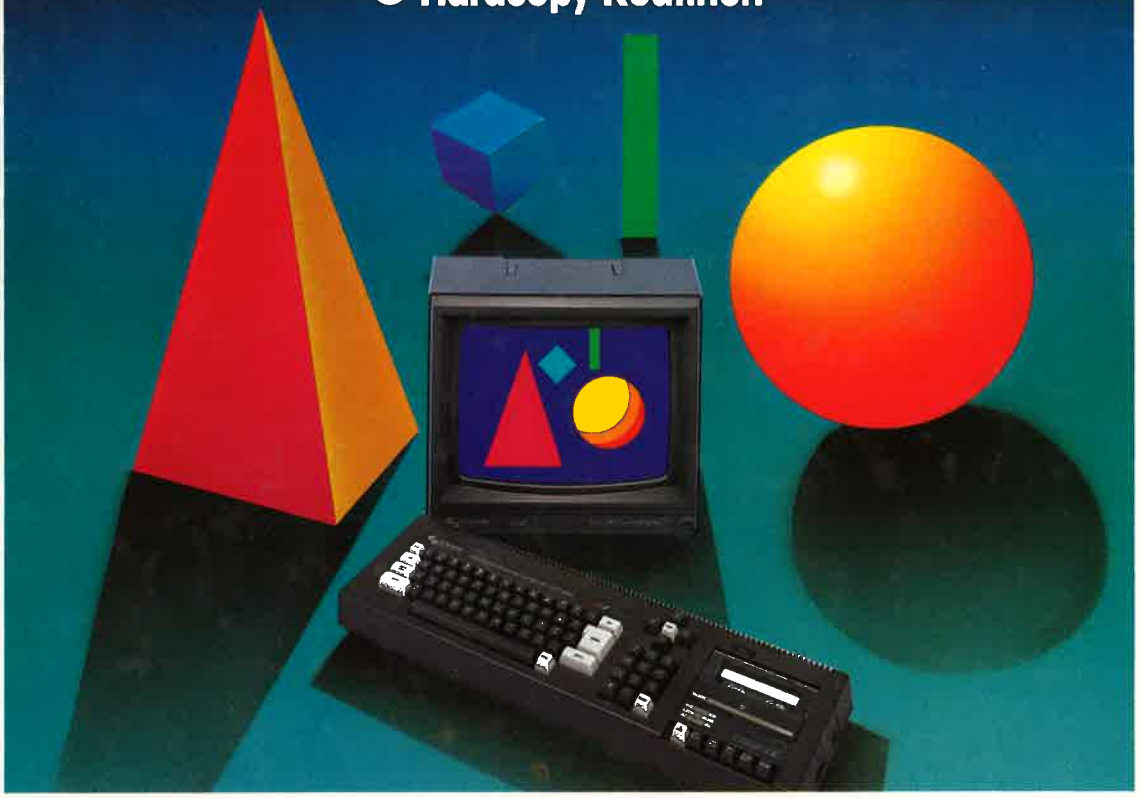
Markt&Technik

Carsten Straush

# Schneider CPC Grafik-Programmierung

**Die faszinierende Welt der Grafik, erklärt an zahlreichen Anwendungsbeispielen. Mit vielen Tips & Tricks: BASIC-Befehlserweiterung ● Sprites ● Hardcopy-Routinen**



Schneider CPC Grafik-Programmierung

Carsten Straush

# Schneider CPC Grafik-Programmierung

Die faszinierende Welt der Grafik, erklärt an zahlreichen Anwendungsbeispielen.

Mit vielen Tips & Tricks:

- BASIC-Befehlserweiterung
- Sprites
- Hardcopy-Routinen

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

**Straush, Carsten:**
Schneider CPC, Grafik-Programmierung: d. faszinierende Welt d. Grafik, erklärt an zahlr. Anwendungsbeispielen; mit vielen Tips & Tricks: BASIC-Befehlserw., Sprites, Hardcopy-Routinen / Carsten Straush. –
Haar bei München : Markt-und-Technik-Verlag, 1985.
ISBN 3-89090-182-4



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86

ISBN 3-89090-182-4


Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

# Inhaltsverzeichnis

**Wichtige Programmlistings**

# Vorwort

Eines der wohl interessantesten und auch faszinierendsten Gebiete bei Computern stellt ohne Zweifel die Grafik dar. Dies nicht nur, weil Sie vom ersten Einschalten der Maschine an mit dem Bildschirm und damit, was sich darauf abspielt, konfrontiert sind.

Schon bei den ersten Versuchen auf der neuen Maschine tauchen jedoch auch die ersten Fragen auf. Wie kann man auf dem Schneider möglichst einfach schöne Diagramme erzeugen? Wie können Titelbilder für Spiele geschrieben und abgespeichert werden? Welche Möglichkeiten bietet der Zeichensatz des CPC? Wie läuft die Ansprache der Grafik und die Farbgebung intern ab? Tausend Fragen, auf die das Schneiderhandbuch nur unvollkommen Antwort gibt.

Wenn man schließlich etwas Erfahrung mit seinem Computer erworben hat, so interessiert man sich dafür, mit welchen Tips und Tricks, der Überlagerung von Zeichen oder dem Rückgriff auf Maschinensprache sich die Grenzen dessen, was man bis jetzt im Grafikbereich erreichen konnte, noch etwas weiter hinausschieben lassen. In diesem Bereich ist man völlig auf sich allein gestellt.

Ich habe daher versucht in diesem Buch einen umfassenden Überblick darüber zu geben, was mit dem Schneider im Grafikbereich machbar ist. Einen kleinen Auszug aus dem Ergebnis meiner Experimente zeigt Ihnen ein Blick auf die Abbildungen in der Mitte dieses Buches. Ich hoffe, Sie haben ebensoviel Spaß daran, die vielen verschiedenen Routinen, Tips und Tricks auszuprobieren, wie es mir Freude bereitet hat, diese auszutüfteln.

Carsten Straush

# 1 Einleitung

CPC Colour Personal Computer ist ein Titel, der Ansprüche weckt. Dieses Buch hat es sich zum Ziel gesetzt, auf einer Reise durch alle Gebiete der Grafik und Grafikanwendungen auf dem Schneider zu untersuchen, was der Computer an grafischen Möglichkeiten bereithält.

Zunächst wenden wir uns in Kapitel 2 dem Bereich "Blockgrafik" zu. Viele Grafikbücher lassen dieses Thema aus ihren Darstellungen heraus. Der Grund dafür ist meist darin zu sehen, daß die dort behandelten Computer nicht über eine Umdefinition des Zeichensatzes und die daraus resultierenden Möglichkeiten verfügen.

Beim CPC ist dies anders. Alle 256 Zeichen, die der Computer auf dem Bildschirm darstellen kann, können vom Benutzer Bildpunkt für Bildpunkt umdefiniert werden, um neue Zeichen zu erzeugen. Sei es einen deutschen Zeichensatz oder neue Grafiksymbole für eine Spielidee. Und noch ein weiterer Punkt ist in Zusammenhang mit der Zeichendarstellung zu sehen. Normalerweise ist diese Art der Bildschirmdarstellung nur zweifarbig (Vordergrund- auf Hintergrundfarbe) möglich. Durch Überlagerungseffekte können beim CPC jedoch maximal 16 Farben in einem Zeichen untergebracht werden. Damit ist es dann sogar möglich, BASIC-Shapes, jene kleinen Männchen und Multicolourfiguren, die Sie alle aus Action- und Adventurespielen kennen, zu realisieren. Durch die transparente Darstellung von Zeichen oder die Zusammenfassung von Blockgrafiksymbolen zu Diagrammen ergeben sich weitere interessante Anwendungen in diesem Grafikbereich.

Wir gehen nacheinander auf die drei Punkte *Zeichensatz* und seine Umdefinition, *Zeichendarstellung* und *Farbgebung* und schließlich auf die Bedeutung der Steuercodes ein und nennen einige Tips und Tricks, mit deren Hilfe sich hier interessante Grafikmöglichkeiten entwickeln lassen. Daneben werden wir in diesem Kapitel auch ein Programm entwickeln, das uns beim Erstellen und Abspeichern von Blockgrafiken für Titelbilder und ähnliche Anwendungen Hilfestellung leistet.

Als nächstes beschäftigen wir uns mit der hochauflösenden Grafik. Wir lernen sie in Kapitel 3 anhand von einfachen Anwendungen aus dem

Bereich der grafischen Statistik kennen. Wir zeigen, wie wir aus Linien und Punkten größere Diagramme, Kreise, Säulen, zwei- und dreidimensionale Figuren und Abbildungen erzeugen können. Ein weiterer Punkt in diesem Bereich ist ein Programm, das fast jede Funktion grafisch darstellen kann. Wir werden uns damit beschäftigen, wie man Trends mit Hilfe von gleitenden Durchschnitten ermitteln kann.

Daneben lassen wir uns vom Computer bei der Erstellung von Diagrammen helfen. Mit einigen Hilfsroutinen wählt der Computer automatisch den optimalen Maßstab und ermöglicht so eine variable Darstellung, die immer einen optimalen Ausschnitt aus allen möglichen Diagrammen präsentiert.

Die hochauflösende Grafik können wir beim Schneider aber noch in vielen anderen Anwendungsgebieten benutzen. Eines davon ist die Gestaltung von Titelgrafiken. Um Anwendungen aus diesem Bereich, um das Malen und Zeichnen mit Hilfe der hochauflösenden Grafik auf dem Schirm unseres Computers geht es im Kapitel 4. Wir entwickeln Stück für Stück ein Malprogramm aus verschiedenen Modulen. Mit vielen Hilfsfunktionen und Unterroutinen, wie zum Beispiel dem automatischen Zeichnen von Figuren, sind wir dann in der Lage, sehr komfortabel eigene Bilder zu entwickeln.

In Kapitel 5 fügen wir dann unser Wissen aus dem Bereich der hochauflösenden BASIC-Grafik mit den Anwendungen und Tricks aus der Blockgrafik zusammen. Wir beschäftigen uns mit der Kombination beider Bereiche. Themen sind hier bewegte Grafiken für Spiele, oder Spezialeffekte wie zum Beispiel simulierte Bewegungen durch Farbwechsel, das Scrollen von Bildschirmen, das Abspeichern einer Spiellandschaft etc. Mit einer ganzen Reihe von Tips und Tricks runden wir dann die Benutzung der Grafikmöglichkeiten von BASIC aus ab.

Der darauf folgende zweite Teil dieses Buches ist dann der Maschinengrafik gewidmet. Dies heißt nun nicht, daß nur in Maschinensprache programmiert wird und die vorgestellten Programme daher auch nur von Maschinensprache aus "bedienbar sind". Aber wir müssen auf einige Systembefehle zurückgreifen, um neue Darstellungsarten realisieren zu können.

Als erstem Themenbereich gehen wir in Kapitel 6 der Abspeicherung der Grafik auf den Grund. Hier wird erklärt, wie der Grafikspeicher in den verschiedenen Darstellungsmodi strukturiert ist und wie die einzelnen Bildpunkte mit dazugehöriger Farbe abgelegt sind. Dazu unternehmen wir noch einen kleinen Exkurs in die Computer-Hardware und schauen uns

an, wie die einzelnen Bausteine des CPC zusammenwirken, um den Inhalt des Grafikspeichers auf dem Bildschirm darzustellen.

Mit dem so erworbenen Hintergrundwissen sind wir nun in Kapitel 7 in der Lage, auf den Grafikspeicher selbst zuzugreifen. Wir beschäftigen uns mit den Möglichkeiten eines Direktzugriffs auf den Grafikspeicher. Im ersten Schritt werden wir dabei noch eine Reihe von Routinen des Betriebssystems benutzen, Unterprogramme, die der CPC auch selbst permánent anwendet um zum Beispiel eine Linie auf dem Bildschirm zu ziehen oder einen Punkt zu testen. Wir werden uns damit beschäftigen, wie wir neue Grafikbefehle definieren können und auch sonst einiges Wissenswertes über die Anwendung von Systemroutinen kennenlernen.

In Kapitel 8 arbeiten wir dann schließlich völlig ohne Netz. Wir lassen auch noch das letzte Hilfsmittel, das Betriebssystem unbenutzt und greifen Punkt für Punkt auf jede Position des Grafikspeichers und dementsprechend natürlich auch unseres Bildschirms zu. Mit Hilfe dieses Wissens ist es uns dann möglich, richtige Software-Sprites, die sogenannten Shapes zu erstellen und zu bewegen. Dabei erreichen wir eine völlig andere Dimension, was die farblichen Möglichkeiten und auch die Zeichenauflösung betrifft. Und dabei werden wir auch noch deutlich schneller. Die Programmierung richtiger Action-Spiele rückt greifbar nahe. Wir werden uns mit der gleichzeitigen Bewegung von mehreren Shapes und mit der Koordination von Bildschirmansprache und Tastaturabfrage beschäftigen, ein wichtiger Punkt, wenn es auf dem Bildschirm schnell zugehen soll.

Im Anhang befindet sich schließlich noch eine Übersicht mit Systemroutinen.

So begeben wir uns nun an den Anfangspunkt unserer Reise, der zugleich auch schon die erste Haltestelle darstellt, den Bereich der BASIC-Grafik und hier speziell das Thema Blockgrafik und Zeichensatz.

# 2 Blockgrafik und Zeichensatz

Im ersten Teil dieses Buches, der Beschäftigung mit der BASIC-Grafik, werden wir sowohl die Möglichkeiten der Blockgrafik und des Zeichensatzes, als auch die der hochauflösenden Grafik kennenlernen. Zu Beginn unserer Betrachtungen wollen wir uns daher einmal klar machen, womit wir es bei den beiden Bereichen überhaupt zu tun haben.

Im Bereich der Zeichendarstellung operieren wir mit einer sehr kompakten und komplexen Art der Bildschirmansprache. Bei jeder Ausgabe wird eine ganze Gruppe von Bildpunkten gleichzeitig verändert und auf den Wert eines vorab fix definierten Zeichens gesetzt.

Etwas anders liegt der Sachverhalt bei der hochauflösenden Grafik. Hier sprechen wir einzelne Bildpunkte beziehungsweise eine Reihe von Bildpunkten als Linie separat, also einzeln, an. Mit der hochauflösenden Grafik ist also grob gesprochen eine gezieltere Ansprache einzelner Bildpunkte möglich, während mit der Zeichenausgabe im Bereich der Blockgrafik immer bereits die Änderung eines größeren Teils des Bildschirms erfolgt.

Im Verlauf dieses Buches werden wir also, beginnend mit relativ großen Änderungen im Bereich der Zeichendefinition und Blockgrafik über die einzelnen Bildpunkte zu einer immer gezielteren und detaillierteren Änderung des Bildschirminhalts kommen, bis wir schließlich am Schluß sogar auf jede einzelne Speicherstelle zugreifen werden, um ganz gezielt noch die Farbe eines Bildpunktes zu ändern.

Diese Unterteilung bezieht sich jedoch immer im wesentlichen auf die Art der Bildschirmansprache, also die Anzahl der geänderten Bildpunkte. In allen drei Varianten ist bei Ausnutzung sämtlicher Tricks und Schliche dieselbe Auflösung realisierbar und auch die Anzahl der benutzbaren Farben ist in allen drei Varianten gleich. Trotzdem sind die verschiedenen Darstellungsarten natürlich nicht identisch. Jede hat ihre ganz besonderen Vor- und Nachteile.

## 2.1 Der Zeichensatz und seine Umdefinition

Wenn wir uns mit der Grafikansprache im Bereich der Blockgrafik und Zeichendarstellung beschäftigen wollen, so müssen wir zunächst einmal untersuchen, wie wir überhaupt Zeichen ausgeben können. Beim CPC, wie auch bei den meisten anderen Homecomputern, gibt es dazu zwei Möglichkeiten. Zum einen kann das Zeichen "direkt" mit dem PRINT-Kommando ausgedruckt werden, zum Beispiel mit

```
PRINT"A"
```

Dieser Befehl druckt ein großes A auf dem Bildschirm aus. Daneben ist es jedoch auch möglich, jedes Zeichen mit seiner Zeichennummer anzusprechen. Jedes Symbol, oder auf englisch *character*, ist durch eine Zeichennummer eindeutig definiert. Für die Zeichennummern von 32 bis 127 gilt dabei der ASCII-Code.

Die Zeichen mit Nummern von 32 bis 47 enthalten im wesentlichen Satz- und Sonderzeichen, wie "$" ";" "," etc. Mit den Nummern von 48 bis 57 erreichen wir die Bildschirmdarstellungen für die verschiedenen Ziffern, danach folgen wieder einige Sonderzeichen, bevor ab Code 65 die Buchstaben des Alphabets und zwar als Großbuchstaben ausgegeben werden. Jeweils um 32 höher finden wir die zugehörigen Kleinbuchstaben. Dieser Datenrahmen des ASCII-Codes wurde beim CPC nun noch um 128 Grafiksymbole mit den Codenummern von 128 bis 255 und eine Reihe von Steuercodes, auf die wir in Kapitel 2.4 noch zurückkomen werden, erweitert.

Wenn wir von einem Zeichen die Zeichennummer kennen, so können wir dieses mit der CHR$-Funktion ausgeben. So liefert uns zum Beispiel

```
PRINT CHR$(65)
```

wiederum ein großes A, da 65 ja der Zeichencode für A ist. Mit Hilfe der CHR$-Funktion können wir uns nun einmal den gesamten Zeichensatz anschauen. Dies erreichen wir mit einer relativ einfachen Schleife.

```
FOR i=32 TO 255:PRINT i,CHR$(i):NEXT i
```

Diese Schleife gibt uns den gesamten Zeichenvorrat, wie er beim Einschalten definiert ist, aus. Wir erhalten für jeden Zeichencode das zugehörige .Zeichen auf dem Bildschirm. Diese Ausgabe ist allerdings keineswegs zwangsläufig, da wir jedes einzelne Zeichen einzeln umdefinieren können. Wie wir dies beim Schneider erreichen können, davon soll jetzt die Rede sein.

### 2.1.1 Die Zeichenmatrix

Zunächst ein paar Hintergrundinformationen. Jedes Zeichen beim Schneider, egal ob es sich um ein alphanumerisches Zeichen, also um Buchstaben oder Zahlen, um ein Satz- oder Sonderzeichen oder ein Grafiksymbol handelt, ist in einer Matrix aus acht mal acht Bildpunkten definiert.

In diesem Raster von acht Spalten und acht Zeilen wird nun ein Zeichen dadurch festgelegt, daß man für jeden einzelnen Bildpunkt, die sogenannten Pixels, angibt, ob dieser auf hell getastet werden soll (mit der Vordergrundfarbe dargestellt) oder ob für ihn eine Dunkeltastung (Darstellung mit der Hintergrundfarbe) erfolgen soll.

Wenn wir nun den Hellzustand mit 1 oder logisch "wahr" bezeichnen und die Dunkeltastung, beziehungsweise die Darstellung mit der Hintergrundfarbe, als 0 oder "unwahr", so haben wir im Endeffekt für die Abspeicherung und Definition eines jeden einzelnen Zeichens 64 Ja/Nein-Entscheidungen zu treffen. Wir müssen nämlich zum Beispiel beginnend mit dem Pixel in der linken oberen Ecke für alle 64 Bildpunkte festlegen, ob diese gesetzt oder rückgesetzt sein sollen.

Einer Ja/Nein-Entscheidung entspricht die Informationseinheit ein Bit. Im Speicher unseres Computers sind jeweils acht Bits zu einem Byte zusammengefaßt, das damit die Farbinformation für eine Zeile unseres Zeichens speichern kann. In acht Bytes oder acht aufeinanderfolgenden Adressen unseres Speichers können wir dann also einen ganzen Character ablegen.

Für die Ablage aller 256 Zeichen, die der CPC auf dem Bildschirm darstellen kann, benötigen wir also genau 2048 Bytes (256 mal 8 Bytes). Diese Punktdefinitionen sind im Lesespeicher unseres Computers, dem ROM abgelegt. Wenn wir nun zum Beispiel `PRINT "A"` eingeben, so spielt sich intern folgendes ab:

Zunächst einmal müssen wir im ROM die entsprechende Stelle finden an der unsere Zeichen abgelegt sind. Dies geht relativ einfach mit Hilfe der Zeichennummer, in unserem Falle also einer 65. Der für die Zeichenabspeicherung im ROM reservierte Bereich beginnt mit der Hexadezimaladresse 3800.

Das Hexadezimalsystem ist ein Zahlensystem, welches als Basis die Zahl 16 benutzt. Im Gegensatz zu unserem gewohnten Zehnersystem, das ja auf der 10 aufbaut. Dazu werden neben den Ziffern 0-9 noch die Buchstaben A-F benutzt, um die 16 benötigten Ziffern darstellen zu können. Wenn

wir also den Hexwert 3800 in das Zehnersystem umrechnen wollen, so müssen wir wie folgt rechnen:

$$3*16^3+8*16^2+0*16^1+0*16^0$$

was dezimal den etwas krummen Wert von 14336 ergibt. An dieser Adresse ist die oberste Punktreihe des Zeichens mit der Nummer 0 abgelegt. Eine Speicherstelle höher, also in hexadezimal 3801, liegt die zweite Reihe und so weiter. In Adresse 3808 hex ist die oberste Punktreihe des Zeichens mit der Nummer 1 gespeichert und so geht es immer weiter im Acht-Byte-Rhythmus.

Der CPC kann also nun die oberste Reihe des Zeichens Nummer 65 relativ einfach bestimmen, indem er zum Anfang dieses Speicherbereiches, der auch als Charactergenerator oder Character-ROM bezeichnet wird, einfach die Zeichennummer multipliziert mit acht addiert. An dieser Stelle (hex 3A08 oder dezimal 14856) ist die oberste Punktreihe unseres "A" abgespeichert.

Der CPC liest diese aus dem ROM ein, zerlegt das Byte in die einzelnen Bildpunkte und setzt diese dann im Grafikspeicher an die Stelle, wo die Ausgabe des "A" erfolgen soll. Dieser Prozeß wiederholt sich für die folgenden sieben Reihen, womit dann schließlich die Zeichendarstellung abgeschlossen ist.

Auf die Frage, wie die Ansprache der einzelnen Bildpunkte dabei erfolgt und wie die verschiedenen Farben gesetzt werden, kommen wir später noch im zweiten Teil dieses Buches bei der Maschinengrafik zurück. Die unterschiedlichen farblichen Möglichkeiten und auch die verschieden hohe Auflösung beziehungsweise Anzahl der darstellbaren Zeichen (80, 40 und 20 Zeichen) ergibt sich aus folgendem Prinzip:

Die normale Punktmatrix, wie wir sie gerade besprochen haben, wird nur im Mode 2 dargestellt. Hier existieren zwei Farben, die Vorder- und die Hintergrundfarbe. Ein gesetztes Pixel wird demnach in der Vordergrundfarbe dargestellt und umgekehrt. In den Modes mit einer größeren Anzahl darstellbarer Farben (MODE 0 und MODE 1) findet nun eine Verdoppelung beziehungsweise Vervierfachung der Pixels in der Horizontalen statt. Das heißt, statt einem Bildpunkt werden zwei oder vier nebeneinanderliegende Bildpunkte mit dem Wert eines Matrixpunktes belegt. Durch dieses gleichgeschaltete Setzen und Löschen von Bildpunkten wird im Endeffekt weniger Speicherplatz benötigt, so daß zusätzlicher Raum für die Ablage der Farbinformationen zur Verfügung steht.

So weit das Grundprinzip. In diesem Zusammenhang soll uns nur interessieren, daß in allen drei Modes immer dieselbe Zeichenmatrix benutzt wird. Sie wird nur in Abhängigkeit von der Darstellungsart gegebenenfalls in die Breite gespreitzt. Wenn wir also nun ein Zeichen umdefinieren, so gilt diese neue Festlegung für die Zeichendarstellung in allen drei verschiedenen Modes.

### 2.1.2 Neue Zeichen selbstgemacht

Wie definieren wir nun eigentlich neue Symbole? Die Festlegung eines neuen Characters ist beim CPC auf zwei Arten möglich. In beiden Fällen müssen dazu zwei Schritte erfolgen:

1. Reservierung von Speicherplatz für die neuen Symbole
2. Definition oder Umdefinition von Zeichen

Beschäftigen wir uns zunächst einmal mit dem ersten Schritt. Um einen geschützten Speicherplatz für die Ablage der neuzudefinierenden Zeichen zu erhalten, kann man beim CPC das Kommando SYMBOL AFTER benutzen. Dieser Befehl legt fest, ab welchem Zeichen oder besser ab welcher Zeichennummer nach oben eine Umdefinition möglich sein soll. Geben wir zum Beispiel ein `SYMBOL AFTER 200`, so sind die Characters mit den Nummern von 200 bis 255 nun frei definierbar.

Diese Prozedur ist deshalb nötig, weil eine Änderung der Speicherstellen im ROM nicht möglich ist. Das ROM ist ein reiner Lesespeicher. Wir müssen daher zuerst den Inhalt des ROM in einen Bereich des Schreib-, Lesespeichers des RAM kopieren, bevor wir darin redigieren können. Dies geschieht mit SYMBOL AFTER. Beginnend mit der Obergrenze des Benutzerspeichers reserviert dieser Befehl für jedes neu zu definierende Zeichen acht Bytes und schiebt die Obergrenze des Benutzerspeichers entsprechend nach unten. Sie können dies einmal nachvollziehen, wenn Sie die nachfolgenden Befehle eingeben.

```
SYMBOL AFTER 256
PRINT HIMEM
```

Die Funktion HIMEM gibt uns die aktuelle Obergrenze des Benutzerspeichers an. Ohne angeschlossene Floppy erhalten Sie jetzt den Wert 44031. Mit Floppy oder bei Verwendung des CPC 664 liegt der Wert ungefähr 1300 Bytes niedriger. Geben Sie nun einmal ein:

```
SYMBOL AFTER 255
```

und wiederholen Sie die HIMEM-Abfrage. Sie haben jetzt ein Zeichen als umdefinierbar angegeben und dafür wurden acht Bytes Speicherplatz benötigt. Entsprechend ist die Speichergrenze um acht Bytes nach unten gerutscht.

Die beim Einschalten vom Rechner angegebene Speicherobergrenze von 43903 ohne Floppy erhalten Sie, wenn Sie `SYMBOL AFTER 240` eingeben. Damit sind die obersten 16 Zeichen umdefinierbar. Wenn Sie also das Symbol mit der Nummer 250 ausgeben, so benutzt der Computer die im RAM abgelegte Zeichenmatrix. Wird dagegen ein Zeichen mit einer niedrigeren Nummer ausgedruckt, so wird weiterhin auf die im ROM abgelegten Ursprungswerte zurückgegriffen.

Bei der Verschiebung der Speicherobergrenze mit SYMBOL AFTER ist allerdings noch eines zu beachten. Es kann nämlich zu einer Kollision mit einer Verschiebung durch MEMORY kommen.

MEMORY dient in erster Linie dazu, einen für Maschinenprogramme reservierten freien Speicherbereich zu schaffen. Dies geschieht ebenso wie bei der Reservierung neuer Zeichen durch das Herabsetzen der Speicherobergrenze. Der Platz zwischen alter und neuer Speicherobergrenze wird dann bei der Ausführung von BASIC-Befehlen nicht mehr angetastet und ist somit ein sicherer Ablageplatz für Maschinenroutinen.

Um nun zu vermeiden, daß durch eine nachträgliche Reservierung von Speicherplatz für weitere umzudefinierende Zeichen ein darunterliegendes Maschinenprogramm zerstört wird, ist im Betriebssystem des Computers eine Sicherung eingebaut. Wird die Speicherobergrenze mit MEMORY verschoben, so ist danach keine Änderung mit SYMBOL AFTER mehr möglich. Wenn Sie also zum Beispiel nach dem Einschalten

```
SYMBOL AFTER 200
MEMORY 40000
```

eingeben und sich entschließen, daß Sie nun doch noch einige weiter unten liegende Zeichen im Bereich zwischen 175 und 200 umdefinieren wollen, so ist dies mit

```
SYMBOL AFTER 175
```

nun nicht mehr möglich. Der CPC verweigert in diesem Fall die Annahme und gibt einen `"Improper argument"`-Fehler aus. Die einzige Möglichkeit, hier doch noch zum Ziel zu kommen besteht darin, daß man mit einem nochmaligen MEMORY die Speicherobergrenze wieder auf den Ursprungswert zurücksetzt, das heißt auf die Adresse, die HIMEM nach

dem ersten SYMBOL AFTER enthielt. In diesem Fall "vergißt" der Schneider die Platzreservierung mit MEMORY und ist dann wieder bereit, Platz für neue Zeichen freizumachen. Als Grundregel sollten Sie sich dennoch merken:

**Die Reservierung mit SYMBOL AFTER muß vor einer weiteren Verschiebung mit MEMORY erfolgen.**

### 2.1.2.1 Die Neudefinition mit SYMBOL

Eine Möglichkeit, neue Zeichen zu definieren besteht in der Verwendung des BASIC-Kommandos SYMBOL. Dieses benötigt, nachgestellt und durch Komma getrennt, 9 weitere Parameter. Als erstes ist dies die Nummer des neu zu definierenden Zeichens, danach folgen die acht Werte für die einzelnen Reihen unseres Zeichens.

Zur Definition können wir dabei beim CPC die Binäreingabe benutzen. Der Schneider kann nämlich nicht nur dezimal eingegebene Zahlenwerte einlesen, sondern er "verdaut" auch eine binäre Eingabe. Während wir Dezimalzahlen ohne vorangestelltes Typenkürzel direkt schreiben können, müssen wir vor eine Binärzahl das Kürzel "&x" setzen um dem Computer verständlich zu machen, daß es sich bei dem folgenden Wert um eine Binärzahl handelt. Mit Hilfe von Binärzahlen können wir nun die einzelnen Bits, die den Bildpunkten in unserer Matrix entsprechen direkt eingeben. Wir wollen dies einmal an einem Beispiel durchexerzieren.

#### 2.1.2.1.1 Definition des deutschen Zeichensatzes

Wie Ihnen sicherlich bei den ersten tastenden Versuchen auf dem Computer aufgefallen ist, verfügt der Schneider über eine englische QWERTY-Tastatur, das heißt y und z sind positionsmäßig vertauscht und im übrigen stellt die Tastatur keinerlei Umlaute zur Verfügung. Wir wollen dies nun ändern. Da die Umlaute im Zeichensatz des CPC nicht standardmäßig definiert sind, müssen wir dazu den Zeichensatz ändern. Wir wollen dabei folgende Belegung wählen.

| Zeichennummer dezimal | Zeichennummer hexadezimal | Buchstabe |
|---|---|---|
| 91 | 5B | Ä |
| 92 | 5C | Ö |
| 93 | 5D | Ü |
| 123 | 7B | ä |
| 124 | 7C | ö |
| 125 | 7D | ü |
| 126 | 7E | ß |

Warum haben wir gerade diese Codes ausgesucht? Die Antwort ist relativ einfach. Die meisten handelsüblichen Drucker, die über einen deutschen Zeichensatz verfügen oder bei denen dieser einstellbar ist, drucken genau bei diesen Symbolen die deutschen Umlaute. Erhält also der Drucker zum Beispiel ein CHR$(91), so druckt er ein großes "Ä" aus. Wenn wir also nun unsere Bildschirmzeichen mit SYMBOL auf diese Codes legen, so ist es egal, ob wir danach ein "Ä" auf dem Bildschirm oder auf dem Drucker ausgeben. Auf jeden Fall erhalten wir die richtige Darstellung.

Bei dieser Definition sind dann übrigens die neuen Umlaute auch auf der Tastatur verfügbar. "Ä" und "Ü" finden sich auf den Tasten für die eckigen Klammern. Das "Ö" liegt zweigeteilt auf dem Schrägstrich und dem Erweiterungsstrich oberhalb des Klammeraffen ("@"). Das "ß" schließlich finden wir in der CTRL-Ebene wieder. Mit <CTRL+2> können wir es ausgeben.

Mit einer deutschen DIN-Tastatur hat diese Belegung wenig gemeinsam. Für eine Überprüfung, ob die Characters richtig festgelegt wurden, reicht jedoch diese Art der Tastaturansprache. In Kapitel 2.3 werden wir bei der Beschäftigung mit der Tastatur dann auch untersuchen, wie wir eine deutsche Normtastatur definieren können.

Doch nun zurück zur eigentlichen Definition. Bei den nun folgenden Bemerkungen sollten Sie immer den Anhang des Handbuches und zwar dort das Kapitel III aufgeschlagen haben. In diesem Teil des Bedienerhandbuches finden Sie den kompletten Zeichensatz des CPC, aufgelöst in die einzelnen Punktmatrizen. Ein gesetzter Bildpunkt ist dabei dunkel markiert, ein rückgesetzter hell. Um nun zu einer binären Darstellung der Zeichenmatrix zu gelangen, ersetzen wir alle hell getasteten Punkte durch 1. Die restlichen Pixel werden genullt. In binärer Schreibweise würde zum Beispiel das große "A" (Zeichen Nummer 65) dann wie folgt aussehen.

```
00011000
00111100
01100110
01100110
01111110
01100110
01100110
00000000
```

Die Konturen des "A" können Sie gut erkennen, wenn Sie sich auf die Bits konzentrieren, die auf 1 gesetzt sind. Wie erhalten wir nun aus unserem "A" ein "Ä"? Dies ist relativ einfach zu erreichen. Wir müssen lediglich in der obersten Reihe die linken, beziehungsweise rechten beiden Bildpunkte setzen und dann die einzelnen Zeilenwerte an den Computer mit SYMBOL zurückgeben.

Hierzu gleich eine kleine Randbemerkung: Sie sollten, wann immer möglich, zumindest die senkrechten Linien eines Buchstabens mit zwei Punkten nebeneinander setzen. Speziell bei der Verwendung eines Farbmonitors kann es ansonsten passieren, daß Sie den entsprechenden Bildpunkt ihres Zeichens durch Unregelmäßigkeiten im technischen Aufbau ihrer Bildröhre (Verschiebung der Schlitzmaske) nicht mehr wiederfinden.

Vor der eigentlichen Zeichendefinition müssen wir jetzt das unterste umzudefinierende Zeichen angeben. Dazu definieren wir

```
SYMBOL AFTER 90
```

Damit sind die Zeichen mit Codes größer als 90 nun zur Umdefinition freigegeben. Als nächstes weisen wir nun dem Zeichen Nummer 92 mit SYMBOL die neuen Werte zu.

```
SYMBOL92,&x11011011,&x00111100,&x01100110,
&x01100110,&x01111110,&x01100110,&x01100110,
&x00000000.
```

Dazu geben wir einfach das Punktraster für die einzelnen Zeilen in binärer Form ein. Wenn Sie die einzelnen Angaben dabei mit der weiter oben angegebenen Matrix für das "A" vergleichen, sehen Sie, daß nur die oberste Zeile verändert wurde. Die unteren sieben Zeilen stellen eine Kopie der Werte dar, die wir bereits bei der Definition des "A" kennengelernt haben.

Nach demselben Prinzip kann man nun natürlich auch die anderen Umlaute definieren. Ein kurzes Programm, welches den deutschen Zeichensatz implementiert, könnte dann zum Beispiel so aussehen.

```
100   REM ************************
110   REM * DEUTSCHER ZEICHENSATZ *
120   REM ************************
130   A=&5B : B=&7B : O=&5C : P=&7C : U=&5D : V=&7D : S=&7E
140   UEG1=&X1100110 : UEG2=&XO : UEG3=&1100110 : UEG4=UEG3 : UEG5=UEG3 :
      UEG6=UEG3 : UEG7=&X111100
150   UEK1=&X1100110 : UEK2=0 : UEK3=UEK1 : UEK4=UEK1 : UEK5=UEK1 : UEK6=UEK1
      : UEK7=&X111110 : UEK8=0
160   AEG1=&X1101011:AEG2=&x00111100:AEG3=&X01100110:AEG4=&X110110:
      AEG5=&X1111110 : AEG6=&X1100110 : AEG7=AEG6 : AEG8=0
170   AEK1=&X11000110 :AEK2=0:AEK3=&X1111000:AEK4=&1100: AEK5=&X1111100 :
      AEK6=&X11001100 : AEK7=&X111011 : AEK8=0
180   OEK1=AEK1 : OEK2=0 : OEK3=&X1111000 : OEK4=&X11001100 : OEK5=OEK4 :
      OEK6=OEK4 : OEK7=OEK3 : OEK8=0
190   OEG1=OEK1 : OEG2=&X111000 : OEG3=&X1101100 : OEG4=&X11000110 : OEG5=OEG4
      : OEG6=OEG3 : OEG7=OEG2 : OEG8=0
200   SZ1=&X111100 : SZ2=&X1100110 : SZ3=SZ2 : SZ4=&X1111100 : SZ5=SZ2 :
      SZ6=SZ2 : SZ7=&X1100100 : SZ8=&X1100000
210   SA=MIN (A,B,O,P,U,V,S)
220   SYMBOL AFTER SA
230   SYMBOL A,AEG1,AEG2,AEG3,AEG4,AEG5,AEG5,AEG7,AEG8
240   SYMBOL B,AEK1,AEK2,AEK3,AEK4,AEK5,AEK6,AEK7,AEK8
250   SYMBOL O,OEG1,OEG2,OEG3,OEG4,OEG5,OEG6,OEG7,OEG8
260   SYMBOL P,OEK1,OEK2,OEK3,OEK4,OEK5,OEK6,OEK7,OEK8
270   SYMBOL U,UEG1,UEG2,UEG3,UEG4,UEG5,UEG6,UEG7,UEG8
280   SYMBOL V,UEK1,UEK2,UEK3,UEK4,UEK5,UEK6,UEK7,UEK8
290   SYMBOL S,SZ1,SZ2,SZ3,SZ4,SZ5,SZ6,SZ7,SZ8
```

**Programmbeschreibung:**

In den ersten Zeilen werden zunächst verschiedene Variablen auf die Werte für die einzelnen Reihen gesetzt, bevor dann ab Zeile 230 die eigentliche Definition mit SYMBOL erfolgt. Die Variable OEK1 steht dabei zum Beispiel für die Reihe 1 eines kleinen "ö". Die angegebenen Werte stellen dabei natürlich nur mögliche Definitionen dar. Wenn Ihnen also das eine oder andere Symbol nicht ganz gefällt, so sollten Sie sich nicht scheuen, Ihren eigenen deutschen Zeichensatz zu entwerfen.

Dieses Verfahren bietet sich immer dann an, wenn man mehrere Umdefinitionen durchführen will. Da sich die Reihenangaben für mehrere Zeilen eines Zeichens oder im Vergleich mit anderen Symbolen oftmals entsprechen, ist es kein Problem eine neue Variable auf den Wert einer alten, gerade schon definierten Zeile zu setzen. Durch dieses Verfahren spart man sich einige Zeit beim Eintippen, und Fehler werden vermieden.

### 2.1.2.1.2 Ein Programm zur Zeichenumdefinition

Nun ist das Ermitteln der verschiedenen Zeilenwerte eine relativ einfache, aber gleichzeitig fehleranfällige und auf die Dauer entnervende Tätigkeit.

Sie eignet sich also geradezu ideal für den Computereinsatz. Dazu müssen wir allerdings ein kleines Programm entwickeln. Zunächst ein paar Worte zur Zielsetzung.

Die fertige Routine soll es uns ermöglichen in einer auf dem Bildschirm abgebildeten Zeichenmatrix mit dem Joystick oder den Cursortasten umherzufahren und dabei beliebige Punkte zu setzen oder zu löschen. Als Ergebnis soll der Computer dann immer das zugehörige SYMBOL-Kommando ausgeben.

**Programmbeschreibung:**

Das Programm Zeichendefinition arbeitet mit zwei Bildschirmfenstern, den WINDOWS#1 und #2. Sie unterteilen den Bildschirm senkrecht. Im linken Teil des Schirms wird permanent das zu definierende Zeichen ausgegeben und zwar in Originalgröße. Im rechten Teil befindet sich eine Matrix aus 8*8 Nullen. Dieser Bildschirmaufbau wird mit den Zeilen bis Zeilennummer 130 erzeugt. Zunächst werden die Bildschirmfarben und die Größe der Windows im Mode 1 festgelegt, bevor dann das Null-Feld mit einer doppelten Schleife (Zeile 100-120) definiert wird.

Es folgt ein Teil zur Joystickabfrage mit Hilfe der Funktion JOY. Diese Funktion enthält in binärer Form gespeichert den Zustand des Joysticks, seine momentane Bewegungsrichtung und eine Aussage darüber, ob der Feuerknopf betätigt wurde. Alle diese Informationen werden in einer einzigen Zahl gespeichert. Jedes einzelne Bit enthält dabei eine Bewegungsrichtung (vergleiche Kapitel 8/Seite 22 des Bedienerhandbuches).

Hier tritt aber nun ein Problem auf. Die JOY-Funktion gibt uns nämlich als Ergebnis die Summe aller Bewegungen zurück. Wenn wir beispielsweise den Joystick gleichzeitig vorwärts und nach rechts bewegen, so gibt uns der CPC als Summe dieser Bewegungen, den Wert 9 (8+1), zurück. Wenn wir nun abfragen, ob der Joystick nach rechts gedrückt ist (JOY=8), so erhalten wir fälschlich eine negative Auskunft.

Bei kombinierten Bewegungen kommen wir also mit dieser einfachen Abfrageart nicht weiter. Da wir aber die Einzelbewegungen für unsere nachfolgenden Änderungen benötigen, müssen wir diesen Gesamtwert wieder in seine Einzelkomponenten zerlegen. Dies erreichen wir durch die Anwendung der AND-Funktion. Wenn wir auf das Gesamtergebnis die AND-Funktion mit dem Wert 8 anwenden, so erhalten wir als Rückgabe 8, falls der Joystick auch nach rechts gedrückt wurde, ansonsten 0.

Mit diesem Trick ist die Zerlegung der Cursorbewegung nun ein Kinderspiel. Die Variablen x und y nehmen die aktuellen Werte für Cursorreihe und -spalte auf und werden unter Beachtung der Randbedingungen (Zeile 210, 220) durch Joystickbewegungen gesetzt. Ab Zeile 260 erfolgt dann die Stellenprüfung. Hier testet der CPC, ob sich an der aktuellen Position des Cursors ein Bildpunkt beziehungsweise die zugehörige "1" oder "0" befindet. In Abhängigkeit davon (f=1 oder f=0) springt der Schneider dann in zwei verschiedene Routinen, die das Cursorblinken bewirken (Z. 510-540 oder Z. 590-620).

Jetzt gibt es noch zwei zusätzliche Bedienmöglichkeiten, auf die der Computer reagieren soll. Durch Druck auf den Feuerknopf soll die Umkehrung des Wertes eines Bildpunktes erfolgen, falls der Cursor auf diesem steht. Das Fire-Signal wird dazu in Zeile 290 abgeprüft. Wurde der Knopf gedrückt, so geht es weiter nach Zeile 340. Befindet sich der Cursor nicht auf einer Pixelposition (f=0), so geht es gleich wieder zurück in die Abfrageschleife. Ansonsten wird der Testcharacter Nummer 255 mit SYMBOL auf die neuen Werte definiert und dann ausgegeben.

Die andere Funktion erreicht man durch Drücken von "^". Hier wird die zur Definition des aktuellen Zeichens notwendige SYMBOL-Folge ausgegeben. Mit nochmaligem Druck auf "^" kommt man wieder in die Joystick-Abfrageschleife zurück (Zeile 470).

**Listing 1:** *Das Programm Zeichendefinition*

```
10 REM ************************
20 REM ** Zeichendefinition **
30 REM ************************
40 INK 0,0:INK 1,15:INK 2,2:INK 3,21
50 MODE 1:BORDER 1
60 WINDOW#1,1,3,1,25:WINDOW#2,4,40,1,25
70 PAPER#1,0:PEN#1,3:PAPER#2,1:PEN#2,2
80 CLS#1:CLS#2
90 DIM m(8,8),w(8):x=2:y=2
100 FOR i= 1 TO 8:FOR j=1 TO 8
110 LOCATE#2,2+4*j,2+2*i:PRINT#2,m(i,j)
120 NEXT j,i
130 SYMBOL AFTER 240
140 REM ***********************
150 REM ** Joystickabfrage **
160 REM ***********************
170 IF (JOY(0) AND 2)=2 THEN y=y+1
180 IF (JOY(0) AND 1)=1 THEN y=y-1
190 IF (JOY(0) AND 4)=4 THEN x=x-1:IF x<2 THEN x=37:y=y-1
200 IF (JOY(0) AND 8)=8 THEN x=x+1:IF x>36 THEN x=2:y=y+1
210 y=MAX(y,2):y=MIN(y,22)
220 IF x<7 OR x>35 OR y<4 OR y>19 THEN f=0:GOTO 280
230 REM ***********************
240 REM ** Stellenpruefung **
250 REM ***********************
260 f=0
270 IF INT((x-3)/4)=(x-3)/4 AND INT((y-2)/2)=(y-2)/2 THEN f=1
280 IF f=1 THEN GOSUB 590 ELSE GOSUB 510
290 IF (JOY(0) AND 16)=16 THEN 340
300 IF INKEY(24)=0 THEN 440 ELSE 170
310 REM ********************
320 REM ** Pixel aendern **
330 REM ********************
340 IF f=0 THEN 170
350 m((x-3)/4,(y-2)/2)=-(m((x-3)/4,(y-2)/2)-1)
360 FOR i= 1 TO 8
370 w(i)=0
380 FOR j=1 TO 8
390 w(i)=2^(8-j)*m(j,i)+w(i)
400 NEXT j,i
410 SYMBOL 255,w(1),w(2),w(3),w(4),w(5),w(6),w(7),w(8)
420 LOCATE#1,2,10:PRINT#1,CHR$(255)
430 GOTO 170
440 LOCATE#2,1,24:z$="SYMBOL 255"
450 FOR i= 1 TO 8:z$=z$+","+MID$(STR$(w(i)),2,5):NEXT i
460 PRINT#2,z$
470 IF INKEY$<>"^" THEN 470 ELSE 170
480 REM ************************
490 REM ** Cursor ohne Zahl **
500 REM ************************
510 PEN#2,3:LOCATE#2,x,y:PRINT#2, CHR$(143)
520 FOR i=1 TO 100: NEXT i
530 PEN#2,2:LOCATE#2,x,y:PRINT#2, CHR$(32)
540 RETURN
550 REM ********************
560 REM ** Cursor mit Zahl **
570 REM ********************
580 REM
590 LOCATE#2,x-1,y:PEN#2,3:PRINT#2,m((x-3)/4,(y-2)/2)
600 FOR i=1 TO 100:NEXT i
610 LOCATE#2,x-1,y:PEN#2,2:PRINT#2,m((x-3)/4,(y-2)/2)
620 RETURN
```

### 2.1.2.2 Zeichenumdefinition mittels Direktzugriff und POKE

Die zweite Möglichkeit ein Zeichen umzudefinieren, ist der Direktzugriff auf den Symbolspeicher, das heißt jene Adressen im RAM, in denen die veränderte Kopie des Zeichensatzes abgelegt ist.

Hierbei müssen wir allerdings ein wenig rechnen. Wie wir wissen legt der CPC ab der Speicherobergrenze, das heißt ab dem Wert, den wir für HIMEM nach SYMBOL AFTER 256 erhalten, die einzelnen Zeichenmatrizen im Acht-Byte-Rhythmus nach unten ab. Wenn wir nun also definieren `SYMBOL AFTER 90`, so verschiebt sich HIMEM um 1328 Bytes ((256-90)x8) nach unten. Beim CPC 464 ohne Floppy wäre dies der Wert 42575, beim CPC 664, der ja die Floppy integriert hat, ergibt sich als Ausgabe 41419.

Ein Byte höher als dieser Wert liegt nun die oberste Punktreihe des untersten neu zu definierenden Zeichens, in unserem Fall also Zeichen Nummer 90, was einem "Z" entsprechen würde. Neun Bytes höher treffen wir auf die oberste Punktreihe des Zeichens Nummer 91 und damit auf das erste von uns umzudefinierende Symbol. In diese Adresse müssen wir nun den Wert für die oberste Punktreihe ablegen. Dies geht relativ einfach mit

```
POKE HIMEM+9,&X11011011
```

Damit haben wir nun die oberste Punktreihe gesetzt. Wir könnten nach demselben Schema nun auch auf die anderen Zeilen zurückgreifen und beispielsweise die zweite Punktreihe mit `POKE HIMEM+10` und so weiter setzen.

Hier können wir jedoch auch auf einem etwas kürzeren und schnelleren Weg zum Ziel kommen. Die unteren Punktreihen entsprechen ja vollständig den Werten, die wir bereits für das "A" kennen. Wir brauchen diese also nur zu kopieren. Dazu müssen wir auch die Zeichen ab 65 mit

```
SYMBOL AFTER 65
```

zuschalten, um auch diese Zeichen verfügbar zu haben. Auf die im ROM enthaltenen Matrizen können wir nämlich nur mit einer Speicherumschaltung, die aber in Maschinensprache erfolgen müßte, zurückgreifen. Da wir aber für den Kopiervorgang zuerst einmal eine Quellmatrix benötigen, ist dieses SYMBOL AFTER notwendig. Die Variable HIMEM rutscht damit gleichzeitig auch nach unten. Die Differenz von Symbol Nummer 91 zu Zeichen Nummer 65 beträgt 26 Zeichen, was, mit 8 Bytes

je Symbol multipliziert, den absoluten Abstand der beiden Characters angibt. Zur Kopie benötigen wir nun eine kurze Schleife.

```
FOR i=HIMEM+2 TO HIMEM+8:POKE i+208,PEEK(I):NEXT i
```

Diese Schleife verlagert nur die Bytes 2 bis 8, das heißt die unteren sieben Punktreihen unseres "A" um 208 Bytes nach oben. Durch die Verschiebung um 208 trifft die Kopie dann genau in die Speicherstellen, die den unteren Teil unseres "Ä" enthalten. Wenn wir nun noch

```
POKE HIMEM+209,&x11011011
```

eingeben, so ist unsere Änderung komplett. Wie Sie aber sicher schon gemerkt haben, hat diese Sache einen Haken. Wir mußten nämlich die Speichergrenze mit SYMBOL AFTER eigentlich unnötig nach unten verschieben und dies kostet natürlich Speicherplatz. Leider ist es nun aber nicht möglich, nachträglich die HIMEM mit SYMBOL AFTER wieder herabzusetzen. Bei SYMBOL AFTER wird nämlich, wie wir ja schon gesehen haben, das ROM in den variablen Speicher kopiert. Diese Kopie würde daher sofort unsere mühsam kopierten Symbole wieder überschreiben, womit diese Operation wenig sinnvoll ist.

Nun könnten wir natürlich mit Hilfe einer Veränderung von Zeigern versuchen, nur die Speicheraufteilung zu ändern. Dieses Verfahren gestaltet sich beim CPC allerdings relativ kompliziert, so daß es mehr Aufwand bereiten würde, als eine direkte Eingabe mit POKE. Es gibt jedoch einen Weg, der diese ganzen Probleme umgeht. Wir können nämlich einen Zeichensatz bei einer Speicherverteilung, also einem Wert für SYMBOL AFTER sichern und ihn dann bei einer anderen wieder einladen. Dazu gleich mehr.

Diese Methode, das heißt die Zeichenänderung durch Kopie, ist aber meist nur sinnvoll, wenn man wirklich größere Änderungen im Zeichensatz bei ähnlichen Symbolen vorzunehmen hat.

In den meisten Fällen wird man jedoch auf die Definition mit SYMBOL oder einen einfachen POKE zurückgreifen. Man kann zum Beispiel aus einem DATA-Feld einen ganz neuen Zeichensatz einladen und dies geht erheblich schneller und benötigt auch weniger Speicherplatz für die Ablage. Unser eben beschriebener deutscher Zeichensatz könnte beispielsweise auch aus einem DATA-Feld eingelesen werden. Dies könnte dann wie folgt aussehen:

```
10 REM ****************************
20 REM * dt.Zeichensatz aus DATA *
30 REM ****************************
```

```
40 DATA 219,60,102,102,126,102,102,0
50 DATA 214,108,198,198,198,108,56,0
60 DATA 102,0,102,102,102,102,60,0
70 DATA 108,0,120,12,124,204,118,0
80 DATA 102,0,60,102,102,102,60,0
90 DATA 0,102,0,102,102,102,62,0
100 DATA 120,204,204,248,204,204,248,128
110 SYMBOL AFTER 91
120 FOR I=HIMEM+1 TO HIMEM+24:READ A:POKE I,A:NEXT I
130 FOR I=HIMEM+257 TO HIMEM+288:READ A:POKE I,A:NEXT I
```

**Programmbeschreibung:**

Diesmal sind die Reihenwerte schon übersetzt in dezimaler Schreibweise angegeben. Jede DATA-Zeile enthält dabei genau ein Symbol. Die einzelnen Zeilenwerte werden dann mit Hilfe von zwei FOR-TO-Schleifen in den Speicher eingelesen.

### 2.1.2.3 Das Laden und Speichern von Zeichensätzen.

Wie wir gerade gesehen haben, ist der Entwurf eines neuen Zeichensatzes eine ziemlich aufwendige Arbeit, unabhängig davon, ob man nun mit POKE oder SYMBOL agiert. Es wäre daher sehr schön, wenn wir uns einige Standardzeichensätze entwerfen könnten.

Diese würden dann beim Einschalten des Computers einmal eingeladen werden und stünden dann für weitere Programmentwicklungen zur Verfügung. Mit dem CPC können wir dies einfach und problemlos realisieren und zwar mit Hilfe der Maschinen-SAVE-Routine. Mit dem BASIC-Befehl

```
SAVE"<NAME>",B,<Anfangsadresse>,<Anzahl
abzuspeichernder Bytes>
```

ist es möglich, einen Teilbereich des Speichers, der sich oberhalb der Speicherobergrenze befindet, das heißt oberhalb der Adresse HIMEM, auf Band oder Floppy zu sichern. In der Umkehrrichtung reicht dann ein einfacher LOAD-Befehl.

Wenn Sie also den Zeichensatz mit Hilfe eines der in den letzten Unterkapiteln beschriebenen Programme umdefiniert haben, so können Sie ihn nun mit

```
SAVE "Zeichensatz",B,HIMEM+1,1320
```

sichern. Der Wert von 1328 ergibt sich dabei aus der Anzahl der zu speichernden Bytes. Wenn Sie also SYMBOL AFTER eingetippt haben, so sind 1320 Speicherstellen zu sichern ((256-91)*8 Bytes je Zeichen). Bei unserem ersten Umdefinitionsprogramm, wo wir ab Character Nummer 90 geändert hatten, wären es dagegen 8 Bytes mehr.

Um sich mit dieser Technik vertraut zu machen, sollten Sie einmal einen beliebigen Zeichensatz definieren und ihn dann auf diese Art sichern. Dann setzen Sie den Computer mit

```
<CTRL> <SHIFT> <ESC>
```

wieder zurück. Die Umdefinition ist damit wieder gelöscht, wovon Sie sich durch PRINT CHR$(91) oder eine ähnliche Abfrage überzeugen können. Sie erhalten in diesem Fall wieder die Normaldefinition, eine eckige Klammer. Geben Sie nun ein:

```
SYMBOL AFTER 90
```

und dann

```
LOAD"Zeichensatz"
```

Der CPC lädt die Zeichen nun in den durch SYMBOL AFTER reservierten Adreßraum ein und mit nochmaligem `PRINT CHR$(91)` können wir uns davon überzeugen, daß der Zeichensatz wieder vorhanden ist.

Hier ist allerdings eine Anmerkung zu machen. Sollten Sie den Zeichensatz auf dem CPC 464 ohne Floppy geschrieben haben und ihn dann mit einer anderen Gerätekonfiguration, also angeschlossener Diskette wieder einlesen wollen, oder umgekehrt, so gibt es ein Problem.

Durch das Anstecken des Zusatzgerätes wird nämlich der gesamte Benutzerspeicher einschließlich der SYMBOL-Definitionen nach unten verschoben und zwar um ungefähr 1300 Bytes. Haben Sie also mit angeschlossener Floppy geschrieben und betreiben den Computer dann ohne Floppy, so liegt der Zeichensatz um jene 1300 Bytes zu niedrig, das heißt in einem Bereich, den das BASIC benutzt und wird dort sehr bald überschrieben.

Im anderen Fall wird er um dieselbe Zahl zu hoch eingelesen und überschreibt damit wichtige Steuerdaten der Floppy. Als Folge davon stürzt normalerweise der Rechner ab. Durch `<CTRL> <SHIFT> <ESC>` können Sie ihn natürlich jederzeit wieder zum "Leben"

zurückerwecken, aber der Zeichensatz ist damit natürlich immer noch nicht eingelesen.

Glücklicherweise können wir aber das LOAD-Kommando auch als Verschiebebefehl benutzen, indem wir nach dem Dateinamen, durch Komma getrennt, eine neue Ladeanfangsadresse für die abgespeicherten Daten bestimmen. Wenn Sie also statt dem einfachen LOAD ein

```
LOAD"Zeichensatz",HIMEM+1
```

ausführen, so funktioniert das Laden unabhängig von der gewählten Gerätekonfiguration und auch unabhängig davon, mit welcher Zusammenstellung Sie das Programm überhaupt geschrieben haben. Der Zeichensatz wird immer ab der Adresse HIMEM+1 geladen.

Mit einem ähnlichen Trick können wir auch etwas gegen den unnötigen Speicherbedarf bei der Kopiervariante unternehmen. Dazu müssen Sie sich nur den Wert notieren, den Sie für HIMEM erhalten, wenn Sie ein SYMBOL AFTER auf den niedrigsten umzudefinierenden Wert ausführen, zum Beispiel das Zeichen 130. Anschließend geben Sie mit SYMBOL AFTER 32 den gesamten Zeichensatz zum Lesen frei.

Nachdem Sie dann durch Kopieren oder durch einfaches POKE ihren eigenen Benutzerzeichensatz festgelegt haben, speichern Sie diesen wie gewohnt ab, nur daß Sie für die Anfangsadresse nicht mehr HIMEM+1, sondern die notierte Zahl vermehrt um 1 verwenden. Bei einem späteren LOAD wird dann nur noch der effektiv benötigte Symbolsatz (ab Zeichen 130) eingelesen. SYMBOL AFTER, in diesem Fall SYMBOL AFTER 130, müssen Sie natürlich trotzdem noch davor eingeben.

## 2.2 Licht und Schatten - zur Farbgebung beim CPC

Nachdem wir uns im letzten Kapitel relativ intensiv mit der Zeichendefinition und den Möglichkeiten neuer Symbole beschäftigt haben, wollen wir uns nun einige Gedanken zur Farbgebung, zu der Art, wie der CPC Farben definiert und zu den verschiedenen Anwendungsmöglichkeiten, die uns dadurch zur Verfügung stehen, machen.

Wie Sie sicher wissen, werden die verschiedenen beim Schneider darstellbaren Farben durch das INK-Kommando festgelegt. Der CPC arbeitet dabei mit einer indirekten Farbwahl. Wenn Sie also zum Beispiel

```
PEN 2
```

eingeben, so wird nicht mit der Farbe Nummer 2 geschrieben (mittelblau), sondern mit der beziehungsweise mit den Farbwerten, die in dem Farbregister mit der Nummer 2 gespeichert sind. Wir können dabei aus 27 verschiedenen Farben auswählen, die eine Grauwertskala, also eine Abstufung nach der von der Farbe ausgehenden Helligkeit, bilden. Farbe 0 entspricht dabei der vollen Dunkeltastung (schwarz). Mit INK 26 erreichen wir die größte Helligkeit (weiß).

Die Beachtung dieser Abstufung in der Grauwertskala ist dann besonders wichtig, wenn man ein Programm schreiben will, das sowohl mit einem grünen als auch mit einem Colour-Monitor laufen soll. Wählt man hier zu nah beieinanderliegende Farben, wie zum Beispiel rot(6) und blau(2), so erhält man zwar auf dem farbigen Bildschirm eine sehr angenehme Farbgebung. Auf dem einfarbigen Schirm dagegen liegen die zugehörigen Grauwerte sehr (zu) nahe beieinander. Ergebnis: Wir erhalten einen mangelhaften Bildkontrast. Die "Farbabstufungen" werden kaum wahrgenommen. Als Grundregel für die Farbdefinition bei Programmen, die auch auf einem grünen Schirm laufen sollen, kann man sich merken:

**Der Unterschied in den Grauwerten von zwei nebeneinander dargestellten Farben sollte wenigstens 6 Helligkeitsstufen betragen.**

Neben den Farbwerten von 0 bis 26 können wir auch noch Zahlen zwischen 27 bis 31 eingeben. Erst bei noch größeren Angaben streikt der CPC mittels `"Improper argument"`.

Diese Werte stellen aber nur eine Kopie der INKs dar, die wir für die unteren Zahlenwerte erhalten und bringen damit nichts Neues.

Um eine Farbe beim Schneider benutzen zu können, sind zwei Schritte notwendig:

1. Definition eines Farbregisters auf die gewünschte Farbe (INK).
2. Definition, daß dieses Farbregister nun für die Stiftfarbe (PEN) oder den Hintergrund (PAPER) benutzt werden soll.

Die Anzahl der bei dieser Festlegung benutzbaren Farbregister hängt davon ab, in welchem MODE man sich gerade befindet. Im MODE 2 (zwei Farben 80 Zeichen/Zeile) können zwei Farbregister (INK 0 und INK 1) definiert werden. Im MODE 1 (vier Farben 40 Zeichen/ Zeile) haben wir es mit vier Farbregistern zu tun (INK 0-INK 3). Im MODE 0 schließlich (16 Farben, 20 Zeichen/Zeile) können wir alle 16 INK-Register auch benutzen (INK 0-INK 15).

Die Definition eines Farbregisters erfolgt dabei mit

```
INK <Farbregisternummer>,<1.Farbe>,<2.Farbe>
```

Hier ist schon gleich auf eine Besonderheit des CPC zu verweisen. Der Computer stellt nämlich Farben grundsätzlich blinkend dar. Wenn Sie nur eine Farbe spezifizieren, so blinkt diese gegen sich selbst, was auf den Betrachter als "stehende" Farbe wirkt. Ansonsten blinken zwei Farben gegeneinander.

Da das Verfahren der indirekten Farbdefinition speziell für den Anfänger und bei der Arbeit im MODE 0 etwas schwierig ist, wurde der nebenstehende Farbplaner entwickelt. In den verschiedenen Spalten tragen Sie dabei, je nach verwendetem Bildschirmmodus die ausgewählten Farben (2, 4 oder 16) am besten mit Farbnummer und Namen ein. Sie haben dann immer eine gute Übersicht darüber, welche Farben Sie in einem Spiel zum Beispiel aktuell zur Verfügung haben. Diese können dann mit PEN und PAPER für die Zeichendarstellung benutzt werden. Für die Arbeit mit der Maschine sollten Sie sich davon ein paar Kopien machen.

Die Farbdarstellung, das heißt die Auswahl des Farbenbündels mit dem man arbeiten will, aus den 27 möglichen Werten, die Kombination verschiedener Farbwerte und dann speziell auch das gezielte Umschalten zwischen Farbtönen stellen eine der wichtigsten Aufgaben bei jeder Programmentwicklung, speziell aber im Spielebereich dar. Hier daher kurz ein paar Worte zu einigen interessanten Tricks und Effekten, die eigentlich erst durch die indirekte Farbdefinition realisierbar sind.

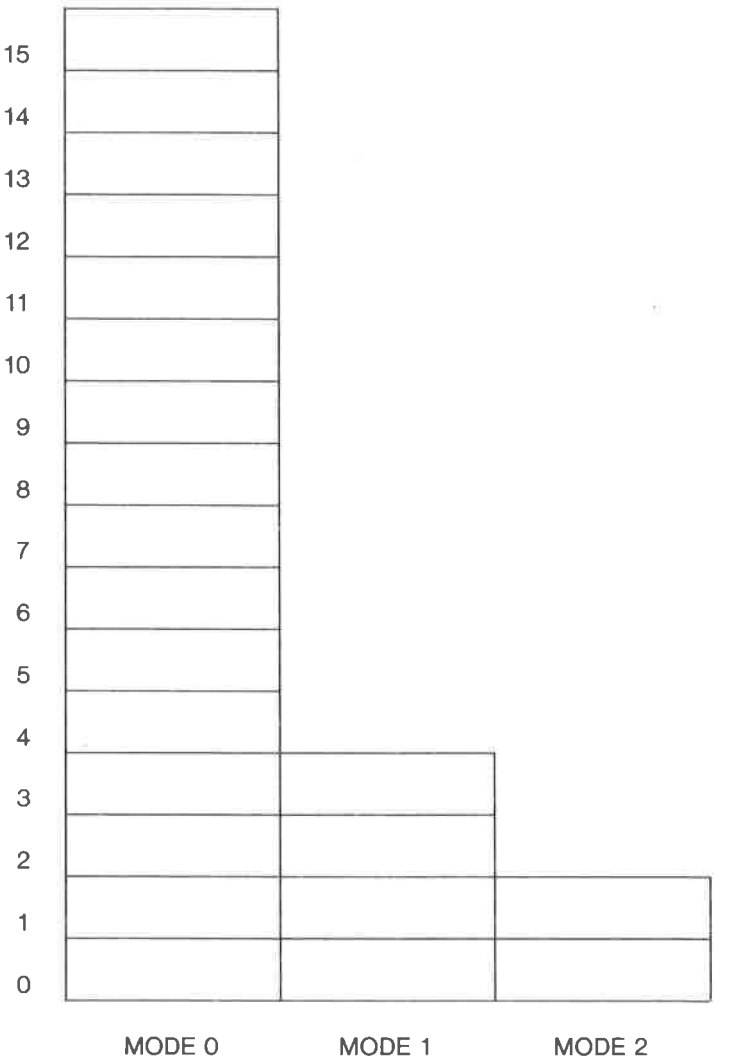



**Bild 2.1:** *Farbplaner*

### 2.2.1 Spezialeffekte mit Hilfe der Farbdefinition

Beginnen wir mit einem kleinen Experiment. Schreiben Sie einmal einen kleinen Text auf den Schirm beispielsweise

```
PRINT"Dies ist ein Probetext"
```

Wenn Sie an den Anfangswerten keine Änderungen vorgenommen haben, so haben Sie jetzt mit gelb (PEN 1, INK 24) auf dunkelblau (PAPER 0, INK 1) geschrieben. Tippen Sie nun einmal ein

```
PEN 2:PRINT"Noch ein Text"
```

Diesen Ausdruck gibt der Computer nun rot wieder. Nun kommt die entscheidende Änderung. Sie definieren nämlich

```
INK 1,15
```

Was ist passiert? Der erste Beispieltext hat die Farbe gewechselt und wird nun in orange beziehungsweise auf einem grünen Monitor dunkler dargestellt. Der zweite Text dagegen ist unverändert. Bei einer INK-Umdefinition wird also der gesamte auf dem Bildschirm befindliche Text, der mit dieser Farbe beschrieben wurde, farblich umdefiniert. Daneben ändert sich natürlich auch die aktuelle Schriftfarbe, wenn PEN oder PAPER diese INK benutzen. Sie können dies leicht überprüfen, wenn Sie nun

```
INK 2,26
```

eingeben. Jetzt ändern sich parallel die Farbe des gerade gesetzten zweiten Textes und auch die aktuelle Schriftfarbe, da diese ja oben auf das Farbregister Nummer 2 definiert wurde. Mit Hilfe dieses Prinzips lassen sich nun eine ganze Reihe interessanter und teilweise auch verblüffender Effekte erzielen, die oft in Spielen angewandt werden. Ein paar besonders hübsche Tricks wollen wir uns jetzt einmal anschauen.

#### 2.2.1.1 Warnmeldungen

Ein Bereich, in dem man direkt, wie gerade beschrieben vorgehen kann, sind die verschiedenen Warnmeldungen, Ausgaben, die immer dann erscheinen sollen, wenn ein Fehler oder ein Crash (bei Spielen) aufgetreten ist. Schauen wir uns einmal an, über welche Varianten wir hier verfügen.

**Der blitzende Bildschirm:**

Vorgehensweise: Man gibt einen Text wie oben beschrieben aus. Tritt dann der Fehler auf, so definiert man kurzzeitig das INK-Register um (auf die Warnfarbe), bevor dann nach einer gewissen Zeit wieder zurückgeschaltet wird. Dieses Verfahren hat jedoch den Nachteil, daß der gesamte Schirm vibriert, wenn zwischendurch nicht auf eine andere Schreibstiftfarbe umgeschaltet wurde.

Dieses Manko können wir relativ leicht umgehen. Es geht jedoch auch viel besser. Zum ersten kann man sich auf ein Blinken des Randes (BORDER) beschränken. Dieses Verfahren wird bei einigen Action-Spielen aus der Weltraum-Szene angewandt, um Beschleunigung, den Hyperantrieb oder das Wirken der Zeitmaschine zu veranschaulichen. Definiert man nämlich

```
INK 1,6,24:BORDER 1
```

so blinkt nun der Rand rot gegen gelb. Dies ist jedoch erst der Anfang. Als nächsten Schritt greifen wir auf die Blinkgeschwindigkeit zu. Dies erreichen wir mit dem Befehl

```
SPEED INK <Zeit erste Farbe>,<Zeit zweite Farbe>
```

Dieser Befehl bestimmt die Blinkfrequenz in 1/50 Sekunden. Er gilt dabei für alle Farbdefinitionen parallel. Wenn wir also zwei Farbregister mit verschiedenen Farben (doppelt) belegen, so erfolgt das Umschalten immer gleichzeitig. Eine Differenzierung bei der Blinkgeschwindigkeit der einzelnen INKs ist nur in Maschinensprache möglich. Arbeiten wir jedoch mit einem reinen BASIC-Programm, so blinken alle Farben auf dem Schirm immer gleichzeitig im gleichen Rhythmus.

Die Normalwerte liegen hier bei (10,10). Eine Farbe wird also genau für 1/5 Sekunde aufgetastet, bevor der Computer die zweite Farbe freigibt. Die Definition der Farbwechsel ist nun aber nicht fix definiert. Wir können das Verhältnis der beiden Einschaltphasen und natürlich auch ihre absolute Dauer jederzeit, jede 1/50 Sekunde ändern. Wenn wir nun diese Zahlen im Zeitablauf ändern, so ergibt sich ein Beschleunigungseffekt. Ein kurzes Programm mag dies illustrieren.

```
10 BORDER 24,6
20 FOR i= 50 TO 3 STEP -1
30 SPEED INK i,i
40 FOR t=1 TO SQR(100*i^2):NEXT t
50 NEXT i
```

**Programmbeschreibung:**

Zuerst wird der Rand mit BORDER auf gelb und rot definiert. Diese Farbe würde nun ohne Änderungen gleichmäßig blinken. Dabei sind dem CPC aber zwei Schleifen im Weg. Die erste FOR-TO-Schleife erhöht die Blinkfrequenz und vermindert die Zeit zwischen zwei Farbwechseln mit Hilfe des nachfolgenden SPEED INK-Kommandos.

Zwischen jedem Schritt dieser Schleife muß der CPC aber erst noch die Zeitschleife in Zeile 30 abarbeiten. Diese führt zu einer Verzögerung in Abhängigkeit von i. Mit abnehmendem i erfolgen also auch die Farbwechsel schneller.

Bis jetzt haben wir nur die Blinkfrequenz geändert. Wir können jedoch auch mit veränderlichen Farben arbeiten. Dies könnte so aussehen.

```
100 FOR i=100 TO 1 STEP-1:BORDER i MOD 16
110 FOR t=1 TO i:NEXT t,i
```

**Programmbeschreibung:**

Jetzt werden die verschiedenen Farben nacheinander durchgetastet. Eine Zeitschleife sorgt wieder für die wechselnde Verzögerung.

Sie können natürlich nun auch beide Effekte, den Farbwechsel und das wechselnde Blinken miteinander kombinieren. Damit sind dann noch weitere Steigerungen möglich.

Neben dem Einsatz im Spielebereich haben Warnungen natürlich auch in etwas größeren Anwenderprogrammen ihren Sinn. Auch hier gibt es mehrere Möglichkeiten. Den optisch besten Effekt erzielt man, indem man einen Text mit ein und derselben Farbe aus mehreren verschiedenen Farbregistern schreibt. Dies könnte beispielsweise so aussehen:

```
10 INK 0,0:INK 1,6:INK 2,6:INK 3,6
20 PRINT"Dies ist ein";:PEN 2:PRINT"Demonstrationsprogramm, ";
30 PEN 1:PRINT"das die Wirkung der ";
40 PEN 3:PRINT"Farbumdefinition ";
50 PEN 1:PRINT"illustrieren soll."
```

Wenn Sie dieses Programm (bitte achten Sie bei der Eingabe auf die richtige Stellung der Strichpunkte) laufen lassen, so erhalten Sie zusammenhängend in einem fort die folgende Textausgabe in roter Farbe:

```
Dies ist ein Demonstrationsprogramm, das die Wirkung der Farbumdefinition
illustrieren soll.
```

Geben Sie nun ein INK 3,24, so leuchtet das Wort Farbumdefinition auf. Der restliche Text bleibt jedoch unverändert. Denselben Vorgang können Sie natürlich auch für Farbregister 2 durchführen. Dann würde zum Beispiel das Wort Demonstrationsprogramm "umkippen". Sie sollten diese Effekte nun ruhig einmal anhand einiger Beispieleingaben durchspielen, um sich mit ihnen vertraut zu machen.

### 2.2.1.2 Unsichtbares wird sichtbar - Das Auftasten von Farben

Mit dem letzten kleinen Programm haben wir auch schon einen weiteren Trick benutzt, der speziell im Spielebereich gerne angewandt wird. Geben Sie einmal ein

```
10 INK 0,0:INK 1,0:INK 2,24:PEN 1:PAPER 0
20 CLS:PRINT"Beispieltext"
30 PEN 2
```

Was sehen Sie, wenn Sie dieses Programm laufen lassen? NICHTS. Jedenfalls, wenn wir einmal das abschließende "Ready" außer Acht lassen. Dennoch wurde unser Beispieltext vollständig geschrieben und ist auch auf dem Schirm vorhanden. Das einzige Problem besteht darin, diesen sichtbar zu machen. Das heißt, für uns ist dies eigentlich schon kein Problem mehr. Ein einfaches

```
INK 1,6
```

und unsere Ausgabe wird wieder sichtbar. Dieser Effekt hat ein breites Anwendungsfeld. Zum ersten können Sie mit Hilfe einer Dunkeltastung eine Code-Wort-Eingabe noch sicherer machen und dabei völlig problemlos auf den INPUT-Befehl zurückgreifen. Wenn man normalerweise ein Programm mit einem Code schützen will, so muß man die INPUT-Eingabe mit INKEY$ und ähnlichen Funktionen simulieren.

Grund: Ein neugieriger Mit-Betrachter kann sonst das Kennwort bei der Eingabe oder Korrektur mitlesen, womit es seinen Zweck natürlich nicht mehr erfüllen kann.

Da bei INPUT die Zeichen immer auf dem Schirm dargestellt werden, ist dies nicht zu vermeiden. Wenn wir nun aber die Schreibfarbe auf den Hintergrund setzen, so erfolgt die Zeichendarstellung schwarz auf schwarz und damit sieht ein Späher nichts mehr.

Dennoch bleiben uns die vollen Editiermöglichkeiten des INPUT-Befehls erhalten. Der CPC analysiert nämlich den Bildschirminhalt anhand der INKs. Und da wir zwei verschiedene Farbregister benutzt haben, kann

der Schneider problemlos beispielsweise zwischen schwarz 1 und schwarz 2 unterscheiden.

Interessanter ist jedoch noch eine andere Anwendung. Will man zum Beispiel in einem Spiel einen komplizierten Bildschirmaufbau realisieren, so kostet dies meist in erheblichem Maße Rechnerzeit. Der Computer braucht eben manchmal ein Weilchen. Dies muß jedoch nicht immer zu eine Zwangs-Kaffeepause führen.

In jedem Fall aber vermittelt der langsame Bildaufbau mit gegebenenfalls mehreren aufeinander folgenden Änderungen einen unprofessionellen Eindruck. Hier können wir mit einem ähnlichen Trick Abhilfe schaffen.

Wir definieren zunächst in der Arbeitsversion eine Bildschirmausgabe. In der endgültigen Version fügen wir nun noch ein paar zusätzliche Zeilen ein. Am Anfang werden alle INKs auf 0 gesetzt. Der Bildschirmaufbau erfolgt damit schwarz auf schwarz. Nachdem das gesamte Bild fertiggestellt ist, wird es dann auf einen Schlag aufgetastet. Das Prinzip zeigt wieder ein kleines Programm.

```
10 '************************
20 '** Demo Farbauftastung **
30 '************************
40 MODE 0
50 FOR i=0 TO 15:INK i,0:NEXT
60 FOR i=1 TO 15:
70 PEN i:PRINT"Diese Zeile INK";i
80 NEXT
90 FOR i=1 TO 15: INK i,i:NEXT
```

**Programmbeschreibung:**

Nach der Umschaltung in den MODE 0 - dieser wurde gewählt, um möglichst viele Farben gleichzeitig darstellen zu können - werden zunächst alle INK-Register auf 0 gesetzt. Dann schreibt der CPC 15 Probetexte mit 15 verschiedenen Vordergrundfarben. Sie werden jedoch unsichtbar (schwarz auf schwarz) geschrieben. Erst durch die Farbdefinition in Zeile 90 werden dann alle Texte auf einen Schlag sichtbar.

Die Routine gibt Ihnen übrigens auch die unteren 15 Farben im Vergleich aus. Dies ist für Farbvergleiche manchmal recht nützlich. Wichtig ist hier aber vor allem, wie - vermeintlich - schnell der Bildschirmaufbau vonstatten geht. Wenn Sie ab und zu auch einmal ein Spiel für den CPC anschauen, so werden Sie diesen Trick des öfteren finden.

Nun ist es wieder an Ihnen, mit diesen Tricks und Effekten zu experimentieren und Ihren Programmbestand darauf zu untersuchen, ob sich die eine oder andere Routine durch derartige Tricks nicht noch aufwerten läßt. Sei es durch eine bessere Benutzerführung oder auch nur durch ein besseres optisches Auftreten des Programms.

## 2.3 Bilder mit der Blockgrafik

Nachdem wir uns ausgiebig mit der Definition von Zeichen und den verschiedenen Darstellungsarten beschäftigt haben, ist es an der Zeit, unsere gesammelten Erfahrungen in eine anwendungsfreundliche Form zu bringen. Wir wollen dies mit mehreren Hilfsprogrammen tun. Als erstes wollen wir uns damit beschäftigen, wie wir die Zeichen der Blockgrafik auch über die Tastatur eingeben können, bevor wir dann die dazu notwendige Routine zu einem größeren Programm erweitern, das uns die Bearbeitung ganzer Blockgrafik-Bilder ermöglicht.

Beginnen wir mit dem ersten Schritt. Bevor wir uns überlegen, wie wir die Tastatur ändern können, müssen wir uns zuerst einmal damit beschäftigen, wie der CPC die Tastatur überhaupt verwaltet.

### 2.3.1 Blockgrafik auf der Tastatur

Auf den ersten Blick ist die Tastatur des CPC eine relativ einfache Einrichtung. Ein Haufen hell- und dunkelgrauer Tasten - 74 an der Zahl um genau zu bleiben - die auf Betätigung Zeichen auf den Bildschirm bringen. Doch ganz so einfach ist die Sache leider, oder vielleicht sollte man besser sagen Gott sei Dank, dann doch nicht. Ein kleines Experiment mag dies verdeutlichen.

Drücken wir nach KEY DEF 67,1,189 auf die Q-Taste, so erscheint ein griechisches Omega. Oder wie wäre es mit Folgendem:

```
KEY DEF 67,1,130
KEY 130,"PEN 0"+CHR$(13)+"INK 0,0:BORDER 0:MODE 0:DRAW 640,400,15"+CHR$(13)
```

Durch Druck auf Q wird es jetzt grafisch und bunt. Doch damit nicht genug. Durch einfache Umdefinition ist es möglich, eine Serie von Grafikzeichen durch Tastendruck auf den Schirm zu bringen und auch

das Arbeiten mit zwei Grafikmodes gleichzeitig (beispielweise MODE 1 und MODE 0) ohne Umstellung, ist möglich.

Bei so viel Neuem ist ein wenig Systematik notwendig. Beim Drücken einer Taste laufen nacheinander drei verschiedene Vorgänge ab.

1. **Tastaturabfrage**: Hier wird festgestellt, welche Tasten gedrückt sind. Diese werden in einem Zwischenspeicher mit der Angabe, ob SHIFT oder CTRL gleichzeitig gedrückt sind, abgelegt.

2. **Tastaturübersetzung**: Dieser Funktionsteil ordnet einer Taste ein Zeichen oder eine Zeichenkette zu.

3. **Zeichendarstellung**: Hier wird schließlich ein Zeichen oder eine Zeichenkette auf dem Schirm ausgegeben.

Wir wollen uns die verschiedenen Funktionen nun einmal nacheinander anschauen. Dabei werden wir uns auf die ersten beiden Punkte konzentrieren. Die Zeichendarstellung kommt später noch (im zweiten Teil dieses Buches) zur Sprache.

#### 2.3.1.1 Der Tastatur auf's Bit geschaut: Die Tastaturdecodierung

Zunächst zum ersten Schritt, der Tastaturabfrage. Für den CPC ist dies eigentlich recht einfach. Insgesamt gibt es 80 Tasten, oder tastenähnliche Gebilde mit Nummern von 0-79, wobei die Nummer 78 nicht belegt ist. Die Zuordnung der Tasten zu ihren Nummern findet sich im Handbuch Anhang 3 auf Seite 16.

Die Tastennummern 0-71 und 79, breit verteilt und munter gestreut, gehören zur eigentlichen Tastatur. Die Joystick-Abfrage benutzt die Nummern 72-77 für Joystick Nr.0. Joystick Nr. 1 überlagert die Tastatur. Das Hochsteuern des Joysticks Nr.1 hat also denselben Effekt wie ein Druck auf die Taste 6. Und den Firebutton von Nummer 1 kann man auch mit dem Betätigen von F oder G nachbilden.

Die etwas seltsame Streuung der Tastennummern ergibt sich dabei aus der Computer-Hardware oder besser der Art, wie der CPC die Tastatur abprüft. Alle Tasten und auch die Schaltverbindungen für die Joysticks sind in einer Matrix aus 8*10 Leitungen angeordnet. Jede Taste befindet sich dabei auf dem Schnittpunkt einer Reihenleitung mit einer Spalte. Durch Testen jeder Reihe gegen jede Spalte (Scannen) kann der Schneider dann überprüfen, welche Tasten aktuell gedrückt sind. Wenn eine Taste gedrückt ist, so sind nämlich die dazu gehörenden zwei Leitungen (Reihe und Spalte) verbunden. Ansonsten stellt der CPC keine Verbindung fest.

Dieses Abprüfen aller Tasten, beziehungsweise der möglichen Reihe/Spalte-Paarungen erfolgt dabei 50 mal pro Sekunde, so daß auch ein nur kurzes Antippen einer Taste registriert werden kann. Die Tastaturabfrageroutine beschränkt sich aber nun nicht nur darauf, zu überprüfen, ob eine Taste gedrückt wurde, sondern stellt auch gleich noch den zum gleichen Zeitpunkt existierenden Zustand von SHIFT und CTRL fest. Dementsprechend kann eine Taste 5 Zustände annehmen:

| | | |
|---|---|---|
| Taste nicht gedrückt | = | -1 |
| Taste gedrückt | = | 0 |
| Taste plus <SHIFT> gedrückt | = | 32 |
| Taste plus <CTRL> gedrückt | = | 128 |
| Taste plus <SHIFT> plus <CTRL> gedrückt | = | 160 |

Mit INKEY(TASTENNUMMER) können wir jederzeit diesen Wert für jede beliebige Taste abfragen und mit Hilfe einer Abfrage der Form

```
IF INKEY(Tastennummer) = <Wert>
```

ist es dann möglich, daraus eine bestimmte Reaktion abzuleiten. Da die INKEY-Funktion softwaremäßig nicht gegen mehrfachen Tastendruck verriegelt ist, sind wir damit auch in der Lage, mehrere Tasten gleichzeitig abzufragen und somit ein Pianokeyboard oder ähnliches zu realisieren.

### 2.3.1.2 Die Zuordnung von Tasten und Bildschirmcodes

Nach der Tastaturabfrage beginnt die Tastaturübersetzung. Wir verlassen dazu die Oberfläche unseres wohlgeordneten BASICS und wandern ein wenig durch den Speicher. Als Gefährt dient uns dabei das im folgenden abgedruckte kleine Analyseprogramm.

```
10 '*********************
20 '** Analyseprogramm **
30 '*********************
40 CLS:INPUT"Anfang";an:INPUT"Ende  ";en
50 FOR i=an TO en
60 IF INKEY(47)<>0 THEN 60
70 PRINT i,PEEK(i);:IF PEEK(i)>32 THEN PRINT CHR$(PEEK(i)) ELSE PRINT
80 NEXT:GOTO 40
```

**Programmbeschreibung:**

Die Speicheranalyseroutine gibt im gewählten Bereich die Inhalte der einzelnen Zellen und ihre Darstellung als Zeichen aus. Um eine verse-

hentliche Ausführung der Steuerzeichenkommandos zu verhindern, erfolgt die Darstellung nur, wenn ein Charactercode größer als 32 gefunden wurde. Die Space-Taste wird dabei als Tote-Mann-Bremse benutzt. Wenn Sie also <SPACE> loslassen, stoppt das Programm bis zum nächsten Tastendruck.

Um dies zu erreichen wurde die INKEY-Funktion angewandt. Die <SPACE>-Taste hat die Nummer 47 und daher überprüft INKEY(47)<>0, ob diese Taste nicht gedrückt ist. Diese Abfrage wiederholt sich dabei so lange, bis die Taste angetippt wird und damit die nächste Speicherzelle ausgegeben wird.

Ziel unserer Untersuchungen sind die Bytes um 45900 und mehr, d.h. die Tastaturübersetzungstabelle. Die eigentliche Tastaturübersetzungstabelle geht von 45900 bis 46139. Insgesamt sind das 240 Bytes, was sich aus drei Ebenen (normal, <SHIFT>, <CTRL>) für alle 80 Tastennummern ableiten läßt. Die Aufteilung dieses Bereiches findet sich in Tabelle 2.1.

Den Anfang macht in Speicherstelle 45900 die Belegung der Normalebene für die CURSOR HOCH-Taste (Nr. 0). Mit der DEL-Taste (Nr. 79) endet dann in 45979 die Übersetzungstabelle für die Normalebene. Das ganze wiederholt sich dann noch einmal für <SHIFT> und <CTRL>.

Die vierte denkbare Ebene (<SHIFT> und <CTRL> gleichzeitig gedrückt) wird vom Computer nicht unterschieden; er springt dann automatisch in die Übersetzungstabelle für <CTRL>. Will man hier trotzdem eine Unterscheidung bewirken, so ist nur der Umweg über die INKEY-Funktion (INKEY(Tastennummer)=160) möglich, um das gleichzeitige Drücken von <CTRL> und <SHIFT> festzustellen.

**Tabelle 2.1:** *Die Tastaturübersetzungstabelle*

| Adresse | Bedeutung |
|---|---|
| 45900 | Belegung Normalebene Taste Nr. 0 |
| 45901 | Belegung Normalebene Taste Nr. 1 |
| . | |
| 45979 | Belegung Normalebene Taste Nr.79 |
| 45980 | Belegung Shiftebene Taste Nr. 0 |
| . | |
| 46059 | Belegung Shiftebene Taste Nr.79 |
| 46060 | Belegung CTRL-Ebene Taste Nr. 0 |
| . | |
| 46139 | Belegung CTRL-Ebene Taste Nr.79 |
| 46140 | Abspeicherung Repeat Taste 0-7 : Repeat=1, kein Repeat=0 |
| 46149 | Ende Abspeicherung Repeat |
| 46150 | Sprungzeiger Exp. 128 |
| 46151 | Zeichen 1 Exp. 128 |
| . | |
| 46150+n | Zeichen n |
| 46151+n | Sprungzeiger Exp. 129 |
| . | |

Die einzelnen Speicherstellen in der Übersetzungstabelle können Werte zwischen 0 und 255 annehmen. 0 und 255 fungieren dabei als Platzhalter, das heißt, es passiert nichts. Bei den anderen Nummern werden entweder Kontrollzeichen-Kommandos ausgeführt oder grafische Symbole und Erweiterungszeichen auf dem Bildschirm dargestellt. Eine genauere Unterteilung findet sich in Tabelle 2.2.

Die Werte zwischen 1 und 31 bringen die Kontrollzeichenkommandos (siehe Kapitel 2.4) auf den Schirm. Beim Einschalten des Rechners werden sie auf die CTRL-Ebene der Tastatur definiert und zwar so, daß sie in aufsteigender Reihenfolge dem Alphabet entsprechen. <CTRL A> bringt also das Kontrollzeichen-Kommando Nr. 1, <CTRL B> Nr. 2 und so weiter.

**Tabelle 2.2:** *Die Bedeutung der Werte in der Übersetzungstabelle*

| Code | Bedeutung |
|---|---|
| 1-31 | Druck der Kontrollzeichen-Kommandos, dabei werden die Nummern 13=Enter und 16=CLR ausgeführt |
| 32-126 | Grafiksatz (Anhang III) |
| 127 | DEL |
| 128-159 | Erweiterungszeichen (expansion characters) |
| 160-223 | Grafiksatz |
| 224 | Copy |
| 225 | CTRL Copy |
| 226-238 | Grafiksatz (Firmwarezeichensatz nach Anhang III) |
| 240 | CSR hoch ausführen |
| 241 | CSR runter |
| 242 | CSR links |
| 243 | CSR rechts |
| 244 | Copy CSR hoch |
| 245 | Copy CSR runter |
| 246 | Copy CSR links |
| 247 | Copy CSR rechts |
| 248 | CTRL+CSR hoch |
| 249 | CTRL+CSR runter |
| 250 | CTRL+CSR links |
| 251 | CTRL+CSR rechts |
| 252 | ESC |
| 253 | CAPS LOCK |
| 254 | SHIFT LOCK |
| 255 | frei SHIFT/CTRL |

Einen Spezialfall bilden dabei die Kontrollzeichen-Kommandos 13 und 16, besser bekannt durch die Tastenaufschriften ENTER beziehungsweise CLR. Im Gegensatz zu den anderen Kontrollzeichen werden diese direkt ausgeführt, das heißt, es wird das Zeichen unter dem Cursor gelöscht, beziehungsweise in die nächste Zeile gesprungen. Will man die anderen Kontrollzeichen ausführen, so muß man sie mit dem PRINT-Kommando ausgeben. Als Beispiel soll uns hier das Kontrollzeichen Nr. 24, welches auf der X-Taste liegt, dienen. Es bewirkt reverses Schreiben, das heißt, die Farben für PEN und PAPER werden ausgetauscht. Mit

```
PRINT"<CTRL X>TEXT<CTRL X>"
```

wird also "TEXT" revers ausgegeben und danach wieder auf Normaldarstellung zurückgeschaltet. Wir werden uns mit der Wirkung der verschiedenen Steuercodes in Kapitel 2.4 noch näher beschäftigen.

Trifft der CPC in der Tastaturübersetzungstabelle auf einen Wert zwischen 32 und 238, so schaut er im Firmwarezeichensatz beziehungsweise bei den neu definierten Zeichen nach und gibt dann den entsprechenden Character auf dem Bildschirm aus. Leider gibt es auch

von dieser goldenen Regel wieder einige Ausnahmen. Die Codes 127, 224 und 225 haben andere Bedeutungen. Sie rufen die Editorroutinen für "Delete" beziehungsweise "Copy" auf.

Auf den anderen größeren Bereich, die *expansion characters* (Erweiterungszeichen) zwischen 128 und 159 kommen wir später noch zurück.

Die letzten 16 Werte (von 240 bis 255) sind wiederum dem Editor vorbehalten. Mit den Werten 240-243 wird der Cursor gesteuert. Die Codes ab Nr. 240 wirken also ähnlich wie <ENTER> oder <CLR> im unteren Bereich.

### 2.3.1.3 Tastaturänderungen mit KEY DEF und POKE

Wie können wir nun konkret die Werte in den Tastaturtabellen ändern?

Dies können wir sowohl mit dem KEY DEF-Kommando als auch mit POKE erreichen. Bei KEY DEF folgt nach dem Befehlstoken zunächst die Nummer der Taste. Danach kommt eine "1", wenn die Taste auf Repeat geschaltet sein soll beziehungsweise eine "0", falls dies nicht gewünscht wird.

Es folgen drei Parameter, die festlegen, mit welchem Zeichen oder welcher Zeichenkette diese Taste in den verschiedenen Ebenen verknüpft sein soll. Direkter, aber teilweise mit ein bißchen Rechenarbeit verbunden, geht es mit POKE. Will man zum Beispiel in der SHIFT-Ebene der Taste 24 (Hochpfeil) das Pfund-Zeichen durch ein ß ersetzen, so gelingt das entweder mit

```
KEY DEF 24,1,94,177
```

oder mit

```
POKE 46004,177
```

Den Wert von 46004 erhält man dabei als Summe aus 45980 (=1. Taste in der SHIFT-Ebene) und 24 (Tastennummer).

Eine andere Anwendung besteht darin, die etwas obskure TAB-Taste zu einer nützlichen Anwendung zu führen. Neben <CAPS LOCK> verfügt der CPC nämlich, wie Sie aus Tabelle 2.2 entnehmen können, auch noch über eine SHIFT LOCK-Funktion. Während bei <CAPS LOCK> die Großbuchstaben permanent eingeschaltet sind, reagieren die Zahlen und die Sonderzeichen ganz normal. <SHIFT LOCK> würde nun auch diesen

Bereich umschalten. Der Code für <SHIFT LOCK> ist 254, die TAB-Taste hat die Nummer 68. Mit

```
KEY DEF 68,1,254,254,254
```

haben wir eine SHIFT LOCK-Taste etabliert (bitte ausprobieren!). Mit derselben Methode können wir nun aber auch eine deutsche Normtastatur definieren und so den in Kapitel 2.1 entwickelten deutschen Zeichensatz auf die Tastatur nach der DIN-Norm legen. Dazu genügen ein paar kurze Kommandos. Zunächst müssen wir y und z vertauschen. Dann müssen noch die Tasten für die Umlaute mit den neu definierten Characters belegt werden. Hier nachfolgend die benötigten Kommandos:

```
KEY DEF 71,1,121,89:' "Z"-Taste auf y,Y
KEY DEF 43,1,122,90:' "Y"-Taste auf z,Z
KEY DEF 28,1,123,91:' ";"-Taste auf ä,Ä
KEY DEF 29,1,124,92:' "'"-Taste auf Ö,ö
KEY DEF 26,1,125,93:' "Ä"-Taste auf Ü,ü
KEY DEF 25,1,126,63:' "-"-Taste auf ß und ?
```

Neben Buchstaben, Zahlen und Sonderzeichen können wir natürlich nun auch unsere Blockgrafiksymbole auf die Tastatur legen.

Bevor wir uns diesem Thema näher zuwenden, müssen wir allerdings noch auf einen weiteren Punkt eingehen, die Repeatdefinition. Dabei handelt es sich um die Frage, ob eine Taste auf Repeat definiert ist oder nicht. Ob sie also bei längerem Tastendruck fortlaufend Zeichen ausgibt oder nur einmal.

Dazu stehen im Speicher des CPC 10 Bytes zur Verfügung. Die Abspeicherung erfolgt bitorientiert, das heißt für jede Taste steht nur ein Bit zur Verfügung. Ist dieses Bit gesetzt, gibt es eine Wiederholung, ansonsten bleibt es bei der ersten, einmaligen Ausgabe eines Zeichens.

Die Informationen darüber sind in den Bytes 46140-46149 abgespeichert, und zwar von rechts nach links laufend. Das niederwertigste Bit von Speicherstelle 46140 steht also für die Taste Nr. 0, das 2er Bit von 46140 für Taste 1 und so weiter. Nach dem Einschalten erhalten wir auf die Abfrage

```
PRINT RIGHT$("00000000"+BIN$(PEEK(46140)),8)
```

die Ausgabe 00000111. Die Taste 0 ist also auf REPEAT gesetzt. Ebenso die Tasten mit den Nummern 1 und 2 (<CSR RECHTS> und <CSR UNTEN>). Geben wir nun

```
KEY DEF 0,0
```

ein, und wiederholen unsere Abfrage, so ist das kleinste Bit auf 0 gesetzt und der Cursor kann nun nicht mehr nach oben laufen. Wenn Sie also nun auf <CURSOR HOCH> tippen, so bewegt sich dieser nur einmal.

```
KEY DEF 0,1
```

setzt die Repeat-Verteilung wieder in den Normalzustand zurück. Der zweite Parameter im KEY DEF-Kommando gibt nämlich an, ob bei einer Taste die Wiederholfunktion aktiv sein soll oder nicht. Neben KEY DEF können wir jedoch natürlich auch mit PEEK und POKE den Repeat-Status ändern. Wollen wir eine Taste auf Repeat setzen, so müssen wir dabei folgendermaßen vorgehen: Zunächst muß anhand der Tastennummer das zu ändernde Byte festgelegt werden. Die Tasten 0-7 sind im ersten Repeat-Byte (Adresse 46140) gespeichert. Je 8 Tasten sind also in einem Repeat-Byte abgelegt. Entsprechend liefert uns

```
PRINT INT(<Tastennummer>/16)+46140
```

die Adresse, die unsere Taste betrifft. Das zu ändernde Bit in diesem Wert bestimmt man dann mit Hilfe der MODULO-Funktion.

```
PRINT <Tastennummer> MOD 8
```

gibt es uns an. Dabei ist Bit 0 das niederwertigste, rechte Bit. Bit 7 wäre das höchstwertige. Um nun den Wiederhol-Status zu ändern, benutzen wir die logischen Verknüpfungen "OR" und "AND". Soll die Taste auf Repeat gesetzt sein, so setzen wir alle anderen Stellen in unserem Byte auf 0 und führen eine OR-Verknüpfung aus. In der umgekehrten Reihenfolge müßten alle anderen Bits auf 1 gesetzt werden. Das zu ändernde wäre zu nullen und mit der so erhaltenen Maske wäre der Inhalt unseres Repeat-Bytes zu verknüpfen.

**Beispiel:**

Taste Nummer 47 soll auf Repeat gesetzt werden. Wir rechnen wie folgt:

```
PRINT INT(47/8)+46140      PRINT 47 MOD 8
```

ergibt den Wert

```
46145                      7
```

Es muß also Bit 7 in Adresse 46145 geändert werden. In unserem Falle, also beim Setzen, wäre das folgende Kommando erfolgreich:

```
POKE 46145,PEEK(46145) OR &x10000000
```

Analog dazu könnten wir mit

```
POKE 46145,PEEK(46145) AND &x01111111
```

ein Rücksetzen erreichen. Es ist natürlich nun nicht besonders sinnvoll diesen direkten Zugriff auf die Repeatspeicherung bei einer zu ändernden Taste vorzunehmen. Der Rechenaufwand stünde in keinem Verhältnis zu den erreichbaren Vorteilen. Soll jedoch parallel bei einer ganzen Reihe von Tasten der Wiederholungsmodus umdefiniert werden und haben die Tasten darüber hinaus nah beieinander liegende Tastennummern, so ist dieses Verfahren deutlich schneller und auch weniger aufwendig. So kann man zum Beispiel mit

```
POKE 46140,255
```

Repeat für die unteren acht Tasten gleichzeitig setzen. Wollte man denselben Effekt mit BASIC erreichen, so wären 8 KEY DEF-Befehle nötig. Die Definition mit POKE ist zum Beispiel dann sinnvoll einsetzbar, wenn wir die abgesetzte 10er Tastatur en bloc auf Repeat (beim Einschalten ist sie auf einmalige Ausgabe programmiert) definieren wollen.

### 2.3.1.4 Tastaturumdefinition per Programm: KEY

Wir wissen nun, wie die Zuordnung von Zeichen zu Tasten und wie die Abspeicherung des Repeatmodus im Computerinneren abläuft. Es ist nun an der Zeit, dieses Wissen in handliche Hilfsprogramme und nützliche Routinen umzusetzen.

Als erstes wollen wir ein Programm-Modul schreiben, mit dem wir die verschiedenen Ebenen unserer Tastatur möglichst komfortabel umdefinieren können. Ein solches Programm sollte in einem Menü mehrere Zeichensätze zur Auswahl bereitstellen, die dann je nach Bedarf auf die verschiedenen Ebenen definiert werden können. Eine Routine, die dies leistet finden Sie unter der Bezeichnung KEY nachfolgend abgedruckt.

## Listing 2: *Programm KEY*

```
10 REM **************
20 REM ** key 464 **
30 REM **************
40 MODE 1:BORDER 0
50 WINDOW#1,1,40,1,3:WINDOW#0,1,40,4,22:WINDOW#2,1,40,23,25
60 INK 0,0:INK 1,24:INK 2,6:INK 3,11
70 PAPER 0:PEN 1:PAPER#1,3:PEN#1,2:PAPER#2,3:PEN#2,2
80 CLS:CLS#1:CLS#2:LOCATE#1,13,2:PRINT#1,"Tastendefinition"
90 DIM nr(26),z(6,26)
100 CLS#2
110 PRINT:PRINT " Linien / Blockgrafik 1                (1)"
120 PRINT" Linien / Blockgrafik 2               (2)"
130 PRINT" Blockgrafik 3 + Sonderzeichen        (3)"
140 PRINT" griechisches Alphabet+Sonderzeichen (4)"
150 PRINT" Grossbuchstaben                      (5)"
160 PRINT" Kleinbuchstaben                      (6)"
170 DATA 67,59,58,50,51,43,42,35,34,27,69,60,61
180 DATA 53,52,44,45,37,36,71,63,62,55,54,46,38
190 RESTORE 170
200 FOR i=1 TO 26:READ nr(i):NEXT i
210 FOR i=128 TO 159:KEY i,CHR$(i):NEXT
220 PRINT#2,"Tastaturbelegung in 'NORMAL'-Ebene.";
230 INPUT#2,n
240 IF n>0 AND n<7 THEN 270
250 PRINT#2, "Falsche Eingabe !!!"
260 FOR i=1 TO 1500:NEXT i:GOTO 230
270 PRINT#2,"Tastaturbelegung  in 'SHIFT'-Ebene ";
280 INPUT#2,s
290 IF s<1 OR s>6 THEN 300 ELSE 320
300 PRINT#2,"Falsche Eingabe !!!"
310 FOR i=1 TO 1500:NEXT i:GOTO 280
320 PRINT#2,"Tastaturbelegung  in  'CTRL'-Ebene ";
330 INPUT#2,c
340 IF c<1 OR c>6 THEN 350 ELSE 390
350 PRINT#2,"Falsche Eingabe !!!"
360 FOR i=1 TO 1500:NEXT i:GOTO 320
370 DATA 129,130,131,132,133,134,135,136,137,138,139,140,141
380 DATA 142,143,144,145,146,147,148,149,150,151,152,153,156
390 DATA 192,193,194,195,196,197,198,199,200,201,202,203,204
400 DATA 205,206,207,208,209,210,211,212,213,214,215,216,217
410 DATA 218,219,220,221,222,223,226,160,161,162,163,164,165
420 DATA 166,167,168,169,170,171,172,173,174,175,152,236,237
430 DATA 176,177,178,179,180,181,182,183,184,185,188,189,190
440 DATA 191,203,226,227,228,229,230,231,232,233,234,235,236
450 DATA 81,87,69,82,84,89,85,73,79,80,65,83,68
460 DATA 70,71,72,74,75,76,90,88,67,86,66,78,77
470 DATA 113,119,101,114,116,121,117,105,111,112,97,115,100
480 DATA 102,103,104,106,107,108,122,120,99,118,98,110,109
490 FOR i=1 TO 6:FOR j=1 TO 26:READ z(i,j):NEXT j,i
500 FOR i=1 TO 26:KEY DEF nr(i),1,z(n,i),z(s,i),z(c,i)
510 NEXT i
520 PRINT" Normal : ";:ON n GOSUB 590,600,610,620,630,640
530 PRINT" SHIFT  : ";:ON s GOSUB 590,600,610,620,630,640
540 PRINT" CONTROL: ";:ON c GOSUB 590,600,610,620,630,640
550 CLS#2:PRINT#2,"O.K. j/n ?"
560 Z$=INKEY$:IF LOWER$(Z$)<>"j" AND LOWER$(z$)<>"n" THEN 560
570 IF LOWER$(z$)="n" THEN ERASE nr,z:GOTO 50
580 END
590 RESTORE 370:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
595 RETURN
600 RESTORE 390:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
```

```
605 RETURN
610 RESTORE 410:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
615 RETURN
620 RESTORE 430:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
625 RETURN
630 RESTORE 450:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
635 RETURN
640 RESTORE 470:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:PRI
NT z$
645 RETURN
650 END
```

**Programmbeschreibung:**

Das Programm besteht aus 3 Teilen. Der erste Abschnitt (bis Zeile 360) enthält die Initialisierungsroutinen, das Setzen der Bildschirmfenster und -farben sowie die Abfrage der Belegung in den einzelnen Ebenen. Dazu stehen sechs verschiedene Zeichensätze zur Verfügung:

- Blockgrafik 1
- Blockgrafik 2
- Blockgrafik 3 (incl.Sonderzeichen)
- griechische Buchstaben und Sonderzeichen
- Großbuchstaben
- Kleinbuchstaben

In den Zeilen 220 bis 360 wird abgefragt, welche Zeichensätze für die verschiedenen Ebenen verwandt werden sollen. Die eigentliche Zeichensatzdefinition hängt nun von zwei Blöcken von DATA-Statements ab.

Den ersten finden Sie in den Zeilen 170 und 180. Er enthält die Tastaturnummern für alle Tasten, die im Normalzustand mit einem Buchstaben belegt sind, beginnend mit dem Wert 67 für das "Q" in der zweiten Reihe der Tastatur oben links. Es folgt mit der Tastennummer 59 der Code für die "W"-Taste.

Der zweite DATA-Block enthält die zu definierenden Zeichensätze. Hier finden sich die Zeichen, die dann auf die einzelnen Tasten definiert werden. Jeweils zwei Zeilen enthalten dabei einen Zeichensatz.

Wir müssen nun nur noch die verschiedenen Zeichen auf die einzelnen Tasten definieren, also einem Zeichencode aus dem zweiten DATA-Block eine Taste zuordnen. Dazu genügen zwei kurze BASIC-Zeilen.

Zeile 490 liest zunächst alle Zeichensätze in ein Array der Größe 6*26 ein. Das Element z(2,1) würde hier beispielsweise das erste Zeichen des

zweiten Zeichensatzes beinhalten, also jenen Zeichencode, der auf die "Q"-Taste zu definieren wäre, falls der zweite Zeichensatz ausgewählt ist. Ein Element dieses Feldes hat daher als ersten Index die Angabe des Zeichensatzes, als zweite Angabe folgt die fortlaufende Nummer auf der Tastatur, beginnend mit 1 für die "Q"-Taste.

Es folgt in Zeile 500 die eigentliche Tastaturdefinition. Hier wurde der Einfachheit halber mit KEY DEF operiert. Die Variablen N, S und C enthalten die Informationen darüber, welcher Zeichensatz in welcher Ebene (NORMAL, SHIFT oder CONTROL) verwendet werden soll. Mit Hilfe einer Schleife werden dann alle 26 Buchstabentasten neu belegt.

Ab Zeile 520 gelangen wir in den dritten Teil unseres Programms. Hier findet eine Kontrollausgabe der gerade definierten Zeichensätze statt. Dazu werden die als Zeichencodes im zweiten Block vorliegenden Symbole mit CHR$ zu einem String zusammengebaut. Jeder String beinhaltet dann bei der Ausgabe 26 Zeichen, also einen vollen Zeichensatz.

### 2.3.1.5 Zeichenketten auf Knopfdruck: expansion characters

Einen wichtigen Punkt haben wir bei dieser Programmbeschreibung aber noch unterschlagen. Betrachten wir nämlich die Zeilen 370 und 380, also den ersten Zeichensatz, so stellen wir fest, daß hier das "Q" mit Zeichen 129 belegt wird, um das Grafikzeichen Nummer 129 auf die "Q"-Taste zu definieren. Für das "W" verwenden wir den Code 130 und so weiter.

Nun werden Sie vielleicht fragen, wo denn da die Besonderheit liege. Schließlich sind wir ja auch bei den anderen Tastennummern nicht anders vorgegangen. Um diese Frage zu beantworten sollten Sie einmal den Computer mit

```
<CTRL><SHIFT><ESC>
```

in den Urzustand zurückversetzen und dann versuchen die Taste 67, also das "Q" auf Zeichen 129 zu programmieren. Normalerweise müßte sich dies mit

```
KEY DEF 67,1,129
```

für die Normalebene erreichen lassen. Wenn Sie nun aber auf "Q" drücken, erhalten Sie als Ausgabe -scheinbar völlig unmotiviert - eine "2", also ein Zeichen mit einem völlig anderen Zeichencode, nämlich 50, wie uns ein kurzer Blick in den Anhang unseres Bedienerhandbuches (Anhang III, Seite 3) zeigt.

Das Schlüsselwort zu diesem Rätsel heißt *expansion characters*, Erweiterungszeichen. Der CPC verfügt über 32 von ihnen mit den Nummern 128-159. Jedes dieser Erweiterungszeichen steht als Platzhalter für ein anderes Symbol oder eine ganze Zeichenkette. Definiert man also eine Taste auf Code 128, so wird nicht das Grafiksymbol mit der Nummer 128 dargestellt, sondern der erste Erweiterungsstring aufgerufen und gegebenenfalls ausgeführt.

Einen dieser Erweiterungsstrings kennen Sie alle. Es handelt sich dabei um das automatische Laden und Anlaufen eines Programms mittels RUN". Sie erreichen ihn durch gleichzeitiges Drücken von <CTRL> und der kleinen ENTER-Taste. Schauen wir nun in der Tastaturübersetzungstabelle nach (Speicherstelle 46066 = CTRL-Tabellenanfang + 6 für ENTER), so finden wir dort den Wert 140.

Trifft der Computer bei der Übersetzung der Tastatur auf einen solchen Wert zwischen 128 und 159, so schaut er im Speicher ab Adresse 46150 nach. Dort sind nacheinander die einzelnen Erweiterungsstrings abgelegt. Die Abspeicherung ist relativ simpel und geschieht mittels geeigneter Verzeigerung. Jeder *expansion character* ist im Speicher durch seine Länge und nachfolgend durch die einzelnen Zeichen abgelegt.

In Adresse 46150 finden wir die Länge des ersten Erweiterungszeichens (Zeichencode 128). Dieser liefert die Verzeigerung zum nächsten String und so weiter. Das Ende des Erweiterungszeichen-Bereiches wird durch eine 0 definiert. Stößt der CPC also auf eine Längenangabe von 0, so bricht er die Bearbeitung ab.

Beim Einschalten wird der *expansion character* Nr. 128 mit 0 belegt. Schauen wir uns daher mit unserem Analyseprogramm aus dem ersten Teil die Adressen ab 46150 an, so erhalten wir in 46150 und 46151 die Werte 1 und 48. Die 1 stellt dabei die Länge des abgespeicherten Erweiterungsstrings dar. 48 ist der Zeichencode für 0 (siehe Anhang III, Seite 3 des Handbuches).

Im folgenden sind die anderen Erweiterungszeichen abgespeichert. Ab Speicheradresse 46174 findet man dann mit der Länge 5 unser eben besprochenes RUN" abgespeichert. Das angehängte CHR$(13), der Code für <ENTER> bedeutet dabei, daß der Erweiterungs-String nach dem Ausdruck auf dem Bildschirm direkt ausgeführt wird.

An dieser Stelle sind einige Korrekturen des Handbuches nötig. Die Länge aller in Erweiterungs-Strings benötigten Zeichen darf, wenn man das KEY-Kommando benutzt, 106 Zeichen nicht übersteigen. Eine darüber hinaus gehende Längenbegrenzung für jeden einzelnen String gibt

es nicht. Geht man der Sache jedoch mit POKE zu Leibe, so kann man diese Grenze relativ schnell sprengen. Gibt man nämlich zum Beispiel am Anfang der Erweiterungszeichen

```
POKE 46150,120
```

ein, so hat man nun Platz für einen Supererweiterungsstring mit 120 Zeichen. Doch hier ist höchste Vorsicht geboten. Weiter hinten in diesem Speicherbereich befinden sich nämlich noch einige nicht nur für den CPC nützliche Adressen, auf deren Löschung oder Änderung er gegebenenfalls sehr eigenartig reagiert.

Daneben ist noch ein weiterer Zusammenhang bei der Definition von Erweiterungsstrings zu beachten: Auch wenn Erweiterungsstrings nicht oder auf "" definiert werden, müssen dennoch aufgrund der Verzeigerung alle Strings vor einem neu definierten angelegt sein. Will man also nur einen String definieren und legt diesen auf Nr.159, so werden auch die davor liegenden 31 Strings mit je einer 0 besetzt. Derartige Fehlprogrammierungen schränken also den verfügbaren Speicherplatz und damit die Möglichkeiten in diesem Bereich erheblich ein.

Da der CPC beim Einschalten die Codes von 128-140 definiert - sie liegen übrigens auf den Tasten der kleinen abgesetzten Zehnertastatur - ist es ganz nützlich, speziell bei Ablage größerer Strings, vorab den Speicherbereich von 46150-46280 mittels einer POKE-Schleife zu nullen. Ansonsten fressen die Zeiger und die nicht benötigten Strings gegebenenfalls 25% des Speicherbereichs auf.

Und noch ein kleiner Trick zum Abschluß: Auch die weiter hinten liegenden Strings sind mit Nullen besetzt. Verzichtet man daher auf eine Vielzahl von Strings, so kann man die dann nicht benutzten Zeiger mit POKE überschreiben und so noch ein paar Bytes mehr herausholen. Jedoch dürfen die entsprechenden Erweiterungszeichen dann auch wirklich nicht benutzt werden, da der CPC sonst ohne Verzeigerung wild im Speicher umherspringt.

Unser Wissen wäre nichts ohne Änderungsmöglichkeiten. Neben dem POKE-Befehl stellt uns das CPC-BASIC das Kommando KEY zur Verfügung. Definiert man mit

```
KEY 128,"ERWEITERUNGS-STRING"
```

einen neuen expansion character, so erscheint das Wort "ERWEITERUNGS-STRING" durch Druck auf die 0 im abgesetzten Zahlenfeld auf dem Bildschirm. Gleichzeitig können wir den String jetzt im Expansionspeicher ab 46150 lesen (bitte nachschauen!).

Mit dieser Methode ist es nun möglich, die Tastatur mit BASIC-Befehlen oder auch kompletten Ketten von Grafikzeichen zu belegen. Wollen wir beispielsweise ein größeres Grafiksymbol aus mehreren "normal großen" Zeichen zusammenfügen, so geht dies relativ einfach, indem wir einen Grafikzeichenstring auf die Tastatur legen, beispielsweise durch

```
KEY 128,CHR$(214)+CHR$(143)+CHR$(215)
```

Mit dieser Methode ist es dann auch möglich die normalerweise nicht verfügbaren Grafikzeichen im Bereich von 128 bis 159 dennoch darzustellen. Man definiert diese einfach auf den Erweiterungs-String mit derselben Nummer. Sie können dann, wie auch alle anderen Zeichen, auf die Tastatur gelegt werden. Mit diesem Trick arbeitet auch das oben beschriebene Zeichensatzprogramm. In Zeile 210 werden die *expansion characters* mit den Blockgrafiksymbolen belegt.

### 2.3.1.6 Hohe Schule der Umdefinition: neue Darstellungsmodes

Mit einer einfachen Umdefinition der Erweiterungszeichen sind unsere Möglichkeiten jedoch noch lange nicht erschöpft. Eine weitere Steigerung erreichen wir, indem wir zusätzlich zu den Befehlen KEY und KEY DEF auch noch das SYMBOL-Kommando benutzen und uns neue Zeichen definieren.

Damit ist es dann nicht nur möglich, die bisher nicht über Tastatur erreichbaren Blockgrafiksymbole (speziell in den Bereichen 128-159 und 240-255) zu erreichen, sondern wir können sogar mit mehreren Grafikmodes gleichzeitig auf dem Bildschirm arbeiten, was besonders in der Titelgestaltung neue Freiräume schafft.

Dies kostet allerdings ein wenig Speicher. Das Prinzip ist dabei folgendes: Definiert man mit dem SYMBOL-Kommando neue Zeichen in den freien Bereichen, so stellt der CPC diese statt der im normalen Firmwarezeichensatz vorhandenen Zeichen dar. Ist also eine Taste mit dem Zeichen Nummer 170 belegt und definieren wir dieses um, so wird nun auf Tastendruck das neue Zeichen dargestellt.

Neben der Definition der Erweiterungszeichen haben wir damit einen weiteren Weg zur Verfügung, mit dem wir die ansonsten unerreichbaren Grafiksymbole in den Bereichen 128-159 oder 240 folgende darstellen können. Wir definieren einfach Zeichen aus den freien Bereichen, zum Beispiel ab 169, mit SYMBOL um.

Soll zum Beispiel das Männchen von CHR$(250) auf diese Art durch die Tastatur aufrufbar sein, so gehen wir wie folgt vor: Zunächst geben wir den Speicher für die neuen Symbole frei. Dies erreichen wir mit SYMBOL AFTER. Nach `SYMBOL AFTER 169` können wir dann mit

```
SYMBOL169,&x00111000,&x0011100,&x00010010,
&x01111100,&x10010000,&x00101000,&x00100100,
&x00100010:POKE 46127,169
```

die Q-Taste in der CTRL-Ebene (Adresse 46127) mit dem neuen Symbol belegen. Die zur Definition notwendigen Reihenwerte entnimmt man dabei entweder dem Handbuch (bei existierenden Zeichen), oder man entwickelt sie mit dem Programm Zeichendefinition aus Kapitel 2.1.

Bei dieser Art der Belegung sind wir jedoch nicht auf ein Zeichen beschränkt. Benutzen wir die Expansion-Strings und definieren wir gleichzeitig unseren Zeichensatz neu, so ist es möglich, auch größere grafische Symbole durch Addition einzelner Zeichen zu erstellen, wie zum Beispiel eine Schachfigur, die 2x2 Zeichen groß ist.

Eine andere interessante Anwendung wäre, auf diese Weise die Beschränkung des CPC auf nur einen Mode zu umgehen. Der CPC schaltet nämlich auf relativ simple Art und Weise zwischen den einzelnen Modi um. Er verdoppelt einfach die Anzahl der in der Horizontalen dargestellten Bildpunkte. Diese Vorgehensweise können wir relativ einfach mit den *expansion characters* und dem SYMBOL-Kommando imitieren.

Schauen wir im Anhang III auf Seite 3 im Handbuch nach, so finden wir zum Beispiel für die Definition der obersten Punktreihe des A die folgende Binärfolge 00011000. Wir trennen diese nun in der Mitte durch und schreiben jede Ziffer zweimal. Wiederholen wir dasselbe für die restlichen sieben Reihen, so haben wir zwei neue Zeichen, die zusammengesetzt ein A von doppelter Breite ergeben. Für unser "A" würde dies folgendermaßen aussehen:

| normaler Buchstabe | verdoppelter Buchstabe linke Hälfte | rechte Hälfte |
|---|---|---|
| 00011000 | 00000011 | 11000000 |
| 00111100 | 00001111 | 11110000 |
| 01100110 | 00111100 | 00111100 |
| 01100110 | 00111100 | 00111100 |
| 01111110 | 00111111 | 11111100 |
| 01100110 | 00111100 | 00111100 |
| 01100110 | 00111100 | 00111100 |
| 00000000 | 00000000 | 00000000 |

Wir definieren dazu zwei Symbole mit SYMBOL auf die Zeichenwerte, wie wir sie analog zu obigem Bild erhalten. Das Zusammenfügen geschieht dann auf relativ einfache Weise, indem man einen Expansion-String auf

```
CHR$(linke Buchstabenhälfte)+CHR$(rechte
Buchstabenhälfte)
```

definiert. Der Rest ist Fleißarbeit. Das weiter unten abgedruckte Programm MULTIMODE definiert auf diese Art und Weise die CONTROL-Ebene der Tastatur neu mit doppelt so großen Großbuchstaben.

**Programmbeschreibung:**

Am Anfang steht ein Bandwurm von SYMBOL-Befehlen, mit denen die Halbzeichen definiert werden. Je zwei Zeilen definieren dabei einen Buchstaben. Die einzelnen Zeilenwerte werden dabei in binärer Form angegeben.

Bei der Eingabe sollten sie darauf achten, daß der Editor des Schneider Vornullen bei der Ausgabe unterdrückt. Die einzelnen Binärstrings sind also unterschiedlich lang. In jedem Fall müssen sie aber eine gerade Anzahl von Ziffern enthalten. Dies können Sie als Kontrolle für die richtige Eingabe benutzen.

Nachdem uns die einzelnen Buchstabenhälften nun zur Verfügung stehen, erfolgt die eigentliche Umdefinition. Hier spart uns unser internes Wissen nun eine Menge Arbeit. Wir benötigen nämlich nur noch 3 Zeilen, um unseren neuen Zeichensatz zu definieren. Hier zeigt sich eindeutig der Vorteil des Direktzugriffs mit POKE gegenüber der KEY DEF-Definition.

In Zeile 310 wird der Expansion-Speicher mit den zusammengefügten Halbsymbolen geladen; zuerst die Länge (2 characters) und dann die Codes für die Symbole. Die nachfolgende DATA-Zeile enthält die Tastaturcodes für das Alphabet in aufsteigender Reihenfolge beginnend mit A=69.

Dieser Tastaturcode wird in Zeile 330 gelesen und dann wird in der Tastaturübersetzungstabelle der Code für den entsprechenden Erweiterungsstring abgelegt. Die Stellung in der Tabelle ergibt sich dabei aus 46060 (CTRL-Tabellen-Anfang) + Tastencode. Nach einmaligem Durchlauf stehen uns dann die neuen Zeichen zur Verfügung.

Da der Trick modeunabhängig funktioniert, haben wir beim Arbeiten im Mode 0 nun auch noch einen neuen Supermode mit nur 10 Zeichen aber 64 eingenommenen Bildpunkten pro Zeichen zur Verfügung. Wem das immer noch nicht genügt, der kann ja nach derselben Methode einen Super-Supermode schaffen, indem er die Werte wiederum verdoppelt und dann vier Zeichen auf jeden Expansionstring legt.

Auch ist es möglich, auf diese Art Schachfiguren oder andere Spielsymbole zu definieren und dann via Tastatur sehr einfach Grafikbildschirme für ein Spiel zu erzeugen. Treibt man dieses Vorgehen auf die Spitze, so ist es möglich, eine oder auch mehrere Textzeilen untereinander durch einen einzigen Tastendruck auf den Schirm zu bringen. Da diese dann direkt in eine Befehlszeile übernommen werden können, ergibt sich beim Programmieren eine deutliche Zeitersparnis. Versuchen Sie doch einmal, eine etwas kompliziertere Spielfigur oder gar einen Teil eines Titelbildes auf diese Weise zu programmieren.

## Listing 3: *Programm MULTIMODE*

```
10 REM ****************
20 REM ** Multimode **
30 REM ****************
40 SYMBOL AFTER 169
50 SYMBOL 169,&X11,&X1111,&X111100,&X111100,&X111111,&X111100,&X1
11100,0
55 SYMBOL 170,&X11000000,&X11110000,&X111100,&X111100,&X11111100,
&X111100,&X111100,0
60 SYMBOL 171,&X11111111,&X111100,&X111100,&X111111,&X111100,&X11
1100,&X11111111,0
65 SYMBOL 172,&X11110000,&X111100,&X111100,&X11110000,&X111100,&X
111100,&X11110000,0
70 SYMBOL 173,&X1111,&X111100,&X11110000,&X1110000,&X11110000,&X1
11100,&X1111,0
75 SYMBOL 174,&X11110000,&X111100,0,0,0,&X111100,&X11110000,0
80 SYMBOL 175,&X11111111,&X111100,&X111100,&X111100,&X111100,&X11
1100,&X11111111,0
85 SYMBOL 176,&X11000000,&X11110000,&X111100,&X111100,&X111100,&X
11110000,&X11000000,0
90 SYMBOL 177,&X11111111,&X111100,&X111100,&X111111,&X111100,&X11
1100,&X11111111,0
95 SYMBOL 178,&X11111100,&X1100,&X11000000,&X11000000,&X11000000,
&X1100,&X11111100,0
100 SYMBOL 179,&X11111111,&X111100,&X111100,&X111111,&X111100,&X1
11100,&X11111111,0
105 SYMBOL 180,&X11111100,&X1100,&X11000000,&X11000000,&X11000000
,0,0,0
110 SYMBOL 181,&X1111,&X111100,&X11110000,&X11110000,&X11110000,&
X11110000,&X111111,0
115 SYMBOL 182,&X11110000,&X111100,0,0,&X11111100,&X111100,&X1111
1100,0
120 SYMBOL 183,&X111100,&X111100,&X111100,&X111111,&X111100,&X111
100,&X111100,0
125 SYMBOL 184,&X111100,&X111100,&X111100,&X11111100,&X111100,&X1
11100,&X111100,0
130 SYMBOL 185,&X111111,&X11,&X11,&X11,&X11,&X11,&X111111,0
135 SYMBOL 186,&X11111100,&X11000000,&X11000000,&X11000000,&X1100
0000,&X11000000,&X11111100,0
140 SYMBOL 187,&X11,0,0,0,&X11110000,&X11110000,&X111111,0
145 SYMBOL 188,&X11111100,&X11110000,&X11110000,&X11110000,&X1111
0000,&X11110000,&X11000000,0
150 SYMBOL 189,&X11111100,&X111100,&X111100,&X111111,&X111100,&X1
11100,&X11111100,0
155 SYMBOL 190,&X111100,&X111100,&X11110000,&X11000000,&X11110000
,&X111100,&X111100,0
160 SYMBOL 191,&X11111111,&X111100,&X111100,&X111100,&X111100,&X1
11100,&X11111111,0
165 SYMBOL 192,0,0,0,0,&X1100,&X111100,&X11111100,0
170 SYMBOL 193,&X11110000,&X11111100,255,255,&X11110011,&X1111000
0,&X11110000,0
175 SYMBOL 194,&X111100,&X11111100,&X11111100,&X11111100,&X111100
,&X111100,&X111100,0
180 SYMBOL 195,&X11110000,&X11111100,255,&X11110011,&X11110000,&X
11110000,&X11110000,0
185 SYMBOL 196,&X111100,&X111100,&X111100,&X11111100,&X11111100,&
X111100,&X111100,0
190 SYMBOL 197,&X1111,&X111100,&X11110000,&X11110000,&X11110000,&
X111100,&X1111,0
195 SYMBOL 198,&X11000000,&X11110000,&X111100,&X111100,&X111100,&
X11110000,&X11000000,0
200 SYMBOL 199,255,&X111100,&X111100,&X111111,&X111100,&X111100,2
55,0
205 SYMBOL 200,&X11110000,&X111100,&X111100,&X11000000,0,0,0,0
```

```
210 SYMBOL 201,&X1111,&X111100,&X11110000,&X11110000,&X11110011,&
X11110000,&X111111,0
215 SYMBOL 202,&X11000000,&X11110000,&X111100,&X111100,&X11001100
,&X11110000,&X111100,0
220 SYMBOL 203,&X11111111,&X111100,&X111100,&X111111,&X111100,&X1
11100,&X11111100,0
225 SYMBOL 204,&X11110000,&X111100,&X111100,&X11110000,&X11110000
,&X111100,&X111100,0
230 SYMBOL 205,&X1111,&X111100,&X111100,&X1111,0,&X111100,&X1111,
0
235 SYMBOL 206,&X11110000,&X111100,0,&X11110000,&X111100,&X111100
,&X11110000,0
240 SYMBOL 207,&X111111,&X110011,&X11,&X11,&X11,&X11,&X1111,0
245 SYMBOL 208,&X11111100,&X11001100,&X11000000,&X11000000,&X1100
0000,&X11000000,&X11110000,0
250 SYMBOL 209,&X111100,&X111100,&X111100,&X111100,&X111100,&X111
100,&X1111,0
255 SYMBOL 210,&X111100,&X111100,&X111100,&X111100,&X111100,&X111
100,&X11110000,0
260 SYMBOL 211,&X111100,&X111100,&X111100,&X111100,&X111100,&X111
1,&X11,0
265 SYMBOL 212,&X111100,&X111100,&X111100,&X111100,&X111100,&X111
10000,&X11000000,0
270 SYMBOL 213,&X11110000,&X11110000,&X11110000,&X11110011,255,&X
11111100,&X11110000,0
275 SYMBOL 214,&X111100,&X111100,&X111100,&X111100,&X11111100,&X1
1111100,&X111100,0
280 SYMBOL 215,&X11110000,&X111100,&X1111,&X1111,&X111100,&X11110
000,&X11110000,0
285 SYMBOL 216,&X111100,&X11110000,&X11000000,&X11000000,&X111100
00,&X111100,&X111100,0
290 SYMBOL 217,&X111100,&X111100,&X111100,&X1111,&X11,&X11,&X1111
,0
295 SYMBOL 218,&X111100,&X111100,&X111100,&X11110000,&X11000000,&
X11000000,&X11110000,0
300 SYMBOL 219,255,&X11110000,&X11000000,&X11,&X1111,&X111100,255
,0
305 SYMBOL 220,&X11111100,&X111100,&X11110000,&X11000000,&X1100,&
X111100,&X11111100,0
310 FOR i=0 TO 25:POKE 46150+3*i,2:POKE 46150+3*i+1,169+2*i:POKE
46150+3*i+2,170+2*i:NEXT
320 DATA 69,54,62,61,58,53,52,44,35,45,37,36,38,46,34,27,67,50,60
,51,42,55,59,63,43,71
330 FOR i=0 TO 25:READ a:POKE 46060+a,128+i:NEXT
```

### 2.3.2 Ein Malprogramm für die Blockgrafik

Mit unseren Hilfsprogrammen aus Kapitel 2.3.1 haben wir bereits einen Grundstock für das Arbeiten mit der Blockgrafik zur Verfügung. Wir können Blockgrafiksymbole durch Tastendruck auf den Bildschirm bringen und uns gegebenenfalls sogar für Sonderanwendungen einen chinesischen Zeichensatz definieren.

Mit zwei weiteren Punkten müssen wir uns aber noch beschäftigen. Zum einen ist die Ablage von Blockgrafikzeichen in einem String eine relativ aufwendige Sache, wenn man damit Titelgrafiken oder Diagramme definieren will. Außerdem muß bei der Definition einer Zeichenkette natürlich bekannt sein, aus welchen Zeichen diese bestehen soll. Für die Entwicklung neuer Bilder verfügen wir also noch nicht über ein geeignetes Werkzeug.

Dies wollen wir nun ändern. Dazu werden wir unser Tastaturdefinitionsprogramm zu einem Hauptprogramm erweitern, welches es uns ermöglichen soll, Titelgrafiken mit Hilfe der Tastatur komfortabel zu entwerfen. Besonderes Augenmerk wollen wir dabei der Abspeicherung von Blockgrafiken widmen.

#### 2.3.2.1 DESIGNER 464 - Eigenschaften und Programmstruktur

Wenn man ein größeres neues Programm entwickeln will, so ist man immer wieder mit einigen grundsätzlichen Fragen konfrontiert. Diese sollten vor dem ersten Eingeben beantwortet werden, um späteren Ärger zu vermeiden. Wir wollen uns an dem nun beschriebenen Blockgrafik-Entwicklungsprogramm mit dem Namen DESIGNER 464 dieses Vorgehen einmal erläutern.

Zunächst einmal muß man sich bei jedem neuen Programm natürlich als wichtigsten Punkt überlegen, was die zu entwickelnde Routine überhaupt leisten soll. Mit einer exakten Zielbeschreibung hat man meist schon die Hälfte des für ein Programm notwendigen Arbeitsaufwandes hinter sich.

Außerdem spart man sich bei dieser Vorgehensweise möglicherweise eine Menge Ärger, wenn später ein bereits bis an die Grenzen ausgereiztes Programmkonzept noch erweitert werden muß, und sich dann herausstellt, daß dies aufgrund einer mangelhaften Struktur nicht mehr möglich ist.

Dazu müssen wir festlegen, über welche Eigenschaften das neue Programm verfügen soll und in welcher Ausführung diese vorhanden sein müssen. Wir spezifizieren dazu den folgenden Forderungskatalog:

**Zieldefinition:**

Das Blockgrafik-Entwicklungsprogramm soll es uns ermöglichen, an einer beliebigen Stelle auf dem Bildschirm mit Hilfe der Tastatur Grafiksymbole in verschiedenen Farben zu setzen, oder Teile des Schirms zu löschen.

Die aktuelle Position soll dabei durch einen blinkenden Cursor veranschaulicht werden. Es muß jederzeit möglich sein, eine neue Bildschirmfarbe zu wählen oder die Tastatur neu zu definieren, um auf neue Grafiksymbole zurückgreifen zu können.

Daneben muß das Programm Routinen zum Abspeichern und Laden von Grafikbildern enthalten, wobei die Abspeicherung so auszuführen ist, daß die mit dem Designer entwickelten Bilder möglichst problemlos durch andere Programme dargestellt werden können.

Mit dieser sehr kompakten Beschreibung haben wir neben der Zieldefinition im engeren Sinn auch schon eine Reihe von weiteren Unterpunkten angerissen, die man normalerweise erst in einem zweiten Schritt behandelt.

**Menütechnik als Basis des Programms**

Es sind dies zuerst einmal die Frage nach der Programmiertechnik, das heißt nach dem grundsätzlichen Aufbau des Programms. Hier kann man mit mehreren Techniken arbeiten. Wenn das Programm mit relativ wenigen Bildschirmausgaben auskommt, so bietet sich die Verwendung eines einzigen Bildschirms an. Auf ihm erfolgen alle Ein- und Ausgaben.

Je größer die Anzahl der Funktionen und damit natürlich auch die Anzahl auszugebender Bildschirme jedoch wird, desto schwieriger ist dieses Verfahren. Schließlich ist es völlig unmöglich, beispielsweise aus 50 oder 60 Funktionen, die gleichzeitig auf dem Bildschirm dargestellt werden, die richtige auszuwählen, und dabei dennoch die Übersicht zu behalten.

In diesen Fällen hat sich die Menütechnik als das geeignete Programmierverfahren herausgestellt. Dabei wird von einem Hauptmenü in mehrere Untermenüs (beispielsweise Kassetten- und Diskettenoperationen, Suchroutinen, Darstellungen von Texten etc.) verzweigt, die dann wieder ihre eigenen spezifischen Unterfunktionen aufweisen.

Man hat somit in einem Unterprogramm immer nur eine spezifische, auf einen Problembereich ausgerichtete Anzahl von Funktionen zur Verfügung. Als Unterpunkt ist hierbei zu entscheiden, mit welchem

Bildschirmmodus gearbeitet werden soll und wie dementsprechend der Bildschirm zu organisieren ist.

Bei DESIGNER werden wir eine Kombination aus beiden Verfahren benutzen, die eigentliche Bilddarstellung erfolgt auf einem einzigen Bildschirm, der auch nicht verändert wird, außer natürlich durch den Benutzer bei der Zeicheneingabe.

Daneben ist es möglich, auf Tastendruck in ein Auswahlmenü zu verzweigen, das dann alle nicht laufend benötigten Funktionen, wie das Abspeichern und Sichern von Bildern oder Teilbildern, die Tastaturumdefinition oder die Farbwahl enthält.

**Datenspeicherung und Softwareschnittstellen**

Die Datensicherung stellt den zweiten wichtigen Unterpunkt dar. Vor der Programmentwicklung sollte man sich nämlich darüber im Klaren sein - speziell, wenn das Programm Dateien oder Informationen in größerem Maße benötigt - wie diese abgespeichert und natürlich auch wieder geladen werden sollen. Zwei Gesichtspunkte sind dabei besonders zu beachten:

- **die Speicher- und insbesondere Ladegeschwindigkeit.** Wenn das Einladen eines Grafikbildschirms mehrere Minuten in Anspruch nimmt, ist diese Speicherart nicht geeignet, um häufiger "vorfabrizierte" Grafikbildschirme zu laden.

- **die Schnittstellen zu anderen Programmen.** Bei unserem DESIGNER heißt dies, daß es nicht besonders sinnvoll ist, Grafikbildschirme um ihrer selbst willen zu entwickeln. Ein Grafikentwicklungsprogramm erreicht seinen eigentlichen Sinn erst im Zusammenspiel mit anderen Routinen. Zum Beispiel wenn es die Titelbilddarstellung für ein anderes Programm liefert, oder relativ komplexe Zeichnungen zur Veranschaulichung von Zusammenhängen in einem anderen Programm bereitstellt.

In beiden Fällen muß natürlich das übernehmende Programm in der Lage sein, die übergebenen Grafikdaten, sei es nun als Gesamtbildschirm oder in Einzelteilen zu übernehmen, und dann auch auf dem Monitor darzustellen. Es muß also eine Softwareschnittstelle zwischen beiden Programmen existieren. Bei DESIGNER werden wir dieses Problem mit Hilfe eines Ladeprogramms lösen. In jedes Programm, das von DESIGNER entwickelte Grafikbildschirme benutzen will, müssen dabei ein paar Be-

fehle eingebaut werden, wonach dann dieses Programm in der Lage ist, die übernommenen Daten auszugeben.

**Bedienmöglichkeiten**

Dieser Problembereich umfaßt nicht nur die Zuordnung der Tasten zu den verschiedenen Funktionen, das heißt die Frage, wo die verschiedenen Programmaufrufe liegen sollten, um eine möglichst gute Arbeit zu ermöglichen. Hier ist auch erst einmal zu klären, ob man überhaupt mit der Tastatur arbeitet.

Als Alternativen stehen hier zum Beispiel JOYSTICK und MAUS, aber auch - was hier nicht als direkt der Tastatur zugehörig betrachtet werden soll - die Cursortasten zur Verfügung. Deren Verwendung bietet sich immer dann an, wenn man wie in unserem Fall, auf einem relativ großen Feld an einem bestimmten Punkt zum Beispiel durch einen Cursor gesteuert, eine Veränderung auf dem Bildschirm ausführen will.

Bei DESIGNER werden wir uns mit dem Cursor auf dem Schirm bewegen. An der aktuellen Cursorposition ist dann die Eingabe von Grafik-Zeichen durch die Haupttastatur möglich.

### 2.3.2.2 Das Flußdiagramm

Neben diesen allgemeinen Unterpunkten haben wir mit der Zieldefinition auch schon die hauptsächlichen Funktionen unseres Programms festgelegt. Ein grobes Abbild, wie unser DESIGNER am Schluß aussehen wird, liefert uns nun das nebenstehende Flußdiagramm. Schauen wir uns die einzelnen Abschnitte etwas näher an.

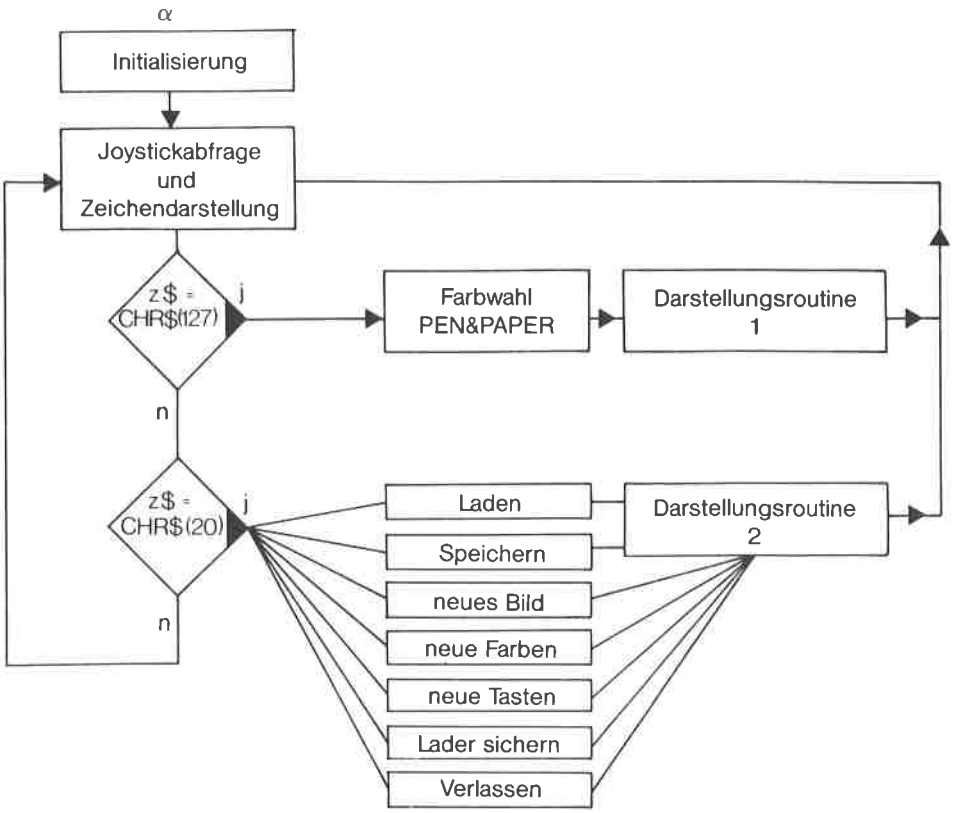



**Bild 2.2:** *Flußdiagramm Programm DESIGNER*

Den Anfang unseres Programms bildet eine Initialisierungsroutine, die die Farbregister setzt, auf den Bildschirmmodus umschaltet und ähnliche Anfangsdefinitionen vornimmt.

Nach dieser gelangen wir in eine Endlosschleife. Mit Hilfe der Cursortasten wird in dieser ein blinkendes Rechteck über den Bildschirm bewegt. An seiner aktuellen Position kann dann durch die Tastatur ein beliebiges Zeichen gesetzt werden. Will man ein ausgegebenes Symbol löschen oder ändern, so geschieht dies einfach durch Überschreiben, das heißt man tippt einen neuen Code über den alten.

Wichtig ist in diesem Zusammenhang, daß uns dabei der gesamte Bildschirm zur Verfügung steht, also 25 Zeilen mit, je nach Modus, 20 oder 40 Zeichen. Es wird also kein Platz für eine Statuszeile oder ähnliches benötigt.

Aus dieser Schleife gibt es zwei Ausgänge. Beide werden über Control-Codes erreicht. Durch Drücken der DEL-Taste kommen wir in eine Farbwahlroutine, die es uns ermöglicht, Vorder- und Hintergrundfarbe zu wechseln und damit unsere Zeichen in einer anderen Farbkombination auszugeben.

Die nachgeschaltete Darstellungsroutine 2 sorgt dabei dafür, daß der Bildschirm nach der Abfrage wieder in den Zustand vor der Farbwahl zurückkehrt. Die Farbabfrage benötigt nur die oberste Bildschirmzeile, der Restschirm bleibt erhalten, so daß auch nur diese nach der Routine wieder regeneriert werden muß.

Eine etwas größere Änderung erfordern dagegen die anderen Zusatzfunktionen. Man erreicht sie durch Drücken auf die Hochpfeiltaste, die dann den gesamten Bildschirm löscht und in ein Hauptmenü verzweigt. Die einzelnen Funktionen sind hierbei:

- Laden und Speichern des aktuellen Grafikbildschirms
- Initialisierung eines neuen Bildes (hier wird der Speicher gelöscht und auf die Ausgangsfarbzusammenstellung zurückgeschaltet).
- neue Farbdefinition (im Gegensatz zur Farbwahlroutine mit Hilfe einer Umdefinition der INK-Register).
- Neu-Definition der Tastatur, um auf andere Blockgrafiksymbole zurückgreifen zu können.
- Lader sichern. Wir hatten schon gesagt, daß der Aufruf eines Grafikbildschirms in einem anderen Programm mit Hilfe eines Laders erfolgen muß. Dieser wird automatisch auf Tastendruck von einer Unterroutine erzeugt und weggeschrieben.

Um diese etwas komplexeren Aufgaben erfüllen zu können, springt der CPC zunächst aus dem Darstellungsbildschirm in ein Funktionswahlmenü, welches dann zur gewünschten Funktion überleitet. Der Hauptbildschirm wird dabei gelöscht.

Die nachgeschaltete Darstellungsroutine 1 muß daher, nachdem die Unterroutine abgearbeitet wurde, den gesamten Bildschirm in dem Zustand wieder herstellen, der vor dem Einsprung in das Hauptmenü vorhanden

war. Da dieser Vorgang in BASIC programmiert zu lange dauern würde, wird hierbei ein kleines Maschinenprogramm eingesetzt.

#### 2.3.2.3 Die Datenspeicherung

Nun stellt sich natürlich sofort die Frage, auf welche Daten diese Maschinenroutine eigentlich zurückgreift. Die Abspeicherung der Farb- und auch Zeichencode-Information erfolgt beim DESIGNER auf eine sehr ungewöhnliche Art, jedenfalls für den CPC. Wer schon einen Homecomputer der ersten oder zweiten Generation besessen hat, wird sich jedoch an dieses Verfahren erinnern.

Beim Schneider wird der Bildschirminhalt Punkt für Punkt zusammen mit den Farben der einzelnen Punkte gespeichert. Wir kommen darauf im zweiten Teil unseren Buches noch zurück. Bei den früheren Homecomputern war es üblich, statt dessen die Zeichen mit ihrem Code zu speichern. Parallel dazu verfügte man dann meist noch über einen zweiten Speicher, den Farbspeicher, der die zugehörige Farbinformation aufnahm.

Die Vorteile dieser Abspeicherungsart liegen auf der Hand. Da im Mode 1 mit 40 Zeichen pro Zeile und 25 Zeilen der gesamte Schirm nur aus 1000 Zeichen besteht, benötigt man auch nur 1000 Bytes für die Ablage der Zeichencodes (und natürlich weitere 1000 Bytes für die zugehörige Farbinformation).

Als Vergleichsdaten hierzu einmal die Angaben für die hochauflösende Grafik beim Schneider: Hier werden insgesamt 16 K, also 16384 Bytes benötigt. Wenn man also einen normalen Bildschirm in hochauflösender Grafik beim Schneider speichert, so benötigt man auf Kassette für die Abspeicherung 8 Blocks.

Vor allem dann, wenn man mehrere dieser Bildschirme nacheinander einladen will, ist dies auch bei Verwendung der Floppy kein Vergnügen. Bei Spielen mag dies im Bereich der Titelgrafik gerade noch angehen. Wenn man aber ein Anwendungsprogramm schreiben will, das gegebenenfalls mehrmals Sachzusammenhänge durch Grafiken verdeutlichen soll, so ist dies ein Ding der Unmöglichkeit. Daher wurde dieser Abspeicherungsweg gewählt.

Zur Ablage des Zeichencodes und der Farbinformation benötigen wir natürlich nun zuerst einmal 2000 Bytes freien Speicherplatz. Diesen Speicherbedarf reservieren wir uns an der Obergrenze unseres Benutzerspeichers, indem wir mit MEMORY die Variable HIMEM herabschieben.

Die Speicheraufteilung im einzelnen gibt dann Bild 2.3 wieder. Ab Adresse 40000 bis einschließlich 40999 ist der Speicher den Zeichencodes vorbehalten, jeweils genau 1000 Bytes höher liegt die zugehörige Farbinformation.

Die Farbabspeicherung beinhaltet dabei sowohl die Vorder- als auch die Hintergrundfarbe. Dabei wird ein kleiner Trick angewandt. Zunächst zerlegen wir ein Byte des Farbspeichers in zwei Halbbytes oder auch NIBBLES. Das obere Halbbyte enthält dabei die linken, höherwertigen 4 Bits. Das untere beschränkt sich auf die rechten, niederwertigen 4 Bits.

Mit 4 Bits sind 16 verschiedene Kombinationen oder Zahlen von 0 bis 15 kodierbar beziehungsweise dekodierbar, was genau für die, im Mode 0 maximal benötigte Gesamtzahl von 16 Farben ausreicht. Wenn wir nun im unteren Halbbyte die Vordergrund (PEN)- und im oberen die Hintergrundfarbe (PAPER) ablegen, so haben wir in einer 8-Bit-Adresse gleichzeitig zwei Farbregister gespeichert und können dennoch für jedes Zeichen Vorder- und Hintergrundfarbe getrennt definieren.

Eine Besonderheit bietet dann noch die Speicherstelle 42001. Sie enthält die aktuelle Anzahl der Zeichen pro Zeile. Wenn wir also im Mode 0 arbeiten, so steht hier eine 20, im Mode 1 eine 40.

Diese Werte werden von der Darstellungsroutine 1 als Kriterium benutzt, um festzustellen, in welchem Mode ein Bild gesichert wurde. Ein Grafikbild wird nun gespeichert, indem man den kompletten Bereich von Adresse 40000 bis einschließlich 42010 als binäres File speichert.

Zur Darstellung benötigen wir natürlich nun noch die Darstellungsroutine 1. Diese wird durch das Blockgrafikentwicklungsprogramm oder den Lader jeweils ab Adresse 42010 nach oben abgelegt.

Diese Maschinenroutine liest jeweils ein Byte aus dem Farb- und aus dem Zeichenspeicher ein, setzt die entsprechenden Vorder- und Hintergrundfarben, und gibt dann das Zeichen an der entsprechenden Stelle auf dem Bildschirm aus.

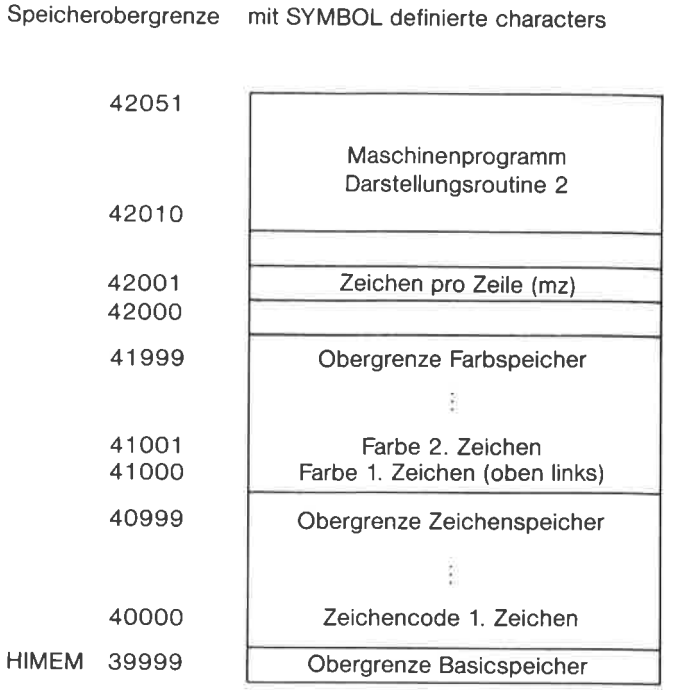



**Bild 2.3:** *Speicherbelegung des Programms DESIGNER*

Für den Umgang und die einfache Benutzung des Blockgrafik-Entwicklungsprogramms ist es nicht notwendig, das Programm in allen Einzelheiten nachzuvollziehen. Wer sich mehr für Maschinensprache und Maschinenspracheprogrammierung interessiert, findet dennoch nachfolgend ein Assemblerlisting und die zugehörige Programmbeschreibung.

**Programmbeschreibung Darstellungsroutine 1:**

Das Maschinenprogramm arbeitet mit zwei Zeigern, die in den Registerpaaren HL und DE abgelegt sind. HL enthält dabei die Adresse des darzustellenden Zeichencodes, DE die zugehörige Farbinformation. Nachdem diese beiden Zeiger auf die Anfangswerte (hex 9C40 = 40000

beziehungsweise hex A028 = 41000) gesetzt worden sind, beginnt die Darstellungsschleife.

Die Farbinformation wird in A eingelesen, danach werden die Registerpaare DE, HL und AF auf dem STACK gesichert. Es folgt der Aufruf einer Betriebssystemroutine

TXT SET PEN

Diese setzt die Schriftfarbe. Dabei ist zu berücksichtigen, daß TXT SET PEN nur die unteren vier Bits des Akkumulators berücksichtigt, so daß wir uns um die Paperfarbe, die ja in den oberen Bits abgelegt ist, keine Sorgen zu machen brauchen. Diese Bits werden ignoriert.

**Listing 4:** *Assemblerlisting Darstellungsroutine 1*

```
:HL=Adresse
:DE=Farbe
LD HL,&9C40                21 40 9C
LD DE,&A028                11 28 A0
<BEGINN>
LD A,(DE)                  1A
PUSH DE                    D5
PUSH HL                    E5
PUSH AF                    F5
CALL TXT SET PEN           CD 90 BB
POP AF                     F1
RRA                        1F
RRA                        1F
RRA                        1F
RRA                        1F
CALL TXT SET PAPER         CD 96 BB
POP HL                     E1
PUSH HL                    E5
LD A,(HL)                  7E
CALL TXT WR CHAR           CD 5D BB
POP HL                     E1
POP DE                     D1
INC HL                     23
INC DE                     13
LD A,&A0                   3E A0
CP H                       BC
JR NZ <BEGINN>             20 E1
LD A,&28                   3E 28
CP L                       BD
JR NZ <BEGINN>             20 DC
RET                        C9
```

Beim Aussprung aus der Farbdefinitionsroutine sind alle Registerpaare erstört, das heißt sie enthalten andere Werte. Dies war auch der Grund dafür, daß wir sie vor dem Ansprung der Routine gesichert haben. Nach

dem Rücksprung holen wir nun zunächst AF vom Stapel zurück und rotieren den Akkumulator viermal nach rechts.

Damit sind nun die anfangs im oberen Teil unseres Bytes gespeicherten 4 Bits für die Hintergrundfarbe nach unten gerutscht. Sie nehmen jetzt die unteren 4 Bits ein und damit sind wir in der Lage,

```
TXT SET PAPER
```

anzuspringen, eine Routine, die ebenfalls wie PEN auch die übergebenden Register zerstört und den Akkumulator für die Farbdefinition verwendet. Hier wird allerdings die Hintergrundfarbe gesetzt. Nach der Rückkehr aus dieser haben wir den Farbdefinitionsteil für das auszugebende Zeichen abgeschlossen. Als nächstes kommt nun der Zeichencode an die Reihe.

Wir holen HL vom Stapel zurück, geben es aber sofort im nächsten Schritt wieder mit PUSH HL zurück. Dies ist deswegen notwendig, weil auch die dritte nun anzuspringende Betriebssystemroutine

```
TXT WR CHAR
```

wiederum die Registerpaare und die Flags verändert. Vor dem Ansprung laden wir allerdings noch den Akkumulator mit dem Inhalt der durch HL adressierten Speicherstelle, das heißt mit dem aktuellen Zeichencode (vergleiche Bild 2.2). TXT WR CHAR benutzt die zuvor eingestellten PEN- und PAPER-Werte, so daß wir uns um unsere Zeichenausgabe keine weiteren Sorgen zu machen brauchen. Mit der Rückkehr aus dieser Routine ist dann ein Zeichen farbrichtig dargestellt.

Als nächstes holen wir uns nun HL und DE wieder vom Stapel zurück und erhöhen beide um 1. Sowohl der Farbzeiger wie auch die Adresse des Zeichencodes weisen nun auf das nächsthöhere Zeichen und wir könnten nun unsere Schleife ab Beginn für das nächste Zeichen wieder durchlaufen. Dies würde sich allerdings ewig, gegebenenfalls unendlich lange wiederholen, so daß wir nun noch eine Aussprungbedingung definieren müssen.

Dies ist jedoch relativ leicht möglich. Wir müssen nämlich nur überprüfen, ob sämtliche Zeichen ausgegeben worden sind. Wir schauen uns dazu an, ob HL bereits den Wert A028 erreicht hat. In diesem Fall wurde als letztes Zeichen das in der Adresse A027 beziehungsweise dezimal 40999 abgelegte Zeichen ausgegeben, womit der komplette Bildschirm dargestellt ist und der Rücksprung erfolgen kann.

Der Befehlssatz des Z80 beinhaltet kein Kommando, mit dem der Prozessor in der Lage wäre einen 16-Bit-Vergleich durchzuführen. Wir

müssen daher in zwei Stufen überprüfen, ob wir die Adresse A028 erreicht haben.

Zunächst vergleichen wir daher H mit hex A0. Wir laden das Register A mit A0 und führen dann einen Vergleich des Akkumulators mit H aus. Ist diese Bedingung schon nicht erfüllt, können wir sofort wieder nach <BEGINN> springen. Ansonsten laden wir den Akkumulator mit hexadezimal 28 und führen einen nochmaligen Vergleich mit dem Register L durch. Solange L kleiner als hex 28 ist, geht es wieder nach <BEGINN>. Im anderen Fall kehrt die Routine ins BASIC zurück.

Mit diesen Vorabinformationen sind wir nun in der Lage, unser Blockgrafikentwicklungsprogramm zu programmieren, beziehungsweise das fertige Programm zu verstehen. Zunächst einmal das LISTING.

## Listing 5: *Programm DESIGNER 464*

```
10 '**************************
20 '** Initialisierungsteil **
30 '**************************
40 BORDER 0
50 MODE 1
60 DIM f(15,2)
70 f(0,1)=0:f(0,2)=0
80 f(1,1)=24:f(1,2)=24
90 f(2,1)=6:f(2,2)=6
100 f(3,1)=2:f(3,2)=2
110 f(4,1)=21:f(4,2)=21
120 f(5,1)=15:f(5,2)=15
130 f(6,1)=7:f(6,2)=7
140 f(7,1)=4:f(7,2)=4
150 f(8,1)=27:f(8,2)=27
160 f(9,1)=18:f(9,2)=18
170 f(10,1)=8:f(10,2)=8
180 f(11,1)=17:f(11,2)=17
190 f(12,1)=9:f(12,2)=9
200 f(13,1)=11:f(13,2)=11
210 f(14,1)=26:f(14,2)=26
220 f(15,1)=24:f(15,2)=6
230 FOR i=0 TO 15
240 INK i,f(i,1),f(i,2)
250 NEXT i
260 DIM nr(26),z(6,26)
270 '**********************
280 '** Masch-PRG laden **
290 '**********************
300 MEMORY 39999
310 KEY DEF 24,1,20
320 DATA 21,40,9c,11,28,a0,1a,d5,e5,f5,cd,90,bb
330 DATA f1,1f,1f,1f,1f,cd,96,bb,e1,e5,7e,cd,5d
340 DATA bb,e1,d1,23,13,3e,a0,bc,20,e1,3e,28,bd
350 DATA 20,dc,c9,x
360 i=42010
370 READ a$:IF a$<>"x" THEN POKE i,VAL("&"+a$):i=i+1:GOTO 370
380 pe=1:pa=0
390 MODE 1:PEN 2:PAPER 0
400 '********************************
410 '** Initialisierungsspruenge **
420 '********************************
430 GOSUB 1750:GOSUB 470:GOSUB 630:GOSUB 1440:GOSUB 1080:GOTO 800
440 '**********************
450 '** Anfangsabfragen **
460 '**********************
470 PRINT:PRINT" Speicher loeschen"
480 FOR i=40000 TO 41000:POKE i,32:NEXT i
490 FOR i=41000 TO 42000:POKE i,0:NEXT i
500 CLS
510 LOCATE 10,3:PRINT"D E S I G N E R  4 6 4":PEN 1
520 PRINT:PRINT:PRINT
530 PRINT" In welchem Modus wollen Sie arbeiten ?"
540 PRINT
550 PRINT"   40-Zeichen pro Zeile=MODE 1   (1)"
560 PRINT"   20-Zeichen pro Zeile=MODE 0   (0)"
570 z$=INKEY$:IF z$<>"1" AND z$<>"0" THEN 570 ELSE MODE VAL(z$):m
z=20+20*VAL(z$)
580 IF z$="1" AND pe MOD 4=pa MOD 4 THEN pe=M1N(pe,pa)+1
590 RETURN
600 '***************
610 '** Farbwahl **
620 '***************
630 IF mz=20 THEN j=15 ELSE j=3
```

```
640 MODE (mz-20)/20
650 PRINT"Farbdefinition:":PRINT:PRINT CHR$(24):PRINT"Farbe 0"+CH
R$(24)
660 FOR i=1 TO j:PEN i:PRINT"Farbe";i:NEXT i
670 PRINT
680 PEN 1
690 PRINT"Farbe wechseln j/n?"
700 z$=LOWER$(INKEY$):IF z$="n" THEN 760 ELSE IF z$<>"j" THEN 700
710 INPUT"Welches Farbregister";reg
720 INPUT"Erste  Farbe";f(reg,1)
730 INPUT"Zweite Farbe";f(reg,2)
740 INK reg,f(reg,1),f(reg,2)
750 CLS:GOTO 650
760 RETURN
770 '************************
780 '** Joystick-Schleife **
790 '************************
800 z$=INKEY$:IF z$=CHR$(240) THEN y=y-1
810 IF z$=CHR$(241) THEN y=y+1
820 IF z$=CHR$(242) THEN x=x-1:IF x<1 THEN y=y-1:x=mz
830 IF z$=CHR$(243) THEN x=x+1:IF x>mz THEN y=y+1:x=1
840 x=MIN(MAX(x,1),mz):y=MIN(MAX(y,1),25)
850 LOCATE x,y:CALL &BB8A:FOR t=1 TO 20:NEXT t:CALL &BB8D
860 IF z$="" THEN 800
870 IF z$=CHR$(127) THEN GOSUB 1010:GOTO 800
880 IF z$=CHR$(20) THEN GOSUB 1210:GOSUB 1440:GOSUB 1080:GOTO 800
890 IF ASC(z$)>=240 OR ASC(z$)<32 THEN 800
900 IF z$=CHR$(16) THEN z$=" "
910 '************************
920 '** Zeichendarstellung **
930 '************************
940 PEN pe:PAPER pa
950 LOCATE x,y:PRINT z$;:POKE 39999+x+mz*(y-1),ASC(z$):POKE 40999
+x+mz*(y-1),pe+16*pa
960 x=x+1:IF x>mz THEN IF y<25 THEN y=y+1:x=1
970 GOTO 800
980 '**********************
990 '** Farbroutine PEN **
1000 '**********************
1010 WINDOW#0,1,mz,1,1:CLS:INPUT"PEN-Farbe";pe:pe=pe MOD 16
1020 INPUT"PAPER-Farbe";pa:pa=pa MOD 16
1030 PEN pe:PAPER pa
1040 GOSUB 1120:RETURN
1050 '*****************************
1060 '** Darstellungsroutine 1 **
1070 '*****************************
1080 CLS:CALL 42010:PEN pe:PAPER pa:RETURN
1090 '*****************************
1100 '** Darstellungsroutine 2 **
1110 '*****************************
1120 CLS:z$=""
1130 FOR i=1 TO mz:z$=z$+CHR$(14)+CHR$(PEEK(40999+i)ö16)+CHR$(15)
+CHR$(PEEK(40999+i))+CHR$(PEEK(39999+i))
1140 NEXT i
1150 WINDOW#0,1,mz,1,25:LOCATE 1,1:PRINT z$;
1160 PEN pe:PAPER pa
1170 z$="":RETURN
1180 '************************
1190 '** Laden und Speichern **
1200 '************************
1210 MODE 1:PEN 1:PAPER 0:CLS
1220 LOCATE 12,3:PRINT"Funktionswahl:"
1230 PRINT:PRINT"          Laden        (1)"
1240 PRINT"          Speichern    (2)"
1250 PRINT"          neues Bild   (3)"
```

```
1260 PRINT"           neue Farben  (4)"
1270 PRINT"           neue Tasten  (5)"
1280 PRINT"           Lader sichern(6)"
1290 PRINT"           Verlassen    (7)"
1300 z$=INKEY$:IF z$="7" THEN 1400
1310 IF z$="3" THEN GOSUB 470:GOTO 1400
1320 IF z$="4" THEN GOSUB 630:GOTO 1400
1330 IF z$="5" THEN GOSUB 1490:GOTO 1400
1340 IF z$="6" THEN GOSUB 1530:GOTO 1400
1350 IF z$<>"1" AND z$<>"2" THEN 1300
1360 PRINT:PRINT
1370 INPUT"Name des Files";n$
1380 IF z$="1" THEN LOAD n$,40000:mz=PEEK(42001):MODE (mz-20)/20:
GOSUB 1440:RETURN
1390 POKE 42001,mz:SAVE n$,b,40000,2002
1400 MODE (mz-20)/20:RETURN
1410 '*******************************
1420 '** Masch-PRG-Daten setzen **
1430 '*******************************
1440 IF mz=20 THEN POKE 42047,&34:POKE 42042,&9E ELSE POKE 42047,
&28:POKE 42042,&A0
1450 RETURN
1460 '************************
1470 '** Tastendefinition **
1480 '************************
1490 GOSUB 1750:RETURN
1500 '*********************
1510 '** Lader sichern **
1520 '*********************
1530 CLS:INPUT"Name des Laders";n$
1540 OPENOUT n$+".lad"
1550 PRINT#9,"10 ' ** Ladeprogramm "+n$+" **"
1560 PRINT#9,"20 memory 39999
1570 z$="25 ":FOR i=0 TO 15:z$=z$+"ink "+MID$(STR$(i),2)+","+MID$
(STR$(f(i,1)),2)+","+MID$(STR$(f(i,2)),2)+":":NEXT i
1580 PRINT#9,z$
1590 PRINT#9,"30 DATA 21,40,9c,11,28,a0,1a,d5,e5,f5,cd,90,bb
1600 PRINT#9,"40 DATA f1,1f,1f,1f,1f,cd,96,bb,e1,e5,7e,cd,5d
1610 PRINT#9,"50 DATA bb,e1,d1,23,13,3e,a0,bc,20,e1,3e,28,bd
1620 PRINT#9,"60 DATA 20,dc,c9,x
1630 PRINT#9,"70 i=42010
1640 PRINT#9,"80 READ a$:IF a$<>"+CHR$(34)+"x"+CHR$(34)+" THEN PO
KE i,VAL("+CHR$(34)+"&"+CHR$(34)+"+a$):i=i+1:GOTO 80
1650 PRINT#9,"50000 '** RUNTIME-MODUL **
1660 PRINT#9,"50010 load"+CHR$(34)+n$+CHR$(34)+",40000:mz=PEEK(42
001):MODE (mz-20)/20
1670 PRINT#9,"50020 IF mz=20 THEN POKE 42047,&34:POKE 42042,&9E E
LSE POKE 42047,&28:POKE 42042,&A0
1680 PRINT#9,"50030 call 42010
1690 PRINT#9,"50040 return
1700 CLOSEOUT
1710 RETURN
1720 REM **************
1730 REM ** key 464 **
1740 REM **************
1750 CLS
1760 LOCATE 10,3:PRINT"D E S I G N E R  4 6 4":PEN 1
1770 LOCATE 12,5:PRINT"Tastaturdefinition"
1780 PRINT:PRINT:PRINT " Linien / Blockgrafik 1              (1)"
1790 PRINT" Linien / Blockgrafik 2                (2)"
1800 PRINT" Blockgrafik 3 + Sonderzeichen         (3)"
1810 PRINT" griechisches Alphabet+Sonderzeichen   (4)"
1820 PRINT" Grossbuchstaben                       (5)"
1830 PRINT" Kleinbuchstaben                       (6)"
1840 DATA 67,59,58,50,51,43,42,35,34,27,69,60,61
1850 DATA 53,52,44,45,37,36,71,63,62,55,54,46,38
```

```
1860 RESTORE 1840
1870 FOR i=1 TO 26:READ nr(i):NEXT i
1880 FOR i=128 TO 159:KEY i,CHR$(i):NEXT
1890 PRINT" Tastaturbelegung in 'NORMAL'-Ebene ";
1900 INPUT n
1910 IF n>0 AND n<7 THEN 1940
1920 PRINT"Falsche Eingabe !!!"
1930 FOR i=1 TO 1500:NEXT i:GOTO 1900
1940 PRINT" Tastaturbelegung  in 'SHIFT'-Ebene ";
1950 INPUT s
1960 IF s<1 OR s>6 THEN 1970 ELSE 1990
1970 PRINT"Falsche Eingabe !!!"
1980 FOR i=1 TO 1500:NEXT i:GOTO 1950
1990 PRINT" Tastaturbelegung  in  'CTRL'-Ebene ";
2000 INPUT c
2010 IF c<1 OR c>6 THEN 2020 ELSE 2060
2020 PRINT"Falsche Eingabe !!!"
2030 FOR i=1 TO 1500:NEXT i:GOTO 1990
2040 DATA 129,130,131,132,133,134,135,136,137,138,139,140,141
2050 DATA 142,143,144,145,146,147,148,149,150,151,152,153,156
2060 DATA 192,193,194,195,196,197,198,199,200,201,202,203,204
2070 DATA 205,206,207,208,209,210,211,212,213,214,215,216,217
2080 DATA 218,219,220,221,222,223,226,160,161,162,163,164,165
2090 DATA 166,167,168,169,170,171,172,173,174,175,152,236,237
2100 DATA 176,177,178,179,180,181,182,183,184,185,188,189,190
2110 DATA 191,203,226,227,228,229,230,231,232,233,234,235,236
2120 DATA 81,87,69,82,84,89,85,73,79,80,65,83,68
2130 DATA 70,71,72,74,75,76,90,88,67,86,66,78,77
2140 DATA 113,119,101,114,116,121,117,105,111,112,97,115,100
2150 DATA 102,103,104,106,107,108,122,120,99,118,98,110,109
2160 FOR i=1 TO 6:FOR j=1 TO 26:READ z(i,j):NEXT j,i
2170 FOR i=1 TO 26:KEY DEF nr(i),1,z(n,i),z(s,i),z(c,i)
2180 NEXT i
2190 CLS
2200 PRINT" Normal : ";:ON n GOSUB 2290,2310,2330,2350,2370,2390
2210 PRINT" SHIFT  : ";:ON s GOSUB 2290,2310,2330,2350,2370,2390
2220 PRINT" CONTROL: ";:ON c GOSUB 2290,2310,2330,2350,2370,2390
2230 PRINT
2240 PRINT" O.K. j/n ?"
2250 Z$=INKEY$:IF LOWER$(Z$)<>"j" AND LOWER$(z$)<>"n" THEN 2250
2260 IF LOWER$(z$)="n" THEN 1750
2270 PRINT:PRINT
2280 RETURN
2290 RESTORE 2040:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2300 RETURN
2310 RESTORE 2060:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2320 RETURN
2330 RESTORE 2080:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2340 RETURN
2350 RESTORE 2100:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2360 RETURN
2370 RESTORE 2120:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2380 RETURN
2390 RESTORE 2140:z$="":FOR i=1 TO 26:READ a:z$=z$+CHR$(a):NEXT:P
RINT z$
2400 RETURN
```

**Programmbeschreibung:**

Bei der nun folgenden detaillierten Beschreibung der einzelnen Programmfunktionen und BASIC-Zeilen sollten Sie immer das Flußdiagramm in Bild 2.1 im Hinterkopf behalten. Wir werden dieses als Basis für die nun folgenden Erklärungen benutzen.

Am Anfang unseres BASIC-Programms stoßen wir auf den Initialisierungsteil. Es werden nacheinander die Rahmenfarbe, der Bildschirmdarstellungsmodus und die Farbregister festgelegt. Je INK werden dabei zwei Register benötigt, um auch blinkende Farben abspeichern zu können.

Alle INK-Register sind dabei in dem ARRAY F(15,2) abgespeichert. Der erste Index bestimmt dabei das zu definierende INK-Register, der zweite Index gibt an, ob es sich um die erste oder zweite zu definierende Farbe handelt. Nachdem das ARRAY mit den Ausgangswerten geladen wurde, werden dann die INK-Register auf diese Farbwerte definiert.

Als nächster Schritt folgt das Einladen unseres Maschinenprogramms in den Speicherbereich ab Adresse 42010. Dazu wird mit MEMORY die Speicherobergrenze auf 39999 herabgesetzt, worauf dann das Maschinenprogramm problemlos und sicher gePOKEt werden kann.

Der nun folgende Teil lautet "Initialisierungssprünge". Hier werden nacheinander die eigentlich zu den Sonderfunktionen gehörenden Programmteile zur Tasten-, Farb- und Modusdefinition aufgerufen, bevor dann der eigentliche Einsprung in unsere endlose Tastaturabfrageschleife erfolgt.

Wir treffen auf diese ab Zeile 830. Hier stoßen wir zunächst auf die eigentliche Cursorbewegungsroutine. In Abhängigkeit von Z$ wird der Cursor entsprechend den Tastencodes von 240 bis 243 bewegt. Das dabei angewandte Prinzip kennen wir ja schon von unserem Programm "Zeichendefinition". Der einzige Unterschied besteht dabei darin, daß mittels INKEY$ und nicht mit der Tastaturabfrage mit INKEY gearbeitet wird.

In Zeile 880 wird dann der Cursor an die aktuelle Bildschirmposition gesetzt. Dies geschieht mit einer Maschinenroutine, die an der HEX-Adresse BB8A ihren Ansprungpunkt hat. Sie setzt den Cursor. Nach einer kurzen Verzögerung durch eine Zeitschleife wird dieser dann mit CALL &BB8D wieder gelöscht.

In jedem Fall folgt danach wieder der Rücksprung nach Zeile 800, gegebenenfalls allerdings über einige Umwege. Der einfachste Fall ergibt sich dabei, wenn kein Zeichen eingegeben wurde. In diesem Fall springt

der CPC sofort wieder an den Schleifenbeginn. Ebenso falls ungültige Zeichen, das heißt Zeichen größer als 240 oder kleiner als 32 benutzt wurden.

Wurde dagegen ein gültiges Zeichen eingegeben, so läuft das Programm weiter in den Teil Zeichendarstellung. Hier werden zunächst PEN und PAPER auf die Variablen "pe" beziehungsweise "pa" gesetzt, die immer den aktuellen PEN- beziehungsweise PAPER-Wert enthalten. Danach wird das gerade eingegebene Zeichen auf dem Bildschirm ausgedruckt.

Gleichzeitig erfolgen allerdings zwei relativ kompliziert aussehende POKE-Operationen. Das erste Kommando setzt dabei den Zeichencode. Wir können diesen relativ einfach mit ASC(z$) bestimmen. Mit Hilfe von x und y, die auf dem Bildschirm die Spalte und Zeilenzahl enthalten, können wir auch problemlos die zu setzende Position im Zeichen- beziehungsweise Farbspeicher errechnen.

Das einzige Problem stellt nun eigentlich bei unserer Art der Abspeicherung die Bestimmung des Farbwertes dar, da wir hier ja mit den zwei Halbbytes konfrontiert werden. Allerdings ist dies auch kein größeres Problem. Wir brauchen nämlich einfach nur den Wert für die Hintergrundfarbe mit 16 zu multiplizieren und dazu den aktuellen PEN zu addieren und schon haben wir auch unseren Farbwert in der gewünschten Abspeicherung.

Nach der Überprüfung, ob mit der Zeichenausgabe der rechte Rand des Bildschirms erreicht wurde, kehrt dann das Zeichendarstellungsprogramm in die Hauptabfrageschleife zurück.

Drücken wir statt der Cursorsteuerungstasten oder einer Zeicheneingabetaste auf DEL so springt der CPC in die Farbroutine ab 1040. Aussprungzeile ist dabei Zeile 900, in der überprüft wird, ob die DEL-Taste gedrückt wurde (CHR$(127)).

Die Farbroutine beschränkt zunächst das Ausgabe-WINDOW auf eine Zeile und zwar auf die oberste Bildschirmzeile. Dies ist notwendig, um sicherzustellen, daß der Bildschirm nur in diesem Bereich geändert wird. Wir haben nämlich zu diesem Zeitpunkt, das heißt beim Aufruf der Farbroutine immer noch das zu entwickelnde Bild auf dem Schirm. Ein unkoordiniertes "Herumirren" auf dem Bildschirm, gegebenenfalls durch Fehlermeldungen etc. verursacht, würde den kompletten Bildschirminhalt zerstören.

Dieses Verfahren hat darüber hinaus den Vorteil, daß man problemlos und schnell während der Bildentwicklung Farbänderungen durchführen

kann. Mittels INPUT werden die Vorder- und Hintergrundfarbwerte ermittelt. Es folgt ein Sprung nach Zeile 1150.

Dies ist die Darstellungsroutine 2, die nur die oberste Bildschirmzeile ersetzt. Dazu wird aus den ersten 20 beziehungsweise 40 Farb- und Zeichencodewerten (je nach Mode) ein STRING zusammengebaut und dieser dann in der obersten Bildschirmzeile ausgegeben. Er überdeckt damit die gerade für die Farbabfrage ausgegebenen Texte und versetzt unseren Bildschirm wieder in den Arbeitszustand zurück.

Nicht ganz so einfach läuft der Prozeß ab, wenn wir auf die <Hochpfeiltaste> drücken. Am Anfang unseres Programms wurde diese auf CHR$(20) definiert (Zeile 340). Als erster Schritt erfolgt beim Druck auf diese Taste der Sprung in das Hauptfunktionswahl-Menü ab Zeile 1240. Die einzelnen Funktionen haben wir schon kennengelernt, so daß wir auf sie nur noch ansatzweise und bei einigen Spezialfällen eingehen wollen.

Die Funktionen 1 und 2 sind relativ einfach zu erklären. Hier findet einfach ein Laden beziehungsweise ein Speichern des Adreßbereichs von Adresse 40000 bis 42010 statt. Mit Funktion 3 "neues Bild" erreichen wir im Prinzip einen Neustart. Der Speicher wird wieder gelöscht und wir gelangen zurück in die Modusabfrage ab Zeile 500.

Neue Farben setzen wir mit Funktion 4, die uns zurück in die Definition für die INK-Register bringt. Funktion 5 "Tastendefinition" kennen wir schon als selbständiges Programm. Die Unterschiede zum Programm KEY werden Ihnen bei einem kurzen Vergleich relativ schnell auffallen.

**Das Ladeprogramm**

Besondere Beachtung verdient eigentlich nur noch der Funktionsteil Lader sichern ab Zeile 1560. Wir haben schon gesagt, daß wir zur Darstellung unserer Grafikbildschirme durch ein anderes BASIC-Programm ein Ladeprogramm benötigen. Dieses muß drei Funktionen erfüllen:

Zum ersten muß es die Speicherkonfiguration für die Abspeicherung der Grafikbildschirme und entsprechend natürlich auch für das Laden derselben herstellen. Dazu genügt ein einfacher BASIC-Befehl, nämlich MEMORY 39999.

Daneben, als zweiten Schritt muß das Ladeprogramm die von dem Grafikbildschirm benötigten Farbwerte in die INK-Register schreiben, damit die Darstellung auch in der richtigen Farbe erfolgen kann.

Als dritten Schritt muß unser Ausgabemaschinenprogramm, das heißt die Darstellungsroutine 1, natürlich auch in einem neu zu entwickelnden BASIC-Programm vorhanden sein, damit überhaupt eine Ausgabe möglich ist. Der Aufruf erfolgt dann jedesmal, wie schon vom DESIGNER gewohnt (Zeile 1110) mit CALL 42010.

Bei der Programmierung dieses Funktionsteils kommt uns nun eine Eigenschaft des CPC zu Gute. Der Schneider ist nämlich in der Lage, Programme, die als ASCII-File abgelegt wurden, als Programm wieder einzuladen und dann auch als Programm auszuführen.

ASCII-Files kennen Sie alle. Wenn Sie schon irgendwann einmal bei dem Schneider mit OPENOUT eine Datei eröffnet haben, um Text einzugeben, hatten Sie mit dieser Art der Datenabspeicherung zu tun. Bei jeder Art von Datenspeicherung auf dem CPC werden nämlich die Daten im ASCII-Format abgelegt, egal ob es sich dabei nun um STRINGs oder um numerische Variable handelt.

Damit ist auch das Wesentliche bereits gesagt. Wir brauchen nämlich nun nur STRINGs, die unsere Programmzeilen enthalten, mit OPENOUT wegzuschreiben, und sie dann wieder mit LOAD als Programm einzuladen. In diesem Fall erkennt der CPC die Strings als Zeilen eines neuen BASIC-Programms, setzt diese in seine BASIC-Befehle um und ist dann auch in der Lage, das so entstandene Programm auszuführen.

Wie dies rein technisch vonstatten geht, sehen Sie ab Zeile 1580. Mit PRINT#9 werden nacheinander die einzelnen BASIC-Zeilen unseres Ladeprogramms in die Ausgabedatei geschrieben. Besonders hervorzuheben sind hierbei die Zeilen 25 und 50010 des Ladeprogramms.

In Zeile 25 werden die INK-Register definiert. Sie können hier sehen, wie es auch möglich ist, ein Programm in einer Ausgabedatei an vom Benutzer eingegebene Werte anzupassen, denn die Farbangaben sind ja keineswegs fix, sondern können vom Benutzer laufend geändert werden. Die aktuell zum Zeitpunkt der Abspeicherung vorhandene Farbdefinition soll aber in das Ladeprogramm übernommen werden. Der Lader muß also die INK-Register auf die aktuellen Werte des ARRAYS F(15,2) setzen.

Die Lösung unseres kleinen Problems finden Sie in Zeile 1600. Hier wird ein relativ komplexer String für alle 16 INK-Register zusammengebaut. Dazu benutzen wir die STR$-Funktion, welche es uns ermöglicht, einen Zahlenwert in einen String zu konvertieren. Diese Strings können wir dann natürlich problemlos auch noch mit eingefügten Kommas und Doppelpunkten addieren und erhalten so eine Summe von INK-Befehlen, die

nur noch eine vorangestellte Zeilennummer brauchen, um als BASIC-Zeile zu wirken.

In Zeile 50010 sehen Sie wie man das Laden von einem eingegebenen Benutzer-STRING abhängig machen kann. Unser Funktionsteil "Lader Sichern" soll nämlich nicht nur für immer feststehendes Ladeprogramm gelten, sondern für beliebige. Wir müssen also zwischen mehreren Namen unterscheiden können, speziell, wenn wir mit der Diskette arbeiten.

Dies erreichen wir, indem wir den LOAD-Befehl nicht zu schnell ausführen, sondern ihm als Namen eine Benutzereingabe, hier in der Variablen n$ gespeichert, mit auf den Weg geben. Damit ist auch dieses Problem gelöst. Als Kontrollauszug finden Sie nachstehend ein probeweise gesichertes Ladeprogramm. Damit können Sie überprüfen, ob Sie den Laderteil richtig eingegeben haben.

Es wurde ein Ladeprogramm in der Ausgangsfarbzusammenstellung gesichert. Sie können dies erreichen, indem Sie sofort nach dem Durchlauf der Anfangsinitialisierung und Druck auf die Hochpfeiltaste die Funktion 6 aufrufen. Der Rest geschieht automatisch, von ein paar Tastendrücken Ihrerseits für PLAY und REC einmal abgesehen.

Arbeiten Sie nun ruhig einmal mit dem Programm und versuchen Sie, ein paar interessante Titelbilder, Titelgrafiken oder Diagramme zu entwerfen und abzuspeichern, und sehen Sie sich dabei an, wie das Programm läuft.

## Listing 6: *Ladeprogramm*

```
10 ' ** Ladeprogramm willi **
20 MEMORY 39999
25 INK 0,0,0:INK 1,24,24:INK 2,6,6:INK 3,2,2:INK 4,21,21:INK 5,15
,15:INK 6,7,7:INK 7,4,4:INK 8,27,27:INK 9,18,18:INK 10,8,8:INK 11
,1
7,17:INK 12,9,9:INK 13,11,11:INK 14,26,26:INK 15,24,6:
30 DATA 21,40,9c,11,28,a0,1a,d5,e5,f5,cd,90,bb
40 DATA f1,1f,1f,1f,1f,cd,96,bb,e1,e5,7e,cd,5d
50 DATA bb,e1,d1,23,13,3e,a0,bc,20,e1,3e,28,bd
60 DATA 20,dc,c9,x
70 i=42010
80 READ a$:IF a$<>"x" THEN POKE i,VAL("&"+a$):i=i+1:GOTO 80
50000 '** RUNTIME-MODUL **
50010 LOAD"willi",40000:mz=PEEK(42001):MODE (mz-20)/20
50020 IF mz=20 THEN POKE 42047,&34:POKE 42042,&9E ELSE POKE 42047
,&28:POKE 42042,&A0
50030 CALL 42010
50040 RETURN
```

## 2.4 Geheimcodes geknackt, den Steuerzeichen auf der Spur

Mit unserem Blockgrafikentwicklungsprogramm aus dem letzten Unterkapitel haben wir jetzt einen gewissen Abschluß erreicht. Wir haben eigentlich alle Gebiete behandelt, die im Reich der Blockgrafik von Bedeutung sind. Oder vielleicht sollten wir besser sagen: fast alle. Denn ein Kapitel steht uns noch bevor. Es handelt sich dabei um die Steuerzeichen, jene Characters, die mit Codes kleiner als 32 aufgerufen werden.

Wenn man ein solches Zeichen mit PRINT oder PRINT CHR$ anspricht, so wird es nicht auf dem Schirm ausgegeben, sondern der Computer führt eine Steuerfunktion aus. Er löscht beispielsweise den Schirm, oder bewegt den Cursor. Dennoch können wir diese Zeichen auch ausdrucken.

Jedem Steuercode ist nämlich ein Grafikzeichen zugeordnet. Um die unsichtbaren Zeichen sichtbar zu machen dient dabei der Code CHR$(1).Er ist vorab einzugeben und enttarnt dann das nachstehende Steuerzeichen. Die folgende Schleife liefert uns die grafischen Äquivalente aller Steuercodes.

```
FOR i=0 TO 32:PRINT CHR$(1)+CHR$(i):NEXT i
```

Die Systemsteuerzeichen gehören wie der Aufruf von Systemroutinen mit dem CALL-Befehl, mit denen wir uns später noch befassen werden, in den Bereich des Betriebssystems. Die Unterschiede liegen in der Funktion und der Art des Aufrufs dieser Routinen. Die Ausgabesteuerzeichen haben im wesentlichen mit dem Bildschirm zu tun, während Betriebssystem-Routinen für fast alle Anwendungszwecke verfügbar sind.

Während die Betriebssystem-Routinen mit dem CALL-Befehl als Maschinenunterprogramme aufgerufen werden, werden die Ausgabesteuerzeichen einfach mit dem PRINT-Befehl ausgegeben. Stößt der CPC in einem Ausgabetext auf ein solches Steuerzeichen, so druckt er nicht dieses Steuerzeichen, sondern geht in die entsprechende Routine des Betriebssystems. Tippt man beispielsweise

```
PRINT "<CURSOR RUNTER>"   =   PRINT"<CTRL+J>"
```

so bewegt sich der Cursor eine ganze Zeile nach unten. Eine nachfolgende Ausgabe erfolgt also eine Zeile niedriger als normal. Die meisten der Ausgabesteuerzeichen sind über Tastatur verfügbar und zwar in der CONTROL-Ebene. Ein Beispiel:

- Drücken von <G> bringt ein kleines g auf den Schirm.

- Drücken von <SHIFT G> erzeugt ein großes G.
- Drücken von <CTRL G> bringt dagegen ein Glockensymbol auf den Bildschirm.

Dieses Zeichen ist für die Systemsteuerzeichen untypisch, da es mit der Bildschirmgestaltung nichts zu tun hat. Dafür hat es den Vorteil, einen eindeutigen Effekt zu erzeugen. Gibt man nämlich

```
PRINT "<CTRL G>"
```

ein, so läßt der CPC einen kurzen Piepser erklingen, vorausgesetzt, man hat den Lautstärkeregler aufgedreht. Da es sich bei den Steuercodes trotz aller Besonderheiten um Zeichen handelt, können wir die damit verbundenen Funktionen aber nicht nur mit PRINT"<Steuerzeichen>", sondern auch über den Umweg mit CHR$ erreichen. CHR$(7) entspricht dabei unserem <CTRL G> und

```
PRINT CHR$(7)
```

erzeugt daher denselben Effekt.

Die Zahl in der CHR$-Funktion ist dabei mit der Nummer des Buchstabens in der CONTROL-Funktion im Alphabet identisch. G ist der siebte Buchstabe im Alphabet und somit liefert CHR$(7) den entsprechenden Code. Analog dazu können wir das Drücken von <CTRL> und <F> durch ein CHR$(6) ersetzen und so weiter. Jedoch ist dabei zu beachten, daß <CTRL C> doppelt belegt ist. Es gibt abwechselnd die Character-Codes 2 und 3 aus.

Somit ist dann <CTRL A> dem CHR$(0) und <CTRL B> dem CHR$(1) äquivalent. Doch schauen wir uns nun einmal die Systemsteuerzeichen im einzelnen an. Beim Ausprobieren und Kennenlernen der Steuerzeichen sollten Sie dabei möglichst kein Programm mehr im Speicher haben, da der CPC sich bei mehrmaligen Fehlbedienungen gelegentlich abstürzt. Sie bekommen ihn zwar dann mit

```
<CTRL> <SHIFT> <ESC>
```

oder auch durch einige andere Tricks wieder in Betrieb. Ein im Speicher befindliches Programm ist aber in jedem Fall verloren.

Insgesamt verfügt der CPC über 32 Steuerbefehle. Einige sind von geringerem Interesse, da sie nur ein Äquivalent zu einfacher anzuwendenden BASIC-Kommandos darstellen. Es gibt jedoch auch eine Reihe von Codes, die wirklich ungeahnte Effekte erzeugen.

Wenn wir von unten beginnen, so stellt CHR$(4) nach CHR$(1) den ersten sinnvoll anwendbaren Steuercode dar. CHR$(4) hat denselben Effekt, wie das MODE-Kommando. Mit einem zweiten Character geben wir dabei den neuen Bildschirmmodus an.

Hierzu zunächst ein paar Worte. Wenn bei einem Systemsteuerzeichen noch Parameter benötigt werden, so müssen diese auch mit der CHR$-Funktion mitgeliefert werden. Es muß dabei das Zeichen übergeben werden, dessen ASCII-Code mit der zu übergebenden Zahl übereinstimmt. Soll der CPC also den Wert 65 erkennen, so muß als zweite Angabe CHR$(65), ein großes "A", mitgegeben werden. Die Übergabe der Parameter erfolgt dann, indem man die CHR$-Funktionen zu einem Gesamtstring addiert, beispielsweise

```
PRINT CHR$(4)+CHR$(2)
```

Diese Zeichenkette würde eine Umschaltung in den MODE 2 beinhalten. Bei CHR$(4) gibt es jedoch auch eine "Default"-Bedingung. Wird kein Parameter spezifiziert, so setzt der Schneider selbst den Wert 1 ein. Ein einfaches

```
PRINT CHR$(4)
```

oder

```
PRINT "<CTRL D>"
```

bewirkt daher ebenfalls ein Umschalten in den MODE 1 und ist damit dasselbe, wie ein PRINT CHR$(4)+CHR$(1).

Mit CHR$(5) kommen wir nun zu einem Zeichen, das schon etwas besonderes leistet. Wie Sie vielleicht wissen, ist es mit dem CPC möglich, die Zeichenausgabe durch den Grafik-Cursor zu dirigieren. Dies wird durch den TAG-Befehl bewirkt. Näheres zu diesem Kommando und seinen Anwendungsmöglichkeiten finden Sie in Kapitel 5.

Wenn man nach TAG ein PRINT-Kommando ausführt, so wird die dadurch definierte Zeichenkette nicht wie gewohnt an der Position ausgedruckt, an der sich der Text-Cursor befindet, sondern der Grafik-Cursor gibt an, wo das Zeichen dargestellt werden soll. Maßgeblich ist dabei der linke obere Eckpunkt des auszugebenden Symbols. Dieser ist mit der (Grafik-) Cursorposition identisch.

Mit CHR$(5) können wir diese Art der Ausgabe nun für ein Zeichen erheblich einfacher darstellen. Wenn wir

```
PRINT CHR$(5)+CHR$(65)
```

ausdrucken, so wird ein großes "A" an der Stelle auf dem Schirm ausgedruckt, wo sich der Grafik-Cursor gerade befindet. Der Vorteil dieser Ausgabe liegt darin, daß wir auf das Rückschalten in den Normal-Modus (Text-Cursor-Ausgabe) verzichten können. Wenn man also nur ein Zeichen oder einen BASIC-Sprite (mehr dazu gleich) bewegen will und gleichzeitig noch anderer Text dargestellt werden soll, so empfiehlt sich diese Darstellungsart.

Die Wirkung von CHR$(7) haben wir schon kennengelernt. Wir kommen nun zu einem anderen Bereich, der bei ehemaligen Commodore-Benutzern das berühmte Erinnnerungs-Klicken auslösen dürfte, den Cursorsteuerzeichen.

### 2.4.1 Die Cursorsteuerzeichen

PRINT CHR$(8) oder PRINT "<CTRL H>" bewegt den Cursor um eine Stelle nach links. Mit CHR$(9) oder <CTRL> und gleichzeitigem Drücken von <I> erreicht man dasselbe nach rechts.

Geben wir PRINT CHR$(10) oder PRINT "<CTRL J>" ein, so geht der Cursor eine Zeile nach unten. Mit CHR$(11) oder <CTRL K> bewegt er sich in der Gegenrichtung.

Die letzten beiden Codes bewegen dabei aber nicht nur den Cursor. Befindet sich der Cursor auf der letzten Zeile, das heißt in Zeile 25, so bewirkt der Ausdruck eines CHR$(10) ein Scrollen des Bildschirms. Der Schirm wird eine Zeile nach oben bewegt. Diesen Effekt kennen wir schon vom einfachen Drucken in der letzten Zeile. Wenn das Zeilenende bei der Textausgabe erreicht wurde, so wird der Schirm um eine Zeile nach oben verschoben.

Interessant ist jedoch, daß wir den Bildschirm mittels CHR$(11) in der obersten Zeile zum Scrollen in der Gegenrichtung, das heißt nach unten, veranlassen können. Dies ist sehr nützlich, wenn wir Texte nach unten oder oben rollen wollen, wobei jeweils eine neue Textzeile ausgegeben werden soll. Das Prinzip sähe dann so aus:

```
SCROLLEN NACH UNTEN     SCROLLEN NACH OBEN

LOCATE 1,1              LOCATE 1,25
PRINT CHR$(11)          PRINT CHR$(10)
LOCATE 1,1              LOCATE 1,25
PRINT"Text"             PRINT"Text"
```

Eine Anwendung für diese Technik bilden Textverarbeitungsprogramme. Aber auch beim Hereinrollen von Titeln im Bereich der Titelgrafik ergeben sich interessante Anwendungsmöglichkeiten. Überlegen Sie doch einmal, wie und wo Sie dieses Rollen einbauen können.

Einen anderen Effekt hat PRINT CHR$(12) oder PRINT "<CTRL L>" Bei Eingabe dieses Codes führt der CPC ein "CLEAR-HOME" aus. Er löscht den Bildschirm und geht auf den Anfang der obersten Bildschirmzeile. Dieser Befehl stellt also das Gegenstück zu dem BASIC-Befehl CLS in Maschinensprache dar.

PRINT CHR$(13) oder PRINT "<CTRL M>" funktioniert ähnlich wie das Drücken der RETURN-Taste. Der Cursor wird auf den linken Rand der aktuellen Zeile bewegt. Damit sind sehr schnelle Cursorbewegungen möglich. Auch das Überschreiben von Texten kann auf diese Weise geschehen, da hier kein Zeilenvorschub erfolgt.

Wir befinden uns also nach der Ausführung des Befehls wieder am Anfang der Zeile, auf der wir gerade geschrieben haben. Probieren Sie einmal

```
PRINT "AB";CHR$(13);"abcd".
```

Was sehen Sie? Sie erhalten nur die Ausgabe "abcd". Die zuerst geschriebenen zwei Buchstaben sind nämlich durch die Rückführung des Cursors überschrieben worden. Mit CHR$(13) ist es also zum Beispiel im Rahmen eines Textverarbeitungsprogramms leicht möglich, die aktuelle Zeile nach einer Änderung ohne nochmaliges LOCATE dennoch richtig deckend auszugeben.

Wenn man die Cursorsteuerzeichen im weiteren Sinne einzeln benutzt, kann man schon einige interessante Effekte erzielen. Ihre ganze Stärke zeigen diese Kürzel jedoch erst in der Kombination. Geben Sie zum Beispiel einmal den folgenden Ausdruck aus.

```
PRINT CHR$(12)+"HIER ANFANG"+STRING$(CHR$(10),5)+"5 ZEILEN RUNTER" +CHR$(10)+
STRING$(CHR$(8),3)+"DANN ZURUECK"+CHR$(10)+ CHR$(13)+"UND AN DEN ANFANG DER
NAECHSTEN ZEILE"
```

Mit relativ wenigen Eingaben haben Sie so eine schon sehr komplexe Bildschirmausgabe vorgenommen. Ein besonderer Vorteil dieser Art der Steuerung liegt nun aber darin, daß die Steuercodes allgemein wie ein normales Zeichen behandelt werden können.

Wir können also eine grafische Ausgabe mit den zugehörigen Kontrollsequenzen in einem String speichern, Teile aus Strings herauslösen und

neu zusammensetzen und haben damit die Möglichkeit sehr schnell und effizient Grafiken und Texte zusammenzubauen und auf den Schirm zu bringen.

Auch können mehrmalig hintereinander auftauchende Steuercodes abgekürzt mit Hilfe von STRING$ eingegeben werden. In unserem obigen Beispiel haben wir dies angewandt, um die "fünf Zeilen runter" mit Hilfe eines Stringbefehls zu erzeugen. Wir haben also jetzt die Möglichkeit, eine Zeichenkette mit Hilfe von Steuercodes optimal zu positionieren und Bewegungen über den ganzen Bildschirm zu programmieren. Damit haben wir die Grenzen der Kontrollcodes jedoch noch lange nicht erreicht.

### 2.4.2 Die Farb- und Grafikcodes

Als nächstes stehen Strings auf dem Programm, die ihre Farbe selbständig ändern. Dies läßt sich mit den nächsten beiden CHR$-Nummern erreichen. Sie stellen das Äquivalent für die BASIC-Befehle PAPER und PEN dar. PRINT CHR$(14) oder PRINT "<CTRL+N>" beziehungsweise PRINT CHR$(15) oder PRINT "<CTRL+O>" sind mit der Ausführung eines PAPER 1 - beziehungsweise PEN 1 - Befehls identisch, und damit für den Normalbenutzer wohl nicht von allzu großem Interesse. Durch Anhängen eines zweiten Characters erhalten wir aber wiederum die anderen Farbregister.

```
PRINT CHR$(15)+CHR$(2)
```

führt also ein PEN 2 aus. Ein Anwendungsfeld hierfür könnte ein Programm sein, das den Bildschirm hauptsächlich aus Strings, zum Beispiel Ketten von Blockgrafiksymbolen aufbaut. Hier wäre es dann möglich, Farbumschaltungen mit in den Text zu integrieren.

Dies bietet ungeahnte Möglichkeiten, erfordert aber ein wenig Umdenken in Bezug auf die gewohnte Art der Farbwahl. Als ersten Einstieg in diesen Bereich sollten Sie einmal die nachfolgenden Befehle ausführen:

```
a$="abcd"+CHR$(15)+CHR$(2)+"efgh"+CHR$(15)+CHR$(3)+"ijkl"
PRINT a$
```

Die Kommandos sollten dabei im MODE 1 in der Ausgangsfarbzusammenstellung ausprobiert werden. Gegebenenfalls ist der Computer also mit

```
<SHIFT><CTRL><ESC>
```

zuerst wieder in den Einschaltzustand zu versetzen. Unsere Zeichenkette wird in drei verschiedenen Farben ausgegeben. Die ersten vier Buchstaben

bleiben gelb. Die zweite Gruppe wird nach der Umschaltung in blau dargestellt. Der dritte Teil unseres Strings erscheint schließlich in rot.

Sogar ein blinkender String ist nach dieser Methode machbar. Dazu definieren Sie beispielsweise INK 3,11,6. Wenn Sie dann

```
PRINT CHR$(14)+CHR$(3)+"Warnmeldung"+CHR$(14)+
CHR$(0)
```

ausgeben, erhalten Sie eine effektvolle Ausgabe. Das zweite CHR$-Kommando dient dabei übrigens dazu, die Hintergrundfarbe für die normale Ausgabe wieder auf dunkelblau zurückzuschalten. Unterläßt man dies, so blinkt danach der ganze Schirm, was ja nicht Sinn unseres Experimentes war.

Natürlich ist es auch besonders im Mode 0 möglich, mehrere Farbumschaltungen nacheinander mit demselben String vorzunehmen. Damit kann zum Beispiel in einer Programmüberschrift jeder Buchstabe einzeln in einer anderen Farbe dargestellt werden, auf Wunsch auch blinkend.

Auf einen weiteren Punkt sollte in diesem Zusammenhang auch noch hingewiesen werden. Dieser betrifft den benötigten Speicherbedarf. Obwohl die Definition mit CHR$ einen ziemlich langen Zeichen-Bandwurm benötigt, jedenfalls im BASIC-Quelltext, werden dann doch in dem dadurch definierten String nur wenige Zeichen durch die Kontrollsequenzen belegt. Da alle Steuercodes "normale Zeichen" sind, benötigen sie nämlich auch jeweils nur den Platz für ein Zeichen, also ein Byte.

Dazu müssen nun natürlich jedesmal noch die weiteren Zeichen für die Paramaterdefinitionen addiert werden. Trotzdem ist das Einladen solcher einmal definierter Strings mit Sicherheit die schnellste und speicherplatzsparendste Methode, um schnell einen Grafik-Bildschirm aufzubauen. Schauen wir uns nun einmal die weiteren Kommandos an.

Interessanter noch als die Farbsteuerbefehle sind die Codes CHR$(19) oder "<CTRL+S>" beziehungsweise CHR$(20) oder "<CTRL+T>".

Mit diesen ist es möglich, in Abhängigkeit von der Position des Cursors einen Teil des Bildschirms zu löschen. Mit CHR$(19) löschen wir dabei den Bildschirm über dem Cursor. Mit der Ausgabe von CHR$(20) löschen wir ab der aktuellen Cursorposition bis in die untere rechte Bildschirmecke. Anfangs- bzw. Endpunkt ist also immer die aktuelle Cursorposition.

Dies ist sehr nützlich, um bei textintensiven Programmen schnell eine Löschung der benötigten Bildschirmbereiche zu erhalten. Der Punkt, an

dem sich der Cursor momentan befindet wird dabei als letzter Punkt mitgelöscht. Wird zum Beispiel der Text in den Bildschirmzeilen 1 bis 10 nicht mehr benötigt, so setzt man den Cursor mit dem LOCATE-Kommando auf das Ende der zehnten Zeile, das heißt je nach eingeschaltetem MODE in Spalte 20, 40, oder 80 und führt dann ein

```
PRINT "<CTRL S>"
```

aus. Damit wird dann der obere Teil des Bildschirms gelöscht. Man braucht dann nicht erst ein Window auf diesen Bereich zu definieren und dieses gegebenenfalls nach einer erneuten Definition der Hintergrundfarbe mit PAPER und CLS zu löschen.

Sie können natürlich auch hier wieder kombinieren. Wenn Sie zum Beispiel vor der Ausführung dieser Befehle mit CHR$(14) die PAPER-Farbe undefinieren, wird mit dieser neuen Farbe gelöscht. Und dann sind natürlich auch diese Kommandos in Strings wieder addierbar.

Nun beinhaltet die Verwendung von CHR$(19) und CHR$(20) bereits eine relativ große Änderung des Schirms. Aber auch hier ist Abhilfe möglich. Wenn Ihnen die gelöschte Fläche zu groß ist, so können Sie mit den Steuercodes CHR$(17) und CHR$(18) auch nur auf eine Zeile zurückgreifen.

Diese Methode bietet sich immer dann an, wenn beim Editieren einer Eingabe der gerade geschriebene Text laufend ausgegeben wird. Dies ist zum Beispiel dann der Fall, wenn eine Eingabe statt mit INPUT durch INKEY$ simuliert wird. Hat man sich bei dieser Methode verschrieben und gibt deshalb einen Code ein, der das Löschen des letzten Zeichens bewirkt, so kann man zwar relativ einfach das letzte Zeichen von dem String, der diese Zeile enthält, abtrennen. Nach der nächsten Ausgabe befindet sich das gelöschte Symbol aber immer noch auf dem Schirm.

Der Grund dafür liegt darin, daß der ausgegebene String ja immer ab derselben Position gedruckt wird. Der String wurde zwar um das letzte Symbol gekürzt. Es werden aber nur die letzten n-1 Characters überschrieben, da der String ja um eben dieses Zeichen kürzer ist, als sein Vorgänger. Das fragliche letzte Zeichen wird also nicht überdeckt.

Wenn man in einem solchen Fall ein abschließendes CHR$(18) mit ausgibt löst sich dieses Problem wie von selbst.

### 2.4.3 Die BASIC-Ersatzcodes

Wir überspringen nun die drei folgenden Codes und setzen mit CHR$(25) wieder ein. Mit diesem Steuerzeichen ist es möglich, das SYMBOL-Kommando durch Steuercodes nachzubilden. Dazu müssen mit dem CHR$(25) 9 weitere Angaben mitgeliefert werden. Die Nummer des zu definierenden Zeichens und natürlich die Werte für die acht Punktzeilen, aus denen jeder Character besteht.

Der erste Wert definiert dabei wie gewohnt die oberste Pixelreihe, der zweite die nächstfolgende und so weiter. Die einzelnen Reihenparameter, beziehungsweise ihre CHR$-Funktionen werden dabei wieder mit "+" verbunden.

Ähnlich wirkt CHR$(26). Mit diesem Code können wir das WINDOW-Kommando ersetzen. Die mit CHR$(26) zu übergebenden vier Werte stehen dabei für die Eckpunktangaben, wie bei WINDOW gewohnt. Die ersten beiden Parameter definieren dabei den linken und rechten Rand. Das zweite Wertepaar gibt die oberste beziehungsweise unterste Bildschirmzeile des Fensters an.

Mit CHR$(28) und CHR$(29) können wir INK und BORDER imitieren. Dabei ist zu beachten, daß die Farben beim CPC grundsätzlich blinkend definiert sind. Wenn Sie also zum Beispiel INK 2,2 eingeben, so stellt der Computer nicht einen blauen Hintergrund dar, sondern er wechselt ständig zwischen zwei Farben ab. Da aber in diesem Fall beide auf blau definiert sind, ist dies unerheblich. Wir haben trotzdem den Eindruck einer stehenden Farbe.

Der CPC benötigt intern aber immer zwei Farbangaben, um eine INK definieren zu können. Deshalb müssen mit CHR$(28) und CHR$(29) auch jeweils zwei Farben mitgeliefert werden, um eine fehlerfreie Ausgabe zu sichern. Zusätzlich ist bei der INK-Definition mit CHR$(28) allerdings auch noch das Farbregister (hier Register 1) anzugeben.

```
PRINT CHR$(28)+CHR$(1)+CHR$(21)+CHR$(21)
```

setzt also INK-Register 1 auf grün. Die Angabe für grün (21) muß dabei im Gegensatz zum INK-Kommando zweimal folgen. Ansonsten blinkt die ausgewählte Farbe gegen grau (Farbe 27), was einen etwas abgeschwächten, pastellfarbenen Blinkeffekt ergibt. Sie können sich diesen einmal mit

```
PRINT CHR$(29)+CHR$(24)
```

anschauen.

Neben WINDOW, SYMBOL und den Farbkommandos sind jedoch auch noch eine Reihe anderer BASIC-Kommandos verfügbar. Mit

```
PRINT CHR$(30)
```

oder

```
PRINT <CTRL ^>
```

wird ein Cursor-HOME ausgeführt, das heißt der Cursor wird auf den Anfang der obersten Bildschirmzeile in die oberste, linke Bildschirmecke gesetzt. Im Gegensatz zu PRINT CHR$(12) wird dabei der Bildschirm allerdings nicht gelöscht. Dieses Kommando kann dazu benutzt werden, schnell in den oberen Teil des Bildschirms zu gelangen.

Hat man dagegen in der Mitte zu tun, so hilft PRINT CHR$(31) mit den zugehörigen Parametern. Dieser Befehl ist das Gegenstück zum LOCATE-Befehl in Maschinensprache. In BASIC benötigt der Befehl zwei Angaben, die Cursorspalte und die zugehörige Zeile. Wir müssen also eigentlich mit zwei nachfolgenden CHR$-Kommandos die Zeilennummer und die Bildschirmspalte eingeben.

Falls wir dies jedoch unterlassen, und das ist der Trick dabei, werden diese auf (1,12) gesetzt. Damit wird der Cursor auf den Beginn der Zeile Nr. 12 gesetzt, eine Option, die freilich beim BASIC-Befehl nicht funktioniert. Die Weitersteuerung des Cursors kann dann mit den einfachen Cursorsteuer-Befehlen (CHR$(8) bis CHR$(11)) problemlos erfolgen.

Wir sind nun am Ende der Steuerbefehle angelangt. Die Creme de la Creme in diesem Bereich haben wir aber zwischendurch - übrigens ganz bewußt - übersprungen. Diese Steuercodes gehören wieder mehr in den Bereich der Farbänderungen. Hier werden aber nicht einfach Farben definiert. Es geht hier eher um das Gebiet der Spezialeffekte.

### 2.4.4 Codes für Special effects

Die Kommandos sind sehr weitreichend, so daß es sinnvoll war, zuerst noch ein wenig Übung mit der Anwendung der Steuercodes zu erlangen, bevor wir uns mit ihnen nun näher befassen wollen. Gehen wir Stück für Stück vor. Als erstes Kommando steht dabei CHR$(24) auf dem Plan. Mit

```
PRINT CHR$(24)
```

oder

```
PRINT "<CTRL X>"
```

haben wir eine andere Möglichkeit, eine REVERS-Darstellung zu erreichen. Der Befehl vertauscht die Schrift- und die Hintergrundfarbe. Wenn wir also an irgendeiner Stelle einen Text besonders hervorheben wollen, so ist dies mit

```
PRINT CHR$(24)+"TEXT"+CHR$(24)
```

leicht möglich. Sie sollten dabei aber unbedingt auf die Rückschaltung mit Hilfe einer nochmaligen Befehlsausführung achten. Sonst bleibt Ihnen die Revers-Darstellung auch in den nächsten Zeilen erhalten.

Ein besonderer Vorteil dieses Steuercodes liegt übrigens darin, daß er unabhängig von irgendwelchen Windows arbeitet. Es ist also egal, ob man die gerade beschriebene Zeichenfolge im WINDOW#3 oder im Hauptwindow ausgibt, oder wie groß die einzelnen Fenster sind. In jedem Fall wird die Schriftfarbe des durch PRINT angesprochenen Fensters mit der zugehörigen Hintergrundfarbe vertauscht. Das leidige Umdefinieren von PAPER und PEN kann also entfallen.

War dies schon ein ganz netter Effekt, so haben wir mit CHR$(23) noch eine Steigerung parat. Mit diesem Steuercode wird der Modus des Grafikfarbstiftes gesetzt. Wenn Sie normalerweise mit DRAW eine Linie ziehen, so wird dabei die Vordergrundfarbe neu, das heißt auf die einzelnen Bildpunkte, gesetzt. Diese Darstellungsart ist unabhängig von der Vorgeschichte, also von dem Inhalt, den die beschriebene Speicherstelle vor dem Schreiben beinhaltete. Wenn Sie nun aber

```
PRINT CHR$(23)+CHR$(1)
```

ausführen, so ändert sich dies. Die nun gesetzte Farbe ergibt sich aus einer Verknüpfung der neuen Vordergrundfarbe mit dem an dieser Stelle vorhandenen Farbwert und zwar in diesem Fall aus einer XOR-Verknüpfung. Wie dies wirkt, wollen wir nun einmal anhand einiger Beispiele untersuchen.

Betrachten wir zunächst den einfachsten Fall, nämlich daß Sie mit INK 1 auf INK 0 schreiben. In diesem Fall ist das Ergebnis wie folgt: War an der Stelle, die Sie mit DRAW oder PLOT nun setzen wollen bereits ein Punkt gesetzt, so wird er rückgesetzt. War der Punkt dagegen mit der Hintergrundfarbe belegt, so erfolgt jetzt die Umdefinition auf Vordergrundfarbe.

| | | | | |
|---|---|---|---|---|
| BIT A | 0 | 0 | 1 | 1 |
| BIT B | 0 | 1 | 0 | 1 |
| Erg. | 0 | 1 | 1 | 0 |

**Tabelle 2.1:** *XOR-Verknüpfung*

Schwieriger wird es, wenn wir mit mehreren Farben arbeiten. Dann tritt bei gesetzten Punkten ein Farbwechsel auf. Der CPC verknüpft nämlich nun die verschiedenen Farbwerte anhand ihrer Bitmuster. Im MODE 1 wird die Farbe eines Punktes in zwei Bits abgelegt. Soll nun ein Punkt der auf das Farbregister 2 gesetzt war, mit INK 3 belegt werden, so verknüpft der CPC die Farbwerte wie folgt:

| | | |
|---|---|---|
| alter Farbwert | 2 = binär | 10 |
| neuer Farbwert | 3 = binär | 11 |
| XOR-Verknüpfung | | 01 |

Der Punkt wird also nun mit INK 1 dargestellt. Die XOR-Funktion setzt also einen Punkt, wenn die Werte der beiden Angaben unterschiedlich sind. Bei identischen Angaben dagegen, wird in jedem Fall der Punkt beziehungsweise ein Farbbit gelöscht.

Wenn Ihnen diese Überlegungen zu theoretisch sind, können Sie der Sache natürlich auch durch einfaches Experimentieren zu Leibe rücken. Geben Sie einfach einmal die folgenden Kommandos ein.

```
PLOT 320,200,2
PRINT CHR$(23)+CHR$(1)
PLOT 320,200,3
```

Sie erhalten dann einen Farbpunkt im Bildschirmzentrum, der nach Ausführung des dritten Kommandos die Farbe wechselt. Allerdings nicht, wie Sie vielleicht meinen, auf rot. Das Überschreiben des blauen Punktes mit "rot" liefert uns als Ergebnis einen gelben Tipser im Bildschirmzentrum.

Eine weitere Art der Farbkombination erhält man mit CHR$(23)+CHR$(2). Hier wird die AND-Funktion auf die einzelnen Punkte angewandt. Ein Punkt wird dabei nur gesetzt, wenn er schon vorhanden ist und auch weiterhin gesetzt sein soll. Gleiches gilt für die Farbbits in den Modes 1 und 0. In drei von vier Fällen hat diese Verknüpfung also als Ergebnis die INK 0.

| | | | | |
|---|---|---|---|---|
| BIT A | 0 | 0 | 1 | 1 |
| BIT B | 0 | 1 | 0 | 1 |
| Erg. | 0 | 0 | 0 | 1 |

**Tabelle 2.2:** AND-Verknüpfung

Beispiel: Es werden dieselben Eingaben durchgeführt, wie bei der XOR-Verknüpfung, nur daß jetzt die Farbwerte AND-verknüpft werden.

| | | | |
|---|---|---|---|
| alter Farbwert | 2 | = binär | 10 |
| neuer Farbwert | 3 | = binär | 11 |
| AND-Verknüpfung | | | 10 |

Man kann die AND-Funktion also anwenden, um einen Bereich des Bildschirms zurückzusetzen, während der Restschirm unverändert bleibt. Das Fachwort hierfür lautet maskieren. Dazu setzt man die erwünschten Punkte eines Zeichens, also jene, die unverändert bleiben sollen in einem Masken-Character mittels SYMBOL auf "1". Druckt man diesen dann über eine Stelle des Schirms, so bleiben in dem so überdruckten Zeichen nur die ausgewählten Punkte erhalten.

Eine Nebenbedingung existiert dabei allerdings noch. Das Setzen unseres Masken-Symbols muß im Grafikmodus geschehen. Ein Ausdruck mit PRINT wird nämlich von dieser Modusumschaltung nicht betroffen. Vor der Ausgabe ist also TAG einzugeben, danach TAGOFF.

Die Wirkung der Parameterangaben 1 und 2 in Zusammenhang mit CHR$(23) haben wir jetzt ausreichend behandelt. Es gibt jedoch noch zwei weitere mögliche Angaben. Senden wir mit dem CHR$(23) ein CHR$(3), so schaltet der CPC den Grafikfarbstiftmodus ebenfalls auf XOR-Verknüpfung. Ein mitgeliefertes CHR$(0) bringt uns wieder in die normale Welt zurück.

Nach so viel Theorie stellt sich nun natürlich die Frage, wozu wir diese Kommandos überhaupt gebrauchen können. Ein paar Anwendungsbeispiele mögen hier Anregungen geben.

### 2.4.4.1 Schnelle Zeichenumdefinition durch Maskierung

Ein einfaches Anwendungsbeispiel ist die Definition unterstrichener Buchstaben mit dieser Methode. Werfen wir einen Blick zurück auf die Thematik von Kapitel 2.1, wo wir uns mit der Zeichendefinition schon ausgiebig beschäftigt haben.

Anhang III im Handbuch liefert uns für jedes Zeichen die zugehörige Zeichenmatrix. Wenn Sie sich die Symbole nun etwas näher anschauen, fällt Ihnen sicher sofort auf, daß bei allen Großbuchstaben die unterste Punktzeile nicht besetzt ist. Wir brauchen nun also unsere Buchstaben bloß mit einem Maskencharacter zu verknüpfen, der in der untersten Zeile einen Strich enthält und schon haben wir den gewünschten Effekt.

Im Zeichensatz des Schneider ist sogar ein solches Zeichen schon vorhanden, nämlich Symbol Nummer 95. Wenn wir nun also ein Wort von vier Buchstaben unterstreichen wollen, so geht dies relativ einfach. Zunächst müssen wir die Zeichen ausgeben. Dies geht mit einem einfachen PRINT beispielsweise

```
10 CLS:PRINT "TEXT"
```

Nun erfolgt die Überlagerung.

```
20 MOVE 0,399
30 PRINT CHR$(23)+CHR$(1)
40 TAG
50 FOR i=1 TO 4
60 PRINT CHR$(95);
70 NEXT i
80 TAGOFF
90 PRINT CHR$(23)+CHR$(0)
```

**Programmbeschreibung:**

Als erster Schritt wird in Zeile 10 ein Probetext ausgegeben. Durch den ersten CHR$-Code in Zeile 30 wird die XOR-Verknüpfung aktiviert. Mit TAG wird dann die nächste Zeichenausgabe an die aktuelle Position des Grafikcursors dirigiert. Dieser wurde mit dem MOVE-Befehl in die linke obere Eckposition gesetzt. Ein nun dargestelltes Zeichen wird also in der ersten Bildschirmzeile ausgegeben. Damit ist die für die Überlagerung notwendige Umschaltung auf den Grafikmodus vollzogen.

Die Schleife in Zeile 50-70 leistet nun das Überdrucken. Viermal hintereinander wird das Zeichen 95 ausgegeben, mit seinem Untergrund XOR-verknüpft und bildet so die Unterstreichung. TAGOFF und das zweite CHR$ schalten den CPC wieder auf die Normaldarstellung zurück.

Mit demselben Prinzip ist es natürlich auch möglich ein Zeichen zu benutzen, das durch Subtraktion, oder besser gesagt durch das Weglassen von Punkten aus einem anderen hervorgegangen ist. Man wendet in diesem Fall einfach die AND-Verknüpfung an. Alle Punkte, die im Maskenzeichen nicht gesetzt waren, werden damit gelöscht.

Ein weiteres Anwendungsfeld für diese Verknüpfungen bilden Farbwechsel. Der CPC selbst wendet diese Techniken beispielsweise an, um die Cursorführung sichtbar zu machen. Wendet man nämlich auf eine Stelle des Schirms nacheinander zweimal die XOR-Funktion an, so hat man nichts geändert.

Dennoch ist bei einmaliger Benutzung sichergestellt, daß sich die so bearbeitete Stelle deutlich vom Restschirm abhebt. Um dies auszuprobieren, sollten Sie einmal das nachfolgende kleine Programm eintippen.

```
10 CLS:PRINT"A"
20 MOVE 0,399:TAG:PRINT CHR$(23)+CHR$(1)
30 IF INKEY$="" THEN 30
40 PRINT CHR$(143);
50 IF INKEY$="" THEN 50
60 MOVE 0,399
70 PRINT CHR$(143);
80 TAGOFF:PRINT CHR$(23)+CHR$(0)
```

**Programmbeschreibung:**

Die Demonstrationsroutine stellt zuerst in der linken oberen Ecke des Bildschirms ein "A" dar. Als nächstes, das heißt nach Tastendruck wird dieses mit CHR$(143), dem ausgefüllten Quadrat, XOR-verknüpft. Nochmaliger Tastendruck schaltet wieder in den Ausgangszustand zurück.

Nach dem ersten Tastendruck sehen Sie das Zeichen so, wie es dargestellt würde, wenn der Cursor auf diese Position gesetzt worden wäre. Kein Wunder, denn der CPC benutzt dazu ja eben diesen Mechanismus.

Sie sollten dieses Programm nun einmal in den verschiedenen Modes ausprobieren, um sich mit der Funktion dieser Verknüpfungen vertraut zu machen. Man muß erst ein wenig mit dieser Möglichkeit experimentiert haben, um sie sinnvoll anwenden zu können.

### 2.4.4.2 Transparent-Darstellungen und Überlagerungseffekte

Wenn wir nun noch einen Schritt weiter gehen, so treffen wir auf den wohl stärksten Steuercode, CHR$(22). Dies ist die Bedienfunktion für die Transparent-Option. Worum handelt es sich dabei?

Normalerweise setzt der CPC bei jedem ausgedruckten Zeichen eine Punktmatrix von 8*8 Punkten im Grafikspeicher ab der Adresse hex C000 im RAM. Dazu werden die einzelnen Punkte mit dem Wert der INK-Register geladen, die dann für die Farbdarstellung benutzt werden sollen.

Auf den genauen Ablauf dieser Prozesse kommen wir noch in Kapitel 6 zu sprechen.

Wenn der CPC normal ein Zeichen ausgibt, so setzt er einen Hintergrundpunkt auf die PAPER-INK, während für die Darstellung des Vordergrundes dann die Schriftfarbe (PEN) gewählt wird.

Soweit zur Normaldarstellung. Bei eingeschalteter Transparent-Option unterbleibt nun das Schreiben des Hintergrundes. Hier wird nur noch die für die Schrift gewählte INK gesetzt. Dies hat zur Folge, daß jetzt ein Überdrucken möglich ist.

Um Ihnen einen Eindruck von diesem Effekt zu vermitteln, sollten Sie einmal das folgende kleine Experiment durchführen. Geben Sie einmal auf einer Zeile einen kleineren Text ein und führen Sie dann auf einer anderen, möglichst freien Zeile ein

```
PRINT CHR$(22)
```

aus. Danach können Sie einmal versuchen den oben geschriebenen Text mit DELETE zu löschen, oder durch Überschreiben zu vernichten. Sie werden dabei einige Probleme bekommen und gegebenenfalls Ihr gelbes Wunder erleben, vorausgesetzt, Sie sind glücklicher Besitzer eines Farbmonitors und dieser befindet sich noch in der Ausgangs-Farbzusammenstellung.

Die Zeichen lassen sich nämlich nicht mehr löschen. Wenn Sie auf die Leertaste drücken, so passiert nicht mehr, als daß sich der Cursor weiter bewegt. Ansonsten kommen nur neue, noch nicht gesetzte Punkte, hinzu.

Wir können diese Möglichkeit nun in vielen Bereichen anwenden. Wollen wir zum Beispiel ein Zeichen aus zwei anderen durch Übereinanderdrucken schaffen, so ist dies mit CHR$(22) genauso wie mit CHR$(23)+CHR$(1) möglich. Allerdings entfällt hier die teilweise etwas komplizierte Verknüpfung. Und auch die Farbeffekte, das Umkippen der Farben tritt nicht auf.

CHR$(22) leistet also ein einfaches Überschreiben. Sie geben mit demselben LOCATE-Befehl zweimal hintereinander verschiedene Zeichen an derselben Position aus, und schon haben Sie Ihr neues Bildschirmsymbol definiert. Probieren Sie doch einmal die Wirkung der folgenden Eingaben aus:

```
PRINT CHR$(22)+CHR$(1)
LOCATE 10,10:PRINT CHR$(97)
```

```
LOCATE 10,10:PRINT CHR$(126)
PRINT CHR$(22)+CHR$(0)
```

Durch Überdrucken wurde hier aus dem "a" und dem Schlangenzeichen ein neues Symbol gewonnen. Dies ist in manchen Fällen einfacher, als mit SYMBOL neue Zeichen zu definieren.

Bei dieser Beschreibung sind wir allerdings noch davon ausgegangen, daß Sie mit derselben Farbe gearbeitet haben. Dies ist jedoch keineswegs zwingend. Wir können nämlich auch zwischen den Zeichenausgaben die Schriftfarbe und natürlich auch die Angaben für den Hintergrund jederzeit mit PEN und PAPER beziehungsweise mit CHR$(14) und CHR$(15) ändern.

Wenn wir nun entsprechende Zeichen definieren, können wir aus einem zweifarbigen Symbol durch Überdrucken ein Multicolor-Zeichen machen. Im Prinzip gehen Sie dabei genauso vor, wie der Einband dieses Buches hergestellt worden ist. Sie "drucken" mehrere Farben nacheinander übereinander und erhalten dann erst die vielfarbige Darstellung.

Ein Unterschied besteht aber doch noch. Während beim Druck zwei Farben eine Mischfarbe ergeben können, so ist dies beim Schneider nicht möglich. Hier herrscht eine klare Farbgebung. Es bleibt nur der zuletzt gesetzte Punkt erhalten. Speziell, wenn man im Mode 0 arbeitet, ist dies jedoch kein Nachteil, da man die hier definierbaren 16 verschiedenen Farben beziehungsweise Farbschattierungen meist sowieso nicht ausnützt.

### 2.4.5 BASIC-Sprites

Um noch weitergehende Effekte zu erzeugen, können wir CHR$(22) dann auch noch mit den anderen Steuerkommandos kombinieren. Da es auch problemlos möglich ist, verschiedene Zeichen durch geeignete Cursorsteuerbefehle verbunden nacheinander auszugeben, ist ein solches Gebilde auch als String abspeicherbar. Wir haben damit dann eine Möglichkeit geschaffen, um auch in BASIC mit Shapes (=Software-Sprites) arbeiten zu können. Als Beispiel hier einmal ein Tricolor-Raumschiff-Shape.

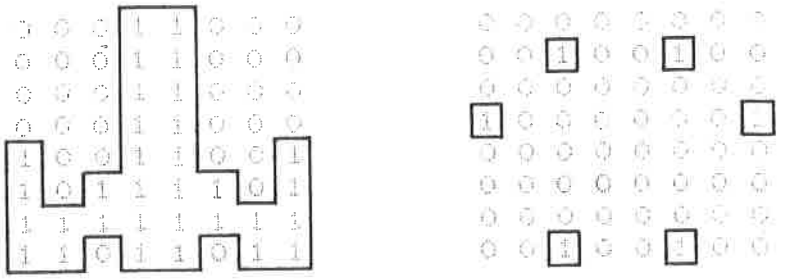


**Bild 2.4:** *Entwurfsschablonen für Raumschiffshapes*

```
10    SYMBOL
      250,&x00011000,&x00011000,&x00011000,&x00011000,&x10011001,&x10111101,&x
      11111111,&x11011011
20    SYMBOL 251,&x0,&x00100100,&x0,&x10000001,&x0,&x0,&x0,&x00100100
30    MODE 0:BORDER 0:INK 0,0:INK 1,27:INK 2,24,6:CLS
40    PRINT CHR$(22):LOCATE 10,12:PEN 1:PRINT
      CHR$(250)+CHR$(15)+CHR$(2)+CHR$(8)+CHR$(251)+CHR$(22)+CHR$(0)
50    IF INKEY$="" THEN 50 ELSE PEN 1:END
```

**Programmbeschreibung:**

Die Bilder, nach denen die beiden Symbole entworfen wurden, finden Sie in Bild 2.3. Für jeden in der Ausgabefarbe gesetzten Punkt gibt es eine 1 im SYMBOL-Befehl. Die anderen Punkte werden mit 0 aufgefüllt. Dies ist jedoch eigentlich kein Problem und erfordert nur ein bißchen Kreativität. Der linke Teil unserer Abbildung liefert das Grundgerüst des Fliegers. Der rechte Teil gehört den Spezialeffekten, dem Blinken der Triebwerke und anderen besonders hervorgehobenen Teile.

Nachdem wir zwei Symbole auf die Werte dieser Schablonen definiert haben, werden dann in Zeile 30 die Eckkoordinaten für die Ausgabe gesetzt: Moduswahl, Setzen der drei Farbregister und Löschen des Bildschirms. Dann folgt die eigentliche Zeichenausgabe, wobei bereits in breitem Rahmen auf die Systemsteuercodes zurückgegriffen wurde.

Zunächst wird die Transparent-Option eingeschaltet und die Ausgabeposition festgelegt. Die Vordergrundfarbe wird auf 1 gesetzt. Für die letzten beiden Angaben wurden dabei BASIC-Befehle verwendet. Wir hätten hier

genausogut auch mit CHR$(31) und CHR$(15) arbeiten können (Wie hätten dann die Characterfolgen gelautet?).

Es folgt ein CHR$-Bandwurm. Schauen wir uns die dabei ablaufenden Funktionen einmal Zeichen für Zeichen an. Zunächst wird in der gerade gesetzten Grundfarbe (INK 1) die erste Zeichenebene ausgegeben, das Rumpf-Raumschiff. Die nächsten beiden Characters bewirken über die PEN-Umschaltung das Setzen der zweiten Vordergrundfarbe, mit der jetzt einige Besonderheiten wie das Triebwerksfeuer und die Kanzelbeleuchtung überlagert werden. Da die dazu verwandte INK 2 auf Blinken definiert wurde, ergibt sich der "realitätsnahe" Effekt.

Bei der Ausgabe der zweiten Vordergrundfarbe ist dabei darauf zu achten, daß der Cursor zuerst wieder zurückbewegt werden muß (CHR$(8)). Ansonsten wird das zweite Zeichen nämlich nicht über, sondern hinter das erste gedruckt. Abschließend sollten sie übrigens, wie auch hier vorgenommen, nie vergessen, die Transparent-Option wieder abzuschalten. Es werden nämlich alle Zeichen überschrieben, gegebenenfalls auch solche, bei denen dies Ihrerseits gar nicht beabsichtigt war.

Auf eine Besonderheit bei der Transparent-Option sollte an dieser Stelle auch noch aufmerksam gemacht werden. Sie wird Ihnen vielleicht beim Spielen mit diesen Möglichkeiten schon aufgefallen sein.

Der CPC hat jedes Zeichen im Grafikspeicher effektiv in Punkten abgelegt. Er kann also einen Überdruck nicht mehr nachträglich erkennen. Wenn Sie also versuchen einen überdruckten Text, der einen Befehl enthält mit dem COPY-Cursor zu übernehmen, um diesen nicht noch einmal eintippen zu müssen, so geht dies nicht. Der CPC erkennt die darunterliegenden Zeichen nämlich nicht mehr.

Mit Hilfe der Transparent-Option und der anderen Steuercodes können nun natürlich auch problemlos mehrfarbige Zeichen beliebiger Größe definiert werden, die wir dann mit LOCATE oder TAG über den Bildschirm bewegen können. Dazu müssen nur entsprechend mehr über- und unterdeckende Characters mit SYMBOL definiert werden.

Man könnte beispielsweise mit 8 Characters ein dreifarbiges (beziehungsweise mit der Hintergrundfarbe 4-farbiges) Makrosymbol in der Größe 2*2 problemlos definieren. Dazu addiert man die ersten beiden Zeichen, führt dann den Cursor mit CHR$(8)+CHR$(8)+CHR$(10) um eine Zeile nach unten und zurück und gibt danach die Zeichencodes für die untere Zeile aus. Jetzt geht es wieder an den Anfang mit CHR$(8)+CHR$(8)+CHR$(11), worauf die nächste Farbe gesetzt werden kann.

Auch ganze Bilder sind mit diesem Verfahren darstellbar. So kann zum Beispiel der Figurensatz eines Schachspiels auf diese Weise definiert werden (mit Schattengebung oder ähnlichen Effekten) und auch Spielfiguren jeder Art sind machbar.

Die Definition der Symbole sieht zwar auf den ersten Blick etwas schwierig aus. Mit dem Programm Zeichensatz aus Kapitel 2.1 dürfte sie jedoch problemlos machbar sein.

Wenn ein Zeichen dann aber erst einmal definiert ist, kann es autark auch mehrmals gleichzeitig auf den Schirm gebracht werden, ohne daß die Abspeicherung viel Platz einnehmen würde. Dies ist besonders bei Adventure-Spielen, die ja noch auf eine Menge Programmtext für die verschiedenen Ereignisräume angewiesen sind, wichtig. Spielen Sie doch ein wenig mit diesen Möglichkeiten. Übung macht auch hier den Meister.

# 3 Das Land der hochauflösenden Grafik

Mit den Überlagerungstricks aus dem letzten Kapitel, verabschieden wir uns nun vom Gebiet der Blockgrafik, dem Ausdruck von Zeichen und deren Überlagerung sowie der durch Spezialkommandos erreichbaren Farbeffekte. Wir verlassen die Zeichendarstellung und wenden uns der Einzelpunktansprache zu. Damit betreten wir einen bis jetzt völlig neuen Bereich, die hochauflösende Grafik.

Wie schon in der Einleitung von Kapitel 2 gesagt, haben wir es hier im wesentlichen mit der Ansprache von Punkten zu tun, und zwar als Einzelpunkte oder auch als Linien.

Als erstes wollen wir uns anhand einiger kleiner Anwendungsbeispiele damit vertraut machen, was, sozusagen als Standard-Features, mit der hochauflösenden Grafik machbar ist. Wir untersuchen dazu, wie wir ein paar ganz normale, simple und einfache Figuren darstellen können, Kreise, Rechtecke und Dreiecke in zwei- und dreidimensionaler Darstellung.

## 3.1 Das Koordinatensystem

Als erstes sollte man wohl, wenn man in die hochauflösende Grafik einsteigt, ein paar Basisfakten voranstellen. Die hochauflösende Grafik beim Schneider arbeitet mit einem Pseudoraster von 640*400 Punkten. Was bedeutet hier nun der Begriff 'Pseudoraster'?

Er soll andeuten, daß zwar virtuell, das heißt in der Kommunikation zwischen Benutzer und Computer, eine solche Matrix von 640 Bildpunkten in der Horizontalen und 400 in der Vertikalen angenommen wird. Die reale Auflösung auf dem Schirm sieht dann jedoch etwas anders aus.

Sie beträgt in allen drei Bildschirmdarstellungsarten (MODES) 200 Bildschirmlinien in der Höhe. Jede Linie weist dabei genau eine Dicke von

einem Punkt auf. Die Anzahl von Bildpunkten in der Horizontalen hängt dagegen vom Mode ab.

Wir haben schon in Kapitel 2 gesehen, daß die farblichen Möglichkeiten und ihr Widerpart, die Auflösung beim CPC in einem gegensätzlichen Verhältnis stehen. Je feiner unsere Auflösung sein soll, je mehr Bildpunkte wir in einer Zeile getrennt ansprechen wollen, desto geringer sind unsere farblichen Möglichkeiten und umgekehrt.

Dementsprechend ist auch die reale Auflösung in den einzelnen Modes unterschiedlich. Im Mode 0 haben wir es mit 160 Bildpunkten zu tun, die wir einzeln adressieren können. Im Mode 1 sind es 320, und im Mode 2 schließlich erreichen wir die eben angesprochenen 640 Punkte Maximalauflösung. Die einzelnen Bildpunkte sind dabei allerdings verschieden breit. Dies ergibt sich daraus, daß der CPC, wie schon gesagt, mehrere Bildpunkte zusammenfaßt, um dann die Gesamtzahl der Bildpunkte mit einer größeren Anzahl von Farben belegen zu können.

Sie können diesen Zusammenhang einmal überprüfen, indem Sie mit dem PLOT-Kommando einige einzelne Punkte auf den Schirm setzen. Adressieren wir zunächst einmal den höchstauflösenden Bildschirmmodus mit

```
MODE 2
```

Hier haben wir es mit 640 Bildpunkten in der Horizontalen zu tun. Wir können also jeden einzelnen Punkt auf dem Fernsehschirm getrennt ansprechen. Jeder Punkt ist dabei einzeln adressierbar. Tippen Sie doch einmal ein

```
PLOT 320,200
```

In der Mitte Ihres Bildschirmes sehen Sie nun, gegebenenfalls auch nur sehr schwach, einen kleinen, feinen Bildpunkt. Als nächsten Befehl tippen Sie nun

```
PLOT 321,200
```

ein. Ihr erster Bildpunkt bekommt nun auf der rechten Seite Gesellschaft. Wenn wir nach diesem Verfahren fortfahren, so dehnt sich langsam die Schar der Bildpunkte nach rechts aus. Die Höhe ist dabei jeweils auf eine Bildschirmzeile begrenzt. Wir können nun eine Variante desselben Themas einführen und diese Befehle einmal im Mode 1 ausprobieren. Zunächst schalten Sie also mit

```
MODE 1
```

in den ersten Bildschirmdarstellungsmodus um und geben dann

```
PLOT 320,200
```

ein. Wie zu erwarten war, finden Sie wieder einen Bildpunkt im Zentrum unseres Bildschirms. Dieser ist allerdings schon etwas stärker, und den Grund werden Sie gleich sehen. Wenn Sie nämlich nun als zweites Kommando

```
PLOT 321,200
```

eingeben, passiert nichts. Der Grund: Es wurden bereits beim ersten PLOT zwei nebeneinanderliegende Bildpunkte gleichzeitig gesetzt. Erst wenn Sie nun

```
PLOT 322,200
```

eintippen, erhalten diese beiden Gesellschaft und zwar wieder durch ein Paar Punkte. In jedem Fall werden also beim CPC im Mode 1 immer zwei Punkte parallel gesetzt. Nun bedarf es keiner besonderen Erklärung mehr, um diesen Vorgang in Richtung auf den Mode 0 weiterzudenken. Hier sind es dann vier Punkte. Das heißt, egal ob Sie

```
PLOT 320,200

PLOT 321,200

PLOT 322,200
```

oder

```
PLOT 323,200
```

eingeben. In jedem Fall erhalten Sie als Ergebnis ein Vierergespann von Punkten im Zentrum des Bildschirms. Und noch ein weiteres Experiment können Sie in diesem Bereich machen. Egal in welchem Modus Sie sich befinden, sollten Sie nun einmal die Y-Koordinate variieren, und zwar beispielsweise mit

```
PLOT 320,201
```

Was passiert? Nichts. Und dies ist auch nicht anders zu erwarten, denn obwohl Sie, wie wir schon gesagt haben, in der Kommunikation mit dem Computer mit 400 Punkten in der Y-Richtung operieren, arbeitet der CPC intern dennoch nur mit 200 Punkten Vertikalauflösung, was den 200 Bildschirmzeilen entspricht. Er teilt die Y-Angabe durch 2 und benutzt das daraus resultierende Ergebnis ohne Rundung als Basis für seine Bildschirmausgabe.

Wenn Sie also zwei Bildschirmlinien ansprechen, wobei die untere einen geraden Y-Wert aufweist, beispielsweise um eine Linie doppelter Dicke zu erzeugen, so ist dies eine ziemlich nutzlose Operation. Die zweite Linie ist mit der ersten identisch. Hier müßte also die um eins niedriger liegende virtuelle Bildschirmlinie benutzt werden. Umgekehrt gilt dieser Zusammenhang natürlich auch für eine Grundlinie mit einem ungeraden Y-Wert.

Nun stellt sich natürlich sofort die Frage, weshalb der CPC überhaupt mit dieser etwas seltsamen Wertangabe arbeitet. Der Sinn dieser etwas vertrackten Koordinatenangabe wird Ihnen aber sofort klar, wenn Sie einmal annehmen, Sie wollten einen Kreis zeichnen, und dabei dieselben Angaben für X- und Y-Ausdehnung benutzen, weil der Kreis ja denselben Radius haben soll.

Die Entfernung vom Mittelpunkt zum obersten Punkt unseres Kreises wäre also beispielsweise 50 und vom Mittelpunkt nach links und rechts ebenso. Wenn Sie nun aber unterstellen, daß der Computer bei Ihrer Wertangabe nur mit 200 Linien arbeiten würde, würde unser Kreis erheblich nach oben und unten ausgedehnt. Es ergäbe sich also im Endeffekt kein Kreis mehr, sondern eine Ellipse.

Um aber nun dennoch relativ einfach Kreise und ähnliche Figuren, bei denen es auf Symmetrie ankommt, darstellen zu können, hat man zu diesem virtuellen Koordinatennetz gegriffen. Daneben hat dieses Verfahren den Vorteil, daß die Genauigkeit der Linienführung bei schrägen Verbindungen erhöht wird. Der Rundungsfehler bei 400 Linien Höhe ist nur halb so groß, als wenn man mit 200 Linien arbeiten würde.

Nach dieser Vorrede kommen wir nun zu unseren eigentlichen Figuren.

## 3.2 Einfache Figuren-selbstgemacht

Zunächst zu ein paar ganz einfachen Operationen. Das Setzen von Punkten haben wir schon durchgeführt. Es erfolgt mit PLOT, wobei hier die Koordinaten direkt, wie bei unseren Beispielen, angegeben werden können oder relativ zu dem zuletzt gesetzten Punkt. Beim Einschalten ist dies immer der Punkt (0,0) in der linken unteren Bildschirmecke, ansonsten eben der zuletzt adressierte. Die Befehle lauten

PLOT und PLOTR

PLOTR würde also zum Beispiel, wenn wir als letzte Operation einen Bildpunkt im Bildschirmzentrum gesetzt hatten, und nun

```
PLOTR 100,0
```

ausführen, den Punkt 420,200 in der absoluten Bildschirmkoordinatenangabe mit Farbe besetzen. Nach PLOT befindet sich unser Grafikcursor immer an der Stelle, wo der Bildpunkt gesetzt wurde. Wir können ihn allerdings auch an einen anderen Punkt bewegen, dazu dienen die Kommandos

MOVE und MOVER

Mit MOVE wird der Cursor auf eine Absolutposition gesetzt. Mit MOVER relativ zu seinem letzten Aufenthaltsort bewegt. Die aktuelle Position des Grafikcursors ist auch der Ausgangspunkt, wenn wir Linien ziehen wollen. Dazu existieren die Kommandos

DRAW und DRAWR

DRAW zieht eine Linie von der aktuellen Cursorposition zu einem absolut angegebenen Punkt. Bei DRAWR werden die Zielrichtungen in X- und Y-Koordinaten relativ angegeben. Wenn wir also zum Beispiel vom Mittelpunkt eine Linie in waagerechter Richtung mit der Länge 50 ziehen wollen, so gibt es zwei Möglichkeiten. In jedem Fall müssen wir zuerst einmal den Grafikcursor auf den Mittelpunkt setzen, es sei denn, er ist dort schon vorhanden. Dies erreichen wir relativ einfach mit

```
MOVE 320,200
```

Als nächstes kommt nun unsere Linie an die Reihe. Entweder definieren wir

```
DRAW 370,200
```

als Absolutangabe oder wir tippen

```
DRAWR 50,0
```

als relative Positionsbestimmung ein. Das Ergebnis ist in beiden Fällen dasselbe. Welches Kommando sich für eine gegebene Aufgabenstellung besser eignet, hängt davon ab, was für Angaben man zur Verfügung hat. Geschlossene Linienzüge, wie wir sie zum Beispiel gleich noch benötigen werden, lassen sich normalerweise besser mit DRAWR angeben. Hat man ein eigenes Koordinatensystem vorliegen, so arbeitet man meist mit DRAW.

**Strahlenbündel**

Nun können wir natürlich auch von unserer Postion nicht nur eine Linie ziehen, sondern eine ganze Reihe von Linien, ein sogenanntes Strahlenbündel. Wenn dabei der Ausgangspunkt immer wieder das Bildschirmzentrum sein soll, so müssen wir nur den Grafikcursor vor jedem DRAW auf diese Position kommandieren und dann das Linienkommando ausführen. So erhalten wir zum Beispiel mit

```
MOVE 320,200:DRAW 320,0
```

eine Senkrechte nach unten. Mit

```
MOVE 320,200:DRAWR -100,0
```

erscheint eine Waagerechte nach links mit der Länge 100. Dies ist nun reichlich theoretisch. Wir wollen uns daher einmal an einigen Beispielen anschauen, wie wir die Linienführung konkret benutzen können.

### 3.2.1 Dreiecke und Vierecke

Dreiecke und Vierecke sind relativ einfache Figuren, die wir direkt in einem Linienzug zeichnen können. Dazu benötigen wir entweder die Absolutkoordinaten der Eckpunkte (bei einer Absolutdefinition) oder die Angaben über die Verschiebung in X- und Y-Richtung (bei einer relativen Definition).

**Verschiedene Dreieckstypen**

Speziell in mathematischen Lernprogrammen kommt es des öfteren vor, daß man Winkel- oder Streckenberechnungen ausführen soll. Schön wäre es, diese mit Hilfe einer kleinen Grafik veranschaulichen zu können. Wir wollen daher einmal ein paar häufig vorkommende Dreieckstypen auf dem Bildschirm des CPC realisieren. Es sind dies

- das rechtwinklige Dreieck (dadurch gekennzeichnet, daß ein Winkel 90 Grad hat).
- das gleichschenklige Dreieck (Wenn man hier die Grundlinie betrachtet, so verlaufen die beiden anderen Seiten zu dieser im selben Winkel).
- das gleichseitige Dreieck (Als Idealform des Dreiecks haben hier alle drei Seiten und alle drei Winkel identische Werte).

Und dann haben wir uns noch ein "normales" Dreieck vorgenommen. Das Programm 'Dreiecke', was Sie nun nachfolgend finden, zeichnet diese Figuren. Wir wollen es uns einmal Stück für Stück anschauen.

**Programmbeschreibung:**

Am Anfang finden Sie eine kurze Initialisierungsroutine, die die Farben und den Rand festlegt sowie den Bildschirm löscht. Dann stoßen wir als erstes auf das rechtwinklige Dreieck. In Zeile 100 sehen Sie dort einen bisher noch nicht besprochenen Befehl, das Kommando

```
ORIGIN
```

**Listing 7:** *Programm DREIECKE*

```
10 ' ***************
20 ' ** Dreiecke **
30 ' ***************
40 INK 0,0:INK 1,24:INK 2,11:INK 3,6
50 BORDER 0
60 CLS
70 '*****************************
80 '** rechtwinkliges Dreieck **
90 '*****************************
100 ORIGIN 56,290
110 DRAWR 210,0,2
120 MOVE 0,0:DRAWR 150,86:DRAW 210,0
130 LOCATE 5,9:PRINT"rechtwinklig"
140 '*********************************
150 '** gleichschenkliges Dreieck **
160 '*********************************
170 ORIGIN 388,290
180 DRAWR 200,0,3
190 MOVE 0,0:DRAWR 100,100:DRAWR 100,-100
200 LOCATE 24,9:PRINT"gleichschenklig";
210 '*****************************
220 '** gleichseitiges Dreieck **
230 '*****************************
240 ORIGIN 56,60
250 DRAWR 200,0
260 DRAWR -100,150:DRAWR -100,-150
270 LOCATE 5,24:PRINT"gleichseitig"
280 '************************
290 '** normales Dreieck **
300 '************************
310 ORIGIN 388,60:DRAWR 200,0,2:DRAWR 50,120:DRAWR -250,-120
320 LOCATE 29,24:PRINT"normal"
330 IF INKEY$="" THEN 330
```

Dieses legt den Bildschirmbezugspunkt fest. Was heißt das? Wir haben eben bei der Einleitung zu diesem Kapitel gesagt, daß der CPC mit Werten von 0 bis 639 in der Waagerechten und von 0 bis 399 in der Horizontalen ein Koordinatensystem aufgebaut hat, dessen Mittelpunkt in der linken unteren Ecke liegt. Mit ORIGIN können Sie nun diesen Mittelpunkt verschieben. Wenn Sie also eintippen

```
ORIGIN 320,200
```

so ist Ihr Bildschirmzentrum jetzt auch im Zentrum des Koordinatennetzes. Der Punkt (0,0) befindet sich also jetzt in der Bildschirmmitte. Dementsprechend haben Sie es natürlich nun auch mit negativen Koordinaten zu tun. Wenn Sie früher

```
PLOT 320,200
```

eingegeben haben, so heißt es jetzt,

```
PLOT 0,0
```

und wollten Sie

```
PLOT 0,200
```

eingeben, was dem äußersten linken Punkt auf der mittleren Bildschirmzeile entspricht, so heißt dieses Kommando jetzt

```
PLOT -200,0
```

ORIGIN ist immer dann sinnvoll anwendbar, wenn man mehrere Befehle in demselben Koordinatenrahmen ausführen will. Zum Beispiel wäre es lästig, zu unserem Dreieck immer noch eine absolute X- und eine absolute Y-Verschiebung addieren zu müssen, damit dieses nicht im Bildschirmzentrum oder in der linken unteren Ecke auftaucht, sondern eben im oberen linken Bildschirmviertel, wo wir das erste Dreieck darstellen wollen.

Dazu verschieben wir einfach den Koordinatenfußpunkt mit ORIGIN auf (56,290) was einer Linie im obersten Viertel unseres Bildschirms entspricht. Dort ziehen wir nun unsere Dreiecke. Als erstes wird die Grundlinie gezeichnet mit der Länge 210 und, was wir noch nicht besprochen haben, mit der Farbangabe 2.

Die ersten beiden Parameter bei DRAW und PLOT beinhalten, wie schon gesagt die Koordinaten, beziehungsweise die relativen Abstände. Als dritte Angabe, und zwar wahlweise (sie kann also auch weggelassen werden) können wir eine Farbe, genauer ein zu verwendendes Farbregister, angeben. Wird diese nicht spezifiziert, so wird mit der bisher definierten Farbe (beim Einschalten INK 1) weitergemalt.

Nachdem die Grundlinie gezogen ist, bewegen wir den Grafikcursor mit MOVE wieder zurück auf den Fußpunkt. Als nächstes ziehen wir dann eine Linie mit dem Abstand 150 in X- und 86 in Y-Richtung. Damit haben wir den dritten Punkt unseres Dreiecks erreicht. Mit

```
DRAW 210,0
```

schließen wir unseren Linienzug. Dieser Dreieckstyp zeigt dabei sehr schön, wie man die Kommandos DRAW und DRAWR in vollkommen unterschiedlichen Anwendungen miteinander kombinieren kann. Es wäre hier relativ schwierig gewesen, den dritten Eckpunkt als absolute Koordinate umzurechnen und direkt mit DRAW anzusteuern. Wenn jedoch die relative Verschiebung bekannt ist, kommt man mit DRAWR relativ günstig an den gewünschten Punkt.

Den zweiten Basiseckpunkt kennen wir dagegen. Er hat die Koordinaten (210,0), auf den ersten Fußpunkt des Dreiecks bezogen. Wir können daher nun die dritte Linie relativ einfach mit

```
DRAW 210,0
```

zeichnen. Der nachstehende LOCATE-Befehl gibt nur noch den Dreieckstyp unterhalb des ersten Dreiecks aus.

Auch die anderen Dreiecke werden nach demselben Prinzip gezeichnet. ORIGIN legt immer wieder die linke Basisecke fest, es folgen MOVE-, DRAW- und DRAWR-Kommandos, die die Linien zeichnen. Sie sollten nun ruhig einmal mit den Parametern spielen, auch wenn dabei möglicherweise die Dreiecke zu etwas anderem als geometrischen Figuren werden, nur um sich einmal mit der Technik und vor allem mit der Koordinatenwahl in verschiedenen Koordinatennetzen etwas näher vertraut zu machen.

### Rechtecke und Quadrate

Nach den Dreiecken stellt uns auch die Erweiterung auf eine weitere Ecke vor keine besonders großen Probleme. Im Gegenteil. Wir wollen hier nur auf zwei besonders häufig vorkommende Grundformen eingehen, das Rechteck und das Quadrat. Um ein Rechteck zu zeichnen, benötigen wir zwei Seitenlängen. Beim Quadrat dagegen genügt eine Angabe.

### Beispiel 1: Rechteck

Die Zeichnung mit DRAW ist denkbar einfach. Wir benutzen hierzu am besten die relative Adressierung. Nehmen wir zum Beispiel an, es sollte ein Rechteck mit den Kantenlängen 200 und 100 gezeichnet werden, so ist dies mit den folgenden vier Befehlen möglich.

```
DRAWR 200,0
DRAWR 0,100
DRAWR -200,0
DRAWR 0,-100
```

und schon ist unser Rechteck fertig.

**Beispiel 2: Quadrat**

Hier ist alles noch einfacher, da ja nur eine Seitenlänge verarbeitet werden muß. Die Befehle lauten bei einer Kantenlänge von beispielsweise 150

```
DRAWR 150,0
DRAWR 0,150
DRAWR -150,0
DRAWR 0,-150
```

### 3.1.2 Vielecke und Kreise

Wir wollen nun einmal die Anzahl der darstellbaren Ecken in unseren Figuren ausdehnen und ein Programm entwickeln, das es uns ermöglicht, ein beliebiges Vieleck auf dem Bildschirm darzustellen, vorausgesetzt jede Seite ist genau gleich lang.

**Listing 8:** *Programm VIELECK*

```
10 ' ************************
20 ' ** Vieleck zeichnen **
30 ' ************************
40 CLS
50 INPUT"Seitenzahl";s
60 ORIGIN 320,200
70 PLOT 150,0
80 FOR i=1 TO s
90 x=150*COS(2*PI*i/s)
100 y=150*SIN(2*PI*i/s)
110 DRAW x,y
120 NEXT i
```

**Programmbeschreibung:**

Die Technik dieses Programms knüpft an das Vorhergehende an. Mit ORIGIN legen wir wieder den Mittelpunkt unseres Koordinatensystems, der genau im Zentrum unseres Vielecks sitzt, fest. In Zeile 70 wird dann ein Anfangspunkt geplottet. Der Abstand, das heißt der Radius unseres Vielecks beträgt dabei 150. Nun folgt eine Schleife (Zeile 80-120), die die einzelnen Seiten unseres Diagramms zieht.

Die Berechnung erfolgt dabei mittels Cosinus und Sinus, im Endeffekt also mit einer abgewandelten Kreisformel. Der einzige Unterschied besteht darin, daß hier nicht der Umfang eines Kreises dargestellt wird, sondern die Sehnen durch Kreisbögen genau gleicher Länge gezogen werden. Dies gilt jedoch nur für kleinere Werte von s.

Läßt man die Seitenzahl gegen unendlich streben, so schrumpft die Länge unserer Sehnen, bis sie schließlich nur noch die Länge eines Punktes beträgt: Wir erhalten aus unserem Vieleck einen Kreis. Sie können dies einmal mit verschiedenen Seitenzahlen ausprobieren, und gegebenenfalls auch den einen oder anderen Parameter variieren.

Etwas wird Ihnen allerdings ziemlich schnell bei dieser Methode auffallen und das ist die relativ lange Rechenzeit. Der CPC benötigt nämlich einige Zeit, um Cosinus- oder Sinus-Funktionen zu berechnen. Besonders wenn man einen exakten Kreis darstellen will, bietet es sich daher an, auf eine etwas andere Methode auszuweichen: Die Bestimmung der Umfangs-Koordinaten mit der Wurzelfunktion.

Wir wollen uns dies einmal am Beispiel einer Ellipse anschauen. Diese unterscheidet sich vom Kreis im wesentlichen dadurch, daß in einer

Richtung die Ausdehnung immer mit einem konstanten Betrag multipliziert wird. Es wird also eine mutwillige Verzerrung vorgenommen. Dadurch wird der Kreis in dieser Dimension entweder auseinandergezogen oder gepreßt. Den Ausgangspunkt unserer Überlegung bildet daher auch ein einfacher Kreis, aus dem sich dann die Ellipse relativ leicht ableiten läßt.

Betrachten Sie dazu einmal Bild 3.1. In diesen Kreis haben wir ein rechtwinkliges Dreieck eingezeichnet. A ist dabei der Radius und gleichzeitig auch die Grundseite unseres Dreiecks. Die anderen beiden Seiten x und y schließen einen Winkel von 90 Grad. X ist dabei bekannt. Es wird von uns von 0 bis a beziehungsweise in der umgekehrten Richtung variiert. Wir wollen nun die zugehörigen Y-Werte bestimmen. Dies geht relativ einfach mit der Gleichung des Pythagoras, die einen Zusammenhang über die Seiten im rechtwinkligen Dreieck liefert. Sie lautet auf unseren Fall angewandt

$$a^2 = y^2 + x^2$$

Was sich nach y aufgelöst als

$$y = \sqrt{a^2 - x^2}$$

schreiben läßt. Doch keine Angst. Damit haben wir schon die ganze Mathematik, die wir in unserem Fall benötigen, hinter uns. Wir können jetzt den gesuchten Y-Wert zu jedem gegebenen x bestimmen und damit unseren Kreis zeichnen. Doch sollten wir dabei ökonomisch vorgehen. Daher noch kurz ein paar Überlegungen zum Ausdruck der Werte.

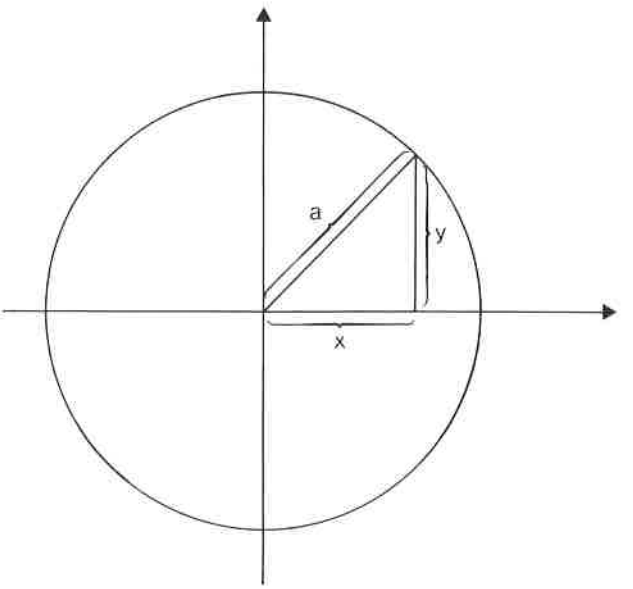



**Bild 3.1:** *Bestimmung der Kreiskoordinaten*

Wie Ihnen sicherlich sofort auffällt, könnten Sie dieses Dreieck an der X-Achse spiegeln und eine Ecke des Dreiecks würde dann wiederum auf einem Kreispunkt landen. Analoges gilt, wenn Sie an der Y-Achse unseres Koordinatensystems spiegeln und zwar für beide Dreiecke, sowohl für das Ursprungsdreieck, wie auch dasjenige, welches wir gerade an der X-Achse gespiegelt hatten. Unser Y-Wert gilt also für vier verschiedene Wertepaare gleichzeitig; wenn er unterhalb der X-Achse liegt allerdings mit negativem Vorzeichen.

Da aber die Bestimmung von y mit der Wurzel-Gleichung den größten Rechenaufwand und damit auch die längste Zeit benötigt, sollten wir die Anzahl der Berechnung auf ein Minimum beschränken. Wir werden daher gleichzeitig alle vier Punkte plotten, was auch relativ einfach möglich ist, indem wir das Zentrum unseres Koordinatensystems wiederum in den Kreismittelpunkt legen. Wir erhalten dann das folgende kleine Ellipsenprogramm.

**Listing 9:** *Programm ELLIPSE*

```
10 ' **************
20 ' ** Ellipse **
30 ' **************
40 CLS:ORIGIN 320,200
50 INPUT"Laenge der Halbachse";a
60 INPUT"Exzentrik";e
70 CLS
80 FOR i=-a TO 0
90 y=e*SQR(a^2-x^2)
100 PLOT x,y:PLOT x,-y:PLOT -x,y:PLOT -x,-y
110 NEXT
```

**Programmbeschreibung:**

In Zeile 40 wird zunächst der Bildschirm gelöscht und der Mittelpunkt unseres Koordinatensystems mit ORIGIN auf das Bildschirmzentrum definiert. Es wird die Länge der Halbachse abgefragt. Wenn wir einen Kreis abbilden wollen, ist dies gleichzeitig der Radius. In Zeile 60 kommt dann die Exzentrik zur Sprache. Für einen Kreis wäre hier eine 1 einzugeben. Ein anderer Input liefert eine Ellipse.

Nach nochmaligem Löschen des Bildschirms geht es dann in die eigentliche Darstellungsschleife ab Zeile 80. Sie läuft von -A bis 0, also über ein Viertel beziehungsweise die Hälfte unseres Kreises. Mit Hilfe der Pythagoras-Gleichung wird Y bestimmt und mit dem Dehnungs- beziehungsweise Streckungsfaktor E für die Exzentrik multipliziert.

Es folgt das vierfache parallele Plotten, bevor dann die nächsten vier Punkte in Angriff genommen werden. Um sich einmal damit vertraut zu machen, welcher PLOT-Befehl nun eigentlich was leistet, sollten Sie diese nacheinander eingeben.

Wenn Sie mit dem ersten beginnen, erhalten Sie den linken oberen Teil unseres Kreisbogens. Das zweite PLOT-Kommando stellt die Spiegelung an der X-Achse dar. Hier wird also der Kreisbogen im dritten Quadranten gezogen. Analog bringen die anderen beiden Plot-Kommandos die Punkte für den ersten und zweiten Quadranten auf den Schirm.

Einen unangenehmen Punkt werden Sie allerdings bei dieser Art der Berechnung auch ziemlich schnell festgestellt haben. In der Nähe der X-Achse liegen nämlich die Punkte relativ weit auseinander, so daß der

Kreisbogen nicht mehr geschlossen wird. Dies rührt daher, daß in diesem Bereich die Y-Werte verhältnismäßig schnell ansteigen.

Hier hilft nur eines: Wir müssen auf eine kleinere Schrittweite ausweichen. In unserer FOR-TO-Schleife haben wir bis jetzt ohne die Angabe des Parameters STEP gearbeitet, so daß der Computer in Einzelschritten vorgegangen ist. Wenn wir nun Zeile 80 abändern, und

```
80 FOR x=-a TO 0 STEP 0.1
```

eingeben, so schließt sich der Kreis. Dieses Verfahren hat allerdings auch eine unangenehme Seite. Jetzt benötigt der Schneider nämlich zehnmal solange um den Kreis fertigzustellen. Durch die kleinere Schrittweite müssen ja zehnmal so viele Berechnungen durchgeführt werden, was speziell in den weiter von der X-Achse entfernten Kreispassagen überhaupt nicht notwendig wäre.

Wenn wir daher zwischen Zeichengeschwindigkeit und exakter Darstellung ein Optimum erreichen wollen, so können wir nicht mehr mit einer dermaßen einfachen Schleife arbeiten, sondern wir müssen für jeden Schritt auch die Schrittweite neu dimensionieren. In diesem Fall ist es nicht mehr möglich mit FOR-TO zu arbeiten, sondern wir müssen uns unsere eigene Ersatz-FOR-TO-Schleife basteln. Diese würde in unserem Fall wie folgt aussehen:

```
75 x=-a:s=0.1
80 x=x+s:IF x>0 THEN END
110 s=(1+10/a)*s:GOTO 80
```

Die Berechnung in Zeile 80 sorgt nun dafür, daß unsere Schrittweite von 2 Angaben abhängig wird. Zum einen von der Länge unserer Halbachse. Je größer wir nämlich unsere Ellipse beziehungsweise unseren Kreis wählen, in desto feineren Schritten müssen wir die Schrittweite erhöhen, um noch permanent eine Deckung der Umfangslinie zu erreichen.

Daneben erfolgt ja fortlaufend eine Multiplikation, so daß die Schrittweite desto größer wird, je näher sich x auf den Mittelpunkt zu bewegt. Beim ersten Schritt ist die Variable S durch Initialisierung in Zeile 75 auf den Wert 0.1 gesetzt worden. Es wird also um ein Zehntel weitergeschritten. Nehmen wir nun einmal an, wir hätten die Länge der Halbachse A=100 eingegeben, so erfolgt der nächste Schritt mit

```
(1+10/100)*S = 0.11
```

Mit diesem kleinen praktikablen Trick haben wir es also geschafft, uns dem Darstellungsoptimum schon relativ weit anzunähern, und trotz einer

brauchbaren Schnelligkeit dennoch die gewünschte optimale Auflösung zu erhalten.

## 3.3 Aufbruch in neue Dimensionen - dreidimensionale Darstellung

Wir können natürlich unsere geometrischen Figuren nun auch dreidimensional oder besser gesagt perspektivisch darstellen. Dies erfordert allerdings einen erheblichen Mehraufwand im Vergleich zur 2-D-Darstellung. Allerdings winkt auch ein reizvoller Lohn für unsere Arbeit, nämlich eine wesentlich verbesserte, eindrucksvollere Wirkung unserer Figuren.

Wir wollen einmal an zwei Beispielen zeigen, wie man eine perspektivische Darstellung von Figuren erreichen kann.

Dazu gibt es zwei grundsätzliche Darstellungstypen, das Gittermodell und die Vollgrafik. Bei einem Gittermodell werden die Begrenzungs- und Decklinien eines Körpers gezogen. Der Rest des Modells bleibt frei. Normalerweise zieht man dabei auch die sonst unsichtbaren, durch den Körper verdeckten Eckkanten.

Im anderen Fall, bei der Vollgrafik, wird dieses Grafik-Modell ausgemalt. Es treten nur die Flächen auf, die man auch normalerweise bei einer Betrachtung des Körpers sehen würde. Die unterschiedlichen Flächen sind dabei meist mit unterschiedlicher Farbe gezogen, um den räumlichen Aspekt noch zu vertiefen. Als Beispiele wollen wir hier einmal Quader und Zylinder betrachten.

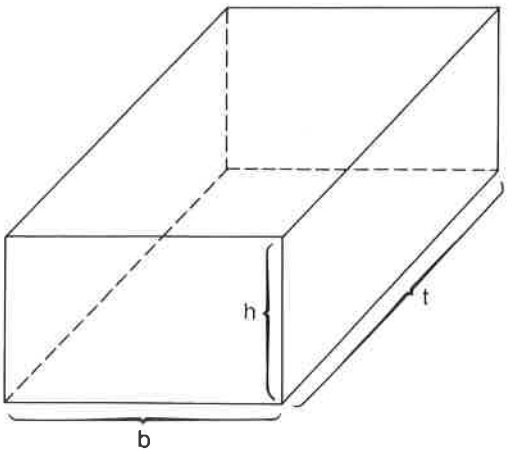



**Bild 3.2:** *Gittermodell eines Quaders*

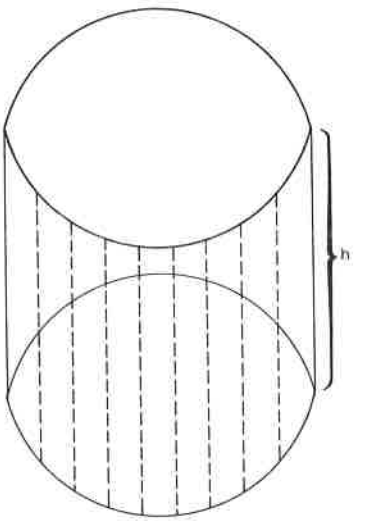



**Bild 3.3:** *Gittermodell eines Zylinders*

Wie die beiden Varianten sich als Figuren ausnehmen, zeigen die Abbildungen 3.2 und 3.3. In beiden Fällen wurde das Gittermodell gezeichnet. Wenden wir uns zunächst dem einfacheren Modell des Quaders zu.

Das Hauptproblem einer jeden dreidimensionalen Darstellung ist die Perspektive. Der wohl wichtigste Punkt dabei ist die Wahl des Winkels, unter dem die nach hinten laufenden Linien erscheinen. Wird er falsch gewählt, so erscheint die ganze Figur künstlich und unwirklich.

Bei kleineren Figuren, das heißt, falls die Tiefe nicht besonders groß ist, kann man hier problemlos in einem Winkel von 45 Grad operieren, ansonsten muß man gegebenenfalls auf geringere Winkelgrade (30 Grad oder auch 20 Grad) ausweichen. Einige kurze Experimente zeigen hier dann meist den richtigen Weg.

### 3.3.1 3-D-Quader als Gittermodell und Vollgrafik

Wir wollen zunächst einmal die Variante eines Draht- oder Gittermodells als den einfacheren Fall analysieren. Zur Abbildung genügen hier simple DRAW-Kommandos unterschiedlicher Länge und Richtung.

In unserem Beispiel werden wir mit einem Winkel von 45 Grad arbeiten, der sich auch mit DRAWR relativ einfach erzeugen läßt. Eine waagerechte Linie erhalten wir, indem wir die Änderung in der Y-Richtung auf 0 setzen und nur eine Verschiebung um X vollziehen. Analog dazu ergibt sich die senkrechte Linie mit gleich gehaltenem X und verändertem Y. Die perspektivisch nach hinten laufende Linie benötigt ein gleichförmiges Ändern von X und Y, um den gewünschten Effekt zu erzielen.

Etwas schwieriger wird es dagegen, wenn wir auf einer Vollgrafik bestehen, das heißt falls die Seiten unseres Quaders voll "ausgemalt" werden sollen. Wenn Sie über einen CPC 664 verfügen, ist es natürlich kein Problem. Sie wenden hier einfach das FILL-Kommando an.

Ansonsten müssen wir, da wir über keinerlei Befehle zum Ausfüllen von Flächen verfügen, uns hier mit einer ganzen Reihe nebeneinandergezogener Linien behelfen. Wenn wir diese jedoch in eine Schleife setzen, ist auch dies kein Problem mehr. Wir müssen nur einfach über die gesamte Breite Linie neben Linie zeichnen. Gleiches gilt für die Deckfläche und die eine (rechte) Seite. Schauen wir uns nun einmal das zugehörige Programm an.

**Programmbeschreibung:**

Am Anfang der Routine "Quader zeichnen" finden Sie die Auswahl nach dem Darstellungsmodus. Wahlweise als Gittermodell oder als Vollgrafik kann unser Quader gezeichnet werden. Ab Zeile 160 treffen wir zunächst auf die erste Variante, das Gittermodell. Zunächst werden Breite, Höhe und Tiefe unseres Quaders abgefragt und unter die zugehörigen Variablen eingeladen.

Es folgt eine Reihe von DRAWR-Befehlen, die den Quader zeichnen. Interessant sind dabei insbesondere in Zeile 230 und 250 die Linien für die schräg laufenden Seitenbegrenzungen. Dadurch, daß die Bewegung jeweils mit identischer Verschiebung in X- und Y-Richtung erfolgt, ergibt sich der schräge Verlauf. Ansonsten werden dieselben Techniken angewandt, die wir schon bei den einfachen, zweidimensionalen Figuren kennengelernt haben.

Ab Zeile 310 kommen wir dann zur Vollgrafik. Zunächst wird in Zeile 370-390 auf die Vorderfront unseres Quaders gezeichnet. Die Schleife läuft hier von 0 bis b, das heißt über die gesamte Breite unserer Vorderfront in Einer-Schritten. Dabei wird jeweils der Fußpunkt mit MOVE gesetzt, bevor dann ein relatives DRAW um die Höhe der Front mit der Farbe 2 (in unserem Fall ist das rot) erfolgt.

Die Zeilen 410-430 bilden die Seitenfläche ab. Die Berechnungsgrundlage ist hier etwas schwieriger, der Effekt allerdings derselbe. Den Fußpunkt der unteren Begrenzungslinie erreichen wir, indem wir zu (b,0) jeweils den Vektor (i,i) addieren und dabei i über die gesamte Tiefe von 0 bis t laufen lassen. Von diesem Fußpunkt werden nun wieder Linien der Höhe h mit der Farbe 3, also hellblau, nach oben gezogen.

Die Deckplatte unseres Quaders wird dann mit Zeile 440-460 gezeichnet. Hier ziehen wir unsere Flächenlinien nicht nach oben, sondern nach rechts. Dazu wird der Grafikcursor auf einen Punkt auf der äußeren linken Seitenlinie bewegt und dann eine Linie mit der Breite b und der Farbe 1 gezogen.

Am Abschluß unseres Programms steht dann eine endlose Abfrageschleife, die in Zeile 470 so lange den Bildschirm unverändert läßt, bis eine Taste gedrückt wurde. Dies hat den Vorteil, daß man nicht immer durch ein störendes "Ready" oder einen Zeilenvorschub, eine Änderung des Bildschirms bekommt.

## **Listing 10:** *Programm QUADER ZEICHNEN*

```
10 '**********************
20 '** Quader zeichnen **
30 '**********************
40 MODE 1
50 INK 0,0:INK 1,21:INK 2,6:INK 3,11
60 BORDER 0
70 PAPER 0:PEN 1
80 CLS
90 PEN 3:PRINT"      Q U A D E R   Z E I C H N E N"
100 LOCATE 12,4:PRINT"Gittermodell  (1)"
110 LOCATE 12,6:PRINT"Vollgrafik    (2)"
120 z$=INKEY$:IF z$<>"1" AND z$<>"2" THEN 120 ELSE IF z$="2" THEN
 300
130 '********************
140 '** Gittermodell **
150 '********************
160 CLS:ORIGIN 50,50
170 INPUT"Breite";b
180 INPUT"Hoehe";h
190 INPUT"Tiefe";t
200 CLS
210 DRAWR b,0,1:DRAWR 0,h:DRAWR -b,0:DRAWR 0,-h
220 MOVER b,0
230 DRAWR t,t:DRAWR 0,h:DRAWR -b,0:DRAWR -t,-t
240 MOVER b,0
250 DRAWR t,t
260 MOVE 0,0:DRAWR t,t:DRAWR b,0:MOVER -b,0:DRAWR 0,h
270 GOTO 470
280 '******************
290 '** Vollgrafik **
300 '******************
310 CLS
320 INPUT"Breite";b
330 INPUT"Hoehe";h
340 INPUT"Tiefe";t
350 ORIGIN 50,50
360 CLS
370 FOR i=0 TO b
380 MOVE i,0
390 DRAWR 0,h,2
400 NEXT i
410 FOR i=0 TO t
420 MOVE b+i,i:DRAWR 0,h,3
430 NEXT i
440 FOR i=0 TO t
450 MOVE 0+i,h+i:DRAWR b,0,1
460 NEXT
470 IF INKEY$="" THEN 470 ELSE RUN
```

### 3.3.2 3-D-Zylinder

Wenden wir uns nun unserer zweiten Figur zu, dem Zylinder. Hier verwenden wir eine Reihe von geometrischen Figuren, die wir bereits kennen. Die Rede ist hier von der Ellipse, die zumindest in unserem Gittermodell schon relativ einfach die obere und untere Grundplatte bildet. Zwei weitere mit DRAW gezogene Linien vollenden den Rest.

Etwas schwieriger wird es dagegen, wenn wir eine Vollgrafik verlangen. Hier muß zwar nur noch eine Ellipse dargestellt werden, die obere Deckplatte. Dafür ist diese aber vollständig auszumalen. Dies ist nun nicht möglich, indem wir jeweils vom Mittelpunkt einen Strahl auf einen Punkt der Kreislinie senden.

Bei Anwendung dieses Verfahrens bleiben nämlich immer noch an irgendwelchen Stellen kleine Punkte in der Deckplatte ohne Farbbelegung, es sei denn man wählt den Abstand zwischen den einzelnen Punkten auf dem Umfang so eng, daß das Zeichnen der Figur sehr lange dauert.

Hier wird daher ein anderes Verfahren angewandt. Wenn wir einmal davon ausgehen, daß das Zentrum unserer Deckplatte den Mittelpunkt eines Koordinatensystems bildet, so können wir eine gute Flächendeckung erreichen, indem wir von jedem Punkt unseres Umfangs eine Verbindung zur Y-Achse, die gleichzeitig parallel zur X-Achse verläuft, ziehen.

Diese übereinander gestapelten Linien decken dann unsere Fläche vollständig ab. Damit haben wir alle wesentlichen Punkte angerissen, die bei einem solchen Programm zu beachten sind.

**Programmbeschreibung:**

Am Anfang finden sich die schon gewohnten Abfragen nach der Darstellungart. Das Gittermodell ist relativ leicht beschrieben. Die Ellipsengleichung, und ihre Darstellung kennen wir schon. Es werden nun zusätzlich zu den vier, normalerweise in einer zweidimensionalen Ellipse gesetzten Punkte gleichzeitig mit der Verschiebung h in Y-Richtung vier weitere Punkte für die untere Deckplatte gesetzt.

Etwas außergewöhnlich ist dabei Zeile 230. Sie wiederholt nämlich denselben Vorgang noch einmal, jedoch mit X-Werten, die jeweils um 1 höher liegen, als ihre Vorgänger in Zeile 220. Dies hat einen relativ einfachen Effekt. Unsere Ellipse wird nämlich noch einmal mit einer Verschiebung um 1 in X-Richtung gezeichnet. Ergebnis: Die Umfangslinie wird doppelt so dick.

Dasselbe Verfahren wenden wir dann bei den senkrechten Seitenlinien an. Sie werden jeweils mit der notwendigen Verschiebung um den Radius A rechts und links doppelt nebeneinander gezeichnet.

Etwas interessanter wird es da schon, wenn wir die Vollgrafikvariante betrachten. Am Anfang stehen wieder das Löschen des Bildschirms und die Eingabe der zwei benötigten Parameter für Radius und Höhe. Der Mittelpunkt unseres Koordinatensystems wird auf das Bildschirmzentrum gesetzt (Zeile 360).

Dann beginnen die eigentlichen Darstellungszeilen. Wir beginnen dabei am äußersten linken Rand unserer Figur. X weist also den Wert -a auf, und y dementsprechend 0. Der Schrittparameter s wurde in Zeile 380 auf 0.1 gesetzt. Er legt fest, in welchen X-Schritten wir bei unseren Berechnungen fortschreiten.

Die eigentliche Schleife läuft nun von Zeile 400-470 einschließlich. Schauen wir uns an, was in den einzelnen Stufen passiert. Zunächst einmal wird x um den Schrittparameter erhöht. Beim ersten Durchlauf wird allerdings durch das vorherige Abziehen in Zeile 390 die Erhöhung wieder zunichte gemacht. Wir operieren also mit x=-a.

Als nächstes wird abgefragt, ob x größer oder gleich 0 sei. Dies wird als Aussprungskriterium aus dieser Routine benutzt. In diesem Fall haben wir nämlich den Mittelpunkt unserer Figur erreicht, und da wir ja alle vier Viertel unserer Deckplatte parallel zeichnen, ist in diesem Fall die Figur dann auch vollendet.

Beim eigentlichen Zeichnen unserer Ellipsenviertel bestimmen wir zunächst in Zeile 410 den Y-Wert mit Hilfe des Pythagoras. Als Wert für die Exzentrizität unserer Ellipse wurde hier 0.5 eingesetzt. Es folgen die eigentlichen DRAW-Zeilen. Dabei wird der Grafikcursor immer auf der Y-Achse (x=0) auf den gerade errechneten Y-Wert gesetzt. Dann erfolgt das parallele Ziehen der Linien bis zum Punkt auf dem Umfang.

Zeile 460 beinhaltet wieder die etwas trickreiche Veränderung des Schrittparameters S. Wir hatten bei unserer Betrachtung der zweidimensionalen Grafik schon gesehen, daß es ziemlich problematisch ist, beim Zeichnen einer Figur und hier speziell beim voll ausgefüllten Zeichnen, zwischen einer optimalen Deckung und einem möglichst schnellen Programmdurchlauf, zu entscheiden.

Der Grund war relativ einfach. Wenn man vom äußersten linken Rand unserer Figur ausgeht, so bewirken bereits relativ kleine Änderungen von X eine ziemlich große Änderung des Y-Wertes. Je mehr man dagegen

dem Mittelpunkt nahe kommt, in desto größeren Schritten könnte man eigentlich X variieren, da sich der Abstand von der X-Achse, das heißt der Wert für Y doch geringfügig erhöht.

Am äußersten linken Rand dagegen muß natürlich mit relativ kleinen Schritten vorangegangen werden. Denn wir wollen ja mit einer Änderung von X jede Bildschirmlinie erreichen. Wenn wir hier einen Ausfall zu verzeichnen haben, so äußert sich dies in vier schwarzen Linien auf dem Schirm genau im Abstand Y.

Neben diesen grundsätzlich für jeden Kreis oder jede Ellipse geltenden Problem ist allerdings noch auf eine andere Tatsache hinzuweisen. Wir müssen nämlich X auch noch im Verhältnis zur Größe unserer Figur variieren. Diesen Gesetzmäßigkeiten trägt die schrittweise Erhöhung von S in Zeile 460 Rechnung.

Mit jedem Schritt, den wir uns dem Zentrum unserer Figur nähern, erfolgt die Änderung in immer größeren Schrittweiten. S wird nämlich Schritt für Schritt durch laufende Multiplikation immer weiter vergrößert. Der Faktor, mit dem S dabei bei jedem Schritt multipliziert wird, ist jedoch nicht fix vorgegeben, sondern hängt wiederum vom Radius ab. Je größer der Radius ist, desto kleiner wird dieser Erhöhungsfaktor gewählt und damit setzt der Übergang zu einer größeren Schrittweite erst später ein.

Schauen wir uns nun den Rest unseres Programms an. Er zeichnet die senkrechte Vorderfront. Wir beginnen mit einer Neudefinition des Bezugsrahmens in Zeile 480. Der Fußpunkt unseres Systems wird mit ORIGIN um die Höhe h verschoben, die unser Zylinder aufweisen soll. Nun wenden wir noch einmal unsere Ellipsengleichung an.

Wir bestimmen nämlich den unteren Rand der unteren Deckplatte. Von dieser aus ziehen wir dann, gleichzeitig von rechts und links auf das Zentrum zustrebend, Linien der Höhe h nach oben, die damit genau auf die Unterkante der oberen Deckplatte treffen. Zur Verstärkung des räumlichen Effektes wurden hierbei für Deckplatte und Zylinderumfang auch noch unterschiedliche Farben gewählt.

Den Abschluß des Programms bildet wieder eine endlose Warteschleife, die erst auf Tastendruck zum Programmabbruch beziehungsweise zum wiederholten Anlauf mit RUN führt. Eine sehr schöne Zylinderausgabe erhalten Sie zum Beispiel, wenn Sie mit dem Radius den Wert 75 und für die Höhe 150 oder 200 eingeben.

## Listing 11: *Programm ZYLINDER ZEICHNEN*

```
10 '************************
20 '** Zylinder zeichnen **
30 '************************
40 MODE 1
50 INK 0,0:INK 1,21:INK 2,6:INK 3,11
60 BORDER 0
70 PAPER 0:PEN 1
80 CLS
90 PEN 3:PRINT"   Z Y L I N D E R   Z E I C H N E N
100 LOCATE 12,4:PRINT"Gittermodell  (1)"
110 LOCATE 12,6:PRINT"Vollgrafik    (2)"
120 z$=INKEY$:IF z$<>"1" AND z$<>"2" THEN 120 ELSE IF z$="2" THEN
 320
130 '*******************
140 '** Gittermodell **
150 '*******************
160 CLS:ORIGIN 320,300
170 INPUT"Radius";a
180 INPUT"Hoehe";h
190 CLS
200 FOR x=-a TO 0
210 y=0.5*SQR(a^2-x^2)
220 PLOT x,y,1:PLOT x,-y:PLOT -x,y:PLOT -x,-y:PLOT x,y-h:PLOT x,-
y-h:PLOT -x,y-h:PLOT -x,-y-h
230 PLOT x+1,y,1:PLOT x+1,-y:PLOT -x+1,y:PLOT -x+1,-y:PLOT x+1,y-
h:PLOT x+1,-y-h:PLOT -x+1,y-h:PLOT -x+1,-y-h
240 NEXT x
250 MOVE -a,-h:DRAW -a,0
260 MOVE -a+1,-h:DRAW -a+1,0
270 MOVE a,-h:DRAW a,0
280 MOVE a+1,-h:DRAW a+1,0
290 GOTO 560
300 '*****************
310 '** Vollgrafik **
320 '*****************
330 CLS
340 INPUT"Radius";a
350 INPUT"Hoehe";h
360 ORIGIN 320,300
370 CLS
380 x=-a:s=0.1
390 x=x-s
400 x=x+s:IF x>=0 THEN 480
410 y=0.5*SQR(a^2-x^2)
420 MOVE 0,y:DRAW x,y,1
430 MOVE 0,y:DRAW -x,y
440 MOVE 0,-y:DRAW x,-y
450 MOVE 0,-y:DRAW -x,-y
460 s=(1+10/a)*s
470 GOTO 400
480 ORIGIN 320,300-h
490 FOR x=-a TO 0
500 y=0.5*SQR(a^2-x^2)
510 MOVE x,-y
520 DRAWR 0,h,2
530 MOVE -x,-y
540 DRAWR 0,h
550 NEXT x
560 IF INKEY$="" THEN 560 ELSE RUN
```

Wir sind nun aber bei unserem Ziehen der senkrechten Linien nicht nur auf eine Farbe beschränkt, sondern wir können hier auch farbliche Unterteilungen durchführen. Hier soll nur einmal eine ganz einfache Drittelteilung gezeigt werden. Wir ändern dazu die Zeilen 520 und 540 wie folgt ab. Die Zeilen 520 und 540 erhalten jeweils denselben Wert.

```
520 DRAWR 0,h/3,2
540 DRAWR 0,h/3,2
```

Zusätzlich fügen wir nun noch zwei neue Zeilen ein

```
525 DRAWR 0,2*h/3,3
545 DRAWR 0,2*h/3,3
```

Der Effekt ist sehr eindrucksvoll. Das untere Drittel unserer Säule bleibt weiterhin rot. Darüber hat sich jedoch nun ein hellblauer Halbzylinder aufgebaut.

Wir sind mit einer solchen Darstellung natürlich nicht nur auf zwei Farben oder zwei ins Verhältnis gesetzte Zahlenwerte (in unserem Beispiel ein Drittel und zwei Drittel) beschränkt, sondern können dieselben Überlegungen auch mit viel mehr Farben, speziell im Mode 0, und einer größeren Anzahl von Zahlen, die in ihrem Verhältnis zueinander dargestellt werden sollen, durchführen.

Die Zylinderunterteilung liefert dann ein relativ plastisches Abbild davon, wie die einzelnen Zahlen im Verhältnis zueinander dastehen. Mit derartigen Problemstellungen, der Veranschaulichung von Zahlenwerten, ihrer Änderung im Zeitablauf, ihrer relativen Größe im Vergleich miteinander, beschäftigt sich die darstellende oder auch grafische Statistik, ein Anwendungsbereich, auf den wir nun etwas näher eingehen wollen.

## 3.4 Grafische Statistik

Die grafische Statistik stellt gleichsam das Paradepferd für den Grafikeinsatz dar. Ihre Aufgabe besteht darin, abstrakte Zahlenwerte absolut oder in ihrem Verhältnis zueinander darzustellen, durch figürliche Darstellung besser begreifbar zu machen.

Wir alle kennen die dabei angewandten grafischen Darstellungsmittel aus den täglichen Presseberichten in Fernsehen oder Zeitung. Kein Wirtschaftsbericht kann heute bestehen, ohne das obligatorische Säulen-

diagramm, das die Kennwerte für das letzte halbe Jahrzehnt widerspiegelt und einen grafischen Eindruck von der Entwicklung der diskutierten Größe bietet.

Bei Bundestagswahlen faszinieren immer wieder die schnellen Torten- oder Kreisdiagramme, die, blitzartig auf den Bildschirm gezaubert, die voraussichtliche Sitzverteilung im neuen Parlament wiedergeben.

Die Anwendung grafischer Darstellungsmittel ist allerdings nicht auf Wirtschaft und Politik und auch nicht auf die Verdeutlichung abstrakter Zahlen beschränkt. Schon von der Schule her sind uns noch einige andere Anwendungsmöglichkeiten, speziell im Bereich der grafischen Analyse, vertraut.

Eine der bekanntesten Anwendungen ist dabei das Aufzeichnen einer Funktion über einen größeren Zahlenbereich (Graph), welches dann einen guten Überblick über den Verlauf der Funktion und bestimmter Eigenschaften wie Maxima, Minima, Nullstellen etc. liefert.

Die dabei benötigten Figuren, wie Kreise, Rechtecke und so weiter kennen wir schon. Nur der Zusammenbau dieser Figuren zu Diagrammen stellt noch eine Neuerung dar. Wir wollen dabei die folgenden Darstellungsarten betrachten:

**Das Balkendiagramm:** Bei dieser Darstellungsart werden verschiedene Werte durch verschieden hohe Rechtecke beziehungsweise Balken repräsentiert. Die Breite der Figuren ergibt sich aus der verstrichenen Zeit, falls auf der X-Achse eine Zeiteinteilung angegeben ist, oder ist fix vorgegeben.

**Das Stabdiagramm:** Es basiert auf demselben Prinzip. Auch hier repräsentiert die Höhe der Stäbe die Größe der entsprechenden Werte. Das Stabdiagramm ist immer dann als Darstellungsmittel geeignet, wenn eine Vielzahl von Werten dargestellt werden soll. Wenn nur wenige Stäbe geplottet werden, wirken diese etwas verloren auf dem Bildschirm.

**Der Funktionsgraph:** Diese Darstellungsmethode eignet sich besonders, wenn man es mit einer Vielzahl von Werten zu tun hat. Die einzelnen Meß- (Input-) Größen werden dabei zu einer Funktion verbunden.

**Kreisdarstellungen:** Die Abbildung von Werten durch Kreise und Zylinder eignet sich besonders, wenn man die Aufteilung einer Gesamtheit auf mehrere Teile sichtbar machen will. Wie dies durch Unterteilung einer Zylinderfläche oder durch verschiedenfarbige

Kreissegmente geht, haben wir uns schon angeschaut. Wir werden daher auf diese Darstellungstechniken hier nicht mehr speziell eingehen. Sie können die dazu nötigen Schritte ja selbst in die Hauptprogramme einfügen.

Bei der Entwicklung von Diagrammen müssen wir uns mit drei Problembereichen auseinandersetzen:

- Die Bezugslinie für unsere Diagrammdarstellung. Möglich sind hier Nulliniendarstellungen. Der Durchschnitt der dargestellten Werte kann ebenso als Basislinie dienen, wie beispielsweise das Minimum oder Maximum der Werte.
- Die Beschriftung. Hier werden wir uns damit beschäftigen, wie eine automatische Achsenbeschriftung machbar ist.
- Der Darstellungsmaßstab. Da es sich hierbei wohl um das schwierigste Problem handelt, wollen wir es daher auch zuerst lösen.

### 3.4.1 Grafikhöhe selbst bestimmt - automatische Maßstabswahl

Zunächst ein paar Erläuterungen zur Fragestellung. Worum geht es?

Nehmen wir einmal an, Sie wollen ein Programm schreiben, das es ermöglicht, 10 verschiedene Werte, die in einem Feld w(i) als w(1), w(2)... w(10) abgelegt sind, darzustellen. Wenn der Wertebereich, in dem die darzustellenden Zahlenwerte liegen, nicht fest vorgegeben ist, stellt sich das Problem der Maßstabswahl.

Wählt man einen relativ niedrigen Maßstab (kleiner Maximalwert auf der Y-Achse) und gibt diesen fest vor, so werden zwar die Unterschiede in der Höhe der Balken sehr gut sichtbar, aber es besteht sehr schnell die Gefahr von Bereichsüberschreitung. Die einzelnen Blöcke wachsen dann über den oberen Bildschirmrand hinaus.

Setzen wir den Maßstab dagegen zu hoch an, so haben wir zwar die Gewißheit, daß alle Blöcke richtig abgebildet werden, aber gegebenenfalls sind die Unterschiede nur noch schwach erkennbar und im Extremfall reduzieren sich unsere Blöcke auf kleine Striche an der Bildunterkante.

Wir haben es hier also mit einem Optimum-Problem zu tun und dieses ist vor allem dann, wenn die Werte relativ weit schwanken, nur durch eine automatische Maßstabswahl lösbar. Der optimale Punkt ist dann erreicht,

wenn der größte Wert das für die Darstellung vorgewählte Bildschirmfenster gerade ausfüllt.

Dazu müssen wir das Maximum und gegebenenfalls das Minimum der Funktionswerte kennen. Dies ist mit einer FOR-TO-Schleife und zwei Spezialbefehlen im Schneider-BASIC jedoch relativ einfach machbar. Das Kommando MAX berechnet das Maximum einer Folge von Zahlen. MIN leistet dasselbe in der umgekehrten Richtung. Eine kleine Routine löst dann unser Problem.

```
100 'Massstabswahl; en=max.Anzahl der Werte; Werte in w(i) gespeichert
110 en=10:mi=1E37:ma=-1E37
120 FOR i=1 TO en:ma=MAX(ma,w(i)):mi=MIN(mi,w(i)):NEXT
```

**Programmbeschreibung:**

Nach dem Durchlauf dieser Schleife haben wir das Minimum (mi) und das Maximum (ma) der darzustellenden Werte, die in dem Feld w(i) gespeichert sein sollen, errechnet. Wichtig ist dabei ein kleiner Trick. Die Variablen mi und ma werden nämlich vorab auf einen möglichst großen (beziehungsweise kleinen) Wert definiert (Zeile 110). Setzen Sie einmal ein paar Werte für w(i) ein und springen Sie dann Zeile 100 mit GOTO 100 an. Mi und ma enthalten danach die gewünschten Extrema.

### 3.4.2 Einige einfache Diagramme

Nach dieser Vorarbeit können wir nun an die Arbeit gehen und als erstes einmal ein ganz einfaches Balkendiagramm entwerfen. Dabei sollten Sie im Hinterkopf behalten, was wir zum Zeichnen der einfachen Rechtecke in Kapitel 3.2 gesagt haben. Das Listing trägt den Titel DRAWSAEULE.

**Programmbeschreibung:**

Als Beispiel für die darzustellenden Werte in diesem Diagramm wurde eine Umsatzstatistik gewählt. Nach den üblichen Vorbereitungsarbeiten, wie Einlesen der Texte, Ausgabe des Titels Balkendiagramm, der Farbgebung und der Abfrage der entsprechenden Stromkosten in einer Schleife, kommen wir zunächst zur Bestimmung der Maxima.

Es folgt die Abfrage der Breite der Blöcke. Hier wird festgelegt, wieviele Linien einen Block bilden sollen. Der Bereich sinnvoller Eingaben ist dabei auf die Werte zwischen 5 und 50 beschränkt. Indem Sie an diesem

Parameter spielen, können Sie die Breite variieren und damit ausprobieren, welches Diagramm den besten Eindruck hinterläßt.

Die nächsten Zeilen (ab Zeile 200) bilden die eigentliche Darstellungsroutine. Zunächst setzen wir das Zentrum unseres Systems mit

```
ORIGIN 0,0
```

in die linke untere Ecke und löschen den Bildschirm. Die eigentliche Darstellung geschieht dann auf den nächsten drei Zeilen in der geschachtelten FOR-TO-Schleife. Die äußere Schleife läuft dabei über alle Werte (Monate), die innere über alle Linien eines Blocks.

Für jeden neuen Monat wird zunächst die Farbe festgelegt. Danach wird die Höhe des entsprechenden Blocks aus den von Ihnen eingegebenen Werten und dem vorher festgestellten Maßstab errechnet. Danach geht es in die innere Schleife.

Mittels der Laufvariablen i und j und der vorher eingegebenen Breite wird der Fußpunkt der zu zeichnenden Linie auf der X - Achse errechnet und danach mit dem MOVE-Kommando der Grafik-Cursor auf diese Position bewegt. Der nachfolgende relative DRAW-Befehl zieht dann eine Linie von entsprechender Höhe in der Farbe F. Dieser Vorgang wiederholt sich so lange, bis ein Block mit entsprechender Breite fertiggestellt wurde.

Danach wird die Monatsschleife für den nächsten Monat abgearbeitet und die Prozedur wiederholt sich wieder. Den Abschluß des Programms bildet dann ein sogenanntes ENDLOSES GOTO.

Diese Zeile hat zur Folge, daß der Computer, sobald er auf sie stößt, in ihr hängen bleibt, da er den GOTO-Befehl laufend ausführt. Dies hat den Zweck, das störende READY am Programmende zu unterdrücken. Wollen Sie das Programm also beenden, so müssen Sie auf die ESC-Taste drücken.

**Listing 12:** *Programm DRAWSÄULE*

```
10 REM ****************************
20 REM ** Saeulengrafik mit DRAW **
30 REM ****************************
40 INK 0,27:INK 1,21:PAPER 0:PEN 1:BORDER 0
50 DATA Januar,Februar,Maerz,April,Mai,Juni
60 DATA Juli,August,September,Oktober,November,Dezember
70 DIM m$(12),m(12)
80 FOR i=1 TO 12:READ m$(i):NEXT i
90 CLS
100 FOR i= 1 TO 12
110 PRINT"Bitte geben Sie die  Umsaetze  fuer den Monat ";
120 PRINT m$(i);" ein";:INPUT m(i)
130 NEXT i
134 REM ******************
135 REM ** Maximalwert **
136 REM ******************
140 ma=0:mi=10000000
150 FOR i= 1 TO 12:ma=MAX(ma,m(i)):mi=MIN(mi,m(i)):NEXT i
160 CLS
170 LOCATE 7,2:PRINT"B A L K E N D I A G R A M M"
180 PRINT"Bitte geben Sie die Breite der Bloecke  an (5-50)";:INP
UT b
190 IF b<5 OR b>50 THEN PRINT"Fehler":GOTO 180
200 CLS:LOCATE 7,2:PRINT"B A L K E N D I A G R A M M"
210 INK 0,0:INK 1,2:INK 2,15:INK 3,21:BORDER 0
220 ORIGIN 280-6*b,0
230 FOR i= 1 TO 12
240 f=(i MOD 3)+1
250 h=m(i)/ma*250
260 FOR j=1 TO b:MOVE b*i+j,80:DRAWR 0,h,f
270 NEXT j,i
280 PEN 2
290 GOTO 290
```

Wenden wir uns nun den anderen Darstellungsarten zu. Wie können wir nun ein Stabdiagramm erzeugen. Der einzige Unterschied zum Balkendiagramm besteht darin, daß sich nun nicht die verschiedenen Blöcke untereinander abdecken, die insgesamt die volle Breite einnehmen, sondern, daß sich Stäbe und Hintergrundfarbe abwechseln.

Was auf den ersten Blick schwieriger aussieht, ist in Wirklichkeit jedoch eine Vereinfachung des eben betrachteten Programms. Löschen wir den Bildschirm, und zeichnen wir statt eines ganzen Blockes nur eine Linie, so erhalten wir genau diesen Effekt. Wenn Sie also in Zeile 260 j nur noch von 1 bis 1 laufen lassen, erhalten Sie genau diese Ausgabe. Sie können j aber auch über zwei Punkte laufen lassen. Der Stab wird dann deutlicher.

Wenden wir uns nun dem letzten grafischen Darstellungsmittel zu, dem Funktionengraph. Wir werden auf diese Darstellungsweise nur kurz schematisch eingehen, da auch sie wieder nur eine Variante desselben Themas darstellt. In unserem Programm "Stabdiagramm" brauchen wir nämlich nur die Zeile mit dem DRAWR durch

```
DRAW 50 * i,h:NEXT i
```

zu ersetzen. Statt Stäbe von der Grundlinie aus zu ziehen, verbindet das Programm nun die einzelnen Funktionspunkte durch Linien. Ein einfacher Funktionengraph ist fertig. Allerdings ist dies nur eine Methode. Man kann auch die Verbindungslinien weglassen und statt des DRAW

```
PLOT
```

verwenden. Während DRAW eine Linie zieht, setzt PLOT nur einen Punkt. Dies führt zu einer etwas anderen Ausgabe. Besonders, wenn die Funktion mit nur wenigen Werten definiert wurde und damit auch relativ lange Linien gezogen werden müssen, ist der Funktionengraph auch sichtbar aus mehreren geraden Teilstücken zusammengesetzt, was gegebenenfalls etwas störend wirkt. Hier hat die Einzelpunktdarstellung Vorteile.

Andererseits wird durch die Linienführung der Zusammenhang der Punkte besser verdeutlicht. Welche Methode daher wieder die beste ist, hängt von der Anwendung und dem Geschmack des Benutzers ab. Am besten probieren Sie einfach beide Varianten aus und entscheiden sich dann für die, die Ihnen besser gefällt.

### 3.4.3 Komfort-Grafik: Randbeschriftung und Mehrfachdarstellungen

Mit den gerade beschriebenen Grafiken haben wir natürlich erst das grundsätzliche Rüstzeug für die Erstellung von anspruchsvollen Diagrammen kennengelernt. Daher nun noch ein paar Worte zu möglichen Erweiterungen.

Zum ersten sind wir natürlich, wie schon gesagt nicht auf die Nullinie als Bezugspunkt angewiesen. Wir können einen beliebigen Bezugspunkt wählen, zum Beispiel das Minimum der Funktion. Um eine Min-Max-Darstellung zu erhalten brauchen wir auch nur ein paar kleine Änderungen in unserem Grundkonzept vorzunehmen. Wir subtrahieren nämlich einfach von allen Werten den minimal möglichen, der ja in der Variablen mi nach dem ersten Durchlauf zur Verfügung steht.

Dies können wir entweder in einer separaten Schleife nach der Min-Max-Berechnung durchführen oder wir ändern die Berechnung der einzelnen Werte einfach ab. Bei der Säulengrafik mit DRAW wäre beispielsweise in Zeile

```
250 h=((m(i)-mi)/(ma-mi))*250
```

zu definieren und schon erscheint die neue Grafik auf dem Schirm.

Eine weitere interessante Erweiterung besteht darin, Achsen und Achsenbezeichnungen mit in das Diagramm zu integrieren. Die Begrenzungslinien können wir ebenfalls mit dem Kommando DRAW zeichnen. Wer noch etwas mehr will, kann auch die Achseneinteilungen mit kleinen Strichen nachbilden.

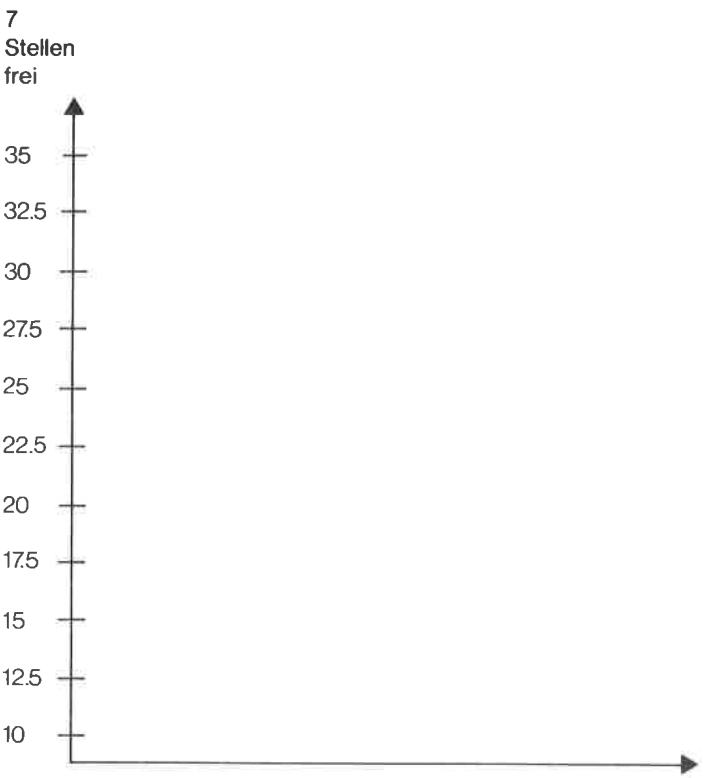



**Bild 3.4:** *Beispiel einer Achsenbezeichnung*

Wenn man so weit fortgeschritten ist, so sollte man sich natürlich auch noch überlegen, ob es nicht sinnvoll wäre, die Achsen auch zu beschriften. Man kann dazu zum Beispiel das PRINT USING-Kommando und hier speziell die Exponentialdarstellung gut verwenden. Diese Form

des PRINT-Befehls hat den Vorteil, daß wir mit einer genau vorgegebenen Länge der ausgegebenen Zeichenkette kalkulieren können.

Als Beispiel nehmen wir einmal an, wir wollten die Y-Achse beschriften und hätten dafür insgesamt 7 Ziffern Platz am linken Bildschirmrand definiert (siehe Bild 3.4). Solange die anzugebenden Zahlen kleiner als 9 Millionen sind, haben wir keine Probleme. Die 7 Stellen reichen in diesem Fall völlig.

Wenn wir dagegen, um das Programm in seiner Anwendung nicht unnötig einzuengen, keine Festlegung in Bezug auf die darzustellenden Werte treffen wollen, so bietet sich die folgende Lösung an.

Wir nehmen einmal an, wir wollten in unserem Balkendiagramm die verschiedenen Zeilen bezeichnen. Der höchste vorkommende Wert ist ma, der niedrigste mi. Die Darstellung erfolgt in 16 Zeilen Höhe, so daß wir die Differenz (ma-mi) durch 16 teilen müssen, um den Abstand zwischen den einzelnen Zeilen zu erhalten. Mit einer einfachen Schleife und PRINT USING können wir dann die ganze Benennung auf einen Schlag erzeugen. Die dazu notwendige Zeile lautet

```
275   diff=(ma-mi)/16:FOR i=4 TO 20:LOCATE 1,i:
      PRINT USING"###.#^^^^";mi+(20.5-i)*diff:NEXT i
```

Diese Gleichung gilt natürlich nur, wenn Sie schon die gerade beschriebene Änderung zur Min-Max-Darstellung in Zeile 250 eingebaut haben. Ansonsten müssen Sie hier für mi 0 einsetzen. In jedem Fall aber erzeugt die neue Zeile 275 durch 16 nacheinander folgende LOCATEs die richtigen Benennungen.

Der Wert 20.5 wurde übrigens deshalb gewählt, weil die Reihe von Bildpunkten in der untersten Zeile unserer Benennung ja dem Minimum entspricht. Der Zeilenmittelpunkt liegt dagegen wertmäßig schon *0.5*diff* höher.

Es gibt aber noch eine ganze Reihe anderer nützlicher Änderungen, die wir einbauen können, zum Beispiel die gleichzeitige Darstellung von Durchschnittswerten, möglichst noch in einer anderen Farbe, oder ein automatisches Glätten der eingegebenen Werte. In beiden Fällen müssen Werte zusammengefaßt und arithmetische oder geometrische Mittelwerte berechnet werden.

Bei der ersten Variante ersetzen die Mittelwerte dann die ursprünglichen Angaben. Es wird also eine Funktion beziehungsweise ein Diagramm dargestellt, welches weniger große Abweichungen unter den verschiede-

nen Werten aufweist. Im anderen Fall wird parallel zur Hauptdarstellung noch eine Durchschnittslinie geplottet.

Im Bereich der Erweiterungen sind Ihrer Fantasie und Ihrem Gestaltungswillen also kaum Grenzen gesetzt. Komfort und Aussagequalität sind immer noch verbesserungsfähig. Experimentieren Sie doch einmal mit den verschiedenen Methoden und Ergänzungen und erweitern Sie doch dann einmal Ihre eigenen Programme mit einer Grafikausgabe.

### 3.4.4 Multigraph - eine universelle Darstellungsroutine

Wir haben uns jetzt schon relativ ausgiebig mit den einzelnen Problemen bei der Diagrammdarstellung beschäftigt. Was jetzt allerdings noch fehlt ist ein allgemein, in einem weiten Bereich anwendbares Programm, das uns die Darstellung verschiedener Diagrammtypen auf Knopfdruck ermöglicht, ohne daß wir alle Überlegungen, die wir auf den letzten Seiten angestellt haben, nun für jedes einzelne Diagramm wiederholen müssen. Oder etwas knapper: Wir brauchen ein universelles Darstellungs-Werkzeug, ein Diagramm-Tool. Dieses soll nun entwickelt werden.

Bevor wir zur eigentlichen Programmentwicklung kommen, wollen wir uns zunächst einmal anschauen, was wir bereits an Routinen oder Unterprogrammen zur Verfügung haben. Dazu betrachten wir unseren bisherigen Wissensstand. Wir wissen bereits, wie die einzelnen Diagrammtypen erzeugt werden. Darüber hinaus sind wir in der Lage, da wir uns auch schon einige Gedanken über Erweiterungen gemacht haben, auch die Beschriftung eines Diagramms durchzuführen.

Wir verfügen damit bereits über die einzelnen Bausteine eines größeren Hauptprogramms. Was noch fehlt ist eine sinnvolle Verknüpfung der einzelnen Funktionen und natürlich eine benutzerfreundliche Auswahl.

Daneben ist ein weiterer Punkt zu beachten. Wir haben nämlich bis jetzt immer vorausgesetzt, daß die Werte, die wir in unserem Diagrammprogramm zeichnen wollten bereits in einem Array vorhanden waren. Bei der nun folgenden Programmentwicklung wollen wir dies nicht mehr voraussetzen. Hier muß also sowohl eine Routine zum Wegspeichern wie auch zum Laden, Ändern und Anfügen von Daten, also zur Datenpflege, eingebaut werden.

Wenn wir uns nun die grundsätzlichen Überlegungen, wie wir sie bei der Entwicklung unseres Programms DESIGNER in Kapitel 2.4 bereits

gemacht haben, wieder vor Augen führen, so können wir die folgende Zieldefinition festschreiben:

**Zieldefinition:** Multigraph soll es uns ermöglichen, aus einem gegebenen Werte-Array die Diagrammtypen Balken-, Strich- und Funktionengraph darzustellen, wobei es möglich sein sollte, diese sowohl in einfacher wie auch in geglätteter Form auf den Bildschirm zu bringen, um Ausreißer aus einer gegebenen Werteserie eliminieren zu können.

Ein weiterer Teil des Programms muß sich mit der Dateneingabe beschäftigen. Es müssen alle für eine komfortable Datenpflege notwendigen Funktionen, wie das Anfügen und Verändern von Daten, das Löschen einer Datei und natürlich auch das Abspeichern und Laden von Dateien, enthalten sein.

Die Werteingabe erfolgt dabei über Tastatur, wobei wahlweise die Werte mit INPUT oder INKEY$ abgefragt werden sollen.

Die Auswahl der einzelnen Funktionen dagegen soll mit der Menütechnik geschehen. Wir wollen dabei in einem 3-Window-System arbeiten. Die obersten drei Bildschirmzeilen nehmen dabei den Programmtitel beziehungsweise den Namen einer Unterfunktion auf und geben an, wo man sich momentan befindet.

In der Mitte unseres Bildschirms, dem Window#2 sollen die Hauptauswahlmöglichkeiten und größere Meldungen des Computers ausgegeben werden. Im Window#0, das auf die unteren vier Bildschirmzeilen definiert ist, erfolgt schließlich die hauptsächliche Kommunikation mit der Maschine, also die Abfrage und Eingabe von Werten sowie die Ausgabe kurzer Computerkommentare.

Ein Bereich ist bis jetzt noch nicht besprochen worden. Es ist dies die Datenschnittschnelle. Denn selbstverständlich muß unser Programm in der Lage sein, als einfache Darstellungsroutine für andere Programme beispielsweise eine größere Dateiverwaltung oder ein mathematisches Programm zu fungieren. Dazu muß es natürlich von diesem Programm Daten übernehmen können.

Dieser Problemkreis wurde hier denkbar einfach gelöst: Die zu übergebende Datei und auch selbstverständlich die Dateien, die das Programm wegspeichert, bestehen aus drei Teilen. Den Anfang bildet ein String, der den Programm- beziehungsweise Dateinamen enthält, es folgt die Anzahl der übergebenen Parameter und dann die Werte selbst.

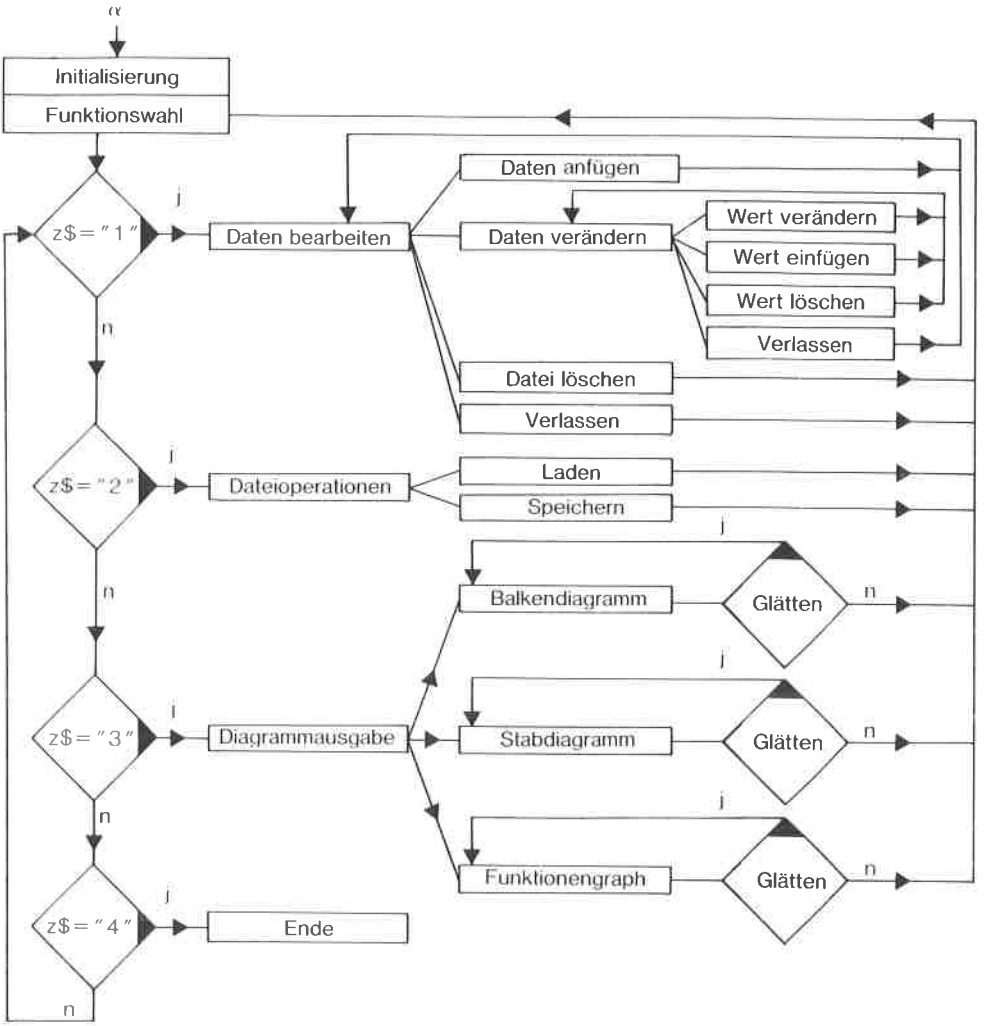



**Bild 3.5:** *Flußdiagramm Programm Multigraph*

Mit dieser Zieldefinition haben wir nun alles notwendige festgelegt, um den nächsten Schritt durchführen zu können, die Entwicklung einer Programmstruktur beziehungsweise eines Flußdiagramms. Dieses finden Sie in Bild 3.5 abgedruckt. Es wird die Basis unserer nun folgenden Programmbeschreibung bilden.

**Programmbeschreibung:**

Am Anfang von Multigraph steht wieder eine Initialisierungsroutine. Sie legt die Bildschirmaufteilung, die Bildschirmfarbe und den Zeichensatz fest. Wir arbeiten hier mit einem nur leicht veränderten deutschen Zeichensatz. Wir können die deutschen Umlaute zwar darstellen, allerdings wurde auf eine Tastaturumdefinition verzichtet. Die Benutzereingaben enthalten normalerweise nur Zahlenwerte, weswegen die Eingabe von Buchstaben und damit auch die Eingabe von deutschen Umlauten entfällt.

Die einzige Stelle, an der das Programm nämlich einmal eine Buchstabenkombination verlangt, ist bei der Angabe des Dateinamens, und eine Tastaturumdefinition nur zu diesem Zweck wäre wohl ein überflüssiger Luxus. Wenn Sie natürlich nicht auf eine DIN-Tastatur verzichten wollen, können Sie ja problemlos eines der Programme aus Kapitel 2 zur Tastaturumdefinition benutzen.

Nach dem Initialisierungsteil gelangen wir ins Hauptmenü. Dieses führt uns zu vier verschiedenen Funktionsteilen. Die dabei beschrittenen Datenwege finden Sie im Flußdiagramm in Form der Rauten. Ist die Input-Variable Z$ in Zeile 50 gleich 1 gesetzt worden, so kommen wir in das Menü "Daten bearbeiten". Dieses weist wieder vier Unterfunktionen auf, deren Ausführungsroutinen Sie ab Zeile 430 finden. Es sind dies

**Daten anfügen:** Diese Funktion fügt an die bestehende Datei einen neuen Wert an. Wenn also unsere Datei zum Eingabezeitpunkt einen Umfang von 100 Werten hat, so wird nun der 101te beschrieben.

**Daten verändern:** Hiermit sind größere Änderungen möglich. Unterfunktionen sind hierbei die gezielte Änderung eines bereits existierenden Wertes, das Einfügen eines Wertes ab einer beliebigen Position und auch das Löschen eines Datenwertes.

**Datei löschen:** Diese Unterfunktion setzt die Gesamtdatei auf 0. Dies ist immer dann notwendig, wenn man mit einer aktuellen Datei gearbeitet hat, diese auch abgespeichert wurde und nun ein neuer

Datenbestand gegebenenfalls zu einem anderen Thema eingegeben werden soll.

**Verlassen:** Diese Funktion führt uns aus dem Menü "Daten bearbeiten" wieder ins Hauptfunktionswahlmenü zurück.

Gleiches gilt für die Funktion "Datei löschen". Die anderen beiden Unterfunktionen (Daten anfügen, Daten verändern) springen dagegen wieder in die Funktion "Daten bearbeiten" zurück, um ein weiteres Anfügen beziehungsweise Verändern schneller zu ermöglichen. Der Umweg über das Funktionswahlmenü entfällt in diesem Fall.

Diese Struktur finden Sie auch im Programm wieder. Ab Zeile 320 beginnt das Menü "Daten bearbeiten". Hier wird zunächst der Bildschirm gelöscht und mit Hilfe von CHR$(10) mit übersprungenen Zeilen das Auswahlmenü ausgegeben. Die Unterfunktionen kennen wir jetzt bereits. Es folgt eine Abfrageschleife mit INKEY$, die in Zeile 370 und 380 festlegt, welches Untermenü angesprungen werden soll.

Die Untermenüs finden Sie dann ab Zeile 400 fortlaufend. Die ersten beiden Unterfunktionen stellen dabei zunächst mit Hilfe einer PRINT USING-Schleife (Zeile 470, 590), die letzten zwölf Werte vor dem letzten Wert in der Datei (Funktion "Daten anfügen") beziehungsweise einem beliebigen anderen Wert (bei "Daten verändern") dar, bevor dann die neue Eingabe abgefragt wird.

Der eingegebene Wert kann dabei maximal neun Vorkomma- und zwei Nachkommastellen aufweisen. Wenn Sie höhere Zahlen eingeben, weist der CPC durch die Fehlerabfrage in Zeile 730 beziehungsweise 490 auf die Fehleingabe hin. Ansonsten ist zu diesen Routinen wenig zu sagen. Es finden im wesentlichen Verschiebungen der einzelnen Werte in dem Wertearray W(I) statt.

Auf eine Besonderheit ist allerdings bei der Unterfunktion "Datei löschen" hinzuweisen. Diese stellt eine hübsche Anwendung unserer Farbdefinitionen aus Kapitel 2 dar. Das Löschen der Datei ist eine relativ schwerwiegende Änderung und besonders wenn man eine Reihe neuer Daten eingegeben hat und vor dem Ziehen einer Sicherheitskopie versehentlich in diese Funktion einsteigt, hätte dies doch sehr unangenehme Konsequenzen. Daher ist hier eine Sicherheitsabfrage eingebaut.

Die eigentliche Abfrage stellt uns dabei vor keine Probleme. Es handelt sich hier (Zeile 880) um eine einfache Abfrage mit INKEY$. Interessanter ist schon das Beiwerk, die Farbänderung, die dieser Unterfunktion erst ihre Würze gibt. In Zeile 860 wird nämlich die Rahmenfarbe auf blink-

endes Rot gegen Gelb definiert (Border 6,24). Der CPC unterstreicht damit quasi noch einmal, wie schwerwiegend diese Funktion in ihrem Ergebnis ist. Das Border 1 in Zeile 880 sorgt wieder für normale Verhältnisse.

Schauen wir uns nun die anderen Operationen an. Zur Funktion 2, Dateioperation, ist relativ wenig zu sagen. Das Dateiformat haben wir schon besprochen. Es handelt sich um das einfache Schreiben beziehungsweise Lesen einer ASCII-Datei.

Der einzige Unterpunkt, auf den hier besonders hinzuweisen ist, ist die volldeutsche Benutzerführung. Mittels `OPENIN"!"` beziehungsweise bei Diskettenbetrieb `OPENIN"!"+N$` ist es nämlich möglich, die englischen Anweisungen zu unterdrücken. Mit ein paar PRINT-Anweisungen und zwischengeschalteter INKEY$-Abfrage können wir dann die englische Funktion problemlos durch ihr deutsches Äquivalent ersetzen. Sie finden diese Vorgänge in Zeile 1040-1180.

Wir wollen nun die Unterfunktion 4 (ENDE) vorziehen. Diese setzt nur das Hauptwindow (WINDOW#0) wieder auf die normale Größe, das heißt den Gesamtbildschirm und beendet das Programm.

Den interessantesten Teil unserer Routine stellt freilich der Unterpunkt 3, die Diagrammausgabe, dar. In Bild 3.5 können Sie die einzelnen Funktionen und die zu ihrer Ausführung nötigen Wege klar erkennen.

Aus dem Obermenü, das im Programm ab Zeile 1200 zu finden ist, gelangt man in die drei Untermenüs für die eigentliche Diagrammdarstellung. Alle drei Routinen rufen dabei zunächst das Unterprogramm "Diagrammvorbereitung" ab Zeile 1790 auf. Dieses stellt durch Abfrage fest, welche Werte überhaupt dargestellt werden sollen und fragt die Bezugslinie (Minimum der Werte, Maximum, Nullinie, Durchschnitt etc.) ab.

Besondere Aufmerksamkeit gebührt jedoch dem Teil zur Benennung der Y-Achse ab Zeile 2100. Eine Anforderung bei der Entwicklung dieses Programms war es ja, das Programm universell verwendbar zu gestalten. Insbesondere was den Wertebereich der einzugebenden Daten anbelangt. Dies stellt uns nun vor einige Probleme, speziell wenn wir voraussetzen, daß wir für unsere gesamte Y-Benennung nur maximal fünf Zeichen reservieren wollen.

Dies schafft natürlich schon die ersten Komplikationen, wenn ein Wert größer als 100000 oder etwa ein kleinerer Wert als 0.1 oder ähnliches, in

unserer Werteschar auftaucht. Daher ist hier ein Programmteil zur Wertekorrektur eingefügt.

Seine Aufgabe besteht darin, eine Exponentialdarstellung künstlich nachzubilden. Gefordert wird dabei, daß die endgültig darzustellenden Zahlen möglichst genau fünf Stellen zu ihrer Angabe benötigen sollen. Mit dieser Wahl hat man, auch wenn die angegebenen Werte nur in einem sehr engen Bereich liegen, dennoch eine gute Trennmöglichkeit.

Wie erreicht man nun, daß beispielsweise ein Wert von 100000 auf exakt fünf Stellen reduziert wird? Dies ist relativ einfach. Man teilt durch 10. Analog müßte man 0.1 mit 106 multiplizieren, um hier einen möglichst optimalen Wert zu erhalten.

Ganz so streng sind nun unsere Anforderungen nicht. Aber wir fordern schon, daß die endgültig abzubildenden Zahlen mindestens dreistellig vor dem Komma und natürlich weniger als fünfstellig, sind. Dazu dienen einige Korrekturschleifen. Zu Ihrem Verständnis müssen wir zurückschwenken auf die Zeilen von 890 bis 1910.

Wir haben in unserer Zieldefinition festgelegt, daß das Programm in der Lage sein muß, aus einer gegebenen Werteschar ab einem bestimmten Wert, eine Anzahl von Eingaben darzustellen. In der Praxis lösen wir dies nun durch zwei Abfragen, nämlich den ersten darzustellenden Wert und die Anzahl der darzustellenden Werte.

Mit diesen beiden Angaben wird nun ein zweites Array DW(I) geladen. Das Kürzel DW steht dabei für die darzustellenden Werte. DW ist normalerweise kleiner als das Array W, kann aber maximal dessen Umfang annehmen.

In der Schleife in Zeile 1910, die das Umschaufeln aus W in DW besorgt, finden Sie daneben auch noch das Setzen der Werte MA und MI, unsere Maximum/Minimumberechnung. Das Maximum aller darzustellenden Werte ist am Schluß, das heißt beim Aussprung aus diesem Programmteil, in MA, das Minimum in MI gespeichert.

In der nun folgenden Abfrage der Bezugslinie wird eine weitere Variable definiert, nämlich BL, dieses Kürzel steht für Basislinie. BL definiert uns damit den Punkt, bei dem die X-Achse die Y-Achse schneiden soll und legt damit den untersten zu beschriftenden Wert fest.

Dies gilt jedoch nur solange BL kleiner als MI und MA ist. Legen wir dagegen unsere Basislinie in die Mitte, so muß das Minimum der Werte für die Berechnung des untersten anzugebenden Punktes benutzt werden. Aus diesem Grund werden in Zeile 20 und 80 zwei neue Variablen

gebildet. WI nimmt den kleineren der beiden Werte MI und BL auf, WA den größeren von MA und BL.

Die Variablen WI und WA charakterisieren damit die äußersten Grenzlinien unserer Diagrammdarstellung, außerhalb derer nichts mehr geht. Hier liegt weder eine Bezugslinie noch haben unsere eingegebenen Werte in diesem Bereich etwas zu suchen.

In Zeile 2130 untersuchen wir nun, ob WA und WI größer als 10000 sind. In diesem Fall muß eine Reduktion der Werte, das heißt eine Division aller Daten des Arrays DW durch 10 erfolgen, um sicherzustellen, daß die Beschriftung mit fünf Zeichen gewahrt bleibt. Falls WA oder WI außerhalb des erlaubten Bereichs liegen, wird die Variable W um eins erhöht. Beim Ansprung unseres Programms ist sie 0 (vergleiche Zeile 2080) und wird nun so lange hochgezählt bis beide Werte (WA und WI) erlaubte Größen enthalten. W stellt damit eine Art Exponenten dar.

Nun ist aber auch der andere Fall möglich, also daß WA oder WI zu klein sind. Dieser Fall wird nun in Zeile 2150 korrigiert. Hier geht es in der umgekehrten Richtung. WA und WI, und damit schließlich auch die Werte des Arrays BW, werden so lange mit 10 multipliziert, das heißt die Variable W um eins vermindert, bis beide Größen den Wert von 100 übersteigen.

Nach beiden Korrekturen wird dann ein String B$ definiert, der unseren Exponenten enthält. Schauen Sie sich einmal an, wie er zusammengesetzt ist. Nach diesen Vorbereitungsaktionen kann dann mit einer Schleife in Zeile 2180 bis 2200 die Beschriftung der Y-Achse ausgegeben werden. Reserviert sind dafür vier Stellen plus eine Ziffernstelle für ein negatives Vorzeichen.

Soweit zur Diagrammvorbereitungsroutine ab Zeile 1790, die von allen drei Programmen, wie schon gesagt, am Anfang aufgerufen wird. Für die Kommunikation der Programme ist hierbei nur noch zu sagen, daß eine fehlerhafte Eingabe in diesem Programmteil die Variable ER auf eins setzt. Nach dem Rücksprung aus 1790 wird immer geprüft, ob ER = 1 gesetzt wurde. In diesem Fall erfolgt dann ein sofortiger Rücksprung ins Hauptmenü.

Ansonsten, das heißt ohne Fehlermeldung, werden dann die drei Diagrammtypen, beziehungsweise genauer gesagt einer davon, dargestellt. Die dazu verwendeten Programmier-Algorithmen kennen wir ja bereits aus der Betrachtung unserer Einzeldiagramme im letzten Unterkapitel.

Interessant ist hier nur noch ein weiterer Punkt, nämlich das schon beschriebene Glätten von Werten, zur Eliminierung von Ausreißern oder zur Trendbestimmung. Dies ist mit allen drei Funktionstypen möglich. Sie finden es immer nach der PRINT-Ausgabe in den Zeilen 1400, 1560 und 1690. Nach dieser Abfrage, die beim Drücken der 1 in das Hauptmenü zurückspringt, und beim Drücken von 2 die nachfolgenden Zeilen und eine nochmalige Diagrammdarstellung durchläuft, finden wir dann die eigentlichen Glättroutinen.

Diese sind denkbar einfach aufgebaut. Es wird nämlich jeder Wert in unserem Array DW gleich dem arithmetischen Mittel von sich selbst und seiner beiden Nachbarn, das heißt des vorhergehenden und des nachfolgenden Wertes gesetzt.

Dies ist natürlich nur für die Werte von zwei bis WN-1, wobei WN die Maximalzahl des Arrays DW darstellt, möglich, denn der letzte und der erste Wert haben ja keinen linken beziehungsweise rechten Nachbarn. Daher müssen DW von eins und DW von WN noch zum Beispiel in Zeile 1450 auf andere Werte gesetzt werden. Hier wird nur ein teilweises arithmetisches Mittel aus dem einen möglichen Nachbarn gebildet.

## Listing 13: *Programm MULTIGRAPH*

```
10 '*****************
20 '** Multigraph **
30 '*****************
40 '************************
50 '** Initialisierung **
60 '************************
70 SPEED WRITE 1:MODE 1
80 SYMBOL AFTER 90
90 SYMBOL 91,219,60,102,102,126,102,102,0:SYMBOL 92,187,108,198,1
98,198,108,56,0:SYMBOL 93,102,0,102,102,102,102,60,0
100 SYMBOL 123,108,0,120,12,124,204,118,0:SYMBOL 124,102,0,60,102
,102,102,60,0:SYMBOL 125,0,102,0,102,102,102,62,0:SYMBOL 126,120,
20
4,204,248,204,204,248,128
110 DIM w(2500):n=0:m=2500:mi=1E+38:ma=-1E+38:DIM dw(20)
120 INK 0,0:INK 1,6:INK 2,24:INK 3,11
130 WINDOW#1,1,40,1,3:WINDOW#0,1,40,21,25:WINDOW#2,1,40,4,20
140 PAPER#1,1:PAPER 0:PAPER#2,0:PEN#1,2:PEN#2,3:PEN 3
150 BORDER 0
160 '*****************
170 '** Hauptmenue **
180 '*****************
190 CLS:CLS#1:CLS#2
200 PRINT#1,CHR$(10)+"              M U L T I G R A P H"
210 PRINT#2,CHR$(10)+CHR$(10)+"        Daten bearbeiten       (1)"
220 PRINT#2,CHR$(10)+"          Speicheroperationen    (2)"
230 PRINT#2,CHR$(10)+"          Diagrammausgabe        (3)"
240 PRINT#2,CHR$(10)+"          Ende                   (4)"
250 z$=INKEY$:IF z$="" THEN 250 ELSE IF ASC(z$)<49 OR ASC(z$)>52
THEN 250
260 ON VAL(z$) GOSUB 320,920,1220,280
270 GOTO 130
280 WINDOW#0, 1,40,1,25:CLS:END
290 '************************
300 '** Daten bearbeiten **
310 '************************
320 CLS:CLS#1:CLS#2:PRINT#1,CHR$(10)+"              Daten bearbeite
n"
330 PRINT#2,CHR$(10)+CHR$(10)+"           Daten anfügen     (1)"
340 PRINT#2:PRINT#2,"           Daten verändern   (2)"
350 PRINT#2:PRINT#2,"           Datei löschen     (3)"
360 PRINT#2:PRINT#2,"           Verlassen         (4)"
370 z$=INKEY$:IF z$="" THEN 370 ELSE IF ASC(z$)<49 OR ASC(z$)>52
THEN 370
380 IF z$="4" THEN RETURN ELSE ON VAL(z$) GOSUB 430,540,860
390 GOTO 320
400 '**********************
410 '** Daten anfuegen **
420 '**********************
430 IF n=m THEN PRINT"Speicher voll! Abspeichern oder löschen.":F
OR i=1 TO 2000:NEXT:RETURN
440 CLS#1:PRINT#1:PRINT#1,"              Daten anfügen"
450 CLS#2:PRINT#2:PRINT#2,"          Nummer    Wertangabe"
460 FOR i=MAX(n-12,1) TO n
470 LOCATE#2,1,i-MAX(n-12,1)+4:PRINT#2,USING"            #### ";i;
:PRINT#2,USING"###########.##    ";w(i)
480 NEXT i:CLS
490 INPUT"neuer Wert";nw:IF ABS(nw)>999999999 THEN PRINT"erlaubte
r Wertebereich überschritten!!":GOTO 490
500 w(n+1)=nw:n=n+1:RETURN
510 '**********************
520 '** Daten verändern **
530 '**********************
540 CLS#2:CLS:CLS#1:PRINT#1:PRINT#1,"              Daten verändern"
550 INPUT"Ab welcher  Nummer  wollen Sie die Datensehen";nr
560 IF nr>m THEN PRINT"Diese Zahl liegt außerhalb des erlaubtenBe
reichs(1-2500)!!":GOTO 550
570 CLS#2:PRINT#2:PRINT#2,"          Nummer    Wertangabe"
580 FOR i=nr TO MIN(nr+12,m)
590 LOCATE#2,1,i-nr+4:PRINT#2,USING"            #### ";i;:PRINT#2,
USING"###########.##    ";w(i)
600 NEXT i:CLS
610 INPUT"Verlassen mit '^'. Welchen Wert ändern";z$
620 IF z$="^" THEN RETURN ELSE IF VAL(z$)<=0 OR VAL(z$)>2500 THEN
```

```
 PRINT"Erlaubter Bereich(1-2500) überschritten!":FOR i=1 TO 2000:
NE
XT:GOTO 610
630 nr=VAL(z$)
640 CLS:PRINT"Was soll mit Nummer";nr;"geschehen:"
650 PRINT"Verändern (1)  Einfügen (2)  Löschen (3)Verlassen (4)"
660 z$=INKEY$:IF z$="" THEN 660 ELSE IF ASC(z$)<49 OR ASC(z$)>52
THEN 660
670 ON VAL(z$) GOSUB 710,770,820:RETURN
680 '******************
690 '** Veraendern **
700 '******************
710 CLS
720 PRINT"Bitte geben Sie den neuen Wert ein"
730 INPUT w(nr):IF ABS(w(nr))>999999999 THEN PRINT"Wert außerhalb
 des erlaubten Bereichs!!":GOTO 710 ELSE n=MAX(n,nr):RETURN
740 '******************
750 '** Einfuegen **
760 '******************
770 IF n=m THEN PRINT"Speicher voll! Abspeichern oder löschen.":F
OR i=1 TO 1000:NEXT:RETURN ELSE IF nr>n THEN PRINT"Hier gibt es n
oc
h keine Daten!!":FOR i=1 TO 1000:NEXT:RETURN
780 n=n+1:FOR i= n TO nr STEP-1:w(i+1)=w(i):NEXT:GOSUB 710:RETURN
790 '**********************
800 '** Wert loeschen **
810 '**********************
820 FOR i= nr TO n-1:w(i)=w(i+1):NEXT i:w(n)=0:n=n-1:RETURN
830 '**********************
840 '** Datei loeschen **
850 '**********************
860 BORDER 6,24:CLS#2:CLS:PRINT#2,"Sie haben den Befehl zum Lösch
en der Da-tei eingegeben. Damit werden alle momen-tan  im  Speich
er
  vorhandenen Daten ge-löscht.  Falls keine Kopie auf Band exi-st
iert, sind sie damit verloren!"
870 PRINT:PRINT"Soll der Befehl trotzdem ausgeführt wer-den j/n?"
880 z$=LOWER$(INKEY$):IF z$="n" THEN BORDER 1:RETURN ELSE IF z$<>
"j" THEN 880 ELSE CLEAR:GOTO 110
890 '************************
900 '** Dateioperationen **
910 '************************
920 CLS:CLS#1:CLS#2
930 PRINT#1:PRINT#1,"              Dateioperationen"
940 PRINT:INPUT"Bitte geben Sie den Namen der Datei ein";n$
950 CLS#2:PRINT#2:PRINT#2,"         aktuelle Datei sichern (1)"
960 PRINT#2:PRINT#2,"         Datei einladen          (2)"
970 PRINT#2:PRINT#2,"         Funktion abbrechen      (3)"
980 z$=INKEY$:IF z$="" THEN 980 ELSE IF ASC(z$)<49 OR ASC(z$)>52
THEN 980
990 IF z$="3" THEN RETURN
1000 IF z$="2" THEN 1120
1010 '******************
1020 '** speichern **
1030 '******************
1040 PRINT"PLAY und REC drücken. Dann Tastendruck!"
1050 z$=INKEY$:IF z$="" THEN 1050 ELSE OPENOUT"!"+n$
1060 PRINT#9,n$:PRINT#9,n
1070 FOR i=1 TO n:PRINT#9,w(i):NEXT
1080 CLOSEOUT:RETURN
1090 '************
1100 '** laden **
1110 '************
1120 PRINT"PLAY drücken. Dann Tastendruck!"
1130 z$=INKEY$:IF z$="" THEN 1130
1140 OPENIN"!":INPUT#9,m$:IF n$=m$ THEN 1160
1150 CLOSEIN:IF INKEY$="^" THEN RETURN ELSE PRINT m$:GOTO 1140
1160 INPUT#9,n:PRINT m$
1170 FOR i=1 TO n:INPUT#9,w(i):NEXT i
1180 CLOSEIN:RETURN
1190 '****************
1200 '** Diagramm **
1210 '****************
1220 CLS:CLS#2:PRINT#2,CHR$(10)+" Bitte wählen Sie die Darstellun
gsart:"
1230 PRINT#2,CHR$(10)+CHR$(10)+"            Balkendiagramm    (1)"
1240 PRINT#2:PRINT#2,"            Stabdiagramm      (2)"
1250 PRINT#2:PRINT#2,"            Funktionengraf    (3)"
```

```
1260 z$=INKEY$:IF LEFT$(z$,1)="+" OR LEFT$(z$,1)="-" OR LEFT$(z$,
1)="."OR LEFT$(z$,1)="&" THEN 1260 ELSE IF VAL(z$)<1 OR VAL(z$)>3
 T
HEN 1260
1270 ON VAL(z$) GOSUB 1320,1500,1630
1280 RETURN
1290 '**********************
1300 '** Balkendiagramm **
1310 '**********************
1320 GOSUB 1790:IF er=1 THEN RETURN
1330 bl=((bl-wi)/(wa-wi))*306+16
1340 f=wa-wi:br=540/wn:MOVE 91,17:DRAWR 0,315,3:MOVE 91,bl:DRAWR
550,0,3
1350 FOR i=16 TO 331 STEP 16:MOVE 85,i:DRAWR 12,0:NEXT i
1360 FOR j=1 TO wn:c=j MOD 3+1
1370 FOR i=(j-1)*br+100 TO j*br+99
1380 MOVE i,bl:DRAW i,((dw(j)-wi)/f)*306+16,c
1390 NEXT i,j
1400 PRINT#2,"          Menü (1)    Glätten (2)"
1410 z$=INKEY$:IF z$<>"1" AND z$<>"2" THEN 1410 ELSE IF z$="1" TH
EN RETURN
1420 FOR i= 2 TO wn-1
1430 dw(i)=(dw(i-1)+dw(i)+dw(i+1))/3
1440 NEXT i
1450 dw(1)=(dw(1)+dw(2))/2:dw(wn)=(dw(wn)+dw(wn-1))/2
1460 WINDOW#3,6,40,4,24:PAPER#3,0:CLS#3:GOTO 1340
1470 '**********************
1480 '** Stabdiagramm **
1490 '**********************
1500 GOSUB 1790:IF er=1 THEN RETURN
1510 bl=((bl-wi)/(wa-wi))*306+16
1520 f=wa-wi:br=540/wn:MOVE 91,17:DRAWR 0,315,3:MOVE 91,bl:DRAWR
550,0,3
1530 FOR i=16 TO 331 STEP 16:MOVE 85,i:DRAWR 12,0:NEXT i
1540 FOR j=1 TO wn:c=j MOD 3+1
1550 FOR i=(j-0.5)*br+98 TO (j-0.5)*br+100:MOVE i,bl:DRAW i,((dw(
j)-wi)/f)*306+16,c:NEXT i,j
1560 PRINT#2,"          Menü (1)    Glätten (2)"
1570 z$=INKEY$:IF z$<>"1" AND z$<>"2" THEN 1570 ELSE IF z$="1" TH
EN RETURN
1580 FOR i= 2 TO wn-1:dw(i)=(dw(i-1)+dw(i)+dw(i+1))/3:NEXT i
1590 dw(1)=(dw(1)+dw(2))/2:dw(wn)=(dw(wn)+dw(wn-1))/2:WINDOW#3,6,
40,4,24:PAPER#3,0:CLS#3:GOTO 1520
1600 '*************
1610 '** F-Graf **
1620 '*************
1630 GOSUB 1790:IF er=1 THEN RETURN
1640 bl=((bl-wi)/(wa-wi))*306+16
1650 f=wa-wi:br=540/wn:MOVE 91,17:DRAWR 0,315,3:MOVE 91,bl:DRAWR
550,0,3
1660 FOR i=16 TO 331 STEP 16:MOVE 85,i:DRAWR 12,0:NEXT i
1670 FOR j=2 TO wn
1680 MOVE (j-1)*br+100,((dw(j-1)-wi)/f)*306+16:DRAW j*br+100,((dw
(j)-wi)/f)*306+16,1:NEXT j
1690 PRINT#2,"          Menü (1)    Glätten (2)"
1700 z$=INKEY$:IF z$<>"1" AND z$<>"2" THEN 1700 ELSE IF z$="1" TH
EN RETURN
1710 FOR i= 2 TO wn-1
1720 dw(i)=(dw(i-1)+dw(i)+dw(i+1))/3
1730 NEXT i
1740 dw(1)=(dw(1)+dw(2))/2:dw(wn)=(dw(wn)+dw(wn-1))/2
1750 WINDOW#3,6,40,4,24:PAPER#3,0:CLS#3:GOTO 1650
1760 '****************************
1770 '** Diagrammvorbereitung **
1780 '****************************
1790 CLS:CLS#1:CLS#2:PRINT#1:PRINT#1,"          M U L T I G R A P
 H"
1800 er=0:IF n<10 THEN PRINT"Nicht genügend Werte vorhanden (min.
10)":FOR i=1 TO 2000:NEXT:er=1:RETURN
1810 '****************************
1820 '** Abfrage Wertebereich **
1830 '****************************
1840 PRINT"Ab welchem Wert soll dargestellt werden";:INPUT nr
1850 IF nr>2490 OR nr<1 THEN PRINT"zulässiger Bereich (1-2490)  v
erlassen!!":FOR i=1 TO 1500:NEXT:GOTO 1790
1860 CLS:PRINT"Wieviel Werte sollen dargestellt werden:10-60 für
Strichdiagramm und F-Graf auch100 und 250.";
1870 INPUT wn
```

```
1880 IF wn<10 OR (wn>60 AND ((wn<>100 AND wn<>250) OR dm<2)) THEN
 PRINT"Fehleingabe!!":FOR i=1 TO 1000:NEXT:GOTO 1860
1890 IF nr+wn>2500 THEN PRINT"Es existieren nur maximal 2500 Wert
e !' Sie haben als Anfang";nr;"und als Anzahl";wn;"angegeben. Dam
it
 wird diese Grenze","überschritten.":FOR i=1 TO 2000:NEXT:GOTO 17
90
1900 ma=-1E+37:mi=1E+37:ERASE dw:DIM dw(wn+1)
1910 FOR i=nr TO wn+nr-1:dw(i-nr+1)=w(i):ma=MAX(ma,w(i)):mi=MIN(m
i,w(i)):NEXT
1920 '*****************************
1930 '** Wahl der Bezugslinie **
1940 '*****************************
1950 CLS:CLS#2
1960 PRINT#2:PRINT#2," Welche Basislinie wählen Sie:"
1970 PRINT#2:PRINT#2:PRINT#2," Minimum der Werte   (1)"
1980 PRINT#2:PRINT#2," Maximum der Werte   (2)"
1990 PRINT#2:PRINT#2," Nulllinie           (3)"
2000 PRINT#2:PRINT#2," Durchschnitt        (4)"
2010 PRINT#2:PRINT#2," individuelle Angabe (5)"
2020 z$=INKEY$:IF LEFT$(z$,1)="+" OR LEFT$(z$,1)="-" OR LEFT$(z$,
1)="."OR LEFT$(z$,1)="&" THEN 2020 ELSE IF VAL(z$)<1 OR VAL(z$)>5
 T
HEN 2020
2030 IF z$="4" THEN sum=0:FOR i=1 TO wn:sum=sum+dw(i):NEXT:bl=sum
/wn:GOTO 2080
2040 IF z$="1" THEN bl=mi:GOTO 2080
2050 IF z$="2" THEN bl=ma:GOTO 2080
2060 IF z$="3" THEN bl=0:GOTO 2080
2070 CLS:PRINT" Maximum:";ma:PRINT" Minimum:";mi:PRINT:INPUT" Bas
islinie";bl:IF ABS(bl)>1E+09 THEN PRINT"Bereich (+/- 1 Mrd) übers
ch
ritten":FOR i=1 TO 2000:NEXT:GOTO 2070
2080 wi=MIN(mi,bl):wa=MAX(ma,bl):w=0:w$=""
2090 IF wa=wi THEN PRINT"Keine Unterschiede in den Werten. Daher
keine Darstellung möglich.":FOR i=1 TO 2000:NEXT:er=1:RETURN
2100 '*****************************
2110 '** Wertangaben berechnen **
2120 '*****************************
2130 IF ABS(wa)/10^w>9999 OR ABS(wi)/10^w>9999 THEN w=w+1:GOTO 21
30
2140 IF w=0 THEN 2150 ELSE w$="*10^"+STR$(w)
2150 IF ABS(wa)/10^w<100 AND ABS(wi)/10^w<100 THEN w=w-1:GOTO 215
0
2160 IF w=0 THEN 2170 ELSE w$="*10^"+STR$(w)
2170 WINDOW#0,1,40,4,24:WINDOW#2,1,40,25,25:br=wa-wi:sb=br/19:CLS
#2:CLS
2180 FOR i=0 TO 19
2190 LOCATE 1,i+2:PRINT USING"#####";(wa-i*sb)/10^w
2200 NEXT:RETURN
```

# 4 Der CPC als Maler

Von der Verwendung der Grafik als Hilfsroutine für andere Programme, sei es im Bereich der grafischen Darstellung oder bei der Aufwertung von Programmen durch schöne Titelbilder, machen wir jetzt den Sprung zur Grafik als Selbstzweck. Kreatives Computern bedeutet nämlich nicht unbedingt, daß man Programme schreibt, die Grafiken benutzen. Viel schöner ist es, sich damit zu beschäftigen, selber Bilder und Grafiken als Selbstzweck zu entwerfen.

Voraussetzung dafür und natürlich auch für ein komfortables Erstellen von Titelbildern mit hochauflösender Grafik ist allerdings ein Programm, das erhebliche Fähigkeiten besitzt und den angehenden Künstler voll in seinem Schaffensdrang unterstützt. Es muß uns nämlich ein komfortables Arbeiten mit der hochauflösenden Grafik erlauben.

Ein solches Programm wollen wir nun entwickeln. Am Anfang von Kapitel 3 haben wir uns ja bereits mit den Mitteln und einer Reihe einfacher Figuren beschäftigt, über die wir im Bereich der hochauflösenden Grafik verfügen.

Erinnert sein soll hier nur an die einfachen Befehle zum Setzen von Punkten und zum Ziehen von Linien. Aber auch die etwas komplexeren Kommandos, wie das Zeichnen von nichtausgefüllten oder ausgefüllten Rechtecken und Kreisen sollten Sie sich hierzu in Erinnerung rufen.

Wenn wir nun all diese grafischen Figuren mit einem cursororientierten Bedienmenü verbinden, das es uns ermöglicht, über den Bildschirm zu wandern und dann an den verschiedenen Positionen, natürlich auch in verschiedenen Farben, diese Grafikelemente zu setzen, so haben wir eigentlich schon alles was wir brauchen.

Große Bedeutung kommt in diesem Zusammenhang natürlich der Cursorführung und hier insbesondere der exakten Positionierung eines Grafikcursors auf jeden beliebigen Punkt auf dem Bildschirm zu. Da wir im Grafikmodus arbeiten, ist dies etwas schwieriger als bei unserem Programm DESIGNER. Der Grund:

Wir können nicht mehr den eingebauten Cursor (Textcursor) benutzen, da dessen Auflösung mit maximal 80 Zeichen in 25 Zeilen natürlich viel zu gering für unsere Anwendung ist. Wir müssen also auf irgendeine Art einen besonderen Grafikcursor auf dem CPC installieren, der es uns ermöglicht, jeden Einzelbildpunkt getrennt anzufahren.

Damit ist dann natürlich auch sofort ein weiteres Problem verbunden. Stellen Sie sich einmal vor, Sie wären mit einem solchen Grafikcursor in der Mitte des Bildschirms und wollten nun beispielsweise in die linke obere Ecke gelangen. Es wären mehrere hundert Einzelschritte notwendig, um das Ziel zu erreichen. Dies drückt natürlich auf die Geschwindigkeit unserer Grafikarbeit. Zu diesem Punkt werden wir also einige Überlegungen anstellen müssen.

Nach diesen Vorbemerkungen sind wir aber nun erst einmal in der Lage, unsere grundsätzliche Zieldefinition für das Programm zu bestimmen. Es trägt den Namen CPC PAINT.

**Zieldefinition:** CPC PAINT soll ein Grafikentwicklungsprogramm für die hochauflösende Grafik werden, welches hauptsächlich mit Joystick oder den Cursorsteuertasten bedient wird. Mit diesen soll es möglich sein, auf einem Gesamtbildschirm an eine beliebige Position zu gelangen und dort grafische Figuren, wie Punkte, Linien, Rechtecke etc., zu setzen.

Über einige Funktionstasten soll es daneben möglich sein, in ein Menü zu gelangen, das Sonderfunktionen bereithält, wie das Laden und Sichern von Bildern, den Mode- und Farbwechsel und natürlich auch die Auswahl der aktuell zu benutzenden Grafikfigur. Das eigentliche Setzen der Grafikfigur erfolgt dann mit <FIRE> beziehungsweise <COPY>.

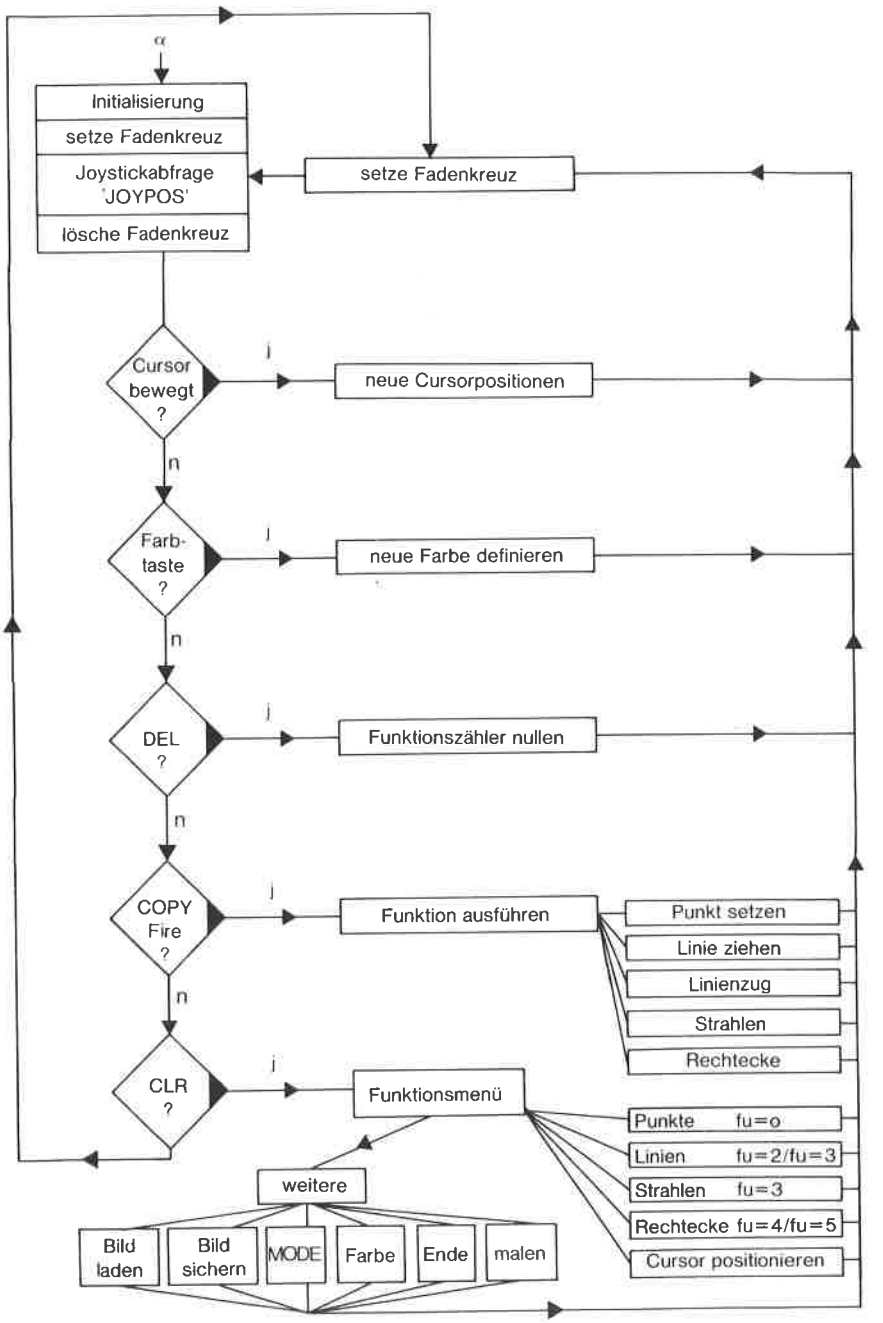



**Bild 4.1:** *Flußdiagramm Programm CPC PAINT*

## 4.1 Einige Probleme vorab

Bevor wir anhand des Flußdiagramms das Programm näher untersuchen, sollen jedoch noch ein paar Worte zu dem Problempunkt Cursorsteuerung gesagt werden.

Wie schon angeführt, besitzt der CPC selbst keinen Grafikcursor, das heißt er besitzt schon einen, aber dieser ist normalerweise unsichtbar. Man kann ihn zwar mit dem Befehl

```
PLOT XPOS,YPOS
```

jederzeit auf dem Schirm sichtbar machen. Als Ergebnis erhält man aber dann nur einen Punkt von der Größe eines Pixels. Ein solcher Cursor weist zwei Nachteile auf:

Erstens ist er natürlich zu klein und strengt damit die Augen unnötig an. Zum zweiten -und dies ist vielleicht noch wichtiger- kann es besonders bei der Benutzung eines Farbmonitors vorkommen, daß durch technische Fehler in der Bildschirmmaske ein Bildpunkt nur zu einem Viertel angesprochen wird, so daß man ihn fast nicht mehr erkennen kann. In diesem Fall ist dann überhaupt keine Orientierung mehr auf dem Bildschirm gegeben.

Diese Aussagen gelten noch in besonderem Maße, wenn man ein vielfarbiges hochauflösendes Bild gezeichnet hat und in diesem noch einige Änderungen durchführen will.

Wir müssen uns also nun überlegen, wie wir einen künstlichen Cursor schaffen können. Dieser muß zwei Eigenschaften haben. Zum ersten muß er es uns natürlich ermöglichen, exakt einen Bildpunkt auszuwählen, das heißt, der Grafikcursor sollte auch immer nur auf einen Pixel deuten. Außerdem muß er gut sichtbar sein, also eine Bildschirmfläche einnehmen, die ungefähr der Größe eines Zeichens entspricht.

Wir werden daher unseren Cursor aus zwei Teilen aufbauen. Als Grundstock nehmen wir einen ganz normalen Bildpunkt, der mit PLOT angesprochen wird, darüber lagern wir jedoch ein Fadenkreuz von der Größe eines Zeichens. Dieses definieren wir mit SYMBOL. In seiner Mitte wird dann jeweils mit PLOT der entsprechende Grafikpunkt aufgetastet.

### 4.1.1 Die Cursorbewegung

Das Grundprinzip ist relativ einfach. Es sind nur noch zwei besondere Punkte, die Kopfschmerzen bereiten. Da wäre zum einen das Setzen unseres Zeichens. Wir haben bis jetzt Zeichen nur in einer Matrix aus maximal 80 Spalten * 25 Zeilen setzen können. Wenn aber nun die Zeichenmitte exakt über dem aktuell bearbeiteten Grafikbildpunkt liegen soll, so müssen wir unseren Pseudografikcursor auch pixelweise bewegen können, und zwar in allen Richtungen.

Dies ist aber mit dem BASIC-Kommando TAG problemlos möglich. Mehr zu diesem Befehl und seiner Anwendung im nächsten Kapitel.

Ein weiterer Punkt ist jedoch etwas schwieriger zu lösen. In dem Moment, wo wir nämlich unseren Cursor auf den Bildschirm setzen, überschreibt er ja den aktuellen Bildschirminhalt. Dieser ist also in diesem Moment verloren. Hier können wir jedoch auf einen Trick zurückgreifen, den wir schon in Kapitel 2.4 bei den Steuercodes kennengelernt haben, nämlich das Kommando

```
PRINT CHR$(23)
```

Mit diesem Befehl ist es möglich, den Modus des Grafikfarbstiftes umzudefinieren. Während beim normalen Ablauf ein gesetzter Bildpunkt die Farbe des zu diesem Zeitpunkt auf dem Schirm befindlichen Punktes überschreibt, also durch die neue Farbe ersetzt, so ist es auch möglich, mit dem nachgestellten Parameter 1 den XOR-Modus für den Grafikfarbstift zu adressieren.

In Kapitel 2.4 haben wir gesehen, daß es möglich ist, mit Hilfe einer zweimaligen XOR-Verknüpfung einen zeitweiligen Farbwechsel auf dem Bildschirm herzustellen, wobei dennoch nach dem zweiten XOR der Ursprungszustand wiederhergestellt war. Wir werden diese Technik nun zur Realisierung unseres Grafikcursors benutzen.

### 4.1.2 Die Abspeicherung der Grafik

Neben dem Aufbau unseres Cursors soll nun noch vor der eigentlichen Programmbeschreibung ein weiterer Punkt zur Sprache kommen. Es ist dies die Abspeicherung der Grafiken. Wenn Sie einen kurzen Blick auf das Flußdiagramm werfen, so werden Sie feststellen, daß es eine Reihe von Unterprogrammen gibt, die nur durch einfachen Tastendruck ausgeführt werden können. Diese benötigen also keine Bildschirmein- oder -

ausgabe und zerstören damit natürlich auch das entworfene Grafikbild nicht.

Eine Reihe anderer Routinen erfordert jedoch unbedingt die Ausgabe eines Menüs oder die Eingabe von Parametern und würde damit auf jeden Fall unsere Grafik zerstören. Diese Problematik kennen wir schon von der Entwicklung unseres Programms DESIGNER in Kapitel 2.3. Allerdings haben wir es hier mit einem unterschiedlichen Problem zu tun. Bei DESIGNER hatten wir unsere Grafiken nicht als hochauflösende Grafik, sondern in kodierter Form als Blockgrafikzeichen in einem separaten Speicher für den Zeichencode und die Farbe abgelegt.

Dieses Verfahren ist natürlich hier im Bereich der hochauflösenden Grafik nicht mehr möglich. Wir müssen daher mit zwei verschiedenen Grafikbildschirmen arbeiten. Wir wollen an dieser Stelle nicht so sehr auf den technischen Ablauf der Grafikspeicherumdefinition eingehen. Diese Problematik wird in den folgenden Kapiteln noch ausführlich zur Sprache kommen. Hier soll es vielmehr um die Anwendung der dazu benötigten Wechselroutinen gehen.

CPC PAINT arbeitet mit zwei verschiedenen Grafikspeichern, die ab der hex-Adresse 4000 beziehungsweise C000 nach oben liegen. Dabei enthält der Adreßbereich ab 4000 alle Kontrollabfragen, die Funktionswahlmenüs und so weiter, während ab Adresse C000 das hochauflösende Grafikbild gespeichert ist.

Im Bildschirmbereich ab hex C000, den auch der CPC normalerweise für alle Ausgaben benutzt, werden durch diesen Trick Änderungen nur dann vorgenommen, wenn der Benutzer sie mit Cursor und den Grafikfunktionen durchführt. Der andere Bereich dagegen wird wie bei jedem normalen Programmablauf laufend gelöscht und auch vom Computer mit Ausgaben wie eben den Menüs besetzt.

Zum Umschalten der Bildschirmausgabe stehen drei Routinen zur Verfügung, die aus relativ wenigen Kommandos bestehen. Sie werden alle mit dem CALL-Befehl und nachgestellter Ansprungadresse aufgerufen.

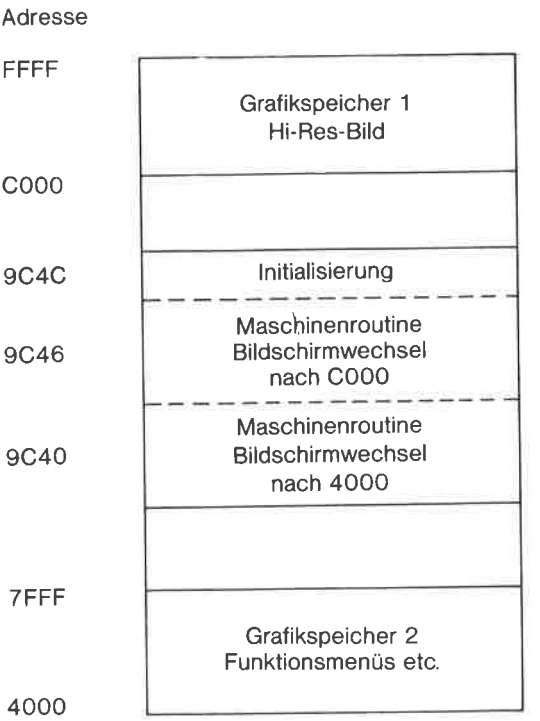



**Bild 4.2:** *Speicheraufteilung Programm CPC PAINT*

Die Ansprungpunkte sind dabei 40000 für das Umschalten auf den unteren Grafikspeicher (hex 4000), 40006 für das Umschalten in den höheren Grafikspeicher (hex C000) und 40012 für die Initialisierung des Programms. Die letzte Routine schaltet dabei etwaige Verschiebungen innerhalb der Bildschirmbereiche (näheres dazu siehe Kapitel 6) wieder zurück. Der Speicher bleibt bei Aufruf dieser Routine permanent im Normzustand. Ein etwaiger Bildschirm-Offset (siehe Kapitel 6.1) wird durch diese Routine korrigiert.

Die Speicheraufteilung sieht also wie in Bild 4.2 gezeigt aus. Die Speicherobergrenze wurde auf den Wert 3FFF herabgesetzt. Wir haben also ab Adresse 4000 den Speicher für unsere Maschinenprogramme und die Grafikspeicher reserviert. Nach dieser Vorbemerkung können wir uns nun an die Analyse des eigentlichen Programms wagen.

## 4.2 Programmbeschreibung

Wie Ihnen ein kurzer Blick auf das Flußdiagramm in Bild 4.1 zeigt, ist das Programm im wesentlichen in Schleifenform aufgebaut. Sie können die Haupt-Schleife relativ einfach erkennen, wenn Sie zunächst einmal annehmen, daß keine Taste gedrückt wurde.

In diesem Fall folgt nach der Initialisierung und dem Setzen des Fadenkreuzes ein Teil zur Joystickabfrage. Nachdem durch die Abfrageroutine die aktuelle Cursorposition korrigiert wurde, wird das Fadenkreuz sofort wieder gelöscht.

Wir nehmen nun erst einmal an, es wurde keine Taste gedrückt. Insofern sind die nachfolgenden fünf Abfragerauten zu verneinen. Aus dem CLR geht es also unten wieder heraus und in den Teil "Setze Fadenkreuz".

Dies ist die Ursprungsschleife. Sie wird permanent durchlaufen und man gelangt aus ihr nur durch irgendeinen Tastendruck heraus. Trotzdem führt aber jeder Tastendruck im Endeffekt wieder in diese Schleife zurück. Dieses Prinzip können Sie auch im Programm relativ gut wiederfinden.

Am Anfang von CPC PAINT steht wie gewohnt der Initialisierungsteil. Das Maschinenprogramm wird ab Adresse 40000 aus den Datazeilen (Zeile 140 und 150) eingeladen und danach die Initialisierungsmaschinenroutine (Adresse 40012) aufgerufen. Es geht weiter nach 1910 in die Farbinitialisierung. Diesen Teil des Programms kennen wir schon vom Programm DESIGNER her.

Als nächstes wird der Character für unser Fadenkreuz definiert. Das erste Auswahlsymbol in Zeile 180 legt dabei ein wirkliches Fadenkreuz fest. In Zeile 190 und 200 liegt ein weiteres Auswahlsymbol verborgen. Dieses stellt einen Kreis dar. Welchen Character Sie im Endeffekt verwenden, ist Geschmackssache. Sie sollten ruhig einmal beide Varianten ausprobieren und prüfen, welches Symbol Ihnen besser gefällt. Auf den weiteren Programmablauf hat diese Definition allerdings keinen existentiellen Einfluß.

### 4.2.1 Die Haupt-Abfrageschleife

Nach der Initialisierung gelangen wir dann ab Zeile 290 in die Routine JOYPOS, die die Cursorabfrage und die Verzweigung zu den Unterfunktionen enthält. Sie ist als endlose Schleife ausgeführt, von Zeile 560 geht es also immer wieder nach 290 zurück.

Aus dieser Schleife sind mehrere Ansprünge möglich, die aber alle als Unterprogrammaufrufe ausgeführt sind, das heißt, das Programm springt in eine Unterfunktion und kehrt danach wieder in die Schleife zurück.

In der Schleife selbst sind dabei mehrere Punkte erwähnenswert. Da wäre zum einen die Tastatur- beziehungsweise Joystickabfrage. Die Abfrage mit `INKEY (<Tastennummer>)` kennen Sie bereits aus Kapitel 2.3 und ebenso natürlich die Joystickabfrage aus der Entwicklung unseres Programms DESIGNER.

Hier wurden beide Techniken kombiniert. Die Steuerung unseres Grafikcursors ist also sowohl mit Joystick wie auch mit den Cursorsteuertasten möglich. Die dazu notwendigen Abfragen finden Sie in Zeile 290-330. Alle Abfragen in diesem Funktionsteil wurden übrigens mit der direkten Tastaturabfragemethode durchgeführt, so auch in Zeile 380-470 das Setzen des Farbregisters F. Die Wahl der anzuwendenden Farbe geschieht dabei über das abgesetzte Zehnertastaturfeld.

Sie sollten aber dabei berücksichtigen, daß durch die Tastaturabfrage eine Trennung der Funktionstasten in diesem Tastenfeld mit den Tasten für die Zahlen in der oberen Reihe unserer Haupttastatur erfolgt. Wenn wir also auf die Taste 1 in der oberen Reihe unserer Tastatur drücken, so passiert nichts. Geben wir dagegen die 1 über unsere Zehnertastatur ein, schaltet das Programm auf Darstellungsfarbe 1 um.

Ein zweiter wichtiger Punkt in diesem Funktionsteil ist das schon angesprochene Setzen und Löschen unseres Grafikcursors. Das dazu notwendige Setzsymbol haben wir ja im Initialisierungsteil schon kennengelernt. Wir müssen dieses nun an die richtige Bildschirmstelle positionieren.

Dies ist insofern etwas problematisch, als der Painter ja in verschiedenen Bildschirmdarstellungsmodi laufen soll. Je nach ausgewähltem Mode ist ein Zeichen unterschiedlich breit. Es besteht also aus unterschiedlich vielen Pixels in der Horizontalen. Insofern muß auch unser Zeichen in den verschiedenen Modi auf eine andere Position gesetzt werden, damit der einen Punkt umfassende Minicursor genau im Zentrum dieses Symbols auftaucht.

Nach welcher Gesetzmäßigkeit dies geschieht, sehen Sie in Zeile 240. Dort erfolgt das erstmalige Setzen unseres Grafikcursors, das Sie auch im Flußdiagramm direkt nach dem Initialisierungsteil finden. Schauen wir uns die einzelnen Befehle etwas näher an.

### 4.2.2 Das Setzen des Grafikcursors

Als erstes finden wir das Setzen des Grafikfarbstiftmodus auf die Funktion 1, also die XOR-Verknüpfung. Nun wird mit MOVE unser Cursor auf eine Position gelegt, die der obersten linken Bildecke unseres Cursorzeichens entspricht. Die Y-Verschiebung im Verhältnis zum aktuellen Punkt ist dabei 8 Bildschirmlinien nach oben. Diese Angabe ist in allen Modi identisch.

Etwas anders sieht es jedoch mit der X-Verschiebung aus. Je nach angewandtem Mode müssen mehr oder weniger Zeichen hier in der Horizontalen subtrahiert werden. Daher auch der etwas seltsam aussehende Ausdruck für die X-Verschiebung.

Wenn wir nun mit TAG die Zeichenausgabe an der Position des Grafikcursors beginnen, wird eine Zeichenmatrix so auf den Bildschirm gesetzt, daß Ihr oberer linker Bildpunkt mit der Position unseres Grafikcursors übereinstimmt. Durch die vorangehende Verschiebung liegt das Cursorzeichen damit zentral um den aktuell angesprochenen Bildpunkt.

Mit TAGOFF normalisieren wir die Zeichenausgabe wieder. Für die Positionierung sind nun vier Variable wissenswert. X und Y enthalten jeweils die aktuelle Position unseres Grafikcursors, und zwar auf einen Bildpunkt genau.

XA und YA (das A steht hier für alt) enthalten die Positionsangaben vor der letzten Änderung. Diese Zwischenspeicherung ist deshalb notwendig, weil die Abfrage des Joysticks und die Änderung der Joystickposition vor der Rücknahme des Fadenkreuzes in Zeile 360 vollzogen wird.

Die Joystickabfrage muß aber vor dem Rücksetzen des Cursors erfolgen. Wenn wir nämlich einen glatten Durchlauf annehmen und das Löschen unseres Fadenkreuzes vor der Joystickabfrage durchführen, so würde nach dem Setzen unseres Grafikcursors in 530 und dem direkten Rücksprung nach 290 dieser sofort wieder gelöscht. Wir sähen also allerhöchstens noch ein kurzes aber sehr störendes Aufflackern auf dem Bildschirm aber keinen sauber blinkenden Cursor. Die Cursorabfrage kostet jedoch einige Zeit, so daß dieses Problem durch die zwischengeschaltete Abfrage gelöst werden kann.

Da nun aber nach der Joystickabfrage die Variablen X und Y nicht mehr den alten, sondern gegebenenfalls einen geänderten Wert, enthalten, ist es notwendig, das Rücksetzen des Zeichens mit den alten zwischenge-

speicherten X- und Y-Koordinaten durchzuführen. Daher wurden die beiden Variablen XA und YA eingeführt.

### 4.2.3 Die Funktionsroutinen

Neben den Farbänderungen in Zeile 380-470 gibt es noch drei weitere Möglichkeiten, um die Joystickschleife zu verlassen. Durch Druck auf <DEL> kommt man in eine Funktion, die den Funktionszähler löscht. Mehr dazu gleich.

Druck auf <CLR> führt uns in ein größes Funktionswahlmenü. Sie finden dieses und die dazugehörigen Funktionen ab Zeile 1240. Zunächst wird das Maschinenprogramm ab Adresse 40000 aufgerufen, das heißt es wird auf den unteren Grafikspeicher umgeschaltet. Wir erinnern uns: Das Zeichnen erfolgte ja im oberen Speicher. Die nun folgenden Ausgaben für Funktionswahl, Bildschirm löschen etc. beeinflussen also unser hochauflösendes (Hi-Res) Grafikbild nicht mehr.

Die verschiedenen Funktionen finden Sie in Zeile 1250-1330 aufgelistet. Hier erfolgt allerdings nicht das eigentliche Setzen der Funktionen, sondern es wird nur die Variable FU auf einen Wert gesetzt. Dieser dient dann beim Aufruf der Funktion mit COPY dazu, festzustellen, welche Funktion benutzt werden soll.

Allerdings gibt es auch hier zwei Abweichungen. Durch Druck auf das Minuszeichen beziehungsweise den Hochpfeil gelangen wir nämlich in zwei weitere Unterfunktionen. Dieses ist einmal das Setzen des Cursors. Wir haben schon bei den Vorbemerkungen zur Programmentwicklung gesagt, daß es etwas aufwendig ist und relativ lange dauert, den Cursor über den gesamten Bildschirm zu bewegen. Daher wurde diese Sonderfunktion eingebaut. Hier sind die neuen Cursorkoordinaten wie gewohnt mit X-Werten zwischen 0 und 640 und Y-Werten zwischen 0 und 400 anzugeben.

Ein weiterer Funktionsteil (bei Hochpfeil) dient dazu, Sonderfunktionen, wie das Sichern eines Bildes, den Modewechsel und den Farbwechsel durchzuführen. Ab Zeile 1670 finden Sie diese Funktion dann aufgelistet. In jedem Fall wird nach der Ausführung dieser Unterfunktion wieder auf unseren Normalbildschirm zurückgeschaltet, wobei gegebenenfalls auch der Modus wieder gewechselt werden muß.

Die Abfragen ab Zeile 1240 laufen nämlich alle im Mode 1 ab. Wenn man also ein Grafikbild im Mode 0 entworfen hat, ist jetzt eine Umschaltung nötig.

Wir wollen uns nun noch einmal anschauen, wie die eigentliche Funktionswahl abläuft. Im Menü "Funktion definieren" haben wir die Variable FU auf einen Wert gesetzt, der der auszuführenden Funktion entspricht. Bei 0 soll also die Funktion "Punkte setzen" ausgeführt werden. Bei 1 springt der CPC das Unterprogramm "Linie ziehen" an. Ist FU=2 so wird ein geschlossener Linienzug gezeichnet.

Bei der Funktion Linienzug wird also nicht jede Einzellinie durch zwei Punkte definiert, sondern der Endpunkt der ersten Linie ist auch gleichzeitig Ausgangsbasis für die nächste. Dies ermöglicht es, auch relativ komplexe Figuren, die aus verbundenen Linien bestehen, relativ einfach zu zeichnen.

Das Setzen der einzelnen Bildpunkte geschieht dabei mit <FIRE> beziehungsweise dem Druck auf die <COPY>-Cursortaste. Im Flußdiagramm ist dies die vierte Raute von oben. Im eigentlichen Programmquelltext liegt dieser Funktionsteil ab Zeile 600. Zunächst einmal schalten wir auf Normaldarstellung unseres Grafikcursors zurück. Wir desaktivieren also die XOR-Verknüpfung. In Abhängigkeit von FU wird dann eines der nachfolgenden Funktionsprogramme ausgeführt.

Hier sind zwei Programmvarianten zu unterscheiden. Zunächst einmal gibt es wie beim "Punkte setzen" die Möglichkeit, daß ein Tastendruck bereits genügt, um die Funktion auszuführen. In unserem Fall wird hier einfach ein PLOT mit der aktuellen Farbe (F) durchgeführt.

Alle anderen Funktionen allerdings benötigen zwei Tastendrücke. Es müssen nämlich in jedem Fall ein Anfangs- und Endpunkt - beziehungsweise zwei Ecken bei der Rechteckdarstellung - angegeben werden. Der Anfangspunkt beziehungsweise die erste Ecke müssen dabei beim ersten Druck auf COPY gespeichert werden. Gleichzeitig sollte auch zur Kontrolle ein Punkt auf dem Bildschirm erscheinen.

Der zweite Tastendruck macht dann die Figur perfekt. Wir wollen diesen Ablauf einmal am Beispiel der Unterroutine "Linie ziehen" durchspielen. Zeile 680 kontrolliert zunächst, ob die Funktion überhaupt angesprochen werden soll. Wenn nicht, werden die weiteren Funktionen überprüft. In unserem Fall ist laut Annahme FU = 1, und so wird dieser Programmteil weiter durchlaufen.

Der nächste Schritt ist nun vom Zustand der Variablen PU abhängig. Diese enthält als FLAG die Aussage darüber, ob bereits ein Punkt für eine Figur eingegeben wurde. Wenn ja, ist jetzt die Linie zu zeichnen. Dies geschieht durch Sprung nach Zeile 730.

Wenn nicht, wird in FA der Farbcode des aktuellen Punktes, das heißt des ersten Bildpunktes unserer Figur gespeichert. Danach wird dieser mit der aktuellen Bildschirmfarbe belegt. Seine Position wird daraufhin in den Variablen X1 und Y1 zwischengespeichert. Das FLAG PU wird auf 1 gesetzt und nach einer Zeitverzögerung geht es zurück in die Joystick-abfrage.

Die Zeitschleife ist dabei deshalb notwendig, weil der Programmdurchlauf aufgrund seiner optimierten Struktur relativ schnell erfolgen kann. Dadurch wäre es möglich, daß Sie mit einem etwas längeren Tastendruck auf <COPY> ein zweimaliges Anspringen dieser Routine auslösen könnten.

Nach dem erstmaligen Ansprung haben wir nun also in X1 und Y1 die eine Ecke beziehungsweise den Beginn unserer Linie gespeichert und FA enthält die dazugehörige Farbe.

Beim zweiten Anlauf sieht die Sache nun etwas anders aus. Nach PU = 1 erfolgt die Verzweigung nach 730. Zunächst wird unser Ausgangspunkt wieder mit der alten Farbe gesetzt. Es folgt das Setzen des Grafikcursors auf die alte Position.

Danach wird eine Linie zur aktuellen Position unseres Grafikcursors auf dem Bildschirm mit der aktuellen Farbe F gezogen. PU wird zurückge-setzt und nach einer Zeitverzögerung, die denselben Effekt hat wie die eben genannte, geht es wieder zurück in die Joystickabfrage.

Auch die anderen Funktionen, die mehrere Angaben benötigen, werden nach demselben Prinzip behandelt.

Nun ist es natürlich auch möglich, daß man gerade einen Anfangs-Punkt gesetzt hat und erst nach dem Druck auf <COPY> beziehungsweise <FIRE> feststellt, daß dieser ja an der falschen Position liegt. Dazu dient die DEL-Taste, die die Routine "Funktionszähler auf 0 setzen", ab Zeile 1180 anspringt.

Ihre Aufgabe besteht darin, PU = 0 zu setzen und die Farbe des gerade belegten Punktes wieder zu normalisieren, das heißt auf ihren vorherigen Wert zu setzen. Dies geschieht in Zeile 1190.

Nach Druck auf <DEL> ist also wieder ein Setzen beider Punkte notwendig.

### 4.2.4 Erweiterungen

Mit CPC PAINT ist man allerdings nicht auf die sechs hier programmierten Funktionen beschränkt. Es ist natürlich problemlos möglich, auch andere Unterroutinen, die wir bereits kennengelernt haben, wie zum Beispiel das Setzen von Kreisen oder Ellipsen und auch das Zeichnen eines regelmäßigen Polygons auf einfachen Tastendruck durchzuführen.

Dazu müssen Sie nur im Menü "Funktion definieren", ab Zeile 1310 die neue Funktion ausführen, ihr einen anderen Wert von FU zuweisen und danach ein kleines Unterprogramm, das auf die entsprechenden FU-Werte reagiert, beispielsweise ab Zeile 1130 in den Funktionsspielplan einfügen. Die Abfrage der einzelnen Funktionsunterpunkte kann dabei, wie beim einfachen Linienziehen beschrieben, erfolgen.

Unbedingt notwendig sind derartige Erweiterungen jedoch nicht. Auch in der hier beschriebenen Variante läßt sich nämlich schon sehr viel erreichen. Einen Eindruck davon, was möglich ist, liefert Ihnen das Bild mit der Gebirgslandschaft im Mittelteil dieses Buches. Es wurde mit dem Programm in ca. einer halben Stunde erstellt.

## Listing 14: *CPC PAINT*

```
10 fu=5
20 ' ******************
30 ' ** CPC-Painter **
40 ' **  28.6.1985  **
50 ' **      by      **
60 ' **   Carsten    **
70 ' **   Straush    **
80 ' ******************
90 ' ************************
100 ' ** Initialisierung **
110 ' ************************
120 CLS:MEMORY &3FFF
130 SPEED WRITE 1
140 DATA 3e,40,cd,08,bc,c9,3e,c0,cd,08,bc,c9,21,00,00,cd,05,bc,c9
,x
150 FOR i=40000 TO 41000:READ a$:IF a$<>"x" THEN POKE i,VAL("&"+a
$):NEXT i
160 CALL 40012:' &4000=call 40000                                &c000=ca
ll 40006
170 GOSUB 1910
180 SYMBOL 255,0,&X1000,&X1000,0,&X1111111,0,&X1000,&X1000
190 '** 2.Symbol **
200 'SYMBOL 255,&X11000,&X111100,&X1100110,&X11000011,&X11000011,
&X1100110,&X111100,&X11000
210 x=320:y=200:xa=320:ya=200
220 CLS:mo=1:pu=0
230 f=1:fa=TEST(x,y):PLOT 660,500,f
240 PRINT CHR$(23)+CHR$(1);:MOVE x-16+8*mo,y+8:TAG:PRINT CHR$(255
);:TAGOFF
250 PLOT x,y,f+1
260 ' ************************
270 ' ** Routine Joypos **
280 ' ************************
290 z=0:IF (JOY(0) AND 4=4) OR INKEY(8)=0 THEN x=x-1:IF x<0 THEN
x=639:y=y+1:z=1
300 IF (JOY(0) AND 8=8) OR INKEY(1)=0 THEN x=x+1:IF x>639 THEN x=
0:y=y-1:z=1
310 IF (JOY(0) AND 1=1) OR INKEY(0)=0 THEN y=y+1:z=1
320 IF (JOY(0) AND 2=2) OR INKEY(2)=0 THEN y=y-1:z=1
330 y= MAX(MIN(y,399),0)
340 PLOT xa,ya,f+1
350 PLOT 700,500,f
360 PRINT CHR$(23)+CHR$(1);:MOVE xa-16+8*mo,ya+8:TAG:PRINT CHR$(2
55);:TAGOFF
370 IF z=1 THEN 520
380 IF INKEY(15)=0 THEN f=0
390 IF INKEY(13)=0 THEN f=1
400 IF INKEY(14)=0 THEN f=2
410 IF INKEY(5)=0 THEN f=3
420 IF INKEY(20)=0 THEN f=4
430 IF INKEY(12)=0 THEN f=5
440 IF INKEY(4)=0 THEN f=6
450 IF INKEY(10)=0 THEN f=7
460 IF INKEY(11)=0 THEN f=8
470 IF INKEY(3)=0 THEN f=9
480 IF INKEY(79)=0 THEN GOSUB 1180
490 IF INKEY(9)=0 OR (JOY(0) AND 16=16) THEN GOSUB 600
500 IF INKEY(16)<>0 THEN 520
510 GOSUB 1240
520 PLOT 660,500,f
530 PRINT CHR$(23)+CHR$(1);:MOVE x-16+8*mo,y+8:TAG:PRINT CHR$(255
);:TAGOFF
```

```
540 PLOT x,y,f+1
550 xa=x:ya=y
560 GOTO 290
570 ' ***************************
580 ' ** Funktion ausfuehren **
590 ' ***************************
600 PRINT CHR$(23)+CHR$(0);
610 ' -------------------
620 ' ** Punkt setzen **
630 ' -------------------
640 IF fu=0 THEN PLOT x,y,f:RETURN
650 ' -------------------
660 ' ** Linie ziehen **
670 ' -------------------
680 IF fu<>1 THEN 790
690 IF pu=1 THEN 730
700 fa=TEST(x,y):PLOT x,y:x1=x:y1=y:pu=1
710 FOR t=1 TO 250:NEXT t
720 RETURN
730 IF pu=1 THEN PLOT x1,y1,fa:MOVE x1,y1:DRAW x,y,f:pu=0
740 FOR t=1 TO 250:NEXT t
750 RETURN
760 ' ----------------
770 ' ** Linienzug **
780 ' ----------------
790 IF fu<>2 THEN 870
800 IF pu=0 THEN fa=TEST(x,y):PLOT x,y:x1=x:y1=y:pu=1:RETURN
810 IF pu=1 THEN PLOT x1,y1,fa
820 MOVE x1,y1:DRAW x,y,f
830 x1=x:y1=y:RETURN
840 ' ---------------
850 ' ** Strahlen **
860 ' ---------------
870 IF fu<>3 THEN 930
880 IF pu=0 THEN fa=TEST(x,y):PLOT x,y:x1=x:y1=y:pu=1:RETURN
890 IF pu=1 THEN PLOT x1,y1,fa:MOVE x1,y1:DRAW x,y,f:RETURN
900 ' ---------------
910 ' ** Rechteck **
920 ' ---------------
930 IF fu<>4 THEN 1040
940 IF pu=1 THEN 980
950 fa=TEST(x,y):PLOT x,y:x1=x:y1=y:pu=1
960 FOR t=1 TO 250:NEXT t
970 RETURN
980 PLOT x1,y1,fa
990 MOVE x1,y1:DRAW x,y1,f:DRAW x,y:DRAW x1,y:DRAW x1,y1
1000 pu=0:RETURN
1010 ' --------------------------
1020 ' ** Rechteck ausgefuellt **
1030 ' --------------------------
1040 IF fu<>5 THEN 1140
1050 IF pu=1 THEN 1090
1060 fa=TEST(x,y):PLOT x,y:x1=x:y1=y:pu=1
1070 FOR t=1 TO 250:NEXT t
1080 RETURN
1090 PLOT x1,y1,fa
1100 FOR i=MIN(y,y1) TO MAX(y,y1)
1110 MOVE x1,i:DRAW x,i,f
1120 NEXT
1130 pu=0:RETURN
1140 RETURN
1150 ' ********************************
1160 ' ** Funktionenzaehler auf Null **
1170 ' ********************************
1180 PRINT CHR$(23)+CHR$(0);
1190 pu=0:PLOT x1,y1,fa
```

```
1200 RETURN
1210 ' **************************
1220 ' ** Funktion definieren **
1230 ' **************************
1240 CALL 40000:MODE 1
1250 PRINT CHR$(10)+"                Funktionswahl"
1260 PRINT:PRINT:PRINT"  Punkte setzen     (0)
1270 PRINT:PRINT"  Linien ziehen     (1)
1280 PRINT:PRINT"  Linienzug         (2)
1290 PRINT:PRINT"  Strahlen          (3)
1300 PRINT:PRINT"  Rechteck          (4)
1310 PRINT:PRINT"  Rechteck (voll)   (5)
1320 PRINT:PRINT"  Cursor setzen     (-)
1330 PRINT:PRINT"  weitere           (^)
1340 z$=INKEY$:IF z$="" THEN 1340
1350 IF z$="-" THEN 1420
1360 IF z$="^" THEN 1510
1370 IF ASC(z$)<48 OR ASC(z$)>53 THEN 1340
1380 pu=0:fu=VAL(z$):MODE mo:CALL 40012:CALL 40006:RETURN
1390 ' **********************
1400 ' ** Cursor setzen **
1410 ' **********************
1420 CLS
1430 PRINT"X-Koordinate";x
1440 PRINT"Y-Koordinate";y
1450 INPUT"Cursorposition X-Koordinate";x
1460 INPUT"Cursorposition Y-Koordinate";y
1470 MODE mo::CALL 40012:CALL 40006:RETURN
1480 ' *************************
1490 ' ** Spezialfunktionen **
1500 ' *************************
1510 CLS:PRINT:PRINT"                Funktionswahl"
1520 PRINT:PRINT:PRINT"  Bild laden     (1)
1530 PRINT:PRINT"  Bild sichern  (2)
1540 PRINT:PRINT"  Modewechsel   (3)
1550 PRINT:PRINT"  Farbwechsel   (4)
1560 PRINT:PRINT"  Programmende  (5)
1570 PRINT:PRINT"  Bild malen    (6)
1580 z$=INKEY$:IF z$="" THEN 1580
1590 IF z$="6" THEN 1620
1600 IF ASC(z$)<49 OR ASC(z$)>53 THEN 1580
1610 ON VAL(z$) GOSUB 1700,1760,1820,2150,1640
1620 CLS:MODE mo:CALL 40012:CALL 40006
1630 RETURN
1640 PRINT:PRINT"Sind Sie sicher j/n"
1650 z$=INKEY$:IF z$="" THEN 1650
1660 IF z$="n" THEN 1620 ELSE END
1670 ' ******************
1680 ' ** Bild laden **
1690 ' ******************
1700 INPUT"Bitte geben Sie den Namen des Bildes ein";n$
1710 LOAD""+n$,&C000
1720 RETURN
1730 ' ********************
1740 ' ** Bild sichern **
1750 ' ********************
1760 INPUT"Bitte geben Sie den Namen des Bildes ein";n$
1770 SAVE""+n$,b,&C000,&4000
1780 RETURN
1790 ' ******************
1800 ' ** Modewechsel **
1810 ' ******************
1820 CLS
1830 INPUT"Neuer Mode";mo
1840 IF mo<0 OR mo>2 THEN PRINT"falsche Eingabe!!":GOTO 1830
1850 MODE mo
```

```
1860 CALL 40006:MODE mo:CALL 40000:MODE 1
1870 RETURN
1880 ' ***************************
1890 ' ** Farbinitialisierung **
1900 ' ***************************
1910 DIM f(15,2)
1920 f(0,1)=0:f(0,2)=0
1930 f(1,1)=24:f(1,2)=24
1940 f(2,1)=6:f(2,2)=6
1950 f(3,1)=2:f(3,2)=2
1960 f(4,1)=21:f(4,2)=21
1970 f(5,1)=15:f(5,2)=15
1980 f(6,1)=7:f(6,2)=7
1990 f(7,1)=4:f(7,2)=4
2000 f(8,1)=27:f(8,2)=27
2010 f(9,1)=18:f(9,2)=18
2020 f(10,1)=8:f(10,2)=8
2030 f(11,1)=17:f(11,2)=17
2040 f(12,1)=9:f(12,2)=9
2050 f(13,1)=11:f(13,2)=11
2060 f(14,1)=26:f(14,2)=26
2070 f(15,1)=24:f(15,2)=6
2080 FOR i=0 TO 15
2090 INK i,f(i,1),f(i,2)
2100 NEXT i
2110 RETURN
2120 '***************
2130 '** Farbwahl **
2140 '***************
2150 IF mo=1 THEN j=3
2160 IF mo=0 THEN j=15
2170 IF mo=2 THEN j=1
2180 MODE mo
2190 PRINT"   Farbdefinition:":PRINT:PRINT CHR$(24):PRINT"Farbe 0
"+CHR$(24)
2200 FOR i=1 TO j:PEN i:PRINT"Farbe";i:NEXT i
2210 PRINT
2220 PEN 1
2230 PRINT"Farbe wechseln j/n?"
2240 z$=LOWER$(INKEY$):IF z$="n" THEN MODE 1:RETURN ELSE IF z$<>"
j" THEN 2240
2250 INPUT"Welches Farbregister";reg
2260 INPUT"Erste  Farbe";f(reg,1)
2270 INPUT"Zweite Farbe";f(reg,2)
2280 INK reg,f(reg,1),f(reg,2)
2290 CLS:GOTO 2190
```

# 5 Titelgrafik und bewegte Figuren - Blockgrafik und HiRes kombiniert

Kombination ist Trumpf. Dies gilt nicht nur für die einzelnen Möglichkeiten innerhalb der verschiedenen Bereiche, sondern auch besonders für die Verbindung von hochauflösender Grafik und Blocksymbolen beziehungsweise Steuerzeichen. Einige Spezialitäten aus den verschiedenen Bereichen gelangen gar erst in der Kombination miteinander zu ihrer vollen Wirkung.

Bevor wir daher im nächsten Kapitel in die Tiefen unseres Speichers hinabtauchen und uns die Grafik sozusagen von unten anschauen, sollen an dieser Stelle noch ein paar Worte zu den Kombinationsmöglichkeiten gesagt werden. Als Anwendungsbeispiele wollen wir dabei die Titelgrafik sowie den Bereich der bewegten Grafik betrachten.

## 5.1 Titelgrafik

Beginnen wir mit der Titelgrafik. Dieser Begriff meint zunächst einmal Titelausgabe, die Vorstellung eines Programm- oder Spielnamens. Wenn wir uns hier auf die Möglichkeiten der Blockgrafik beschränken, so können wir maximal eine Fläche von der Größe eines Zeichens im Mode 0 mit Farbe füllen. Die dadurch erzielbare Zeichengröße wirkt jedoch auf dem Gesamtschirm meist etwas kläglich.

Soll unsere Schrift bildschirmfüllend erscheinen, so müssen wir hier auf Abhilfe sinnen. Dazu gibt es mehrere Möglichkeiten. Zunächst können wir natürlich auf unsere Zeichenadditionstricks aus Kapitel 2 zurückgreifen und mehrere umdefinierte Symbole zu einer größeren Schrift addieren. Dies ist jedoch meist zu aufwendig, wenn man auf Hilfsprogramme wie den DESIGNER verzichten will. Sehr schöne und auch individuelle Titelgrafiken kann man jedoch mit der hochauflösenden Grafik erzeugen.

Eine hübsche Anwendung der hochauflösenden Grafik besteht nämlich darin, vergrößerte Zahlen und Buchstaben auf dem Bildschirm darzustellen. Damit sind dann sehr flexible Titel machbar. Sei es nun ein verschnörkelter Titel für ein mittelalterliches Adventure, die zackig-futuristische Überschrift für ein neues Weltraum-Action-Spiel oder auch nur die Namensausgabe eines Dateiverwaltungsprogrammes.

Mit den Befehlen DRAW und MOVE sind vergrößerte Buchstaben und Zahlen sehr leicht darstellbar. Die Erzeugung kann dabei in zwei grundsätzlichen Varianten erfolgen. Zum einen werden nur die Konturen des jeweiligen Zeichens mittels einfacher Liniengrafik gezeichnet. Bei der anderen Variante wird auch der Körper des Zeichens mit der entsprechenden Schriftfarbe ausgefüllt.

Als eine weitere sehr effektvolle Möglichkeit kann man ein ausgefülltes Zeichen dann auch noch andersfarbig umrahmen. Wählt man in diesem Zusammenhang eng beieinanderliegende Farben wie zum Beispiel dunkel- und mittelblau, so kommen Schatteneffekte zustande, und das Zeichen wirkt plastisch. Wir wollen uns nun einmal anschauen, wie wir solche Großzeichen erzeugen können.

Weist der zu entwerfende Buchstabe geradlinige Begrenzungen auf, so kann man es relativ einfach mit dem DRAWR-Befehl erzeugen. Den Anfangspunkt unseres Zeichens legen wir damit bei einem absoluten MOVE fest und ziehen dann mit DRAWR die Begrenzungslinien.

Wollen wir zum Beispiel den Text "CPC 464" demonstrativ auf dem Bildschirm darstellen, so bietet es sich an, in der oberen Reihe CPC und darunter 464 abzubilden.

Wir bewegen dazu den GRAFIK-Cursor auf den linken Fußpunkt unseres ersten Characters in der oberen Zeile und ziehen dann die entsprechenden Begrenzungslinien. Dasselbe Spiel wiederholt sich dann für die anderen fünf Zeichen. Das kleine Beispielprogramm "Großbuchstaben" demonstriert das Prinzip einer solchen Darstellung.

**Programmbeschreibung:**

Durch die REM-Markierungen sind die verschiedenen Buchstaben gut zu erkennen. Ein Basispunkt jedes Zeichens, normalerweise die linke untere Ecke, wird mit MOVE beziehungsweise MOVER gesetzt. DRAWR-Kommandos sorgen dann für die eigentliche Darstellung.

**Listing 15:** *Programm Großbuchstaben*

```
10 REM ********************************
20 REM ** Grossbuchstaben mit DRAW **
30 REM ********************************
40 INK 0,0:INK 1,2:INK 2,6:INK 3,24:BORDER 0
50 CLS
60 REM *******
70 REM ** C **
80 REM *******
90 ORIGIN 30,0:MOVE 100,280:DRAWR 0,100,1:DRAWR 60,0:DRAWR 0,-20:
DRAWR-40,0
100 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
110 REM *******
120 REM ** P **
130 REM *******
140 MOVER 150,0:DRAWR 0,100:DRAWR 60,0:DRAWR 0,-50:DRAWR -40,0:DR
AWR 0,-50
150 DRAWR-20,0
160 MOVER 20,65:DRAWR 0,20:DRAWR 20,0:DRAWR 0,-20:DRAWR -20,0
170 REM *******
180 REM ** C **
190 REM *******
200 MOVE 400,280:DRAWR 0,100:DRAWR 60,0:DRAWR 0,-20:DRAWR-40,0
210 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
220 REM *******
230 REM ** 4 **
240 REM *******
250 MOVE 150,100:DRAWR 0,80:DRAWR -35,-35:DRAWR 50,0
260 DRAWR 0,-12:DRAWR -80,0:DRAWR 65,65:DRAWR 15,0
270 DRAWR 0,-100:DRAWR -15,0
280 REM *******
290 REM ** 6 **
300 REM *******
310 MOVER 100,0:DRAWR 60,0:DRAWR 0,55:DRAWR -45,0:DRAWR 0,30:DRAW
R 45,0
320 DRAWR 0,15:DRAWR -60,0:DRAWR 0,-100
330 MOVER 20,17:DRAWR 0,20:DRAWR 20,0:DRAWR 0,-20:DRAWR -20,0
340 REM *******
350 REM ** 4 **
360 REM *******
370 MOVER 174,-15:DRAWR 0,80:DRAWR -35,-35:DRAWR 50,0
380 DRAWR 0,-12:DRAWR -80,0:DRAWR 65,65:DRAWR 15,0
390 DRAWR 0,-100:DRAWR -15,0
400 GOTO 400
```

Um sich mit dieser Art der Programmmierung vertraut zu machen, sollten Sie ruhig einmal mit den Angaben in den DRAW-Kommandos spielen. Die Wirkung der einzelnen Buchstaben hängt nämlich in starkem Maße von der Proportionalität der einzelnen Buchstabenlinien zueinander ab. Man muß bei den ersten Versuchen in diesem Bereich daher erst ein Gefühl dafür entwickeln, welche Proportionen gut zusammenwirken.

Nun können wir aber auch mit anderen Schriftarten operieren. Hier werden nur einige Beispiele als Anregung angeführt. Diese sollten Sie allerdings keineswegs in Ihrem Forscherdrang einengen, sondern nur - wie auch die anderen vorgestellten Grafikanwendungen in diesem Kapitel - als Basis für eigene Experimente dienen.

Etwas schwieriger wird es schon, wenn wir statt einer normalen Linie ausgefüllte Zeichen verlangen. Dazu müssen wir nämlich statt einer Linie mehrere Linien nebeneinander ziehen, um so den Körper des Zeichens mit der Schriftfarbe zu füllen. Das Prinzip kennen wir schon von unseren ausgefüllten Rechtecken aus Kapitel 3.

Am einfachsten können wir ausgefüllte Zeichen in einer Schleife erzeugen, in der wir den Fußpunkt unserer Zeichen ständig verschieben und den DRAW-Befehl wiederholen. Wir arbeiten also denselben DRAW-Befehl von verschiedenen Ausgangspositionen aus, mehrmals ab. Dies ist eine sehr Platz-, das heißt Programmzeilensparende Möglichkeit, um schnell ausgefüllte Flächen zu erhalten.

Es sind damit aber auch interessante Effekte erzielbar. Verschieben wir zum Beispiel den Fußpunkt unseres Zeichens um je einen Bildpunkt nach rechts unten, so erhalten wir etwas verzerrte, aber auch teilweise abgerundete Zeichen mit doppelter Strichstärke. Dieser Effekt läßt sich noch verstärken, indem man dasselbe noch mal nach links oben wiederholt.

Den entgegengesetzten Effekt erzielt man, indem man die Verschiebung nach rechts oben, beziehungsweise links unten, vornimmt. Je nach gewünschter Dicke kann man natürlich auch mehr als nur eine Parallellinie ziehen. Die anderen Linien ergeben sich dann durch Verschieben um zwei, drei, vier etc. Bildpunkte.

Kommen wir nun zu einer weiteren Schriftart, auf die wir in diesem Kapitel eingehen wollen: den umrandeten ausgefüllten Buchstaben. Er stellt nichts anderes als die Kombination der ersten beiden Verfahren dar. Wir malen zuerst den Buchstaben aus und ziehen dann die Umrandung mit einer anderen Farbe noch einmal nach.

Wir haben damit nun vier verschiedene Darstellungsarten beschrieben:

- normale Schrift
- doppelte Schrift durch Verschiebung
- ausgefüllte Schrift
- umrahmte Schrift

Diese Schrifttypen finden Sie am Beispiel für den ersten Buchstaben, das "C" dargestellt. Andere Buchstaben können Sie sich nach demselben Prinzip ja relativ einfach selbst definieren. Das Programm "Schriftdemo" stellt sie en bloc dar.

**Listing 16:** *Programm Schriftdemo*

```
10 REM ******************
20 REM ** Schriftdemo **
30 REM ******************
40 INK 0,0:INK 1,2:INK 2,6:INK 3,24:BORDER 0
50 PAPER 0:CLS:PEN 3
60 LOCATE 15,1:PRINT"Schriftdemo"
70 PEN 2
80 REM **************
90 REM ** einfach **
100 REM **************
110 ORIGIN 0,0:MOVE 150,280:DRAWR 0,100,1:DRAWR 60,0:DRAWR 0,-20:
DRAWR-40,0
120 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
130 LOCATE 9,10:PRINT"normal"
140 REM ******************
150 REM ** verschoben **
160 REM ******************
170 MOVE 400,280:DRAWR 0,100,1:DRAWR 60,0:DRAWR 0,-20:DRAWR-40,0
180 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
190 MOVE 402,279:DRAWR 0,100,1:DRAWR 60,0:DRAWR 0,-20:DRAWR-40,0
200 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
210 LOCATE 23,10:PRINT"verschoben"
220 REM ******************
230 REM ** ausgefuellt **
240 REM ******************
250 FOR i= 1 TO 20:MOVE 150+i,120:DRAWR 0,100,1:NEXT i
260 FOR i= 1 TO 40
270 MOVE 170+i,120:DRAWR 0,20:MOVE 170+i,200:DRAWR 0,20
280 NEXT i
290 LOCATE 6,20:PRINT"ausgefuellt"
300 REM **************
310 REM ** umrahmt **
320 REM **************
330 FOR i= 1 TO 20:MOVE 400+i,120:DRAWR 0,100,1:NEXT i
340 FOR i= 1 TO 40
350 MOVE 420+i,120:DRAWR 0,20:MOVE 420+i,200:DRAWR 0,20
360 NEXT i
370 MOVE 400,120:DRAWR 0,100,3:DRAWR 60,0:DRAWR 0,-20:DRAWR-40,0
380 DRAWR 0,-60:DRAWR 40,0:DRAWR 0,-20:DRAWR -60,0
390 LOCATE 25,20:PRINT"umrahmt"
400 GOTO 400
```

**Programmbeschreibung:**

Als oberste Ausgabe finden Sie noch einmal das einfache Rahmen-C. Darunter die ausgefüllte Variante. Die Verschiebung nach rechts unten, beziehungsweise nach links oben, demonstriert der dritte Programmteil. Verändern wir in diesem die MOVE-Werte entsprechend, so können wir die Verschiebung nach rechts oben/links unten ebenso erhalten. Das letzte Programm gibt dann den Effekt der umrahmten ausgefüllten Figuren wieder.

Als Farbkombination wurde hier ein relativ starker Farbgegensatz (blau gegen gelb) gewählt. Durch einfaches Umdefinieren des Farbregisters 3 in

Zeile 40 (INK 3,11) können Sie aber auch mit anderen Randfarben experimentieren und überprüfen, wie die Farben des CPC im Zusammenspiel wirken. Bei der gerade angegebenen Farbkombination ergibt sich der schon beschriebene Schatteneffekt.

Zum Abschluß unserer Betrachtungen soll nun noch einmal gezeigt werden, wie eine komplette Titelgrafik, die nach diesen Punkten aufgebaut wurde, aussehen kann. Das nun folgende Vorspannprogramm wurde als Titel für ein strategisches Weltraumspiel entwickelt. Besondere Aufmerksamkeit sollten Sie beim Anschauen des Programmtextes dem Hereinrollen des Titels zuwenden.

**Programmbeschreibung:**

Den Anfang unserer Routine bilden einige Farb- und Bildschirmdefinitionskommandos. Als nächstes erfolgt die Ausgabe eines Sternenhimmels. Dieser besteht aus einer einfachen PLOT-Schleife mit Random-Koordinaten. Die übergebenen Parameter werden also mit der Zufallsfunktion RND erzeugt. Daraus resultiert auch das jedesmal unterschiedliche "Sternbild".

In den Zeilen 150-550 wird dann der Programmname ausgegeben. Das Setzen mit DRAW und MOVE entspricht dabei dem schon zum vorherigen Programm Gesagten.

In Zeile 590-610 sehen Sie dann, wie man einen Lauftext in eine solche Titelgrafik integrieren kann. Hier sollte ein Copyright-Vermerk mit einer Laufschrift ausgegeben werden. Diese ist auch kein Problem. Zunächst basteln wir den gesamten auszugebenden Text zu einem Gesamtstring zusammen. Aus diesem werden dann mit einer Schleife nacheinander Teilstücke ausgegeben.

Die Ausgabe beginnt dabei mit einem Leerstring von 20 Zeichen, der am Anfang unseres Strings definiert wurde. Beim nächsten Schleifendurchlauf wird ab dem zweiten Zeichen ausgegeben. Es werden damit 19 Spaces und der erste Buchstabe unseres auszugebenden Textes in der untersten Bildschirmzeile ausgegeben. Dieser erscheint also am rechten Bildschirmrand. Beim dritten Durchlauf sind dann schon zwei Zeichen auf dem Schirm, bis sich unser Ausgabebandwurm immer weiter Zeichen für Zeichen auf den Schirm schiebt.

Nach der Textausgabe wird der Bildschirm dann wieder gelöscht, oder besser gesagt auf "zeichenlosen Sternenhimmel" zurückgeschaltet. Dazu schieben wir den aktuellen Text langsam nach oben aus dem Schirm.

Zunächst muß jedoch erst noch die unterste Bildschirmzeile gelöscht werden. Diese wurde auf Window#1 definiert, so daß ein CLS auf dieses Window die Zeile auf die Hintergrundfarbe definiert (Zeile 610).

Nun wird der Schirm in einer Schleife, die über i von 1 bis 15 läuft, zeilenweise nach oben geschoben. Wir benutzen dabei den CHR$(10)-Code, den wir ja schon aus Kapitel 2.4 kennen. Er bewirkt hier die notwendige Verschiebung.

Die mit CALL aufgerufene Systemroutine wartet dabei bis zu dem Zeitpunkt mit der Bildschirmausgabe, in dem der Strahl der Bildröhre mit der Strahlrückführung beschäftigt ist. In dieser Zeit wird kein Punkt auf dem Schirm gesetzt und deshalb führt eine Änderung des Grafikspeichers, wie sie ein solches Verschieben erzeugt, auch nicht zu einer Verschiebung bei der Darstellung. Das lästige Bildschirmflimmern wird damit ausgeschaltet.

Nach dem Ende dieser Routine ist der Schirm in seinem Ausgangszustand wieder hergestellt. Das Spiel kann beginnen. Einen Eindruck von der Titelgrafik, gibt Ihnen das Photo in der Buchmitte.

## Listing 17: *Programm Titelgrafik*

```
10 ' ******************
20 ' ** Titelgrafik **
30 ' ******************
40 MODE 0
50 INK 0,0:INK 1,24:INK 2,2:INK 3,6:INK 4,1
60 WINDOW#0,1,20,1,22:WINDOW#1,1,20,23,25
70 PAPER#0,0:PAPER#1,0:PEN#0,1:PEN#1,3
80 CLS#0:CLS#1:BORDER 0
90 ' ********************
100 ' ** Sternenhimmel **
110 ' ********************
120 FOR i= 1 TO 75: PLOT 640*RND(1),340*RND(1)+60,1
130 NEXT i
140 ORIGIN 0,0:MOVE 10,220
150 ' ********
160 ' ** O **
170 ' ********
180 DRAWR 0,100,2:DRAWR 60,0:DRAWR 0,-100:DRAWR -60,0:MOVER 20,20
190 ' ********
200 ' ** d **
210 ' ********
220 DRAWR 0,60:DRAWR 20,0:DRAWR 0,-60:DRAWR -20,0
230 MOVE 90,220:DRAWR 0,50:DRAWR 50,0:DRAWR 0,-50:DRAWR -50,0
240 MOVER 20,15:DRAWR 0,20:DRAWR 15,0:DRAWR 0,-20:DRAWR -15,0
250 MOVER -20,-15:MOVER 50,0:DRAWR 0,100:DRAWR 18,0:DRAWR 0,-100
260 DRAWR -18,0
270 ' ********
280 ' ** y **
290 ' ********
300 MOVE 200,220:DRAWR -30,50:DRAWR 15,0:DRAWR 25,-39:DRAWR 25,39
:DRAWR 15,0
310 DRAWR -60,-100:DRAWR -15,0:DRAWR 25,50
320 ' ********
330 ' ** s **
340 ' ********
350 MOVE 270,220:DRAWR 60,0:DRAWR 0,30:DRAWR -50,0:DRAWR 0,10
360 DRAWR 50,0:DRAWR 0,10:DRAWR -60,0:DRAWR 0,-27:DRAWR 50,0:DRAW
R 0,-10
370 DRAWR -50,0:DRAWR 0,-10
380 ' ********
390 ' ** s **
400 ' ********
410 MOVE 350,220:DRAWR 60,0:DRAWR 0,30:DRAWR -50,0:DRAWR 0,10
420 DRAWR 50,0:DRAWR 0,10:DRAWR -60,0:DRAWR 0,-27:DRAWR 50,0:DRAW
R 0,-10
430 DRAWR -50,0:DRAWR 0,-10
440 ' ********
450 ' ** e **
460 ' ********
470 MOVE 430,220:DRAWR 60,0:DRAWR 0,10:DRAWR -50,0:DRAWR 0,10
480 DRAWR 50,0:DRAWR 0,30:DRAWR -60,0:DRAWR 0,-50:MOVER 10,30
490 DRAWR 40,0:DRAWR 0,10:DRAWR -40,0:DRAWR 0,-10
500 ' ********
510 ' ** e **
520 ' ********
530 MOVE 510,220:DRAWR 60,0:DRAWR 0,10:DRAWR -50,0:DRAWR 0,10
540 DRAWR 50,0:DRAWR 0,30:DRAWR -60,0:DRAWR 0,-50:MOVER 10,30
550 DRAWR 40,0:DRAWR 0,10:DRAWR -40,0:DRAWR 0,-10
560 ' ******************
570 ' ** Textausgabe **
580 ' ******************
```

```
590 z$=SPACE$(20)+"ein Space - Adventure  "+CHR$(164)+" 1985 by C
.S. Alle Rechte der Verbreitung gleich welcher Form vorbehalten!"
600 FOR i= 1 TO LEN(z$)-20:LOCATE#1,1,2:GOSUB 670:GOSUB 670:PRINT
#1,MID$(z$,i,20):FOR t= 1 TO 15:GOSUB 670:NEXT t,i
610 FOR t= 1 TO 250:GOSUB 670:NEXT t:CLS#1
620 ' ***************
630 ' ** Scrollen **
640 ' ***************
650 FOR i= 1 TO 15:GOSUB 670:LOCATE#0,1,22:CALL &BD19:GOSUB 670:P
RINT#0,"
"
660 FOR j= 1 TO 3:PLOT RND(1)*640,RND(1)*16+48,1:NEXT j,i
670 FOR t=1 TO 50:NEXT t
680 RETURN
```

## 5.2 Bewegte Grafik

Nachdem wir uns ausgiebig mit den Möglichkeiten der hochauflösenden Grafik bei der Titelgestaltung beschäftigt haben, wollen wir uns nun ansehen, wie wir Blockgrafik und HiRes kombinieren können, um damit bewegte Bilder zu erzeugen. Eine Möglichkeit dazu, sofern wir den gesamten Schirm bewegen wollen, haben wir ja gerade schon kennengelernt: das Scrollen des Gesamtbildschirms. In BASIC ist jedoch auch ein Einzelzugriff auf Bildschirmteile oder Einzelzeichen möglich.

Basis für diese Verschiebungen und Bewegungen ist das Kommando TAG, welches es ermöglicht, ein Zeichen an der aktuellen Position des Grafikcursors auszugeben. Wir haben es schon bei der Entwicklung unseres Grafikprogramms im letzten Kapitel kennengelernt.

Der TAG-Befehl benutzt eine Eigenart des CPC im Bereich der Grafikabspeicherung, die Ablage der Bildschirminformationen in Form einzelner Bildpunkte, der Pixels. Jedes Zeichen ist dabei in einer Matrix aus Bildpunkten definiert, die, wenn dieses Zeichen dann auf den Bildschirm gebracht werden soll, einzeln gesetzt werden. Mehr dazu im nächsten Kapitel. Damit ist es dann möglich, Characters nicht nur auf die Bildschirm(text)zeilen, sondern auch zwischen diese zu positionieren.

Der nachfolgende Dreizeiler demonstriert den Zusammenhang. Zuerst wird in den Zeilen 12 und 13 jeweils ein großes "A" untereinander ausgegeben. Eine Spalte weiter rechts erfolgt dann der Ausdruck eines "A" mit dem TAG-Befehl. Dieses wird zwischen die beiden Textzeilen gesetzt. Dabei ist darauf zu achten, daß nach dem PRINT-Befehl ein Semikolon eingegeben wird. Sonst erscheint ein etwas seltsamer Haken auf dem Bildschirm.

```
CLS:LOCATE 20,12: PRINT "A"
LOCATE 20,30: PRINT "A"
ORIGIN 0,0:MOVE 336,200:TAG:PRINT"A;:TAGOFF
```

Da diese Zeilen im Direkt-MODUS stehen, sollten sie in den oberen Bildschirmzeilen eingegeben werden, damit die Textausgaben nicht überschrieben werden.

Durch die Änderung der Position des GRAFIK-Cursors können wir nun mit Hilfe des TAG-Befehls unseren Buchstaben über das Bild wandern lassen. Dies demonstriert das nachfolgende Programm:

```
10 REM ************************
20 REM ** Bewegter Buchstabe **
30 REM ************************
40 TAG
50 ORIGIN 0,0:FOR i=150 TO 200
60 MOVE 150,i:PRINT "<A>";:NEXT i
```

**Programmbeschreibung**

Nach TAG und dem Setzen des Koordinatennullpunktes mit ORIGIN wird der Grafikcursor abwechselnd auf steigende Y-Koordinaten gesetzt und dann wieder ein Probe-"A" ausgegeben. Jeder folgende Buchstabe überdeckt dabei seinen Vorgänger. Es entsteht ein wanderndes Zeichen.

Nun ist es relativ sinnlos, Buchstaben kreuz und quer über den Bildschirm zu jagen. Interessanter ist es da schon, statt dessen ein Raumschiff-Symbol oder ein Männchen - wie in den Codes ab 240 im Handbuch (Anhang III) - als Bewegungsobjekt zu verwenden. Damit sind dann schon auf einfache Weise Action-Spiele und ähnliche Anwendungen programmierbar. Hier nur einmal das Prinzip:

Die Bewegung unserer Figur soll mit Joystick oder Cursortasten erfolgen. Damit bietet sich die Cursorsteuerungsroutine aus dem PAINTER als Eingaberoutine geradezu an. Was nun noch fehlt, ist ein zu setzendes Zeichen. Wir wollen hier einmal auf eine Konstruktion ähnlich unserem Multicolor-Raumschiff-Sprite aus Kapitel 2.4 zurückgreifen. Dabei können wir uns nämlich auch gleich einmal anschauen, wie Bewegungen mit mehreren Zeichen darstellbar sind. Diese beiden Routinen verbinden wir nun und erhalten das folgende Programm:

**Listing 18:** *Programm TAG-Raumschiff*

```
10 ' **********************
20 ' ** TAG-Raumschiff **
30 ' **********************
40 MODE 0
50 INK 0,0:INK 1,27:INK 2,11:INK 3,6:BORDER 0
60 SYMBOL 250,&X0,&X100001,&X1111111,&X100011,&X1,&X1,&X111,&X111
1
70 SYMBOL 251,&X0,&X10000100,&X11111110,&X11000100,&X10000000,&X1
0000000,&X11100000,&X11110000
80 SYMBOL 252,&X11111,&X1111,&X111,&X1111,&X11111,&X111111,&X1100
1,&X0
90 SYMBOL 253,&X11111000,&X11110000,&X11100000,&X11110000,&X11111
000,&X11111100,&X10011000,0
100 a$=CHR$(250)+CHR$(251)
110 b$=CHR$(252)+CHR$(253)
120 ' **********************
130 ' ** Routine Joypos **
140 ' **********************
150 IF (JOY(0) AND 4=4) OR INKEY(8)=0 THEN x=x-1:IF x<0 THEN x=63
9:y=y+1
160 IF (JOY(0) AND 8=8) OR INKEY(1)=0 THEN x=x+1:IF x>639 THEN x=
0:y=y-1
170 IF (JOY(0) AND 1=1) OR INKEY(0)=0 THEN y=y+1
180 IF (JOY(0) AND 2=2) OR INKEY(2)=0 THEN y=y-1
190 y= MAX(MIN(y,399),0)
200 ' **********************
210 ' ** Zeichenausgabe **
220 ' **********************
230 PLOT 700,800,1
240 MOVE x,y:TAG:PRINT a$;:TAGOFF
250 MOVE x,y-16:TAG:PRINT b$;:TAGOFF
260 PLOT x+20,y-28,3:PLOT x+24,y-28:PLOT x+36,y-28:PLOT x+40,y-28
270 PLOT x+20,y-8,2:PLOT x+40,y-8
280 GOTO 140
```

**Programmbeschreibung:**

Am Programmanfang finden Sie wieder einen kleinen Initialisierungsteil, in dem die zu setzenden Farbwerte und die Teile des Raumschiffs (4 Symbole) definiert werden. Gearbeitet wird hier mit einer 2*2 Zeichenmatrix. Es steht uns also für die Definition der Zeichen ein Feld von der Größe von 2*2 normalgroßen Characters zur Verfügung.

Nach der Angabe mit SYMBOL werden die nebeneinander liegenden Teilzeichen dann zu einem oberen Zeichenteil (a$) und einem unteren Zeichenteil (b$) zusammengefügt. Diese müssen wir nun nur noch durch geeignete Setz-Befehle übereinander positionieren, um das gewünschte Symbol zu erhalten.

Zunächst muß nun die aktuelle Cursorposition bestimmt werden. Dazu benutzen wir wie schon gesagt die Routine JOYPOS aus dem PAINTER in leicht abgewandelter Form. Das Setzen von z kann hier entfallen.

Die eigentliche Zeichenausgabe finden Sie dann ab Zeile 230. Zunächst wird durch PLOT außerhalb des erlaubten Bildschirmbereiches (0-639 in

X-Richtung; 0-399 in Y-Richtung) die Grafikfarbe definiert. Danach setzen wir den internen Grafikcursor mit MOVE auf die obere linke Ecke des oberen Doppelzeichens und dieses wird danach gesetzt (Zeile 240). In Zeile 250 wiederholt sich diese Prozedur für das untere Halbzeichen. In beiden Setzvorgängen wird dabei nach der Zeichenausgabe wieder auf normale Textausgabe (TAGOFF) zurückgeschaltet.

Die nun folgenden Zeilen (260 und 270) dienen im wesentlichen der Zeichenverschönerung. Sie machen aus unserem Monocolor-Shape ein 4-farbiges Zeichen. Dazu werden in unserer Zeichenmatrix nun mit gezieltem PLOT einzelne Punkte nach der Ausgabe des Hauptzeichens noch einmal mit einer anderen Farbe gesetzt. Berechnungsgrundlage ist dabei wiederum die linke obere Zeichenecke, die Koordinate (X,Y). Relativ zu dieser werden nun die einzelnen Positionen beziehungsweise Verschiebungen berechnet. Nach der Zeichenausgabe geht es dann wieder zurück nach Zeile 140 in die Cursorabfrage.

Bewegte Grafiken lassen sich allerdings auch noch auf ganz anderen Wegen erstellen. Beispielsweise kann man in einer Figur, zum Beispiel einem sektorierten Kreis die einzelnen Sektoren nacheinander mit verschiedenen Farben belegen und damit eine Bewegung imitieren. Diese optische Täuschung wird dabei als Laterna-magica-Effekt bezeichnet. Wie ein solcher Vorgang konkret ablaufen kann zeigt das kleine Demonstrationsprogramm Laterna magica.

**Programmbeschreibung:**

Das Programm stellt zunächst einen Kreis auf dem Bildschirm dar, wobei die unterschiedlichen Sektoren mit verschiedenen Farben belegt werden. Danach wird in den Zeilen 170-220 die Definition der INK-Register permanent gewechselt. Jeweils nach einer Zeitverzögerung um t werden die Farbregister einmal ringsum umdefiniert. Die Zeitschleife und damit die Periode, mit der die Farbwechsel erfolgen, können dabei laufend durch eine Abfrage in Zeile 160 geändert werden.

Probieren Sie doch hier einmal den Wert 2 aus. Sie erhalten eine langsame Bewegung. Experimentieren Sie nun ruhig einmal mit anderen Eingaben und schauen Sie sich dabei das Ergebnis an. In die Zeitabfrage gelangen Sie übrigens bei "sich drehendem" Kreisel durch Druck auf eine beliebige Taste.

Wir haben nun eine ganze Reihe von Möglichkeiten kennengelernt, mit denen man Bewegungen auf dem CPC erzeugen kann. Wenn Sie diese nun nicht zufriedenstellen, weil der Effekt nicht überzeugend ausgeführt wird

oder die Darstellungsgeschwindigkeit zu gering ist, so bleibt Ihnen nur ein Ausweg: Der Direktzugriff auf die Grafik in Maschinensprache.

Diesem Gebiet sind nun die restlichen Seiten dieses Buches gewidmet. Endziel ist dabei die Entwicklung eines leistungsstarken Spriteeditors. Dazu sind jedoch einige Vorarbeiten nötig. Im nächsten Kapitel steht daher als erstes die Struktur des Grafikspeichers auf dem Plan.

**Listing 19:** *Programm Laterna magica*

```
10 ' ***************
20 ' laterna magica
30 ' ***************
40 auf=0.01122
50 CLS:BORDER 0
60 ORIGIN 320,200
70 FOR i=0 TO 3:INK i,0:NEXT
80 FOR i=0 TO PI/3 STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*COS(i),1:N
EXT
90 FOR i=PI/3 TO 2*PI/3 STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*COS(i
),2:NEXT
100 FOR i=2*PI/3 TO PI STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*COS(i)
,3:NEXT
110 FOR i=PI TO 4*PI/3 STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*COS(i)
,1:NEXT
120 FOR i=4*PI/3 TO 5*PI/3 STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*CO
S(i),2:NEXT
130 FOR i=5*PI/3 TO 2*PI STEP auf:MOVE 0,0:DRAW 50*SIN(i),50*COS(
i),3:NEXT
140 INK 1,24:INK 2,11:INK 3,6
150 WINDOW#2,1,40,23,25
160 INPUT#2,"Zeit";zeit
170 INK 1,24:INK 2,11:INK 3,6
180 FOR t=1 TO zeit:NEXT t
190 INK 1,6:INK 2,24:INK 3,11
200 FOR t=1 TO zeit:NEXT t
210 INK 1,11:INK 2,6:INK 3,24
220 FOR t=1 TO zeit:NEXT t:IF INKEY$="" THEN 170 ELSE 150
```

# 6 Grafik intern: Speicheranalyse

In den bisherigen Kapiteln haben wir schon des öfteren festgestellt, daß der CPC über eine ganze Reihe von Besonderheiten und Eigenarten verfügt, die man bei normalen Homecomputern eigentlich nicht gewohnt ist. Erinnert sei hier nur einmal an die speziellen Verknüpfungstechniken aus Kapitel 4. Der Schneider verfügt über die Möglichkeit, normale Zeichen, also Buchstaben und Blockgrafiksymbole an der Position des Grafikcursors darzustellen, womit wir in Kapitel 5 bereits einige besondere Effekte erzielen konnten.

In diesem Kapitel wollen wir nun einmal hinter die hellgrau/dunkelgraue Fassade unseres Computers schauen, und uns damit beschäftigen, wie die Grafikspeicherung intern abläuft. Wir werden uns dabei auf den Grafikspeicher selbst konzentrieren und die übrige Hardware beiseite lassen.

## 6.1 Der Aufbau des Grafikspeichers in den verschiedenen Modes

Die Abspeicherung der Grafik beim Schneider ist eine Wissenschaft für sich, und um Sie hier gleich vorzuwarnen: Wenn Sie der Meinung sind, daß ein bestimmtes Problem in diesem Bereich auf eine spezielle Art und Weise mit Sicherheit nicht sinnvoll zu lösen ist, so kann es vorkommen, daß Sie gerade hierbei durch den CPC eines Besseren belehrt werden. Die Grafikabspeicherung spiegelt nämlich wirklich guten englischen Spleen wider.

Beginnen wir mit ein paar Basisfakten. Die Grafik ist im variablen Speicher unseres Computers, dem RAM abgelegt, und zwar, sofern Sie hier keine Änderung durchgeführt haben, ab der Adresse 49152 beziehungsweise hexadezimal C000 bis zur Speicherobergrenze (Adresse FFFF). Dieser Speicherbereich umfaßt exakt 16384 Bytes, wobei jedes Byte noch 8 Bits oder 8 Ja/nein-Entscheidungen aufnehmen kann.

Das Konzept, das beim Schneider angewandt wird, lautet MEMORY-MAPPED-PIXEL-SCREEN. Ein Ausdruck, der sich beim besten Willen nicht eindeutschen läßt. Gemeint ist folgendes: Jeder Bildpunkt auf dem Schirm findet sich auch in gewisser Form im Speicher wieder, wobei das RAM quasi als Landkarte für den aufzubauenden Bildschirm fungiert.

Dies läßt uns nichts Gutes ahnen, denn schließlich bedeutet die Landkarteneigenschaft ja, daß unser Grafikschirm nicht mit der eigentlichen Zeichendarstellung oder den Bildpunkten identisch ist. Bei näherem Hinsehen finden wir dies auch bestätigt.

Obwohl die Zeichen in allen drei Bildschirmmodi (Mode 0, Mode 1, Mode 2) gleich aussehen, sind sie dennoch in völlig anderer Art und Weise abgespeichert. Hinzu kommt noch, daß man nicht sagen kann, daß ein Bildpunkt oder eine Reihe von Bildpunkten auf dem Schirm nun durch eine Adresse im Speicher repräsentiert wird. Die Lokalisierung der Bildpunkte im Speicher ist variabel.

Mit ein wenig Systematik können wir diese auf den ersten Blick etwas verzwickte Angelegenheit aber dennoch in den Griff bekommen. Wir beginnen dabei mit einer Groborientierung im Speicher unseres Computers.

### 6.1.1 Groborientierung im Speicher

Für diese sind zwei Zeiger, sogenannte Pointer, verantwortlich. Beide haben einen Namen: *Bildschirmbasis* und *Bildschirmoffset*.

**Die Bildschirmbasis** legt fest, in welchem der vier 16K-Blöcke unseres Speichers der Grafikbildschirm liegen soll. Beim Einschalten ist, wie wir schon gesagt haben, der oberste Speicherblock adressiert, das heißt die Adressen von C000-FFFF.

Wir können jedoch auch die Bildschirmbasis in einen der drei anderen 16K-Blöcke verlagern und dort unseren Grafikspeicher aufbauen. Realitätsnah ist dabei jedoch nur ein Zugriff auf den zweiten Block von Adresse 4000-7FFF.

In den anderen beiden Blöcken liegen nämlich lebensnotwendige Systeminformationen. Ein Versetzen der Bildschirmbasis und gegebenenfalls noch gefolgt vom einem Löschen des Speichers, würde diese Systeminformationen sofort zerstören und hätte damit ein "Abstürzen" des Rechners zur Folge.

**Der Bildschirm-Offset** legt nun innerhalb unseres 16 K Bereichs fest, in welcher Adresse die linke obere Bildschirmecke, der Bildschirmanfang, abgespeichert werden soll.

Auf den ersten Blick wirkt dies etwas konfus, denn man sollte ja eigentlich annehmen, daß im ersten Byte unseres ausgewählten 16K-Blocks auch das erste Pixel gespeichert ist. Beim Einschalten ist dies auch so. Der Bildschirm-Offset ist in diesem Fall 0.

Je länger der CPC arbeitet, desto näher rückt jedoch der Zeitpunkt, an dem der Computer den ersten Bildschirm-Scroll, also das Herabrollen des Schirms um eine Bildschirmzeile ausführt. Aufgrund der internen Speicherarchitektur ist dies mit wenigen Handgriffen, dem Versetzen dieses Zeigers, möglich. Nach dem ersten Scroll des Gesamtschirms stimmt die Speicherbelegung dann nicht mehr.

Für unsere nun folgenden Betrachtungen, wollen wir jedoch von dem Zustand ausgehen, der beim Einschalten vorliegt: Der Grafikspeicher reicht von Adresse hex C000 bis hex FFFF und der Bildschirm-Offset hat den Inhalt 0000, das heißt die Adresse C000 enthält auch das erste Byte unseres Bildschirms, den Bildschirmanfang. Soweit zur Grobgliederung.

### 6.1.2 Die Feinstruktur

Als nächstes wollen wir uns mit der inneren Struktur des Grafikspeichers beschäftigen. Dazu sollten Sie einmal das folgende kleine Demonstrationsprogramm eintippen.

```
FOR I=&C000 TO &FFFF:POKE I,&FF:
LOCATE 1,1:PRINT HEX$(I):NEXT I
```

**Programmbeschreibung:**

Diese kleine Routine demonstriert auf eindrucksvolle Weise den Aufbau unseres Grafikspeichers. Vor dem Eintippen sollten Sie allerdings den Computer mit

```
<CTRL><SHIFT><ESC>
```

in den Ausgangszustand zurückversetzen. Mit Hilfe der POKE-Schleife werden nacheinander, beginnend mit der ersten Speicherstelle, alle Zellen des Grafikspeichers auf den Wert 255 gesetzt. Dieser Wert (hex FF) ist ein Byte, bei dem alle Bits gesetzt sind. Gleichzeitig wird an der Position

(1,1), also in der linken oberen Bildschirmecke, ausgegeben, welche Speicheradresse gerade gesetzt wird.

Und hier sehen Sie nun ein sehr interessantes Phänomen. Es werden nämlich, beginnend mit der obersten Bildschirmlinie eine Reihe roter Linien von links nach rechts auf dem Schirm gezogen.

Normalerweise würden wir dabei nun erwarten, daß nach der ersten roten Linie die zweite gesetzt wird und so weiter. Das Ergebnis unseres Experimentes sieht jedoch anders aus. Statt der zweiten folgt auf die erste die neunte. Nach dieser wird die siebzehnte Zeile gezogen und so weiter im 8-Linien-Rhythmus.

Wenn Sie diesen Vorgang weiter beobachten, so werden Sie feststellen, daß der CPC bei der Adresse C400 ziemlich genau die Bildschirmmitte erreicht und bei C800 wieder von vorne beginnt. Jetzt werden die zweiten Bildschirmlinien aller Bildschirmzeilen gezogen. Dieser Vorgang läuft dann wieder 2000 Bytes weiter bis zur Adresse D000, an der mit der Ausgabe der dritten Pixel-Linie begonnen wird.

Wir haben es also mit einer etwas seltsamen Art der Verkettung der Bildschirmlinien zu tun. Ab dem Bildschirmanfang, in unserem Fall der Adresse C000, liegt die oberste Pixel-Linie. Sie benötigt für ihre Abspeicherung 80 Bytes. Die nächsten 80 Bytes repräsentieren dann die Bildschirmlinie 9 und so weiter. Bild 6.1 gibt diesen Aufbau noch einmal als Diagramm wieder.

| | | |
|---|---|---|
| 1.Linie aller Textzeilen | 1.Bildschirmlinie | C000 --- C04F |
| | 9.Bildschirmlinie | C050 --- C09F |
| | . | |
| | . | |
| | 193.Bildschirmlinie | C780 --- C7CF |
| 2.Linie aller Textzeilen | 2.Bildschirmlinie | C800 --- C84F |
| | 10.Bildschirmlinie | C850 --- C89F |
| | . | |
| | . | |
| | 194.Bildschirmlinie | CF80 --- CFCF |
| 8.Linie aller Textzeilen | 8.Bildschirmlinie | F800 --- F84F |
| | 16.Bildschirmlinie | F850 --- F89F |
| | . | |
| | . | |
| | 200.Bildschirmlinie | FF80 --- FFCF |

**Bild 6.1:** *Die Ablage der Bildschirmlinien im Speicher*

Wenn wir diesen Vorgang nun einmal von der Textseite her betrachten, so brauchen wir also für die oberste Bildschirmlinie aller 25 Text-Zeilen, die wir auf dem Bildschirm darstellen können, einen Block von 80 Bytes * 25 Zeilen = 2000 Bytes. Nach diesen 2000 Bytes schließen sich dann die zweiten Bildschirmlinien aller Textzeilen an.

Dies ist jedoch nicht ganz richtig. Das exakte Maß, das uns für die Abspeicherung einer Bildschirmlinie für alle Textzeilen zur Verfügung steht beträgt nämlich nicht 2000 Bytes, sondern computerkonform 2048 Bytes oder 2 K.

Wir benötigen nun aber nur, wie wir gesehen haben, genau 2000 Bytes. Damit stellt sich sofort die Frage, was mit den restlichen 48 Bytes passiert. Die Antwort ist relativ einfach: Nichts. Diese Adressen bleiben unbesetzt. Sie werden nicht belegt.

Beim Durchlauf unseres kleinen Programms können Sie dies zum Beispiel daran erkennen, daß selbst in dem Moment, wo der gesamte Schirm bereits mit roter Farbe vollgefüllt ist, der Zähler in der oberen linken Bildschirmecke immer noch weiterläuft und zwar um genau jene 48 "überflüssigen" Bytes.

Daneben wird Ihnen vielleicht auch aufgefallen sein, daß wenn die Bildschirmlinie in der untersten Textzeile am unteren Rand angelangt war, eine gewisse Pause eingetreten ist, in der scheinbar nichts passierte. Die Pausenzeit lag dabei ungefähr in der Größenordnung, die ansonsten bei der Bildschirmdarstellung benötigt wurde, um eine 1/2 bis 3/4-Bildschirmlinie auf dem Schirm zu zeichnen.

In beiden Fällen ist der Grund derselbe. In dieser Zeit hat der CPC Adressen gePOKEt, die für die aktuelle Bildschirmdarstellung nicht benötigt wurden und deren Änderung deswegen auch keinen Effekt auf dem Schirm hatte.

Diese Bemerkungen sollten Sie nun aber nicht dazu (ver-)leiten, jene 48 Bytes freudig als freien und sicheren Speicherraum zu betrachten. Denn wie wir schon gesehen haben, kann der CPC ja seinen Bildschirmanfang mit Hilfe des OFFSET-Zeigers verschieben.

In diesem Fall bleiben zwar nach der Abspeicherung eines Zeilenblocks auch wieder 48 Bytes frei. Durch die Verschiebung auf eine andere Anfangsadresse liegen die freien Bytes nun aber in anderen Adressen. Es werden also gegebenenfalls unsere "sicher" abgelegten Informationen überschrieben und an anderer Stelle taucht ein anderer scheinbar sicherer Freiraum auf.

### 6.1.3 Die Abspeicherung der einzelnen Bildpunkte

Wie sieht nun die Abspeicherung der einzelnen Bildpunkte innerhalb einer Bildschirmlinie aus? Zunächst können wir sagen, daß die acht Bits eines Bytes auch acht Bildpunkten entsprechen. Das Byte in Adresse C000 gibt also an, auf welchen Wert die ersten acht Bildpunkte in der obersten Bildschirmlinie gesetzt werden sollen.

Die Art der Definition hängt nun aber von dem Bildschirmdarstellungsmodus ab, in dem wir uns gerade befinden. Am einfachsten ist es in Mode 2. Hier haben wir es noch mit einer heilen Computerwelt zu tun. Das höchstwertige, linke Bit, also Bit 7, gibt auch den Wert des äußerst linken Pixels unseres Bytes an.

Wenn wir also beispielsweise im Mode 2 in Adresse C000 den Binärwert 11001100 abspeichern, so werden die ersten beiden Pixels oder Bildpunkte mit der Farbe 1, besser gesagt mit der im Farbregister 1 gespeicherten Farbe, dargestellt. Die nächsten beiden mit INK 0. Für das zweite Halbbyte wiederholt sich dieser Vorgang dann noch einmal.

Hierbei können wir schon zwei Aussagen festhalten. Auch bei der Abspeicherung des Grafikbildschirms tauchen immer noch keine absoluten Farbwerte auf, das heißt wir haben es immer noch mit der indirekten Farbdefinition über die INK-Register zu tun.

Oder um noch konkreter zu werden, die 1 entspricht nicht der Farbe dunkelblau, die ja den Farbwert 1 trägt, sondern repräsentiert das Farbregister 1. Das am weitesten links liegende Pixel würde also in der Farbe dargestellt, auf die das Farbregister 1 zu diesem Zeitpunkt definiert ist.

Dementsprechend kann auch ein gerade gesetzter Bildschirm sofort farblich verändert werden, indem man nur die INK-Register umdefiniert. Eine Veränderung des Inhaltes des Grafikbildschirms muß dazu nicht erfolgen.

Dieser einfache Zusammenhang zwischen Farbgebung und den einzelnen Bits eines Bytes gilt jedoch nur für den hochauflösenden Mode 2. In den anderen beiden Modi findet nun eine Zusammenfassung der Bildpunkte statt.

Im Mode 1 werden immer zwei Pixels gleichzeitig gesetzt, im Mode 0 sind es sogar vier, die mit derselben Farbe belegt werden. Durch diesen Trick ist der CPC in der Lage, ohne zusätzlichen Speicheraufwand dennoch mehrere Farben darstellen zu können. Unseren zwei zusammengefaßten Bits im Mode 1 entsprächen ja im Mode 2 zwei Bits.

Statt einem Bit haben wir also nach der Zusammenfassung nun zwei Bits zur Verfügung, um die Farbe zu dekodieren. Dies ist auch der Grund dafür, daß wir im Mode 1 mit vier verschiedenen Farben arbeiten können, genau den Kombinationen, die wir mit zwei Bits dekodieren können (00,01,10,11).

Analog sieht die Sache im Mode 0 aus. Die Zusammenfassung von vier Bits schafft uns eine 4-Bit-Gruppe, die wir für die Farbabspeicherung benutzen können. Mit vier Bits lassen sich 16 Kombinationen speichern. Dies gibt uns die 16 adressierbaren Farben, genauer gesagt Farbregister, über die wir im Mode 0 verfügen. Nun ist es leider nicht so, daß wir die zwei Bits, denen im Mode 2 die Einzelbildpunkte entsprachen, im Mode 1 für die Farbdarstellung requirieren können.

Die Farbdecodierung beim CPC ist nämlich wohl der seltsamste Teil dieses Computers. Bild 6.2 gibt den Zusammenhang zwischen den Bits und der Farbe der einzelnen Bildpunkte an. Die einzelnen Bits sind dabei wirklich in der dort angegebenen Reihenfolge für die Farbgebung maßgeblich.

Das Farbregister, das die Farbe für die linken vier Pixels im Mode 0 liefert, dekodieren wir also, indem wir die Bits 1, 5, 3 und 7 des zugehörigen Bytes in dieser Reihenfolge hintereinander schreiben. Der daraus resultierende 4-Bit-Binärwert gibt uns dann das Farbregister an.

**Ein Beispiel:**

Nehmen wir einmal an, wir würden im Mode 0 nach

```
<CTRL><SHIFT><ESC>
```

```
POKE&C000,&X10100101
```

eingeben, also die erste Stelle unseres Bildschirms mit binär 10100101 belegen, so hätte dies folgenden Effekt (vergleiche Bild 6.2): Die linken vier Bildpunkte würden mit INK 5, also schwarz gesetzt, die rechten vier Pixels dagegen mit INK 10 (mittelblau).

Sie können dies relativ einfach überprüfen, indem Sie im Mode 0 die gerade beschriebenen Befehle eingeben. In der linken oberen Bildschirmecke sehen Sie dann ein schwarzes und ein blaues kleines Feld. Dies sind die acht gesetzten Bildpunkte, vier schwarze links und vier blaue rechts. Mit

```
PEN 5
```

und

```
PEN 10
```

können Sie sich dann noch einmal davon überzeugen, daß die Farbkombinationen auch richtig wiedergegeben wurden. Um sich mit dieser wirklich etwas seltsamen Art der Farbabspeicherung vertraut zu machen, sollten Sie nun einmal nach dieser Methode einige Bildpunkte in den anderen Modi definieren und sich dann auf dem Schirm anschauen, was Sie als Ergebnis erhalten.

| | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|
| Linksbündiges Pixel | Bits 1,5,3,7 | Bits 3,7 | Bit 7 |
| . | | | Bit 6 |
| . | | Bits 2,6 | Bit 5 |
| . | | | Bit 4 |
| . | Bits 0,4,2,6 | Bits 1,5 | Bit 3 |
| . | | | Bit 2 |
| . | | Bits 0,4 | Bit 1 |
| Rechtsbündiges Pixel | | | Bit 0 |

**Bild 6.2:** *Die Farbcodierung*

## 6.2 Von der Theorie zur Praxis

Mit dem im letzten Unterkapitel erworbenen Wissen sind wir nun in der Lage eine Reihe interessanter Anwendungen zu realisieren. Zuerst wollen wir auf die Umschaltung von Grafikbereichen zu sprechen kommen und uns etwas näher mit den Routinen SCR SET BASE und SCR SET OFFSET beschäftigen.

Unter beiden Aufrufen hat der CPC Unterprogramme abgespeichert, die es uns ermöglichen, auf Systempointer, nämlich die Bildschirmbasis und den -offset zuzugreifen. Wir haben zumindest ein Programm schon angewandt, SCR SET BASE. Mit Hilfe dieser Routine haben wir beim PAINTER die Umschaltung zwischen den beiden Grafikspeichern realisiert. Wir wollen nun einmal auf die Grundlagen dieser Programme eingehen und uns dann auch anschauen, wie wir diese Routinen von BASIC ausnutzen können.

SCR SET BASE und SCR SET OFFSET sind sogenannte Firmwareroutinen, Unterprogramme im Betriebssystem unseres Computers, die auch von

diesem selbst benutzt werden, um bestimmte genau definierte Aktionen, beispielsweise das Setzen eines Bildpunktes auszuführen.

Diese Unterprogramme sind dabei nach Sachgebieten eingeteilt und tragen Namen, die ihren Nutzungszweck kennzeichnen. Dementsprechend besteht dann jede Routine aus dem Namen des Blocks von Routinen zu dem sie gehört (SCR=SCREEN, Bildschirmpaket) und einer weiteren Bezeichnung, die ihrer Funktion entspricht (hier: Basis setzen, Offset setzen).

Alle Firmwareroutinen sind dabei über Sprungzeiger erreichbar. In der Haupt-Firmware-Sprungtabelle ab Adresse hex BB00 sind ihre Ansprungpunkte aufgeführt. Wenn man beispielsweise mit CALL die dort angegebene Adresse anspringt, so wird die zugehörige Aktion ausgeführt. Beim Ansprung von &BC08 wird also die Routine SCR SET BASE benutzt.

Der Akkumulator (Register A) muß dabei das signifikante (höherwertige) Byte der Basisadresse enthalten. Dieses wird dann noch mit einer Maske AND-verknüpft - die Wirkung dieser Verknüpfungstechnik kennen Sie ja bereits aus Kapitel 2.4 - um sicherzustellen, daß nur ein glatter 16K-Bereich benutzt wird.

Da wir nur zwischen 16K-Speicherbereichen umschalten können, müssen wir uns als nächstes überlegen, welche Bereiche wir vom Systemaufbau her überhaupt nutzen können. Wir haben schon gesagt, daß für die Abspeicherung der Grafik nur die Adreßbereiche von 4000 bis 7FFF beziehungsweise von C000 bis FFFF zur Verfügung stehen. Dies soll nun erläutert werden.

Wenn wir unseren Speicher einmal etwas näher untersuchen, so finden wir die folgende Grobunterteilung:

In den untersten 16K des Computers befinden sich einige lebenswichtige Routinen, die beispielsweise für die Umschaltung zwischen Speicherbereichen benutzt werden, und der Anfang unseres BASIC Quelltextes. Dieser Speicherbereich scheidet daher als Grafikspeicher aus. Eine Änderung, speziell in den untersten Bytes, führt normalerweise dazu, daß sich das System "aufhängt".

16k höher, also von hex 4000 bis hex 7FFF treffen wir auf einen 16K-Block der variabel genutzt werden kann. Bei sehr langen Programmen wird er teilweise noch vom BASIC benutzt. Grundsätzlich ist dieser Bereich aber frei.

Wenn wir also die Speichergrenze nach hex 4000 herabschieben und in diesem Block Änderungen vornehmen, so hat dies keinen direkten Effekt

auf die Funktionsfähigkeit des CPC. Allerdings vermindert sich der für Variable und BASIC-Quelltext zur Verfügung stehende Speicherplatz erheblich.

Die nächsten 16K stehen wieder unter Zugriffsverbot. Den Grund dafür kennen Sie bereits. Zwischen den Adressen hex 8000 und BFFF liegen nämlich beispielsweise die Ansprungpunkte unserer Firmwareroutinen. Änderungen dürfen auch hier nicht erfolgen.

Was nun noch übrig bleibt, ist der "normale" Grafikspeicher, den der Schneider beim Einschalten initialisiert, also die Adressen von &C000 bis &FFFF.

Wir haben somit bei normalem Speicheraufbau zwei Blöcke von Adressen zur Verfügung in denen wir unsere Grafik aufbewahren können, der Haupt-Grafikspeicher ab Adresse &C000 und seinen Widerpart ab &4000. Damit sind nun die Grundlagen geschaffen, um die drei in Kapitel 4 benutzten Maschinenroutinen zu erklären.

Die beiden Programme zum Umschalten des Speichers können wir dabei zusammenfassen. In jedem Fall muß hier nur der Akkumulator mit &40 beziehungsweise &C0 geladen werden. Danach wird SCR SET BASE angesprungen. Abschließend erfolgt dann die Rückkehr ins BASIC.

Ähnlich sieht es für die Maschineninitialisierungsroutine aus. Hier muß das Registerpaar HL den geforderten Abstand enthalten. Wenn wir also HL auf 0000 setzen, so wird der Bildschirmoffset zurückgesetzt. Mit Hilfe dieser Routine können Sie sich die Wirkung des Offsets aber auch sehr eindrucksvoll vor Augen führen, indem Sie statt der 0000 einen Verschiebevektor einsetzen. Dazu ein kleines Hilfsprogramm.

```
10 MEMORY 39999
20 DATA 21,00,00,CD,05,BC,C9,X
30 FOR I=40000 TO 410000:READ A$:IF A$<>"X" THEN POKE I,VAL("&"+A$):NEXT I
40 INPUT"Welche Verschiebung";a
50 c=INT(a/256):b=a-c*256:POKE 40001,b:POKE 40002,c
60 CALL 40000
70 GOTO 40
```

**Programmbeschreibung:**

Am Programmanfang setzen wir die Speicherobergrenze auf 39999 herab und schaffen damit einen sicheren, reservierten Speicherplatz für die Ablage unseres Maschinenprogramms. In Zeile 40 wird dann abgefragt, wie groß die Verschiebung sein soll und dieser Wert wird dann in 40001 beziehungsweise 40002 gePOKEt. Es folgt der Ansprung des Maschinen-

programms, das den Offset entsprechend der Vorgabe setzt. Danach geht es zurück in die Abfrage.

Bei der Ausführung des Programms wird entsprechend ihrer Eingabe der Bildschirm verrückt, das heißt Bildschirmteile werden gegeneinander verschoben. Wenn Sie dieses Programm direkt nach dem Einschalten oder nach

```
<CTRL><SHIFT><ESC>
```

eingeben, so sollte die Eingabe 0, den Schirm unbeeinflußt lassen. Geben Sie dagegen 80 ein, so wandert der Schirm um eine Zeile nach oben und dies auch wenn Sie diese Eingaben in der Bildschirmmitte ausgeführt haben. Bedingung dafür ist jedoch, daß Sie bis zum ersten Lauf der Routine kein Scrollen auf dem Schirm ausgeführt hatten. Ansonsten ergibt sich auch bei der Eingabe von 0 eine Verschiebung.

Hier noch eine kleine Randbemerkung: Wie schon gesagt benutzt der CPC die Verschiebung des Offset-Pointers, um relativ schnell ein Scrollen des Bildschirms ausführen zu können. Jeder MODE-Wechsel und jedes CLS setzt jedoch unter anderem auch den Offset-Pointer zurück. Danach ist der CPC also wieder im Ausgangszustand. Statt `<CTRL><SHIFT><ESC>` können wir also auch mit

```
15 MODE 1
```

kontrollierte Bedingungen schaffen. Danach sollten Sie nun einmal den Wert 80 eingeben. Der Schirm wandert dann um eine Zeile nach oben und die oberste Textzeile befindet sich jetzt am unteren Bildschirmrand. Wenn Sie nun beispielsweise den Offset auf 320 erhöhen, so springt der Schirm nach oben.

Diese Änderungen haben allerdings keinerlei Auswirkungen auf die aktuelle Bildschirmausgabe, die ja durch die virtuelle Cursorposition angegeben wird. Trotz aller Schieberei rechnet der CPC bei Verwendung dieser Routine dennoch richtig weiter. Nach der Verschiebung kann es also vorkommen, daß Sie oberhalb einer Zeile schreiben, die Sie gerade schon einmal benutzt haben und dies, obwohl der Cursor nach unten gewandert ist. Der Offset macht's möglich.

Wir haben nun zwar einige nützliche Maschinenprogramme aus dem Inneren unseres Computers kennengelernt. Diese können wir aber bis jetzt noch nicht ausreichend komfortabel bedienen. Insbesondere fehlt uns die Möglichkeit, Daten an Maschinenprogramme weiterleiten zu können und neue BASIC-Befehle zu definieren. Auf diese Problematik wollen wir im folgenden Kapitel näher eingehen.

# 7 Grafikkommandos - selbst definiert

Der Schneider verfügt über zwei Möglichkeiten, um ohne Änderungen in der Systemhardware Maschinenprogramme von BASIC aus anspringen zu können: den Aufruf eines Systemprogrammes mit CALL und die Einbindung eines Maschinenprogrammes in das Betriebssystem mit Hilfe einer sogenannten RSX. Dieses Kürzel steht für *Resident System Extension*, was übersetzt ungefähr als *Bleibende Systemerweiterung* interpretiert werden kann.

In beiden Fällen ist dabei auch die Übergabe von Daten möglich. Schauen wir uns zunächst die Wirkung der beiden Varianten im Vergleich an.

**CALL**: Der CALL-Befehl hat das Format

```
CALL <Ansprungadresse>,Wert 1,Wert 2,.., Wert n
```

wobei die Wertangaben optional sind. Sie können also entfallen. Trifft der BASIC-Interpreter unseres CPC auf dieses Kommando, so ruft er die Maschinenroutine, die ab der angegebenen Adresse liegt, auf. Die übergebenen Daten sind dabei auf einem Zwischenspeicher, dem Stack, abgelegt. Die Register der CPU enthalten die zur Datenabfrage und Interpretation notwendigen Informationen. Wir kommen auf die genaue Art und Weise der Datenübergabe gleich noch zu sprechen.

**RSX**: Die RSX stellt eine Erweiterung des Betriebssystems dar. Ihr Aufbau ist um einiges komplizierter als bei der Verwendung eines einfachen CALL. Allerdings erhält man als Lohn für seine Anstrengungen auch einen erheblich verbesserten Bedienkomfort.

Im Prinzip handelt es sich bei der RSX um ein Maschinenprogramm, genauer eine Serie von Maschinenroutinen, über die ein Programmkopf in einem bestimmten Format so gestülpt wurde, daß die Programme dann durch den Interpreter aufgerufen und ausgeführt werden können. Dementsprechend erfolgt der Aufruf eines Maschinenprogrammes innerhalb einer RSX dann auch ähnlich wie die Ansprache eines BASIC-Kommandos. Das Format lautet dabei

```
|<Name RSX-Routine>,Wert 1,Wert 2, ... ,Wert n
```

und ähnelt damit dem CALL-Aufruf sehr stark. Auch die Datenübergabe erfolgt auf dieselbe Art und Weise. Bevor wir daher näher auf die Unterschiede zwischen RSX und CALL-Kommando eingehen, beschäftigen wir uns zunächst einmal mit der eigentlichen Datenübergabe, das heißt dem Zustand der CPU-Register und des Stack beim Aufruf dieser Programme.

## 7.1 Die Prozessorregister

Die Register A und B sowie das Indexregister IX stehen in direktem Zusammenhang mit der Anzahl der übermittelten Daten. In A befindet sich ihre Anzahl. Es können maximal 32 Werte an eine Maschinenroutine oder RSX übergeben werden. B gibt die Differenz zum Maximalwert 32 oder 20 hex an.

Die einzelnen übermittelten Daten sind dabei auf einem Stapelspeicher, dem Stack, abgelegt. IX gibt dabei die Adresse des untersten Elements dieses Speichers an. Jeder Wert, den wir den Befehlen mitgeben, wird in eine 16-Bit-Binärzahl umgewandelt. Diese wird dann byteweise (zuerst die 8 niederwertigen Bits und dann die 8 höherwertigen) auf dem Stapel abgelegt. Der erste übergebene Wert liegt damit zuoberst. Mit zunehmenden Werten dehnt sich der Stapel dann immer weiter nach unten aus, womit auch IX kleinere Werte enthält. Wir wollen uns das Zusammenwirken der drei Register nun einmal anhand eines Beispiels anschauen.

**Beispiel:**

Wir wollen einmal ein ganz normales CALL-Kommando betrachten.

```
CALL 40000,250,1000
```

Wie sehen nun unsere Registerinhalte aus? Für A und B ist diese Frage relativ einfach zu beantworten. Es wurden 2 Parameter übergeben. A enthält folglich eine 2 und B eine 30 (32-2). Der Stapel sieht wie folgt aus:

```
00
FA
03
E8
```

IX deutet nun auf die letzte Adresse dieses Speichers, also den Wert E8. Die hier angegebenen Hexadezimalwerte erhalten Sie übrigens, indem Sie sich mit der Funktion HEX$ das hexadezimale Äquivalent der nach dem CALL-Kommando stehenden Zahlen ausgeben lassen.

```
PRINT HEX$(250,4)
```

liefert uns als Ergebnis FA. Führen wir denselben Befehl mit dem Wert 1000 durch, so erhalten wir als Ergebnis die Ausgabe 03E8. Diese schreiben wir nun einfach von oben nach unten in unseren Stapel und erhalten damit den oben beschriebenen Stapelaufbau.

Als nächstes müssen wir uns nun damit beschäftigen, über welche Möglichkeiten wir verfügen, um die im Stack stehenden Daten nun in Register unseres Prozessors (der CPU) einzuladen, damit wir mit diesen dann in unseren Maschinenprogrammen arbeiten können.

Dies ist jedoch relativ einfach möglich. Der Z80-Prozessor verfügt nämlich über Befehle, die das indizierte Laden eines Registers der CPU aus der durch das Indexregister IX adressierten Speicherstelle ermöglichen. Wenn wir also unsere 1000 in den Registern H und L abspeichern wollen, so genügen zwei einfache Kommandos.

```
LD H,(IX+1)
LD L,(IX+0)
```

Damit befinden sich die beiden Bytes dieser Zahl als 16-Bit-Wert gespeichert in dem Registerpaar HL. Dasselbe Verfahren können wir natürlich nun auch verwenden, um die 250 beispielsweise in das Registerpaar DE einzuladen. Nach

```
LD D,(IX+3)
LD E,(IX+2)
```

ist unsere Datenübertragung perfekt, und die auszuführende Maschinenroutine, die auf diese Daten zurückgreifen soll, kann nun angesprungen werden. An deren Ende muß dann der Maschinensprachebefehl RET (&C9) stehen, der den CPC anweist, ins BASIC zurückzukehren.

Damit verfügen wir nun über alle Informationen, die wir benötigen, um an eine Maschinenroutine Daten mit CALL zu übergeben und diese dann auszuführen. Wir können damit nun beispielsweise ein Maschinenprogramm zum Umschalten des Grafikspeichers schreiben, welches es uns ermöglicht, den zu benutzenden Bereich gleich mit anzugeben. Der Aufruf soll dabei

```
CALL 40000,<Inhalt Register A>
```

lauten, wobei der einzige Parameter durch das signifikante, höherwertige Byte der Bildschirmbasisadresse zu ersetzen ist. Wenn wir also auf den unteren Grafikspeicher umschalten wollen, so müßte hier der Wert &40 eingegeben werden, ansonsten &C0. Die nachstehenden niederwertigen 8 Bits werden durch SCR SET BASE ja nicht berücksichtigt.

Ein Maschinenprogramm, das auf den Dateninput reagiert, ist dann problemlos zu realisieren. Es benötigt nur 3 Befehle:

```
LD A,(IX+0)              DD 7E 00
CALL SCR SET BASE        CD 08 BC
RET                      C9
```

Die Hexcodes DD 7E stehen dabei für das Laden des Akkumulators aus der durch IX adressierten Speicherstelle. Der dritte Hexwert gibt dann die Differenz an (in unserem Fall keine = 0). Wenn Sie diese 7 Hexcodes nun nach

```
MEMORY 39999
```

mit

```
POKE 40000,&DD
POKE 40001,&7E
POKE 40002,&00
POKE 40003,&CD
POKE 40004,&08
POKE 40005,&BC
POKE 40006,&C9
```

in den Speicher ab der Adresse 40000 einladen, so können Sie dann mit `CALL 40000,&40` auf den unteren Speicher schalten und mit `CALL 40000,&C0` wieder in die normale Benutzung überwechseln.

Wir können unseren Aufruf jedoch noch komfortabler gestalten, indem wir die Umschaltung zwischen den Bildschirmen als BASIC-Kommandos ausführen. Dies wollen wir nun besprechen.

## 7.2 RSX-Erweiterungen

Beginnen wir mit einigen grundsätzlichen Bemerkungen zur Ablage der RSX. Eine resistente Systemerweiterung ist immer im RAM, genauer in

den zentralen 32K-RAM abgespeichert. Dies ist der erste Unterschied zum CALL-Befehl, der den gesamten RAM-Speicher unseres Computers adressieren kann.

Während wir also mit CALL jede Adresse zwischen dezimal 0 und 65535 als Beginn eines Maschinenprogrammes aufsuchen können, sind wir bei der Verwendung des RSX-Befehls auf die Adressen von hex 4000 bis hex BFFF beschränkt. Eine RSX legt man daher meist an der Speicherobergrenze ab.

Woraus besteht nun eine RSX? Um diese Frage zu beantworten, gehen wir am besten von der Art des Aufrufes aus. Wie wir schon gesagt haben, schafft eine Systemerweiterung mit RSX neue Befehle (BASIC-Erweiterungs-Befehle!). Die Ansprache einer RSX-Routine geschieht daher auch mit einem BASIC-ähnlichen Befehl.

Unsere Grafikspeicher-Verschiebeprogramme könnten wir zum Beispiel mit den Kürzeln OGRAPH (für das Umschalten auf die obere Grafik) und UGRAPH (für die Bewegung in der Umkehrrichtung) bezeichnen. Wenn wir dann in einem BASIC-Programm diese Befehle mit dem immer noch voranzustellenden Erweiterungsstrich ("|") eingeben, so müßte der CPC automatisch auf den gewünschten Speicherbereich umschalten.

Das setzt natürlich zuerst einmal voraus, daß der Computer überhaupt von der Existenz der zugehörigen Ausführungsroutine Kenntnis hat und weiß, daß diese irgend etwas mit den beiden Kommandos zu tun hat. Wir müssen das Betriebssystem also erst einmal von der Existenz dieser Routine in Kenntnis setzen. Die RSX muß initialisiert werden.

Ist dann eine RSX-Funktion einmal in das Betriebssystem eingeklinkt worden, so kann man sie immer wieder im Verlauf des Programms ohne weitere Informationen an das Betriebssystem aufrufen. Alles andere funktioniert dann genauso, wie bei einem BASIC-Befehl gewohnt.

Die Initialisierungsprozedur erfolgt dabei mit Hilfe einiger Routinen aus dem Betriebssystemkern, den Kernal-Routinen. Wir wollen uns die Einbindung der RSX in das Betriebssystem einmal anhand unserer beiden Befehle anschauen.

Eine RSX besteht aus 2 großen Teilen:

**Einem Sprungzeigerteil**: Dieser enthält die Ansprungpunkte aller Maschinenroutinen, die zur RSX gehören sollen, und eine Namenstabelle, die angibt, bei welchen Namen welches Maschinenprogramm aufgerufen werden soll und

**den Maschinenprogrammen**: Diese werden durch die RSX ausgeführt. Zu diesem Teil ist kaum etwas zu sagen. Wir können die auszuführenden Maschinenprogramme in beliebiger Reihenfolge (gegebenenfalls auch ineinander verschachtelt) an einer beliebigen Stelle im RAM postieren.

Etwas anders sieht es dagegen mit unserer Sprungtabelle aus. Diese darf, wie schon gesagt, nur in den zentralen 32K des Speichers liegen, also nicht unterhalb eines ROMS. Normalerweise wird man sie, wie schon von der CALL-Routine gewohnt, an der Speicherobergrenze - also zum Beispiel im Bereich von hex A000 oder auch dezimal 40000 nach oben - ablegen.

Wie wir schon gesehen haben, besteht die Sprungzeigertabelle aus 2 Teilen: dem eigentlichen Sprungzeigerblock und den Definitionen/Namen der neuen Befehle. Für den Aufbau dieser beiden Blöcke gelten eine Reihe von Konventionen, die unbedingt eingehalten werden müssen, damit der Betriebssystemkern, das Kernal, die Maschinenroutinen in den BASIC-Interpreter einbinden und später dann auch ausführen kann. Sie finden den Aufbau eines RSX-Sprungblocks in Listing 20 am Beispiel unserer beiden Befehle.

**Listing 20:** *Assemblerlisting RSX Grafikswitch*

```
A000  LD HL,0000                      21 00 00
A003  LD (A010),HL                    22 10 A0
A006  LD BC,<RSX-Anfang=A020>         01 20 A0
A009  LD HL,<Kernal-Speicher=A035>    21 35 A0
A00C  JP KL LOG EXT                   C3 D1 BC
.
A010  Zwischenspeicher OFFSET LOW
A011  Zwischenspeicher OFFSET HIGH
.
A020  Anfang Namenstabelle            28 A0
A022  JP UGRAPH                       C3 40 A0
A025  JP OGRAPH                       C3 60 A0
A028  U                               55
A029  G                               47
A02A  R                               52
A02B  A                               41
A02C  P                               50
A02D  H + 80 HEX                      C8
A02E  O                               4F
A02F  G                               47
A030  R                               52
A031  A                               41
A032  P                               50
A033  H + 80 HEX                      C8
A034    Trennungsnull                 00
```

```
A035  Kernal-Speicher 3 Bytes frei
.
A040  'PRG:UGRAPH
A040  CALL SCR GET LOCATION          CD 0B BC
A043  EX DE,HL                       EB
A044  LD A,&40                       3E 40
A046  CALL SCR SET BASE              CD 08 BC
A049  LD HL,(A010)                   2A 10 A0
A04C  CALL SCR SET OFFSET            CD 05 BC
A04F  LD (A010),DE                   ED 53 10 A0
A053  RET                            C9

A060  'PRG:OGRAPH
A060  CALL SCR GET LOCATION          CD 0B BC
A063  EX DE,HL                       EB
A064  LD A,&C0                       3E C0
A066  CALL  SCR SET BASE              CD 08 BC
A069  LD HL,(A010)                   2A 10 A0
A06C  CALL SCR SET OFFSET            CD 05 BC
A06F  LD (A010),DE                   ED 53 10 A0
A073  RET                            C9
```

Den Anfang dieses Maschinensprachelistings bildet die Initialisierungsroutine für die RSX. Wir kommen auf diesen Programmteil gleich noch zurück. Ab Adresse hex A020 finden Sie dann die eigentliche RSX-Sprungtabelle.

In den ersten 2 Bytes unseres Sprungblocks steht eine Adresse, die Verzeigerung auf den Anfang der Namenstabelle. Danach folgen als Jump-Kommandos (JP) die Ansprungpunkte der verschiedenen Routinen. Diese sind im 3-Byte-Format hintereinander abgelegt. Der Aufruf erfolgt dabei durch unbedingten direkten Sprung (JP).

Ab der Adresse der Namenstabelle liegen dann die einzelnen Befehlsnamen. Ein Befehlsname kann aus einem oder mehreren Buchstaben und/oder Punkten bestehen. Leerschritte innerhalb des Namens sind nicht zugelassen, da der Computer den Leerschritt als Trennmarkierung zwischen einzelnen BASIC-Kommandos im Interpreter beziehungsweise im Editor benutzt.

Würden wir also einen Befehl mit Leerschritt definieren, so würde der BASIC-Interpreter ihn als 2 neue BASIC-Kommandos interpretieren. Wenn wir nun einmal davon ausgehen, daß die Einzelwörter keinen Sinn ergeben, also als einzelne Kommandos nicht definiert sind, gibt es einen "Syntax Error".

Die ASCII-Codes der Namen sind nun hintereinander, beginnend mit dem ersten Namen abzulegen. Dabei ist allerdings noch ein zusätzlicher Punkt

zu beachten. Da Befehlsnamen und auch Variablennamen beim CPC keine vorgegebene Länge haben, muß dem Computer irgendwie mitgeteilt werden, daß der Name beziehungsweise die Namensdefinition beendet ist.

Dies geschieht, indem man zum letzten Zeichen des Schlüsselwortes 128 oder 80 hex addiert. Dadurch wird das höchste Bit (BIT 7) gesetzt, was der CPC als Endemarkierung wertet. Aus diesem Grund ist es natürlich auch klar, daß Grafikzeichen (mit Werten über 128) in Befehlswörtern nichts zu suchen haben. Der CPC würde diese in ein um 128 im ASCII-Code niedriger liegendes Zeichen übersetzen und das Befehlswort als beendet betrachten. Da er dann mit den nachfolgenden Zeichen nichts mehr anfangen könnte, ergäbe sich wiederum eine Fehlermeldung.

Den Abschluß der Namenstabelle bildet ein Byte mit dem Wert 00. Nach dem Ende der Tabelle sind dann noch 3 Bytes für den Betriebssystemkern (das Kernal) zu reservieren. Hiermit stellt der Computer die Verbindung zum BASIC-Interpreter her.

Wenn Sie diese Konventionen beachten, ist die Initialisierung der RSX kein Problem mehr. Der Anfang unserer RSX-Befehlstabelle ist in das Registerpaar BC zu laden. Die Adresse des ersten für das Kernal reservierten Bytes gehört in HL. Wenn Sie dann noch die Betriebssystemroutine KL LOG EXT aufrufen, ist Ihre Arbeit beendet. Die neuen Routinen sind dann als Erweiterungskommandos integriert. Den Rest leistet der CPC selber.

Die dazu notwendigen Befehle finden Sie am Anfang unseres Assemblerlistings. Der erste Befehl gehört dabei nicht zur Initialisierungsroutine. Er löscht vielmehr die Speicherstellen A010 und A011. In diesen Adressen werden unsere beiden Maschinenprogramme den letzten OFFSET zwischenspeichern. Um eine definierte Anfangsposition zu erhalten, müssen diese beiden Adressen am Anfang auf 0 gesetzt werden.

Damit wären wir nun auch bei der Erklärung der eigentlichen Maschinenprogramme angelangt. Diese sind etwas komplizierter aufgebaut als die einfachen Versionen, die mit CALL arbeiteten, weisen dafür aber wiederum einen höheren Bedienkomfort auf.

Ab Adresse A040 finden Sie die Entsprechung für das Kommando UGRAPH. Zunächst wird eine Betriebssystemroutine angesprungen, SCR GET LOCATION. Diese stellt eine kombinierte Umkehrroutine zu SCR SET BASE und SCR SET OFFSET dar. Sie liest die aktuellen Werte der beiden Pointer und gibt diese in Register A (signifikantes Byte der Basis) beziehungsweise in das Registerpaar HL (OFFSET) zurück.

Als nächsten Schritt sichern wir nun HL in DE und laden dann den Akkumulator mit &40. Es folgt der Ansprung von SCR SET BASE, die Umschaltung auf den unteren Grafikspeicher. Damit wäre nun in der einfachsten Variante unsere Arbeit beendet und wir könnten ins BASIC zurückkehren.

Ein komfortabler Verschiebebefehl muß es jedoch auch ermöglichen, mit verschiedenen Werten für den OFFSET zu operieren. Wenn beispielsweise auf dem Hauptbildschirm ein Scroll ausgeführt wurde, so würde ansonsten auch der Inhalt unseres Nebenspeichers neu interpretiert. Es fände also auch hier eine Verschiebung statt. Dies darf jedoch nicht passieren, wenn man permanent zwischen zwei Schirmen umschalten will, um dort die verschiedenen Ausgaben zu positionieren (wobei ja nicht jedesmal ein Löschen erfolgen darf).

Daher setzen UGRAPH und OGRAPH auch gleichzeitig noch den OFFSET neu - und zwar auf den alten Wert, der in dem nun aktuell ausgewählten Grafikspeicher vor der letzten Umschaltung Geltung hatte.

Dazu wird der letzte OFFSET, der in A010 und A011 zwischengespeichert war in HL eingeladen. Als nächstes geht es in SCR SET OFFSET, die den OFFSET-Pointer auf den in HL gespeicherten Wert setzt, womit nun auch der OFFSET-Pointer korrigiert ist.

Nach der Umschaltung auf den alten OFFSET-Wert muß natürlich nun noch der neue, gerade überschriebene Wert gesichert werden, denn bei einer Rückschaltung müssen wir diesen ja nun wieder einladen. Am Anfang unserer Routine hatten wir ihn in DE gesichert. Wir laden nun unseren Zwischenspeicher aus diesem Registerpaar und können dann schließlich mit RET wieder ins BASIC zurückkehren.

Die Interpretation des Befehls UGRAPH ist damit abgeschlossen. Um das Kommando OGRAPH ausführen zu können, müssen wir in unserer Befehlskette nun nur eine einzige Angabe ändern, nämlich die Angabe der Bildschirmbasis. A muß dazu mit C0 geladen werden. Das restliche Programm bleibt jedoch gleich.

Unsere Befehlserweiterung ist damit nun komplett. Wir müssen sie nur noch im Speicher unterbringen. Dazu dient ein kleines Ladeprogramm, das wir von seinem Grundaufbau her ja schon kennen. Nachdem Sie das Programm eingetippt haben, sollten Sie es vor dem ersten RUN-Versuch unbedingt erst einmal sichern. Bereits ein kleiner Fehler in einem DATA kann nämlich dazu führen, daß sich der Computer abstürtz.

**Listing 21:** *Programm GRAPHIKSWITCH RSX*

```
10 '************************
20 '** graphikswitch rsx **
30 '**      by C.S.       **
40 '************************
50 MEMORY &4000-1
60 DATA 21,00,00,22,10,a0,01,20,a0,21,35,a0,c3,d1,bc,x
70 DATA 28,a0,c3,40,a0,c3,60,a0,55,47,52,41,50,c8,4f,47,52,41,50,
c8,00,00,00,00,x
80 DATA cd,0b,bc,eb,3e,40,cd,08,bc,2a,10,a0,cd,05,bc,ed,53,10,a0,
c9,x
90 DATA cd,0b,bc,eb,3e,c0,cd,08,bc,2a,10,a0,cd,05,bc,ed,53,10,a0,
c9,x
100 FOR k=0 TO 3
110 FOR i=k*&20+&A000 TO k*&20+&A020
120 READ a$:IF a$="x" THEN 140 ELSE POKE i,VAL("&"+a$)
130 NEXT i
140 NEXT k
150 FOR i=&A000 TO &A100:PRINT HEX$(i,2),HEX$(PEEK(i),2):NEXT
```

Sie sollten nun ruhig ein wenig mit der RSX-Erweiterung spielen und sich überlegen, wie Sie eigene Maschinenprogramme auf das RSX-Format bringen können. Als Ausgangsbasis für Ihre Überlegungen kann Ihnen dabei Listing 20 dienen. Ein weiteres Anwendungsbeispiel für eine - jetzt allerdings schon komplexe - RSX ist der Spritegenerator, den wir nun entwickeln wollen.

# 8 Ein Spritegenerator für den CPC

Nach der Vorarbeit in den letzten beiden Kapiteln sind wir nun in der Lage, uns unserer Abschlußaufgabe zu widmen, der Programmierung eines Spritegenerators, beziehungsweise Spriteeditors. Zunächst ein paar definitorische Aussagen.

**Sprites** sind Ihnen wahrscheinlich aus dem Bereich der Action-Spiele bekannt. Es handelt sich dabei um jene schnell bewegten Raumschiffe, Geister und ähnliche Kreaturen, die Welträume, Adventure-Szenerien usw. bevölkern. Von Sprites spricht man insbesondere, wenn diese durch die Computerhardware erzeugt werden können.

Man muß dazu eine bestimmte Kombination von Bits oder Bytes in einige ausgewählte Adressen schreiben und der Videogenerator, ein Baustein, der für den Bildaufbau zuständig ist, erzeugt daraus dann **selbständig** den Sprite und stellt diesen an einer anzugebenden Stelle auf dem Schirm dar.

Es existiert also ein Grafikspeicher. Aus diesem wird jedoch nur ein Teil aller Bildschirmpunkte ausgelesen. Einige Teile des Bildschirms werden dagegen mit den im Spritespeicher gespeicherten Daten belegt. Der Sprite blendet also den Grafikspeicher, der den Hintergrund liefert, teilweise aus.

Wenn der Anwender dagegen Sprites künstlich erzeugen muß, indem er das Setzen des Sprites durch Zugriff auf einzelne Bildpunkte des Grafikspeichers imitiert, so spricht man von **Shapes**. Das Setzen des Sprites erfolgt hier also nicht mehr durch die Computerhardware, sondern wird via Software vorgenommen.

Der Schneider selbst verfügt über keine eingebauten Sprites, so daß wir diese als Shapes imitieren müssen. Wir werden die Begriffe Sprite und Shape daher synonym verwenden.

Einen kleinen Ausflug in den Bereich der Sprites haben wir schon zu Beginn von Kapitel 2.4 vorgenommen, als wir mehrfarbige Blockgrafikzeichen erzeugt haben. Wenn wir nun Wert auf größere farbliche Möglichkeiten und eine höhere Arbeitsgeschwindigkeit legen, so müssen wir auf Maschinenspracheprogramme zurückgreifen.

Wollen wir dabei mit Shapes arbeiten, so benötigen wir zwei verschiedene Programme. Zunächst einmal muß ein Spritedarstellungsprogramm vorhanden sein. Dieses muß es ermöglichen, einen Sprite an einer beliebigen Stelle auf dem Bildschirm darzustellen oder zu löschen. Dabei sollte der Hintergrund unbeeinflußt bleiben.

Daneben brauchen wir natürlich auch noch eine Art Mal-Utility ähnlich unserem Painter, die uns beim Entwurf unseres Sprites unterstützen soll, einen sogenannten Spriteeditor. Die Umrechnung auch schon einiger weniger Bildpunkte wird schnell zur Sisyphos-Arbeit, speziell, wenn man dabei auch noch auf die Farbspeicherformate des CPC Rücksicht nehmen muß. Hier ist ein Programm gefordert, das uns die Kleinarbeit der Spritedefinition abnimmt.

Für das Verhältnis von Spriteeditor zu Spritedarstellungsroutine greifen wir dabei auf das schon beim PAINTER bewährte Konzept zurück. Das Entwicklungsprogramm, in unserem Fall also der Editor, muß durch eine spezielle Unterroutine in der Lage sein, das Darstellungsprogramm inclusive Spritedaten als ASCII-File wegzuschreiben. Die so gespeicherte Datei können wir dann ja problemlos als Programm wieder einlesen.

Auch für den Aufbau unseres Editorprogramms greifen wir auf Bewährtes zurück. Wir verlangen wiederum eine cursororientierte Setz- beziehungsweise Löschfunktion, mit der wir unseren Shape ändern wollen, wobei wir allerdings mit einer Lupenfunktion sehr komfortabel arbeiten wollen.

Während wir beim PAINTER gezwungen waren, Bildpunkte in ihrer Originalgröße einzeln zu setzen, da ja der Gesamtbildschirm unser Arbeitsfeld darstellte, so ist dies bei dem nun zu entwickelnden Programm nicht mehr der Fall.

Wir können hier mit vergrößerten Bildpunkten arbeiten, die wir auf Knopfdruck ändern, wobei gleichzeitig unser Sprite, der ja nur einen kleinen Teil unseres Schirms einnimmt, in der Originalgröße parallel dargestellt wird.

Nach diesen Vorüberlegungen sind wir nun wieder in der Lage unsere Zieldefinition anzugeben.

**Zieldefinition:**

Der Spriteeditor soll es uns ermöglichen, auf dem Bildschirm mit einem Cursor umherzufahren, Farben auszuwählen und in einem vergrößerten Raster die Sprites zu setzen. Zur Ergebniskontrolle wollen wir gleichzeitig

an einer anderen Stelle auf dem Bildschirm, hier wird es die linke untere Ecke sein, unseren Shape permanent dargestellt erhalten.

Als Nebenroutinen soll der Spriteeditor Unterprogramme enthalten, die es uns ermöglichen, die Eckkoordinaten eines Sprites festzulegen, wie die Ausdehnung in X- und Y-Richtung sowie die zu verwendenden Farben. Des weiteren muß es natürlich möglich sein, den Shape zu sichern. Die Abspeicherung soll hierbei inclusive Darstellungsroutine erfolgen.

Bevor wir an die eigentliche Programmentwicklung gehen können, muß nun noch ein besonderer Unterpunkt, die Aufteilung des Speichers und damit zusammenhängend die Ablage der Spritedaten, besprochen werden.

## 8.1 Die Ablage des Sprites

Für die Abspeicherung aller Systemroutinen, die wir benötigen und auch der Shapedaten ist der Speicherbereich ab dezimal 40000 nach oben reserviert. Die Aufteilung dieses Bereiches gibt Ihnen Bild 8.1 wieder. Diese Unterteilung gilt dabei sowohl für den Editor als auch für das Darstellungsprogramm.

Wir können vier Teile in unserem Speicher gut unterscheiden. Der unterste Bereich ist uns aus dem letzten Kapitel schon recht geläufig. Es handelt sich um die Grundstruktur, die wir benötigen, um eine RSX zu etablieren.

```
9FE0  LD BC,9FF0                01 F0 9F
9FE3  LD HL,9FFD                21 FD 9F
9FE6  JP KL LOG EXT             C3 D1 BC
9FE9  RET                       C9
9FF0  <Adresse Namen>           F5 9F
9FF2  1.Routine                 C3 02 A0
9FF5  1.Name  S                 53
9FF6          P                 50
9FF7          R                 52
9FF8          I                 49
9FF9          T                 54
9FFA          E + 80 HEX        C5
9FFB  Trennungsnull             00 00
9FFD  KERNAL-Speicher           00 00 00
-----------------------------------------------------------------
A000  Y-Koordinate
A001  X-Koordinate
A002  Darstellungsroutine SPRITESET
-----------------------------------------------------------------
A080  SAVER (Konvertierungsprogramm)
-----------------------------------------------------------------
```

```
A100  Converter
-----------------------------------------------------------------
A200  SPRITE-Ausdehnung X
A201  SPRITE-Ausdehnung Y
A202  SPRITE-Daten
  .
  .
  .
A202+X*Y-1
-----------------------------------------------------------------
```

**Bild 8.1:** *Speicheraufbau SPRITEEDITOR*

Mit dem erstmaligen Durchlauf unseres Darstellungsprogramms wird eine BASIC-Befehlserweiterung in das Betriebssystem eingeklinkt, der Befehl SPRITE. Dieser benötigt noch zwei Parameter, die Angaben für die X- beziehungsweise Y-Koordinate des darzustellenden Shapes.

Ab Adresse A000 finden Sie nun die eigentliche Spritedarstellungsroutine, SPRITESET. Dieses kurze Maschinenprogramm wird es uns ermöglichen einen Shape an einer beliebigen Stelle auf den Bildschirm zu setzen.

Das Spriteentwicklungsprogramm benötigt noch zwei weitere Routinen. Da wäre zunächst einmal das Konvertierungsprogramm SAVER ab Adresse A080, das die Umformung der Spritekoordinaten für die Abspeicherung vornimmt.

Daneben benötigt der Editor noch den CONVERTER ab Adresse A100. Ab A200 finden wir dann die eigentlichen Farbdaten, das heißt die Farbbildpunkte, die unseren Shape ausmachen. In den ersten beiden Adressen ist dabei die Ausdehnung unseres Sprites in X- beziehungsweise Y-Richtung abgelegt. Danach folgen die einzelnen Bildpunkte. Es ist nun an der Zeit, uns mit dem Verhältnis der einzelnen Maschinenroutinen etwas näher auseinanderzusetzen.

## 8.2 Die Maschinenunterprogramme

Die Ansprache der Bildpunkte ist, wie wir ja schon aus Kapitel 6 wissen, beim CPC eine ziemlich vertrackte Sache. Aus diesem Grund gestaltet sich auch das Setzen der einzelnen Bildpunkte mit der Routine SPRITESET relativ schwierig. Wir müssen nämlich mit einer ganzen Reihe von Vektoren arbeiten, unter anderem auch dem Bildschirmoffset und

natürlich den aus dem Aufbau des Grafikspeichers resultierenden Verschiebungen.

Dies hat zur Folge, daß die mit dem Bildschirm-Handling befaßten Maschinenprogramme eine relativ komplexe Struktur aufweisen. Auf eine detaillierte Erklärung der Einzelroutinen soll deshalb hier verzichtet werden. Wir werden die Routinen als Black-Box-Programme betrachten. Was leisten nun die einzelnen Unterprogramme?

### 8.2.1 Die Routine SPRITESET

Aufgabe dieses Unterprogramms ist es, ein in einzelnen Bytes abgelegtes Farbmuster positionsrichtig auf dem Bildschirm darzustellen. Die Routine arbeitet dabei byteorientiert. Bei der Abspeicherung des Sprites muß dabei bereits auf die Farbabspeicherung beim CPC Rücksicht genommen werden.

SPRITESET geht davon aus, daß die 8 darzustellenden Bits bereits die 8 darzustellenden Farbpunkte enthalten. Es handelt sich bei diesem Programm also um eine Art Oberroutine, die nur noch die Positionierung der einzelnen Bytes im Grafikspeicher durchführt.

Als Inputs benötigt sie dabei vier Angaben, die in den Speicherstellen A000, A001, A200 und A201 stehen. Es sind dies die Positionsangaben des Shapes sowie seine Ausdehnung in X- beziehungsweise Y-Richtung. Mit Hilfe dieser Angaben ist SPRITESET dann in der Lage, aus den Spritedaten die zugehörige Bildpunktkombination in die richtigen Adressen des Grafikspeichers einzuladen.

Für die konkrete Arbeit ist nun folgendes zu beachten. In beiden Fällen, das heißt sowohl bei der Sprite-Darstellungsroutine als auch beim Editor benötigen wir natürlich das Maschinenprogramm SPRITESET. Allerdings wird dieses beim Spriteeditor in modifizierter Version benutzt. Vor der Ausgabe eines jeden Bytes wird nämlich die Unterroutine CONVERTER angesprungen. Bei der Darstellungsvariante unterbleibt dies.

### 8.2.2 Das Unterprogramm CONVERTER

Aufgabe dieses Programmes ist es, ein Byte, welches im Akkumulator übergeben wird, vom Normalformat, das beim Editor für die Datenab-

speicherung benutzt wird, in das vom CPC im Mode 0 benötigte Farbschema zu bringen.

Beide Routinen, der Spriteeditor, wie auch die später zu benutzende Darstellungsroutine benutzen dieselben Adressen für die Ablage des Sprites. In jeder Adresse ist auch dasselbe Byte beziehungsweise genau dieselbe Anzahl von Bildpunkten gespeichert.

Der Unterschied besteht jedoch in der Art der Abspeicherung. Die Darstellungsroutine arbeitet mit Daten, die bereits das vom CPC im Mode 0 geforderte Farbschema einhalten (vergleiche 6.1). Für den Editor ist dieses Verfahren jedoch nicht tragbar. Die Änderung eines Bildpunktes, würde einen gezielten Zugriff auf verschiedene Bits eines Bytes bedeuten, die auch noch in der richtigen Reihenfolge mit den einzelnen Farbbits besetzt werden müßten.

In BASIC würden die dazu notwendigen Umformungsoperationen zu lange dauern, so daß hier auf ein anderes Konzept der Farbabspeicherung zurückgegriffen wurde. Dabei wird ein Byte einfach in der Mitte geteilt und die dabei entstehenden Halbbytes oder auch Nibbles enthalten dann direkt als Binärzahlen den Farbenwert der einzelnen Bildpunkte. CONVERTER führt dann vor der Ausgabe als Sprite durch den Editor die notwendigen Umformungen durch.

### 8.2.3 Unterprogramm SAVER

Die Funktion dieses dritten und kleinsten Maschinenprogrammes ergibt sich aus der Beschreibung des CONVERTERS. Wir haben dort gesagt, daß der Spriteeditor und die später zu benutzende Darstellungsroutine unterschiedliche Datenformate benutzen. Der Editor hat die Farbinformationen in den Nibbles abgelegt, während das Darstellungs-programm mit konvertierten Daten arbeitet.

Vor der Abspeicherung der Spritedaten muß also das Datenfeld des Sprites konvertiert werden. Eine Routine, die dies für ein Byte leistet, kennen wir schon. Wir müssen nämlich nur jedes Byte unseres Datenfeldes mit dem CONVERTER umwandeln und haben dieses dann in der konvertierten Form vorliegen. Dies leistet nun SAVER. Es formt die einzelnen Daten in das CPC-Format um und gibt diese dann an dic Ausgabedatei aus.

## 8.3 Der Koordinatenrahmen

Bevor wir unser Programm anhand eines Flußdiagramms näher analysieren können, sind noch ein paar Worte zum verwendeten Koordinatensystem nötig.

Für die Positionierung des Sprites ist nämlich nicht das normale Koordinatensystem Basis, sondern der computerinterne Koordinaten-rahmen stellt die Berechnungsgrundlage für dieses Programm dar. Der CPC rechnet nämlich, wie wir schon in Kapitel 6.1 gesehen haben mit 200 Zeilen a 80 Bytes. Die Zeile 0 ist dabei im Gegensatz zur normalen Grafikdefinition die oberste Zeile unseres Bildschirms.

Für die Ansprache unseres Shapes stehen uns also bei gegebener Spritegröße von X Punkten in der Horizontalen und Y Bildpunkten in der Vertikalen die Koordinaten

0 bis 80-X/2 für die X-Koordinate des Sprites

0 bis 200-Y für die Y-Koordinate des Sprites

zur Verfügung, wobei X und Y die Ausdehnung des Shapes in der jeweiligen Richtung enthalten. Die restlichen Linien beziehungsweise Spalten werden dann durch unseren Sprite eingenommen. Bezugsbasis und damit der relevante Punkt für unsere Berechnungen ist also wie bei der Zeichendarstellung die obere linke Ecke unseres Sprites.

Nach diesen Vorbemerkungen können wir nun an die eigentliche Programmentwicklung gehen.

## 8.4 Das Flußdiagramm

Den Anfang bildet wieder die Entwicklung eines Flußdiagramms. Sie finden es in Bild 8.2. Am Beginn unseres Programms sehen Sie wieder eine Initialisierungsroutine, die neben den gewohnten Aufgaben nun auch die Maschinenprogramme einlädt.

Es folgt ein Teil zur Farbabfrage, der als Unterprogramm ausgeführt ist und somit von verschiedenen Punkten aus angesprungen werden kann. Als nächsten Unterpunkt kommen wir nun zur Abfrage der Spritegröße. Die mit dem Programm entwickelbaren Shapes können zwischen 2 und 12 Punkten in X-Richtung groß sein, wobei hier aufgrund der byteweisen Abspeicherung nur gerade Werte erlaubt sind. In der Vertikalen haben

wir eine etwas größere Punktezahl zur Verfügung. Hier sind zwischen 1 und 25 Bildpunkte Höhe möglich.

Nun gelangen wir in den Bilddarstellungsteil. Es wird ein Raster von der Größe unseres Sprites gezeichnet, wobei jeder Bildpunkt durch ein Rechteck von der Größe eines Zeichens repräsentiert wird. Ist ein Punkt in der Hintergrundfarbe darzustellen, so handelt es sich um ein Rechteck, bei dem nur die Begrenzungslinien gezeichnet werden. Ansonsten wird das Rechteck als ausgefülltes Zeichen mit der zu setzenden Farbe belegt.

Im linken Teil unseres Bildschirms werden dann noch die einzelnen darzustellenden Farben ausgegeben.

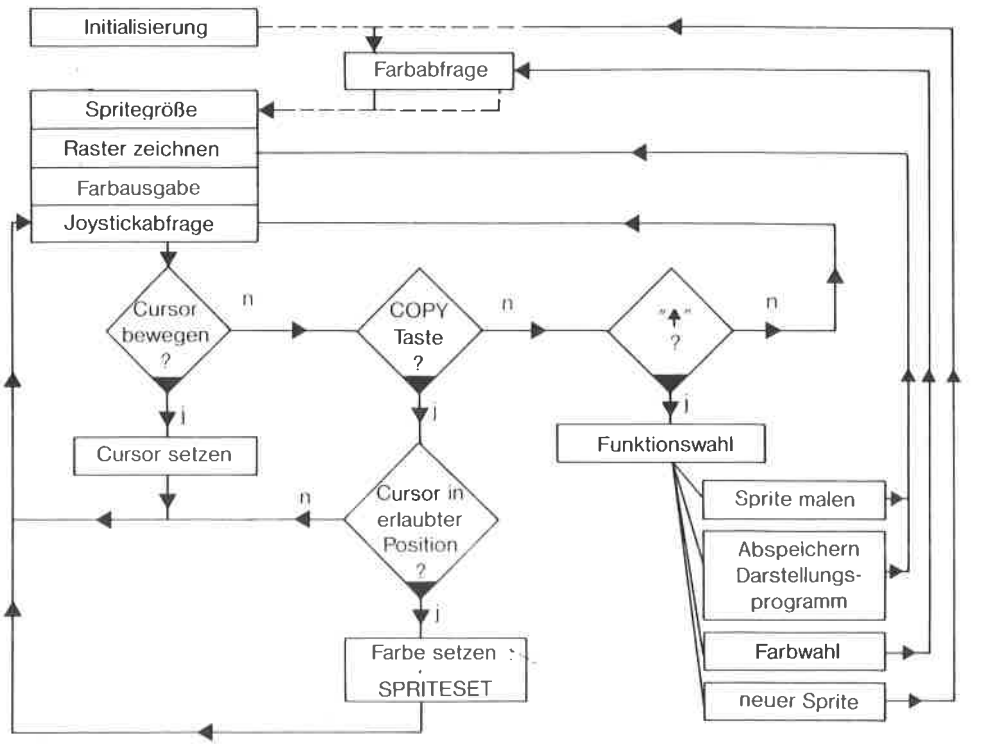



**Bild 8.2:** *Flußdiagramm Programm Spriteeditor*

Als nächstes gelangen wir in die Joystickabfrageroutine. Die Hauptabfrageschleife läuft dabei entlang der drei Entscheidungsrauten, jeweils den "n"-Markierungen folgend, zurück in den Teil Joystickabfrage. Solange nichts passiert, wird in dieser Schleife nur permanent ein blinkender Cursor dargestellt. Das Prinzip ist dabei dasselbe wie beim DESIGNER. Die Betriebssystemroutinen BB8A und BB8D übernehmen das Setzen und Löschen des Cursors.

Aus dieser Schleife gibt es nun drei Ausgänge. Zum ersten kann es sein, daß eine Cursortaste gedrückt, beziehungsweise der Joystick bewegt wurde. In diesem Fall muß zu einer Routine verzweigt werden, die den Cursor neu positioniert. Dies wird mit der ersten Raute überprüft.

Aus der zweiten Raute erfolgt der Aussprung in "j"-Richtung, wenn die COPY-Taste gedrückt wurde. In diesem Fall soll eine neue Farbe auf der Farbleiste ausgewählt werden, beziehungsweise auf dem Schirm ein Punkt mit der neuen Farbe gesetzt werden. In einer zweiten Entscheidung wird dazu zunächst überprüft, wo sich der Cursor befindet und dann gegebenenfalls die notwendige Aktion eingeleitet.

Wenn ein Bildschirmpunkt mit einer neuen Farbe belegt worden ist, so wird danach der neu definierte Sprite mit Hilfe von SPRITESET ausgegeben.

Die dritte Entscheidungsraute dient schließlich zur Ansteuerung der Sonderfunktionen. Durch Druck auf <^> gelangt man dabei in ein Funktionswahlmenü, das die Unterfunktionen "Sprite malen", "Abspeichern eines Darstellungsprogramms", "Farbwahl" und "neuer Sprite" beinhaltet. Was die einzelnen Routinen dabei bewirken, können Sie relativ einfach anhand der Datenflußangaben feststellen. Alle Unterfunktionen rufen nur schon bekannte Unterprogramme auf.

Nach dieser Grobanalyse kommen wir nun zur eigentlichen Programmbeschreibung.

## 8.5 Die Programmbeschreibung

Die einzelnen Teilprogramme, die wir bei der Beschreibung unseres Flußdiagramms schon kennengelernt haben, können wir im Programmquelltext relativ gut wiederfinden.

Am Anfang stehen die Routinen, die das Einlesen der Maschinenroutinen beinhalten. Die einzelnen Maschinenprogramme können Sie dabei gut anhand der REM-Markierungen erkennen. Ab Zeile 580 treffen wir dann auf die Bildschirmdefinition, die den Schirm in die einzelnen Windows aufteilt und die Farben definiert.

Weiter geht es dann bei Zeile 1040 mit der Abfrage der Spritegröße. Zuvor wird jedoch noch als Unterprogramm die Farbabfrageroutine ab Zeile 690 ausgeführt. Hier besteht die Möglichkeit die Farbdefinition zu verändern.

Der Programmteil SPRITEDEF auf den wir ab Zeile 1160 treffen, hat dann die Aufgabe, das Rechteckraster, also unsere Cursormatrix, zu setzen. Dazu werden die Farbwerte, die ja im Datenbereich unseres Sprites als Nibbles gespeichert sind, benutzt. Mit Hilfe dieser Angaben wird dann jeweils mit PEN die Zeichenfarbe festgelegt und das entsprechende Rechteck (CHR$(143)) ausgegeben.

Es folgt in 1300 bis 1360 die Darstellung unserer Farbreihe, ebenfalls in Form von farbigen Rechtecken. In 1360 wird abschließend dann erstmalig unser Sprite ausgegeben.

Wir kommen damit nun in die eigentliche Joystickabfrage, deren Prinzip uns ja schon bekannt ist. Auch die im nun folgenden Teil "Cursor bewegen" durchgeführten Direktabfragen von Tasten und Joystickrichtungen kennen wir bereits aus anderen Programmen.

In Zeile 1500 finden wir die Abfrage, ob <COPY> gedrückt wurde. Falls dies der Fall ist, läuft das Programm normal weiter. Ansonsten wird nur noch der Cursor mittels Aufruf von BB8A und BB8D dargestellt und es geht zurück in die Joystickabfrage. Die FOR-TO-Schleifen stellen dabei einfache Zeitverzögerungen dar.

Beim Setzen der Farben ist insbesondere die Übertragung der neuen Farbwerte in den Spritespeicher erwähnenswert. Sie finden diese Prozedur in Zeile 1620 und 1630. Hier werden die beiden Halbbytes mit Hilfe der Variablen z bestimmt und dann ein Byte, das die neue Farbkombination enthält, zusammengebaut. 1640 gibt dieses dann wieder an den Spritespeicher zurück. Danach ruft dieser Programmteil den Shape zu einer nochmaligen Darstellung auf.

Für das Setzen des Sprites ist dabei folgendes zu beachten. SPRITESET wendet beim Setzen des Shapes eine XOR-Verknüpfung der Bildpunkte (vergleiche Kapitel 2.4) an. Die XOR-Verknüpfung läßt den Hintergrund unverändert, sofern sie zweimal auf denselben Bildpunkt angewandt wird.

Das Löschen eines Sprites geschieht daher durch nochmaligen Aufruf. Um die Wiederherstellung des Hintergrundes braucht man sich bei diesem Verfahren keine Sorgen zu machen.

Zum Abschluß unserer Programmbeschreibung noch kurz ein paar Worte zum Funktionswahlmenü und den damit erreichbaren Routinen. Die einzige erwähnenswerte Funktion stellt hierbei das Unterprogramm "Sprite speichern" dar. Ab Zeile 1920 finden Sie die Ausführungsbefehle für diese Funktion.

Zunächst wird der Name unseres Sprites abgefragt und dann eine Datei auf diesen Namen eröffnet. Danach wird nun unser Darstellungsprogramm Zeile für Zeile in die Stringvariable N$ geladen - das Prinzip kennen wir ja schon vom DESIGNER - und dann schließlich an die Ausgabedatei ausgegeben.

Wie ein solches Programm dann nach dem Speichervorgang aussieht, zeigt Ihnen das kleine Testprogramm Ballon. Die Zeilen bis 100 wurden dabei vom Editor erzeugt. Ab 100 sind dann noch einige Kommandos angefügt worden, die zunächst den abgespeicherten Sprite, einen CPC-Ballon an mehreren zufälligen Positionen darstellen, worauf dann eine Cursorabfrageroutine folgt, die es uns ermöglicht, mit einem dieser Ballons (der an der Position 50,50 dargestellt wurde) vor unserer Bergwelt in verschiedene Richtungen zu gleiten.

Das Programm demonstriert dabei auch gleich, wie Sprite-Bewegungen programmiert werden können. Wenn Sie mit der großen Routine Probleme haben, können Sie auch zuerst einmal Zeile 250 aufrufen. Hier wird unser Luftballon automatisch diagonal über den Schirm bewegt. Nach derselben Methode können Sie nun auch Raumschiffe, Männchen etc. über den Bildschirm "jagen". Alles, was Sie dazu benötigen, ist ein mit dem Editor erzeugter Sprite, den Sie mit dem Kommando |SPRITE aufrufen. Das "|", das Sie auch an einigen Stellen im Listing wiederfinden, stellt dabei das Druckerzeichen für den Erweiterungsstrich (SHIFT + @) dar.

Experimentieren Sie nun ruhig einmal ein wenig mit diesem neuen Befehl und überlegen Sie doch einmal, in welchem Ihrer Programme Sie Sprites sinnvoll einsetzen können. Zum Abschluß noch zwei kleine Tips:

Wenn das Darstellungsprogramm einmal durchlaufen wurde, so kann es durch DELETE oder ein Nachladeprogramm gelöscht werden. Das Darstellungsprogramm bis Zeile 100 enthält ausschließlich Laderoutinen und als einziges auszuführendes Kommando den Aufruf der RSX-Initialisierungsroutine mit CALL. Nachdem dieses Kommando ausgeführt

wurde, ist die RSX dann etabliert, und benötigt die Laderoutinen nicht mehr.

Ein weiterer Punkt betrifft die Bewegungsgeschwindigkeit unseres Sprites. Statt der von der Cursorabfrageroutine durchgeführten Sprünge um 1 oder maximal 2 können Sie speziell in Y-Richtung natürlich auch mit viel größeren Änderungen arbeiten. Wenn Sie beispielsweise die Y-Änderung auf den Wert 10 erhöhen, so "jagt" unser Ballon jetzt fast wie ein Starfighter über den Schirm.

## Listing 22: *Programm SPRITEEDITOR*

```
10 REM ******************
20 REM ** Spriteeditor **
30 REM ******************
40 REM
50 REM **********************
60 REM ** Initialisierung **
70 REM **********************
80 MEMORY 40000:SPEED WRITE 1
90 POKE &A000,155:POKE &A001,1
100 x=9:y=1:cf=1:DIM f$(15)
110 SYMBOL 255,255,129,129,129,129,129,129,255
120 SYMBOL 254,255,255,255,195,195,255,255,255
130 REM **************
140 REM ** Umlaute **
150 REM **************
160 SYMBOL 240,102,0,60,102,102,102,60,0
170 REM ** ae **
180 SYMBOL 241,102,0,120,12,124,204,118,0
190 REM ** ue **
200 SYMBOL 242,102,0,102,102,102,102,62,0
210 REM ****************************
220 REM ** sprite systemroutinen **
230 REM ****************************
240 REM
250 REM ****************
260 REM ** spriteset **
270 REM ****************
280 RESTORE:FOR i%=&A002 TO &A100
290 READ z$:IF z$="xx" THEN 430
300 POKE i%,VAL("&"+z$):NEXT i%
310 NEXT
320 DATA ed,5b,00,a0,01,00,a2,cd,03,b9
330 DATA c5,26,00,7b,e6,f8,6f,29,4d,44
340 DATA 29,29,09,7b,e6,07,cb,27,cb,27
350 DATA cb,27,c6,c0,84,67,06,00,4a,09
360 DATA d1,1a,47,13,1a,4f,13,c5,e5,1a
370 DATA cd,00,a1,ae,77,13,23,10,f6,e1
380 DATA 7c,c6,08,67,30,05,01,af,3f,ed
390 DATA 42,c1,0d,20,e4,cd,00,b9,c9,xx
400 REM **************
410 REM **converter**
420 REM **************
430 FOR i%=&A100 TO &A200
440 READ z$:IF z$="xx" THEN 540
450 POKE i%,VAL("&"+z$):NEXT i%
460 DATA c5,06,02,4f,3e,00,cb,79,28,02
470 DATA cb,cf,cb,71,28,02,cb,ef,cb,69
480 DATA 28,02,cb,df,cb,61,28,02,cb,ff
490 DATA 10,04,cb,1f,c1,c9,cb,11,cb,11
500 DATA cb,11,cb,11,cb,27,18,d6,xx,xx
510 REM **********
520 REM **Saver**
530 REM **********
540 RESTORE 570:FOR i%=&A080 TO &A100
550 READ z$:IF z$="xx" THEN 590
560 POKE i%,VAL("&"+z$):NEXT i%
570 DATA 3a,90,a0,cd,00,a1,32,90,a0,c9,xx,xx
580 REM ****************************
590 REM ** PRG-Initialisierung **
600 REM ****************************
610 MODE 1:INK 0,0:INK 1,27:INK 2,24:INK 3,6
620 WINDOW#1,1,40,1,3:WINDOW#2,1,40,4,22:WINDOW#3,1,40,23,25
630 PAPER#1,1:PEN#1,2:PAPER#2,0:PEN#2,3:PAPER#3,1:PEN#3,2
640 CLS#1:CLS#2:CLS#3:BORDER 0
650 GOSUB 690:GOTO 1040
660 REM ******************
670 REM ** Farbabfrage **
680 REM ******************
```

```
690 f$(0)=" 0":f$(1)="27"
700 f$(2)=" 8":f$(3)=" 6"
710 f$(4)="24":f$(5)="21"
720 f$(5)="21":f$(6)="15"
730 f$(7)=" 7":f$(8)="10"
740 f$(8)="10":f$(9)="12"
750 f$(10)=" 4":f$(11)=" 2"
760 f$(12)="26":f$(13)=" 6  24"
770 f$(14)=" 2   6":f$(15)=" 6  21"
780 CLS#3:CLS#1:LOCATE#1,15,2:PRINT#1,"Farbdefinition"
790 CLS#2:PRINT#3:PRINT#3," Wollen Sie diese Farbwahl ";CHR$(241)
;"ndern j/n ?"
800 PRINT#2:PRINT#2,"  Nr.     Farbe(n)"
810 FOR i= 0 TO 15
820 LOCATE#2,3,3+i:PRINT#2,RIGHT$("0"+STR$(i),2);"        "f$(i):N
EXT i
830 z$=INKEY$:IF LOWER$(z$)<>"j"THEN IF LOWER$(z$)<>"n" THEN 830
ELSE RETURN
840 CLS#3:PRINT#3
850 PRINT#3," Welches Register ";CHR$(241);"ndern";:INPUT#3,reg
860 IF reg<2 THEN PRINT#3," Nur Nr.2 bis Nr.15!!":FOR i=1 TO 2500
:NEXT:GOTO 840
870 INPUT#3," Farbcode 1 ";f1
880 PRINT#3," Doppelbelegung (Blinken) erw";CHR$(242);"nscht j/n"
890 z$=INKEY$:IF z$="n" OR z$="N" THEN 940
900 IF z$<>"j" AND z$<>"J" THEN 890
910 INPUT#3," Farbcode 2 ";f2
920 f$(reg)=RIGHT$("00"+STR$(f1),2)+"  "+RIGHT$("00"+STR$(f2),2)
930 GOTO 950
940 f$(reg)=RIGHT$("00"+STR$(f1),2)
950 LOCATE#2,12,3+reg:PRINT#2,f$(reg);SPACE$(9)
960 PRINT#3
970 PRINT#3," Weiteres Register ";CHR$(241);"ndern j/n?"
980 PRINT#3
990 z$=INKEY$:IF LOWER$(z$)="n" THEN RETURN
1000 IF LOWER$(z$)<>"j" THEN 990 ELSE 840
1010 REM ********************
1020 REM ** Spriteabfrage **
1030 REM ********************
1040 CLS#1:CLS#2:CLS#3
1050 PRINT#1:PRINT#1,"              Dimensionierung"
1060 INPUT#3," Ausdehnung in x-Richtung (2-12)";xr
1070 IF xr<2 OR xr>12 THEN PRINT#3," Bereich "+CHR$(242)+"berschr
itten!!":GOTO 1060
1080 IF INT(xr/2)<>xr/2 THEN PRINT#3," nur gerade Zahlen!!":GOTO
1060
1090 INPUT#3," Ausdehnung in y-Richtung (1-25)";yr
1100 IF yr<1 OR yr>25 THEN PRINT#3," Bereich "+CHR$(242)+"berschr
itten!!":GOTO 1090
1110 POKE &A200,xr/2:POKE &A201,yr
1120 FOR i=&A202 TO &A400:POKE i,0:NEXT
1130 REM ****************
1140 REM ** Spritedef **
1150 REM ****************
1160 MODE 0
1170 FOR i=2 TO 15
1180 u=VAL(LEFT$(f$(i),3)):v=VAL(MID$(f$(i),4))
1190 IF (v=0 AND LEN(f$(i))>4) OR v<>0 THEN 1210
1200 INK i,u:GOTO 1220
1210 INK i,u,v
1220 NEXT i
1230 PEN 1:FOR j= 0 TO yr-1:FOR i= 0 TO xr-2 STEP 2:LOCATE 9+i,j+
1
1240 w=PEEK(43010+xr/2*j+i/2):v=INT(w/16):u=w-16*v
1250 IF v=0 THEN PEN 1:PRINT CHR$(255)
1260 IF v<>0 THEN PEN v:PRINT CHR$(143)
1270 LOCATE 10+i,j+1:IF u=0 THEN PEN 1:PRINT CHR$(255)
1280 IF u<>0 THEN PEN u:PRINT CHR$(143)
1290 NEXT i,j:PEN 1
1300 REM ************************
```

```
1310 REM ** Ausgabe Farbreihe **
1320 REM ************************
1330 LOCATE 1,1:PRINT"Farben:"
1340 LOCATE 1,3:PRINT" 0    "+CHR$(255)
1350 FOR i= 1 TO 15:LOCATE 1,3+i:PRINT RIGHT$("0"+STR$(i),2)+"
"+CHR$(15)+CHR$(i)+CHR$(254)+CHR$(15)+CHR$(1)
1360 NEXT i:CALL &A002
1370 REM ***********************
1380 REM ** Joystickabfrage **
1390 REM ***********************
1400 IF INKEY(0) =0 OR (JOY(0) AND 1)=1 THEN y=MAX(y-1,1)
1410 IF INKEY(2) =0 OR (JOY(0) AND 2)=2 THEN y=MIN(y+1,25)
1420 IF INKEY(8) =0 OR (JOY(0) AND 4)=4 THEN x=MAX(x-1,6)
1430 IF INKEY(1) =0 OR (JOY(0) AND 8)=8 THEN x=MIN(x+1,8+xr)
1440 GOSUB 1500
1450 IF INKEY(24)=0 THEN 1710
1460 GOTO 1400
1470 REM ********************
1480 REM ** Cursor bewegen **
1490 REM ********************
1500 IF INKEY(9)=-1 AND JOY(0)<>16 THEN 1660
1510 IF x<>6 THEN 1560
1520 IF 2<y AND y<19 THEN cf= y-3
1530 IF cf=0 OR y<3 OR y>18 THEN 1550
1540 LOCATE x,y:PEN cf:PRINT CHR$(254)
1550 GOTO 1660
1560 f1=0:IF 8<x AND y<=yr THEN f1=1
1570 IF f1<>1 THEN 1660
1580 PEN cf:LOCATE x,y
1590 IF cf=0 THEN PEN 1:PRINT CHR$(255); ELSE PRINT CHR$(143);
1600 CALL &A002
1610 z=(x-8+(y-1)*xr)/2
1620 xw=cf*16+(PEEK(&A201+z+0.5)AND 15)
1630 zw=cf+(PEEK(&A201+z)AND 240)
1640 IF INT(z)<>z THEN POKE &A201+z+0.5,xw ELSE POKE &A201+z,zw
1650 CALL &A002
1660 FOR i= 1 TO 20:NEXT:LOCATE x,y:CALL &BB8A:FOR i= 1 TO 100:NE
XT:CALL &BB8D
1670 RETURN
1680 REM ************************
1690 REM ** Sonderfunktionen **
1700 REM ************************
1710 MODE 1:INK 0,0:INK 1,27:INK 2,24:INK 3,6
1720 WINDOW#1,1,40,1,3:WINDOW#2,1,40,4,22:WINDOW#3,1,40,23,25
1730 PAPER#1,1:PEN#1,2:PAPER#2,0:PEN#2,3:PAPER#3,1:PEN#3,2
1740 CLS#1:CLS#2:CLS#3:BORDER 0
1750 LOCATE#1,15,2:PRINT#1,"Spritewahl"
1760 PRINT#2:PRINT#2,"          Sprite speichern (1)"
1770 PRINT#2:PRINT#2,"          Farbwahl         (2)"
1780 PRINT#2:PRINT#2,"          Sprite malen     (3)"
1790 PRINT#2:PRINT#2,"          neuen Sprite     (4)"
1800 z$=INKEY$:IF z$="" THEN 1800
1810 IF z$="1" AND xr=0 THEN PRINT#3," kein Sprite vorhanden!!":F
OR i=1 TO 2000:NEXT:GOTO 1740
1820 IF z$="1" THEN 1880
1830 IF z$="2" THEN GOSUB 780:GOTO 1720
1840 IF z$="3" THEN 1160
1850 IF z$<>"4" THEN 1800
1860 PRINT#3:PRINT#3,"          Sind Sie sicher  j/n"
1870 z$=LOWER$(INKEY$):IF z$="n" THEN 1720 ELSE IF z$<>"j" THEN 1
870 ELSE 650
1880 CLS#1:CLS#3
1890 REM ******************
1900 REM ** Abspeichern **
1910 REM ******************
```

```
1920 PRINT#1:PRINT#1,"                Abspeichern"
1930 INPUT"Name des Sprites";n$
1940 PRINT#3,"Dr"+CHR$(242)+"cken Sie PLAY und RECORD und dann ei
-ne beliebige Taste."
1950 IF INKEY$="" THEN 1950
1960 OPENOUT"!"+n$
1970 n$="1 MEMORY 40000:POKE &A000,155:POKE &A001,1:border 0:RES1
ORE:FOR i%=&A002 TO &A100:READ z$:IF z$="+CHR$(34)+"xx"+CHR$(34)+
"
THEN 10":PRINT#9,n$
1980 n$="2 POKE i%,VAL("+CHR$(34)+"&"+CHR$(34)+"+z$):NEXT i%":PRI
NT#9,n$
1990 n$="3 DATA dd,56,02,01,00,a2,cd,03,b9":PRINT#9,n$
2000 n$="4 DATA c5,26,00,7b,e6,f8,6f,29,4d,44":PRINT#9,n$
2010 n$="5 DATA 29,29,09,7b,e6,07,cb,27,cb,27":PRINT#9,n$
2020 n$="6 DATA cb,27,c6,c0,84,67,06,00,4a,09":PRINT#9,n$
2030 n$="7 DATA d1,1a,47,13,1a,4f,13,c5,e5,1a,00,00,00":PRINT#9,n
$
2040 n$="8 DATA ae,77,13,23,10,f6,e1,7c,c6,08,67,30,05,01,af,3f":
PRINT#9,n$
2050 n$="9 DATA ed,42,c1,0d,20,e4,cd,00,b9,c9,xx":PRINT#9,n$
2060 n$="10 for i=&a200 to &a400:READ z$:IF z$="+CHR$(34)+"xx"+CH
R$(34)+" THEN 50:else poke i,val(z$):next i":PRINT#9,n$
2070 n$="11 data "+MID$(STR$(PEEK(41472)),2)+","+MID$(STR$(PEEK(4
1473)),2)+","
2080 FOR i=41473 TO 41483+PEEK(&A200)*PEEK(&A201) STEP 10:FOR j=1
 TO 10:z=PEEK(i+j):POKE &A090,z:CALL &A080:z=PEEK(&A090):GOSUB 22
00
:n$=n$+v$+","
2090 NEXT j:u=(i-41473)/10:PRINT#9,LEFT$(n$,LEN(n$)-1):n$=STR$(u+
12)+" data ":NEXT i
2100 n$=n$+"xx":PRINT#9,n$
2110 n$="50 ":FOR i=0 TO 15:u=VAL(LEFT$(f$(i),3)):v=VAL(MID$(f$(i
),4)):IF v=0 AND LEN(f$(i))<4 THEN n$=n$+"ink"+STR$(i)+","+STR$(u
)+
":":GOTO 2130
2120 n$=n$+"ink"+STR$(i)+","+STR$(u)+","+STR$(v)+":"
2130 NEXT i:PRINT#9,n$
2140 n$="60 RESTORE 70:FOR i=&9fe0 to &9fff:read a$:poke i,val("+
CHR$(34)+"&"+CHR$(34)+"+a$):next i"
2150 PRINT#9,n$
2160 n$="70 data 01,f0,9f,21,fd,9f,c3,d1,bc,c9,0,0,0,0,0,0,f5,9f,
c3,02,a0,53,50,52,49,54,c5,0,0,0,0,0,0,0
2170 PRINT#9,n$
2180 n$="80 call&9fe0":PRINT#9,n$
2190 n$="100 rem ** Ihr Programm **":PRINT#9,n$:CLOSEOUT:GOTO 171
0
2200 v$=RIGHT$("000"+MID$(STR$(z),2),3):RETURN
2210 FOR i=&A200 TO &A300:PRINT HEX$(i),HEX$(PEEK(i)):NEXT
```

## Listing 23: *Programm BALLON*

```
1 MEMORY 40000:POKE &A000,155:POKE &A001,1:BORDER 0:RESTORE:FOR i
%=&A002 TO &A100:READ z$:IF z$="xx" THEN 10
2 POKE i%,VAL("&"+z$):NEXT i%
3 DATA dd,56,02,01,00,a2,cd,03,b9
4 DATA c5,26,00,7b,e6,f8,6f,29,4d,44
5 DATA 29,29,09,7b,e6,07,cb,27,cb,27
6 DATA cb,27,c6,c0,84,67,06,00,4a,09
7 DATA d1,1a,47,13,1a,4f,13,c5,e5,1a,00,00,00
8 DATA ae,77,13,23,10,f6,e1,7c,c6,08,67,30,05,01,af,3f
9 DATA ed,42,c1,0d,20,e4,cd,00,b9,c9,xx
10 FOR i=&A200 TO &A400:READ z$:IF z$="xx" THEN 50:ELSE POKE i,VA
L(z$):NEXT i
11 DATA 6,25,000,000,000,000,000,000,000,000,000,000
12 DATA 000,000,000,000,243,243,000,000,000,064
13 DATA 192,192,128,000,000,192,240,240,192,000
14 DATA 000,208,240,240,224,000,064,240,240,240
15 DATA 240,128,064,228,051,240,218,128,064,216
16 DATA 114,103,240,128,064,216,114,103,240,128
17 DATA 064,216,051,229,240,128,064,216,114,229
18 DATA 240,128,000,196,114,240,202,000,000,208
19 DATA 240,240,224,000,000,064,240,240,128,000
20 DATA 000,064,208,224,128,000,000,000,192,192
21 DATA 000,000,000,000,032,016,000,000,000,000
22 DATA 032,016,000,000,000,000,032,016,000,000
23 DATA 000,065,097,146,130,000,000,065,097,146
24 DATA 130,000,000,065,195,195,130,000,000,065
25 DATA 195,195,130,000,000,000,000,000,000,000
26 DATA 000,000,000,000,000,000,000,000,000,000
27 DATA 000,000,000,000,000,000,000,000,000,000
28 DATA xx
50 INK 0, 0:INK 1, 27:INK 2, 8:INK 3, 6:INK 4, 24:INK 5, 21:INK 6
, 15:INK 7, 7:INK 8, 10:INK 9, 12:INK 10, 4:INK 11, 2:INK 12, 26:
IN
K 13, 6, 24:INK 14, 2, 6:INK 15, 6, 21:
60 RESTORE 70:FOR i=&9FE0 TO &9FFF:READ a$:POKE i,VAL("&"+a$):NEX
T i
70 DATA 01,f0,9f,21,fd,9f,c3,d1,bc,c9,0,0,0,0,0,0,f5,9f,c3,02,a0,
53,50,52,49,54,c5,0,0,0,0,0,0,0
80 CALL &9FE0
100 REM ** Ihr Programm **
110 MODE 0
120 ORIGIN 0,0:DRAWR 100,130,11:DRAWR 30,-80:DRAWR 30,-30:DRAWR 4
0,70:DRAWR 20,30
130 DRAWR 100,130:DRAWR 30,-80:DRAWR 30,-30:DRAWR 40,70:DRAWR 20,
30
140 DRAWR 100,-30:DRAWR 30,-80:DRAWR 30,30:DRAWR 140,70:DRAWR 20,
-30
150 FOR i=1 TO 5:x=RND(1)*74:y=RND(1)*175:öSPRITE,x,y
160 NEXT i
170 x=50:y=50:xa=50:ya=50
180 CALL &BD19:öSPRITE,x,y
190 IF INKEY(0)=0 THEN y=MAX(y-2,0)
200 IF INKEY(2)=0 THEN y=MIN(y+2,174)
210 IF INKEY(8)=0 THEN x=MAX(x-1,1)
220 IF INKEY(1)=0 THEN x=MIN(x+1,74)
230 IF xa=x AND ya=y THEN 190
240 öSPRITE,xa,ya:xa=x:ya=y:öSPRITE,x,y:GOTO 190
250 FOR i=1 TO 70 STEP 2:öSPRITE,i,i:öSPRITE,i,i:NEXT i
```

# Anhang: Nützliche Systemroutinen

| | |
|---|---|
| BB8A | Setzt den Cursor an der durch LOCATE angegebenen Position. |
| BB8D | Löscht den Cursor wieder. Dieser muß dabei gesetzt worden sein. Wird ein nicht gesetzter Cursor gelöscht oder ein gesetzter noch einmal gesetzt, so kann es zu seltsamen Effekten kommen. |
| BC05 | SCR SET OFFSET gibt die in HL gespeicherte Adresse als Bildschirmverschiebung, Anfangsadresse des Schirms an die Hardware weiter. |
| BC08 | SCR SET BASE gibt den in A gespeicherten Wert als signifikantes (höherwertiges) Byte der Bildschirmbasis-adresse an die Hardware aus. |
| BC0B | SCR GET LOCATION fragt die aktuellen Werte für Bild-schirmbasis und -offset ab und gibt diese in A und HL zurück. |

## Stichwortverzeichnis

# Weitere Fachbücher aus unserem Verlagsprogramm

## COMMODORE

### Das Commodore 128-Handbuch
**Juli 1985, 383 Seiten**

In diesem Buch finden Sie einen Querschnitt durch alle wichtigen Funktions- und Anwendungsbereiche des Commodore 128. Sie werden mit dem C64/C128-Modus und der Benutzung von CP/M 3.0 vertraut gemacht, erfahren alles über die Grafik- und Soundmöglichkeiten des C128, lernen die Techniken der Speicherverwaltung und das Banking kennen und werden in die Programmierung mit Assemblersprache sowie die Grafikprogrammierung des 80-Zeichen-Bildschirms eingeführt. Ein umfassendes Handbuch, das Sie immer griffbereit haben sollten!

**Best.-Nr. MT 809, ISBN 3-89090-195-9**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### BASIC 7.0 auf dem Commodore 128
**Juli 1985, 239 Seiten**

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128PC zu ziehen. Sie eignen sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen: Listenverarbeitung, indexsequentielle Dateiverwaltung, Grafikdarstellungen und Sounderzeugung. Ein unentbehrliches Lehrbuch, das sich auch für den geübten Anwender als Nachschlagewerk eignet.

**Best.-Nr. MT 808, ISBN 3-89090-170-0**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### WordStar 3.0 mit MailMerge für den Commodore 128 PC
**November 1985, 435 Seiten**

WordStar ist ein umfangreiches und leistungsfähiges Textverarbeitungsprogramm und damit sicherlich zu Recht das meistverkaufte Programm seiner Art. Doch bedeutet dies nicht unbedingt, daß es auch einfach zu bedienen ist. Hier setzt dieses Buch an: Es macht in vorbildlicher Weise mit allen Möglichkeiten von WordStar und MailMerge vertraut und ist damit eine ideale Ergänzung zum Handbuch. Es versammelt alle wichtigen Informationen für den effektiven Einsatz dieser Programme auf dem Commodore 128 PC.

**Best.-Nr. MT 780, ISBN 3-89090-181-6**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### dBASE II für den Commodore 128 PC
**November 1985, 280 Seiten**

Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Reportgenerator, der im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt.

**Best.-Nr. MT 838, ISBN 3-89090-189-1**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### Multiplan für den Commodore 128 PC
**November 1985, 226 Seiten**

MULTIPLAN wurde ursprünglich für das 16-Bit-Betriebssystem MS-DOS entwickelt. Inzwischen ist aber auch die in diesem Buch beschriebene CP/M-Version für den Commodore 128 PC auf dem Markt, die den vollen Leistungsumfang der 16-Bit-Version enthält.

Das vorliegende Buch soll eine praktische Einführung in den Umgang mit MULTIPLAN auf dem Commodore 128 PC geben. Anhand von praxisnahen Beispielen werden alle Befehle und Funktionen in der Reihenfolge beschrieben, die der Arbeit in der Praxis entspricht. Bereits nach Abschluß des ersten Kapitels werden Sie in der Lage sein, eigene kleine MULTIPLAN-Anwendungen zu realisieren.

**Best.-Nr. MT 836, ISBN 3-89090-187-5**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### Die Floppy 1571
**Dezember 1985, ca. 400 Seiten**

Dieses Buch soll es sowohl dem Einsteiger als auch dem fortgeschrittenen Programmierer ermöglichen, die vielfältigen Möglichkeiten dieses neuen Gerätes voll auszuschöpfen. Sämtliche Betriebsarten und Diskettenformate werden ausführlich erläutert. Anhand vieler Beispiele werden Sie in die Dateiverwaltung mit dieser Floppy eingeführt. Der Benutzer lernt die zahlreichen Systembefehle kennen und erfährt zugleich wichtige Grundlagen für das Arbeiten mit dem Betriebssystem CP/M.

**Best.-Nr. MT 793, ISBN 3-89090-185-9**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### C 64 Fischertechnik Messen, Steuern, Regeln
**November 1985, ca. 200 Seiten**

Ziel dieses Buches ist es, jedem Besitzer eines Commodore 64/VC20 eine neue Welt zu erschließen: die Welt der Roboter, der computergesteuerten Fertigungsstraßen. Alles, was Sie benötigen, ist einer der beiden genannten Computer und der Fischertechnik Computing Baukasten mit dazugehörigem Interface.

**Best.-Nr. MT 844, ISBN 3-89090-194-8**
**(sFr. 27,60/öS 233,20)** **DM 29,90**

### Mini-CAD mit Hi-Eddi-Plus
**November 1985, ca. 160 Seiten inkl. Diskette**

Neben den »Standardbefehlen« zum Setzen und Löschen von Punkten, dem Zeichnen von Linien, Kreisen und Rechtecken sowie dem Ausfüllen unregelmäßiger Flächen und dem Verschieben und Duplizieren von Bildschirmbereichen bietet Hi-Eddi eine Reihe von Besonderheiten, die dieses Programm von anderen Grafikprogrammen abhebt: bis zu sieben Grafikbildschirme stehen gleichzeitig zur Verfügung; es besteht die Möglichkeit, Text in die Grafik einzufügen, die Bildschirme zu verknüpfen oder in schneller Folge durchzuschalten.

**Best.-Nr. MT 735, ISBN 3-89090-136-0**
**(sFr. 44,20/öS 374,40)** **DM 48,—**

Die angegebenen Preise sind Ladenpreise

# Weitere Fachbücher aus unserem Verlagsprogramm

### BASIC-Programmierung PC-10/PC-20
**Oktober 1985, ca. 500 Seiten**
Ein amerikanisch-lockerer BASIC-Kurs von dem kalifornischen Professor Lien. Durch seine Systematik ideal als Kursunterlage für PC-10/20 und Kompatible. Mit Einführung in das PC-10-System und Tastendarstellung im Text.
**Best.-Nr. PW 559, ISBN 3-921803-66-7**
**(sFr. 54,30/öS 460,20)** **DM 59,—**

### C64 - Wunderland der Grafik
**Juli 1985, 236 Seiten inklusive Beispieldiskette**
Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen. Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann. Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik.
**Best.-Nr. MT 756, ISBN 3-89090-130-1**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### Das C64-Profihandbuch
**Juli 1985, 410 Seiten**
Ein Buch, das alle wichtigen Informationen für professionelle Anwendungen mit dem C64 enthält. Mit allgemeinen Algorithmen, die auch auf andere Rechner übertragbar sind, und vielen Utilities, getrennt nach BASIC- und Maschinenprogrammen. Besonders nützlich: erweiterte PEEK- und POKE-Funktionen.
**Best.-Nr. MT 749, ISBN 3-89090-110-7**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### Programmieren unter CP/M mit dem C64
**Juni 1985, 290 Seiten**
Wenn Sie wissen wollen, wie das Betriebssystem CP/M 2.2 auf dem C64 implementiert ist, außerdem einiges über Turbo-Pascal, Nevada-Fortran, MBASIC-80 erfahren wollen, dann ist dieses Buch genau richtig für Sie! Mit Schaltplänen zur eigenen Fertigung des CP/M-Moduls. Für eingefleischte C64-Profis.
**Best.-Nr. MT 751, ISBN 3-89090-091-7**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### C64 - Programmieren in Maschinensprache
**August 1985, 327 Seiten inklusive Beispieldiskette**
In diesem Buch finden Sie über 100 Beispiele zur Assembler-Programmierung mit viel Kommentar und Hintergrundinformationen: das Schreiben von Maschinenprogrammen · Rechnen und Texten mit vorhandenen Routinen · Bedienung von Drucker und Floppy · wie man BASIC- und Maschinenprogramme verknüpft · Erstellen von eigenen Befehlen in Modulform. Für Profis!
**Best.-Nr. MT 830, ISBN 3-89090-168-9**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### Einführungskurs: Commodore 64
**Mai 1984, 276 Seiten**
Die Programmiersprache BASIC · Einsatzgebiete des Commodore 64-BASIC: Grafik, Musik, Dateiverwaltung · mit vielen Beispielprogrammen, häufig benötigten Tabellen und nützlichen Tips · für Einsteiger und Fortgeschrittene.
**Best.-Nr. MT 685, ISBN 3-89090-017-8**
**(sFr. 35,—/öS 296,40)** **DM 38,—**

### Commodore 64 - leicht verständlich
**Juni 1984, 154 Seiten**
Informationen für den Computer-Neuling · Installation und Inbetriebnahme · Programmieren in BASIC · Grafik und Töne · Auswahl von Hardware und Zubehör · Software für Ihren Computer · die ideale Einführung in das Arbeiten mit Ihrem Commodore 64.
**Best.-Nr. MT 700, ISBN 3-89090-022-4**
**(sFr. 27,50/öS 232,40)** **DM 29,80**

### Das Commodore 64-LOGO-Arbeitsbuch
**September 1984, 225 Seiten**
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
**Best.-Nr. MT 720, ISBN 3-89090-063-1**
**(sFr. 31,30/öS 265,20)** **DM 34,—**

### Der sensible Commodore 64
**Januar 1985, 130 Seiten**
Eine Softwaresammlung zu den technologischen Neuerscheinungen im Commodore 64 · für Erstbenutzer wie für Experten ein Buch zur optimalen Softwarenutzung.
**Best.-Nr. PW 727, ISBN 3-921803-45-4**
**(sFr. 27,50/öS 232,40)** **DM 29,80**

### 35 ausgesuchte Spiele für Ihren Commodore 64
**September 1984, 141 Seiten**
Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore 64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die Ihre Programmierkenntnisse vertiefen wollen!
**Best.-Nr. MT 774, ISBN 3-89090-064-X**
**(sFr. 23,—/öS 193,40)** **DM 24,80**

### BASIC mit dem Commodore 64
**April 1984, 320 Seiten**
Ein BASIC-Lehrbuch für den jugendlichen Anfänger · übersichtlich gegliederte Lernprogramme · Alles über INPUT-GOTO · Let-Befehle · Editorfunktionen · POKE-Befehle für die Grafik · geeignet auch als Leitfaden für Lehrer und Eltern.
**Best.-Nr. MT 657, ISBN 3-922120-91-1**
**(sFr. 44,20/öS 374,40)** **DM 48,—**

Die angegebenen Preise sind Ladenpreise

## Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

# Weitere Fachbücher aus unserem Verlagsprogramm

### Das große Spielebuch – Commodore 64
**Februar 1984, 141 Seiten**
46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.
**Best.-Nr. MT 603, ISBN 3-922120-63-6**
**(sFr. 27,50/öS 232,40)** **DM 29,80**
**Best.-Nr. MT 604 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Spiele für den Commodore 64
**November 1984, 196 Seiten**
Bewährte alte und raffinierte neue Spiele für Ihren Commodore 64 · klar und übersichtlich gegliederte Programme im Commodore-BASIC · Sie lernen: wie man Unterprogramme einsetzt · eine Tabelle aufbauen und verarbeiten · Programme testen · mit vielen Programmiertricks · für Anfänger.
**Best.-Nr. MT 792, ISBN 3-89090-074-7**
**(sFr. 23,—/öS 193,40)** **DM 24,80**
**Best.-Nr. MT 795 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Grafik & Musik auf dem Commodore 64
**Oktober 1984, 336 Seiten**
68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.
**Best.-Nr. MT 743, ISBN 3-89090-033-X**
**(sFr. 35,—/öS 296,40)** **DM 38,—**

### Commodore 64 Listings – Band 1: Spiele
**Oktober 1984, 199 Seiten**
Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise.
**Best.-Nr. MT 748, ISBN 3-89090-068-2**
**(sFr. 23,—/öS 193,40)** **DM 24,80**
**Best.-Nr. MT 804 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Commodore 64 Listings
### Band 2: Dateiverwaltung · Schule · Hobby
**Oktober 1984, 179 Seiten**
Ein Buch mit Programmen für die ganze Familie · DATAVE – Eine Dateiverwaltung · mathematische Funktionen · Konjugation und Deklination in Latein · Regressionsanalyse · Bundesligatabelle.
**Best.-Nr. MT 766, ISBN 3-89090-071-2**
**(sFr. 23,—/öS 193,40)** **DM 24,80**

### Commodore 64 – Multiplan
**1984, 230 Seiten**
Multiplan jetzt auch für den Commodore 64 · der volle Leistungsumfang der 16-Bit-Version · Einführung in die Arbeitsweise von Tabellenkalkulationsprogrammen · praxisnahe Beispiele · Beschreibung aller Befehle und Funktionen. Ein Buch nicht nur für Anfänger.
**Best.-Nr. MT 655, ISBN 3-922120-89-X**
**(sFr. 44,20/öS 374,40)** **DM 48,—**

### Das Commodore 64-Buch, Bd. 1: Ein Leitfaden für Erstanwender
**Mai 1984, 270 Seiten**
Der Commodore 64 und seine Handhabung · Einführung in die Grafik · Balkendiagramme · Einführung in die Spritetechnik · BASIC-Erweiterungen in Assembler · Ein Leitfaden für Erstanwender, die sich bereits BASIC-Kenntnisse angeeignet haben. Alle Beispiele auf Diskette erhältlich!
**Best.-Nr. MT 591, ISBN 3-922120-61-X**
**(sFr. 44,20/öS 374,40)** **DM 48,—**
**Best.-Nr. MT 592 (Beispiele auf Diskette)**
**(sFr. 58,—/öS 522,—)** **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Das Commodore 64-Buch, Bd. 2: BASIC-Spiele
**Mai 1984, 181 Seiten**
Spiele nicht nur zum Abtippen · Programmlisting · Programmbeschreibung · Variablenübersicht · Programme nach Anleitung frei ergänzbar · das ideale Buch, um Programmieren spielend zu lernen · für Anfänger.
**Best.-Nr. MT 593, ISBN 3-922120-68-7**
**(sFr. 35,—/öS 296,40)** **DM 38,—**
**Best.-Nr. MT 594 (Beispiele auf Diskette)**
**(sFr. 58,—/öS 522,—)** **DM 58,—***
*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Die angegebenen Preise sind Ladenpreise

## Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

# Weitere Fachbücher aus unserem Verlagsprogramm

### Das Commodore 64-Buch, Bd. 5: Ein Leitfaden durch Simon's BASIC

**Juli 1984, 322 Seiten**

Ausführliche Besprechung aller Befehle · viele erklärende Beispiele · mit kommentiertem Assembler-Listing · das richtige Nachschlagewerk für den geübten Commodore 64-Benutzer.

**Best.-Nr. MT 599, ISBN 3-922120-71-7**
**(sFr. 35,—/öS 296,40)** **DM 38,—**
**Best.-Nr. MT 600 (Beispiele auf Diskette)**
**(sFr. 58,—/öS 522,—)** **DM 58,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Das Commodore 64-Buch, Bd. 6: Spiele

**Mai 1984, 190 Seiten**

Programmieren auf dem Commodore 64 spielend gelernt · leicht verständliche Spielanleitungen · Programmlisting mit anschließender Programmbeschreibung · Variablenübersicht · Tips zum Ändern und Ergänzen des Programms.

**Best.-Nr. MT 619, ISBN 89090-072-5**
**(sFr. 35,—/öS 296,40)** **DM 38,—**
**Best.-Nr. MT 620 (Beispiele auf Diskette)**
**(sFr. 58,—/öS 522,—)** **DM 58,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Das Commodore 64-Buch, Bd. 7: Ein Leitfaden für Profis

**August 1984, 210 Seiten**

Der Commodore 64 als Klaviatur · Noten schreiben mit hochauflösender Grafik · relative Dateien am Beispiel einer kleinen Adreßverwaltung · Joystick und Paddles · Grafikspeicher unter Kernal · Interrupt-Manager · für Profis, die die letzten Möglichkeiten ihres Commodore 64 ausreizen wollen.

**Best.-Nr. MT 731, ISBN 3-89090-067-4**
**(sFr. 35,—/öS 296,40)** **DM 38,—**
**Best.-Nr. MT 784 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

### Die Floppy 1541

**April 1985, 434 Seiten**

Für alle Programmierer, die mehr über ihre VC 1541-Floppystation erfahren wollen. Der Vorgang des Formatierens · das Schreiben von Files auf Diskette · die Funktionsweise von schnellen Kopier- und Ladeprogrammen · viele fertige Programme · Lesen und Beschreiben von defekten Disketten · Für Einsteiger und für fortgeschrittene Maschinensprache-Programmierer.

**Best.-Nr. MT 806, ISBN 3-89090-098-4**
**(sFr. 45,10/öS 382,20)** **DM 49,—**
**Best.-Nr. MT 710 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

## ATARI

### GEM für den Atari 520ST

**Juli 1985, 189 Seiten**

Eine programmierte Einweisung in die hervorragenden Möglichkeiten des GEM, der neuen grafischen Benutzeroberfläche des Atari: Drop-Down-Menüs, Window- und Symboltechnik und die Mausbedienung. Besonders interessant, für den fortgeschrittenen Anwender: der interne Aufbau von GEM, wie man diese Features für eigene Programme einsetzen kann, und die Verbindung zum TOS-Betriebssystem.

**Best.-Nr. MT 794, ISBN 3-89090-173-5**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

### Der Atari 520ST

**Juli 1985, 148 Seiten**

Ein Buch, das alle Informationen für den stolzen Besitzer eines gerade erworbenen Atari 520ST enthält: ausgiebige Diskussion des neuen Benutzerkonzepts, die spezifischen Merkmale der Gerätebedienung, das Betriebssystem TOS, Einsatzkonzepte des GEM, Beschreibung der CPU, Speicheraufteilung und Schnittstellen. Auch als Nachschlagewerk unbedingt zu empfehlen.

**Best.-Nr. MT 796, ISBN 3-89090-172-7**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### Spiel und Spaß mit dem Atari

**Mai 1984, 338 Seiten**

Einfache Programme in BASIC · wie man ein Spiel entwickelt · Lernstoff trainieren · Zahlen und Logik · Grafik · Farben · Töne und Musik · den Atari-Computer spielend erforschen.

**Best.-Nr. MT 672, ISBN 3-89090-002-X**
**(sFr. 38,60/öS 327,60)** **DM 42,—**

### Das Atari-Programmierhandbuch

**März 1985, 403 Seiten**

Alles was Sie über die Bedienung und die Programmierung Ihres Computers in BASIC wissen müssen · Speicherarten · grafische Symbole · spezielle Funktionen · Zubehörteile · Organisation eines Programms einschließlich Flußdiagramm und ihr Gebrauch · der 6502-Prozessor · mit vielen Programmierbeispielen für den ATARI 800 (400/600) · ein unentbehrliches Buch für die richtige Kaufentscheidung!

**Best.-Nr. MT 753, ISBN 3-89090-062-3**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

Die angegebenen Preise sind Ladenpreise

# Weitere Fachbücher aus unserem Verlagsprogramm

### Das Atari-Buch, Band 1
**Juli 1984, 158 Seiten**
Die grundlegenden Programmiermöglichkeiten für Ihren Atari · mit einem Spiel zum Eingewöhnen · Erstellung von Text und Grafik · Player Missiles · BASIC-Besonderheiten · ausführliche Assemblerlistings im Anhang · ein Einsteiger-Buch, vollgepackt mit Informationen.
**Best.-Nr. MT 703, ISBN 3-89090-039-9**
**(sFr. 29,50/öS 249,60)** **DM 32,—**
**Best.-Nr. MT 783 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***
*** Inkl. MwSt. Unverbindliche Preisempfehlung.**

### Das Atari-Buch, Band 2
**Oktober 1984, 197 Seiten**
Spezielle Programmiermöglichkeiten und Maschinenprogramme · BASIC-Kenntnisse und das Studium des Handbuchs (Das Atari-Buch, Bd. 1) werden vorausgesetzt · für alle, die die hervorragenden Grafik- und Soundeigenschaften des Atari ausnutzen wollen!
**Best.-Nr. MT 704, ISBN 3-89090-072-0**
**(sFr. 29,50/öS 249,60)** **DM 32,—**
**Best.-Nr. MT 775 (Beispiele auf Diskette)**
**(sFr. 38,—/öS 342,—)** **DM 38,—***
*** Inkl. MwSt. Unverbindliche Preisempfehlung.**

### Mein Atari-Computer
**1983, ca. 400 Seiten**
Alles über Aufbau und Bedienung des Atari-Computers · Programmieren in BASIC · Grafikfunktionen · Tonerzeugung · abgeleitete trigonometrische Funktionen · Tabellen zur Zahlenumwandlung · das Standardwerk für Anfänger.
**Best.-Nr. PW 554, ISBN 3-921803-18-7**
**(sFr. 54,30/öS 460,20)** **DM 59,—**

### Sprühende Ideen mit Atari-Grafik
**Januar 1985, ca. 250 Seiten**
Eine Einführung in die Grafikmöglichkeiten des Atari · die Gestaltgesetze von Objekten, Farbgebung, Bildschirmentwürfe · BASIC-Kenntnisse erforderlich.
**Best.-Nr. PW 716, ISBN 3-921803-39-X**
**(sFr. 45,10/öS 382,20)** **DM 49,—**

### Computer für Kinder - Ausgabe ATARI
**Februar 1985, 114 Seiten**
Ein BASIC-Programmierbuch ausdrücklich für Kinder geschrieben · mit einem besonderen Abschnitt für Lehrer und Eltern.
**Best.-Nr. PW 728, ISBN 3-921803-43-8**
**(sFr. 27,50/öS 232,40)** **DM 29,80**

### Lerne BASIC auf dem Atari
**November 1984, 321 Seiten**
Dieses Buch führt sowohl Kinder als auch Erwachsene in die Grundlagen des Atari-BASIC ein · Action-Spiele · Brettspiele · Wortspiele · Hinweise · Erklärungen · Übungen · amüsant und leicht verständlich präsentiert · zum Selbststudium geeignet.
**Best.-Nr. MT 692, ISBN 3-89090-007-0**
**(sFr. 35,—/öS 296,40)** **DM 38,—**

## SCHNEIDER-FAMILIE

### Der CPC 464 für Ein- und Umsteiger
**Februar 1985, 260 Seiten**
Eine praxisorientierte Spiel- und Arbeitshilfe für den Schneider CPC 464 · BASIC · Grafik · Sound · Tastaturanwendung · Kassettenrecordereinsatz · alle Befehle kompakt und systematisch dargestellt · modular aufgebaute Beispielprogramme auch zur Textverarbeitung und Datenverwaltung · der ideale Grundstock für Ihre CPC 464-Programmbibliothek!
**Best.-Nr. MT 801, ISBN 3-89090-090-9**
**(sFr. 42,30/öS 358,80)** **DM 46,—**

### CPC 464 - Programmieren in Maschinensprache
**Juli 1985, 276 Seiten**
Vom Speicheraufbau bis hin zum Z80-Befehlssatz wird der fortgeschrittene BASIC-Programmierer in das Innenleben seines Schneider-Computers eingeweiht. Wichtige ROM-Routinen und ausgewählte Werkzeuge wie Disassembler und Monitor werden als nützliche Utilities für die eigene Programmerstellung mitgeliefert. **Alle Beispiele auf Kassette erhältlich.**
**Best.-Nr. MT 829, ISBN 3-89090-166-2**
**(sFr. 42,30/öS 358,80)** **DM 46,—**
**Best.-Nr. MT 833 (Kassette)** **DM 19,90***
**(sFr. 19,90/öS 179,10)**
*** Inkl. MwSt. Unverbindliche Preisempfehlung**

### ROM-Listing CPC 464/664/6128
**November 1985, ca. 450 Seiten**
Ausführliche Hardware-Beschreibung: Prozessor Z80A, Videocontroller 6845 CRTC, Gate Array 20 RA 043, Sound Generator AY-3-8912, I/C-Baustein 8255 PIO, Expansion-Port. Die ROMs: Speicheraufteilung, Interrupt-Verwaltung, Datenformate, Erweiterungs- und Änderungsmöglichkeiten. Das ROM-Listing: Betriebssystem, BASIC-Interpreter.
**Best.-Nr. MT 711, ISBN 3-89090134-4**
**(sFr. 58,90/öS 499,20)** **DM 64,—**

Die angegebenen Preise sind Ladenpreise

# Weitere Fachbücher aus unserem Verlagsprogramm

### WordStar 3.0 mit MailMerge für den Schneider CPC
**September 1985, 435 Seiten**
WordStar ist ein umfangreiches und leistungsfähiges Textverarbeitungsprogramm und damit sicherlich zu Recht das meistverkaufte Programm seiner Art. Doch bedeutet dies nicht unbedingt, daß es auch einfach zu bedienen ist. Hier setzt dieses Buch an: Es macht in vorbildlicher Weise mit allen Möglichkeiten von WordStar und MailMerge vertraut und ist damit eine ideale Ergänzung zum Handbuch. Es versammelt alle Informationen für den effektiven Einsatz dieser Programme auf dem Schneider CPC.
**Best.-Nr. MT 779, ISBN 3-89090-180-8**
**(sFr.45,10/öS 382,20)** **DM 49,—**

### Schneider CPC Grafik-Programmierung
**November 1985, ca. 200 Seiten**
Dieses Buch wendet sich an die Schneider CPC-Besitzer, die alles über die Grafikfähigkeiten ihres Computers wissen wollen. Es bietet einen umfassenden Überblick über die verschiedenen Anwendungsbereiche der Grafikprogrammierung: zwei- und dreidimensionale Diagrammdarstellungen, Definition und Bewegung von Sprites, Entwurf von Titelgrafiken oder den Einsatz der Grafik bei der Unterstützung anderer Programme.
**Best.-Nr. MT 782, ISBN 3-89090-182-4**
**(sFr.42,30/öS 358,80)** **DM 46,—**

### dBASE II für den Schneider CPC
**September 1985, 280 Seiten**
Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integriertor Reportgonorator, dor im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt. Im Unterschied zu dem schon früher erschienenen Buch »Das Datenbanksystem dBASE II« (MT 740) geht dieses speziell auf die dBASE-Version für die Schneider-CPC-Computer mit dem Betriebssystem CP/M ein.
**Best.-Nr. MT 837, ISBN 3-89090-188-3**
**(sFr.45,10/öS 382,20)** **DM 49,—**

### CPC BASIC-Kurs
**November 1985, ca. 250 Seiten**
Dieses Buch soll den Einstieg in die Bedienung und Programmierung der Schneider-Familie (464, 664, 6128) erleichtern und richtet sich daher an alle Anwender, für die das Gebiet »Computer« noch Neuland ist. Ein Buch, das für jeden Schneider CPC-Besitzer interessant ist.
**Best.-Nr. MT 828, ISBN 3-89090-167-0**
**(sFr. 42,30/öS 358,80)** **DM 46,—**

### MULTIPLAN für den Schneider CPC
**September 1985, 226 Seiten**
Das vorliegende Buch soll eine praktische Einführung in den Umgang mit MULTIPLAN auf dem Schneider CPC geben. Anhand von praxisnahen Beispielen werden alle Befehle und Funktionen in der Reihenfolge bechrieben, die der Arbeit in der Praxis entspricht. Bereits nach Abschluß des ersten Kapitels werden Sie in der Lage sein, eigene kleine MULTIPLAN-Anwendungen zu realisieren. Ein Merkmal von MULTIPLAN ist, daß Kalkulationen schnell und einfach erstellt werden können.
**Best.-Nr. MT 835, ISBN 3-89090-186-7**
**(sFr.45,10/öS 382,20)** **DM 49,—**

## SINCLAIR

### ZX-Spectrum Hardware
**Januar 1985, 147 Seiten**
Dieses Buch vermittelt Ihnen ein fundiertes Basiswissen über Aufbau und Entwicklung eigener Hardware · Ausführliche Beschreibung der einzelnen ICs mit Abbildungen und 2-System-Schaltplänen · Anschluß einer PIO-Ansteuerung von Dezimalanzeigen · Leuchtdioden · Relais · DIL-Schalter · Eine akkugepufferte Hardwareuhr mit vierstelliger Anzeige · Soundgenerator mit drei Kanälen.
**Best.-Nr. MT 737, ISBN 3-89090-092-5**
**(sFr. 27,50/öS 232,40)** **DM 29,80**

## TI 99/4A

### 21 LISTige Programme für den TI-99/4A
**November 1984, 224 Seiten**
Umfangreiche Spiele aller Art für den TI-99/4A · nützliche Utilities · Adressenverwaltung · Vokabel-Programm · für manche Programme ist das Extended-BASIC-Modul, die Speichererweiterung (32 K), ein Disketten-Laufwerk oder Joysticks erforderlich!
**Best.-Nr. MT 754, ISBN 3-89090-065-8**
**(sFr. 23,—/öS 193,40)** **DM 24,80**

## PROGRAMMIERSPRACHEN

### BASIC-Grundkurs mit dem Commodore 64
**März 1985, 377 Seiten**
Ein praxisorientierter Leitfaden für die Programmierung in BASIC · die Besonderheiten des Commodore-BASIC · umfangreiche Befehlsübersicht · Einführung in die aktuelle Thematik der Datenkommunikation: Btx oder MailBox · das ideale Buch für Jungprogrammierer, die ihre Anfangsschwierigkeiten überwinden wollen!
**Best.-Nr. MT 633, ISBN 3-89090-045-3**
**(sFr.40,50/öS 343,20)** **DM 44,—**

Die angegebenen Preise sind Ladenpreise

**Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler**

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München

# Weitere Fachbücher aus unserem Verlagsprogramm

### Das Commodore 64-LOGO-Arbeitsbuch
**September 1984, 225 Seiten**
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
**Best.-Nr. MT 720, ISBN 3-89090-063-1**
**(sFr. 31,30/öS 265,20)** **DM 34,—**

### BASIC für Einsteiger
**Juni 1984, 239 Seiten**
Ein Arbeitsbuch für den absoluten Anfänger · BASIC-Anweisungen Schritt für Schritt erklärt und anhand von einfachen Beispielen erläutert · das beliebte Arbeitsmittel für Lehrkräfte und für den interessierten Computerfan.
**Best.-Nr. MT 680, ISBN 3-89090-024-0**
**(sFr. 29,50/öS 249,60)** **DM 32,—**

### MSX BASIC
**April 1985, 236 Seiten**
Alles über den neuen Heimcomputerstandard MSX: zusätzlich zum »normalen« BASIC können mit insgesamt mehr als 150 Befehlen und Funktionen Grafiken erstellt, Töne erzeugt, Melodien komponiert und ganze Spielhandlungen programmiert werden · 32 Sprites garantieren abwechslungsreiche Action-Spiele · die Hardware des MSX-Systems · nützliche Hinweise zur Dateibehandlung · das MSX-BASIC anhand der Entwicklung eines Spielszenarios mühelos lernen · drei vollständige Spiele: Der eisige Planet, Autorennen und Bilder entwerfen · mit ausführlicher Befehlsübersicht · für Anfänger!
**Best.-Nr. MT 805, ISBN 3-89090-107-7**
**(sFr. 40,50/öS 343,20)** **DM 44,—**
**Best.-Nr. MT 825 (Beispiele auf Kassette)**
**(sFr. 19,80/öS 178,20)** **DM 19,80***
*** Inkl. MwSt. Unverbindliche Preisempfehlung.**

### LOGO: Grafik, Sprache, Mathematik
**1984, 257 Seiten**
Eine Einführung in LOGO als Lehr- und Lernsprache unter besonderer Berücksichtigung des Apple-LOGO · Grafikprozeduren · Zeichenkettenmanipulationen · Probleme der Rekursivität · Sprachbildung und Sprachforschung · Grundlagen der Arithmetik · mit umfassendem Glossar.
**Best.-Nr. MT 648, ISBN 3-922120-60-1**
**(sFr. 38,60/öS 327,60)** **DM 42,—**

## ALLGEMEININTERESSE

### Microcomputer-Grundwissen
**1978, 304 Seiten**
Eine allgemeinverständliche Einführung in die Mikrocomputer-Technik · optimal als Einstieg für Elektronik-Laien.
**Best.-Nr. PW 156, ISBN 3-921803-02-0**
**(sFr. 33,10/öS 280,80)** **DM 36,—**

### Im Land der Abenteuer
**Juni 1984, 146 Seiten**
Verzweifelt? Steckengeblieben? Keine Ahnung, wie's mit Ihrem Lieblings-Adventure weitergeht? Keine Panik! – Die Rettung naht! Hier finden Sie die Lösung für 14 Top-Hits auf dem Adventure-Sektor, darunter auch die komplette Lösung zu »Time Zone«!
**Best.-Nr. MT 699, ISBN 3-89090-021-6**
**(sFr. 25,90/öS 218,40)** **DM 29,80**

### Lexikon der modernen Elektronik
**2. überarbeitete und erweiterte Auflage**
**Februar 1985, 340 Seiten**
3000 Fachbegriffe aus der allgemeinen Elektronik · Mikroelektronik · Mikro-Computer-Technik und Software · aus dem Englischen übersetzt und ausführlich erklärt · das ideale Nachschlagewerk für Beruf, Ausbildung und Hobby.
**Best.-Nr. MT 752, ISBN 3-89090-080-1**
**(sFr. 47,80/öS 405,60)** **DM 52,—**

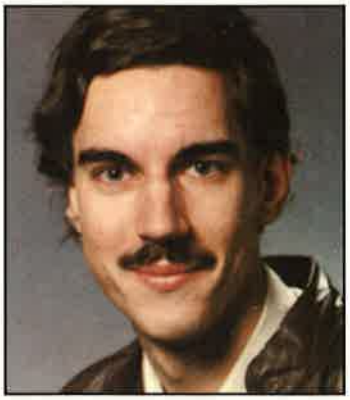
### Btx professionell eingesetzt
**August 1984, 287 Seiten**
Alles über den effizienten Einsatz von Bildschirmtext · völlig neue Möglichkeiten in Marketing und Werbung, bei Dienstleistungen, bei der Informationsdistribution und Schulung · Btx professionell angewandt erhöht die Produktivität und Kommunikationsqualität, senkt Kosten und steigert den Gewinn · für Computer-Profis.
**Best.-Nr. MT 530, ISBN 3-922120-52-0**
**(sFr. 62,60/öS 530,40)** **DM 68,—**

### Drucker-Handbuch
**Januar 1985, 188 Seiten**
Richtig kaufen — problemlos anschließen — optimal nutzen! Ein informativer Leitfaden für alle, die vor dem Kauf eines Druckers stehen · Arbeitsweise der verschiedenen Druckertypen · Druckeranschluß an verschiedene Rechnertypen/Schnittstellen · Druckerzubehör · geeignet auch als Nachschlagewerk!
**Best.-Nr. MT 742, ISBN 3-89090-077-1**
**(sFr. 35,—/öS 296,40)** **DM 38,—**

Die angegebenen Preise sind Ladenpreise

**Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler**

Markt & Technik Verlag AG Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München



CARSTEN STRAUSH
geboren am 12. 2. 1963, kam über die ersten Taschenrechner und die legendären 8080-Experimentier-Kits zur Computerei. Hier arbeitete er auf vielen der bekannteren Mikrocomputertypen wie TRS-80, Sinclair, Commodore und gelangte schließlich auch zum CPC 464. Neben Veröffentlichungen und der Softwareentwicklung beschäftigt er sich auch mit dem Software-Marketing für alle gebräuchlichen Homecomputer und PCs.

# Schneider CPC Grafik-Programmierung

Dieses Buch wendet sich an all jene, die mehr über den Schneider CPC und seine Fähigkeiten im Bereich der Grafik wissen wollen. Es bietet einen guten Überblick über die verschiedenen Anwendungsbereiche der Grafikprogrammierung wie zwei- und dreidimensionale Diagrammdarstellungen, die Definition und Bewegung von Sprites, den Entwurf von Titelgrafiken oder den Einsatz der Grafik bei der Unterstützung anderer Programme.

Ausführlich werden alle interessanten Bereiche mit den für den Anwender besonders wichtigen Informationen behandelt, so zum Beispiel der Aufbau des Grafikspeichers, Hardwaregrundlagen oder die Benutzung von Systemroutinen unter Direktzugriff auf den Bildschirmspeicher. Auch die Zusammenarbeit mit den externen Geräten, wie der Abspeicherung von Bildschirmen auf Diskette und Kassette oder der Druckerausgabe von Grafiken, kommt dabei zur Sprache.

Die Darstellung erfolgt jedoch nicht als graue Theorie, sondern mit einer Fülle von für den Anwender sofort nachvollziehbaren Beispielen. Zusätzliche Würze in der Darstellung bringen eine ganze Reihe von special effects, Tips und Tricks.

Außer einem Schneider-Computer, wahlweise CPC 464 oder 664 mit oder ohne Farbmonitor und einigen grundlegenden Kenntnissen im Bereich der BASIC-Programmierung sind dabei keine Voraussetzungen nötig, um schon bald eigene Programme grafisch aufzuwerten.

Als Unterstützung dazu liefert das Buch mehrere Grafikprogramme und eine BASIC-Befehlserweiterung, die das Entwickeln von Zeichnungen und Bildern auf dem Bildschirm des Computers erheblich erleichtern.

Zu dem Buch gibt es eine Kassette mit allen beschriebenen Programmen, die unter der Bestell-Nr. MT 873 beim Verlag erhältlich ist.

ISB N 3-89090-182-4

4 001057 901827

Markt&Technik
Verlag Aktiengesellschaft

DM 46,-
sFr. 42,30
öS 358,80