladen werden.
HL muß mit dem Y-Koordinatenoffset geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte A, B, C, D, E, H, L und die Werte der Statusbits gehen verloren.

**&BBC6** Position des Grafikcursors lesen

Aufgabe/Wirkung: Die Routine ermittelt die aktuellen Koordinaten des Grafikcursors.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: DE enthält die X-Koordinate.
HL enthält die Y-Koordinate.

Anmerkung: Die Inhalte von A und F gehen verloren.

**&BBC9** Koordinaten-Nullpunkt setzen (GRA SET ORIGIN)

Aufgabe/Wirkung: Die Routine definiert die aktuelle Lage des Koordinaten-Nullpunktes für grafische Ausgaben.

Einsprungbedingungen: DE muß mit der X-Koordinate geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L und F gehen verloren.

**&BBCC** Abfrage Koordinaten-Nullpunkt (GRA GET ORIGIN)

Aufgabe/Wirkung: Die Routine ermittelt die aktuelle Lage des Koordinaten-Nullpunktes.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: DE enthält die X-Koordinate.
HL enthält die Y-Koordinate.

| | |
|---|---|
| Anmerkung: | Alle anderen Register bleiben unverändert. Die X- und die Y-Koordinate werden in Standardkoordinaten angegeben. Das Koordinatenpaar 0,0 entspricht der linken unteren Ecke des Bildschirms. |
| **&BBCF** | Fensterweite<br>(GRA WIN WIDTH) |
| Aufgabe/Wirkung: | Setzen des linken und des rechten Randes des Grafikfensters. |
| Einsprungbedingungen: | DE muß mit der X-Koordinate des linken Randes geladen werden.<br>HL muß mit der X-Koordinate des rechten Randes geladen werden. |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Die Registerinhalte A, B, C, D, E, H, L und F gehen verloren. |
| **&BBD2** | Fensterhöhe<br>(GRA WIN HEIGHT) |
| Aufgabe/Wirkung: | Setzen des unteren und des oberen Randes des Grafikfensters. |
| Einsprungbedingungen: | DE muß mit der X-Koordinate des unteren Randes geladen werden.<br>HL muß mit der X-Koordinate des oberen Randes geladen werden. |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Die Registerinhalte von A, B, C, D, E, H, L und F gehen verloren. |
| **&BBDB** | Grafikfenster löschen<br>(GRA CLEAR WINDOW) |
| Aufgabe/Wirkung: | Fenster auf Hintergrundfarbe setzen. |
| Einsprungbedingungen: | keine |
| Übergabeparameter beim Aussprung: | keine |
| Anmerkung: | Die Registerinhalte von A, B, C, D, E, H, L und F gehen verloren. |

**&BBDE** Vordergrundfarbe (Grafik)
(GRA SET PEN)

Aufgabe/Wirkung: Setzen der Vordergrundfarbe für den Grafikstift.

Einsprungbedingungen: A muß mit dem gewünschten Farbwert geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Der in A stehende Wert wird mit einer Maske logisch verknüpft, um eine für den Betriebsmodus gültige Vereinbarung zu gewährleisten.
Die Maskenwerte lauten:

MODE 0 = &0F
MODE 1 = &03
MODE 2 = &01

Die Registerinhalte von A und F gehen verloren.

**&BBE4** Hintergrundfarbe für Grafik
(GRA SET PAPER)

Aufgabe/Wirkung: Die Routine setzt das Grafikfenster auf eine definierte Hintergrundfarbe.

Einsprungbedingungen: A muß mit dem gewünschten Farbwert geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Farbangabe in A wird mit einem Maskenwert logisch verknüpft, um eine für den Betriebsmodus gültige Vereinbarung zu gewährleisten.
Die Maskenwerte lauten:

MODE 0 = &0F
MODE 1 = &03
MODE 2 = &01

Die Registerinhalte von A und F gehen verloren.

**&BBE7** Abfrage der Hintergrundfarbe (Grafik) (GRA GET PAPER)

Aufgabe/Wirkung: Mit der Routine wird die aktuelle Hintergrundfarbe abgefragt und an das aufrufende Programm übergeben.

Einsprungbedingungen: keine
Übergabeparameter: A enthält den Farbwert.
Anmerkung: Die Werte der Statusbits gehen verloren.

**&BBEA** Absolutes Setzen eines Grafikpunktes.

Aufgabe/Wirkung: Es wird ein Punkt an einer angegebenen Koordinate gesetzt.

Einsprungbedingungen: DE muß mit der X-Koordinate des Punktes geladen werden.
HL muß mit der Y-Koordinate geladen werden.

Übergabeparameter beim Aussprung: keine
Anmerkung: Die Inhalte der Register A, B, C, D, E H, L und die Werte der Statusbits gehen verloren.

**&BBED** Relatives Setzen eines Grafikpunktes (GRA PLOT RELATIVE)

Aufgabe/Wirkung: Es wird ein Grafikpunkt gesetzt, dessen Zielkoordinate durch einen Koordinatenoffset definiert wird.

Einsprungbedingungen: DE muß mit dem X-Koordinatenoffset des Punktes geladen werden.
HL muß mit dem Y-Koordinatenoffset geladen werden.

Übergabeparameter beim Aussprung: keine
Anmerkung: Die Inhalte der Register A, B, C, D, E H, L und die Werte der Statusbits gehen verloren.

**&BBF0** Abfrage eines Grafikpunktes (GRA TEST ABSOLUTE)

Aufgabe/Wirkung: Die Routine fragt einen in absoluten Koordinaten vorgegebenen Grafikpunkt auf dessen Vorder- bzw. Hintergrundfarbe ab, sofern er innerhalb des Grafikfensters liegt.

Einsprungbedingungen: DE muß die X-Koordinate enthalten.
HL muß die Y-Koordinate enthalten.

Übergabeparameter beim Aussprung: A enthält den Wert der Vorder- oder der Hintergrundfarbe des angesteuerten Punktes.

Anmerkung: Die Inhalte der Register BC, DE, HL und F gehen verloren.

**&BBF3** Abfrage eines Grafikpunktes (GRA TEST RELATIVE)

Aufgabe/Wirkung: Die Routine fragt einen durch einen Koordinatenoffset vorgegebenen Grafikpunkt auf dessen Vorder- bzw. Hintergrundfarbe ab, sofern er innerhalb des Grafikfensters liegt.

Einsprungbedingungen: DE muß den X-Koordinatenoffset enthalten.
HL muß den Y-Koordinatenoffset enthalten.

Übergabeparameter beim Aussprung: A enthält den Wert der Vorder- oder der Hintergrundfarbe des angesteuerten Punktes.

Anmerkung: Die Inhalte der Register BC, DE, HL und F gehen verloren.

**&BBF6** Grafiklinie absolut (GRA LINE ABSOLUTE)

Aufgabe/Wirkung: Die Routine zeichnet eine gerade Linie zwischen dem aktuellen Ort des Grafikcursors und einem vorgegebenen Zielpunkt, der durch eine absolute Koordinatenangabe definiert ist.

Einsprungbedingungen: DE muß mit der X-Koordinate des Zielpunktes geladen werden.
HL muß mit der Y-Koordinate des Zielpunktes geladen werden.

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L und F gehen verloren.

**&BBF9** Grafiklinie relativ
(GRA LINE RELATIVE)

Aufgabe/Wirkung: Die Routine zeichnet eine gerade Linie zwischen dem aktuellen Ort des Grafikcursors und einem vorgegebenen Zielpunkt, der durch einen Koordinatenoffset bestimmt wird.

Einsprungbedingungen: DE muß mit dem X-Koordinatenoffset des Zielpunktes geladen werden.
HL muß mit dem Y-Koordinatenoffset des Zielpunktes geladen werden.

Anmerkung: Die Inhalte der Register A, B, C, D, E, H, L und F gehen verloren.

**&BBFC** Textausgabe im Grafikmodus
(GRA WR CHAR)

Aufgabe/Wirkung: Gibt Text an der aktuellen Position des Grafikcursors aus.

Einsprungbedingungen: Der ASCII-Code des Zeichens muß im Akkumulator vorhanden sein.

Übergabeparameter beim Aussprung: keine

Anmerkung: Das Zeichen wird mit dem linken unteren Matrixpunkt an der Grafikcursorposition ausgegeben. Der Grafikcursor wird automatisch auf eine neue Zeichenposition bewegt.
Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BC0E** Bildschirmmodus
(SCR SET MODE)

Aufgabe/Wirkung: Bildschirmmodus 0, 1 oder 2 setzen.

Einsprungbedingungen: Im Akkumulator muß der entsprechende Modus vereinbart sein.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BC14** Bildschirm löschen
(SCR CLEAR)

Aufgabe/Wirkung: Bildschirmspeicher wird geleert.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BC32** Farbwert setzen
(SCR SET INK)

Aufgabe/Wirkung: Setzt die für die Vordergrundfarben benötigten Farbwerte.

Einsprungbedingungen: A muß die Farbnummer enthalten. B wird mit der ersten Farbe, C mit der zweiten Farbe für Farbwechsel geladen.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört. Wenn B und C voneinander abweichen, erfolgt ein periodischer Farbwechsel.

**&BC4D** Zeile rollen
(SCR HW ROLL)

Aufgabe/Wirkung: Bewegt den gesamten Bildschirminhalt um ein Zeichen nach oben oder unten.

Einsprungbedingungen: Abwärtsrollen: B muß den Wert 0 enthalten. Aufwärtsrollen: B muß einen Wert verschieden von 0 enthalten. A: Hintergrundfarbe für die neue Zeile.

Parameterübergabe beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BC59** Grafikmodus (SCR ACCESS)

Aufgabe/Wirkung: Bildschirmmodus für Grafik festlegen.

Einsprungbedingungen: A muß mit dem Schreibmodus geladen werden.
Hierbei gilt:
0: Normalmodus: Farbe wird durch INK bestimmt.
1: XOR-Modus: Farbe wird durch Exklusiv-ODER-Verknüpfung mit der alten Farbe gebildet.
2: UND-Modus: Farbe wird durch UND-Verknüpfung mit der alten Farbe gebildet.
3: ODER-Modus: Farbe wird durch ODER-Verknüpfung mit der alten Farbe gebildet.

Parameterübergabe beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BCA7** SOUND-Kanal zurücksetzen (SOUND RESET)

Aufgabe/Wirkung: Setzt den Status des Geräuschgeneratorchips zurück.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BCAA** Ton auf Kanal schalten (SOUND QUEUE)

Aufgabe/Wirkung: Lädt einen Ton in den SOUND-Puffer.

Einsprungbedingungen: HL muß mit der Adresse eines Tonereignisses geladen werden.

Übergabeparameter beim Aussprung: keine

Anmerkung: Der Inhalt von HL ist zerstört, wenn ein Tonereignis geladen wurde.
Das Übertragsbit ist in diesem Fall gesetzt, andernfalls zurückgesetzt.
Die Inhalte der Register A, B, C, D, E und IX gehen verloren. Alle übrigen Statusbits werden zerstört.

Bedeutung der SOUND-Register:

Byte 0: Kanal und Synchronisation
Byte 1: Einhüllende der Amplitude
Byte 2: Hüllkurve der Frequenzmodulation
Byte 3, 4: Tonhöhe
Byte 5: Rauschperiode
Byte 6: Startamplitude
Byte 7: Dauer oder Wiederholperiode

Bedeutung der Bits von Byte 0:

Bit 0: Kanal A
Bit 1: Kanal B
Bit 2: Kanal C
Bit 3: warte auf Kanal A
Bit 4: warte auf Kanal B
Bit 5: warte auf Kanal C
Bit 6: Haltefunktion
Bit 7: Warteschlange leeren

**&BCAD** Status der Warteschlange (SOUND CHECK)

Aufgabe/Wirkung: Abfrage der Warteschlange (SOUND-Puffer)

Einsprungbedingungen: A muß den Kanal spezifizieren
Bit 0 gesetzt: Kanal A
Bit 1 gesetzt: Kanal B
Bit 2 gesetzt: Kanal C

Übergabeparameter beim Aussprung: A: Kanalstatus
Bit 0 bis 2: freie Plätze im SOUND-Puffer
Bit 3: Kanal wartet auf Rendezvous mit A
Bit 4: Kanal wartet auf Rendezvous mit B

Bit 5: Kanal wartet auf Rendezvous mit C
Bit 6: Kanal wird angehalten
Bit 7: Der getestete Kanal ist aktiv

Anmerkung: Die Inhalte der Register B, C, D, E H und L gehen verloren. Alle Statusbits werden zerstört.

**&BCAD** SOUND-Puffer auffüllen
(SOUND ARM EVENT)

Aufgabe/Wirkung: Nachladen einer SOUND-Anweisung, wenn ein Platz im SOUND-Puffer frei wird.

Einsprungbedingungen: A muß mit der Kanalkennung geladen werden.
Es gilt:
Bit 0: Kanal A
Bit 1: Kanal B
Bit 2: Kanal C
HL muß mit der Adresse geladen werden, unter der das Ton- oder Geräuschereignis vereinbart wurde.

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Registerinhalte von A, B, C, D, E, H und L gehen verloren. Die Statusbits werden zerstört.

**&BCD3** Tonfreigabe
(SOUND RELEASE)

Aufgabe/Wirkung: Die mit einem Haltebit gekennzeichneten Ton- oder Geräuschereignisse einzelner Kanäle werden für die Wiedergabe freigegeben.

Einsprungbedingungen: A muß mit der Kanalkennung geladen werden.
Es gilt:
Bit 0: Kanal A
Bit 1: Kanal B
Bit 2: Kanal C

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte der Registerpaare AF, BC, DE, HL und IX gehen verloren.

**&BCB6** Tonereignis sperren (SOUND HOLD)

Aufgabe/Wirkung: Die Tonausgabe wird unterbrochen.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: Übertragsbit gesetzt, wenn bei der Ausführung eine aktuelle Tonausgabe unterbrochen wurde, ansonsten zurückgesetzt.

Anmerkung: Die Inhalte der Register A, B, C, H und L gehen verloren. Die Werte der Statusbits werden zerstört.

Die Tonausgabe wird unmittelbar durch Ausführung der Unterroutinen SOUND CHANNEL, SOUND RELEASE oder CONTINUE SOUND wieder aufgenommen.

**&BCB9** Tonausgabe wieder aufnehmen (CONTINUE SOUND)

Aufgabe/Wirkung: Alle angehaltenen Töne werden zur Ausgabe freigegeben.

Einsprungbedingungen: keine

Übergabeparameter beim Aussprung: keine

Anmerkung: Die Inhalte der Registerpaare AF, BC, DE, HL und IX gehen verloren.

**&BCBC** Lautstärke-Einhüllende (SOUND AMPL ENVELOPE)

Aufgabe/Wirkung: Es wird eine von 15 verschiedenen, vom Anwender frei definierbaren, Lautstärkeeinhüllenden aktiviert.

Einsprungbedingungen: A wird mit der Nummer der Hüllkurve geladen.
HL muß mit der Adresse geladen werden, bei der die Tabelle der Hüllkurvenparameter beginnt. Die Bytes dieser Tabelle sind folgendermaßen organisiert:
Byte 0: Anzahl der Hüllkurvenabschnitte.

Bytes 1,2,3: Parameter des ersten Abschnitts.
Bytes 4,5,6: Parameter des zweiten Abschnitts.
Bytes 7,8,9: Parameter des dritten Abschnitts.
Bytes 10,11,12: Parameter des vierten Abschnitts.
Bytes 13,14,15: Parameter des fünften Abschnitts.

Die Bytes der einzelnen Abschnitte müssen folgendermaßen belegt werden:

Byte 0: Schrittzahl
Byte 1: Schrittweite
Byte 2: Pausendauer

Übergabeparameter beim Aussprung: HL enthält die Tabellenadresse + 16, wenn die vereinbarte Hüllkurve korrekt ist. Im anderen Fall bleibt der Inhalt von HL unverändert. Die Inhalte der Register A, B und C bleiben unverändert, wenn die Hüllkurvenvereinbarung nicht korrekt ist. Andernfalls gehen die Inhalte verloren.
Das Übertragsbit ist gesetzt, wenn die Einhüllende korrekt vereinbart wurde, andernfalls zurückgesetzt.

Anmerkung: Die Inhalte von D und E gehen in jedem Fall verloren. Die Werte der oben nicht genannten Statusbits werden zerstört.

**&BCBF** Frequenzmodulation (SOUND TONE ENVELOPE)

Aufgabe/Wirkung: Es werden die Parameter für eine Frequenzmodulierende vereinbart.

Einsprungbedingungen: A muß mit der Hüllkurvennummer geladen werden.
HL enthält die Startadresse für eine Parametertabelle.

Übergabeparameter beim Aussprung: HL enthält die Tabellenadresse + 16, wenn die vereinbarte Hüllkurve korrekt ist. Im anderen Fall bleibt der Inhalt von HL unver-

ändert. Die Inhalte der Register A, B und C bleiben unverändert, wenn die Hüllkurvenvereinbarung nicht korrekt ist. Andernfalls gehen die Inhalte verloren.
Das Übertragsbit ist gesetzt, wenn die Einhüllende korrekt vereinbart wurde, andernfalls zurückgesetzt.

Anmerkung: Die Inhalte von D und E gehen in jedem Fall verloren. Die Werte der oben nicht genannten Statusbits werden zerstört.

**&BCC2** Adresse einer Lautstärkeeinhüllenden (SOUND A ADDRESS)

Aufgabe/Wirkung: Aufsuchen der Hüllkurvenparameter für eine angegebene Hüllkurvennummer.

Einsprungbedingungen: A muß mit der Hüllkurvennummer (Wert zwischen 1 und 15) geladen werden.

Übergabeparameter beim Aussprung: BC enthält die Anzahl von Bytes für die angegebene Hüllkurve, wenn diese korrekt vereinbart wurde. Andernfalls wird der Inhalt von BC nicht beeinflußt.

HL enthält die Adresse der Tabelle, wenn die Hüllkurve richtig vereinbart wurde. Im anderen Fall wird HL nicht beeinflußt.

Das Übertragsbit wird bei korrekter Einhüllender gesetzt, sonst zurückgesetzt.

Anmerkung: Der Inhalt von A geht verloren. Alle anderen Statusbitwerte werden zerstört.

**&BCC5** Adresse einer Frequenzmodulierenden (SOUND T ADDRESS)

Aufgabe/Wirkung: Aufsuchen der Hüllkurvenparameter für eine frequenzmodulierende Einhüllende.

Einsprungbedingungen: A muß mit der Hüllkurvennummer (Wert zwischen 1 und 15) geladen werden.

| | |
|---|---|
| Übergabeparameter beim Aussprung: | BC enthält die Anzahl von Bytes für die angegebene Hüllkurve, wenn diese korrekt vereinbart wurde. Andernfalls wird der Inhalt von BC nicht beeinflußt. |
| | HL enthält die Adresse der Tabelle, wenn die Hüllkurve richtig vereinbart wurde. Im anderen Fall wird HL nicht beeinflußt. |
| | Das Übertragsbit wird bei korrekter Einhüllender gesetzt, sonst zurückgesetzt. |
| Anmerkung: | Der Inhalt von A geht verloren. Alle anderen Statusbitwerte werden zerstört. |

Ausführliche Hinweise über diese und noch eine Fülle anderer ROM-Routinen finden Sie in der Schneider-Publikation: „Das komplette CPC 464 Betriebssystem, Firmware-Handbuch", SOFT 258.

Anhang O

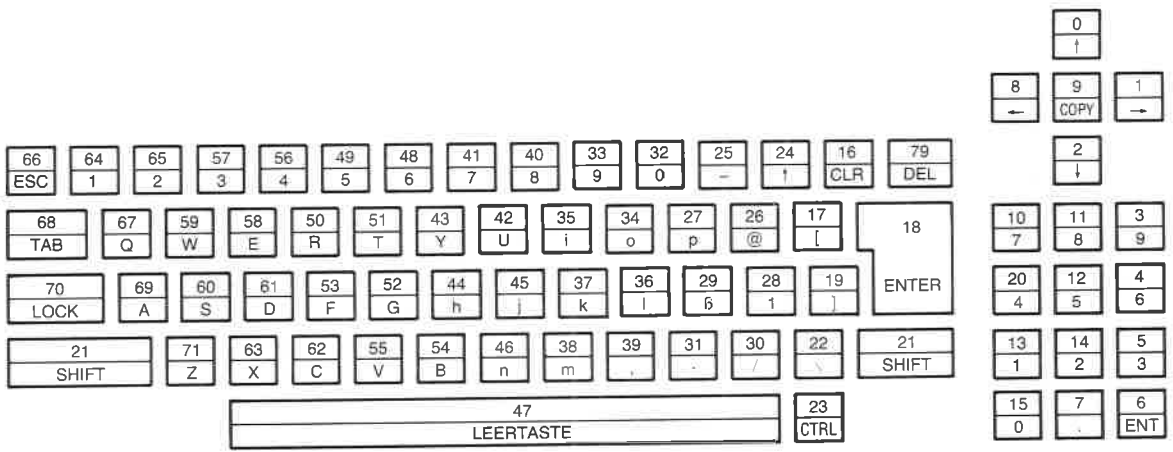
# Musikalische Notenwerte, deren Frequenzwerte und SOUND-Parameter

| Note | Frequenz | Parameter |
|---|---|---|
| $C_1$ | 32.703 | 1911 |
| $Cis_1$ | 34.648 | 1804 |
| $D_1$ | 36.708 | 1703 |
| $Dis_1$ | 38.891 | 1607 |
| $E_1$ | 41.203 | 1517 |
| $F_1$ | 43.654 | 1432 |
| $Fis_1$ | 46.249 | 1351 |
| $G_1$ | 48.999 | 1276 |
| $Gis_1$ | 51.913 | 1204 |
| $A_1$ | 55.000 | 1136 |
| $B_1$ | 58.270 | 1073 |
| $H_1$ | 61.735 | 1012 |
| C | 65.406 | 956 |
| Cis | 69.296 | 902 |
| D | 73.416 | 851 |
| Dis | 77.782 | 804 |
| E | 82.407 | 758 |
| F | 87.307 | 716 |
| Fis | 92.499 | 676 |
| G | 97.999 | 638 |
| Gis | 103.826 | 602 |
| A | 110.000 | 568 |
| B | 116.541 | 536 |
| H | 123.471 | 506 |
| c | 130.813 | 478 |
| cis | 138.591 | 451 |
| d | 146.832 | 478 |
| dis | 155.564 | 402 |
| e | 164.814 | 379 |
| f | 174.614 | 358 |
| fis | 184.997 | 338 |
| g | 195.998 | 319 |
| gis | 207.652 | 301 |
| a | 220.000 | 284 |
| b | 233.082 | 268 |
| h | 246.942 | 253 |
| $c^1$ | 261.626 | 239 |
| $cis^1$ | 277.183 | 225 |
| $d^1$ | 293.665 | 213 |
| $dis^1$ | 311.127 | 201 |
| $e^1$ | 329.628 | 190 |
| $f^1$ | 349.228 | 179 |
| $fis^1$ | 369.994 | 169 |
| $g^1$ | 391.995 | 159 |
| $gis^1$ | 415.305 | 150 |
| $a^1$ | 440.000 | 142 |
| $b^1$ | 466.164 | 134 |
| $h^1$ | 493.883 | 127 |
| $c^2$ | 523.251 | 119 |
| $cis^2$ | 554.365 | 113 |
| $d^2$ | 587.330 | 106 |
| $dis^2$ | 622.254 | 100 |
| $e^2$ | 659.255 | 95 |
| $f^2$ | 698.457 | 89 |
| $fis^2$ | 739.989 | 84 |
| $g^2$ | 783.991 | 80 |

| Note | Frequenz | Parameter |
|---|---|---|
| gis$^2$ | 830.609 | 75 |
| a$^2$ | 880.000 | 71 |
| b$^2$ | 932.328 | 67 |
| h$^2$ | 987.767 | 63 |
| c$^3$ | 1046.502 | 60 |
| cis$^3$ | 1108.731 | 56 |
| d$^3$ | 1174.659 | 53 |
| dis$^3$ | 1244.508 | 50 |
| e$^3$ | 1318.510 | 47 |
| f$^3$ | 1396.913 | 45 |
| fis$^3$ | 1479.978 | 42 |
| g$^3$ | 1567.982 | 40 |
| gis$^3$ | 1661.219 | 38 |
| a$^3$ | 1760.000 | 36 |
| b$^3$ | 1864.655 | 34 |
| h$^3$ | 1975.533 | 32 |
| c$^4$ | 2093.004 | 30 |
| cis$^4$ | 2217.461 | 28 |
| d$^4$ | 2349.318 | 27 |
| dis$^4$ | 2489.016 | 25 |
| e$^4$ | 2637.021 | 24 |
| f$^4$ | 2793.826 | 22 |
| fis$^4$ | 2959.955 | 21 |
| g$^4$ | 3135.963 | 20 |
| gis$^4$ | 3322.438 | 19 |
| a$^4$ | 3520.000 | 18 |
| b$^4$ | 3729.310 | 17 |
| h$^4$ | 3951.066 | 16 |
| c$^5$ | 4186.009 | 15 |
| cis$^5$ | 4434.922 | 14 |
| d$^5$ | 4698.636 | 13 |
| dis$^5$ | 4978.032 | 13 |
| e$^5$ | 5274.041 | 12 |
| f$^5$ | 5587.652 | 11 |
| fis$^5$ | 5919.911 | 11 |
| g$^5$ | 6271.927 | 10 |
| gis$^5$ | 6644.875 | 9 |
| a$^5$ | 7040.000 | 9 |
| b$^5$ | 7458.621 | 8 |
| h$^5$ | 7902.133 | 8 |

# Anhang P
# Tastencodes und Tastenbelegungen

| Tasten-bezeichnung | Positions-code | ASCII-Code | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | SHIFT | | CTRL | |
| | | dez. | hex. | dez. | hex. | dez. | hex. |
| ESC | 66 | 27 | 1B | 33 | - | - | - |
| 1 | 64 | 49 | 31 | 34 | 21 | - | - |
| 2 | 65 | 50 | 32 | 35 | 22 | 126 | 7E |
| 3 | 57 | 51 | 33 | 36 | 23 | - | - |
| 4 | 56 | 52 | 34 | 37 | 24 | - | - |
| 5 | 49 | 53 | 35 | 38 | 25 | - | - |
| 6 | 48 | 54 | 36 | 39 | 26 | - | - |
| 7 | 41 | 55 | 37 | 40 | 27 | - | - |
| 8 | 40 | 56 | 38 | 41 | 28 | - | - |
| 9 | 33 | 57 | 39 | 42 | 29 | - | - |
| 0 | 32 | 48 | 30 | 95 | 5F | 31 | 1F |
| - | 25 | 45 | 2D | 61 | 3D | - | - |
| ^ | 24 | 94 | 5E | 163 | A3 | 30 | 1E |
| CLR | 16 | 16 | 10 | 16 | 10 | 16 | 10 |
| DEL | 79 | 127 | 7F | 127 | 7F | 127 | 7F |
| TAB | 68 | 09 | 09 | 09 | 09 | 225 | E1 |
| Q | 67 | 113 | 71 | 81 | 51 | 17 | 11 |
| W | 59 | 119 | 77 | 87 | 57 | 23 | 17 |
| E | 58 | 101 | 65 | 69 | 45 | 05 | 05 |
| R | 50 | 114 | 72 | 82 | 52 | 18 | 12 |
| T | 51 | 116 | 74 | 84 | 54 | 20 | 14 |
| Y | 43 | 121 | 79 | 89 | 59 | 25 | 19 |
| U | 42 | 117 | 75 | 85 | 55 | 21 | 15 |
| I | 35 | 105 | 69 | 73 | 49 | 09 | 09 |
| O | 34 | 111 | 6F | 79 | 4F | 15 | 0F |
| P | 27 | 112 | 70 | 80 | 50 | 16 | 10 |
| @ | 26 | 64 | 40 | 124 | 7C | 00 | 00 |
| [ | 17 | 91 | 5B | 123 | 7B | 27 | 1B |
| ENT | 18 | 13 | 0D | 13 | 0D | 13 | 0D |




ZEICHENBELEGUNG DER TASTEN
UND DEREN TASTATURCODES

Anhang Q

# Adreßbelegung des Bildwiederholspeichers

```
Zeile  1     C000 ..... C04F
Zeile  2     C800 ..... C84F
Zeile  3     D000 ..... D04F
Zeile  4     D800 ..... D84F
Zeile  5     E000 ..... E04F
Zeile  6     E800 ..... E84F
Zeile  7     F000 ..... F04F
Zeile  8     F800 ..... F84F

Zeile  9     C050 ..... C09F
Zeile  10    C850 ..... C89F
Zeile  11    D050 ..... D09F
Zeile  12    D850 ..... D89F
Zeile  13    E050 ..... E09F
Zeile  14    E850 ..... E89F
Zeile  15    F050 ..... F09F
Zeile  16    F850 ..... F89F

Zeile  17    C0A0 ..... C0EF
Zeile  18    C8A0 ..... C8EF
Zeile  19    D0A0 ..... D0EF
Zeile  20    D8A0 ..... D8EF
Zeile  21    E0A0 ..... E0EF
Zeile  22    E8A0 ..... E8EF
Zeile  23    F0A0 ..... F0EF
Zeile  24    F8A0 ..... F8EF

Zeile  25    C0F0 ..... C13F
Zeile  26    C8F0 ..... C93F
```

```
Zeile   27     D0F0 ..... D13F
Zeile   28     D8F0 ..... D93F
Zeile   29     E0F0 ..... E13F
Zeile   30     E8F0 ..... E93F
Zeile   31     F0F0 ..... F13F
Zeile   32     F8F0 ..... F93F

Zeile   33     C140 ..... C18F
Zeile   34     C940 ..... C98F
Zeile   35     D140 ..... D18F
Zeile   36     D940 ..... D98F
Zeile   37     E140 ..... E18F
Zeile   38     E940 ..... E98F
Zeile   39     F140 ..... F18F
Zeile   40     F940 ..... F98F

Zeile   41     C190 ..... C1DF
Zeile   42     C990 ..... C9DF
Zeile   43     D190 ..... D1DF
Zeile   44     D990 ..... D9DF
Zeile   45     E190 ..... E1DF
Zeile   46     E990 ..... E9DF
Zeile   47     F190 ..... F1DF
Zeile   48     F990 ..... F9DF

Zeile   49     C1E0 ..... C22F
Zeile   50     C9E0 ..... CA2F
Zeile   51     D1E0 ..... D22F
Zeile   52     D9E0 ..... DA2F
Zeile   53     E1E0 ..... E22F
Zeile   54     E9E0 ..... EA2F
Zeile   55     F1E0 ..... F22F
Zeile   56     F9E0 ..... FA2F

Zeile   57     C230 ..... C27F
Zeile   58     CA30 ..... CA7F
Zeile   59     D230 ..... D27F
Zeile   60     DA30 ..... DA7F
Zeile   61     E230 ..... E27F
```

```
Zeile  62    EA30 ..... EA7F
Zeile  63    F230 ..... F27F
Zeile  64    FA30 ..... FA7F

Zeile  65    C280 ..... C2CF
Zeile  66    CA80 ..... CACF
Zeile  67    D280 ..... D2CF
Zeile  68    DA80 ..... DACF
Zeile  69    E280 ..... E2CF
Zeile  70    EA80 ..... EACF
Zeile  71    F280 ..... F2CF
Zeile  72    FA80 ..... FACF

Zeile  73    C2D0 ..... C31F
Zeile  74    CAD0 ..... CB1F
Zeile  75    D2D0 ..... D31F
Zeile  76    DAD0 ..... DB1F
Zeile  77    E2D0 ..... E31F
Zeile  78    EAD0 ..... EB1F
Zeile  79    F2D0 ..... F31F
Zeile  80    FAD0 ..... FB1F

Zeile  81    C320 ..... C36F
Zeile  82    CB20 ..... CB6F
Zeile  83    D320 ..... D36F
Zeile  84    DB20 ..... DB6F
Zeile  85    E320 ..... E36F
Zeile  86    EB20 ..... EB6F
Zeile  87    F320 ..... F36F
Zeile  88    FB20 ..... FB6F

Zeile  89    C370 ..... C3BF
Zeile  90    CB70 ..... CBBF
Zeile  91    D370 ..... D3BF
Zeile  92    DB70 ..... DBBF
Zeile  93    E370 ..... E3BF
Zeile  94    EB70 ..... EBBF
Zeile  95    F370 ..... F3BF
Zeile  96    FB70 ..... FBBF
```

```
Zeile   97     C3C0 ..... C40F
Zeile   98     CBC0 ..... CC0F
Zeile   99     D3C0 ..... D40F
Zeile   100    DBC0 ..... DC0F
Zeile   101    E3C0 ..... E40F
Zeile   102    EBC0 ..... EC0F
Zeile   103    F3C0 ..... F40F
Zeile   104    FBC0 ..... FC0F

Zeile   105    C410 ..... C45F
Zeile   106    CC10 ..... CC5F
Zeile   107    D410 ..... D45F
Zeile   108    DC10 ..... DC5F
Zeile   109    E410 ..... E45F
Zeile   110    EC10 ..... EC5F
Zeile   111    F410 ..... F45F
Zeile   112    FC10 ..... FC5F

Zeile   113    C460 ..... C4AF
Zeile   114    CC60 ..... CCAF
Zeile   115    D460 ..... D4AF
Zeile   116    DC60 ..... DCAF
Zeile   117    E460 ..... E4AF
Zeile   118    EC60 ..... ECAF
Zeile   119    F460 ..... F4AF
Zeile   120    FC60 ..... FCAF

Zeile   121    C4B0 ..... C4FF
Zeile   122    CCB0 ..... CCFF
Zeile   123    D4B0 ..... D4FF
Zeile   124    DCB0 ..... DCFF
Zeile   125    E4B0 ..... E4FF
Zeile   126    ECB0 ..... ECFF
Zeile   127    F4B0 ..... F4FF
Zeile   128    FCB0 ..... FCFF

Zeile   129    C500 ..... C54F
Zeile   130    CD00 ..... CD4F
Zeile   131    D500 ..... D54F
```

```
Zeile  132   DD00 ..... DD4F
Zeile  133   E500 ..... E54F
Zeile  134   ED00 ..... ED4F
Zeile  135   F500 ..... F54F
Zeile  136   FD00 ..... FD4F

Zeile  137   C550 ..... C59F
Zeile  138   CD50 ..... CD9F
Zeile  139   D550 ..... D59F
Zeile  140   DD50 ..... DD9F
Zeile  141   E550 ..... E59F
Zeile  142   ED50 ..... ED9F
Zeile  143   F550 ..... F59F
Zeile  144   FD50 ..... FD9F

Zeile  145   C5A0 ..... C5EF
Zeile  146   CDA0 ..... CDEF
Zeile  147   D5A0 ..... D5EF
Zeile  148   DDA0 ..... DDEF
Zeile  149   E5A0 ..... E5EF
Zeile  150   EDA0 ..... EDEF
Zeile  151   F5A0 ..... F5EF
Zeile  152   FDA0 ..... FDEF

Zeile  153   C5F0 ..... C63F
Zeile  154   CDF0 ..... CE3F
Zeile  155   D5F0 ..... D63F
Zeile  156   DDF0 ..... DE3F
Zeile  157   E5F0 ..... E63F
Zeile  158   EDF0 ..... EE3F
Zeile  159   F5F0 ..... F63F
Zeile  160   FDF0 ..... FE3F

Zeile  161   C640 ..... C68F
Zeile  162   CE40 ..... CE8F
Zeile  163   D640 ..... D68F
Zeile  164   DE40 ..... DE8F
Zeile  165   E640 ..... E68F
Zeile  166   EE40 ..... EE8F
```

```
Zeile  167   F640 ..... F68F
Zeile  168   FE40 ..... FE8F

Zeile  169   C690 ..... C6DF
Zeile  170   CE90 ..... CEDF
Zeile  171   D690 ..... D6DF
Zeile  172   DE90 ..... DEDF
Zeile  173   E690 ..... E6DF
Zeile  174   EE90 ..... EEDF
Zeile  175   F690 ..... F6DF
Zeile  176   FE90 ..... FEDF

Zeile  177   C6E0 ..... C72F
Zeile  178   CEE0 ..... CF2F
Zeile  179   D6E0 ..... D72F
Zeile  180   DEE0 ..... DF2F
Zeile  181   E6E0 ..... E72F
Zeile  182   EEE0 ..... EF2F
Zeile  183   F6E0 ..... F72F
Zeile  184   FEE0 ..... FF2F

Zeile  185   C730 ..... C77F
Zeile  186   CF30 ..... CF7F
Zeile  187   D730 ..... D77F
Zeile  188   DF30 ..... DF7F
Zeile  189   E730 ..... E77F
Zeile  190   EF30 ..... EF7F
Zeile  191   F730 ..... F77F
Zeile  192   FF30 ..... FF7F
Zeile  193   C780 ..... C7CF
Zeile  194   CF80 ..... CFCF
Zeile  195   D780 ..... D7CF
Zeile  196   DF80 ..... DFCF
Zeile  197   E780 ..... E7CF
Zeile  198   EF80 ..... EFCF
Zeile  199   F780 ..... F7CF
Zeile  200   FF80 ..... FFCF

Zeile  201   C7D0 ..... C81F
```

Hinweis: Die Adreßwerte

C7F0... C7FF
CFD0... CFFF
D7D0... D7FF
DFD0... DFFF
E7D0... E7FF
EFD0... EFFF
F7D0... F7FF
FFD0... FFFF

sind nicht belegt!

# Literaturverzeichnis


Aschoff, M.: *Was der CPC 464 alles kann*, Vogel-Buchverlag, Würzburg 1985

Englisch, L. u. a.: *CPC-464 Tips & Tricks*, Data Becker, Düsseldorf 1984

Englisch, L. u. a.: *CPC 464 INTERN*, Data Becker, Düsseldorf 1985

Hänsel, J. und Schrage, H.: *Basiswissen CPC 464*, Westermann, Braunschweig 1985

Kampow, F.: *Das BASIC-Trainingsbuch zum CPC 464*, Data Becker, Düsseldorf 1984

Kramer, S.: *Maschinensprache auf dem CPC 464*, Hueber-Verlag, München 1985

Krum, R. und Hund, R.: *Das große BASIC-Lexikon zum Schneider Computer CPC 464*, Heim Fachverlag, Darmstadt-Eberstadt 1985

Lüers, R.: *BASIC-Programme zum CPC 464*, Data Becker, Düsseldorf 1984

Schneider, H. L.: *Arbeiten mit dem Schneider CPC 464*, SYBEX-Verlag, Düsseldorf 1985

Straush, C.: *Der Schneider CPC 464*, Markt&Technik, München 1985

Szepanowski, N.: *CPC 464 für Einsteiger*, Data Becker 1984

Unger, E.: *Das Standard BASIC-Buch zum Schneider Computer CPC 464*, Heim Fachverlag, Darmstadt-Eberstadt 1984

Voß, W.: *BASIC leicht und schnell gelernt am Schneider Computer CPC 464*, Heim Fachverlag, Darmstadt-Eberstadt 1984

*Das Schulbuch zum CPC 464*, Data Becker, Düsseldorf 1984

Weminghoff, A.: *DuMont's Handbuch zum Schneider CPC*, DuMont, Köln 1985

Wittig, S.: *BASIC-Brevier für den Schneider CPC 464*, Heise Verlag, Hannover 1985

Wynford, J.: *BASIC auf dem CPC 464*, Hueber Verlag, München 1985

Zaks, R.: *Schneider CPC 464: Mein erstes BASIC Programm*, SYBEX-Verlag, Düsseldorf 1985

Zaks, R.: *Programmierung des Z80*, SYBEX-Verlag, Düsseldorf, 8. Auflage 1985

*CPC 464 Assemblerkurs*, Reihe: Mister Micro, SYBEX-Verlag, Düsseldorf 1985

Zaks, R.: *CP/M Handbuch*, SYBEX-Verlag, Düsseldorf 1984

Godden, B.: *Das komplette CPC 464 Betriebssystem – Firmware Handbuch*, Schneider Computer Division, Türkheim 1985

# Stichwortverzeichnis

# Die SYBEX Bibliothek

## Einführende Literatur

**MEIN ERSTER COMPUTER**
**von Rodnay Zaks** – Der unentbehrliche Wegweiser für jeden, der den Kauf oder den Gebrauch eines Mikrocomputers erwägt, das Standardwerk in 3., überarbeiteter Ausgabe. 304 Seiten, 150 Abbildungen, zahlreiche Illustrationen, Best.-Nr.: **3040** (1984)

**VORSICHT! Computer brauchen Pflege**
**von Rodnay Zaks** – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemeingültige Regeln für die pflegliche Behandlung Ihres Systems. 240 Seiten, 96 Abbildungen, Best.-Nr.: **3013** (1983)

**MEIN HEIMCOMPUTER**
**von N. Hesselmann** – zeigt, was ein Heimcomputer ist und was man mit ihm anfangen kann, von den Chips bis zu Tips für den Kauf. 256 Seiten, 124 Abb., Best.-Nr. **3064** (1985)

**CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik**
**von Rodnay Zaks** – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 2., überarbeitete und aktualisierte Ausgabe. 576 Seiten, 325 Abbildungen, Best.-Nr.: **3601** (1985)

**EINFÜHRUNG IN DIE TEXTVERARBEITUNG**
**von Hal Glatzer** – beschreibt, woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 248 Seiten, 67 Abbildungen, Best.-Nr. **3018** (1983)

**SYBEX MIKROCOMPUTER LEXIKON**
– die schnelle Informationsbörse! Über 1500 Definitionen, Kurzformeln, Begriffsschema der Mikroprozessor-Technik, englisch/deutsches und französisch/deutsches Wörterbuch, Bezugsquellen. 192 Seiten, Format 12,5 x 18 cm, Best.-Nr.: **3035** (1984)

**COMPUTER TOTAL VERRÜCKT**
**von Daniel Le Noury** – mit diesem Buch kommen Sie wieder zur Besinnung, nachdem Sie sich halbtot gelacht haben. Ca. 100 Cartoons rund um den Computer. 96 Seiten, Best.-Nr. **3042** (1984)

## Sprachen

### BASIC

**BASIC COMPUTER SPIELE/Band 1**
**herausgegeben von David H. Ahl** – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Best.-Nr. **3009**

**BASIC COMPUTER SPIELE/Band 2**
**herausgegeben von David H. Ahl** – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Best.-Nr.: **3010**

**BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK**
**von Alan Miller** – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

**PLANEN UND ENTSCHEIDEN MIT BASIC**
**von X. T. Bui** – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

**BASIC FÜR DEN KAUFMANN**
**von D. Hergert** – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. 208 Seiten, 85 Abbildungen, Best.-Nr.: **3026** (1983)

**GRUNDKURS IN BASIC**
**von U. Ströbel** – die Einführung in die meistgenutzte Programmiersprache für Lehrer, Schüler und das Selbststudium (Reihe SYBEX Informatik). 208 Seiten, mit Abb., Best.-Nr. **3058** (1985), Lehrerbegleitheft Best.-Nr. **3091** ( 1985)

## Pascal

**EINFÜHRUNG IN PASCAL UND UCSD/PASCAL**
**von Rodnay Zaks** – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

**DAS PASCAL HANDBUCH**
**von Jacques Tiberghien** – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3005** (1982)

**PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK**
**von Alan Miller** – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3007** (1982)

**50 PASCAL-PROGRAMME**
**von B. Hunter** – eine kommentierte Sammlung nützlicher Programme für Anwendungen im Geschäft und Privatbereich, für mathematische Anwendungen oder Spiele. Ca. 340 S., ca. 48 Abb., Best.-Nr. **3065** (erscheint 1985)

**GRUNDKURS IN PASCAL Bd. 1**
**von K.-H. Rollke** – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

**GRUNDKURS IN PASCAL BAND 2**
**von K.H. Rollke** – Mit diesem Buch wird der Pascal-Grundkurs aus der Reihe SYBEX Informatik abgerundet. Für Lehrer, Schüler, Teilnehmer an Pascal-Kursen, Studenten und Autodidakten. 224 Seiten, mit Abb., Best.-Nr. **3061** (1985), Lehrerbegleitheft Best.-Nr. **3090**

## Assembler

**PROGRAMMIERUNG DES Z80**
**von Rodnay Zaks** – ein umfassendes Nachschlagewerk zum Z80-Mikroprozessor – jetzt in einer durch Lösungen ergänzten Ausgabe. 2., erweiterte Ausgabe. 656 Seiten, 176 Abbildungen, Best.-Nr.: **3099** (1985)

**Z80 ANWENDUNGEN**
**von J. W. Coffron** – vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren. 296 Seiten, 204 Abbildungen, Best.-Nr.: **3037** (1984)

**PROGRAMMIERUNG DES 6502 mit 6510/65C02/65SC02**
**von Rodnay Zaks** – Programmierung in Maschinensprache mit dem Mikro-Prozessor 6502 und anderen Mitgliedern der 65xx Familie, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 3. überarbeitete und erweiterte Ausgabe. Ca. 450 Seiten, ca. 170 Abbildungen, Best.-Nr.: **3600** (1985)

**FORTGESCHRITTENE 6502-PROGRAMMIERUNG**
**von Rodnay Zaks** – hilft Ihnen, schwierige Probleme mit dem 6502 zu lösen, stellt Ihnen Maschinenroutinen zum Arbeiten mit einem Hobbyboard vor. 288 Seiten, 140 Abbildungen, Best.-Nr.: **3047** (1984)

**6502 ANWENDUNGEN**
**von Rodnay Zaks** – das Eingabe-/Ausgabe-Buch für Ihren 6502-Microprozessor. Stellt die meistgenutzten Programme und die dafür notwendigen Hardware-Komponenten vor. 288 Seiten, 213 Abbildungen, Best.-Nr.: **3014** (1983)

**PROGRAMMIERUNG DES 8086/8088**
**von J. W. Coffron** – lehrt Sie Programmierung, Kontrolle und Anwendung dieses 16-Bit-Mikroprozessors; vermittelt Ihnen das notwendige Wissen zu optimaler Nutzung Ihrer Maschine, von der internen Architektur bis hin zu fortgeschrittenen Adressierungstechniken. 312 Seiten, 107 Abbildungen, Best.-Nr.: **3050** (1984)

**PROGRAMMIERUNG DES 68000**
**von C. Vieillefond** – macht Sie mit dem 32-bit-Prozessor von leistungsstarken Rechnern wie Macintosh und Sinclair QL vertraut; erläutert die Struktur des 68000, den Aufbau des Speichers, die Adressierungsarten und den Befehlssatz. Ca. 460 Seiten, 150 Abb., Best.-Nr. **3060** (erscheint 1985)

# Spezielle Geräte

## Apple

**APPLE II BASIC HANDBUCH**
**von D. Hergert** – ein handliches Nachschlagewerk, das neben Ihren Apple II, II+ oder IIe stehen sollte. Dank vieler Tips und Vorschläge eine wesentliche Erleichterung fürs Programmieren. 304 Seiten, 116 Abbildungen, Best.-Nr. **3036** (1984)

**PROGRAMME FÜR MEINEN APPLE II**
**von S. R. Trost** – enthält eine Reihe von lauffähigen Programmen samt Listing und Beispiellauf. Hilft Ihnen, viele neue Anwendungen für Ihren APPLE II zu entdecken und erfolgreich einzusetzen. 192 Seiten, 158 Abbildungen, Best.-Nr.: **3029** (1983)

**WAS IST WO IM APPLE**
**von William F. Luebbert** – eine systematische Arbeitshilfe für Besitzer eines Apple II, II+ oder IIe mit vielen alphabetisch und numerisch sortierten Apple-Routinen und Speicheradressen, die Ihnen hilft, viel Arbeit und Zeit zu sparen. 496 Seiten, mit 84 Abb., Best.-Nr. **3098** (erscheint 1985)

**APPLE II/IIe ASSEMBLER KURS**
**Reihe MISTER MICRO** – Assembler-Programmierung auf dem Apple leicht gemacht. Das Buch vermittelt alle Instruktionen für den 6502-Prozessor. Der Assembler kann jederzeit für eigene Programme eingesetzt werden. 240 Seiten, Buch und Diskette, Best.-Nr. **3408** (1984)

## Commodore

**COMMODORE 64 BASIC HANDBUCH**
**von D. Hergert** – zeigt Ihnen alle Anwendungsmöglichkeiten Ihres C64 und beschreibt das vollständige BASIC-Vokabular anhand von praktischen Beispielen. 208 Seiten, 92 Abbildungen, Best. Nr.: **3048** (1984)

**FARBSPIELE MIT DEM COMMODORE 64**
**von W. Black und M. Richter** – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

**COMMODORE 64 – GRAFIK + DESIGN**
**von Ch. Platt** – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. 280 S., 150 Abb., teils vierfarbig. Best.-Nr. **3073** (1984)

**SPASS AN MATHE MIT DEM COMMODORE 64**
**von H. Danielsson** – zeigt Ihnen mit vielen Beispielen, wie der C 64 für schulische oder private Berechnungen genutzt werden kann. 280 Seiten mit Abb., Best.-Nr. **3072** ( 1985)

**COMMODORE 64 BASIC-KURS MIT HONEY-AID**
**Reihe MISTER MICRO** – BASIC auf dem C64 durch Praxis lernen; mit dem integrierten Lernpaket (Buch + Software). Außer vielen Übungsprogrammen: Honey-Aid – eine universell einsetzbare BASIC-Erweiterung mit 28 zusätzlichen Befehlen. 352 Seiten, Buch und Kassette, Best.-Nr. **3400**, Buch und Diskette, Best.-Nr. **3401** (1984)

**COMMODORE 64 ASSEMBLER-KURS**
**Reihe MISTER MICRO** – zeigt in Theorie und Praxis, wie Sie den 6510-Prozessor Ihres C64 programmieren. Der mitgelieferte Assembler ist universell einsetzbar. 296 Seiten, Buch und Kassette, Best.-Nr. **3402**, Buch und Diskette, Best.-Nr. **3403** (1984)

## Sinclair

**SINCLAIR ZX SPECTRUM BASIC HANDBUCH**
**von D. Hergert** – eine wichtige Hilfe für jeden SPECTRUM-Anwender. Gibt eine Übersicht aller BASIC-Begriffe, die auf diesem Rechner verwendet werden können, und erläutert sie ausführlich anhand von Beispielen. 288 Seiten, 188 Abbildungen, Best.-Nr.: **3027** (1983)

**PASCAL AUF DEM ZX-SPECTRUM**
**von R. Dupont/K.-H. Rollke/M. Szeliga** – zeigt, wie die leistungsstarke Programmiersprache Pascal auf einem preiswerten Rechner genutzt werden kann. Mit vielen Beispielen, für Einsteiger und Fortgeschrittene. 240 S., 56 Abb., Best.-Nr. **3087** (1985)

**SPECTRUM BASIC KURS**
**Reihe MISTER MICRO** – BASIC auf dem Spectrum schnell gelernt mit dem integrierten Lernpaket, das die Programmierung in Theorie und Praxis vermittelt. 312 Seiten, Buch und Kassette, Best.-Nr. **3409** (1984)

## Andere Geräte

**SCHNEIDER CPC 464: MEIN ERSTES BASIC PROGRAMM**
**von Rodnay Zaks** – zahlreiche farbige Illustrationen und viele Diagramme helfen Ihnen, auf spielerische Weise in BASIC zu programmieren; ohne Vorkenntnisse nutzbar. 208 Seiten, zahl. farb. Abb., Best.-Nr. **3096** (1985)

## Systemsoftware

**CP/M-HANDBUCH**
**on Rodnay Zaks** – das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M., 2. überarbeitete Ausgabe. 356 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

**PROGRAMMIEREN MIT CP/M**
**von A. R. Miller** – vermittelt die Feinheiten von CP/M und hilft, die Möglichkeiten dieses populären Betriebssystems zu erweitern. 424 Seiten, 97 Abb., Best.-Nr. **3077** (1985)

**MIKROPROZESSOR INTERFACE TECHNIKEN** (3. überarbeitete Ausgabe)
**von Rodnay Zaks/Austin Lesea** – Hardware- und Software-Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 432 Seiten, 400 Abbildungen, Format DIN A5, Best.-Nr.: **3012** (1982)

## Anwendungssoftware

**EINFÜHRUNG IN WORDSTAR**
**von Arthur Naiman** – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

SYBEX

# Mein Schneider CPC

Das Buch, das nach dem Handbuch kommt und neben jedem Schneider CPC liegen sollte. Sie finden eine Fülle wertvoller Informationen über den leistungsstarken Colour Personal Computer, die Sie in dieser gebündelten Form bislang vergeblich gesucht haben!

**Aus dem Inhalt**

- Der Schneider CPC und seine Hardware-Umgebung
- Programmieren in BASIC
- Umfangreiches BASIC-Lexikon
- Grafikentwurf und Fenstertechniken
- Musik- und Geräuscherzeugung
- Einführung in die Mikroprozessorentechnik und in die Maschinensprache
- Arbeiten mit dem Disketten-Laufwerk
- Betriebssysteme AMSDOS und CP/M 2.2

Über 10 Anhänge, zahlreiche Programmbeispiele und Abbildungen sowie viele Tips und Tricks runden das Buch ab.
**Mein Schneider CPC** ist eine unentbehrliche Hilfe für jeden, der dieses hervorragende Heimcomputersystem von Grund auf kennenlernen und beherrschen will.

**Über die Autoren**

**Dr.-Ing. Norbert Hesselmann** ist Mitglied des deutschen SYBEX-Lektorates und Autor zahlreicher Fachbuch- und Zeitschriften-Artikel zum Thema Mikrocomputer. Bei SYBEX publizierte er bisher **Mein DRAGON 32, Arbeiten mit dem Macintosh** und **Mein Heimcomputer.**

**Christoph Hesselmann,** bei Drucklegung Gymnasiast in Aachen und Computerfan, ist Gründer eines regionalen Computerclubs und Herausgeber einer Mikrocomputer-Zeitschrift.

ISBN 3-88745-602-5