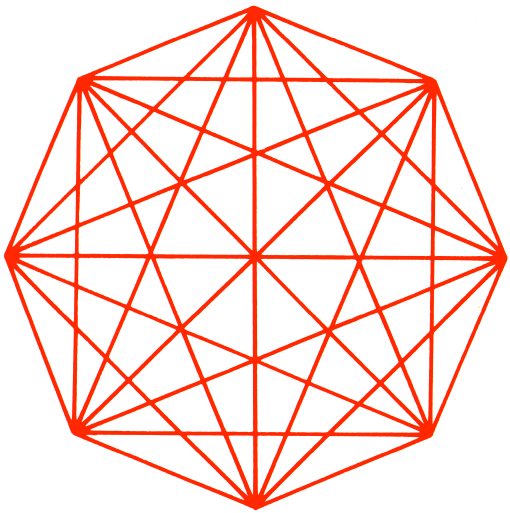
Plenge

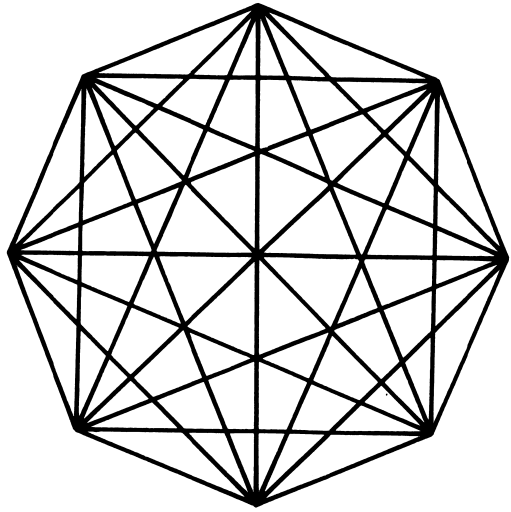
DAS GRAFIKBUCH ZUM COMMODORE 64



EIN DATA BECKER BUCH

Plenge

DAS GRAFIKBUCH ZUM COMMODORE 64



EIN DATA BECKER BUCH

ISBN 3-89011-009-6

1. Auflage

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Vorwort

Grafik ist nicht nur eine der Hauptstärken des COMMODORE 64; COMMODORE hat diese Stärke auch sehr sorgfältig versteckt. Der Basic - Anfänger kennt Grafik nur bewundernd von den vielen fertigen Programmen und Aktionspielen, die natürlich über entsprechende Routinen - meist in Maschinensprache - die grafischen Fähigkeiten des COMMODORE 64 voll ausnutzen. Dieses Buch soll nun jedem COMMODORE 64 - Anwender die Möglichkeit geben, auch in eigenen Programmen die Vielzahl grafischer Möglichkeiten seines Computers zu nutzen.

Als Autor konnten wir Herrn Axel Plenge gewinnen, der den COMMODORE 64,besonders von der Grafikseite her, wie kaum ein anderer kennt und dies mit der beliebten Supergraphik ja bereits eindeutig unter Beweis gestellt hat. Ihm machte das Schreiben dieses Buches so viel Spaß, und er entdeckte dabei so viele interessante Sachen, daß das Grafikbuch gleich fast 50 Seiten umfangreicher wurde als geplant. Für Sie als Leser hat dies viele Vorteile, und wir dürfen Ihnen viel Spaß beim Ausprobieren der vielen Anregungen und Programme von Axel Plenge wünschen.

Dr. Achim Becker

Inhaltsverzeichnis

1. Kapitel
Einleitung

"64 K RAM, 8 unabhängige Sprites, frei auf dem Bildschirm bewegbar, hochauflösende Graphik mit 320x200 Punkten, Multicolorgraphik (160x200 Punkte), 16 Farben, 40 Zeichen, 25 Zeilen, veränderbarer Zeichensatz, sensationelle Interruptmöglichkeiten, ..., ein Computer, den jeder kennenlernen, nein, besitzen muß! ..."
So oder ähnlich wird -wohl auch zurecht- unser guter alter Commodore 64 angepriesen. Kaum jemand, der sich auch nur ein wenig mit Computern auskennt, wird an solchen Anzeigen vorbeischauen, denn Ihr 64er kann wirklich viel.
Doch bald nach dem Kauf und dem hoffnungsvollen Studium der Betriebsanleitung aber bekommt man ein merkwürdiges Gefühl in der Magengrube: Kein Wort von der hochauflösenden Graphik, geschweige denn von solchen Vorzügen wie Zeichensatzveränderung oder Interuptfähigkeiten. Selbst das für die Maßstäbe des Handbuches relativ ausführliche Kapitel über die Sprites und ihre Programmierung verschleiert die wahren Möglichkeiten des Gerätes.
Doch wir sollten nicht zu streng mit dieser Kurzanleitung zu Gerichte ziehen. Um auch nur annähernd alle Vorzüge unseres Rechners zu beschreiben und gar ihre Anwendung zu demonstrieren, bedarf es einer etwas umfangreicheren Lehrbuchkonzeption.
An dieser Stelle greift nun das vorliegende Graphikbuch zum 64er an. Dieses Buch soll Ihnen umfassende Kenntnisse über das graphische Innenleben eines Computers vermitteln, der nicht umsonst zum Computer des Jahres 1983 gewählt wurde.
Durch das gesamte Buch zieht sich eine typische Dreiteilung. Dabei werden alle Fähigkeiten des Rechners unter drei verschiedenen Gesichtspunkten besprochen und von allen Seiten beleuchtet. Diese drei Sinnabschnitte behandeln die folgenden Themen:

- Hardwaregrundlagen
- grundsätzliche Programmierung
- Anwendungen

Im ersten großen Abschnitt (Kapitel 3) erfahren Sie alles (wirklich alles), was es über die Bildschirmsteuerung durch den VIC (Videocontroller), dem Organisator des gesamten Bildgeschehens, zu erfahren gibt. Hier sehen Sie, welche Bedeutung beispielsweise den vielen Registern dieses "Managers" zukommt, was Sprites sind und wie sie organisiert sind, was man tun muß, um Graphik einzuschalten, zu erstellen und zu bedienen. Ihnen wird endlich klar, was es mit der bisher ziemlich undurchsichtig erscheinenden Graphik-, Zeichensatz- oder Bildschirmspeicher - Verschiebung auf sich hat, allgemein wie der gesamte Speicheraufbau des 64ers beschaffen ist und welche Möglichkeiten sich ergeben...

Doch damit kann man gemeinhin noch nicht viel anfangen, wenn einem das 'Know how' fehlt, das Wissen um die Programmierung der vielen verschiedenen Bildschirmfunktionen. So erfahren Sie im 4. Kapitel, wie Sie die Graphik starten und wie Sie die ersten einfachen Figuren wie Punkte, Linien, Kreise oder Ellipsen auf der Graphikanzeige erscheinen lassen. Sprites und ganze Zeichensätze werden erstellt bzw. verändert. Hierzu werden Ihnen zwei sehr komfortable Editorprogramme zur Verfügung gestellt, mit denen Sie ohne viele Mühe, bequem und handlich diese Dinge erledigen können. Sprites werden auf dem Bildschirm bewegt und auf Kollisionen etc. überprüft. Weitere Kapitel führen Sie in das Laden, Speichern und die Erstellung von Hardcopys der Graphik ein und erläutern Ihnen die Interrupttechniken. Schließlich, als kleiner Höhepunkt, ermöglicht ein Graphik-Paket den schnellen und problemlosen Umgang mit diesem Schatz Ihres Computers.

Wo das 4. Kapitel die Programierung der einzelnen Optionen des Rechners getrennt behandelt, werden im 5. großen Sinnabschnitt schließlich Beispiele der Anwendung des Gelernten unter Kombination aller seiner graphischen Fähigkeiten Thema unser Betrachtungen. Hier erfahren Sie, wie Sie das Zusammenspiel dieser Dinge organisieren und welche Möglichkeiten sich ergeben. Kurz, hier entdecken Sie den tatsächlichen Nutzen

des 64ers; auf diesen Abschnitt können Sie verweisen, wenn Sie jemand fragt: "Ein Computer? Was soll ich damit?"

Wir hoffen, damit jeden 64er - Anwender zu einem absoluten "Graphik-Freak" zu machen, einem Fachmann in allen Fragen, der zu Rate gezogen wird, immer dann, wenn andere nicht mehr weiter wissen. Und sollte doch einmal eine Gedächtnislücke auftreten, so schlägt er kurz einmal im Anhang nach und schon weiß er bescheid. Umfangreiche Hinweise auf weiterführende Literatur runden das Buch ab.

Noch etwas in 'eigener' Sache: Wir haben eingesehen, daß nicht jeder die Zeit und Lust aufbringt, jedes der vielen Programme -vor allem der langen- einzutippen. Da aber ein wesentlicher Informationsteil ohne diese Routinen verloren ginge, haben wir uns entschlossen, Ihnen eine Diskette zur Verfügung zu stellen, auf der alle Programme dieses Buches plus weiterer nützlicher Utilities abrufbereit und lauffertig gespeichert sind. Ich versichere Ihnen, die Anschaffung wird sich lohnen.

2. Kapitel
Bits and Bytes

Um die vielen verschiedenen Faktoren festzulegen, mit denen Sie die zahllosen Möglichkeiten der Graphikvariation bestimmen können, werkeln Sie direkt in der Speicherstruktur Ihres Gerätes bzw. mit den unterschiedlichen Registern (s.u.) der integrierten Schaltkreise. Bekommen Sie keinen Schreck! Wir wollen Sie nicht etwa mit elektronischen Einzelheiten bombardieren. Zum Verständnis dieser Dinge bedarf es nur etwas Mathematik, die relativ einfach zu durchschauen ist und oft schon zum Stoff von 5- oder 6-Klässlern gehört. Es geht unter anderem um die Darstellung von Zahlen im sogenannten Binär- oder Dualsystem.

Für einen gewöhnlichen Computer, der ja bekanntlich aus einer Unzahl von elektronischen Leitungen und Bausteinen besteht, gibt es lediglich zwei Zustände, aus denen seine ganze kleine Welt besteht: Strom ein --- Strom aus. Da wir Menschen nun aber von ihm eine ganze Menge mehr verlangen, mußten wir uns etwas einfallen lassen, um mit diesem Mangel zu leben. Wollen wir zum Beispiel eine Zahl im Computer darstellen, so kommen wir nicht mit diesen beiden Zuständen aus. Wir könnten zwar dem Zustand "Strom aus" die Zahl 0 und "Strom ein" die Zahl 1 zuordnen, werden damit aber wohl nicht weit kommen, da wir außer 0 und 1 noch unendlich viele andere Zahlen darstellen wollen.

2.1 Dezimalsystem

Im alltäglichen Leben stehen uns 10 Ziffern (0-9) zur Verfügung (deshalb der Name "Dezimal"system von decem lat. - zehn), mit denen wir durch einen kleinen Trick sämtliche weiteren Zahlen darstellen können: Wir reihen einfach mehrere Ziffern zu einer großen Zahl zusammen. Wollen wir beispielsweise bis 1000 zählen, so sind wir bereits bei der Ziffer 9 am Ende unseres Ziffernvorrates. Jedem ist bekannt, daß wir danach einfach wieder von vorne (bei 0) anfangen zu zählen,

wobei wir jedoch als Kennzeichen, daß wir bereits einmal bei 9 angelangt sind eine 1 vor die laufende Ziffer schreiben. Die nächsten Zahlen nach 9 lauten bekanntlich 10 (eins null), 11 (eins eins), 12 (eins zwei) usw. Bei 19 sehen wir uns dem gleichen Problem gegenübergestellt. Doch wieder beginnen wir bei 0 und erhöhen lediglich die vorgestellte Ziffer um eins. Das Ergebnis: 20 (zwei null). Sind wir bei dieser ersten Ziffer ebenfalls bei 9 angekommen (99), so beginnen wir hier gleichfalls einfach wieder bei 0 und stellen davor die Zahl 1 (100). Nach diesem Prinzip werden schließlich sämtliche ganzen Zahlen dargestellt. Dabei werden die einzelnen Ziffern einer Zahl Stellen genannt und diese wiederum als Einer, Zehner, Hunderter, Tausender, ... bezeichnet. Eine Zahl z, deren Ziffern (d_1, d_2,...) bekannt sind, läßt sich somit wie folgt errechnen:

$$z = d_0 * 10^0 + d_1 * 10^1 + d_2 * 10^2 + \ldots$$

wobei d jeweils die Einer-, Zehner-, Hunderter-, ... -stellen bezeichnet (eine kleine Anmerkung: eine Zahl hoch 0 ergibt stets 1, also 10 hoch 0 ist gleich 1, genauso wie z.B. 5 hoch 0 gleich eins ist --- Ausnahme: 0 hoch 0 ist nicht definiert). Eine andere Darstellung ist die folgende (hier an dem Beispiel der willkürlich gewählten Zahl 3124):

10^4	10^3	10^2	10^1	10^0
0	3	1	2	4

Die obere Zeile nennt dabei den Wert der einzelnen Stellen, die untere Zeile den Faktor d.

2.2 Dualsystem

Wir besitzen, wie gesagt, zehn Ziffern, um unsere Zahlen einigermaßen übersichtlich darzustellen. Unser armer Computer muß sich jedoch mit 2 Ziffern begnügen (0 und 1), wie oben dargelegt. Wie aber zählt er dann bis 1000? Ganz einfach: genauso wie wir! Also:

0, 1

damit ist sein Ziffernschatz "verbraucht" und er startet

wieder bei 0, setzt aber gleichfalls als Kennzeichen eine 1 davor und erhält:

10

Es geht weiter mit:

11,100,101,110,111,1000,1001,1010,1011,1100,1101,....

Wie Sie sehen, ist das sogenannte Binärsystem völlig analog zu dem uns vertrauten Dezimalsystem aufgebaut. Entsprechend läßt sich der Wert einer Dualzahl durch die folgende Formel errechnen:

$$z = b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2 + \ldots$$

wobei die Parameter $b_{0,1,2,\ldots}$ die einzelnen Ziffern, angefangen von der ersten (Einerstelle) bis zur höchsten Dualstelle, darstellen. Kennen Sie also die Ziffern einer Dualzahl, so ist es Ihnen mit Hilfe dieser Formel möglich, sie in eine Dezimalzahl umzurechnen. Wie oben kann dies ebenfalls durch die folgende Tabelle erreicht werden (hier an dem Beispiel der Zahl 10110100):

$2^7=$	$2^6=$	$2^5=$	$2^4=$	$2^3=$	$2^2=$	$2^1=$	$2^0=$	
128	64	32	16	8	4	2	1	
1	0	1	1	0	1	0	0	
128		+32	+16		+4			= 180

Eine solche Stelle nennt man nun in der Computerfachsprache ein Bit, d.h. eine Informationseinheit. Unter Informationseinheit (oder Bit) versteht man also die Möglichkeit zweier Zustände (ja oder nein bzw. 1 oder 0). In Ihrem Rechner sind nun 8 solcher Bits (oder Dualstellen) zu einer Einheit zusammengefaßt, dem sogenannten Byte. Mit diesen 8 Bits lassen sich also Zahlen von 0-255 darstellen. Mit zwei Byte (=16 Bits) dagegen schon von 0-65535. Wollen wir ein Byte also in eine Dezimalzahl umrechnen (was notwendig wird, wenn wir von Basic aus Änderungen direkt an bestimmten Speicherstellen vornehmen wollen), so verwenden wir selbstverständlich wieder unsere Tabelle (oder die Formel).

Wollen wir dagegen umgekehrt eine Dezimalzahl in eine Dualzahl umrechnen, so gehen wir wie folgt vor (am Beispiel der Zahl 180):

```
180 : 128 = 1 Rest 52
 52 :  64 = 0 Rest 52
 52 :  32 = 1 Rest 20
 20 :  16 = 1 Rest  4
  4 :   8 = 0 Rest  4
  4 :   4 = 1 Rest  0
  0 :   2 = 0 Rest  0
  0 :   1 = 0 Rest  0
```

$180_{(d)} = 10110100_{(b)}$

Hier wurde also die umzurechnende Zahl 180 nacheinander durch die Potenzwerte aus der Tabelle geteilt. Das Ergebnis stellt dabei jeweils eine Ziffer der gewünschten Dualzahl dar, der Rest dieser Division wird für die übrigen Rechnungen weiterverwendet.
Eine zweite Methode ist die folgende. Sie eignet sich besonders für Computerprogramme, da sie rekursiv und damit besonders einfach und schnell ist:

```
180 : 2 = 90 Rest 0
 90 : 2 = 45 Rest 0
 45 : 2 = 22 Rest 1
 22 : 2 = 11 Rest 0
 11 : 2 =  5 Rest 1
  5 : 2 =  2 Rest 1
  2 : 2 =  1 Rest 0
  1 : 2 =  0 Rest 1
```

Hier wird, wie Sie sehen, ständig durch 2 geteilt. Der jeweilige Rest stellt dabei startend von der Einerstelle eine Dualziffer dar. Das Ergebnis der Divison wird weiter für die folgenden Rechnungen verwandt, solange, bis es 0 wird.
Um zu kennzeichnen, daß es sich bei einer bestimmten Zahl um eine Dualzahl handelt, setzen wir vereinbarungsgemäß ein Prozentzeichen (%) vor die jeweilige Zahl. Dies ist üblich und wird auch im Weiteren verwendet.
Wie Sie in den ganzen Ausführungen sehen, werden unsere Zahlen auf die Dauer umständlich lang und unübersichtlich. Der Computer kann damit sehr viel besser umgehen. Wir aber sehnen uns nach unserer guten alten Dezimalschreibweise. Doch hier ist die Umrechnung stets etwas schwierig, wie Sie sahen.

Aus diesem Grunde hat man sich etwas anderes ausgedacht, das

2.3 Hexadezimalsystem

Das Hexadezimal- oder 16er-System bietet hier einige Vorteile, die Sie im folgenden kennenlernen werden. Es besitzt 16 verschiedene Ziffern. Da wir von unserem Dezimalsystem her jedoch nur 10 Ziffern kennen, müssen wir sechs weitere erfinden. Alle verfügbaren Ziffern lauten:

Dez	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	$0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F

Die fehlenden Ziffern wurden also durch Buchstaben symbolisiert. Um eine Hexadezimalzahl als solche zu kennzeichen, ist es üblich, ein $ (Dollarzeichen) vor diese Zahl zu setzen. Wie Sie sich denken können, läuft die Berechnung oder Umrechnung zwischen den einzelnen Systemen völlig analog. Mithilfe der folgenden Tabelle rechnen Sie eine Hexadezimalzahl in eine 10er-Zahl um (am Beispiel $FE2A):

16^3 = 4096	16^2 = 256	16^1 = 16	16^0 = 1	
F	E	2	A	
15*4096	+14*256	+2*16	+10*1	= 65066

Diese Verwandlung ist ebenfalls rekursiv möglich.
Die umgekehrte Rechnung lautet dagegen:

```
65066 : 4096 = 15 Rest 3626
 3626 :  256 = 14 Rest   42
   42 :   16 =  2 Rest   10
   10 :    1 = 10 Rest    0
   65066(d) = FE2A(h)
```

Sie sehen, diese Umrechnung folgt den selben Gesetzmäßigkeiten, wie unter Dualzahlen beschrieben. Die dort erwähnte rekursive Methode ist hier selbstverständlich ebenfalls anwendbar.

Welchen Vorteil bietet nun das beschriebene Hexadezimalsystem?
Zunächst einmal ist es durch dieses System möglich, Zahlen sehr kurz und übersichtlich anzugeben. Dies allein ist jedoch noch nicht ausschlaggebend. Wichtig ist, daß sich die Umrechnung von Hexadezimal- in Dualzahlen äußerst einfach gestaltet. Jeweils 4 Binärziffern nämlich ergeben stets eine Hexadezimalziffer. Ein Byte kann also durch 2 Hexziffern festgelegt werden. Z.B.:

```
% 1101 0110
$   D    6
```

Sie sehen, wie einfach und übersichtlich so eine sonst umständlich lange Dualzahl beschrieben werden kann. Im Anhang finden Sie zu Ihrer Unterstützung Dez-Hex-Dual - Conversionstabellen.

2.4 Logische Operationen

Um später die Inhalte bestimmter Bytes oder Speicherstellen zu verändern oder zu manipulieren, sind verschiedene logische Operationen nützlich, die auch vom Commodore - Basic aus angewählt werden können und von denen hier zwei kurz beschrieben werden sollen: AND und OR. Bei beiden sind stets zwei miteinander zu verknüpfende Dualzahlen notwendig.

a) AND:
Nehmen wir einmal an, Sie wollen ein bestimmtes Bit in einem Byte nur dann erhalten, wenn es gleichzeitig in einem anderen Byte steht. In diesem Falle verwenden Sie die AND-Verknüpfung. Sie wird Bit für Bit vorgenommen und kann durch die folgende Verknüpfungstabelle beschrieben werden:

	0	1
0	0	0
1	0	1

Sie sehen also, daß das Ergebnis nur dann gleich 1 ist,

wenn beide verknüpften Bits ebenfalls gleich 1 sind, andernfalls bleibt alles 0. Wollen wir nun zwei vollständige Bytes (bestehend aus je 8 Bits) miteinander "ANDieren", so sieht das Ganze so aus:

```
  10110010
AND 01100111
  00100010
```

Hier wurde also jedes einzelne Bit des ersten mit dem korrespondierenden Bit des zweiten Bytes durch AND miteinander verknüpft.
Diese Operation wird neben der als nächstes beschriebenen ständig verwendet, um z.B. lediglich ein Bit eines Bytes zu verändern, während die anderen erhalten bleiben.

b) OR:

Die OR-Verknüpfung kann -wie auch AND- durch eine sogenannte Verknüpfungstabelle dargestellt werden:

	0	1
0	0	1
1	1	1

Wie ersichtlich wird das Ergebis 1 schon dann, wenn bereits eines der beiden zu verknüpfenden Bits gleich 1 ist. Ein Byte kann somit etwa so geORt werden:

```
  10110010
OR 01100111
  11110111
```

3. Kapitel
Hardwaregrundlagen

Ihr Commodore 64 besitzt eine ungeheure Vielzahl an Möglichkeiten, Graphiken zu erstellen und zu kontrollieren. Da er jedoch keinerlei Befehle zur Ausnutzung dieser Dinge in seinem Basic zur Verfügung hat, ja, die Möglichkeit hochauflösender Graphik nicht einmal in seinem Handbuch erwähnt wird, muß alles, was die Graphik betrifft, selbst entweder direkt in Maschinensprache oder von Basic aus programmiert werden, es sei denn, Sie besitzen eine entsprechende Graphikerweiterung, die Ihnen diese Befehle an die Hand gibt. Doch keine noch so gute Erweiterung kann auf alle Fähigkeiten des CBM 64 eingehen. Deshalb bedarf es einer guten Kenntnis der in Ihrem Rechner verwirklichten Graphikorganisation, um sie alle zu nutzen. Außerdem ist es äußerst interessant, festzustellen, wie perfekt die einzelnen Dinge zusammenspielen, oder wo es manchmal ärgerliche Haken oder Ösen, durch die man hindurchschlüpfen kann, gibt.
Bevor wir uns also mit der Programmierung der Graphik beschäftigen und den vielen Anwendungsmöglichkeiten, die den Nutzen dieser Rechnereigenschaft (auch oder gerade bei der Verwendung einer entsprechenden Erweiterung, die auf jeden Fall zu empfehlen ist) erst richtig zum Ausdruck bringt, wollen wir daher eine detailierte Kenntnis über die einzelnen Positionen der Graphik und Ihre Organisation vermitteln. Zugegeben, es ist nicht ganz einfach, aber wir wollen versuchen es Ihnen so nahe wie möglich zu bringen.

3.1 Die Register des VIC

Zunächst einmal werden Ihnen die 47 Register des VIC, also des Video Interface Chips, dem zentralen Prozessor, der u.a. die gesamte Bildschirmausgabe steuert, kurz und übersichtlich beschrieben. Mithilfe dieser Register werden (fast) alle Funktionen bezüglich Graphik, Text usw. gesteuert. Sie besitzen daher fundamentalsten Wert für Ihr Verständnis über

Graphik und Bildschirmmanipulationen und sollten schon (bis auf einige Kleinigkeiten) weitestgehendst verstanden werden. Diese Übersicht wird dann in den folgenden Abschnitten und Kapiteln vertieft und näher erläutert. Wenn Sie also nicht sofort alles verstehen, so ist das nur verständlich und praktisch zwingend. Wir werden jedoch stets mit diesen Dingen arbeiten, weswegen Sie im Anhang des vorliegenden Buches noch einmal eine kurze Übersicht über die Registerbelegung jenes integrierten Schaltkreises finden. Grundsätzlich ist diese folgende Beschreibung als Nachschlagewerk gedacht und auch verfaßt.

Nun aber zu den Registern:

VIC-Register --- Basisadresse: 53248 ($D000)

Reg. Dez	Hex	Kurzbeschreibung	Startbelegung Dez	Dual
00	$00	x-Koordinate Sprite 0	00	%0000 0000

Dieses Register beinhaltet die 8 unteren Bits (0-255) der x-Koordinate des ersten Sprites (Sprite 0). Das oberste, 9. Bit (auch MSB = most significant bit genannt) wird dagegen in Register 16 gespeichert. Dies ist notwendig, da die x-Koordinate größer als 255 werden kann.

01	$01	y-Koordinate Sprite 0	00	%0000 0000

Wie oben, nur ohne Übertrag.

02-15 $02-0F Koordinaten der übrigen 7 Sprites

(Aufbau wie oben). Sprite 1: Reg. 2/3; Sprite 2: Reg. 4/5 usw.

16	$10	MSB der x-Koordinaten	00	%0000 0000

Hier befinden sich die Überläufe (9. Bit) aus den x-Koordinaten Registern. Jedem Bit ist ein Sprite zugeordnet. Bit 0 für Sprite 0 / Bit 1 für Sprite 1 usw.

17	$11	Steuerregister 1	155	%1001 1011

Bit 0-2: Bildschirmverschiebung oben/unten
Bit 3: =0: 24 Zeilen / =1: 25 Zeilen
Bit 4: =0: Bildschirm aus / =1: ein
Ist der Bildschirm aus, so wird die CPU nicht mehr vom VIC unterbrochen (was z.B. bei der Generierung von Sprites für bis zu 40 Millisekunden geschehen kann) und Ihr Programm läuft evt. etwas schneller und gleichmäßiger.
Bit 5: =1: Standart Bitmap Mode
Bit 6: =1: Extended Colour Mode
Bit 7: Übertrag aus Register 18 ($12)

18	$12	Rasterzeilen-IRQ	55	%0011 0111

Wird dieses Register beschrieben, so geben Sie hier

die Bildschirmrasterzeile an, bei deren Aufbau durch den VIC ein IRQ ausgelöst werden kann. Wird es dagegen gelesen, so steht in ihm stets die aktuelle Rasterzeile, die der VIC gerade aufbaut. Der Übertrag dieses Registers steht in Register 17.

19 $13 Lightpen-x-koordinate 00 %0000 0000

x-Koordinate (Rasterkoordinate) der Bildschirmposition, die gerade aufgebaut wurde, als ein Signal vom Lightpen kam (Lightpenleitung = 0).

20 $14 Lightpen-y-koordinate 00 %0000 0000

Wie Register 19, jedoch y-Koordinate.

21 $15 Sprite ein/aus 00 %0000 0000

Hier werden die einzelnen Sprites ein- oder ausgeschaltet. Jedem Bit ist ein Sprite zugeordnet (wie in Register 16).

22 $16 Steuerregister 2 08 %0000 1000

Bit 0-2: Bildschirmverschiebung links/rechts
Bit 3: =0: 38 Zeilen / =1: 40 Zeilen pro Zeile
Bit 4: =1: Multicolor Modus
Bit 5-7: unbenutzt

23 $17 Spritevergrößerung y-Richtung 00 %0000 0000

Jedem Sprite ist ein Bit zugeordnet. Bit=1: Sprite wird doppelt so breit. (s. auch Reg. 29)

24 $18 VIC-Basisadressen 20 %0001 0100

Hier werden einige der oberen Bits der Startadressen von Videoram und Zeichensatzspeicher abgelegt. Durch Änderung ist eine Verschiebung dieser Bereiche möglich (s. auch Reg. 0 der CIA 2). Die Belegung lautet:
Bit 0 : unbenutzt
Bit 1-3: Adressbits 11-13 des Zeichensatzes (*2048)
Bit 4-7: Adressbits 10-13 des Videorams (*1024)
Die im CBM 64-Handbuch auf Seite 158 angegebene Belegung dieses Registers ist falsch!

25 $19 Interrupt Request Reg. (IRR) 15 %0000 1111

Hier kann die Ursache für einen IRQ festgestellt werden:

Bit 0 = 1: Ursache: Rasterzeilen-IRQ (Reg. 18)
Bit 1 = 1: Ursache: Sprite-Hintergr. Koll. (Reg. 31)
Bit 2 = 1: Ursache: Sprite-Sprite Koll. (Reg. 30)
Bit 3 = 1: Ursache: Lightpen sendet Impuls
Bit 4-6 : unbenutzt
Bit 7 = 1: mindestens eines der ersten 4 Bits ist 1.

Dieses Register muß (soweit es verwendet wird) nach dem Ereignis wieder gelöscht werden. Dies geschieht, indem man den gerade aus gelesenen Wert wieder hineinschreibt.

26 $1A Interrupt Mask Register (IMR) 00 %0000 0000

Hier wird vom Programmierer ausgewählt, durch welches Ereignis ein IRQ ausgelöst werden soll. Die Belegung entspricht der des Registers 25. Ist ein Bit sowohl in diesem, wie auch gleichzeitig im Reg. 25 gesetzt, so wird ein IRQ ausgelöst, d.h. Alle im Reg. 26 gesetzten Bits ermöglichen die Auslösung eines IRQ durch Reg. 25.

27 $1B Priorität 00 %0000 0000

Jedem Sprite ist ein Bit zugeordnet. Bit=1: Hintergrundzeichen hat Priorität vor dem Sprite / Bit=0: Sprite vor Hintergrundzeichen

28 $1C Multicolor-Sprites 00 %0000 0000

Jedem Sprite ist ein Bit zugeordnet. Bit=1: Sprite wird im Multicolor Modus gezeichnet.

29 $1D Spritevergrößerung x-Richtung 00 %0000 0000

Jedem Sprite ist ein Bit zugeordnet. Bit=1: Sprite wird in x-Richtung vergrößert (doppelt so hoch).

30 $1E Sprite-Sprite-Kollision 00 %0000 0000

Jedem Sprite ist ein Bit zugeordnet. Berührt ein Sprite ein anderes, so werden die beiden entsprechenden Bits dieser Sprites gesetzt. Gleichzeitig wird Bit 2 des Registers 25 (IRR) =1. Nach dem

Ereignis muß dieses Register gelöscht werden, da sich die Bits nicht selbsttätig zurücksetzen.

31 $1F Sprite-Hintergrund-Kollision 00 %0000 0000
Wie Register 30. Hier jedoch wird die Berührung eines Sprites mit einem Hintergrundzeichen (gesetzter Punkt) registriert.

32 $20 Rahmenfarbe 14 %0000 1110

33 $21 Hintergrundfarbe 0 06 %0000 0110

34-36 $22-24 Hintergrundfarben 1-3 01 02 03

37/38 $25/26 Sprite Multicolor 0/1 04 00

39-46 $27-2E Farbe Sprite 0-7 01 02 03 04 05 06 07

Nach der dezimalen und hexadezimalen Nummernangabe der einzelnen Register, die bei der Ansteuerung stets zu der Basisadresse hinzuaddiert werden muß, folgt, wie Sie sehen, der Registername und die Startbelegung. Unter Startbelegung verstehen wir dabei den Wert, der nach dem Einschalten des Computers in das jeweilige Register als Initialisierungswert eingeschrieben wird. Dieser Wert wird in unserer Übersicht Dezimal (z.B. für Basic-Programmierer) und Dual angegeben.

Zusätzlich zu diesen Registern des VIC gibt es natürlich noch einige andere Speicherstellen, die besonders interessant im Zusammenhang mit der Graphikprogrammierung sind. Der Vollständigkeit halber seien sie hier hinten angefügt:

a) Spritedefinitionspointer:
Um dem VIC mitzuteilen, wo in seinem Adressierungsbereich er die 63 Byte lange Definition eines bestimmten Sprites findet, müssen die 8 letzten Bytes des Videoram mit einem Pointer belegt werden (im Originalzustand: Speicherstellen 2040 - 2047 bzw. $07F8 - $07FF). Multipliziert man ihn mit 64, so gibt er die Adresse des Definitionsbeginns eines Sprites relativ zu dem Adressbereich des VIC an

(einzustellen mit Register 0, Bit 1 und 0 der CIA 2 (s.u.)). Dabei ist jedem der 8 Bytes ein Sprite (von 0-7) zugeordnet und die Pointer können Werte von 0-255 annehmen (16 Kbyte Adressierung).

b) höchstwertige Adressbits des VIC-Adressbereiches

Neben dem VIC-Register 24 gibt es noch eine weitere Speicherstelle, mit welcher z.B. Videoram oder Zeichensatz verschoben werden können. Es ist dies das Register 0 der CIA 2 (= Complex Interface Adapter 2) mit der Adresse 56576 ($DD00), speziell Bit 0 und 1. Diese beiden Bits ergeben die obersten zwei Adressbits (Bit 14 und 15) der Basisadresse des VIC

Vorsicht! Die Bits sind LOW-Aktiv, d.h. ist ein Bit=1 so gilt es als 0 und umgekehrt!

c) Joystick/Paddle/Tastatur

Hierfür sind (u.a.) die ersten zwei Register der CIA 1 zuständig:

Register 0 (Adresse: 56320/$DC00):
Normalbetrieb:
Bit 0-7: Reihenauswahl der Tastatur
weitere Aufgaben:
Bit 0-4: Joystick 0: Bit 0=1: oben
(Port 1) Bit 1=1: unten
Bit 2=1: links
Bit 3=1: rechts
Bit 4=1: Knopf
Bit 6/7: Paddle-Set-Auswahl: Bit 6=1: Set A
(Port 1) Bit 7=1: Set B
Nur eins der zwei Bits darf = 1 sein!

Für den Joystick-Betrieb muß hier zunächst das Register 0 durch ein POKE 56322,224 (Register 2 - $DC02) auf Eingabe gestellt werden. Rückstellung: POKE 56322,255 (gilt nicht unbedingt für Register 1).

Register 1 (Adresse: 56321/$DC01):
Normalbetrieb:
Bit 0-7: Spaltenrückmeldung der Tastatur

weitere Aufgaben:
Bit 0-4: wie Register 0 nur für Port 2
(Joystick 1 / Paddles)

Soweit in aller Kürze das Wichtigste zur CBM 64-Graphik. Nun lassen Sie sich genauer in die vielen Geheimnisse Ihres Rechners einweihen - Geheimnisse, die über Jahrtausende stets unter dem Siegel der Verschwiegenheit nur von Programmierer-ohr zu Programmiererohr weitergegeben wurden. Sie werden staunen, wie sich Ihnen plötzlich Welten auftun und Sie sich fürwahr traun in den 7. Programmierhimmel versetzt fühlen. Aufgemerkt nun also und die Ohren gespitzt!

3.2 Die Betriebsarten des VIC

Mit dem Video Interface Chip ist eine große Menge von Einstellungen möglich. Grob unterteilt man diese in drei Kategorien:

- hochauflösende Graphik mit dem Einzelpunktmodus (Standart Bitmap Mode)
- Sprites
- Textmodus (= Zeichen aus einem festen Zeichensatz)

Hinzu kommen noch zwei Modi, die Sie jeweils für diese drei Grunddarstellungen zusätzlich wählen können:

- Normalfarbenmodus
- Multicolormodus

und eine weitere Möglichkeit, die jedoch nur in Zusammenhang mit dem Text- und nicht gemeinsam mit dem Multicolormodus verwendet werden darf, der:

- Extended Color Modus

Die einzelnen Modi sollen im folgenden kurz skizziert werden, um Ihnen einen Überblick zu gewähren. Die nähere Besprechung geschieht dann aber in den späteren Abschnitten (für die Begriffe Farbram, Videoram, Graphikspeicher, Zeichensatzspeicher schauen Sie bitte unter # 3.3).

A. Einzelpunktmodus

a) Normaler Einzelpunktmodus

Im normalen Einzelpunktmodus, der durch das Setzen der Bits 5 und 6 des VIC-Registers 17 ausgewählt wird (Register 22, Bit 4=0), besteht ein direkter Zusammenhang zwischen Bildschirm und Graphikspeicher. Ein Bit des Graphikspeichers korrespondiert mit einem Punkt des Bildschirms.

Die Auflösung beträgt 320x200 Punkte, der Graphikspeicher nimmt somit einen Raum von etwa 8 K ein. Die Farbe kommt aus dem etwa 1 K großen Videoram, wobei je ein Byte des Videorams die Farbinformation für ein 8x8-Punkte großes Feld des Bildschirms liefert. Dabei gilt für jedes Bit des Graphikspeichers die folgende Farbherkunfts - Zuordnung:

Bit=0: 4 untere Bits des Videorams
Bit=1: 4 obere Bits des Videorams

b) Multicolor-Einzelpunktmodus

In dieser Betriebsart (wählbar durch Setzen der Bits 5 und 6 des VIC-Registers 17 und des 4. Bits des 22. Registers) sind jeweils 2 Bit des Graphikspeichers für einen doppelt breiten Punkt des Bildschirms zuständig. Die Auflösung beträgt daher nur 160x200 Punkte. Dafür sind pro 8x8-Punkte Feld insgesamt 4 verschiedene Farben wählbar. Mit Hilfe der erwähnten 2 Bit pro Punkt wird festgelegt, welche dieser Farben ein Punkt besitzen soll. Dabei gilt folgende Zuordnungen der Farbquellen:

Bits=00: Hintergrund-Farbregister 0
Bits=01: Videoram untere 4 Bits
Bits=10: Videoram obere 4 Bits
Bits=11: Farbram

B. Sprites

Sprites sind frei definierbare Objekte fester Auflösung, veränderlicher Größe und Farbe, von denen 8 völlig unabhängig voneinander gleichzeitig auf den Bildschirm gebracht werden können. Ihre Priorität untereinander und in Bezug auf die Hintergrundzeichen, sowie Ihre Position auf dem Bildschirm (Bewegungsauflösung: 512x256, also über den Bildschirmrand hinaus) können variiert werden. Die Spritedefinition wird in 63 Bytes untergebracht. Man unterscheidet:

a) Normale Sprites

Diese besitzen eine Auflösung von 24x21 Punkten. Jedes Bit der Spritedefinition repräsentiert einen Punkt der Spritematrix. Für die Farbe gilt dabei:

Bit=0: durchsichtig
Bit=1: Sprite Color Register (Reg. 39-46)

b) Multicolor-Sprite

Bei Multicolor-Sprites bestimmen jeweils 2 Bit der Definition einen doppelt breiten Punkt auf dem Bildschirm. Folglich schrumpft die Punkteauflösung auf 12x21 Punkte. Diese 2 Bit bestimmen die Herkunft der Farbe eines Punktes in folgender Weise:

Bits=00: durchsichtig
Bits=01: Multi Color Register 0
Bits=10: Multi Color Register 1
Bits=11: Sprite Color Register

Es ist möglich, gleichzeitig Sprites beider Darstellungsarten auf dem Bildschirm zu erzeugen.

C. Zeichendarstellung

Bei der Zeichendarstellung wird eine im Zeichensatzspeicher (Zeichengenerator) festgelegte Punkteanordung als festes Zeichen in einer 8x8-Matrix verwendet (für jedes Zeichen sind somit 8 Byte norwendig). In diesem Zeichensatzspeicher sind z.B. alle Buchstaben oder Zahlen festgehalten. Ein Byte des Videoram gibt dann die

Information, welches der maximal 2x256 Zeichen auf dem Bildschirm erscheinen soll.

a) Normale Zeichendarstellung:
In dieser Betriebsart wird das vollständige Byte aus dem Videoram als Zeiger auf ein Bitmuster des Zeichengenerators verwendet. Gleichzeitig sind maximal 256 verschiedene Zeichen auf dem Bildschirm darstellbar. Ein Bit des Bitmusters bestimmt dabei die Farbe eines einzelnen Punktes des Zeichens. Die Farbe stammt aus:

Bit=0: Hintergrundfarbregister 0
Bit=1: untere 4 Bits des Farbram

b) Zeichen im Multicolormodus:
In diesem Modus sind beide Möglichkeiten der Darstellung vorhanden: Normale und Multicolor-Darstellung. Ist das 3. Bit des Farbrams =0, so wird das Zeichen in der normalen 8x8-Matrix dargestellt. Wie unter a) beschrieben wird die Farbe des Zeichens ganz normal festgelegt mit der Einschränkung, daß zwangsläufig nur 8 Farben für die gesetzten Bits des Zeichenmusters möglich sind (das 3. Bit des Farbrams ist ja gleich 0).
Ist das besagte 3. Bit jedoch gleich 1, so bestimmen jeweils 2 Bit des Zeichenmusters die Farbe eines doppelt breiten Punktes des Zeichens. Die Auflösung beträgt somit nur 4x8 Punkte. Ein Zeichen aber kann nun aus insgesamt 4 Farben bestehen. Die Herkunft dieser Farben wird durch die beiden Bits eines Punktes bestimmt:

Bits=00: Hintergrundfarbregister 0
Bits=01: Hintergrundfarbregister 1
Bits=10: Hintergrundfarbregister 2
Bits=11: übrige drei Bits des Farbram

Die Farbe drei kann also ebenfalls nur eine von 8 Farben sein. Die restlichen drei Farben sind für alle Zeichen gleich.

c) Zeichen im Extended Color Modus

In diesem Modus kann jedes Zeichen des Bildschirms eine von 4 Hintergrundfarben erhalten. Die Zeichendarstellung an sich entspricht dem Normalmodus (8x8-Matrix). Für die gesetzten Bits des Zeichenmusters (Bit=0) stammt die Farbe ebenso wie im Normalmodus aus dem Farbram. Die gelöschten Bits dagegen, die die Hintergrundfarbe bestimmen, stammen aus verschiedenen Quellen. Die zwei höchsten Bits (Bit 6/7) des Videorams, also des Speichers, der die Zeichen enthält, legen dabei diese Quelle fest:

Bits=00: Hintergrundfarbregister 0
Bits=01: Hintergrundfarbregister 1
Bits=10: Hintergrundfarbregister 2
Bits=11: Hintergrundfarbregister 3

Da aber von den 8 Bits des Videorams nur noch 6 als Zeiger auf das Zeichenmuster übrig bleiben, können nur noch 64 verschiedene Zeichen gleichzeitig auf den Bildschirm gebracht werden!

Soweit der Überblick über die verschiedenen Betriebsarten des Videocontrollers, die in den ## 3.4 bis 3.6 näher erläutert werden. Als nächstes folgt eine weitere Spezialität Ihres Commodore 64, die ihn sehr flexibel macht.

3.3 Die Speicherverwaltung des CBM 64

Dieses Kapitel stellt einige Anforderungen an die Kenntnis um Speicheraufbau, Speicheradressierung und ähnliche Dinge und sollte daher von blutigen Laien übersprungen werden. Auch die Leser, die noch nichts über die Graphikmöglichkeiten dieses Gerätes oder gar überhaupt noch nichts über Graphik wissen, sollten sich nach einer kurzen Lektüre des Paragraphen 3.3.2.1 sogleich an das Kapitel 3.4 machen. Um sich allerdings effektiv und vollständig mit der Erstellung von Graphiken zu beschäftigen, ist eine Lektüre der folgenden Seiten unumgänglich, zumal noch einige Aussagen über die allgemeine Speicherhandhabung Ihres Gerätes gemacht werden. In diesem Zusammenhang sollte darauf hingewiesen werden, daß die meisten und wertvollsten Funktionen nur in Maschinensprache (Assembler) erreichbar sind. Aus diesem Grunde lohnt sich auf jeden Fall, sich vielleicht einmal mit diesem Themenkomplex zu beschäftigen. Oft besteht eine grundlose Hemmschwelle zu dieser Sprache. Dabei ist Sie für Leute, die bereits Basic programmieren, relativ leicht zu erlernen, besonders, wenn man ein gutes Buch zur Hand hat. Da DATA BECKER ein wirklich exellentes Werk über Maschinensprache speziell für 64er-Anwender herausgebracht hat, ist dieser Mangel behoben. Auch das nötige Rüstzeug (Maschinenspracheмonitor und Assembler) sind vorhanden. Sie werden sehen, bald programmieren Sie lieber in Assembler, als in dem klobigen und langsamen Basic. Sollten Sie sich jedoch nicht für diesen Weg entscheiden, und trotzdem viel mit Graphik hantieren, so empfiehlt sich eine entsprechende Graphikerweiterung, die es inzwischen ebenfalls für den 64er gibt. Hier stehen Ihnen einfache Befehle zur Verfügung, die das Programmieren für Sie zum Kinderspiel werden lassen. Hier sollte man sich natürlich auf dem Markt umgucken. Ein Tip: in Bezug auf Graphik ist die Geschwindigkeit ein sehr wesentlicher Faktor! Vergleichen Sie einmal und Sie werden auf das richtige Programm stoßen. Nun aber zum Wesentlichen:

Da sämtliche Bildschirminhalte gespeichert werden müssen, damit der VIC ständig weiß, was er auf den Bildschirm bringen

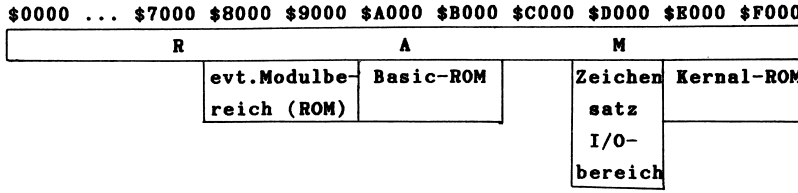
soll, bedarf es einer Menge an Speicherplatz für Text oder Graphik. Für den Graphikspeicher beispielsweise werden sage und schreibe 8 K, für die dazugehörige Farbe noch einmal 1 K, bei Multicolor sogar 2x1 K benötigt. Dieser Speicherplatz steht dann für andere Zwecke nicht mehr zur Verfügung. Aus diesem Grunde wurden einige Möglichkeiten geschaffen, die Lage dieser Bereiche selbst zu wählen, um optimales Arbeiten zu ermöglichen. Gleichzeitig stehen Bereiche zur Verfügung, die sonst nicht oder nur sehr schwer (z.B. von Basic aus) genutzt werden.

Um die folgenden Ausführungen zu verstehen, muß zunächst einiges über den Speicheraufbau Ihres Rechners gesagt werden: Der Commodore 64 besitzt einen 6510 als Hauptprozessor, auch CPU (Central Processing Unit) genannt, der für sämtliche Rechenvorgänge, also Programme benötigt wird. Dieser 6510 hat den gleichen Befehlssatz wie sein Vorgänger 6502, ist also softwarekompatibel zu ihm. Doch gibt es beim 6510 einige weitere Leitungen, die unter anderem die Speicherverwaltung unterstützen. Er besitzt damit zwei eigene Register mit den Adressen 0 und 1. Für uns ist lediglich das Register 1 von Bedeutung. Eine weitere Eigenart dieser CPU, die sie diesmal mit dem 6502 gemeinsam hat, ist der sogenannte Adressierungsbereich. Darunter versteht man die Größe des Speichers, den der Rechner ansteuern kann. Dieser beträgt bei Ihrem Rechner 64 K, da für die Adressierung, also die Ansteuerung von Speicherstellen, insgesamt 16 Bit zur Verfügung stehen. Damit können also die Adressen 0-65535 ($0000-$FFFF) angesteuert werden.

Nun wissen Sie aber, daß Ihr CBM 64 allein schon 64 K RAM besitzt (unter RAM = Random Access Memory versteht man Speicher, der gelesen und beschrieben werden kann. Sein Inhalt geht beim Ausschalten des Gerätes verloren. Er dient also als Arbeitsspeicher. Im Gegensatz hierzu kann der sogenannte ROM = Read Only Memory nur gelesen werden. Sein Inhalt wird nicht verändert, auch wenn der Computer ausgeschaltet wird. Seine Aufgabe ist die Beherbergung des Betriebssystems, des Basicinterpreters, des Zeichensatzes usw.).

Zu diesen genannten 64 K kommen weiterhin aber noch die verschiedenen ROM-Bereiche und Register der einzelnen Peripheriebausteine (VIC, CIA, Synthesizer etc.), die

insgesamt noch einmal einen Platz von 24 K benötigen. Wohin aber mit soviel Speicher, der normal gar nicht angesteuert werden kann?
Die Lösung ist die Speicherüberlappung. Diese wird so realisiert, daß mehrere Speicherbereiche (z.B. ROM und RAM) dieselben Adressen besitzen, also eigentlich an derselben Stelle im Speicher stehen. Der Aufbau sähe dann so aus:



Im gerade eingeschalteten Zustand sind alle diejenigen Bereiche lesbar, die in diesem Schema von unten sichtbar sind. Also:

$0000-$9FFF: RAM
$A000-$BFFF: ROM
$C000-$CFFF: RAM
$D000-$DFFF: I/O
$E000-$FFFF: ROM

In diese Tabelle schiebt sich bei dem Betrieb eines Moduls noch der Bereich $8000-$9FFF als Modul-ROM ein.
Dem Computer muß nun jedoch mitgeteilt werden, welchen Bereich er ansteuern soll. Dies hängt dabei von einigen Faktoren ab. Die für uns wichtigsten sind:

- Lese oder Schreibzugriff des 6510
- Zugriff des VIC oder des 6510
- Ansteuerung des Bereiches v. $D000-$DFFF oder nicht
- Inhalt des Registers 1, Bits 0-2 des 6510

Grundsätzlich müssen wir hier zwischen den beiden Prozessoren VIC und 6510 unterscheiden, da sie praktisch zur gleichen Zeit unterschiedliche Bereiche ansteuern. Um die Ansteuerung durch den VIC müssen wir uns in sofern kümmern, als wir ja bei Graphik o.ä. festlegen müssen, in welche Bereiche wir die

einzelnen Speicher (Videoram, ...) legen wollen, wir müssen also feststellen, ob der VIC diese überhaupt erreicht.
Die Ansteuerung des 6510 betrifft uns unmittelbar und auch, wenn wir überhaupt nichts mit Graphik bzw. Bildschirmsteuerung zu tun haben. Die Möglichkeiten von Basic aus sind zwar einigermaßen beschränkt, da viele Bereiche fest in Gebrauch sind, von Maschinensprache (Assembler) aber steht uns alles frei zur Verfügung. Um einen Überblick zu erhalten, beginnen wir mit:

3.3.1 Die Speicherzugriffe des 6510

3.3.1.1 Lesen eines Bytes

Soll ein Byte einer bestimmten Adresse (z.B. durch PEEK) gelesen werden (darunter versteht man auch den Ablauf eines Maschinenprogrammes), so hängt die Auswahl, welche der sich überlappenden Speicherbereiche angesprochen werden, -sofern für uns beeinflußbar- nur von dem Inhalt des sogenannten Datenregisters der 6510 CPU (Register 1) ab. Dort haben die ersten 3 Bits (0-2) die Funktion, auf bestimmte Bereiche umzuschalten. Die drei Bits sind nach dem Einschalten des Gerätes gesetzt, was zu der im obigen Schema verdeutlichten Speicherkonfiguration führt, und haben im Einzelnen die folgenden Funktionen:

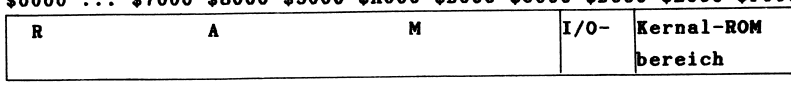
A. Bit 0/1 - LORAM/HIRAM:

a) Bits 0/1=11:
Sind diese Bits beide 1, so wird bei einem Lesezugriff auf die Adressen $A000-BFFF das dort befindliche Basic-RAM, bei einem Zugriff auf die Adressen $E000-$FFFF das dortige Kernal-ROM angesteuert. Dies ist der Normalzustand. Ist ein Modul im $8000er Bereich installiert, so ist auch dieses eingeschaltet. Die Speicherkonfiguration sähe dann also so aus:

$0000 ... $7000	$8000 $9000	$A000 $B000	$C000	$D000 $E000	$F000
R A M	evt.Modulbereich (ROM)	Basic-ROM		I/O-bereich	Kernal-ROM

b) Bits 0/1=10:

Ist lediglich das Bit 1 gesetzt, so ist der Modul- und der Basic-ROM - Bereich ausgeschaltet und statt dessen wird nun aus dem darunter befindlichen RAM gelesen. Wir haben also folgende Belegung:



Diese Konfiguration kann (auch von Basic aus) genutzt werden, indem z.B. der Basic - Interpreter in das unten liegende RAM copiert wird und, nachdem einige Veränderungen vorgenommen wurden, von nun an in diesem RAM als modifiziertes Basic liegt. Ein Basicprogramm dazu könnte z.B. so aussehen:

```
10 FOR AD=10*4096 TO 12*4096-1
20 POKE AD, PEEK(X) : REM BASIC-ROM INS RAM COPIEREN
30 NEXT AD
40 REM HIER NUN VERAENDERUNGEN EINPOKEN ...
50 POKE 1, PEEK(1) AND 254 : REM RAM EINSCHALTEN
```

Zu beachten ist hierbei, daß bei einem Modulbetrieb der Bereich von $8000-$9FFF ebenfalls copiert werden muß, um einen Absturz zu verhindern, da er gleichfalls ausgeschaltet wird.
Gehen Sie mit diesem Register 1 der CPU besonders vorsichtig um (vor allem bei den folgenden Belegungen), da hier jede Änderung tiefgreifende Einschnitte in die Speicherorganisation Ihres Rechners mit sich führt und bei einer Fehlbelegung oft nur der Ausschaltknopf die einzige Möglichkeit ist, Ihren Rechner wieder "zum Leben zu erwecken" (keine Angst, Ihrem Rechner passiert dabei natürlich nichts).

c) Bits 0/1=01:

Ist nun nur das Bit 0 gesetzt, so haben Sie sämtlichen ROM-Speicher ausgeschaltet. Der I/O-Bereich von $D000-$DFFF bleibt davon jedoch unberührt, kann also weiterhin ohne Änderungen angesteuert werden. Wir erhalten damit 60 K ansteuerbares RAM:

$0000 ... $7000	$8000	$9000	$A000	$B000	$C000	$D000	$E000	$F000
R A M						I/O-bereich	R	A M

d) Bits 0/1=00:

Sind beide Bits gelöscht, so ist sämtlicher RAM eingeschaltet. Kein ROM und auch keine I/O-Funktionen können mehr ausgelesen oder verändert werden (obwohl der VIC und die anderen Bausteine weiterhin alle ihre Register lesen können und somit das Bild erhalten bleibt!). Die Konfiguration ist denkbar einfach:

$0000 ... $7000	$8000	$9000	$A000	$B000	$C000	$D000	$E000	$F000
R A M								

Dies ist die einzige Möglichkeit, die unter dem I/O-Bereich und dem Zeichensatz liegenden 4 K RAM mit zu nutzen.

B. Bit 2 - CHAREN:

Dieses Bit dient dazu, den verschiebbaren Zeichengenerator (s.u.) auch für den 6510, also für den Programmierer lesbar zu machen, um ihn z.B. zu copieren und dann zu verändern (s.u.).

a) Bit 2=1:

Dieses Bit bezieht sich lediglich auf den Bereich der $D-Seiten ($D000-$DFFF) des Commodorespeichers. Die anderen Adressen werden nicht beeinflußt. Im Normalzustand liegt es gesetzt vor. Damit kann der 4 K große Zeichensatz, der ebenfalls genau den Bereich von $D000-$DFFF einnimmt, nur vom VIC gelesen werden. Statt dessen stehen die I/O-Adressen, das sind die Register des VIC, des SID (Sound Interface Device) und der CIAs, sowie der Farbram zur freien Verfügung. Wir finden also folgende Einrichtungen in besagtem Raume vor:

$D000 ...	$D400 ...	$D800 ...	$DC00	$DD00	$DE00	$DF00
VIC-Reg.	SID-Reg.	Farbram	CIA 1	CIA 2	I/O 0	I/O 1

b) Bit 2=0

Ist das Bit gelöscht, so kann auch der Programmierer bzw. die 6510 CPU den Inhalt des Zeichensatzspeichers lesen. Dabei belegt er alle 4 K des I/O-Bereiches. Versuchen Sie das jedoch in Basic, so wird sich Ihr Rechner abrupt innerhalb der nächsten 1/60 Sekunde von Ihnen verabschieden, da 60 mal pro Sekunde eine sogenannte Interruptroutine, von der wir im # 3.7 mehr hören werden, aufgerufen wird, die mit dem in der CIA 1 befindlichen Timer hantiert, der dann natürlich nicht mehr ansprechbar ist, wenn dieses Bit auf 0 gesetzt ist. In Maschinensprache muß zunächst das Interruptflag gesetzt werden (Befehl: SEI), um den Aufruf dieser Interruptroutine zu unterbinden. Später wird durch CLI der Interrupt wieder ermöglicht. Das Gleiche muß geschehen, wenn durch das Löschen von Bit 1 das Kernal-ROM ausgeschaltet wird, in dem sich die Interruptroutine befindet.

3.3.1.2 Schreiben eines Bytes

Wird ein Schreibzugriff auf den Speicher unternommen (z.B. durch POKE), so ändern sich die Verhältnisse grundsätzlich:
Da es nicht sinnvoll ist, etwas in den ROM-Speicher zu schreiben, wurde es eingerichtet, daß jeder Schreibzugriff unabhängig von der Einstellung im Register 1 der CPU den unter dem ROM liegenden RAM erreicht. Mit einer Ausnahme: Der Bereich von $D000 bis $DFFF. Hier wird normalerweise nur der I/O-Speicherbereich erreicht. Wollen Sie auch hier den RAM durch einen Schreibzugriff ansprechen, so stehen Ihnen zwei Möglichkeiten zur Auswahl:

- Ausschalten aller ROMs
- Einschalten des Zeichensatzspeichers

Ersteres geschieht bekanntlich, indem Sie die ersten beiden Bits (0 und 1) des 1. CPU-Registers =0 setzen, der zweite Zustand wird durch Löschen des Bits 2 dieser Speicherstelle erreicht (s.o.).
Nun werden Sie wohl auch das kleine oben angeführte

Basic-Programm vollständig verstehen. In Zeile 20 wurde dort durch den Befehl POKE AD, PEEK(AD) der Inhalt des Basic-ROMs gelesen, durch den Schreibbefehl (POKE) aber nicht etwa wieder dorthin zurückgeschrieben, sondern in das darunterliegende RAM! Sie sehen, welche Bedeutung dieser Tatsache zukommt.

3.3.2 Die Speicherzugriffe des VIC

Neben dem Hauptprozessor, der für den Ablauf aller Programme zuständig ist, muß selbstverständlich noch der Videocontroller (VIC) auf den Speicher zugreifen, um Bildschirminformationen wie Farbe oder Punktsetzung zu erhalten. Da hier natürlich Zugriffe z.B. auf das Kernal-ROM oder den Basicinterpreter sinnlos sind, wurde hier einiges anders gestaltet, als dies bei der CPU-Ansteuerung der Fall ist. Die Umschaltungen zwischen verschiedenen Speicherbereichen nimmt der VIC dabei selbständig vor. Für uns ergibt sich damit ein mehr oder weniger statisches Bild im Vergleich zu den vielen Möglichkeiten des Programmierers bei der gerade besprochenen CPU. Wir müssen also lediglich wissen, welche Bereiche der VIC für welche Zwecke ansteuert und welche nicht. Gleichzeitig zeigt sich die Speicherverwaltung des VIC von einer anderen Seite äußerst dynamisch. Sie selbst können nämlich unter Einhaltung verschiedener Bedingungen bestimmen, wo welche Speicherfunktionen des Videocontrollers liegen sollen. Mit anderen Worten, Sie haben die Möglichkeit, Graphikspeicher, Zeichensatz, Videoram und Spritespeicher in die Speicherbereiche zu verlegen, die Ihnen angenehm erscheinen.
Doch bevor wir uns damit beschäftigen, wollen wir zunächst ein wenig zu den Speicherfunktionen des VIC sagen, um die Begriffe zu klären, die von nun an unser täglich Brot sein werden.

3.3.2.1 Die Speicherfunktionen des VIC:

Um die verschiedenen Aufgaben zu erfüllen, die dem VIC zugeordnet sind (Text, Graphik, Sprites, Farbe, ...) bedarf es umfangreicher Speicherräume mit den unterschiedlichsten Funktionen:

- Zeichengenerator
- Videoram
- Farbram
- Graphikspeicher
- Spritedefinitionen

Im Folgenden sollen die Bedeutung und der Nutzen jeder dieser einzelnen Positionen, die schon in früheren Kapiteln erwähnt worden waren, dargelegt und erläutert werden. Die näheren Funktionen und der genaue Aufbau jedoch werden in den folgenden ## 3.4 bis 3.6 abgehandelt. Die hier befindliche Auflistung soll lediglich als Orientierung und Begriffserklärung dienen, um das hernach zu Sagende verständlich zu machen, da wir dort ständig mit diesen Dingen umgehen.

a) Zeichengenerator:
Unter Zeichengenerator (auch Zeichensatz oder Zeichensatzspeicher genannt) verstehen wir denjenigen Speicher, der die Definitionen bzw. das sogenannte Bitmuster für jedes einzelne Zeichen enthält, das wir durch einfachen Tastendruck auf den Bildschirm bringen können. Er umfaßt insgesamt 2x2 K und damit die Information für 512 Zeichen, von denen allerdings jeweils nur ein Teil gleichzeitig auf den Bildschirm gebracht werden kann (s. ## 3.2, 3.6, 4.4).

b) Videoram:
Der Videoram umfaßt etwa 1 K und hat verschiedene Aufgaben. Im normalen Zustand (Einschaltzustand) dient er als Zeichenspeicher (nicht zu verwechseln mit "Zeichensatzspeicher"), in dem der sogenannte Bildschirmcode (ein Code für Zeichen ähnlich dem ASCII-Code, er dient als Zeiger auf den Zeichengenerator) für jedes einzelne Zeichen, das sich zur Zeit auf dem Bildschirm befindet, abgelegt ist.

Im Graphikmodus erhält der Videoram die Funktion des Farbspeichers, der die Punkt- und die Hintergrundfarbe bzw. in Multicolor Farben 1 und 2 jedes 8x8-Punktefeldes (in MC: 4x8) des Graphikbildschirms bestimmt (s. ## 3.4, 3.6, 4.2, 4.4).
Eine weitere Funktion des Videoram ist die Beherbergung der Pointer auf die Spritedefinitionen in den letzten 8 Bytes.

c) Farbram:

Der Farbram umfaßt wie der Videoram ca. 1 K und liegt fest in dem Bereich $D800-$DBFF (55296-56319). Er ist lesbar und beschreibbar, jedoch sind jeweils nur die unteren 4 Bits aktiv. Die oberen 4 können nicht verändert werden und sind stets gesetzt. Im normalen Modus dient er als Speicher für die Farbe der Textzeichen auf dem Bildschirm. Im Graphikmodus hat er nur bei Multicolor eine Funktion. Dort stellt er die MC-Farbe 3 jeweils für ein 4x8-Punktefeld des Graphikbildes dar.

d) Graphikspeicher:

Der sogenannte Graphikspeicher, der umfangreichste Bildspeicher überhaupt, beinhaltet, wie der Name sagt, den Inhalt eines Graphikbildes. Dabei muß jeder einzelne Punkt des Bildschirms separat gespeichert werden. Bei einer Auflösung von 320x200 (in hochauflösender Graphik) sind dies 64000 Punkte, für deren Speicherung etwa 8 K benötigt werden.

e) Spritedefinitionen:

Um das Aussehen der verschiedenen Sprites zu speichern, werden vom Benutzer diverse Speicherbereiche reserviert, deren Umfang sich jeweils auf 63 Bytes erstreckt. In diesen 63 Bytes sind alle 21x24 Punkte eines normalen Sprite vermerkt.

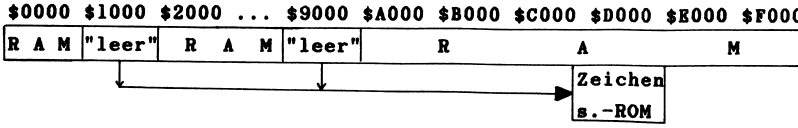
3.3.2.2 VIC-Speicheransteuerung:

Die Ansteuerung der verschiedenen Speicherbereiche durch den Videocontroller ist weitaus einfacher und übersichtlicher als die vielen Möglichkeiten, die die CPU bietet. Hier kann man einfach nach dem Grundsatz vorgehen: Wenn im folgenden nichts anderes gesagt wird, so wird stets der RAM (auch in dem Bereich von $D000-$DFFF = 53248-57343) vom Videocontroller angesprochen, selbst wenn für den Programmierer lediglich ROM erreichbar ist (also unabhängig von der Einstellung im Register 1 der CPU). Dies ist äußerst wichtig, da dadurch z.B. der Graphikspeicher platzsparend unter den ROM gelegt werden kann, also keinen Basic-Speicherplatz verschwendet, wie dies z.B. in der SUPERGRAPHIK 64 verwirklicht wurde. Dieser Grundsatz gilt für:

- Videoram
- Graphikspeicher
- Spritedefinitionen
- Zeichengenerator

Diese drei Speicher werden also stets aus dem RAM geholt --- mit zwei Ausnahmen:
Werden durch eine Einstellung (wie im nächsten Abschnitt beschrieben) die Speicherbereiche von $1000-$1FFF (4096-8191) oder von $9000-$9FFF (36864-40960) angesprochen (etwa, wenn man versucht die Graphikseite nach $8000-$9FFF zu legen, oder in der Einschaltkonfiguration für die Sprites die Blöcke 64-127 (Bereich $1000-$1FFF) zu wählen), so werden nicht etwa die Informationen aus diesen RAM-Bereichen, sondern aus dem Zeichengenerator-ROM geholt, das bei $D000-$DFFF liegt (s.o.).
Diese zunächst merkwürdige Besonderheit erklärt sich aus der Tatsache heraus, daß nach dem Einschalten Ihres Rechners für den VIC nur die unteren 16 K des 64-Speicherbereiches ansteuerbar sind (s.u.). Der feste Zeichensatz, der ja bekanntlich für die Herstellung der einzelnen Textzeichen auf dem Bildschirm notwendig ist, liegt jedoch bei $D000-$DFFF unter den I/O-Adressen. Aus diesem Grunde wurde der Adresse $1000-$1FFF jener Sonderstatus zugesprochen und der Zeichensatz logisch eigentlich nach $1000-$1FFF verlegt. Die anderen

Funktionen fallen dem dann leider auch zum Opfer. Der Bereich $9000 resultiert ebenfalls aus dieser Sonderstellung.
Bemerkenswert ist in diesem Zusammenhang, daß bei einer direkten Adressierung (also nicht indirekt über die Adresse $1000) der verschiedenen Funktionen (auch des Zeichensatzes) durch den Programmierer in den Bereich $D000 ff. nicht etwa der dort liegende Zeichensatz-ROM vom VIC angesprochen wird, sondern vielmehr der darunter liegende RAM. Die Speichereinteilung sieht im Schema also folgendermaßen aus:



Dies nur der Vollständigkeit halber. Mehr davon in dem folgenden Abschnitt.
Bei dem Farbram ist die Sache fast noch einfacher: Da er nicht verschiebbar ist, wie die anderen Bereiche (s.u.), wird er stets aus dem Bereich $D800-$DBFF (55296-56319) gewonnen. Hier ist jedoch zu beachten, daß er einen eigenen, vom darunter liegenden RAM verschiedenen Bereich belegt, der für den Programmierer in der Ebene der I/O-Adressen liegt (s.o.). Der Farbram bei $D800 darf also nicht mit dem normalen RAM bei $D800 verwechselt werden.

3.3.2.3 Verschieben der Bildschirmspeicher:

Wohl mit eine der schönsten und praktischsten Dinge in der Speicherverwaltung des Videocontrollers ist die Möglichkeit, die einzelnen Bildschirmspeicher in dem gesamten Speicher Ihres Rechners zu verschieben. Sie können also den Graphikspeicher, den Sie für Ihre Graphiken verwenden, sowohl z.B. nach $2000 (8192) schaffen, als auch, wenn Sie die dortige Lage im Basicbereich stört, meinetwegen etwa nach $E000 (41440) unter den ROM verschieben. Vielleicht legen Sie sogar zwei oder mehr Graphikseiten an, von denen Sie dann eine bearbeiten können, während die andere sichtbar für den Beobachter bleibt, wie dies z.B. in der Graphikerweiterung SUPERGRAPHIK 64, die eben schon angesprochen wurde, möglich ist.

Oder Sie verschieben den Zeichensatzspeicher in einen anderen Bereich und können sich so Ihren eigenen ganz persönlichen Zeichensatz erstellen (z.B. mit Umlauten, Sonderzeichen, ... - s. hierzu auch ## 3.6, 4.4). Es eröffnet sich Ihnen eine derartige Fülle von Möglichkeiten, wie Sie sich zur Zeit wahrscheinlich noch gar nicht vorstellen können!
Doch bei jeder Verschiebung halten Sie stets im Auge, welche Speicherbereiche der VIC überhaupt ansteuern kann, was soeben im vorherigen Abschnitt dargelegt wurde. So wird es beispielsweise nie gehen, eine Spritedefinition in den RAM bei $1000-$1FFF zu legen, da dort -wie Sie wissen- der VIC nicht RAM sondern den Zeichensatz-ROM bei $D000-$DFFF ließt (s.o.). Deshalb ist das Verständnis des Paragraphen 3.3.2.2 für die folgenden Ausführungen von unbedingter Notwendigkeit!

a) Allgemeine Verschiebung:

Der VIC oder Videocontroller kann von Haus aus, also intern für sich lediglich 16 K ($0000 - $3FFF oder %0000 0000 0000 0000 bis %0011 1111 1111 1111) adressieren. Unser Adressierungsbereich umfaßt aber 64 K, also 4 mal so viel. Dem VIC fehlen demnach die obersten zwei Adressenbits (Bits 14 und 15). Sie müssen von außen zugeführt werden. Hierfür ist ein Register zuständig, das (selbstverständlich) bereits in # 3.1 erwähnt wurde, es ist das

Register 0, Bits 0/1 der CIA 2 ($DD00=56576)

Diese beiden Bits stellen die gesuchten zwei obersten Adressbits für die Speicheradressierung des VIC dar (im folgenden Schaubild unterstrichen):

Adressbits $F E D C B A 9 8 7 6 5 4 3 2 1 0

Man könnte Sie also einfach einsetzen und hätte die vollständige Adresse. Die Sache hat aber einen kleinen Haken. Diese beiden Bits sind LOW-Aktiv, d.h. sind sie

gesetzt, so gelten Sie als gelöscht und umgekehrt. Wollen wir die richtige Adresse erhalten, so müssen wir sie erst umdrehen (invertieren). Haben wir dies erledigt, so kennen wir den Bereich, den der VIC nun ansteuern kann, d.h. es verschieben sich automatisch alle Bildspeicherfunktionen, die vom VIC angesteuert werden (außer der Farbram), jeweils in 16 K-Schritten:

- Videoram
- Graphikspeicher
- Zeichengenerator
- Spritedefinitionen

Zu Ihrer Unterstützung seien hier die Speicherbereiche tabellarisch festgehalten, die durch eine bestimmte Belegung dieser Bits in die Reichweite des VIC gelangen:

B 0/1	Adr-B	erreichbare Adressen
11	00	$0000-$3FFF (0-16383)
10	01	$4000-$7FFF (16384-32767)
01	10	$8000-$BFFF (32768-49151)
00	11	$C000-$FFFF (49152-65535)

Unter "B 0/1" wird hier die Belegung der zwei Bits aus Register 0 der CIA 2 verstanden. "Adr-B" sind dann die daraus resultierenden Adressbits 14 und 15 für die VIC-Speicheradressierung. Die originale Belegung ist: B 0/1=11, also der erste Fall in der Tabelle. Nur so kann der Videoram von $0400-$07FF (1024-2047) gehen und der Zeichensatz (durch die Sonderstellung der Adresse $1000 (s.o.)) bei $D000 (53248) liegen.
Ein Beispiel: Angenommen, Sie wollen aus irgendeinem Grunde den Videoram, der ja die Speicherung aller Bildschirmzeichen vornimmt, nach $C400 (bzw. 50176 = 49152+4*256) verschieben (abgesehen einmal davon, daß ohne weitere Änderungen dann keine Zeichen mehr auf dem Bildschirm verändert werden können). Zu diesem Zweck geben Sie lediglich den Befehl:

```
POKE 56576, PEEK(56576) AND 253 OR 0
```

ein, und schon holt sich der VIC seine Informationen nur noch aus dem Bereich zwischen $C000 und $FFFF, was in unserem Falle zu einem wilden Chaos auf dem Bildschirm führt (nur die Farbe kann richtig verändert werden, wegen der Unverschiebbarkeit des Farbrams), das durch den Befehl

POKE 56576, PEEK(56576) AND 253 OR 3

wieder rückgängig gemacht werden kann (was Sie dort blind in die Tastatur eingeben erscheint natürlich erst nach dem <return> auf dem Bildschirm, falls Sie keinen Fehler gemacht haben.).

b) Verschieben des Videoram:

Zusätzlich zu dieser allgemeinen Verschiebung kann u.a. separat auch noch der Videoram innerhalb dieses 16 K Adressierungsbereiches in kleineren Schritten verschoben werden. Diese Verschiebung ist möglich durch die Veränderung des VIC-Registers 24 ($18), speziell der Bits 4-7 (s. # 3.1). Diese 4 Bits legen hier einen Teil der Adresse zur Ansteuerung des Videorams fest. Es sind dies die Adressbits 10-13 ($A-$D). Das folgende Schaubild mag das erläutern:

Adressbits	$F E	D C B A	9 8 7 6 5 4 3 2 1 0
	CIA2	Reg. 24	V I C - i n t e r n
	B0/1	Bits 4-7	

Der Videoram kann also in 1 K-Schritten innerhalb des gesamten 16 K-Adressraumes verschoben werden. Nach dem Einschalten lauten die 4 besagten Bits: %0001. Aus diesem Grunde liegt unser Videoram (in Funktion als Textspeicher) bei $400 (1024). Wichtig ist, daß diese Adresse -anders als die unter a) beschriebene Verschiebemöglichkeit- nur und ausschließlich für den Videoram gilt. Bei einer Veränderung dieser Bits bleiben die anderen Speicherbereiche in ihrer Lage unverändert.

Ein Beispiel: Wir wollen unseren Videoram, also den Textspeicher, der noch bei $400 (1024) liegt, nach $800 (2048) verlegen (auch hier erwarten wir natürlich

Nonsense, da hier der Basic-Speicher beginnt). Dies erreichen wir mit dem Befehl:

```
POKE 53248+24, PEEK(53248+24) AND 15 OR 2*16
```

um uns wieder in die heimischen Gefilde, sprich: zu unserem alten Videoram zu begeben, drücken wir ein:

```
POKE 53248+24, PEEK(53248+24) AND 15 OR 1*16
```

Diese Verschiebemöglichkeit kann z.B. von Nutzen sein, wenn Sie so große Basic-Programme verwenden, daß Sie den Speicherbereich von $400-$7FF ebenfalls nutzen möchten, oder Sie verwenden zwei Textseiten, die Sie dann ständig umschalten oder ... oder ... oder ...

c) Verschieben des Zeichengenerators:

Neben dem Videoram können Sie auch den Zeichengenerator innerhalb des unter a) gewählten 16 K-Raumes verschieben. Auch hier dient uns das 24. Register des VIC als Zwischenspeicher für einige Adressbits. Diesmal sind es lediglich 3, die die Adressbits 11-13 darstellen, weswegen wir den insgesamt 2x2 K großen Zeichensatz lediglich in 2 K-Schritten verschieben können:

Adressbits	$F E	D C B	A 9 8 7 6 5 4 3 2 1 0
	CIA2	Reg.24	V I C - i n t e r n
	B0/1	Bitl-3	

Interessant ist, daß auch das Betriebssystem Ihres Computers von diesen 3 Bits Gebrauch macht. Wenn Sie durch die Tastenkombination <C=><shift> auf den alternativen Zeichensatz umschalten, so ändert Ihr Rechner das 11. Bit der Zeichensatzadresse durch Änderung des 1. Bits von VIC-Register 24 (s. auch # 3.6).
Wichtig ist dabei das unter # 3.3.2.2 zu der Sonderstellung des Adressenbereiches von $1000-$1FFF (4096-8191) Gesagte. Wollen Sie den Zeichengenerator verschieben, um z.B. einen eigenen zu betreiben, so ist es vielleicht sehr nützlich, wenn Sie auch den # 3.3.1 gelesen haben.

d) Verschieben des Graphikspeichers:

Auch die Lage des Graphikspeichers kann gewählt werden. Da er jedoch 8 K groß ist, paßt er nur zweimal in den 16 K-Adressierungsbereich hinein. Aus diesem Grunde können Sie auch nur ein separates Bit für ihn wählen (außer natürlich der allgemeinen Verschiebung). Es ist dies das 3. Bit des VIC-Registers 24, das nun praktisch zwei Aufgaben besitzt:

- Zeichensatzverschiebung
- Graphikspeicherverschiebung

Wie beim Zeichensatz bestimmt es das 13. Bit der Graphikspeicheradresse:

Adressbits	$F E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	CIA2	B	V	I	C	-	i	n	t	e	r	n			
	B0/1	3													

Liegt Ihr VIC-Adressbereich z.B. bei $0000-$3FFF (0-16383), so können Sie wählen, ob der Graphikspeicher bei $0000 oder $2000 (8192) beginnen soll (in diesem Fall empfiehlt sich natürlich zweiteres, da ansonsten Nullseite, Stack usw. in Mitleidenschaft gezogen würden). Alles Weitere erfahren Sie im nächsten Kapitel 3.4.

3.4. Punktgraphik

Nach so vielen Einzelheiten über Register und Speicherverwaltung sollten wir uns nun einmal speziell dem Aufbau der hochauflösenden bzw. der Multicolor- Graphik, also der sogenannten Punktgraphik (da jeder Punkt einzeln angesteuert werden kann) widmen - ein hochinteressantes Thema und unumgänglich für jeden Graphikprogrammierer. Dieser Abschnitt (und die vier folgenden) sind die hardwaremäßigen Vorbereitungen auf die im Kapitel 4 dargelegten Programmiermöglichkeiten. Versuchen Sie also, auch wenn Sie nicht alles verstehen sollten, sich in diesen Komplex hinein zu denken und wenigstens in etwa den Aufbau der Graphik gegenwärtig zu haben. Dabei wird sich, wie Sie sehen, die recht komplizierte Farbrealisierung Ihres Gerätes in den verschiedenen Graphikarten besonders später in der Anwendung als ziemlich schwierig herausstellen. Geben Sie jedoch nicht auf. Ein Buch und ganz besonders eins dieser Art sollte sowieso mindestens zweimal gelesen werden. Mit manchen Passagen werden Sie wahrscheinlich täglich arbeiten, wenn Sie sich näher der Graphik widmen wollen. Fangen wir aber gleich einmal an:

3.4.1. Farben:

Ihr Commodore 64 verfügt über die Möglichkeit, 16 Farben sowohl im Graphikbetrieb, als auch (wie sicher schon bekannt) für die Textgestaltung zu verwenden. Diese 16 Farben besitzen jeweils einen sogenannten Farbcode, der als Binärzahl in die verschiedenen Register gespeichert wird, die dem Gerät zur Zeichendarstellung verhelfen. Wird z.B. in das 32. Register des Video Interface Chips der Wert 0 gePOKEd, so nimmt der äußere Bildschirmrahmen die Farbe schwarz an. Im Folgenden sind die den einzelnen Farben zugeordneten Codes aufgelistet (im Anhang finden Sie eine vollständige Tabelle, die sich mit dem gleichen Thema beschäftigt):

Code		Farbe	Code		Farbe
Dez	Hex		Dez	Hex	
0	$00	schwarz	8	$08	orange
1	$01	weiß	9	$09	braun
2	$02	rot	10	$0A	hellrot
3	$03	türkis	11	$0B	grau 1
4	$04	violett	12	$0C	grau 2
5	$05	grün	13	$0D	hellgrün
6	$06	blau	14	$0E	hellblau
7	$07	gelb	15	$0F	grau 3

Diese Tabelle wird Sie ständig in allen Bereichen der Graphik begleiten, sei es, Sie arbeiten mit Sprites, Graphik, Text oder was auch immer. Sie sollten Sie also stets im Auge halten. Über den Einsatz der Farben wird Ihnen in den entsprechenden Kapiteln Auskunft gegeben.

3.4.2. Hochauflösende Graphik (HGR)

Ihr Rechner hat die Möglichkeit, von Haus aus zwei verschiedene Graphikarten zu bedienen:

- hochauflösende Graphik (HGR)
- Multicolorgraphik (MC)

Erstere bietet Ihnen ein Graphikfeld von 320 Punkten in x-Richtung (waagerecht) und 200 Punkten in y-Richtung (senkrecht). Man spricht von einer Auflösung von 320x200. Dies verschafft Ihnen ein Reservoir von insgesamt 64.000 Punkten jeweils in gleicher Dichte verteilt auf Ihrem Bildschirmfenster.
Natürlich muß die Graphik genauso wie der Text oder die Farbe gespeichert sein. Schließlich muß der VIC das auf dem Bildschirm entstehende Bild alle ca. 1/20 Sekunden selbst auf Ihrem Fernseher oder Monitor neu erstellen, damit Sie es ständig beobachten können (s. Lightpenkapitel). Dies geschieht für die Punktgraphik mit Hilfe des sogenannten Graphikspeichers. Jeder Punkt ist einzeln ansprechbar und in diesem Graphikspeicher durch ein Bit repräsentiert. Wie Sie wissen (s. Kapitel 2) ist ein Bit eine Informationseinheit

und kann die Werte 1 oder 0 annehmen. Jeweils 8 Bit hintereinander bezeichnet man bekanntlich als ein Zeichen (Wort) bzw. als ein Byte. Ein Byte kann also 2 hoch 8 = 256 verschiedene Werte annehmen. Diese kommen durch die verschiedenen Kombinationen von gesetzten (1) und nicht gesetzten (0) Bits zustande.
Ein Byte repräsentiert also 8 Punkte auf Ihrem Bildschirm. Folglich bedarf es 64000/8 = 8000 Bytes (etwa 8 Kilobytes (8 K)), um den gesamten Bildschirminhalt der HGR zu speichern. Wie Sie vielleicht aus dem # 3.3.2.3 wissen, falls Sie sich ihn durchgelesen haben, können diese 8 K irgendwo im 64 K-Speicher Ihres Rechners plaziert werden. Haben Sie dies getan (s. # 4.2.1.1), so können Sie anfangen. Vorher aber müssen Sie erfahren, wie denn eigentlich der Aufbau des Graphikbildes aus den Speicherinformationen vonstatten geht:

a) <u>Graphikaufbau:</u>

Man könnte sich vorstellen, daß der Graphikspeicher ein direktes Abbild des Bildschirms ist, also Reihe für Reihe nacheinander alle notwendigen Bytes hintereinander folgen. Doch die Angelegenheit ist nicht so einfach, wie sie ausschaut. Der Graphikspeicher besitzt (das ist hardwaremäßig einfacher zu gestalten) den gleichen Aufbau wie der später beschriebene Zeichengenerator. Ein Zeichen besteht, das kann man hier schon einmal vorwegnehmen, aus 8x8-Punkten. Sie besitzen also eine sogenannte 8x8-Punktematrix (Matrix ist der gebräuchliche Begriff für Raster). Grundlage des Graphikspeichers ist ebenfalls eine 8x8-Matrix, die jeweils durch 8 Bytes des Graphikspeichers dargestellt wird. Da eine solche Matrix also 8 Punkte hoch und breit ist, passen insgesamt 320/8 = 40 solcher Päckchen in eine Zeile und 200/8 = 25 in eine Spalte (genauso ist der Aufbau im Textmodus). Ein Byte liefert nun die Information für eine 8-Punkte breite Reihe eines solchen Päckchens. 8 Bytes untereinander (im Speicher natürlich hintereinander) stellen somit diesen Block dar. Dabei bilden die einzelnen korrespondierenden Bits jedes dieser 8 zusammengehörenden Bytes -also Bits der gleichen Nummer oder Wertigkeit- untereinander liegende Punkte. Dies kann anhand eines kleinen Schaubildes verdeutlicht werden:

		Spalte 0								Spalte 1								....
	Bit:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	usw.
Z	Byte 0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
e	Byte 1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
i	Byte 2	.	.	.	.	.	.	.	.	.	.	.	.					
l	Byte 3	.	.	.	.	.	.	.	.									
e	Byte 4	.	.	.	.	.	.	.	.									
	Byte 5	.	.	.	.	.	.	.	.									
0	Byte 6	.	.	.	.	.	.	.	.									
	Byte 7	.	.	.	.	.	.	.	.									
Z	Byte 320	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
.	Byte 321	.	.	.	.	.	.	.	.	.	.	.	.					
1	Byte 322	.	.	.	.	.	.	.	.									
	usw.																	

Es wird also Zeile für Zeile aus diesen Päckchen generiert. Noch etwas zu Terminologie: Eine Spalte nennt man die 8 Punkte dicken senkrechten "Balken", von denen jeweils 40 nebeneinander auf den Bildschirm passen. Eine Zeile ist das gleiche, nur waagerecht ausgerichtet. 25 Zeilen passen somit untereinander in das Graphikfenster. Jede Spalte besteht aus 8 senkrechten Reihen, jede Zeile aus 8 waagerechten Reihen. Damit passen 320 senkrechte und 200 waagerechte Reihen in ein Bild. Jede Spalte, Zeile und Reihe ist numeriert (startend bei 0) und kann so genau lokalisiert werden. Diese Vereinbarungen sind im folgenden wichtig, um Verwechslungen zu vermeiden.

Um nun auf einfache Weise einen einzelnen Punkt anzusprechen, gibt man am besten jedem Punkt eine sogenannte Koordinate. Dies sind zwei Werte (x und y), die die Nummer der senkrechten (x-Koordinate) und die der waagerechten Reihe (y-Koordinate) angeben, in denen sich der Punkt befindet. Damit ist jeder Punkt des Bildschirms eindeutig bestimmt. Um nun aus dieser Koordinate die Position des entsprechenden Bytes und Bits im Graphikspeicher zu berechnen, behilft man sich einer entsprechenden Formel, die im # 4.2 unter "Zeichnen eines Punktes" angeführt und erläutert wird (vielleicht versuchen Sie es interessehalber einmal selbst. Das Schema der Zeilenanfangsadressen des Graphikspeichers im Anhang sollte Ihnen dabei eine gute Hilfe sein).

Zum Schluß noch eine Bemerkung: Wie Sie vielleicht schon bemerkt haben, werden nicht alle Bytes der genannten 8 K für den Graphikspeicher verwendet, genauer gesagt nur 8000 ($1F40). Die restlichen 192 ($C0) Bytes können von Ihnen für andere Zwecke genutzt werden. Ähnliches gilt, wie Sie sehen werden, auch für die anderen Speicherbereiche

b) Farbaufbau:

Zu jedem hochauflösenden Bild, d.h. zu jedem Graphikspeicher, gehört auch ein Farbspeicher. Dieser wird durch den sogenannten Videoram dargestellt. Der Videoram dient normalerweise dazu, Text oder allgemein Zeichen, die durch Tastendruck auf dem Bildschirm erscheinen, zu speichern, damit der VIC sein Bild erstellen kann. Wenn Sie ihn also nicht verschieben und somit dort lassen, wo normalerweise der Text gespeichert ist, dann erscheinen die verschiedenen Zeichen des ursprünglichen Text-Bildschirms als kleine Farbquadrate auf dem Graphikbildschirm (Über die Verschiebemöglichkeiten von Videoram usw. gibt Ihnen # 3.3.2.3 Auskunft). Diesen Effekt können Sie bei dem folgenden Programm beobachten:

```
100 V = 53248 : REM BASISADRESSE VIDEORAM ($D000)
110 REM
120 REM ************
130 REM ** TEIL 1 **
140 REM ************
150 REM
160 REM GRAPHIK EINSCHALTEN:
170 POKE V+17, PEEK(V+17) OR 6*16 : REM BITS 5 UND 6
VIC-REGISTER 17 SETZEN
180 REM GRAPHIKSPEICHER NACH $2000 (8192) VERSCHIEBEN:
190 POKE V+24, PEEK(V+24) OR 8 : REM BIT 3 VIC-REGISTER 24
SETZEN
200 REM WAIT 198,255 : GOTO 220 : REM AUF TASTE WARTEN
210 END
220 REM
```

```
230 REM ************
240 REM ** TEIL 2 **
250 REM ************
260 REM
270 REM GRAPHIK AUSSCHALTEN:
280 POKE V+17, PEEK(V+17) AND 9*16+15 : REM BITS 5  UND  6
VIC-REGISTER 17 LOESCHEN
290 REM ZEICHENGENERATOR RUECKSETZEN:
300 POKE V+24, PEEK(V+24) AND  15*16  +  7  :  REM  BIT  3
VIC-REGISTER 24 LOESCHEN
310 END
```

Dieses Programm können Sie in zwei Variationen laufen lassen:

1.) so, wie es hier steht
2.) indem Sie das erste REM in Zeile 200 weglassen

Im ersten Fall endet das Programm sofort nach dem Umschalten und Sie können weitere Eingaben machen, die Sie nun allerdings nicht mehr normal sehen, sondern -wie gesagt- jeden Buchstaben als Farbquadrat. Wollen Sie wieder auf normalen Text zurückschalten, so geben Sie ein:

```
RUN 220
```

Im zweiten Fall wartet das Programm auf eine Taste Ihrerseits und schaltet dann die Graphik automatisch wieder aus (s. # 4.2)
In diesem Beispiel wird der Graphik - Farbbezug deutlich. Tatsächlich bestimmt ein Byte des Videoram die Farbe für ein 8x8-Punktefeld der HGR. In HGR kann dabei für jedes solches Kästchen sowohl die Hintergrundfarbe, also die Farbe der nicht gesetzten Punkte (oder Bits), und die sogenannte Punktfarbe, d.h. die Farbe der gesetzten Punkte (oder Bits) jeweils aus den 16 verschiedenen Farben gewählt werden. Dabei bestimmen in jedem Byte des Videoram die obersten 4 Bits die Punkt- und die unteren 4 die Hintergrundfarbe dieses Kästchens. Die Farbauflösung ist also weitaus geringer als die normale Graphik erlaubte. Dies war insofern notwendig, als es arg zu viel Speicher

verschlingen würde, wenn jedem der 64.000 Punkte eine eigene Farbe zugemessen werden könnte (zumal ein normaler Farbfernseher damit sowieso Probleme hätte). Sie bräuchten dafür 64.000 * 4 = 256.000 Bits, also 32 K RAM. Die Bearbeitungsgeschwindigkeit wäre ebenfalls erheblich herabgesetzt, was einen eigenen Graphikprozessor notwendig machen würde!
Der Videoram ist nun etwas einfacher organisiert, als der Graphikspeicher. Hier werden Byte für Byte und Zeile für Zeile nacheinander in den Speicher abgelegt. Der Aufbau sieht dann so aus:

	Spalte									
Zeile	0	1	2	3	4	5	6	7	...	39
0	$00	01	02	03	04	05	06	07	...	1D
1	$1E	1F	20	21	22	23	24	25	...	4F
2	$50	51	52	53	....					
...										
24	$3C0	3C1	...							

Im hochauflösenden Graphikbetrieb ist jedem Byte des Videoram also ein eindeutig bestimmtes 8x8-Feld zugeordnet. Hat man die Speicheradresse eines Punktes (abzüglich der Startadresse des Graphikspeichers), so braucht man diese lediglich durch 8 zu teilen und schon besitzt man die Adresse des korrespondierenden Videorambytes, zu der man nun nur noch die Startadresse des Videoram hinzuaddieren muß.

3.4.3. Multicolorgraphik (MC):

Bevor Sie sich diesem Abschnitt widmen, sollten Sie sich zunächst einmal mit dem letzten Paragraphen (# 3.4.2) beschäftigt haben, da das in jenem Teil des Buches vermittelte Wissen hier zum Teil vorausgesetzt wird.
Wie Sie dort gesehen haben, besitzt Ihr Commodore 64 eine recht hohe Graphikauflösung. Die Farbe kommt dabei jedoch (trotz 16 verschiedener Töne) etwas zu kurz. Um dieses speicherplatzbedingte Manko auszugleichen, haben sich die Konstrukteure entschlossen, einen zweiten Graphikmodus einzuführen, den eine größere Farb-, dafür allerdings eine niedrigere Graphikauflösung auszeichnet: Den Multicolor-modus.
Der Multicolormodus ermöglicht es, in einem 8x8-Block statt zwei, insgesamt 4 Farben gleichzeitig zu verwenden. Auch hier findet der 8 K-Graphikspeicher Verwendung. Nun werden allerdings jeweils 2 Bit jedes Bytes für die Bestimmung eines doppelt breiten Bildschirmpunktes benötigt. Die Auflösung beträgt demnach 160 doppelt breite Punkte in x-Richtung (doppelt breit deshalb, da ansonsten das Bildschirmfenster natürlich nur halb so groß wie normal wäre) und 200 Punkte in y-Richtung (Auflösung: 160x200).
Die Farbe stammt nun nicht mehr lediglich aus dem Videoram, sondern es werden gleichzeitig noch das Hintergrund-farbregister 0 des VIC und der Farbram hinzugezogen. Wir unterteilen diese Bereiche in 4 sogenannte Farbkanäle, die von 0 bis 3 durchnumeriert sind. Jedes Bitpaar, das ja für einen Punkt zuständig ist und damit Werte von 0-3 (%00-%11) annehmen kann, teilt dem VIC nun mit, aus welchem Kanal er die Farbe des Punktes beziehen soll (In HGR war es bekanntlich so, daß das eine zuständige Bit angab, ob die Farbe aus dem Hintergrundkanal oder dem Punktfarbkanal stammte). Im Graphikspeicher steht also nicht, welche Farbe (Farbcode) ein Punkt haben soll, sondern vielmehr, wo dieser eigentliche Farbcode steht. Die Bitpaar - Kanal - Speicher - Beziehung wird in dem angefügten Schema verdeutlicht:

Bitcode	Kanalnr.	Speicherbereich des Kanals
00	0	Register 33 des VIC
01	1	untere 4 Bits des Videoram

10	2	obere 4 Bits des Videoram
11	3	Farbram

Unter Bitcode verstehen wir hier die Binärzahl, die die zwei Bits darstellen, die jeweils für einen Punkt zuständig sind. Der Speicherbereich eines Kanals ist der Teil des Speichers, der durch einen Kanal bzw. durch eine Bitcodeeinstellung angesprochen wird.
Ein Beispiel: Im Graphikspeicher wird ein Punkt durch die zwei Bits mit den Belegungen 0 und 1 dargestellt. Der resultierende Bitcode %01 spricht den Kanal 1 an und damit die unteren 4 Bits des zuständigen Bytes des Videoram. Dieses zuständige Byte ermittelt man auf genau die gleiche Weise, wie unter # 3.4.2 (HGR) dargestellt.
Neu hierbei ist nun, daß die Farbe 3 bzw. der Kanal 3 aus dem Farbram stammt. Dieser Farbram ist normalerweise für die Farbe des im Videoram befindlichen Textes zuständig. Da der Farbram nicht verschiebbar ist, kommt es an dieser Stelle zwangsläufig zu Überschneidungen, wenn gleichzeitig Multicolor und Text verwendet werden. Auch hier läßt sich wieder die Adresse des für einen Punkt zuständigen Bytes des Farbram auf die unter HGR angegebene Weise bestimmen, da er genauso organisiert ist wie der Videoram.
Wie in HGR können diese Farben jeweils für ein 8x8- bzw. (wegen der halben Auflösung) 4x8-Punktefeld durch ein zuständiges Byte des Video- oder Farbram festgelegt werden. Dabei ergibt sich die Möglichkeit, jedem solchen Kästchen seine eigene Farbkombination zuzuweisen. Lediglich Kanal 0, also die Hintergrundfarbe stammt für die gesamte Graphik aus dem Hintergrundfarbregister 0 des VIC (Reg. 33) und ist damit für das ganze Bild einheitlich.
Wichtig ist bei der Erstellung von Multicolor - Graphiken, auf die Verzerrung in x-Richtung zu achten, die durch die doppelte Punktdicke zustande kommt.
Zum besseren Verständnis sei hier noch einmal ein Schema der Speicherstruktur des Graphikspeichers in Multicolor angeführt, das Ihnen das oben Gesagte noch einmal veranschaulichen soll:

		Spalte 0								Spalte 1								....
	Bit:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	usw.
Z	Byte 0	<->	<->	<->	<->					<->	<->	<->	<->					
e	Byte 1	<->	<->	<->	<->					<->	<->	<->	<->					
i	Byte 2	<->	<->	<->	<->					<->	<->							
l	Byte 3	<->	<->	<->	<->													
e	Byte 4	<->	<->	<->	<->													
	Byte 5	<->	<->	<->	<->													
0	Byte 6	<->	<->	<->	<->													
	Byte 7	<->	<->	<->	<->													
Z	Byte 320	<->	<->	<->	<->					<->	<->	<->	<->					
.	Byte 321	<->	<->	<->	<->					<->	<->							
1	Byte 322	<->	<->	<->	<->													
	usw.																	

In diesem Schema stellen die "<->" jeweils den doppelt breiten Punkt dar, der auf dem Bildschirm erscheint. An dieser Stelle möchte ich Sie noch einmal auf die unterschiedliche Belegung des Videoram bei HGR und MC hinweisen. Zugegebenermaßen wird die Handhabung der Graphik durch diese schwer durchschaubaren Verhältnisse und Unterschiede nicht gerade vereinfacht. Auch scheint mir die Aufteilung der Farbauflösung nicht gerade gelungen, da sich hierdurch, wie wir noch im 4. Kapitel sehen werden, einige Probleme ergeben. Trotzdem läßt sich doch Einiges mit ihr erreichen. Sie werden sehen, schöne Effekte sind an der Tagesordnung.

3.5. Sprites

Eines der hervorstechendsten Merkmale Ihres Commodore 64 sind natürlich die Sprites. Sprites sind eigenständige kleine Graphiken, die unabhängig voneinander und von dem übrigen Bildschirminhalt in dem Text- oder Graphikfenster bewegt werden können. Insgesamt haben Sie die Möglichkeit, 8 Sprites gleichzeitig auf den Bildschirm zu bringen.
Sprites können bezüglich Ihrer Farbe, Ihrer Größe und der Priorität vor den Hintergrundzeichen und auch gegeneinander variiert werden. Sie können Kollisionen zwischen Sprites untereinander und mit dem Hintergrund feststellen. Zudem besitzen Sie auch hier die Möglichkeit, zwischen den beiden Spritemodi

- normal
- Multicolor

zu wählen, wobei Sie dies bei jedem einzelnen Sprite getrennt bestimmen können. All diese Funktionen können sehr leicht mit Hilfe des VIC (Videocontroller 6567) und seinen Registern realisiert werden. Zunächst aber wollen wir uns einmal mit dem Aufbau der Sprites beschäftigen. Da wir es dabei in besonderem Maße mit der Binärarithmetik und den verschiedenen Registern Ihres Computers zu tun haben werden, sollten Sie sich vorher diese Kapitel (## 2 und 3.1) zu Gemüte führen oder, falls Sie diese noch nicht richtig verstanden haben, noch einmal konzentriert lesen.
Während der Erörterung des Spriteaufbaus sollten Sie zwei Dinge stets im Kopf behalten:
Sie können (wie gesagt) gleichzeitig insgesamt 8 verschiedene Sprites auf dem Bildschirm darstellen. Jedem Sprite ist eine spezifische Nummer (0-7) zugeordnet, die Sie durch das gesamte Kapitel begleiten wird.

3.5.1. Aufbau und Farbe normaler Sprites

Jedes normale Sprite besteht aus 504 Punkten, die Sie einzeln setzen oder löschen können. Verwendet wird dabei eine 24x21-Punktematrix, d.h. ein Sprite ist 24 Punkte breit und 21 Punkte hoch. Innerhalb dieses Bereiches können Sie nun die unterschiedlichsten Graphiken oder Figuren erstellen.
Um die Definition, d.h. das Aussehen unserer Sprites zu speichern und dem VIC mitzuteilen, bedarf es insgesamt 504/8 = 63 Bytes, da jeder einzelne Punkt als ein Bit abgelegt wird und ein Byte -wie Sie wissen- 8 Bits umfaßt. Da jedes Sprite eine Breite von 24 Punkten besitzt, passen in eine Reihe genau 24/8 = 3 Bytes hinein. D.h. die ersten drei Bytes bestimmen die 24 Punkte der ersten Reihe. Dementsprechend wird mit der zweiten, dritten, vierten usw. Reihe verfahren. Jeweils 3 hintereinanderfolgende Bytes legen eine Reihe fest. Die nächste Reihe beginnt dann ebenfalls mit dem nächsten Byte. Wir können diesen Sachverhalt in einer kleinen Skizze darlegen:

		Spalte 0	Spalte 1	Spalte 2
Reihe/Bit:		7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0	Byte 0	. . . . .	. . . . .	. . . . .
1	Byte 3	. . . . .	. . . . .	. . . . .
2	Byte 6	. . . . .	. . . . .	. . . . .
3	Byte 9	. . . . .	. . . . .	. . . . .
4	Byte 12	. . . . .	. . . . .	. . . . .
			usw.	
20	Byte 60	. . . . .	. . . . .	. . . . .

Wollen Sie also ein Sprite eingeben, so geben Sie die zuständigen Bytes Spalte für Spalte und Reihe für Reihe nacheinander ein. Wie dies in Basic realisiert wird, erfahren Sie in dem großen Kapitel 4.
Sie können den insgesamt 8 verschiedenen Sprites, die Sie gleichzeitig auf den Bildschirm bringen können, jeweils unterschiedliche der zur Verfügung stehenden 16 Farben zuordnen. Dabei erhalten alle gesetzten Punkte die Farbe aus dem für das jeweilige Sprite zuständigen Spritefarbregister (VIC-Register 39-46). Wollen Sie also Sprite 5 weiß zeichnen, so belegen Sie das VIC-Register 39+5 = 44, also die Speicher-

stelle $D02C (53292) mit dem Wert 1 (für weiß). Alle nicht gesetzten Punkte wirken transparent d.h. durchsichtig und sind daher nicht zu sehen.

3.5.2 Aufbau und Farbe eines Multicolorsprites

Nicht jedes Sprite besitzt diesen 24x21-Punkte-Aufbau. Sie können jeweils zwischen hochauflösenden und sogenannten Multicolor - Sprites wählen. Erstere besitzen die gerade geschilderte Matrix und die dazugehörige Farbgebung. Die Multicolor - Sprites hingegen werden ähnlich der Multicolor - Graphik gebildet. Aus diesem Grunde besitzt ein solches Sprite in x-Richtung die halbe Auflösung. Hier befinden sich also nur 12, jedoch doppelt breite Punkte in einer Zeile. Dafür aber kann ein Gebilde aus insgesamt vier verschiedenen Farben (mit der Hintergrundfarbe) zusammengesetzt sein, während -wie gesagt- ein hochauflösendes (HGR-) Sprite nur zwei Farben beinhaltet. Diese vier Farben sind vergleichbar mit den Kanälen der MC-Graphik und in den verschiedenen VIC-Registern untergebracht.
Um für jeden der 12x21 = 252 Punkte eines MC-Sprites den Farbkanal zu bestimmen, aus dem die Farbe dieses doppelt breiten Punktes stammen soll, werden jeweils 2 Bits (Bitcode) verwendet, die bekanntlich die Werte 0-3 (%00-%11) annehmen können und damit die Nr. des Kanals festlegen. Der VIC holt sich dann aus diesem Kanal die Farbe des Punktes. Die Zuordnung der Kanäle zu den einzelnen Bitcodes demonstriert die folgende Tabelle:

Kanalnr.	Bitcode	Farbspeicher
0	00	durchsichtig
1	01	Multicolor Reg. 0 (VIC-Reg 37)
2	10	Multicolor Reg. 1 (VIC-Reg 38)
3	11	Sprite Color Reg. (Reg. 39-46)

Wie Sie sehen, kann lediglich die Farbe des Kanals 3 für alle 8 Sprites unterschiedlich sein, da diese für jedes Sprite in einem eigenen Register steht. Die anderen Farben (Farben 1 und 2 neben der Hintergrundfarbe) sind jeweils für alle Sprites gleich, da sie aus identischen Registern gewonnen

werden. Entgegen den Angaben des CBM 64 Benutzerhandbuchs können auch in Multicolor sämtliche 16 Farben zur Erstellung Ihrer Figuren verwendet werden.
Um ein Sprite in Multicolor auf dem Bildschirm erscheinen zu lassen, ist es notwendig, dem VIC diese Absicht in einem speziellen Register mitzuteilen. Es ist dies das Register 28 ($1C), in dem jedem Bit des 8 Bit großen Bytes eins der 8 Sprites zugeordnet ist. Ist hier ein Bit gesetzt, so wird von nun an das entsprechende Sprite zu einem Multicolor - Sprite. Die Bitzuordnung dieses Registers ist die folgende:

Bit:	b7	b6	b5	b4	b3	b2	b1	b0
Wert:	128	64	32	16	8	4	2	1
Sprite:	s7	s6	s5	s4	s3	s2	s1	s0

Wollen Sie z.B. Sprite 4 als Multicolorsprite verwenden, so setzen Sie in dem Register 28 des VIC gleichfalls das Byte 4, d.h. Sie POKEn, sofern Sie von Basic aus hantieren, wie folgt:

```
POKE 53248+28, 16
```

Wollen Sie mehrere Sprites derart darstellen (z.B. Sprites 1, 5 und 7), so geben Sie beispielsweise ein:

```
POKE 53248+28, 1 OR 32 OR 128  oder:
POKE 53248+28, 1 + 32 + 128
```

(Ausnahmsweise kann der OR-Befehl auch durch eine Addition ersetzt werden) Der Aufbau eines MC-Sprites gleicht ansonsten einem normalen Sprite. Auch jenes wird in 63 Bytes abgelegt. Zur Veranschaulichung ein entsprechendes Diagramm:

		Spalte 0								Spalte 1								Spalte 2							
Reihe/Bit:		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	Byte 0	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				
1	Byte 3	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				
2	Byte 6	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				
3	Byte 9	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				
4	Byte 12	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				
										usw.															
20	Byte 60	<->	<->	<->	<->					<->	<->	<->	<->					<->	<->	<->	<->				

Wie Sie sich vielleicht denken können, stellt hier ein "<->" jeweils einen doppelt breiten Punkt dar, der durch ein Bitpaar codiert wird.

3.5.3. Spritedefinition - Farbe

Wollen Sie nun ein Sprite auf den Bildschirm bringen, so haben Sie zunächst einige Dinge zu beachten. Als erstes sollten Sie sich natürlich erst einmal Gedanken über das Aussehen ihres Objektes machen und entsprechend die 63 notwendigen Bytes bereitstellen. Dazu werden Ihnen in Kapitel 4.3 einige Hilfen und Tips gegeben (u.a. ein sehr komfortabler Spriteeditor).
Als Nächstes sollten Sie sich überlegen, wo in Ihrem Speicher Platz für diese 63 Bytes vorhanden ist. Dabei müssen Sie selbstverständlich das unter # 3.3.2 Gesagte mit berücksichtigen. In der Einschaltkonfiguration z.B. gibt es nur relativ wenige Möglichkeiten, Sprites unterzubringen. Hier ist Platz für lediglich 4 verschiedene Spritedefinitionen. Wollen Sie noch mehr unterbringen, so müssen Sie schon entweder einfach den Basicanfang (normal bei $0801 = 2049) verschieben (durch UmPOKEn der Speicheradressen $2B/$2C (43/44)), wobei Sie allerdings beachten sollten, daß dies vor dem Einspeichern oder Einladen eines Basicprogrammes geschehen muß, oder Sie verlegen den Videoram und haben den alten Videoramspeicher von $400-$7FF zur freien Verfügung, was jedoch bei der gleichzeitigen Textanzeige ein UmPOKEn einer Speicherstelle notwendig macht, die das Highbyte des Videorambeginns enthält (normal: $04; das Highbyte wird dezimal errechnet durch: INT(Videoramstart / 256). Diese zu

veränderne Speicherstelle hat die Adresse $288 = 648. Haben Sie z.B. den Videoram nach $0800 (= 2048) verlegt (was nebenbei ebenfalls eine Verlegung des Basic-Starts notwendig macht) und wollen Sie dort trotzdem Text darstellen, so geben Sie ein:

```
POKE 648, INT(2048/256)
```

Nach diesem UmPOKEn muß dann unbedingt ein <shift><clr/home> bzw. ein PRINT CHR$(147) zum Löschen des Bildschirms folgen. Doch in den meisten Fällen kommen Sie mit 4 verschiedenen Spritedefinitionen aus, da Sie für zwei gleich aussehende Sprites keine neue Definition abspeichern brauchen, wie Sie gleich sehen werden.
Um dem VIC zu ermöglichen, die von Ihnen abgelegte 63-Byte Definition zu finden und zu lesen, müssen Sie den 16 K-Adressierungsbereich des VIC in 256 Blöcke mit je 64 Bytes unterteilen. Diese Blöcke werden von 0-255 durchnumeriert. Nach dem Einschalten hätten die Blöcke die folgenden Startadressen (bei einer Verschiebung des gesamten VIC-Adressraumes durch Ändern der Bits 0/1 von Register 0 der CIA 2 (s.o.) muß hier natürlich die neue Basisadresse hinzuaddiert werden):

Block	Startadresse
0	$0000 - 0
1	$0040 - 64
2	$0080 - 128
3	$00C0 - 192
4	$0100 - 256
usw.	
255	$3FC0 - 16320

In jedem solchen Block kann nun eine einzige Spritedefinition untergebracht werden. Dabei hat das letzte Byte des Blockes keine Bedeutung, da ja nur 63 Bytes für ein Sprite benötigt werden. In der normalen Einstellung stehen Ihnen jedoch lediglich die Blöcke:

Block	Adressbereich
11	$02C0-$02FE (704- 766)
13	$0340-$037E (832- 894)
14	$0380-$03BE (896- 958)
15	$03C0-$03FE (960-1022)

zur freien Verfügung, wobei jedoch angemerkt werden muß, daß die letzten 3 Bereiche sich mit dem Bandpuffer überschneiden, bei dem Arbeiten mit der Datasette also gelöscht werden. Dann beginnt die Möglichkeit des Spritegebrauchs erst wieder bei $2000 (8192), also Block 128 (Vorsicht bei langen Basic-programmen und großem Speicherbedarf!), da der Bereich von $1000-$1FFF (4096-8192) bekanntlich dem Sonderstatus unterliegt (s. # 3.3.2.2) und daher nicht benutzbar ist. Bei einer Verschiebung des 16 K-Adressbereiches gelten natürlich evt. andere Beschränkungen.

Was fangen wir aber mit diesen Dingen an?
Um dem VIC jetzt mitzuteilen, in welchem Block er denn die von Ihnen abgelegte Spritedefinition findet, müssen Sie diese Blocknummer in eins der 8 letzten Bytes des Videoram legen (s. # 3.1). Sie liegen in der Einschaltkonfiguration bei $07F8-$07FF (2040-2047).
Wenn Sie einmal nachrechnen, wieviel Bytes eigentlich für die Speicherung eines Text - Bildschirminhalts benötigt werden, so kommen Sie auf lediglich 40x25 = 1000. Der Videoram umfaßt aber genau 1 K, also 1024 Bytes. Die restlichen 24 Bytes werden normalerweise nicht gebraucht und können von Ihnen frei verwendet werden, bis auf die letzten 8 Bytes. Sie werden eben für den gerade genannten Zweck benötigt. Jedem Byte ist dabei ein Sprite in der folgenden Weise zugeordnet (die Adressen gelten für die Position des Videoram nach dem Einschalten und verschieben sich natürlich in dem Falle einer Änderung der Videoramadresse mit diesem):

Register :	$07F8	07F9	07FA	07FB	07FC	07FD	07FE	07FF
	2040	2041	2042	2043	2044	2045	2046	2047
Spritenr.:	0	1	2	3	4	5	6	7

Ein Beispiel: Angenommen, Sie haben ein Sprite in den Bereich von $03C0-$03FE (960-1022), also in Block 15 gelegt. Jetzt

wollen Sie, daß sowohl Sprite Nr. 2 wie auch Sprite Nr. 6 so aussieht, wie Sie es in Block 15 definiert haben. In diesem Falle schreiben Sie mittels:

```
POKE 2040+2, 15 : POKE 2040+6, 15
```

den Wert 15 als Zeiger auf Block 15 in die entsprechenden Register ein. Sie sehen, daß auf diese Weise mehrere Sprites das gleiche Aussehen haben können, indem Sie einfach die Zeiger auf den gleichen Block setzen.
Haben Sie nun in Block 14 ein weiteres Sprite definiert und wollen z.B. das Sprite Nr. 2 in seinem Aussehen ändern, so genügt lediglich ein

```
POKE 2040+2, 14
```

um damit schlagartig die Definition zu wechseln (s. # 4.3).

Wir wollen jetzt die beiden Sprites, die wir soeben definiert haben auch auf dem Bildschirm erscheinen lassen. Dazu müssen wir sie jedoch zunächst einmal einschalten. Das VIC-Register 24, "Sprite ein/aus", übernimmt diese Funktion. Jedem der 8 Bits des Registers ist ein Sprite in der gleichen Weise zugeordnet, wie uns dies schon von der Multicolorwahl (Reg. 28) her bekannt ist:

Bit:	b7	b6	b5	b4	b3	b2	b1	b0
Wert:	128	64	32	16	8	4	2	1
Sprite:	s7	s6	s5	s4	s3	s2	s1	s0

In unserem Fall der Sprites 2 und 6 müssen wir also eintippen:

```
POKE 53248+24, 64 OR 4          oder
POKE 53248+24, 64 + 4
```

Doch damit brauchen die Sprites noch nicht sichtbar zu sein, da sie meist erst in den Bildschirm hineinverschoben werden müssen. Um Näheres über die Programmierung der Sprites zu erfahren, sehen Sie bitte unter Kapitel 4 nach.

3.5.4. Weitere Spriteeigenschaften

Doch mit dem einfachen Definieren, der Farbwahl und dem Einschalten haben wir noch längst nicht alles ausgeschöpft, was uns an Gestaltung und Veränderung der Sprites zur Verfügung steht. Die folgenden Zeilen zeigen Ihnen, was die Sprites erst zu DEN Sprites macht.

3.5.4.1. Positionieren

Die erste Möglichkeit der Variation und wohl auch die wichtigste ist die Wahl der jeweiligen Bildschirmposition jedes einzelnen Sprites. Sie können also bestimmen, wo auf Ihrer Mattscheibe Ihre Figuren zum Stehen kommen sollen. Dies ist besonders wichtig, da dadurch Bewegungen und schöne Effekte erzeugt werden können, wie es im 4. Kapitel dargelegt wird.
Dabei unterteilt man den Bildschirm in sogenannte Koordinaten x und y, wie es uns bereits von der Graphik her bekannt ist. Dabei ist allerdings folgendes zu beachten:
Die Spritekoordinaten stellen stets die Position der unteren linken Ecke eines Sprites dar.
Die Spritebewegung besitzt eine Auflösung von 512x256 Punkten, also weit mehr, als auf dem Bildschirm darstellbar. Das Raster, also der Abstand bzw. die Größe der Punkte, ist dabei identisch mit dem der hochauflösenden Graphik. Es sind also 320 Punkte in x-Richtung und 200 in y-Richtung zu sehen. Damit wird es Ihnen aber möglich, die Sprites jeweils bei Bewegungen aus dem Bildschirm hinausfahren zu lassen; am verdeckenden Rand sehen Sie also einen stets kleiner werdenden Teil Ihres Sprites (s. Kapitel 4).
Der Nullpunkt der Spritekoordinaten liegt weit außerhalb des Text- oder Graphikfensters oben links in der Ecke. Der erste sichtbare Punkt dieses Koordinatenrasters und damit der Nullpunkt der normalen Graphik besitzt schon die Koordinaten x=20 und y=30. Hier erst ist das Sprite also vollständig zu sehen. Um damit von Sprite- auf Graphikkoordinaten umzurechnen, müssen Sie bei ersteren stets 20 vom x- und 30 vom y-Wert abziehen (umgekehrt: Graphik- in Spritekoordinaten umrechnen durch hinzuaddieren dieser Größen). Bei

x=320+20=340 und y=200+30=220 ist das Sprite nicht mehr zu sehen.
Um dem VIC nun mitzuteilen, wo er welches der 8 Sprites auf dem Bildschirm unterbringen soll, stehen Ihnen seine ersten 17 Register (von 0 bis 16) zur Verfügung. Das 16. Register nimmt dabei eine Sonderstellung ein und wird etwas weiter unten besprochen. Die 16 zuständigen Speicheradressen sind jeweils paarweise den 8 Sprites zugeordnet. Das erste Element dieser Registerpaare gibt dabei die x-, das zweite die y-Koordinate des entsprechenden Sprites an:

Sprite :	s0	s1	s2	s3	s4	s5	s6	s7
x-K. Reg.:	0	2	4	6	8	10	12	14
y-K. Reg.:	1	3	5	7	9	11	13	15

Wollen wir also beispielsweise Sprite 6 auf die Koordinaten x=100, y=150 setzen, so brauchen wir lediglich einzutippen:

```
POKE 53248 + 2*6     , 100
POKE 53248 + 2*6 + 1, 150
```

und schon kommt unser vorher definiertes Sprite dort zu stehen. Wie Sie aber vielleicht bereits gemerkt haben, können wir von den oben genannten 512 Punkten der Spritebewegungsauflösung lediglich 256 Punkte erreichen (ein Byte kann maximal 256 verschiedene Werte annehmen). Aus diesem Grunde mußte noch ein weiteres Register eingerichtet werden, das für das oberste 8. Bit (MSB = Most Significant Bit = höchstwertiges Bit) der x-Koordinate jedes Sprites zuständig ist: Register 16. In diesem Byte ist wieder jedem Sprite ein Bit zugeordnet, das die gesuchte Information beinhaltet:

Bit:	b7	b6	b5	b4	b3	b2	b1	b0
Wert:	128	64	32	16	8	4	2	1
Sprite:	s7	s6	s5	s4	s3	s2	s1	s0

Wollen wir also unser Sprite auch über die Koordinate x=255 hinaus auf den Bildschirm bringen, so ist das entsprechende Bit dieses Registers zu setzen. Zu dem Wert im regulären x-Koordinatenregister ist dann 256 hinzu zu zählen.

3.5.4.2. Vergrößerung

So schaut unser Sprite ja schon recht hübsch aus - wir geben uns voll zufrieden - doch Ihr Rechner noch nicht. Er bietet Ihnen einige schöne weitere Kleinigkeiten, die Ihr Herz erfreuen sollen und werden. In dem Registerschatz des Videocontrollers befinden sich nämlich u.a. noch zwei bisher nicht besprochene Adressen. Dies sind die Register 23 und 29. Mithilfe dieser beiden Bytes können Sie Ihr Sprite vergrößern. Dabei ist das erste der zwei hier genannten für eine Vergrößerung in y-, also eine Längsdehnung, das zweite für eine Vergrößerung in x-Richtung, also eine Vertikaldehnung, zuständig. Beidesmal ist der Streckungsfaktor gleich 2, d.h. jeder Punkt eines Sprites wird doppelt so hoch bzw. breit. Sie können beide Möglichkeiten getrennt oder gemeinsam (also sowohl Vergrößerung in x- als auch in y-Richtung) anwenden, was jeweils verschiedene Effekte mit sich bringt. Der Aufbau der beiden Register dürfte Ihnen inzwischen bekannt vorkommen und ist in den jeweiligen Diagrammen z.B. unter Positionierung nachzuschauen, da auch hier jedem Sprite ein Bit zugeordnet ist. Ist das entsprechende Bit gelöscht, so wird nicht, ist es gesetzt, so wird vergrößert. Die Verhältnisse können etwa so dargestellt werden:

Vergrößerung	Vergrößerungsfaktor	Punktmatrix
keine	1x1	24x21
x-Richtung	2x1	48x21
y-Richtung	1x2	24x42
x/y-Richtung	2x2	48x42

Unter Punktmatrix ist hierbei natürlich die Matrix gemeint, die auf dem Bildschirm erscheint (sie ist ja auch bei Multicolor - Sprites identisch), also die Anzahl der Punkte des Bildschirms, die von einem Sprite maximal überdeckt werden. Zu bemerken ist weiterhin, daß ein vergrößertes Sprite nicht mehr vollständig am linken oder oberen Rand (oder an beiden) verschwindet, auch wenn die Spritekoordinaten gleich null werden.

3.5.4.3. Priorität

Was passiert nun aber, wenn sich zwei eingeschaltete Sprites oder ein Sprite und z.B. ein Buchstabe überlappen? Überdecken Sie sich und wenn ja, wer verdeckt wen? Dies soll in diesem Abschnitt geklärt werden.

a) Sprite-Sprite-Überlappung:
In diesem Fall ist die Sache denkbar einfach: Den einzelnen Sprites sind bekanntlich Nummern von 0-7 zugeordnet. Überlappen sich jetzt zwei Sprites, so wird dasjenige Sprite "über" dem anderen liegen, es also verdecken, welches die niedrigere Nummer besitzt. D.h. das Sprite z.B. mit der Nummer 0 wird ein Sprite mit der Nummer 5 an den Stellen verdecken, an denen Sie sich überlappen (genau genommen wird das Sprite 0 Sprite 5 nur dort überdecken, wo es nicht transparent (durchsichtig) ist (s. Spritedefinition)).

b) Sprite-Hintergrundzeichen-Überlappung
Bei einer Überlappung eines Hintergrundzeichens mit einem Sprite wird die Sache schon komplizierter, aber auch interessanter. Zunächst aber eine Begriffserklärung: Im folgenden sind unter Hintergrundzeichen stets irgendwelche gesetzten Punkte verstanden, sei es z.B. ein Buchstabe, ein Sonderzeichen oder Graphik.
Wir können, so ist es eingerichtet, hierbei selber wählen, ob ein Sprite von diesen Hintergrundzeichen überdeckt wird, das Sprite sich also praktisch hinter den verschiedenen Objekten des Bildschirms befindet, oder ob es diese selbst verdeckt, also vor Ihnen steht. Für diese Funktion existiert ein weiteres Register des VIC, Register 27. Der Aufbau ist Ihnen wohl inzwischen geläufig, er entspricht dem der Register 16, 21, 23, 28 und 29. Auch hier ist jedem Bit ein Sprite zugeordnet. Ist nun ein Bit gelöscht, was nach dem Einschalten (wie in # 3.1 ersichtlich) der Fall ist, so erscheint das jeweilige Sprite vor den übrigen Zeichen, ist es dagegen gesetzt, so überdecken alle Hintergrundzeichen das betreffende Sprite.

Mit Hilfe dieser wertvollen Eigenschaften ist es möglich,

3-dimensionale Graphik oder Bewegungen darzustellen. In Kapitel 4 wird Ihnen einiges dazu gesagt.

3.5.4.4. Kollisionen

Besonders für Spiele eine unschätzbare Einrichtung: Die Feststellung von Kollisionen bzw. Berührungen zwischen Sprites untereinander und mit Hintergrundzeichen wird Ihnen durch verschiedene andere Register sehr einfach gemacht. Hierfür sind u.a. die Speicherstellen 30 und 31 des VIC zuständig.
In der ersten dieser beiden wird automatisch registriert, wenn sich zwei Sprites im Laufe der Zeit einmal berühren (Überlappung). Dabei ist jedes Bit dieses Registers für eins der acht Sprites zuständig (s.o.). Berühren sich nun zwei Sprites, so werden hier die beiden korrespondierenden Bits gesetzt. Kollidieren also Sprite 6 und 2, so lautet der Inhalt des Registers: %0100 0010. Dieser Inhalt bleibt solange bestehen, bis er (als Zeichen dafür, daß er abgefragt wurde) wieder vom Programmierer z.B. durch

```
POKE 53248+30, 0
```

gelöscht wird. Gleichzeitig mit diesen beiden Bits wird noch ein anderes gesetzt. Es ist dies das Bit 2 des VIC-Registers 25 (IRR), welches uns noch völlig unbekannt ist und auch erst im # 3.7 erläutert wird. Soviel sei gesagt: Falls von Register 26 erlaubt, kann hier also durch eine Kollision ein IRQ (Interrupt Request) ausgelöst werden.
Das zweite Register mit der Nummer 31, das uns in diesem Zusammenhang interessiert, ist für den Vermerk einer Kollision eines Hintergrundzeichens mit einem Sprite zuständig. Findet ein solches Ereignis demnach statt, so wird hier das dem jeweiligen kollidierten Sprite zugeordnete Bit gesetzt (wie Register 30). Berührt z.B. Sprite 2 einen gesetzten Punkt des Bildschirms, so steht hier: %0000 0010. Auch dieses Register muß nach der Abfrage auf dieselbe Art und Weise wieder gelöscht werden wie eben beschrieben. Und auch hier wird in dem Register 25 diesmal das Bit 1 gesetzt, um bei Bedarf einen IRQ auszulösen.

3.6. Text/Zeichensatz

3.6.1. Textseitenorganisation

Damit sich der Rechner alle Ausgaben, die auf dem Bildschirm stehen, merken kann (er muß dieses Bild auf Ihrem Fernseher schließlich alle 1/20 Sekunden selbst erstellen (s. # 3.7)), legt er sämtliche Zeichen, die Sie im normalen Textmodus (z.B. nach dem Einschalten) durch Tastendruck eingeben, in den uns sicher schon bekannten Videoram ab. Dieser umfaßt etwa 1 K (in Wahrheit nur 40x25 = 1000 Bytes) und geht im Normalzustand von der Speicherstelle $0400 (1024) bis hin zu $07FF (2047). Über die Verschiebemöglichkeiten gibt Ihnen # 3.3.2 Auskunft. In der hochauflösenden und der Multicolor Graphik wird dieser Speicher für die Beherbergung der Farbe verwendet.
Den einzelnen Zeichen werden jeweils bestimmte Codes zugeordnet und in den Videoram abgelegt, wenn dieses Zeichen an einer bestimmten Stelle auf dem Bildschirm erscheinen soll. Die Zuteilung von Codes kennen Sie sicher bereits von der sogenannten ASCII - Codierung. Die Bildschirmcodes jedoch werden nach einem anderen System gebildet. Während Die CBM - ASCII - Tabelle, wie Sie sie im Anhang ihres Bedienungshandbuches finden, manchmal verschiedenen Werten gleiche Zeichen zuordnet und gleichzeitig sogenannte Controlcodes vorhanden sind, die auf dem Bildschirm kein Zeichen erbringen, sind die Bildschirmcodes eindeutig und ohne Lücken verteilt, da Sie neben sämtlichen normalen Zeichen gleichfalls noch die inversen Zeichen unterscheidbar machen müssen - wie anders sollte der Rechner anhand eines Codes anders wissen, ob er nun ein Zeichen normal oder invers darstellen soll. Wie Sie wissen wird dies von der Tastatur aus durch die Umschaltung mittels zweier Controlcodes (<rvs on> und <rvs off> = CHR$(18) und CHR$(146)) bewerkstelligt. Eine Tabelle der Bildschirmcodes finden Sie im Anhang. Addieren Sie 128 jeweils zu den einzelnen Werten hinzu, so erhalten Sie das gleiche Zeichen in inverser Form.

3.6.1.1. Normaler Text

Neben den Zeichen muß aber für jedes Zeichen gleichfalls die Zeichenfarbe gespeichert werden, da ja bekanntlich rein theoretisch jedes Zeichen eine andere Farbe besitzen kann. Hierfür existiert ein weiterer sogenannter Farbram mit der gleichen Größe wie der Videoram, der die notwendigen Informationen enthält. Dieser Farbram liegt bei $D800-$DFFF (55296-56295) und wird ebenfalls in der Multicolor - Graphik als Farbspeicher verwendet. Jedes Byte dieses Bereiches bestimmt die Farbe des dazugehörigen Zeichens des Videoram, dessen Aufbau identisch ist.
Die Hintergrundfarbe des Textbildes wird dagegen durch ein einziges Register des VIC angesprochen. Dieses Register (Register 33) liegt in der Speicherstelle 53248+33 und kann z.B. mit

```
POKE 53248+33,0 : REM HINTERGRUNDFARBE = SCHWARZ
```

verändert werden.

3.6.1.2. Multicolor - Modus

Für den Multicolor - Modus der Zeichendarstellung schauen Sie bitte unter # 3.2 nach, wo dieser entsprechend beschrieben ist.

3.6.1.3. Extended Colour - Modus

Ihr Commodore 64 besitzt neben dem gerade beschriebenen normalen Textmodus mit einer Hintergrundfarbe für alle Zeichen einen weiteren, in dem Sie für jedes Zeichen eine andere Hintergrundfarbe wählen können (jeweils 4 Hintergrundfarbregister, also 4 frei wählbare Hintergrundfarben stehen zur Verfügung). Diese Darstellungsart heißt: extended Colour - Modus.
Wie gesagt, stehen Ihnen hier für jedes Zeichen eine von 4 Hintergrundfarben zur Verfügung, die Sie durch EinPOKEn der jeweiligen Werte in die folgenden Register verändern können:

Hintergrundfarbe 0: VIC-Register 33
Hintergrundfarbe 1: VIC-Register 34
Hintergrundfarbe 2: VIC-Register 35
Hintergrundfarbe 3: VIC-Register 36

Wie Sie sehen, entspricht das Farbregister 0 dem normalen Register zur Festlegung der Hintergrundfarbe. In diese Adressen legen Sie dann natürlich den jeweiligen Farbcode, den Sie dem Anhang entnehmen können (0-15).
Um den Extended Colour - Modus einzuschalten, müssen Sie lediglich etwa durch

```
POKE 53248+17, PEEK(53248+17) OR 4*16
```

das 6. Bit des VIC-Registers 17 ($11) setzen. Durch

```
POKE 53248+17, PEEK(53248+17) AND 256 - 4*16
```

wird es wieder gelöscht.
Wollen Sie nun festlegen, welche dieser Hintergrundfarben die einzelnen Zeichen schließlich besitzen, so müssen Sie folgendes wissen:
Die oberen 2 Bits jedes Bytes aus dem Videoram, der eigentlich (wie wir soeben erfahren haben) lediglich die verschiedenen Zeichen des Textfensters speichert, legen dies jeweils für jedes Zeichen fest. Da aber diese Bits normalerweise ebenfalls dazu verwendet werden, um die verschiedenen Zeichen zu codieren, stehen Ihnen im extended Colour - Modus (ECM) nur 64 Zeichen zur Verfügung.
Alle Graphikzeichen und im Kleinschrift/Großschrift - Modus ebenfalls die Großbuchstaben, sowie alle inversen Zeichen fallen dem zum Opfer. Steuern Sie diese trotzdem an, so wird eines der erlaubten 64 Zeichen mit einer anderen Hintergrundfarbe erscheinen. Welche Zeichen davon wie betroffen sind, können Sie am besten der Tabelle der Bildschirmcodes im Anhang entnehmen. Dabei gilt folgende Zuordnung:

MSBs	Bildschirmcodes	Hintergrundfarbe
00	000-063/$00-$3F	HF 0
01	064-127/$40-$7F	HF 1
10	128-191/$80-$BF	HF 2
11	192-255/$C0-$FF	HF 3

Unter MSBs verstehen wir hier die beiden obersten Bits des Videoram, also Bits 6 und 7 (MSB = Most Significant Bit). Vergleichen Sie diese Tabelle mit der angegebenen Bildschirmcodetabelle im Anhang.
Ein Beispiel: Sie wollen ein B mit der Hintergrundfarbe 7 = gelb auf den Bildschirm bringen. Dafür belegen Sie das gewünschte Hintergrundfarbregister (hier 1) mit dem Wert 7 für gelb und POKEn eine 2 für B zuzüglich 64 für die Ansteuerung des Registers 1 an die entsprechende Stelle im Bildschirmspeicher oder geben mittels PRINT-Statement das Zeichen <shift><B>, also CHR$(98) auf dem Bildschirm aus. Im Programm sähe dies dann so aus:

```
10 V = 53248 : REM VIC-REG-BASISADRESSE
20 POKE V+17, PEEK(V+17) OR 4*16 : REM ECM EINSCHALTEN
30 POKE V+34, 7 : REM HINTERGRUNDFARBE 1 = GELB
40 POKE 1024, 2+64 : REM ZEICHEN OBEN LINKS IN DIE ECKE
```

oder:

```
10 V = 53248 : REM VIC-REG-BASISADRESSE
20 POKE V+17, PEEK(V+17) OR 4*16 : REM ECM EINSCHALTEN
30 POKE V+34, 7 : REM HINTERGRUNDFARBE 1 = GELB
40 PRINT CHR$(98) : REM ZEICHEN AN DIE CURSORPOSITION
```

Nach Ablauf dieser Programme befinden Sie sich weiterhin in diesem Modus und können ein wenig herumprobieren. Wie Sie wieder herauskommen, wissen Sie bereits (s.o.).

3.6.2. Zeichensatzorganisation

Neben den ungewöhnlich variationsreichen Graphikmöglichkeiten bietet Ihnen Ihr Commodore 64 noch weitere Kostbarkeiten. Eine dieser Fähigkeiten ist die softwaremäßige Veränderung des Zeichensatzes. Sie ist die Grundlage fast aller Spiele und ist dasjenige Mittel (neben den Sprites), das alle Spiele auf dem CBM 64 so unheimlich schnell und trotzdem graphik- und effektreich werden läßt. Ohne diese Möglichkeit ist eine vernünftige Spielprogrammierung undenkbar geworden. Wo sich andere Computer mit riesigen Graphikspeichern herumquälen, dort schnippst Ihr Commodore 64 einmal mit dem kleinen Finger.

Zunächst einmal eine Definition: Unter Zeichensatz verstehen wir die Gesamtheit aller Zeichen (Buchstaben und Graphikzeichen), die Sie im Textmodus durch Drücken verschiedener Tasten und Tastenkombinationen (siehe <shift> und <C=> (Commmodore - Taste)) auf den Bildschirm bringen können.
Die Form und das Aussehen dieser Zeichen muß dem Computer natürlich bekannt, also irgendwo und irgendwie gespeichert sein. Gleichzeitig sollten Sie auch nach irgendwelchen Kriterien geordnet sein, damit Ihr Rechner weiß, daß er beispielsweise ein B auf den Bildschirm bringen soll, wenn Sie die Taste B drücken. Diese Informationen sind natürlich in allen Computern gespeichert.
Beim CBM 64 ist dieser Speicher so angelegt, daß er softwaremäßig, also vom Programmierer erreichbar ist, d.h. sein Inhalt kann ausgelesen und beispielsweise irgendwo in einen anderen Speicherbereich copiert (übertragen) werden. Diese Möglichkeit wurde ausführlich in dem Paragraphen 3.3.1 dargelegt. Weiterhin besitzt Ihr Rechner die Fähigkeit, die Speicheradresse, aus der er die Form der einzelnen Zeichen abliest, zu verändern (s. # 3.3.2). Sie haben also die Wahl, Ihrem CBM 64 zu sagen, daß er sich die Zeichengestalt von nun an z.B. aus dem Speicherbereich ab $2000 (= 8192) also aus dem RAM holen soll. Diesen Speicherbereich können wir natürlich selbst verändern.
Haben wir vorher den Zeichensatz aus dem ursprünglichen ROM in diesen Bereich copiert, so bemerken wir zunächst keine

Änderung, da ja alle Information erhalten geblieben ist. Verändern wir jedoch Teile dieses Speicherbereiches, so ändern wir damit gleichzeitig die Form eines bestimmten Zeichens.
Um nun die einzelnen Zeichen nach unseren Wünschen zu gestalten, müssen wir wissen, wie die Form eines Zeichens gespeichert wird. Dies sei im folgenden erklärt:
Ihr Computer besitzt insgesamt 4 Zeichensätze mit je 128 Zeichen, von denen jeweils nur 2 gleichzeitig auf dem Textbildschirm erscheinen. Wir wollen im Folgenden diese vier Zeichensätze kurz benennen:

Satz I/1 - Normal - Großschrift/Graphikzeichen
Satz I/2 - Invers - Großschrift/Graphikzeichen
Satz II/1 - Invers - Groß-/Kleinschrift
Satz II/2 - Invers - Groß-/Kleinschrift

Bekanntlich können Sie die beiden Zeichensätze I und II durch die gleichzeitige Betätigung der Tasten <C=> und <shift> von Hand aus wechseln. Vom Programm aus dienen hierzu die ASCII-Werte 14 und 142 (Anmerkung: 142 = 128+14), d.h. Sie können mit

```
PRINT CHR$(14);
```

auf Satz II und mit

```
PRINT CHR$(142);
```

auf Satz I umschalten. Mit

```
PRINT CHR$(8);
```

blockieren Sie dabei die Möglichkeit der Umschaltung über die Tastatur, die ja auch während des Programmlaufs möglich ist, und mit

```
PRINT CHR$(9);
```

heben Sie diese Blockade wieder auf (s. hierzu auch das CBM 64 - Benutzerhandbuch auf den Seiten 135-137).

Die Umschaltung zwischen Sätzen 1 und 2 bewerkstelligen Sie durch die Verwendung von <RVS ON> und <RVS OFF>.
In dem Zeichensatzspeicher müssen natürlich alle diese 4 Zeichensätze getrennt aufgelistet sein. Sie haben also die Möglichkeit 4x128 = 512 Zeichen zu verändern (Wie gesagt, können davon jedoch nur jeweils 256 verschiedene Zeichen gleichzeitig angezeigt werden.).
Jedes Zeichen besteht auf dem Bildschirm aus einer Matrix von 8x8 Punkten, wie Sie vielleicht schon wissen. Entsprechend müssen also im Zeichensatzspeicher diese 8x8 = 64 Punkte repräsentiert sein. Das wird erreicht, indem jeder Punkt des Zeichens auf dem Bildschirm -ähnlich wie in HGR- durch ein Bit im Speicher vertreten ist. Somit setzt sich ein Zeichen - Bit - Muster aus 8 Byte zu je 8 Bit zusammen. Jedes Byte repräsentiert eine der 8 Zeilen des Zeichens. Ein gesetztes Bit bedeutet also einen gesetzten Punkt auf dem Bildschirm. Wir können uns die Speicherung eines Zeichens wie folgt vorstellen:

Bit	7	6	5	4	3	2	1	0
Byte 0	.	.	.	.	.	.	.	.
Byte 1	.	.	.	.	.	.	.	.
Byte 2	.	.	.	.	.	.	.	.
Byte 3	.	.	.	.	.	.	.	.
Byte 4	.	.	.	.	.	.	.	.
Byte 5	.	.	.	.	.	.	.	.
Byte 6	.	.	.	.	.	.	.	.
Byte 7	.	.	.	.	.	.	.	.

Der Zeichensatzspeicher ist also aus insgesamt 512 hintereinanderliegender Definitionen dieser Art zu je 8 Bytes zusammengesetzt. Er benötigt also einen Speicherbereich von 4 K (= 4096 Bytes), der normalerweise im ROM von $D000 - $DFFF (dezimal: 53248 - 57344) liegt. Dieser Bereich ist jedoch von Basic aus nicht auszulesen. Zur Demonstration sei an dieser Stelle gezeigt, wie ein normales, großes B im Zeichensatzspeicher definiert wird:

Bit:	7	6	5	4	3	2	1	0
Byte 0:	.	*	*	*	*	.	.	.
Byte 1:	.	*	*	.	.	*	*	.
Byte 2:	.	*	*	.	.	*	*	.
Byte 3:	.	*	*	*	*	.	.	.
Byte 4:	.	*	*	.	.	*	*	.
Byte 5:	.	*	*	.	.	*	*	.
Byte 6:	.	*	*	*	*	.	.	.
Byte 7:	.	.	.	.	.	.	.	.

Wir erhalten also folgende 8 Bytes:

Byte 0: 0111 1000 = $78 = 120
Byte 1: 0110 0110 = $66 = 102
Byte 2: 0110 0110 = $66 = 102
Byte 3: 0111 1000 = $78 = 120
Byte 4: 0110 0110 = $66 = 102
Byte 5: 0110 0110 = $66 = 102
Byte 6: 0111 1000 = $78 = 120
Byte 7: 0000 0000 = $00 = 000

Diese acht Werte stehen nun an der Stelle in dem Zeichensatzspeicher, die für das große, normale B reserviert ist.
Bei Multicolorzeichen stehen jeweils 2 Bit für einen doppelt breiten Punkt des Zeichens, wodurch sich die Auflösung einer Matrix auf 4x8 Punkte pro Zeichen verringert. Da der Vorgang weitestgehendst parallel zur Multicolorgraphik und -sprites funktioniert und da das Wichtigste hierzu bereits unter # 3.2 gesagt wurde, wollen wir lediglich die Punkt-Bit-Beziehung anhand einer kleinen Graphik darstellen:

Bit:	7	6	5	4	3	2	1	0
Byte 0:	<->		<->		<->		<->	
Byte 1:	<->		<->		<->		<->	
Byte 2:	<->		<->		<->		<->	
Byte 3:	<->		<->		<->		<->	
Byte 4:	<->		<->		<->		<->	
Byte 5:	<->		<->		<->		<->	
Byte 6:	<->		<->		<->		<->	
Byte 7:	<->		<->		<->		<->	

Das nächste Problem ist die Ermittlung der Position einer Zeichendefinition. Ausgangspunkt aller Berechnungen sind die Bildschirmcodes der einzelnen Zeichen, der Codes also, die zur Bestimmung eines Zeichens im Videoram stehen (s. Anhang). Der Rest ist relativ einfach: Da jedes Zeichen 8 Byte benötigt, müssen wir nur den Wert des Bildschirmcodes mal 8 nehmen und die Basisadresse des Zeichensatzes, also die Anfangsadresse, bei der unser Zeichensatz beginnt, hinzuaddieren. Bei der Basisadresse ist zu beachten, daß Zeichensatz I und II unterschieden werden müssen. Ein kleines b des Klein- / Großschriftmodus mit dem Bildschimcode 2 liegt 2 K entfernt vom großen B des Großschrift / Graphik-zeichenmodus mit ebenfalls dem Code 2 (nicht zu verwechseln mit dem großen B des Klein- / Großschriftmodus). Im originalen Zeichensatz lauten die Basisadressen:

- Groß/Graphikzeichen: $D000 (53248)
- Klein/Großschrift : $D800 (55296)

Aus soeben Gesagtem ergibt sich die Formel:

adresse = basisadresse + 8 * bildschirmcode

Für das Zeichen B im normalen Zeichenspeicher wäre dieses:

adresse = $D000 + 8 * 2 = $D010 = 53256

Im Kapitel 4 werden Sie ausführlich erfahren, wie Sie diese Änderungen am besten vornehmen (u.a. mit einem sehr komfortablen Zeicheneditor) und wie Sie das Gelernte anwenden können.

3.7. IRQ-Möglichkeiten

Eine der goldenen Eigenschaften Ihres Gerätes ist die mannigfaltig verwendete Interrupt - Einrichtung. Unter Interrupt versteht man die gesteuerte Unterbrechung eines Programms an einer beliebigen Stelle, verursacht durch irgendein Ereignis. Ist eine Unterbrechung ausgelöst worden, so springt der interne Prozessor unterprogrammmäßig (Merken der Rücksprungadressen) an eine bestimmte Stelle im Speicher, die indirekt adressiert wird, und durchläuft dort eine sogenannte Interrupt- oder Unterbrechungsroutine. Nach Beendigung dieser Routine kehrt er genau an dieselbe Stelle des unterbrochenen Programms zurück und fährt ganz normal fort - bis zur nächsten Unterbrechung. Soweit in aller Kürze eine Beschreibung des Interruptvorganges.

Keine Angst, Sie können ruhig weiterlesen, auch wenn Sie von Interrupt noch nie etwas gehört haben und auch der Maschinensprache nicht mächtig sind. Fast alle hier vorgestellten Möglichkeiten, mit Interrupt zu arbeiten, funktionieren gleichfalls ohne diese - nur in Assemblerprogrammen verwendbare - Raffinesse. Falls Sie auf diesem Gebiet also nicht so bewandert sind, sollten Sie sich die Stellen, in denen von Interrupt die Rede ist, trotzdem durchlesen, um einen Überblick zu erhalten. Lassen Sie sich aber nicht abschrecken, da im Anschluß daran stets auch die Möglichkeiten, die von Basic aus existieren, erläutert werden.

Ein Interrupt ist also eine gewollte und programmtechnisch vorgesehene Unterbrechung (kein Abbruch!). Interrupts sind grundsätzlich ebenfalls in Basic möglich (z.B. die Supergraphik 64 bietet dieses Hilfsmittel), die hier besprochenen sind jedoch hardwaremäßig eingerichtete (durch Software gesteuerte) Unterbrechungen der CPU (Central Prozessing Unit - Zentraler Microprozessor). Nun gibt es für den in Ihrem Gerät verwendeten Microprozessor vier verschiedene Interrupts:

- Reset
- NMI (Non Maskable Interrupt)
- BRK (Break)
- IRQ (Interrupt Request)

a) Reset:

Erster, er kann nicht softwaremäßig unterdrückt werden, wird nach dem Einschalten ausgelöst und initialisiert den Computer. Am Ende dieses Vorganges steht die (jedem, der den 64er einmal eingeschaltet hat, bekannte) Kopfschrift auf dem Bildschirm:

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
```

b) NMI:

Dieser nicht maskierbare, d.h. ebenfalls unbedingte Interrupt wird ausgelöst, wenn Sie auf die <restore>-Taste drücken. Er wird gleichfalls benötigt, um die RS 232 - Schnittstelle zu bedienen, falls vorhanden. Sie können die indirekte Sprungadresse des NMI in den Speicherstellen $318/$319 (792/793) erfahren oder ändern.

c) BRK:

Dies ist ein sogenannter Softwareinterrupt, der von einem Assembler - Programm aus betätigt werden kann. Er wird dann ausgelöst, wenn der Prozessor auf den BRK-Code $00 stößt (indirekter Sprung zur der Adresse, die in den Speicherstellen $316/$317 (790/791) steht).

d) IRQ:

Hier ist er endlich! Alle oben genannten Interrupts sollen uns hier nicht weiter interessieren und sind nur der Vollständigkeit halber erwähnt worden. Die eigentlich für uns interessante Unterbrechung ist dieser sogenannte maskierbare Interrupt. Maskierbar heißt, daß per Software gesagt werden kann, ob er ausgelöst werden darf oder nicht. Sie können ihn also beliebig abschalten, wenn es Ihnen gefällt. Hierfür existieren die beiden Assembler-befehle SEI (Set Interruptflag - verhindert Interrupt) und

CLI (Clear Interruptflag - ermöglicht Interrupt). Gleichzeitig können Sie beim Commodore 64 in speziellen Registern auswählen, welche Ursache von einer ganzen Palette an Möglichkeiten einen IRQ auslösen darf und welche nicht. So können beispielsweise die Timer der CIA nach Ihrem Ablauf eine Unterbrechung hervorrufen, was vom Basic - Betriebssystem genutzt wird, indem es alle 1/60 Sekunde den normalen Programmablauf unterbricht und in die ROM-IRQ-Routine springt (indirekte Adresse in $314/$315; dezimal: 788/789). Hier wird der Cursor blinken gelassen, die interne Uhr (TI$) gestellt, und die Tastatur (z.B. STOP-Taste) abgefragt. Doch wir können die Auslösung eines IRQ gleichfalls einigen anderen Ereignissen als nur dem Ablauf eines Timers ermöglichen. Die für uns interessanten sind:

- Rasterzeilendurchlauf
- Lightpen
- Sprite-Sprite-Kollision
- Sprite-Hintergrund-Koll.

Wenn wir nun die indirekte Adresse der IRQ-Routine des Betriebssystems auf eine eigene Interruptroutine richten, können wir diese Möglichkeiten ausnutzen. Wie dies programmtechnisch zu lösen ist, wird Ihnen in den entsprechenden Abschnitten des Kapitels 4 dargelegt. Der Vorteil der IRQ-Verwendung für diese Anwendungen ist, daß das Ereignis sofort gemeldet wird, ohne daß bis zur nächsten Abfrage gewartet werden muß (neben dem IRQ gibt es natürlich bei allem die Möglichkeit, auf das Ereignis im Laufe des Programms durch eine einfache Abfrage zu testen). Dies hat besonders große Bedeutung bei dem Rasterzeilen-IRQ, da hier die Vorgänge sehr schnell ablaufen müssen. Nun aber zu den angekündigten Einzelbesprechungen:

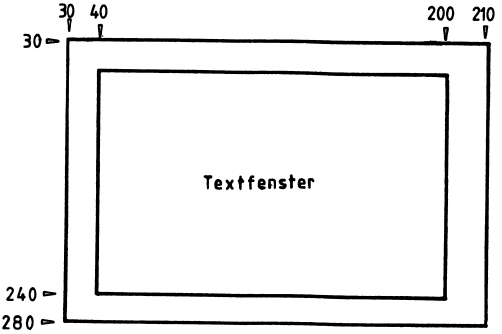
3.7.1. Bildschirmrasterzeilen

Eine der am wenigsten verstandenen, dafür jedoch äußerst ansprechenden Fähigkeiten Ihres Computers ist der Rasterzeileninterrupt. Sie können mit diesen Dingen eine große Anzahl von Effekten erzielen, die Ihr Herz höher schlagen lassen: Mehrere Hintergrundfarben, mehr als 8 Sprites, gemischte Graphik- und Textanzeige usw. usw. Doch bevor wir uns in Kapitel 4 mit der Realisierung dieser Dinge beschäftigen, wollen wir zunächst einmal die Frage klären, was unter sogenannten Rasterzeilen eigentlich zu verstehen ist.

Dazu müssen wir verstehen, wie auf dem Fernseher oder Monitor ein Bild entsteht. Wie Sie vielleicht wissen, besteht der Bildschirm aus einer Lochrasterplatte in Verbindung mit einer phosphoreszierenden Schicht, die durch den Aufprall der Elektronen eines Elektronenstrahles punktuell zu leuchten beginnt. Dieser Elektronenstrahl geht nun Zeile für Zeile jeden einzelnen Punkt des Lochrasters durch und läßt ihn - je nach Bedarf - aufleuchten oder nicht. Dies geschieht in einer ungeheuren Geschwindigkeit, so daß der Strahl pro Sekunde 20 mal ein neues Bild erzeugt, also 20 mal über sämtliche Punkte des Punktrasters hinüberfegt. So entsteht für uns der Eindruck eines bewegten Bildes. Dieser Elektronenstrahl wird durch die komplizierten Apparaturen am hinteren Ende einer Bildröhre gesteuert. Doch die Information, ob ein Punkt des Bildschirms nun aufleuchten oder erblassen soll, muß von einer anderen Quelle stammen. Beim normalen Fernseher sind dies die über den Äther gesandten und von der Antenne eingefangenen Signale der Sendestationen. In unserem Fall muß der Computer dieses Signal erzeugen. Er also muß praktisch Reihe für Reihe und Punkt für Punkt durchgehen, und das "an/aus"-Signal senden. Diese Aufgabe übernimmt im Falle des CBM 64 ebenfalls der Videocontroller (VIC).

Normalerweise wird das alles intern geregelt, ohne daß der Programmierer darauf Einfluß nehmen oder überhaupt daran beteiligt sein könnte. Anders beim 64er: Hier besitzt die Software die Möglichkeit festzustellen, welche Rasterzeile der VIC gerade erstellt. Dies kann u.a. durch das Lesen des VIC-Registers 18 erfolgen. Hierzu muß zunächst aber etwas gesagt werden:

Da bei der Rasterzeilenerstellung selbstverständlich neben dem eigendlichen Text- oder Graphikfenster ebenfalls der Rahmen mit übergeben werden muß und die Strahlablenkung etwas über den Bildschirm hinausgeht, stellt sich die Koordinateneinteilung etwas anders dar, als wir sie von der Graphik oder den Sprites her kennen. Zur Veranschaulichung des im folgenden Gesagten vergleichen Sie bitte das unten gezeigte Schaubild, das eine Skizze des Bildschirms mit dem Fenster darstellt.



Der Strahl startet in der obersten Reihe. Diese besitzt nun aber nicht etwa die Nummer 0 oder 1, wie man vermuten könnte, sondern die obere Kante Ihres Bildschirms beginnt bereits bei Reihe Nr. 30 ($1E) (wobei die angegebenen Randwerte von Fernseher (Monitor) zu Fernseher leicht variieren können). Sie endet (untere Bildschirmkante) ca. bei Nr. 280 ($118). Das eigentliche Bildschirmfenster, das normalerweise verwendet werden kann, hat an der oberen Kante den Rasterzeilenwert von etwa 40, während die untere mit der Rasterzeile Nr. 240 übereinstimmt.
Wie Sie sehen, unterteilt der VIC das Textfenster (genau übereinstimmend mit der Punktauflösung) in 200 Zeilen (Reihen). Sie werden im Kapitel "Joystick" sehen, daß dies im Falle der Spaltenauflösung nicht so einfach ist.

Wie gesagt können wir jetzt selber aktiv in das Geschehen eingreifen, bzw. uns über den jeweiligen Stand unterrichten. Was heißt das nun?
Zum einen können wir (wie oben erwähnt) dem Register 18 des Videocontrollers die Nummer der Rasterzeile, die er gerade

dem Sichtgerät sendet, entnehmen. Da ein Register jedoch Werte von 0 bis maximal 255 annehmen kann, der VIC aber mindestens bis zu 280 Reihen sendet, wird das fehlende oberste (8.) Bit vom VIC-Register 17 geliefert. Wie Sie aus der in # 3.1 gezeigten Tabelle entnehmen, stellt das 7. Bit dieses Registers 17 das gesuchte oberste Bit der Rasterzeilennummer dar. Somit können wir durch einfaches Lesen die genaue Rasterzeile erfahren. Da jedoch ein Bild pro Sekunde 20 mal erneuert wird, und mindestens 280 Reihen pro Bild an den Bildschirm gesendet werden müssen, wird eine Reihe in ca. 1 sek./(20*280) = 0,00018 sek., also in 0,18 milli- oder 180 mikrosekunden (eine Mikrosekunde ist eine Millionstel Sekunde), das sind 5600 Reihen pro Sekunde, aufgebaut. Wenn man bedenkt, daß eine Reihe weiterhin noch aus einer großen Zahl von Punkten (s. # 3.7.2) besteht, kann man sich in etwa vorstellen, wo hier die Zeitverhältnisse pro Punkt liegen (etwa bei 0,89 mikrosekunden).
Damit scheint die softwaremäßige Behandlung auch in Assembler (immerhin dauert ein Befehl in Assembler mindestens eine 500.000stel Sekunde (= 2 mikrosekunden) - bei manchen Befehlen das bis zu 3,5 fache) unmöglich, da praktisch dauernd abgefragt werden müßte, wo sich der Strahl gerade befindet, um eine bestimmte Reihe anzusteuern.
Doch dies ist aufgrund einer äußerst interessanten Einrichtung nicht notwendig. Sie können nämlich den VIC dazu veranlassen, einen IRQ (wie oben beschrieben), also eine Unterbrechung Ihres Programms, auszulösen, wenn er gerade eine bestimmte Reihe des Bildes aufbaut (genau einen bestimmten Punkt anzusprechen, wäre aufgrund der winzigen Punktdurchlaufzeiten (s.o.) sinnlos). Für diesen Zweck schreiben Sie die gewünschte Zeile, bei deren Strahldurchlauf ein Interrupt ausgelöst und damit eine Interruptroutine aufgerufen werden soll, genau in dasselbe Register 18, aus dem wir sonst die aktuelle Position des Strahl ziehen. Auch hier dient Bit 7 des 17. Registers als Highbyte. Wir müssen dem Computer (bzw. dem VIC) nur noch mitteilen, daß er ab sofort diese Unterbrechung ausführen soll, wenn er die bestimmte Reihe erreicht hat. Dies geschieht mit Hilfe der beiden Register 25 und 26 ($19/$1A). Ersteres ist das sogenannte Interrupt Request Register (IRR). Hier wird angegeben, welche Ursache ein durch den VIC ausgelöster IRQ

hat. Dabei gilt:

Bit 0 = 1: Rasterzeileninterrupt
Bit 1 = 1: Interrupt durch Sprite-Hgrund-Koll.
Bit 2 = 1: Interrupt durch Sprite-Sprite-Koll.
Bit 3 = 1: Interrupt durch Lightpen
Bits 4-6 : unbenutzt
Bit 7 = 1: Interrupt hat eine der 4 Ursachen

Sie sehen also, daß hier der Programmierer durch Abfrage des 7. Bits erfährt, ob eins der 4 unteren Bits gleich 1 ist, ein Interrupt also durch eine der 4 Möglichkeiten verursacht wurde (wie gesagt kann er ja auch andere Ursachen haben). Nach jeder Abfrage muß dieses Register wieder zurückgesetzt werden, da ansonsten direkt nach der durchlaufenen Interrupt-routine wieder ein IRQ ausgelöst wird usw. - "Absturz"! Dies geschieht, indem man denselben Wert, den man aus diesem Register ausgelesen hat wieder hierhin zurückschreibt.
Trotzdem wüßte der VIC immer noch nicht, wodurch er einen IRQ auslösen sollte, würden wir nicht Gebrauch von dem Register 26 machen. Hier existiert die gleiche Zuordnung der einzelnen Bits wie im gerade beschriebenen Register 25 (außer 7. Bit). Hier bedeutet allerdings ein gesetztes Bit, daß das betreffende Ereignis von Stund an ein IRQ-Auslöser sein kann. Wollen wir beispielsweise, daß der VIC immer dann unser Programm unterbricht und der Prozessor dann unsere IRQ-Routine anspringt, deren Adresse wir in $314/$315 (= 788/789; s.o.) abgelegt haben, wenn er die Reihe 100 erreicht hat, so schreiben wir zunächst 100 in dieses Register 18 (MSB = 0!), löschen Register 25, indem wir den Inhalt lesen und wieder rückschreiben, und setzen das Bit 0 des Registers 26 gleich 1. Wie das alles programmtechnisch unter einen Hut gebracht wird, sollten Sie nun in dem entsprechenden Paragraphen des Kapitels 4 nachlesen.

3.7.2. Lightpen

Bevor Sie diesen Paragraphen durcharbeiten, sollten Sie wenigstens die Ausführungen in 3.7.1 über die Entstehung des Bildes auf dem Bildschirm gelesen haben, da die folgenden Erklärungen auf diesem Wissen aufbauen.
Ihr Commodore 64 besitzt verschiedene Möglichkeiten, steuernde Geräte als Peripheriebausteine anzuschließen. Zu diesem Zwecke befinden sich an der rechten Seite Ihres Computers, wenn Sie sich das einmal anschauen wollen, zwei sogenannte Controlports. Dies sind Steckbuchsen für den Anschluß von Joysticks, Paddles, Lightpen oder sogar selbstgebaute Steuer- bzw. Meßeinrichtungen (wie z.B. Thermometer, Feuchtigkeitsmesser, Impulsgeber etc.). Die Steckerbelegung wird in Ihrem CBM 64 -Benutzerhandbuch auf der Seite 141 beschrieben. Sie brauchen diese nicht unbedingt zu kennen, um beispielsweise einen Joystick an Ihr Gerät anzuschließen und ihn richtig zu gebrauchen. Wichtig ist nur, daß der Eingang für den unten beschriebenen Lightpen identisch ist mit dem des Feuerknopfes eines in Port 1 gesteckten Joysticks. Um etwas vorzugreifen: Sie können also auch mit dem Feuerknopf von Port 1 einen Interrupt auslösen, was sicher hochinteressant nicht nur für Spiele ist. Sie können damit Geräte anschließen, die im Computer einen IRQ auslösen können, eine Möglichkeit, die wertvolle Konsequenzen hat!
Eine der hochinteressanten Einsatzmöglichkeiten ist der Lightpen:
Unter Lightpen (oder Lichtgriffel) versteht man einen handlichen Stift, der zur Eingabe oder Bestimmung eines Punktes auf dem Bildschirm dient und den direkten Kontakt zwischen Ihnen und dem Fernseher (Monitor) gestattet. Mit dem Lichtgriffel ist es also möglich, durch ein simples Auflegen der Stiftspitze auf den Bildschirm dem Computer eine Bildschirmposition einzugeben.
Wie geht das nun vonstatten? Sie zeigen mit Ihrem in Controlport 1 gesteckten Lichtgriffel auf einen Punkt des Bildschirms. Dabei ist es egal, ob sich dieser Punkt innerhalb oder außerhalb des eigentlichen Textfensters befindet. Der Computer ist alsdann in der Lage, diesen Punkt zu identifizieren, er kennt also die Koordinaten dieses Punktes. Wenn Sie diese in Ihrem Programm abfragen, können

Sie beispielsweise feststellen, ob sich an der Stelle ein bestimmtes Objekt (Buchstabe oder eine Graphik) befindet. Oder Sie zeichnen genau an dieser Stelle einen Punkt in die Graphik, so daß Sie per Hand auf den Bildschirm zeichnen können! Eine andere Idee wäre, den Lightpen als komfortable Cursorsteuerung einzusetzen. Es gibt eine Menge Möglichkeiten der Verwendung.

Doch nun zur technischen Seite des ganzen Geschehens. Wie stellt der Computer fest, wo auf dem Bildschirm nun gerade der Lightpen positioniert ist? Sie wissen, daß ein Bild des Fernsehers (Monitors) aus vielen kleinen Punkten zusammengesetzt ist, die alle 1/20 Sekunden von einem Elektronenstrahl, der auf die erwähnte Lochrasterplatte fällt, zum Aufleuchten gebracht werden. Der Lightpen registriert nun, sofern er auf einen Punkt des Bildschirms gerichtet ist, dieses kurze Aufleuchten und sendet einen Impuls an den Computer. Achten Sie bei der Verwendung des Lightpen deshalb darauf, keine schwarze Hintergrundfarbe zu wählen und den Helligkeitsregler Ihres Bildausgabegerätes nicht zu niedrig einzustellen. Der genannte Impuls erreicht den VIC, der sofort die aktuelle Rasterzeile und die (uns bisher unbekannte) Rasterspalte in zwei Registern als x,y-Punktkoordinaten ablegt. Es sind dies die beiden VIC-Register 19 und 20 (x- und y-Anteil). Hier kann jetzt ein Programm die beiden Werte auslesen und verwerten (z.B. indem es an dieser Stelle einen Punkt zeichnet). Zunächst aber muß noch Einiges zu dem Koordinatensystem gesagt werden. Die y-Einteilung, also die Einteilung nach Rasterzeilen kennen Sie bereits. Sie wurde in # 3.7.1 ausführlich erörtert. Die x-Einteilung ist nun etwas komplizierter. Hier gibt es jeweils halb so viele ansprechbare Rasterpunkte, wie wir von der Graphik her kennen, d.h. ein angegebener Rasterpunkt steht für zwei wahre Graphikpunkte. Es ergeben sich folgende Randwerte: Die linke Kante Ihres Bildschirms besitzt etwa den Randwert 30, die rechte ist Spalte Nr. 210. Das reguläre Bildfenster aber liegt links bei ca. 40 und rechts bei 200, womit wir 160 = 320/2 Rasterpunkte im Textfenster besitzen. Angenommen, wir haben zwei Werte für die Rasterkoordinaten aus den Registern 19/20 entnommen, und wollen genau an dieser Stelle einen Punkt zeichnen. Dann müssen wir die Rasterkoordinaten in solche für die Graphik umrechnen. Dies können wir nach den

obigen Ausführungen durch die folgenden Formeln vornehmen:

```
x = (xp-40)*2
y = yp-40
```

wobei x und y die Graphik- und xr, yr die Rasterkoordinaten darstellen. Jetzt können wir an der errechneten Stelle einen Punkt setzen.
Das ist die einfache und auch von Basic aus programmierbare Möglichkeit. Doch auch hier bietet Ihnen Ihr Computer eine Möglichkeit, mit der Interrupttechnik zu arbeiten:

Sendet der Lightpen nämlich einen Impuls, so wird das 3. Bit des VIC-Registers 25 gesetzt. Haben wir vorher in Register 26 ebenfalls das 3. Bit gesetzt, kann ein Interrupt ausgelöst werden. Ihr Programm wird also unterbrochen und Ihre Interruptroutine aufgerufen, die das Ereignis bearbeitet. Auch hier werden Programmierbeispiele etc. in Kapitel 4 gegeben.

3.7.3. Sprite-Kollisionen

Wenn Sie sich den Abschnitt 3.5.4.4 durchgelesen haben, wissen Sie bereits, daß Sie eventuelle Berührungen von Sprites untereinander oder mit Hintergrundzeichen feststellen können, indem Sie den Inhalt der VIC-Register 30 und 31 lesen und analysieren. Wie Sie wissen, ist hier jedem Sprite ein Bit zugeordnet und diejenigen Bits gesetzt, deren Sprite kollidiert ist. Doch es gibt eine weitere Möglichkeit aus Assembler heraus Kollisionen zu registrieren. Wie Sie sich schon denken können, beruht diese Möglichkeit wieder auf der Interrupttechnik. Sie können den VIC (wieder durch den Gebrauch der Register 25/26) veranlassen, bei irgendeiner Kollision einen Interrupt auszulösen. Auch hier werden wieder Berührungen zwischen Sprites und dem Hintergrund und Sprite - Sprite - Kollisionen unterschieden. In Register 25 (IRR) ist dafür jeweils ein Bit reserviert. Bit 3 wird gesetzt, wenn eine im Register 30 näher angegebene Berührung zwischen zwei Sprites stattgefunden hat. Entsprechendes passiert mit Bit 2, falls die Berührung zwischen einem Sprite und einem

Hintergrundzeichen stattfand. Ein Interrupt wird, wie Sie sich sicher denken können, aber erst ausgelöst, wenn Sie vorher das korrespondierende Bit in Register 26 gesetzt haben. Vergessen Sie nicht, die IRQ-Adresse umzulegen und nach jedem IRQ das IRR durch Zurückschreiben der Lesedaten zu löschen.

4. Kapitel
Grundsätzliche Graphikprogrammierung

Nachdem wir jetzt genügend graue Theorie über die Graphikfähigkeiten Ihres Rechners gehört haben, wollen wir uns in diesem Kapitel mit der Realisierung dieser Dinge beschäftigen. Denn was nützt uns das alles, wenn wir nicht wissen, wie wir Sprites oder hochauflösende Graphik einsetzen. Es ist das "Know how", das uns fehlt. Aus diesem Grunde gibt es zu jedem der obigen Abschnitte des 3. einen dazugehörigen aus diesem Kapitel, der die Programmierung bzw. die Anwendung der einzelnen Möglichkeiten behandelt.
Programme werden möglichst in Basic, bei den vielen Maschinenspracheroutinen zusätzlich ein Basiclader angegeben. Alle Programme sind in REM-Zeilen bzw. mit Kommentaren dokumentiert. Die wichtigen und neuen Programmteile werden auch im Text beschrieben. Bei den Basicroutinen für die einzelnen Graphikfiguren und -anwendungen muß in aller Deutlichkeit gesagt werden, daß dies selbstverständlich nur Hilfen sind; die entsprechenden Assemblerroutinen führen die gewünschte Funktion sehr viel schneller aus. Deswegen lohnt sich -wie auch oben schon einmal erwähnt- der Erwerb von Maschinensprachekenntnissen ungeheuer. Wenn Sie Basic einigermaßen gut beherrschen, ist der Schritt dorthin nicht mehr weit. Versuchen Sie es einmal!

4.1 Text und Graphik auf dem Low-Res-Bildschirm

Die erste und einfachste Möglichkeit, Graphiken auf Ihrem Bildschirm darzustellen, ist das Arbeiten mit den originalen Graphikzeichen, die Sie an der vorderen Seite der einzelnen Tasten Ihres Rechners finden. Schon hiermit lassen sich mit etwas Phantasie (sie wird bei der Graphikerstellung stets benötigt) wunderbare und vielfarbige Bilder erzeugen. Die Graphikzeichen lassen sich auf verschiedene Art und Weise erreichen:

a) Ansteuerung durch die Tastatur:

Die Zeichen, die auf der linken Seite der Tasten stehen, lassen sich durch gleichzeitiges Drücken dieser jeweiligen normalen- mit der <shift>-Taste auf dem Bildschirm darstellen. Diejenigen, die rechts stehen, werden mit der gedrückten <C=> (<commodore>) -Taste angewählt. Zusätzlich zu diesen normalen können Sie noch sogenannte inverse Zeichen erzeugen, indem Sie vorher gleichzeitig auf die Tasten <ctrl> und <rvs on> drücken (oder Sie geben ein: PRINT CHR$(18)). Wollen Sie die ausgesuchten Zeichen nicht mehr invers darstellen, so genügt ein <ctrl> mit <rvs off> (PRINT CHR$(146)).

Farben:

Die 16 verschiedenen Zeichen-Farben, die auf dem Bildschirm dargestellt werden können, sind ebenfalls durch die Tastatur anwählbar (eine umfassende Tabelle der Farben und ihrer verschiedenen Zuordnungen finden Sie im Anhang): Dabei drücken Sie für die ersten 8 Farben (numeriert nach den Farbcodes) gleichzeitig mit der <ctrl>-Taste eine der Tasten <1-8> (die dabei entstehende Farbe steht ebenfalls auf der Frontseite der 8 Tasten). Wollen Sie dagegen den folgenden Zeichen eine der 8 letzten Farben (Farben Nr. 8-15) geben, so drücken Sie einfach die <C=>- mit den erwähnten 8 Farbtasten.

b) Ansteuerung durch das PRINT-Statement:

Jedes Zeichen besitzt einen bestimmten, sogenannten

ASCII-Code. Durch Angabe dieses Codes kann jedes Zeichen eindeutig bestimmt werden (dabei besteht zwischen inversen und normalen Zeichen allerdings kein Unterschied). Den ASCII-Code eines Zeichens können Sie durch den Basic-Befehl ASC in der folgenden Weise ermitteln (In diesem Falle für das Zeichen "A"):

```
PRINT ASC("A")                oder
Z$ = "A" : PRINT ASC(Z$)
```

Der Rechner schreibt: 65. Kennen Sie umgekehrt den ASCII-Code eines Zeichens, so finden Sie dieses wie folgt durch den Befehl CHR$ (ebenfalls für das Zeichen "A"):

```
PRINT CHR$(65)                oder
C = 65 : PRINT CHR$(C)
```

Und schon steht ein A auf dem Bildschirm. Der Vorteil dieser Codierung ist eine Berechenbarkeit von Zeichen, d.h. Sie können ein Zeichen durch irgendeinen Algorithmus (Rechenvorschrift) bestimmen. Gleichzeitig werden Vergleiche o.ä. sehr vereinfacht.

Farben:
Auch die verschiedenen Farben (neben vielen anderen Control - Funktionen wie <clr/home> etc.) besitzen ASCII-Codes. Leider werden im CBM 64-Handbuch nur die der ersten 8 Farben angegeben; Hier daher eine vollständige Auflistung:

ASCII	Farbe	ASCII	Farbe
144	schwarz	129	orange
5	weiß	149	braun
28	rot	150	hellrot
159	cyan	151	grau 1
156	violett	152	grau 2
30	grün	153	hellgrün
31	blau	154	hellblau
158	gelb	155	grau 3

c) Ansteuerung durch POKE:

Alle Zeichen, die sich auf dem Bildschirm befinden, werden in einem besonderen Speicher abgelegt: dem Videoram (s. # 3.6). Hier müssen natürlich normale und inverse Zeichen unterschieden werden, während Controlzeichen, die nicht auf dem Bildschirm erscheinen, nicht vermerkt werden brauchen. Somit wird bei der Abspeicherung ein anderer Code, der sogenannte Bildschirmcode verwendet (s. Kapitel 4). Wollen Sie also direkt in diesen Speicher POKEn, so müssen Sie sich jenes Codes bedienen. Mit

POKE 1024, 1

beispielsweise bringen Sie ein A in die linke obere Ecke des Bildschirms. Dieses A ist jedoch noch nicht zu sehen (falls dort nicht zufällig vorher schon ein Zeichen stand). Es fehlt die Farbe, die im sogenannten Farbram abgespeichert wird und z.B. mit

POKE 55296,5

angewählt wird (die 5 stellt den Farbcode dar, der im Farbram abgespeichert wird). Das Zeichen wird grün (Farbe 5). Diese Zusammenhänge sind ausgiebig in Kapitel 3.6.1 dargelegt.

Auf den folgenden Seiten werden ihnen drei Programme vorgestellt, die alle das gleiche Ergebnis (s. Bild) erbringen, welches jedoch auf drei unterschiedliche Arten entsteht, ohne Rücksicht darauf, daß sich die eine oder andere Methode in diesem Falle so gut wie gar nicht für den demonstrierten Zweck eignet. Sie sollen Ihnen lediglich zeigen, wie die einzelnen Möglichkeiten der Zeichendarstellung in vivo, also direkt im Programm realsiert werden können.

AN DIESER STELLE SOLLTE EIGENTLICH
EIN KLEINES PROGRAMM STEHEN
AUS DRUCKTECHNISCHEN GRUENDEN JEDOCH
MUSSTEN WIR ES IN DEN
ANHANG (6.10) VERLEGEN

```
100 REM ************************
110 REM **                    **
120 REM **  LOW-GRAPHIK/CHR$  **
130 REM **                    **
140 REM ************************
150 REM
160 PRINT CHR$(147) : PRINT : PRINT : REM BILDSCHIRM
LOESCHEN/LEERZEILEN
170 FOR X=1 TO 290 : REM 290 ASCII-CODES EINLADEN
180 READ CH : REM LESE DATA
190 PRINT CHR$(CH); : REM SCHREIBE ZEICHEN
200 NEXT X
210 REM
220 REM ******************
230 REM **  DATAZEILEN  **
240 REM ******************
250 REM
260 DATA  32,  32, 205,  32, 206,  13
270 REM
280 DATA  32, 205, 213, 201,  32,  32
290 DATA  32,  32,  32,  32,  32,  32
300 DATA  32,  32,  32, 213, 203,  13
310 REM
320 DATA  32,  32, 202, 203, 205,  13
330 REM
340 DATA  32, 206,  32, 205,  32,  32
350 DATA  32,  32,  32,  32,  32,  32
360 DATA  32, 213, 203,  13
370 REM
380 DATA  32,  32,  32,  32,  32,  32
390 DATA  32,  32,  32,  32,  32,  32
400 DATA  32, 175,  13
410 REM
420 DATA  32,  32,  32,  32,  32,  32
430 DATA  32,  32,  32,  32, 175, 175
440 DATA 175, 204, 204, 175, 175,  13
450 REM
460 DATA  32,  32,  32,  32,  32,  32
470 DATA  32,  32,  32, 206, 205,  32
480 DATA  32,  32,  32,  32,  32, 205
490 DATA  13
```

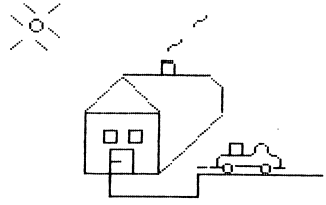
```
500 REM
510 DATA  32,  32,  32,  32,  32,  32
520 DATA  32,  32, 206,  32,  32, 205
530 DATA  32,  32,  32,  32,  32, 182
540 DATA  13
550 REM
560 DATA  32,  32,  32,  32,  32,  32
570 DATA  32, 206,  32,  32,  32,  32
580 DATA 205,  32,  32,  32,  32, 182
590 DATA  13
600 REM
610 DATA  32,  32,  32,  32,  32,  32
620 DATA  32, 207, 183, 183, 183, 183
630 DATA 208,  32,  32,  32,  32, 206
640 DATA  13
650 REM
660 DATA  32,  32,  32,  32,  32,  32
670 DATA  32, 165, 176, 174, 176, 174
680 DATA 170,  32,  32,  32, 206,  13
690 REM
700 DATA  32,  32,  32,  32,  32,  32
710 DATA  32, 165, 173, 189, 173, 189
720 DATA 170,  32,  32, 206,  32,  32
730 DATA 176, 174, 213, 201,  13
740 REM
750 DATA  32,  32,  32,  32,  32,  32
760 DATA  32, 165,  32, 207, 208,  32
770 DATA 170,  32, 206,  32,  32, 213
780 DATA 177, 177, 203, 202, 201,  13
790 REM
800 DATA  32,  32,  32,  32,  32,  32
810 DATA  32, 165,  32, 207, 170,  32
820 DATA 170, 206,  32,  32,  45, 177
830 DATA 215, 195, 195, 215, 177,  13
840 REM
850 DATA  32,  32,  32,  32,  32,  32
860 DATA  32, 183, 183, 207, 183, 183
870 DATA 183,  32,  32,  32, 207, 183
880 DATA 183, 183, 183, 183, 183, 183
890 DATA 183, 183,  13
900 REM
```

```
910 DATA  32,  32,  32,  32,  32,  32
920 DATA  32,  32,  32, 204, 175, 175
930 DATA 175, 175, 175, 175, 165,  13

100 REM ************************
110 REM **                    **
120 REM **  LOW-GRAPHIK/POKE  **
130 REM **                    **
140 REM ************************
150 REM
160 FA = 5 : REM FARBE = GRUEN
170 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
180 VR = 1024 + 2*40 : REM POKE-STARTADRESSE (VIDEORAM)
190 FR = 55296 + 2*40 : REM POKE-STARTADRESSE (FARBRAM)
200 FOR Y=0 TO 15 : REM 16 ZEILEN
210 READ ZA : REM ANZAHL DER ZEICHEN IN DER ZEILE HOLEN
220 VR = VR+40  :  REM  NAECHSTE  ZEILE  (40  SPEICHERSTELLEN
WEITER)
230 FR = FR+40  :  REM  NAECHSTE  ZEILE  (40  SPEICHERSTELLEN
WEITER)
240 FOR X=0 TO ZA-1 : REM ZA ZEICHEN POKEN
250 READ BC : REM BILDSCHIRMCODE LESEN
260 POKE VR+X, BC : REM UND IN VIDEORAM SCHREIBEN
270 POKE FR+X, FA : REM FARBE EINPOKEN
280 NEXT X
290 NEXT Y
300 REM
310 REM ************************
320 REM **  BILDSCHIRMCODES  **
330 REM ************************
340 REM
350 DATA   5
360 DATA  32,  32,  77,  32,  78
370 REM
380 DATA  17
390 DATA  32,  77,  85,  73,  32,  32
400 DATA  32,  32,  32,  32,  32,  32
410 DATA  32,  32,  32,  85,  75
420 REM
430 DATA   5
440 DATA  32,  32,  74,  75,  77
```

```
450 REM
460 DATA  15
470 DATA  32,  78,  32,  77,  32,  32
480 DATA  32,  32,  32,  32,  32,  32
490 DATA  32,  85,  75
500 REM
510 DATA  14
520 DATA  32,  32,  32,  32,  32,  32
530 DATA  32,  32,  32,  32,  32,  32
540 DATA  32, 111
550 REM
560 DATA  17
570 DATA  32,  32,  32,  32,  32,  32
580 DATA  32,  32,  32,  32, 111, 111
590 DATA 111,  76,  76, 111, 111
600 REM
610 DATA  18
620 DATA  32,  32,  32,  32,  32,  32
630 DATA  32,  32,  32,  78,  77,  32
640 DATA  32,  32,  32,  32,  32,  77
650 REM
660 DATA  18
670 DATA  32,  32,  32,  32,  32,  32
680 DATA  32,  32,  78,  32,  32,  77
690 DATA  32,  32,  32,  32,  32, 118
700 REM
710 DATA  18
720 DATA  32,  32,  32,  32,  32,  32
730 DATA  32,  78,  32,  32,  32,  32
740 DATA  77,  32,  32,  32,  32, 118
750 REM
760 DATA  18
770 DATA  32,  32,  32,  32,  32,  32
780 DATA  32,  79, 119, 119, 119, 119
790 DATA  80,  32,  32,  32,  32,  78
800 REM
810 DATA  17
820 DATA  32,  32,  32,  32,  32,  32
830 DATA  32, 101, 112, 110, 112, 110
840 DATA 106,  32,  32,  32,  78
850 REM
```

```
860 DATA  22
870 DATA  32,  32,  32,  32,  32,  32
880 DATA  32, 101, 109, 125, 109, 125
890 DATA 106,  32,  32,  78,  32,  32
900 DATA 112, 110,  85,  73
910 REM
920 DATA  23
930 DATA  32,  32,  32,  32,  32,  32
940 DATA  32, 101,  32,  79,  80,  32
950 DATA 106,  32,  78,  32,  32,  85
960 DATA 113, 113,  75,  74,  73
970 REM
980 DATA  23
990 DATA  32,  32,  32,  32,  32,  32
1000 DATA  32, 101,  32,  79, 106,  32
1010 DATA 106,  78,  32,  32,  45, 113
1020 DATA  87,  67,  67,  87, 113
1030 REM
1040 DATA  26
1050 DATA  32,  32,  32,  32,  32,  32
1060 DATA  32, 119, 119,  79, 119, 119
1070 DATA 119,  32,  32,  32,  79, 119
1080 DATA 119, 119, 119, 119, 119, 119
1090 DATA 119, 119
1100 REM
1110 DATA  17
1120 DATA  32,  32,  32,  32,  32,  32
1130 DATA  32,  32,  32,  76, 111, 111
1140 DATA 111, 111, 111, 111, 101
```



Das erste der drei Programme zeigt die Erstellung eines Bildes mittelst PRINT-Statement, die hier schnellste und

kürzeste und damit in diesem Fall wohl günstigste Möglichkeit. Hier wird das Bild nach dem Löschen des Bildschirmes, was in Zeile 160 durch ein

PRINT CHR$(147)

geschieht, durch je ein PRINT-Statement pro Zeile zusammengesetzt. Dabei wird ausgiebig von den erwähnten Graphikzeichen Gebrauch gemacht. In den REM-Zeilen dahinter erfahren Sie, auf welchen Tasten Sie die einzelnen Gebilde finden (die Leerzeichen sind dabei natürlich ausgelassen). Sie müssen diese Tasten dann nur noch gemeinsam mit <shift> oder <C=> drücken (wie oben beschrieben), und schon erscheint das gewünschte Zeichen auf Ihrem Bildschirm.

Im zweiten Programm wird das gespeicherte Bild ebenfalls durch PRINT-Statements erzeugt. Sie enthalten jedoch nicht direkt die einzelnen Zeichen, sondern diese werden über den Umweg der ASCII-Codes erzeugt. Die verschiedenen ASCII-Codes sind dabei in DATA-Zeilen untergebracht. Wie Sie vielleicht wissen, werden in DATA-Zeilen verschiedene durch Komma abgetrennte Elemente gespeichert, die dann durch den Befehl READ nacheinander(!) in einen beliebigen Speicher (hier CH) eingelesen werden können (s. CBM 64 - Handbuch Kapitel 8). Dieses Einlesen geschieht in unserem Programm in Zeile 180. Die Variable CH enthält nun den ASCII-Wert des als nächstes auszugebenden Wertes. In Zeile 190 wird das zugeordnete Zeichen dann gePRINTet. Das Ganze spielt sich in einer FOR...NEXT - Schleife ab, die insgesamt 290 mal durchläuft, um alle 290 Daten einzulesen. Am Ende jeder Bildzeile (die in den DATA-Zeilen durch ein REM getrennt sind) steht, wie Sie sehen, die Zahl 13. Dies ist der ASCII-Code für <return> und veranlaßt den Computer das nächste Zeichen an den Anfang der nächsten Zeile zu setzen. Wie Sie weiterhin sehen, werden hier eine ganze Menge DATAs benötigt, was diese Methode in unserem Falle recht uneffektiv gestaltet. Ein wenig geändert wäre die Lage, wenn man statt der vielen Codes für die Leerzeichen (ASCII = 32) am Anfang einer Zeile einen Merker angibt, der über die Anzahl der Leerzeichen Auskunft gibt, die ausgegeben werden müssen, bevor die richtigen Graphikzeichen erscheinen. Lassen Sie sich einmal etwas einfallen.

Beispiel Nr. 3 demonstriert uns die Anwendung des POKE-Befehls, um Zeichen direkt in den Videoram einzuschreiben. Auch hier werden diesmal die Bildschirmcodes in DATA-Zeilen untergebracht. Doch neben dem einfachen Einschreiben eines Zeichens in den Speicher müssen Sie weiterhin noch die Farbe jedes einzelnen Buchstaben etc. in den Farbram eintragen, wie oben dargelegt. Kern des Programms sind zwei ineinander verschachtelte FOR...NEXT - Schleifen. Die äußere (Z. 200-290) erhöht (nach dem Durchlauf der inneren) die Nummer der Zeile, die mit Graphikzeichen gefüllt werden soll. Vor dem Start der inneren Schleife wird zunächst ein Wert aus den DATA-Zeilen in die Variable ZA gelesen (Z. 210), der die Anzahl der Zeichen in der jeweiligen Bildschirm-Zeile angibt. Dieser dient dazu die Anzahl der inneren Schleifendurchläufe zu bestimmen. In dem Inneren dieser Schleife werden nun nacheinander die Bildschirmcodes der verschiedenen Zeichen durch READ eingelesen (Z. 250) und an die laufende Adresse im Videoram gePOKEt (Z. 260). Alsdann schreiben wir in die korrespondierende Stelle des Farbrams den Wert 5 für die Farbe grün (Z. 270), der in Zeile 160 festgelegt wurde. Auch hier bietet sich natürlich die gleiche Verkürzung der DATA-Zeilen wie im zweiten Beispiel beschrieben an.

Wie stellt man aber nun am günstigsten ein eigenes Graphikbild zusammen? Hier gibt es die unterschiedlichsten Möglichkeiten, und jeder wird selbst entscheiden, welche von ihnen ihm am einfachsten erscheinen. Allgemein kann aber gesagt werden, daß ein richtig schönes Bild mit vielen Details eine recht zeitaufwendige Sache ist, besonders, wenn man dazu noch selbstdefinierte Zeichen mit ins Spiel bringt, wie dies in Paragraph 4.4 dargelegt ist. Doch können auch einfache errechnete (also durch eine bestimmte Rechenvorschrift erzeugte) Bilder oft schöne Effekte erzeugen. Dies demonstriert z.B. das folgende Beispiel:

```
100 REM ********************
110 REM **                  **
120 REM **  ZUFALLSBILD     **
130 REM **                  **
140 REM ********************
150 REM
```

```
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 PRINT CHR$(RND(1)*3 + 177); : REM EINES DER DREI  ZEICHEN
ASCII=177/178/179
180 GOTO 170
```

Vielleicht versuchen Sie dieses Programm einmal zu verstehen. Ein Tip: RND(1) ergibt Zufallszahlen von 0-1.
Eine weitere Möglichkeit zur Erstellung von Bildern ist mit der Verwendung einer besonderen Routine verbunden. Diese Routine positioniert den aktuellen Cursor an eine beliebige Stelle des Bildschirms. Wie Sie wissen ist dies mit dem originalen Basic nur innerhalb einer Zeile möglich. Die kleine Unterroutine ab Zeile 1000 des folgenden Programms aber läßt einen beliebigen Zugriff zu:

```
100 REM ***************************
110 REM **                       **
120 REM **  SINUSKURVE IM TEXT   **
130 REM **                       **
140 REM ***************************
150 REM
160 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
170 FOR X=0 TO 39
180 Y=13*SIN(X/3)+12 : REM FUNKTION
190 GOSUB 1000 : PRINT"*"; : REM POSITION BERECHNEN
200 NEXT X : END
210 REM
220 REM POSITIONSBERECHNUNG:
230 REM ********************
1000 PRINT CHR$(19);:IF Y>0 THEN FOR Z=1 TO Y:PRINT:NEXT T
1010 PRINT TAB(X);: RETURN
```

Gezeichnet wird eine Sinuskurve (s. # 5.1) mithilfe des Sternzeichnes (*). Dabei werden dem Unterprogramm in Zeile 1000 in den Speichern X und Y die Parameter für die Spalte (X) und die Reihe (Y) der gewünschten Cursorposition übergeben. Nach einem <home> (PRINT CHR$(19);) werden solange carrige returns gesendet, bis die gewünschte Zeile ereicht ist. Dann muß nur noch durch einen TAB-Befehl die richtige Spalte ausgewählt werden. Dieses kleine, aber äußerst effektive Unterprogramm erlaubt Ihnen wunderschöne Graphiken schon im Textmodus.

Ein weiteres recht schönes und übersichtliches Verfahren der Erstellung von Bildern besteht darin, das gewünschte Bild auf dem Fernseher direkt mit dem Cursor und der Tastatur zu erstellen. Dabei lassen Sie die Farbe am besten zunächst einmal außer betracht. Alsdann schreiben Sie vor jede einzelne Zeile (entweder mit Insert, wobei Sie beachten müssen, daß dadurch eventuell Zeilen unten fortgeschoben werden, oder durch einfaches Überschreiben) zunächst die Zeilennummer, ein PRINT (abkürzbar mit einem Fragezeichen (?)) und ein Anführungszeichen (") und gefolgt schließlich von <return> (Sie brauchen also nicht unbedingt ein zweites Anführungszeichen, wenn die Zeile mit diesem PRINT-Ausdruck endet). Damit haben Sie schon einmal die wesentlichen Bildinhalte gespeichert. Doch die Sache hat einige Haken:
Erstens muß in einer Zeile noch Platz für die Zeilennummer, Frage- und Anführungszeichen sein. Soll dieser Platz ebenfalls genutzt werden, so muß dies nachträglich im Programm geschehen. Vorsicht ist in der letzten Bildschirmzeile geboten. Sollten Sie zufällig mit dem Cursor nach unten über diese hinwegrollen, so wird der gesamte Bildschirm nach oben geschoben und die oberste Zeile verschwindet. Ein weiteres Problem: Sie dürfen in keiner Zeile die letzte Spalte verwenden. In diesem Fall würde eine neue Bildschirmzeile eingeschoben, was zu diversen Schwierigkeiten führt.
Weiterhin können keine inversen Zeichen direkt in ein PRINT-Statement aufgenommen werden. Sollte Ihr Bild solche enthalten, so müssen Sie sie durch folgende Sequenz ersetzen:

<rvs on><....><rvs off>

Mit <....> sind alle hintereinander folgenden (im Programm normalen) Zeichen gemeint, die auf dem Bildschirm invers dargestellt werden sollen.
Drittens können Sie ebenfalls keine Farben direkt in ein PRINT-Statement mit aufnehmen. Dies muß durch den Einbau der entsprechenden Farb - Controlcodes in ein PRINT-Statement nach dem Editieren geschehen. Eine Bemerkung zum Schluß: Die Control-Zeichen werden bekanntlich nur dann richtig in einen

PRINT - Ausdruck eingebaut (und nicht sofort ausgeführt), wenn Sie vorher ein Anführungszeichen (") gegeben haben. Dies ist aber bei nachträglichen Einfügungen sehr störend und auch nicht unbedingt notwendig. Sie können nämlich durch Einfügen einer Leerstelle mit <inst> ebenfalls an diese Stelle ein Controlzeichen setzen. Wollen Sie also in dem String "AB" zwischen A und B ein Controlzeichen einfügen, so kann dies einfach durch <inst> <controlzeichen> geschehen.
Wie Sie sehen, ist diese Methode der Bilderzeugung nicht gerade sehr komfortabel, doch für den Anfang oder den Gelegenheitsdesigner akzeptabel. Wollen Sie aber professionell Bilder in größerer Stückzahl und Qualität erzeugen, so sollten Sie sich einen kleinen Bildeditor schreiben, also ein Programm, mit dem Sie einfach durch Bewegen eines (selbst erzeugten) Cursors ein Bild mit allen Farben und Möglichkeiten erstellen können. Dies ist zugegebenermaßen nicht ganz einfach, aber ein sicher lohnendes Projekt. Diejenigen, die sich für dieses Thema besonders interessieren, sollten hierzu unbedingt noch den Paragraphen 4.4 gelesen haben.

4.2 Programmierung der Punktgraphik

Besonders die komplizierte Organisation der hochauflösenden oder gar der Multicolor - Graphik macht einem zu schaffen, wenn man versucht, diese zu bedienen. Allein die Ansteuerung eines Punktes in einem (gedachten) Koordinatensystem verursacht schon recht großes Kopfzerbrechen und schließlich einen enormen Rechenaufwand. Die bloße Berücksichtigung aller notwendigen Faktoren zum Einschalten der Graphik bedarf eines guten Überblicks, der sich erst nach einiger Beschäftigung mit dem Thema Graphik einstellt. Die Erstellung von einfachen Linien oder sogar Kreisen ist dabei so schwer und bedarf vieler mathematischer Kenntnisse, daß sie schon nur noch von recht firmen Programmierern gelöst werden können. Aus diesem Grunde werden hier die verschiedenen Routinen (also Programmteile) vorgestellt, die zur Realisierung der in Kapitel 3 dargelegten Möglichkeiten Ihres Rechners notwendig sind. Es ist dabei nicht unbedingt erforderlich, daß jeder einzelne Schritt eines Programms verstanden ist, da letztendlich die Anwendung dieser Dinge ausschlaggebend ist. Wer also nicht weiß, was beispielsweise sin oder cos bedeuten, der sollte über die einzelnen Stellen (hier der Kreiserzeugung) hinweglesen. Trotzdem sollte er den entsprechenden Abschnitt in Kapitel 3 (Abschnitt 3.4) über die Grundlagen der Graphik gelesen haben. Für die Interessierten jedoch können solche Informationen wertvoll sein, um die einzelnen Routinen für eigene Zwecke abzuwandeln oder Teile daraus für ähnliche Aufgaben zu verwenden.
Sie sollten sich jedoch darüber im Klaren sein, daß eine Basicroutine, so übersichtlich sie sein mag, bei weitem nicht die Geschwindigkeit besitzt, wie ein entsprechendes Maschinenspracheprogramm. Damit sind viele Effekte allein in Basic nur sehr träge zu verwirklichen. Wenn Sie sich einmal anschauen, wie lange es in Basic dauert, einen Kreis zu zeichnen, so werden Sie mir da wohl in aller Entschiedenheit und ohne zu zögern zustimmen. Um dieses Manko zu eliminieren, werden wir Ihnen am Ende dieses 4. Kapitels ein kleines Assembler - Graphik-Aid (samt Basiclader) zusammenstellen,

das Ihnen komprimiert die Möglichkeiten verschafft, die die Kernpunkte jeder Graphik darstellen. Die einzelnen Funktionen des Graphik-Packetes können Sie -quasi als kleine Basic-erweiterung- von Basic aus ansteuern. Wollen Sie nur in Basic programmieren, so beherzigen Sie die Tips, die Ihnen im Anhang zur Optimierung Ihrer Programme gegeben wurden.
Bei allen Programmen wird davon ausgegangen, daß der Graphik-speicher bei $2000-$3FFF (8192-16383) und der Videoram weiterhin bei $0400-$07FF (1024-2047) liegen. Dies macht zwar ein Arbeiten mit Text und Graphik zugleich unmöglich, ist jedoch programmtechnisch besser zu bewältigen. Achten Sie aber bei langen Programmen und/oder vielen Speichern darauf, daß diese nicht mit der Graphikseite kollidieren. Sollte dies einmal geschehen, so setzen Sie in der ersten Zeile Ihres Programms einfach durch

```
POKE 45,0 : POKE 46,64
```

den Start der Variablen hoch auf $4000 (16384). Dabei ist jedoch zu beachten, daß bei jeder Programmveränderung die Graphikseite zerstört wird und bei einem Abspeichern auf Diskette oder Kassette die Graphik mit übertragen wird. Wollen Sie also Veränderungen an Ihrem Programm vornehmen nachdem es einmal gestartet worden ist, so müssen Sie es erst einmal wieder einladen und direkt nach der Veränderung abspeichern, bevor Sie es wieder starten!
Wichtig bei allen Angaben ist, daß Sie diese direkt am Computer ausprobieren. Nur so werden Sie Herr über die Unmasse an Fakten und Zusammenhängen und nur so lernen Sie damit umzugehen. Der Computer ist Praxis!
Doch jetzt wollen wir endlich anfangen. Krempeln wir uns also die Ärmel hoch, spucken dreimal in die Hände und los geht's!

4.2.1. Initialisierung der Graphik

Bevor wir unsere Figuren auf den Bildschirm zaubern, müssen wir natürlich erst einmal dafür sorgen, daß überhaupt Graphik zu sehen ist. Dazu gehört, daß der Bildschirm zunächst gelöscht wird, da im Anfang stets Einiges an "Müll" erscheinen wird. Der Graphikspeicher wurde schließlich vorher anderweitig genutzt. Zu dem Löschen des Graphikbildschirms gehört natürlich auch ein Löschen der Farbe bzw. das Herstellen eines einfarbigen Bildschirms. Anschließend wollen wir sicher wieder zurück, um Text anzuzeigen. Wir benötigen also insgesamt vier getrennte Programmteile, sogenannte Routinen, allein um in die Graphik einzusteigen:

- Graphik einschalten
- Graphik löschen
- Farbe löschen
- Graphik ausschalten

Diese vier Rechenvorschriften (Algorithmen) werden im folgenden einzeln vorgestellt und besprochen. Sie werden sehen, daß sie in jedem späteren Programm, das sich mit der Graphik beschäftigt, wieder in Form von Unterprogrammen auftauchen werden. Sie sollten also zu Ihrem ständigen Repertoire gehören.

4.2.1.1. Einschalten der Graphik

Nun ist es also soweit, wir können beginnen. Stellen wir zunächst einmal die Dinge zusammen, die zum Einschalten benötigt werden:

a) Speicherlage:
Zunächst einmal müssen wir uns einigen, wo im gesamten Speicherbereich des 64ers die einzelnen Funktionen wie Videoram und Graphikspeicher liegen sollen. Hierfür sind Register 24 (Bits 3 und 4-7) des Videocontrollers und das Register 0 (Bits 0 und 1) der CIA 2 zuständig, deren Funktionen ausgiebig in dem Abschnitt 3.3 erläutert werden. In allen unseren Anwendungen werden wir uns -wie

oben schon erwähnt- mit dem Graphikspeicher nur in dem Bereich von $2000 bis $3FFF (8192-16383) aufhalten, der Videoram liegt bei $0400-$07FF (1024-2047). Wollen Sie Ihre Graphiken in anderen Bereichen ablegen, so müssen Sie entsprechende Änderungen vornehmen.

b) Graphikart:

Wir müssen uns entscheiden, ob wir unser Bild in Multicolor, die bekanntlich eine höhere Farbauflösung zuläßt, dafür aber weniger Punkte in x-Richtung besitzt, oder ob wir die hochauflösende Graphik wählen mit nur einer Farbe pro 8x8-Punkte - Feld. Der Aufbau und die Unterschiede dieser beiden Graphikarten wurden bereits in Paragraph 3.4 ausgiebig erörtert.

Der zweite Punkt ist schnell gelöst. Wir wollen uns mit der hochauflösenden Graphik beschäftigen. Diese wird durch Setzen der Bits 5 und 6 (Bit 6 muß gleichfalls gesetzt werden!) von Register 17 des VIC und das Löschen von Bit 4 des Registers 22 eingeschaltet. Letzteres ist normalerweise gelöscht, braucht also nicht unbedingt gleich Null gesetzt werden. Das alles passiert durch zwei einfache POKEs, die in der unten folgenden Routine in den Zeilen 10070 und 10080 stehen.
Auch die Adresslagenwahl ist in unserem Falle recht einfach. Da wir uns nicht aus dem unteren 16 K-Bereich unseres Speichers herausbewegen (Sie wissen, daß der VIC nur 16 K addressieren kann (s. # 3.3.2) und im Normalzustand die untersten 16 K für ihn erreichbar sind), brauchen wir keine Veränderungen im Register der CIA 2 zu unternehmen. Lediglich die Basisadresse der Graphikseite muß durch Setzen des 3. Bits von Register 24 in den oberen Teil der 16 K, also nach unseren $2000 (8192) gelegt werden. In dem folgenden Unterprogramm wird dies in Zeile 10090 erreicht:

```
10000 REM ****************************
10010 REM **                        **
10020 REM **  GRAPHIK EINSCHALTEN   **
10030 REM **                        **
10040 REM ****************************
10050 REM
10060 V = 53248 : REM BASISADRESSE - VIDEOCONTROLLER
```

```
10070 POKE V+17, PEEK(V+17) OR (8+3)*16 : REM GRAPHIK EIN
10080 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10090 POKE V+24, PEEK(V+24) OR 8 :  REM  GRAPHIK  NACH  $2000
(8192)
```

Die Befehle AND und OR sind in Kapitel 2 beschrieben. Die Zeilennummern sind extra hoch, um die Routine leicht an ein Programm anzuhängen. Der Textmodus kann auch durch <run/stop> <restore> wieder eingeschaltet werden.

4.2.1.2. Löschen der Graphik

Da in dem Speicher, den nun unsere Graphik einnimmt, vorher stets etwas anderes stand, erhalten wir nach dem Einschalten der Graphik ein recht wildes Durcheinander von Strichen oder sonstigen Punkten. Um nun jeden Graphikpunkt zu löschen, müssen wir jedes Bit des Graphikspeichers auf 0 setzen. Mit einem POKE sind wir in der glücklichen Lage, gleich 8 Bits (Byte), also 8 Punkte gleichzeitig anzusprechen. Es genügt also eine FOR...NEXT - Schleife, in der -angefangen von der Graphik - Startadresse ($2000 = 8192) bis zum Ende bei $3FFF (16383)- alle Bytes gelöscht werden. Da nun ein Bild aber nur 320x200 = 64000 Punkte, also 64000/8 = 8000 Bytes besitzt, brauchen wir tatsächlich nur 8000 Bytes zu löschen:

```
10200 REM  ************************
10210 REM  **                    **
10220 REM  **  GRAPHIK LOESCHEN  **
10230 REM  **                    **
10240 REM  ************************
10250 REM
10260 BG = 8192 : REM BASISADRESSE DES GRAPHIKSPEICHERS
10270 FOR X=BG TO BG+8000 : REM 8000 BYTES
10280 POKE X,0 : REM LOESCHEN
10290 NEXT X
```

Wie Sie sehen, dauert dieser Vorgang recht lange. Haben Sie sich einmal das Graphik-Packet am Ende des Kapitels abgeschrieben, so werden Sie sehen, wie schnell so etwas in Assembler gehen kann. Die Variable BG gehört streng genommen

nicht zu der Routine, genauso, wie die Variable V im oberen Programm. Sie sollten am Anfang eines jeden Programmes gesetzt werden. Wollen Sie die einzelnen Routinen als Unterprogramme laufen lassen, so müssen sie mit dem Befehl RETURN abgeschlossen werden.

4.2.1.3. Löschen der Farbe

Die Farbe liegt bei der hochauflösenden Graphik stets im Videoram (s. # 3.4). Dabei bestimmen die obersten 4 Bits eines jeden Bytes die Farbe der gesetzten Punkte in der Graphikseite, die unteren 4 dagegen die Farbe der nicht gesetzten Punkte, also quasi die der Hintergrundfarbe. Da der Videoram vor dem Einschalten der Graphik den Text enthielt, zeigen sich auch nach dem Löschen noch kleine Farbquadrate an den Stellen, an denen vorher Text stand. Um auch diese zu eliminieren, müssen wir im Videoram die Farbe einheitlich setzen. Dies wird in der folgenden Routine vorgenommen:

```
10400 REM   **********************
10410 REM   **                  **
10420 REM   **  FARBE LOESCHEN  **
10430 REM   **                  **
10440 REM   **********************
10450 REM
10460 BF = 1024 : REM BASISADRESSE DES VIDEORAM
10470 FA = 6*16 + 7 : REM PUNKT-FARBE=BLAU/HINTERGRUND=GELB
10480 FOR X=BF TO BF+1000 : REM 1000 BYTES
10490 POKE X, FA : REM MIT PUNKT- UND HINTERGRUNDFARBE
10500 NEXT X
```

Hier gilt natürlich das Gleiche bezüglich der Variablen BF, wie im vorigen Programm dargelegt. FA ist ebenfalls eine Variable, die der Routine vom übergeordneten Programm übergeben wird und den Wert enthält, der in jedes Byte des Videoram geschrieben werden soll und damit Punkt- und Hintergrundfarbe bestimmt. Sie sollten (besonders hier) ein wenig an den Programmen verändern, um sie richtig zu verstehen. Dies allerdings muß mit der nötigen Vorsicht geschehen, da wir uns direkt im Herz des Rechners befinden. Lassen Sie

beispielsweise BF gleich 0 werden, so wird Ihr Computer nicht zögern, sich von Ihnen zu verabschieden, da Sie direkt die Null-Seite des Speichers manipulieren, das "Kurzzeitgedächtnis" des Betriebssystems.

4.2.1.4. Ausschalten der Graphik

Bislang konnten Sie sich immer nur durch <run/stop> <restore> aus der Graphikanzeige retten. Doch wird dadurch zwangsläufig Ihr Programm beendet, was nicht unbedingt im Sinne des Erfinders ist. Um diese Funktion regulär in unsere Routinensammlung aufzunehmen, müssen wir sämtliche Veränderungen rückgängig machen, die wir in dem Teil "Graphik einschalten" unternommen haben:

- Bits 5/6 - Register 17 löschen
- Bit 4 - Register 22 löschen
- Bit 3 - Register 24 löschen

Dies geschieht im folgenden Programm:

```
10600 REM ****************************
10610 REM **                        **
10620 REM **  GRAPHIK AUSSCHALTEN   **
10630 REM **                        **
10640 REM ****************************
10650 REM
10660 V = 53248 : REM BASISADRESSE - VIDEOCONTROLLER
10670 POKE V+17, PEEK(V+17) AND 255-6*16 : REM GRAPHIK AUS
10680 POKE V+22, PEEK(V+22) AND 255-1*16 : REM MULTICOLOR AUS
10690 POKE V+24, PEEK(V+24) AND 255-8 : REM ZEICHENSATZ
WIEDER NACH $1000 (4096)
```

Damit haben wir alle wichtigsten Dinge, um die Graphik zu bedienen. Nun können wir uns den schwierigeren Zusammenhängen widmen, die uns ermöglichen, auch etwas auf unserem Bild darzustellen.

4.2.2. Einfache Figuren in der Punktgraphik

Nachdem wir uns mit den Dingen beschäftigt haben, die wir zum Ein- und Ausschalten der Graphik benötigen, kommen wir nun zu den ersten Gehversuchen der Graphikprogrammierung. Angefangen mit der Darstellung eines einfachen Punktes auf dem Bildschirm gehen wir über zu den geometrischen Grundformen der Linie und des Kreises, aus denen näherungsweise fast alle anderen Figuren hergestellt werden können.

4.2.2.1. Punkt

Wir wollen, wie an anderer Stelle schon des öfteren erwähnt, das gesamte Graphikfeld in sogenannte Koordinaten unterteilen. Dabei stellt der erste Wert stets die x-Koordinate, also die Anzahl der Punkte zwischen dem jeweiligen Punkt und dem linken Bildschirmfensterrand (0-319). Der zweite genannte Wert ist dann der y-Anteil der Koordinate, also die Anzahl der Punkte zwischen dem Punkt und der oberen Bildschirmkante (0-199). Der Nullpunkt (Koordinaten: 0,0) liegt demnach in der oberen linken Ecke des Fensters. Die untere rechte Ecke dagegen besitzt die Koordinaten 319,199.
Soweit, sogut. Doch dies ist unsere Vereinbarung. Wir können dem Computer selbst nicht die entsprechenden Koordinaten angeben, um einen Punkt zu bestimmen. Wenn Sie die entsprechenden Kapitel gelesen haben (# 3.4.2.), so kennen Sie den Aufbau der hochauflösenden Graphik und ihre Speicherorganisation. Um nun aus den angegebenen Koordinaten auf das Byte und das Bit zu schließen, das den betreffenden Punkt bestimmt, müssen wir zunächst einige Umrechenarbeit leisten. Sie brauchen die folgenden Ausführungen nicht unbedingt zu verstehen. Den Interessierten unter Ihnen sei die Herleitung der im folgenden Programm verwendeten Formel dargelegt.
Lassen Sie uns zunächst einmal den Einfluß der y-Koordinate untersuchen:
Um die Nummer der Graphikzeile (eine Zeile besteht aus 8 Reihen) zu berechnen, in der sich der Punkt befindet, müssen wir die y-Koordinate lediglich durch 8 teilen (ohne Rest):
zeilennummer = INT(yK/8)

Da jede Zeile aus 320 Bytes besteht (jede Reihe besteht aus 320/8 = 40 Byte), müssen wir diese Nummer mal 320 nehmen, um die Startadresse der betreffenden Zeile relativ zur Startadresse des Graphikspeichers zu erhalten:

```
zeilenadresse = 320 * INT(yK/8)
```

Der Rest der eben durchgeführten Division stellt nun die Nummer der Reihe in dieser Zeile dar und muß nur noch hinzuaddiert werden:

```
reihenadresse = 320 * INT(yK/8) + (y AND 7)
```

Der Einfluß der x-Koordinate ist etwas schwieriger, da hier nicht nur einzelne Bytes, sondern sogar die Bits unterschieden werden müssen:
Als erstes berechnen wir die Adresse des angesprochenen Bytes relativ zur Startadresse der betreffenden Reihe (s.o.). Wir rechnen:

```
byteadresse = 8 * INT(xK/8)
```

Nun berechnen wir die Position des gewünschten Bits in dem betreffenden Byte durch Erstellung einer Maske. Das jeweilige Bit wird in der Maske gesetzt, alle anderen sind gleich 0:

```
maske = 2↑(7-(xK AND 7))
```

Diese Einzelteile werden - wie in der folgenden Routine gezeigt - zusammengesetzt:

```
10700 REM ************************
10710 REM **                    **
10720 REM **  PUNKTBERECHNUNG   **
10730 REM **      (SETZEN)      **
10740 REM ************************
10750 REM
10760 RA = 320 * INT(YK/8) + (YK AND 7)
10770 BA = 8 * INT(XK/8)
```

```
10780 MA = 2↑(7-(XK AND 7))
10790 AD = SA + RA + BA
10800 POKE AD, PEEK(AD) OR MA
10810 REM
10900 REM ************************
10910 REM **                    **
10920 REM **  PUNKTBERECHNUNG   **
10930 REM **    (LOESCHEN)      **
10940 REM ************************
10950 REM
10960 RA = 320 * INT(YK/8) + (YK AND 7)
10970 BA = 8 * INT(XK/8)
10980 MA = 255 - 2↑(7-(XK AND 7))
10990 AD = SA + RA + BA
11000 POKE AD, PEEK(AD) AND MA
11010 REM
11020 REM INTERNE PARAMETER:
11030 REM ******************
11040 REM RA: REIHENADRESSE
11050 REM BA: BYTEADRESSE
11060 REM MA: MASKE
11070 REM AD: ZIELADRESSE
11080 REM
11090 REM VORZUGEBENDE PARAMETER:
11100 REM ************************
11110 REM SA: GRAPHIKSPEICHERSTARTADRESSE (Z.B. 8192)
11120 REM XK: X-KOORDINATE
11130 REM YK: Y-KOORDINATE
```

Wie Sie sehen, unterscheiden wir hier zwischen dem Setzen und dem Löschen eines Punktes in HGR. Tatsächlich müssen diese Fälle getrennt behandelt werden. Die Variable SA gibt die Anfangsadresse des betreffenden Graphikspeichers an und wird bei uns stets bei 8192 gehalten. Natürlich werden die beiden Routinen wie die im letzten Abschnitt vorgeführten meist als Unterprogramme verwendet und enden daher zum größten Teil mit einem RETURN. Dies erkennen Sie bereits in dem folgenden Programm, das die inzwischen vorgeführten Routinen anwendet:

```
100 REM     ******************
110 REM     **              **
120 REM     **  SINUSKURVE  **
130 REM     **              **
140 REM     ******************
150 REM
160 V=53248 : REM STARTADRESSE DES VIC
170 SA = 8192 : REM STARTADRESSE DES GRAPHIKSPEICHERS
175 POKE V+32, 10 : REM RAHMENFARBE
180 GOSUB 10000 : REM GRAPHIK EINSCHALTEN
190 GOSUB 10200 : REM GRAPHIK LOESCHEN
200 FA = 7*16 + 2 : GOSUB 10400 : REM FARBE SETZEN
210 YK = 100 : REM X-ACHSE ZEICHNEN
220  FOR XK=0 TO 319
230   GOSUB 10700 : REM PUNKT ZEICHNEN
240  NEXT XK
250 XK = 160 : REM Y-ACHSE ZEICHNEN
260  FOR YK=0 TO 199
270   GOSUB 10700 : REM PUNKT ZEICHNEN
280  NEXT YK
290  FOR XK=0 TO 319 : REM SINUSKURVE ZEICHNEN
300   YK = 70 * SIN (XK/25.5) + 99
310   GOSUB 10700 : REM PUNKT ZEICHNEN
320  NEXT XK
330 POKE 198,0 : REM TASTEN LOESCHEN
340 WAIT 198,255 : REM AUF TASTE WARTEN
350 GOSUB 10600 : REM GRAPHIK AUS
360 END
370 REM
10000 REM ****************************
10010 REM **                        **
10020 REM **  GRAPHIK EINSCHALTEN   **
10030 REM **                        **
10040 REM ****************************
10050 REM
10070 POKE V+17, PEEK(V+17) OR (8+3)*16 : REM GRAPHIK EIN
10080 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10090 POKE V+24, PEEK(V+24) OR 8 :  REM  GRAPHIK  NACH  $2000
(8192)
10100 RETURN
10110 REM
```

```
10200 REM  ************************
10210 REM  **                    **
10220 REM  **  GRAPHIK LOESCHEN  **
10230 REM  **                    **
10240 REM  ************************
10250 REM
10270 FOR X=SA TO SA+8000 : POKE X,0 : NEXT X
10300 RETURN
10310 REM
10400 REM   ***********************
10410 REM   **                   **
10420 REM   **  FARBE LOESCHEN   **
10430 REM   **                   **
10440 REM   ***********************
10450 REM
10460 BF = 1024 : REM BASISADRESSE DES VIDEORAM
10480 FOR X=BF TO BF+1000 : POKE X,FA : NEXT X
10510 RETURN
10520 REM
10600 REM ****************************
10610 REM **                        **
10620 REM **  GRAPHIK AUSSCHALTEN   **
10630 REM **                        **
10640 REM ****************************
10650 REM
10670 POKE V+17, PEEK(V+17) AND 255-6*16 : REM GRAPHIK AUS
10680 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10690 POKE V+24,  PEEK(V+24)  AND  255-8  :  REM  ZEICHENSATZ
WIEDER NACH $1000 (4096)
10695 RETURN
10700 REM ************************
10710 REM **                    **
10720 REM **  PUNKTBERECHNUNG   **
10730 REM **     (SETZEN)       **
10740 REM ************************
10750 REM
10760 RA = 320 * INT(YK/8) + (YK AND 7)
10770 BA = 8 * INT(XK/8)
10780 MA = 2↑(7-(XK AND 7))
10790 AD = SA + RA + BA
10800 POKE AD, PEEK(AD) OR MA
10810 RETURN
```

Bis auf ein paar Änderungen (z.B. wurde in dem Unterprogramm "Graphik löschen", aus Geschwindigkeitsgründen weitestgehendst auf REM-Zeilen verzichtet). Sind die verwendeten Routinen identisch mit den bisher vorgestellten. Probieren Sie ruhig einmal die einzelnen Dinge aus (besonders in der Zeile 300 sollten Sie die verschiedenen Zahlen verändern). Nur so lernen Sie mit ihnen umzugehen.

4.2.2.2. Linie

Schon etwas schwieriger gestaltet sich das Zeichnen einer Linie zwischen zwei beliebigen Punkten auf dem Bildschirm. Man sieht dies zwar täglich in irgendwelchen Programmdemos, macht sich jedoch nie richtig Gedanken darüber, welche Überlegungen dahinter stecken. Das Problem ist: wie stelle ich fest, welche Punkte des Bildschirms auf dieser Linie liegen. Um es zu lösen müssen wir uns ein wenig mit der sogenannten analytischen Geometrie beschäftigen. Bekommen Sie keinen Schreck! Hinter diesem monströsen Begriff verbirgt sich etwas ganz harmloses (jedenfalls in dem Rahmen, der uns hier interessiert) und wenn es Sie nicht so sehr interessieren sollte, etwa weil sich Ihnen damit üble Kindheitserinnerungen verbinden, dann können Sie die folgenden Zeilen ruhig überlesen. Was wir suchen ist eine Formel, mit der wir die Punkte einer Geraden berechnen können, deren Eckpunkte gegeben sind.
Nahezu jeder von uns wird schon einmal in irgendeinem Zusammenhang (meist aus der Schule her) von der sogenannten normierten Geradengleichung gehört haben:

```
y = mx + n
```

wobei x und y die Koordinaten eines Punktes auf einer Geraden, m die Steigung der Geraden und n den Schnittpunkt mit der y-Achse darstellen. Durch einfache Umformung dieser Formel erhalten wir:

```
n = y-mx
```

Kennen wir nun zwei Punkte der Geraden (unsere Endpunkte x1,y1 und x2,y2), so können wir gleichfalls die zwei folgenden Formeln aufsetzen, die wir gleichsetzen können:

$$n = y_1 - mx_1 \quad \text{----} \quad n = y_2 - mx_2$$

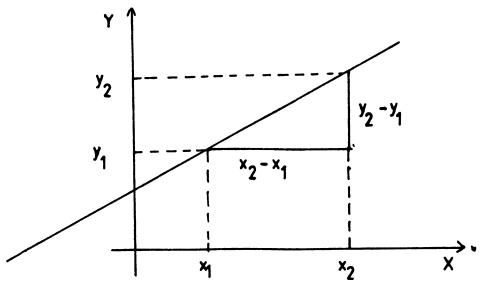
$$\rightarrow \quad y_1 - mx_1 = y_2 - mx_2$$

$$\Leftrightarrow \qquad m = \frac{y_2 - y_1}{x_2 - x_1}$$

Letzere Formel läßt uns nun die Steigung m der obigen Gleichung ausrechnen. Ist n=0, so geht die Gerade durch den Ursprung mit Koordinaten 0,0. Verschieben wir diesen Ursprung der Gerade zu einem Endpunkt, so müssen wir entsprechend die beiden Koordinaten (in diesem Fall x_2 und y_2) zu x und y hinzuaddieren. Die folgende Formel gibt uns nun die endgültige Geradengleichung wieder, die bereits die verschobene Gerade angibt und in die m eingesetzt wurde:

$$y = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_2) + y_2$$

Diese Formel ist die Grundlage des unten dargestellten Programms und wird stückweise in den Zeilen 10970, 11000 und 11020 errechnet, wobei die x-Koordinate XK stets von X2 nach X1 läuft, und für jeden solchen x-Wert der entsprechende y-Wert bestimmt wird. Ein Schaubild mag diese Formel erläutern:



Das einzige Problem bei dieser Formel entsteht, wenn wir eine Senkrechte zeichnen wollen. In diesem Fall wird x1=x2 und damit der Nenner der Steigung gleich 0, was zu einem DIVISION BY ZERO ERROR führt. Wir umgehen diese Unkorrektheit, indem wir in Zeile 10990 verzweigen und dort direkt eine Senkrechte zeichnen. Wie Sie sehen werden und was schon oft erwähnt wurde, kommt ein Basicprogramm in der Geschwindigkeit mit einem Maschinenprogramm natürlich nicht mit. Trotzdem mag Ihnen diese Routine, die Sie ebenfalls als Unterprogramm verwenden können, gute Dienste leisten.

```
100 REM        ***************
110 REM        **           **
120 REM        **  GERADE   **
130 REM        **           **
140 REM        ***************
150 REM
160 V=53248 : SA=8192
170 GOSUB 10000 : REM GRAPHIK EIN
180 FA = 1*16 + 0 : GOSUB 10400 : REM FARBE SETZEN
190 GOSUB 10200 : REM GRAPHIK LOESCHEN
270 X1=110:Y1=120:X2=130:Y2=140:REM ENDPUNKT-KOORDINATEN
280 GOSUB 10900 : REM GERADE
290 WAIT 198,255 : REM AUF TASTE WARTEN
300 GOSUB 10600 : REM GRAPHIK AUS
310 END
320 REM
10000 REM ****************************
10020 REM **  GRAPHIK EINSCHALTEN   **
10040 REM ****************************
10050 REM
10070 POKE V+17, PEEK(V+17) OR (8+3)*16 : REM GRAPHIK EIN
10080 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10090 POKE V+24, PEEK(V+24) OR 8 :  REM  GRAPHIK  NACH  $2000
(8192)
10100 RETURN
10110 REM
```

```
10200 REM  ************************
10220 REM  **  GRAPHIK LOESCHEN  **
10240 REM  ************************
10250 REM
10270 FOR X=SA TO SA+8000 : POKE X,0 : NEXT X
10300 RETURN
10310 REM
10400 REM   ***********************
10420 REM   **  FARBE LOESCHEN  **
10440 REM   ***********************
10450 REM
10460 BF = 1024 : REM BASISADRESSE DES VIDEORAM
10480 FOR X=BF TO BF+1000 : POKE X,FA : NEXT X
10510 RETURN
10520 REM
10600 REM ****************************
10620 REM **  GRAPHIK AUSSCHALTEN  **
10640 REM ****************************
10650 REM
10670 POKE V+17, PEEK(V+17) AND 255-6*16 : REM GRAPHIK AUS
10680 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10690 POKE V+24,  PEEK(V+24)  AND  255-8  :  REM  ZEICHENSATZ
WIEDER NACH $1000 (4096)
10695 RETURN
10700 REM ************************
10720 REM **  PUNKTBERECHNUNG  **
10730 REM **      (SETZEN)      **
10740 REM ************************
10750 REM
10760 RA = 320 * INT(YK/8) + (YK AND 7)
10770 BA = 8 * INT(XK/8)
10780 MA = 2^(7-(XK AND 7))
10790 AD = SA + RA + BA
10800 POKE AD, PEEK(AD) OR MA
10810 RETURN
10900 REM
10910 REM ************************
10930 REM **  GERADE ZEICHNEN  **
10950 REM ************************
10960 REM
10970 DY=Y2-Y1:DX=X2-X1:REM DIFFERENZEN
```

```
10980 YK=Y2:XK=X2:REM Y-START
10990 IFDX=0THEN FOR YK=Y2TOY1 STEPSGN(-DY):GOSUB  11060:NEXT
YK:GOTO 11050:REM SENKR.
11000 DD=DY/DX:REM STEIGUNG
11010 FOR XK=X2 TO X1 STEP SGN(-DX)
11020 ZK=INT(DD*(XK-X2)+Y2):REM GERADENGLEICHUNG
11030 IF   ZK<>YK   THEN   YK=YK+SGN(-DY):GOSUB     11060:GOTO
11030:REM SENKR. ZEICHNEN
11040 GOSUB11060:NEXT XK: REM NAECHSTE X-KOORD.
11050 RETURN
11060 GOSUB10760:XK=XK+1:GOSUB10760:XK=XK-1:RETURN:REM
DOPPELT BREIT ZEICHNEN
```

Sollten Sie es einmal leid sein, stets darauf zu warten, bis der gesamte Bildschirm gelöscht ist, so setzen Sie einfach vor die Zeile 190 ein REM, um diese Prozedur zu unterdrücken. In der obigen Routine werden einige Speicher verwendet, deren Inhalt im folgenden kurz erläutert sei:

Eingabewerte:
X1/Y1 bzw.
X2/Y2 : Endkoordinaten der Linie

interne Werte:
DX/DY : Differenzen der Koordinatenpaare
DD   : Die Steigung m
XK/YK : Koordinaten des aktuellen Punktes
ZK   : Zwischenspeicher

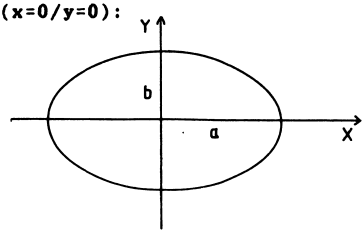
Zwei Punkte müssen hier noch erläutert werden: Zum einen die Funktion SGN, zum anderen die Zeile 11060.
SGN besitzt eine recht nützliche Eigenschaft: Ist die Zahl, die in den Klammern steht positiv, so ist das Ergebnis 1, ist sie negativ, so nimmt es den Wert -1 an (bei 0 wird SGN ebenfalls 0). Die Funktion dient also zur Bestimmung des Vorzeichens.
In Zeile 11060 wird jeder Punkt, der angesteuert wird dupliziert, so daß ein doppelt breiter Punkt entsteht. Dies ist notwendig, da einzelne Punkte, die in x-Richtung keinen Nachbarn besitzen entweder gar nicht oder nur sehr schwach zu sehen sind. So, und jetzt viel Spaß bei Ihrer Linienkreation.

4.2.2.3. Ellipse/Kreis

Eine weitere wichtige und viel verwendete Figur ist der Kreis oder allgemeiner die Ellipse. Mit Ihnen lassen sich schöne Effekte erzeugen. Auch hierzu wird Ihnen im folgenden eine kleine Demonstrationsroutine angegeben, mit der Sie Ellipsen bzw. Kreise zeichnen können. Doch vorher sollten wir für die Interessierten unter Ihnen die mathematischen Grundlagen darlegen, die für das Verständnis dieser Funktion vonnöten sind. Wir werden uns in diesem Buch mit insgesamt zwei Möglichkeiten der Kreis- bzw. Ellipsenerzeugung beschäftigen. Die erste etwas einfacher zu verstehende wird hier angeführt. Die zweite, sie resultiert aus der Verwendung sogenannter Polarkoordinaten und erlaubt das Zeichnen von Kreisbögen, finden Sie unter dem Abschnitt "Kuchendiagramme" im 5. Kapitel (# 5.1.3). Doch hier seien Sie zunächst in die übliche Darstellungsweise eingeführt:
In unserer Routine gehen wir von der sogenannten Mittelpunktsgleichung der Ellipse aus:

$$\frac{x^2}{a} + \frac{y^2}{b} = 1$$

Dabei bedeuten x und y die jeweiligen Koordinaten der Randpunkte der Ellipse. a ist der Radius in x-Richtung und b derjenige in y-Richtung. Der Mittelpunkt der Ellipse liegt im Koordinatenursprung (x=0/y=0):



Um diese Gleichung in unser Programm einzufügen, müssen wir sie zunächst einmal nach y auflösen:

$$y = b * \sqrt{1 - \frac{x^2}{a^2}}$$

Mit dieser recht kompliziert aussehenden Gleichung können wir nun die Punkte eines Ellipsenrandes berechnen. Dabei ist jedoch zu beachten, daß dabei stets nur gleichzeitig ein Bogen von 90 Grad gezeichnet werden kann, da eine Ellipse streng genommen keine Funktion darstellt (Relation). Um die vier anderen Bögen zu zeichnen, müssen wir die Vorzeichen von x und y umkehren. Für x geschieht das im unten stehenden Programm durch die FOR...NEXT-Schleife in Zeilen 11150 - 11190, in dem F2 nacheinander die Werte -1 und 1 annimmt. y dagegen wird in Zeile 11180 negiert. Da die obige Gleichung nur für Ellipsen mit dem Mittelpunkt bei x=0 und y=0 gilt, müssen wir entsprechende Summanden zu x und y hinzufügen, wie unten gezeigt.
Wollen Sie mit unten stehendem Programm einen Kreis zeichnen, so müssen Sie a und b (also die Speicher XR/YR), und damit die beiden Radien gleich groß werden lassen, weil ein Kreis lediglich einen Sonderfall einer Ellipse darstellt.

```
100 REM        ***************
110 REM        **             **
120 REM        **  ELLIPSE    **
130 REM        **             **
140 REM        ***************
150 REM
160 V=53248 : SA=8192
170 GOSUB 10000 : REM GRAPHIK EIN
180 FA = 1*16 + 0 : GOSUB 10400 : REM FARBE SETZEN
190 GOSUB 10200 : REM GRAPHIK LOESCHEN
270 XR=40:YR=20:XM=160:YM=100:REM
X/Y-RADIUS===MITTELPUNKTKOORDINATEN
280 GOSUB 11100 : REM ELLIPSE
290 WAIT 198,255 : REM AUF TASTE WARTEN
300 GOSUB 10600 : REM GRAPHIK AUS
310 END
320 REM
10000 REM ****************************
10020 REM **  GRAPHIK EINSCHALTEN  **
10040 REM ****************************
10050 REM
10070 POKE V+17, PEEK(V+17) OR (8+3)*16 : REM GRAPHIK EIN
10080 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
```

```
10090 POKE V+24, PEEK(V+24) OR 8 :  REM  GRAPHIK  NACH  $2000
(8192)
10100 RETURN
10110 REM
10200 REM  ************************
10220 REM  **  GRAPHIK LOESCHEN  **
10240 REM  ************************
10250 REM
10270 FOR X=SA TO SA+8000 : POKE X,0 : NEXT X
10300 RETURN
10310 REM
10400 REM   ***********************
10420 REM   **  FARBE LOESCHEN  **
10440 REM   ***********************
10450 REM
10460 BF = 1024 : REM BASISADRESSE DES VIDEORAM
10480 FOR X=BF TO BF+1000 : POKE X,FA : NEXT X
10510 RETURN
10520 REM
10600 REM ****************************
10620 REM **  GRAPHIK AUSSCHALTEN  **
10640 REM ****************************
10650 REM
10670 POKE V+17, PEEK(V+17) AND 255-6*16 : REM GRAPHIK AUS
10680 POKE V+22, PEEK(V+22) AND 255-16 : REM MULTICOLOR AUS
10690 POKE V+24,  PEEK(V+24)  AND  255-8  :  REM  ZEICHENSATZ
WIEDER NACH $1000 (4096)
10695 RETURN
10700 REM ************************
10720 REM **  PUNKTBERECHNUNG  **
10730 REM **      (SETZEN)      **
10740 REM ************************
10750 REM
10760 RA = 320 * INT(YK/8) + (YK AND 7)
10770 BA = 8 * INT(XK/8)
10780 MA = 2↑(7-(XK AND 7))
10790 AD = SA + RA + BA
10800 POKE AD, PEEK(AD) OR MA
10810 RETURN
11100 REM
```

```
11110 REM ************************
11120 REM **  ELLIPSE ZEICHNEN  **
11130 REM ************************
11140 REM
11150 FOR F2=-1 TO 1 STEP 2 : REM RECHTS/LINKS-FLAG
11160 FOR X=0 TO F2*(XR) STEP F2
11170 ZK = YR * SQR(1-X^2/XR^2):XK=X+XM : REM KREISGLEICHUNG
11180 YK = YM + ZK:GOSUB 10760:YK = YM - ZK:GOSUB 10760 : REM
PUNKTE OBEN/UNTEN
11190 NEXT X,F2:RETURN
```

Die 4 Übergabeparameter sind:

XM/YM : Koordinaten des Mittelpunktes
XR/YR : x-/y-Radius (a und b)

4.3 Spriteprogrammierung

Eine der wohl faszinierendsten Eigenschaften Ihres Rechners ist die Fähigkeit, insgesamt 8 sogenannte Sprites gleichzeitig auf den Bildschirm zu bringen. Wenn Sie den entsprechenden Abschnitt im dritten Kapitel (# 3.5) gelesen haben, dann besitzen Sie schon einen kleinen Überblick über die Spriteorganisation und die hardwaremäßige Verwirklichung dieser Bildschirmobjekte. Hier nun lernen Sie, wie Sie mit ihnen umgehen und was Sie dabei zu beachten haben.

4.3.1. Erstellung von Sprites

Das erste Problem jeder Spriteprogrammierung ist die Erstellung eines solchen Objektes, denn dies ist natürlich die Voraussetzung für jede Manipulation. Doch schon dies ist eine recht schwieriges Unterfangen, da die Ablegung der Sprites im Speicher relativ kompliziert ist.
Wie sie aus # 3.5 wissen, besitzt ein Sprite eine Auflösung von 24x21 Punkten (in Multicolor: 12x21). Jeder Punkt wird durch ein Bit (zwei Bit bei MC) im Speicher repräsentiert. Je 8 Punkte sind somit in einem Byte zusammengefaßt. Um eine Zeile zu bestimmen, sind damit 24/8 = 3 Bytes notwendig. Diese drei stehen im Speicher direkt hintereinander. Die nächsten drei Bytes definieren dann die zweite Zeile und so fort.
Um ein Sprite zu erstellen, legt man sich vorzugsweise eine Schablone an, die sie im Anhang finden und sich am besten abzeichnen und mehrmals kopieren sollten (oder Sie verwenden den unten stehenden Spriteeditor). In diese Schablone können Sie jeden einzelnen Punkt Ihres Sprites mit Bleistift als kleines Kreuz (bzw. in Multicolor als Ziffer, stellvertretend für die jeweilige Farbe) eintragen und erhalten so ein vollständiges und übersichtliches Bild des zukünftigen Raumschiffes, Vogels oder Buchstabens. Doch Vorsicht! Sie sollten darauf achten, daß Sie jeweils mindestens zwei Punkte nebeneinander zeichnen, da ein einzelner, ohne linken oder rechten Nachbarn nicht, oder nur sehr schwach auf dem Bildschirm erscheint. Dies gilt nicht für Multicolor, da hier ein Punkt sowieso schon doppelte Breite besitzt.
Im Anschluß daran ersetzen Sie jedes Kreuzchen durch eine 1, jedes freie Feld durch eine 0 oder, falls Sie ein Mulcicolor-sprite entwerfen, durch die binäre Zahl, die sich aus der eingetragenen Ziffer ergibt. Nun fassen Sie jeweils 8 dieser Nullen und Einsen zu einem Byte zusammen und errechnen sich nach der Konversionstabelle im Anhang die entsprechende Dezimalzahl. Auf diese Weise erhalten Sie insgesamt 63 Zahlen von 0 bis 255, die den Inhalt der 63 Bytes einer Spritedefinition wiederspiegeln.
Von Basic aus gibt es verschiedene Möglichkeiten, Sprite-definitionen abzulegen und wieder einzulesen. Die erste und wohl einfachste ist die Speicherung dieser 63 Daten in

DATA-Zeilen. Natürlich können Sie sie platzsparend möglichst eng hintereinander packen, doch zweckmäßigerweise und aus Gründen der Übersichtlichkeit sollten Sie in etwa die folgende Form besitzen:

```
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 002,000,064
1030 DATA 001,000,128
1040 DATA 000,129,000
1050 DATA 000,066,000
1060 DATA 000,060,000
1070 DATA 000,126,000
1080 DATA 000,195,000
1090 DATA 001,141,128
1100 DATA 003,044,192
1110 DATA 031,255,248
1120 DATA 062,153,124
1130 DATA 125,066,190
1140 DATA 255,255,255
1150 DATA 001,255,128
1160 DATA 001,189,128
1170 DATA 003,060,192
1180 DATA 015,000,240
1190 DATA 015,000,240
1200 DATA 000,000,000
```

Sie sehen zwar nicht sofort, daß es sich hierbei um das Fahrzeug eines Außerirdischen handelt, doch die 3x21-Bytestruktur wird doch recht deutlich. Jede DATA-Zeile enthält hier die Information für eine Spritezeile. Um jedoch diese Daten in den eigentlichen Speicher zu lesen (in die bekannten Blöcke), müssen wir noch eine kleine Routine hinzufügen, die etwa so aussehen könnte:

```
100 AD = 13*64 : REM ADRESSE BLOCK 13
110 FOR X=0 TO 62
120  READ DT : REM 63 DATEN LESEN
130  POKE AD+X, DT : REM IN BOLCK 13 POKEN
140 NEXT X
```

Dieser Zusatz liest nacheinander die 63 Daten ein und schreibt sie in den Speicher (zu den Blöcken s.u.). Diese Form der Spritespeicherung benötigt jedoch eine ganze Menge Speicherplatz. Eine weitere, platzsparendere Möglichkeit der Speicherung ist das Ablegen eines Sprites auf Diskette oder Kassette z.B. als Sequentielles File. Auf diese Weise können Sie an jeder beliebigen Stelle des Programms ein Sprite einlesen, das Sie auf Diskette gespeichert halten. So ist es ihnen beispielsweise möglich, ganze Datenbanken auf einer Diskette anzulegen, aus denen sich Ihr Programm die notwendigen Teile ausliest.
Am Anfang der Erzeugung eines solchen Definitionsfiles stehen dabei wieder unsere DATAs. Mit Hilfe des folgenden Programms können Sie nun die einzelnen Werte aus den bekannten DATA-Zeilen herauslesen und als Sequentielles File auf Diskette ablegen:

```
10 OPEN 1,8,2,"SPRITE,S,W" : REM FILE ZUM SCHREIBEN EROEFFNEN
20 FOR X=0 TO 62
30  READ DT : REM 63 DATEN LESEN
40  PRINT#1, CHR$(DT) : REM AUF DISKETTE SCHREIBEN
50 NEXT X              : REM (ASCII-FORMAT)
60 CLOSE 1 : REM FILE SCHLIESSEN
```

Der Name des entstehenden Files ist "SPRITE". Um diese Daten wieder einzulesen und direkt in den entsprechenden Speicher zu POKEn, dürfte Ihnen diese Routine behilflich sein:

```
10 AD = 13*64 : REM ADRESSE BLOCK 13
20 OPEN 1,8,2,"SPRITE,S,R" : REM SEQ. FILE ZUM LESEN
EROEFFNEN
30 FOR X=0 TO 62
40  INPUT#1, DT$ : REM DATEN LESEN (ASCII-FORMAT)
50  POKE AD+X, ASC(DT$+CHR$(0)) : REM UND POKEN
60 NEXT X
70 CLOSE 1 : REM FILE SCHLIESSEN
```

selbstverständlich gibt es noch die Möglichkeit, ein Sprite direkt als Programmfile abzuspeichern und ebenso einzuladen. Wie Sie sehen, ist die ganze Sache ziemlich kompliziert und macht Ungeübten einiges zu schaffen. Aus diesem Grunde wird

Ihnen im folgenden ein Programm vorgestellt, das Ihnen die Arbeit der Spriteerstellung wesentlich erleichtert. Dieser Spriteeditor, der teilweise in Basic und Maschinensprache geschrieben wurde, gibt Ihnen komfortable Möglichkeiten in die Hand, ein hochauflösendes Sprite zu erstellen und schließlich in Ihr Programm einzubauen. Er erzeugt Programmfiles, die auf die gleiche Art und Weise gelesen werden können, wie in der letzten Routine demonstriert, wenn Sie die Zeile 20 dort durch die folgende Zeile ersetzen:

```
20 OPEN 1,8,2,"SPRITE,P,R" : REM PROGRAMMFILE ZUM LESEN OEFFNEN
```

Sicher ist es eine ganze Menge Arbeit, dieses Programm abzutippen, aber es lohnt sich. Bevor Sie es starten, sollten Sie es zunächst einmal abspeichern, da verschiedene Basiczeiger "verdreht" werden! Haben Sie sich bei der Eingabe der DATAs vertan, so wird ihnen dies durch eine entsprechende Fehlermeldung kundgetan.

```
100 REM **********************
110 REM **                    **
120 REM **  SPRITEFORMER      **
130 REM **                    **
140 REM **********************
150 REM
160 REM INITIALISIERUNG:
170 REM ****************
180 GOSUB2730:REM MASCHINENROUTINEN EINLESEN
190 POKE 53280,0:POKE 53281,0:REM HINTERGRUND-/RAHMENFARBE
200 POKE650,255:REM ALLE ZEICHEN REPEAT
210 POKE 45,0:POKE 46,80:RUN 220:REM BASICENDE=$5000
220 REM
230 REM MASCHINENROUTINEN:
240 REM ******************
250 IN%=18432:REM INITROUTINE
260 PU%=18632:REM PUNKT EINZEICHNEN
270 NE%=18567:REM KOORDINATENSYSTEM
280 LA%=18503:REM ZEICHENSATZ LADEN
290 SP%=18531:REM ZEICHENSATZ SPEICHERN
300 CA%=18758:REM CATALOG
```

```
310 BE%=18712:REM BEFEHLSIDENTIFIZIERUNG
320 IV%=18830:REM INVERTIEREN
330 VR%=18844:REM VERSCHIEBEN-RECHTS
340 VL%=18870:REM VERSCHIEBEN-LINKS
350 VO%=18895:REM VERSCHIEBEN-OBEN
360 VU%=18935:REM VERSCHIEBEN-UNTEN
370 Q   =  704:REM SPRITEBLOCK-ADRESSE
380 V   =53248:REM VIDEOCONTROLLER
390 REM
400 REM CONTROLZEICHEN:
410 REM ***************
420 C0$=CHR$(147):REM BILDSCHIRM LOESCHEN
430 C1$=CHR$( 19):REM HOME
440 C2$=CHR$(183):REM HOCHSTRICH
450 C8$=CHR$( 99)+CHR$( 99)+CHR$(
99):C3$=CHR$(117)+C8$+CHR$(105):REM OBERER FENSTERR.1
460 C4$=CHR$(106)+C8$+CHR$(107):REM UNTERER SPRITEFENSTERRAND
1
470 C5$=CHR$(117)+C8$+C8$+CHR$(105):REM OBERER RAND 2
480 C8$=CHR$(106)+C8$+C8$+CHR$(107):REM UNTERER RAND 2
490 C9$=CHR$( 98):REM MITTELSTRICH (SENKR)
500 C6$=CHR$( 18):REM RVS ON
510 C7$=CHR$(146):REM RVS OFF
520 NA%=828:REM FILENAMENLAENGE($C800)
530 GA%=186:REM GERAETEADRESSE($BA)
540 TA%=821:REM TASTE/BEFEHLSCODE
550 SG%=  1:REM SPRITEGROESSE
560 YK%=822:REM Y-KOORD
570 XK%=823:REM X-KOORD
580 REM
590 REM FARBEN DEFINIEREN:
600 REM ******************
610 DATA 144,  5, 28,159,156, 30, 31,158
620 DATA 129,149,150,151,152,153,154,155
630 DIM C$(16):FOR Y=0 TO 15:READ X:C$(Y)=CHR$(X):NEXT Y
640 N=1:F(0)=0:F(1)=1:V$=" ":SYS IN%:REM FARBEN/INIT
650 REM
660 REM LOESCHROUTINE (FELDAUFBAU):
670 REM ***************************
680 SYS IN% : REM SPRITE LOESCHEN
690 PRINT C0$
```

```
700 PRINT C1$;SPC(13);C$(7);"SPRITE-CREATION"
710 PRINT SPC(12);C$(1);"(C) BY AXEL PLENGE"
720 PRINT C$(4);:FOR X=1 TO 40:PRINT C2$;:NEXT X
730 PRINTC$(7)"  7"C$(6)"6543210"C$(7)"7"C$(6)"6543210"C$(7)"
7"C$(6)"6543210";
740 SYS NE% : REM NETZ ZEICHNEN
750 GOSUB 1820:PRINT:PRINT:REM STATUSFELD ERSTELLEN
760 PRINT:PRINT:PRINTTAB(30);C3$
770 FOR X=1 TO 3:PRINTTAB(30);C9$;"   "C9$:NEXT
X:PRINTTAB(30);C4$:REM TESTSPRITE1
780 PRINTTAB(27);"  ";C5$;"  ":FOR X=1 TO
5:PRINTTAB(27);"  ";C9$;"      ";C9$:NEXT X
790 PRINTTAB(27);"  ";C8$;"  ";:REM TESTSPRITE2
800 POKE 53248+21,3:X=0:Y=0:REM SPRITES AN/X-,Y-KOORDINATE=0
810 REM
820 REM EINGABESCHLEIFE:
830 REM *****************
840 A=X+2:B=Y+4:GOSUB 2450:REM POSITIONIEREN
850 POKE XK%,X:POKE YK%,Y:F=0:REM KOORDINATEN UEBERGEBEN
860 PRINT C$(7);C6$;" ";CHR$(157);:REM BLINKPHASE AN
870 FOR S=1 TO 50:GETA$:IF A$<>"" THEN 890
880 NEXT S:SYS PU%:FOR S=1 TO 50:GET A$:IF A$="" THEN NEXT
S:GOTO 860:REM AUSSCHALTEN
890 REM
900 REM BEFEHLSERKENNUNG:
910 REM ******************
920 SYS PU%:C=ASC(A$):POKE TA%,C:SYS BE%:S=PEEK(TA%):REM
BEF-UEBERGABE/RUECKMELDUNG
930 REM VERTEILUNG:
940 ON S GOTO 1050,1050,1070,1070,1090,1090
950 ON S-6 GOTO 1110,1110,1910,1910,1910,1910
960 ON S-12 GOTO 1910,1910,1910,1910,650,1360
970 ON S-18 GOTO 1450,1490,1570,1130,2150
980 ON S-23 GOTO 1200,1970,1240,810
990 REM
1000 REM BEFEHLSBEARBEITUNG:
1010 REM ********************
1020 REM
1030 REM CURSORBEWEGUNG:
1040 REM ****************
1050 X=X+1:IF X=24 THEN X=0:GOTO 1090
```

```
1060 GOTO 810:REM RECHTS
1070 X=X-1:IF X<0 THEN X=23:GOTO 1110
1080 GOTO 810:REM LINKS
1090 Y=Y+1:IF Y=21 THEN Y=0
1100 GOTO 810:REM RUNTER
1110 Y=Y-1:IF Y<0 THEN Y=20
1120 GOTO 810:REM HOCH
1130 REM
1140 REM BEENDEN:
1150 REM ********
1160 A=2:B=15:GOSUB 2450:REM POSITIONIEREN
1170 PRINT C6$;C$(7);"BEENDEN?";C7$;C$(6):INPUT T$
1180 IF T$="J" OR T$="JA" THEN SYS 64738:REM KALTSTART
1190 GOTO 690
1200 REM
1210 REM CATALOG:
1220 REM ********
1230 PRINT C0$:SYS CA%:GOSUB 2490:GOTO 690
1240 REM
1250 REM VERSCHIEBUNG:
1260 REM **************
1270 GOSUB 2530:GOSUB 2440:PRINT C$(1);"VERSCHIEBUNG":REM
MELDEFELD
1280 PRINT TAB(27)"NACH:":PRINT TAB(27)"RECHTS(R),":PRINT
TAB(27)"LINKS(L),"
1290 PRINT TAB(27)"OBEN  (O),":PRINT TAB(27)"UNTEN(U):"
1300 GOSUB 2490
1310 IF T$="R" THEN SYS VR%:GOTO1350
1320 IF T$="L" THEN SYS VL%:GOTO1350
1330 IF T$="O" THEN SYS VO%:GOTO1350
1340 IF T$="U" THEN SYS VU%
1350 GOSUB 2530:GOTO 700
1360 REM
1370 REM SPRITEGROESSE:
1380 REM ***************
1390 ON SG%+1 GOSUB 1410,1420,1430,1440
1400 POKE V+23,A:POKE V+29,B:SG%=(SG%+1) AND 3:GOTO 810
1410 A=2:B=2:RETURN
1420 A=0:B=2:RETURN
1430 A=2:B=0:RETURN
1440 A=0:B=0:RETURN
```

```
1450 REM
1460 REM SPRITE INVERTIEREN:
1470 REM ********************
1480 SYS IV% : GOTO 700
1490 REM
1500 REM SPRITE SPEICHERN:
1510 REM ******************
1520 GOSUB 2530
1530 GOSUB 2440:PRINT C6$;C$(1);"SPRITEAB-":PRINT
TAB(27);C6$;"SPEICHERUNG";C7$
1540 GOSUB 1700:IF F=1 THEN F=0:GOTO 1490:REM
EINGABE/FEHLERABFR.
1550 IF F=2 THEN F=0:GOTO 1630:REM FEHLER
1560 SYS SP%:GOTO 1650:REM SPEICHERN
1570 REM
1580 REM SPRITE LADEN:
1590 REM **************
1600 GOSUB 2530
1610 GOSUB 2440:PRINT C6$;C$(1);"SPRITE":PRINT
TAB(27);C6$;"LADEN:";C7$
1620 GOSUB 1700:IF F=1 THEN F=0:GOTO 1570
1630 IF F=2 THEN F=0:GOTO 690
1640 SYS LA%
1650 REM FEHLERABFRAGE (NUR FUER DISK!):
1660 OPEN 1,8,15:INPUT#1,DS,DS$,DT,DB:CLOSE1
1670 IF DS<20 THEN 690:REM OK
1680 PRINT:T$=STR$(DS)+","+DS$+","+STR$(DT)+","+STR$(DB)
1690 GOSUB 2600:PRINT T$:FOR S=1 TO 2000:NEXT S:GOTO 690:REM
BLINKEN
1700 REM
1710 REM NAMENEINGABE:
1720 REM **************
1730 A$="":PRINT:PRINT TAB(27)"FILENAME"C$(6):PRINT
TAB(27);:INPUT A$:T=LEN(A$)
1740 S=VAL(RIGHT$(A$,1))
1750 IF S<>0 AND LEFT$(RIGHT$(A$,2),1)=";" THEN T=T-2:POKE
GA,S:REM GERAETEADR.
1760 IF T=0 THEN F=2:RETURN:REM KEIN NAME
1770 IF T>17 THEN 1800
1780 REM NAMEN AN MASCHINENROUTINEN:
1790 POKE NA%,T:FOR S=1 TO T:POKE
```

```
NA%+S,ASC(MID$(A$,S,1)):NEXT S:RETURN
1800 PRINT CHR$(145);:GOSUB 65535:T$=C6$+"LAENGE!"+C7$:GOSUB
2590:REM FEHLERMELDUNG
1810 PRINT C$(6):F=1:RETURN
1820 REM
1830 REM STATUSFELD ERSTELLEN:
1840 REM **********************
1850 A=27:B=4:GOSUB 2450
1860 GOSUB 2530
1870 A=27:B=5:GOSUB 2450
1880 PRINT TAB(27);C$(7);"FARBEN:"
1890 PRINT TAB(27);C$(2);:FOR S=1 TO 7:PRINT CHR$(163);:NEXT
S:PRINT C$(6)
1900 FOR S=0 TO 1:PRINT TAB(27);"GRDF.";S;":";F(S):NEXT
S:RETURN
1910 REM
1920 REM PLOT:
1930 REM *****
1940 S=((S-12) AND 2)/2:REM PLOTFARBE FESTSTELLEN
1950 T=X/8:AD=INT(T):T=2^(7-8*(T-AD)):AD=Y*3+AD+Q
1960 POKE AD,PEEK(AD) AND (255-T) OR S*T:GOTO 810
1970 REM
1980 REM FARBENWAHL:
1990 REM ***********
2000 PRINTCHR$(147)
2010 A=0:B=4:GOSUB 2450:PRINT
TAB(4);C$(1)"F"C$(2)"A"C$(3)"R"C$(4)"B"C$(5)"E";
2020 PRINT C$(6)"N"C$(7)"W"C$(4)"A"C$(6)"H"C$(2)"L"C$(7)":"
2030 PRINT TAB(4);C$(1);CHR$(172);:FOR S=1 TO 32:PRINT
CHR$(162);:NEXT S:PRINT CHR$(187)
2040 FOR S=1 TO 2:PRINT TAB(4);C6$;CHR$(161);
2050 FOR T=0 TO 15:PRINT C$(T);"  ";:NEXT T:PRINT
C$(1);C7$;CHR$(161):NEXT S
2060 PRINT
TAB(4);C6$;CHR$(161);" 0 1 2 3 4 5 6 7 8 9101112131415";C7$;C
HR$(161)
2070 PRINT:PRINT C$(6);"    FUER GRUNDFARBENNR.(F1/F3): ";
2080 GOSUB 2490:T=ASC(T$)-133:REM FUNKTIONSTASTE
2090 IF T<0 OR T>1 THEN GOSUB 2590:GOTO 690:REM FEHLER
2100 IF T>1 THEN T=T-4
2110 PRINT T:T$="":INPUT "    FARBE ";T$:S=ABS(INT(VAL(T$)))
```

```
2120 IF T$="" OR S>15 THEN GOSUB 2590:GOTO690:REM FEHLER
2130 F(T)=S:POKE V+33,F(0):REM HINTERGRUNDFARBE SETZEN
2140 POKE V+39,F(1):POKE V+40,F(1):GOTO 690:REM SPRITEFARBE
SETZEN
2150 REM
2160 REM BEFEHLSSATZ:
2170 REM ************
2180 POKE V+21,0 : REM SPRITES AUS
2190 PRINT C0$;C6$;C$(2)"               ";C$(7);
2200 PRINT "BEFEHLSSATZ";C$(2);"               ";C7$;
2210 PRINT C$(4);:FOR S=1 TO 40:PRINT CHR$(184);:NEXT S:PRINT
2220 PRINT C$(1)" NR.  "C6$"BEFEHL "C7$"-"C$(5)"
FUNKTION"C$(4)
2230 FOR S=1 TO 10:PRINT "----";:NEXT S
2240 PRINT C$(1)" (1)  "C6$"(><...)"C7$"-"C$(5)" CURSORBEWEGU
NGEN"
2250 PRINT C$(1)"      "C6$"(2QWA )"C7$;C$(5)
2260 PRINT C$(1)" (2)  "C6$"(F1-F8)"C7$"-"C$(5)" PLOT IN
FARBEN 0-15"
2270 PRINT C$(1)" (3)  "C6$"(F)    "C7$"-"C$(5)" FARBEN 0-15
F. F1/3 DEF."
2280 PRINT C$(1)" (4)  "C6$"(B)    "C7$"-"C$(5)" BEFEHLSSATZ"
2290 PRINT C$(1)" (5)  "C6$"(G)    "C7$"-"C$(5)" SPRITE
GROESSE"
2300 PRINT C$(1)" (6)  "C6$"(I)    "C7$"-"C$(5)" SPRITE
INVERTIEREN"
2310 PRINT C$(1)" (7)  "C6$"(V)    "C7$"-"C$(5)" SPRITE
VERSCHIEBEN"
2320 PRINT C$(1)" (8)  "C6$"(L)    "C7$"-"C$(5)" SPRITE
LOESCHEN"
2330 PRINT C$(1)" (9)  "C6$"(CTRLG)"C7$"-"C$(5)" GET-SPRITE
LADEN"
2340 PRINT C$(1)"(10)  "C6$"(CTRLS)"C7$"-"C$(5)" SAVE-SPRITE
SPEICHERN"
2350 PRINT C$(1)"(11)  "C6$"(C)    "C7$"-"C$(5)" DIREKTORY/CA
TALOG"
2360 PRINT C$(1)"(12)  "C6$"(CTRLX)"C7$"-"C$(5)" BEENDEN"
2370 GOSUB 2490:POKE V+21,3:GOTO 690:REM WARTEN+SPRITES AN
2380 REM
2390 REM UNTERPROGRAMME:
2400 REM ***************
```

```
2410 REM
2420 REM POSITIONIERUNG:
2430 REM ****************
2440 A=27:B=5:REM MELDEFELD
2450 PRINT C1$;:FOR S=2 TO B:PRINT:NEXT S:PRINT
TAB(A);:RETURN
2460 REM
2470 REM TASTENEINGABE:
2480 REM ***************
2490 WAIT 198,255:GET T$:RETURN
2500 REM
2510 REM MELDEFELD LOESCHEN:
2520 REM ********************
2530 GOSUB 2440
2540 FORS=1TO6:PRINTTAB(27);:FORT=1TO4:PRINT"   ";:NEXT T:
PRINT:NEXT S :REM MELDEFELD LOESCHEN
2550 RETURN
2560 REM
2570 REM FEHLERBLINKEN:
2580 REM ***************
2590 T$="UNZULAESSIG!"
2600 A=4:B=18:GOSUB2450:PRINTC$(1):FOR S=1 TO 9:PRINT
TAB(4)T$:GOSUB 2630:PRINT CHR$(145);
2610 PRINT TAB(4)"            ";
2620 PRINT CHR$(145):GOSUB 2630:NEXT S:PRINT
TAB(4)"            ":F=1:RETURN
2630 FOR T=1 TO 75:NEXT T:RETURN:REM WARTESCHLEIFE
2640 REM
2650 REM ************************
2660 REM **                    **
2670 REM **  MASCHINENROUTINEN  **
2680 REM **                    **
2690 REM ************************
2700 REM
2710 REM DATAS WERDEN NACH DEM STARTEN GELOESCHT !!!
2720 REM
2730 FOR I = 1 TO 16 : READ X : NEXT I : REM VORDERE DATAS
UEBERSPRINGEN (FARBEN)
2740 FOR I = 18432 TO 18969
2750 READ X : POKE I,X : S=S+X : NEXT
2760 DATA 162, 62,169,  0,157,192,  2,202, 16,250,169, 11
```

```
2770 DATA 141,248,  7,141,249,  7,169, 16,141,  0,208,169
2780 DATA 163,141,  1,208,169,  7,141,  2,208,169,201,141
2790 DATA   3,208,169,  3,141, 16,208,141, 21,208,169,  2
2800 DATA 141, 23,208,141, 29,208,169,  0,141, 27,208,141
2810 DATA  28,208,169,  1,141, 39,208,141, 40,208, 96,173
2820 DATA  60,  3,162, 61,160,  3, 32,249,253,169,  2,166
2830 DATA 186,160,  0, 32,  0,254,169,  0,162,192,160,  2
2840 DATA  76,213,255,173, 60,  3,162, 61,160,  3, 32,249
2850 DATA 253,169,  2,166,186,160,  0, 32,  0,254,169,192
2860 DATA 133,  2,169,  2,133,  3,169,  2,162,255,160,  2
2870 DATA  76,216,255,160,  0,169, 13, 32,210,255,152, 72
2880 DATA  56,233, 10,144, 12,168,233, 10,144,  4,168,169
2890 DATA  50, 44,169, 49, 44,169, 32, 32,210,255,152,  9
2900 DATA  48, 32,210,255,104,168,162,  0, 32,206, 72,169
2910 DATA  29, 32,210,255,232,224, 24,208,243,169,165, 32
2920 DATA 210,255,200,192, 21,208,194, 96,174, 55,  3,172
2930 DATA  54,  3,138, 72,152, 72,133,  2, 10,101,  2,133
2940 DATA   2,138,160,255, 56,233,  8,200,176,251,105,  8
2950 DATA 170,152, 24,101,  2,168,169,  0, 56,106,202, 16
2960 DATA 252, 57,192,  2,208,  3,160,  0, 44,160,  6,162
2970 DATA   6,185, 12, 73, 32,210,255,200,202,208,246,104
2980 DATA 168,104,170, 96,146, 31,111, 31,146,157, 18, 28
2990 DATA  32, 31,146,157,173, 53,  3,162,  0,160, 28,232
3000 DATA 221, 43, 73,240,  3,136,208,247,142, 53,  3, 96
3010 DATA  87, 29, 81,157, 65, 17, 50,145,133,137,134,138
3020 DATA 135,139,136,140, 76, 71, 73, 19,  7, 24, 66, 67
3030 DATA  70, 86,169, 36,133,  2,169,  1,162,  2,160,  0
3040 DATA  32,249,253,169,  2,166,186,160,  0, 32,  0,254
3050 DATA 169,  0,162,  0,160, 64,134, 95,132, 96, 32,213
3060 DATA 255,165, 95,164, 96, 32, 55,165,173,  0,  3, 72
3070 DATA 173,  1,  3, 72,169, 61,141,  0,  3,169,227,141
3080 DATA   1,  3, 32,195,166,104,141,  1,  3,104,141,  0
3090 DATA   3, 96,162, 62,189,192,  2, 73,255,157,192,  2
3100 DATA 202, 16,245, 96,162,  0,160,  3, 24,126,192,  2
3110 DATA 232,136,208,249,169,  0,106, 29,189,  2,157,189
3120 DATA   2,224, 63,208,233, 96,162, 63,160,  3, 24, 62
3130 DATA 191,  2,202,136,208,249,169,  0, 42, 29,194,  2
3140 DATA 157,194,  2,138,208,234, 96,162, 62,134,  2,160
3150 DATA  21,132,  3,166,  2,189,132,  2, 72,189,192,  2
3160 DATA 168,104,157,192,  2,152,202,202,202,198,  3,208
3170 DATA 239,166,  2,202,134,  2,224, 59,208,221, 96,162
```

```
3180 DATA   2,134,  2,160, 21,132,  3,166,  2,189,252,  2
3190 DATA  72,189,192,  2,168,104,157,192,  2,152,232,232
3200 DATA 232,198,  3,208,239,198,  2, 16,226, 96
3210 IF S <> 61707 THEN PRINT "FEHLER IN DATAS !!" : END
3220 PRINT "OK" : RETURN
```

```
END OF ASSEMBLY!

                    0010          .LS
                    0020          ;
                    0030          ;MASCHINENROUTINEN:
                    0040          ;******************
                    0050          ;
                    0060          ;
                    0070          .OS
                    0080          .BA $4800          ;STARTADRESSE
                    0090          .MC $0800
                    0100          ;
                    0110          ;SPRUNGADRESSEN UND REGISTER:
                    0120          ;****************************
                    0130          ;
                    0140 CONASS   .DE 1
                    0150 CHROUT   .DE $FFD2          ;ZEICHENAUSGABE
                    0160 FNPAR    .DE $FDF9          ;FILENAMENPARAMETER SETZEN
                    0170 FPAR     .DE $FE00          ;FILEPARAMETER SETZEN
                    0180 SAVE     .DE $FFD8          ;SPEICHERN AUF DISK/KASSETTE
                    0190 LOAD     .DE $FFD5          ;LADEN VON DISK/KASSETTE
                    0200 V        .DE $D000          ;VIDEOCONTROLLER (53248)
                    0210 BLOCK    .DE 704            ;SPRITEBLOCK 11
                    0220 BLOCKN   .DE 11             ;BLOCKNUMMER 11
                    0230 LAENGE   .DE $033C          ;FILENAMENLAENGE
                    0240 MODUS    .DE $0334          ;SPEICHERMODUS
                    0250 TASTE    .DE $0335          ;BEFEHLSTASTENDRUCK
                    0260 YKOORD   .DE $0336          ;FELD-Y-KOORDINATE
                    0270 XKOORD   .DE $0337          ;FELD-X-KOORDINATE
                    0280          ;
                    0290          ;TESTSPRITE LOESCHEN+PARAMETER:
                    0300          ;******************************
                    0310          ;
4800- A2 3E         0320 INIT     LDX #62            ;63 BYTES
4802- A9 00         0330          LDA #$00
4804- 9D C0 02      0340 I1       STA BLOCK,X        ;BLOCK 11 LOESCHEN
4807- CA            0350          DEX
4808- 10 FA         0360          BPL I1
480A- A9 0B         0370          LDA #BLOCKN        ;BLOCK 11
480C- 8D F8 07      0380          STA $07F8          ;POINTER SPRITE 0 AUF 11
480F- 8D F9 07      0390          STA $07F9          ;POINTER SPRITE 1 AUF 11
4812- A9 10         0400          LDA #16
4814- 8D 00 D0      0410          STA V+0            ;SPRITE 0:X-KOORD. HIGHBYTE
4817- A9 A3         0420          LDA #163
4819- 8D 01 D0      0430          STA V+1            ;Y-KOORDINATE
481C- A9 07         0440          LDA #7
481E- 8D 02 D0      0450          STA V+2            ;SPRITE 1:X-KOORD. HIGHBYTE
4821- A9 C9         0460          LDA #201
4823- 8D 03 D0      0470          STA V+3            ;Y-KOORDINATE
4826- A9 03         0480          LDA #%00000011     ;X-K. HIGHBYTES=1
4828- 8D 10 D0      0490          STA V+16
482B- 8D 15 D0      0500          STA V+21           ;SPRITES EINSCHALTEN
482E- A9 02         0510          LDA #%00000010
4830- 8D 17 D0      0520          STA V+23           ;SPRITE 1: Y-VERGROESSERUNG
4833- 8D 1D D0      0530          STA V+29           ;SPRITE 1: X-VERGROESSERUNG
4836- A9 00         0540          LDA #$00
4838- 8D 1B D0      0550          STA V+27           ;SPRITE-PRIORITAET
483B- 8D 1C D0      0560          STA V+28           ;NORMALFARBENSPRITES
483E- A9 01         0570          LDA #$01
4840- 8D 27 D0      0580          STA V+39           ;SPRITE 0: WEISS
4843- 8D 28 D0      0590          STA V+40           ;SPRITE 1: WEISS
4846- 60            0600          RTS                ;ZURUECK
                    0610          ;
                    0620          ;SPRITE LADEN:
```

```
                    0630          ;*************
                    0640          ;
4847- AD 3C 03      0650 LADEN    LDA LAENGE        ;NAMENLAENGE
484A- A2 3C         0660          LDX #L,LAENGE     ;FILENAMENADRESSE LOW-
484C- A0 03         0670          LDY #H,LAENGE     ;HIGHBYTE
484E- 20 F9 FD      0680          JSR FNPAR
4851- A9 02         0690          LDA #$02          ;LOGISCHE FILENUMMER
4853- A6 BA         0700          LDX *$BA          ;GERAETEADESSE
4855- A0 00         0710          LDY #$00          ;SECUNDAERADRESSE
4857- 20 00 FE      0720          JSR FPAR
485A- A9 00         0730          LDA #$00          ;LOAD/VERIFY-FLAG
485C- A2 C0         0740          LDX #L,BLOCK      ;STARTADRESSE (LOWBYTE)
485E- A0 02         0750          LDY #H,BLOCK      ;STARTADRESSE (HIGHBYTE)
4860- 4C D5 FF      0760          JMP LOAD
                    0770          ;
                    0780          ;SPRITE SPEICHERN:
                    0790          ;*****************
                    0800          ;
4863- AD 3C 03      0810 SPEICH   LDA LAENGE        ;FILENAMENLAENGE
4866- A2 3C         0820          LDX #L,LAENGE     ;FILENAMENADRESSE (LOW)
4868- A0 03         0830          LDY #H,LAENGE     ;FILENAMENADRESSE (HIGH)
486A- 20 F9 FD      0840          JSR FNPAR
486D- A9 02         0850          LDA #$02          ;LOGISCHE FILENUMMER
486F- A6 BA         0860          LDX *$BA          ;GERAETEADRESSE
4871- A0 00         0870          LDY #$00          ;SECUNDAERADRESSE
4873- 20 00 FE      0880          JSR FPAR
                    0950          .LS
4876- A9 C0         0960          LDA #L,BLOCK
4878- 85 02         0970          STA *$02          ;STARTADRESSE (LOWBYTE)
487A- A9 02         0980          LDA #H,BLOCK
487C- 85 03         0990          STA *$03          ;STARTADRESSE (HIGHBYTE)
487E- A9 02         1000          LDA #$02          ;NULLSEITENADRESSE DER STARTADRESS
4880- A2 FF         1010          LDX #L,BLOCK+$3F  ;ENDADRESSE (LOWBYTE)
4882- A0 02         1020          LDY #H,BLOCK+$3F  ;ENDADRESSE (HIGHBYTE)
4884- 4C D8 FF      1030          JMP SAVE          ;SPEICHERE SPRITE
                    1450          .LS
                    1460          ;
                    1470          ;ARBEITSFELD HERSTELLEN:
                    1480          ;***********************
                    1490          ;
4887- A0 00         1500 NETZ     LDY #$00          ;ZEILENZAEHLER = 0
4889- A9 0D         1510 N0       LDA #$0D          ;CARRIGE RETURN
488B- 20 D2 FF      1520          JSR CHROUT
488E- 98            1530          TYA
488F- 48            1540          PHA
4890- 38            1550          SEC
4891- E9 0A         1560          SBC #10
4893- 90 0C         1570          BCC N1            ;LEERZEICHEN BEI Y<10
4895- A8            1580          TAY
4896- E9 0A         1590          SBC #10
4898- 90 04         1600          BCC N2            ;Y<20
489A- A8            1610          TAY
489B- A9 32         1620          LDA #'2
489D- 2C            1630          .BY $2C           ;NAECHSTEN BEF. UEBERSPRINGEN
489E- A9 31         1640 N2       LDA #'1
48A0- 2C            1650          .BY $2C           ;NAECHSTEN BEF. UEBERSPRINGEN
48A1- A9 20         1660 N1       LDA #$20          ;" " LEERZEICHEN
48A3- 20 D2 FF      1670          JSR CHROUT        ;AUSGEBEN
48A6- 98            1680          TYA               ;REST
48A7- 09 30         1690          ORA #$30          ;ASCII HERSTELLEN
48A9- 20 D2 FF      1700          JSR CHROUT        ;AUSGEBEN
48AC- 68            1710          PLA
48AD- A8            1720          TAY
48AE- A2 00         1730          LDX #$00
48B0- 20 CE 48      1740 N3       JSR PUNKT         ;EINEN PUNKT ZEICHNEN
48B3- A9 1D         1750          LDA #$1D          ;CURSOR RECHTS
```

```
48B5- 20 D2 FF  1760         JSR CHROUT
48B8- E8        1770         INX               ;NAECHSTER PUNKT
48B9- E0 18     1780         CPX #24           ;24 PUNKTE PRO ZEILE
48BB- D0 F3     1790         BNE N3
48BD- A9 A5     1800         LDA #$A5          ;STRICH (CHR$(165))
48BF- 20 D2 FF  1810         JSR CHROUT
48C2- C8        1820         INY               ;NAECHSTE ZEILE
48C3- C0 15     1830         CPY #21           ;21 ZEILEN
48C5- D0 C2     1840         BNE N0
48C7- 60        1850         RTS
                1860         ;
                1870         ;EINE KOORDINATE ZEICHNEN:
                1880         ;*************************
                1890         ;
48C8- AE 37 03  1900 PUNKT2  LDX XKOORD        ;ANSTEUERUNG DURCH BASIC
48CB- AC 36 03  1910         LDY YKOORD        ;X/Y-KOORDINATE
48CE- 8A        1920 PUNKT   TXA
48CF- 48        1930         PHA
48D0- 98        1940         TYA
48D1- 48        1950         PHA               ;KOORDINATEN RETTEN
48D2- 85 02     1960         STA *$02
48D4- 0A        1970         ASL A
48D5- 65 02     1980         ADC *$02          ;Y-KOORDINATE*3
48D7- 85 02     1990         STA *$02          ;UND RETTEN
48D9- 8A        2000         TXA
48DA- A0 FF     2010         LDY #$FF          ;ZAEHLER=0
48DC- 38        2020         SEC
48DD- E9 08     2030 P3      SBC #$08
48DF- C8        2040         INY
48E0- B0 FB     2050         BCS P3            ;X-KOORDINATE/8
48E2- 69 08     2060         ADC #$08          ;REST
48E4- AA        2070         TAX
48E5- 98        2080         TYA
48E6- 18        2090         CLC
48E7- 65 02     2100         ADC *$02          ;Y*3+INT(X/8)
48E9- A8        2110         TAY
48EA- A9 00     2120         LDA #$00
48EC- 38        2130         SEC               ;EIN BIT SETZEN
48ED- 6A        2140 P0      ROR A             ;RICHTIGES BIT HERAUSSUCHEN
48EE- CA        2150         DEX               ;REST ERNIEDIRIGEN
48EF- 10 FC     2160         BPL P0
48F1- 39 C0 02  2170         AND BLOCK,Y       ;ANDERE BITS DES SPRITEBYTES LOESC
48F4- D0 03     2180         BNE P1            ;BIT (=PUNKT) GESETZT?
48F6- A0 00     2190         LDY #$00          ;NEIN
48F8- 2C        2200         .BY $2C           ;BIT-BEFEHL
48F9- A0 06     2210 P1      LDY #$06          ;BIT GESETZT!
48FB- A2 06     2220         LDX #$06
48FD- B9 0C 49  2230 P2      LDA PKTTAB,Y      ;ZEICHEN AUS PUNKTDARSTELLUNGSTABE
4900- 20 D2 FF  2240         JSR CHROUT
4903- C8        2250         INY
4904- CA        2260         DEX
4905- D0 F6     2270         BNE P2            ;NAECHSTES ZEICHEN
4907- 68        2280         PLA
4908- A8        2290         TAY
4909- 68        2300         PLA
490A- AA        2310         TAX               ;KOORDINATEN WIEDERHOLEN
490B- 60        2320         RTS
                2330         ;
                2340         ;ZEICHEN FUER EINE KOORDINATE:
                2350         ;*****************************
                2360         ;
490C- 92 1F 6F  2370 PKTTAB  .BY 146 031 111 031 146 157
490F- 1F 92 9D
4912- 12 1C 20  2380         .BY 018 028 032 031 146 157
4915- 1F 92 9D
                2390         ;
```

```
                    2400        ;TASTE ALS BEFEHL IDENTIFIZIEREN:
                    2410        ;********************************
                    2420        ;
4918- AD 35 03      2430 BEFIDE LDA TASTE         ;TASTENCODE
491B- A2 00         2440        LDX #$00          ;BEFEHLSCODE
491D- A0 1C         2450        LDY #$1C          ;27-1 BEFEHLE (ZAEHLER)
491F- E8            2460 B1     INX               ;BEFEHLSCODE ERHOEHEN
4920- DD 2B 49      2470        CMP BEFTAB-1,X    ;TASTE MIT TABELLE VERGLEICHEN
4923- F0 03         2480        BEQ B2            ;GEFUNDEN
4925- 88            2490        DEY               ;NAECHSTER BEFEHL
4926- D0 F7         2500        BNE B1            ;NOCH NICHT FERTIG
4928- 8E 35 03      2510 B2     STX TASTE         ;BEFEHLSCODE ALS RUECKMELDUNG
492B- 60            2520        RTS               ;ZURUECK ZU BASIC
                    2530        ;
                    2540        ;BEFEHLSTASTENTABELLE:
                    2550        ;*********************
                    2560        ;
                    2570        ;    W/CRSR-RE/Q/CRSR-LI/A/CRSR-UN
492C- 57 1D 51      2580 BEFTAB .BY 087 029  081  157  065  017
492F- 9D 41 11
                    2590        ;    2/CRSR-OB/ F1/ F2/ F3/ F4
4932- 32 91 85      2600        .BY 050 145   133 137 134 138
4935- 89 86 8A
                    2610        ;    F5/ F6/ F7/ F8/ L / G
4938- 87 8B 88      2620        .BY 135 139 136 140 076 071
493B- 8C 4C 47
                    2630        ;    I/CTRL.S/CTRL.G/CTRL.X/ B
493E- 49 13 07      2640        .BY 073 019   007     024  066
4941- 18 42
                    2650        ;    C / F / V
4943- 43 46 56      2660        .BY 067 070 086
                    2670        ;
                    2680        ;DISKCATALOG:
                    2690        ;************
                    2700        ;
4946- A9 24         2710 CATALG LDA #$24          ;"$" ALS FILENAME
4948- 85 02         2720        STA *$02
494A- A9 01         2730        LDA #$01
494C- A2 02         2740        LDX #$02
494E- A0 00         2750        LDY #$00          ;S.O.
4950- 20 F9 FD      2760        JSR FNPAR
4953- A9 02         2770        LDA #$02
4955- A6 BA         2780        LDX *$BA          ;GERAETEADRESSE
4957- A0 00         2790        LDY #$00
4959- 20 00 FE      2800        JSR FPAR
495C- A9 00         2810        LDA #$00          ;LOAD/VERIFY-FLAG
495E- A2 00         2820        LDX #$00
4960- A0 40         2830        LDY #$40          ;STARTADRESSE
4962- 86 5F         2840        STX *$5F
4964- 84 60         2850        STY *$60
4966- 20 D5 FF      2860        JSR LOAD          ;LADE CATALOG WIE BASICPROGRAMM
4969- A5 5F         2870        LDA *$5F          ;SIMULIERE:
496B- A4 60         2880        LDY *$60          ;BASIC-PROGRAMMSTARTADRESSE
496D- 20 37 A5      2890        JSR $A537         ;BASICZEILEN BINDEN
4970- AD 00 03      2900        LDA $0300
4973- 48            2910        PHA
4974- AD 01 03      2920        LDA $0301         ;ORIGINALEN WARMSTARTVEKTOR
4977- 48            2930        PHA               ; RETTEN
4978- A9 3D         2940        LDA #$3D
497A- 8D 00 03      2950        STA $0300
497D- A9 E3         2960        LDA #$E3
497F- 8D 01 03      2970        STA $0301         ;UND AUF RTS SETZEN
4982- 20 C3 A6      2980        JSR $A6C3         ;LISTBEFEHL AUSFUEHREN
4985- 68            2990        PLA
4986- 8D 01 03      3000        STA $0301
4989- 68            3010        PLA
```

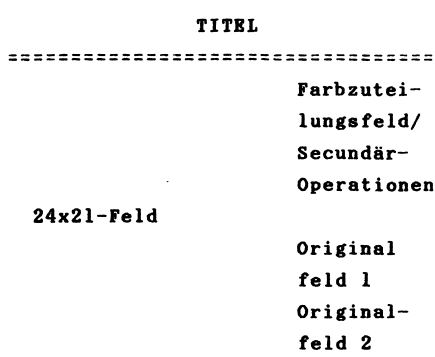
```
498A- 8D 00 03  3020          STA $0300          ;ALTEN VEKTOR WIEDERHOLEN
498D- 60        3030          RTS                ;ZURUECK ZU BASIC
                3040          ;
                3050          ;SPRITE INVERTIEREN:
                3060          ;*******************
                3070          ;
498E- A2 3E     3080 INVERS   LDX #62            ;62+1 BYTES
4990- BD C0 02  3090 IN1      LDA BLOCK,X        ;BYTE HOLEN
4993- 49 FF     3100          EOR #$FF           ;INVERTIEREN
4995- 9D C0 02  3110          STA BLOCK,X        ;ZURUECKSCHREIBEN
4998- CA        3120          DEX
4999- 10 F5     3130          BPL IN1            ;NAECHSTES BYTE
499B- 60        3140          RTS
                3150          ;
                3160          ;SPRITE VERSCHIEBEN:
                3170          ;*******************
                3180          ;
499C- A2 00     3190 RECHTS   LDX #$00           ;BYTEZAEHLER ABSOLUT
499E- A0 03     3200 RE1      LDY #$03           ;BYTEZAEHLER IN ZEILE
49A0- 18        3210          CLC
49A1- 7E C0 02  3220 RE2      ROR BLOCK,X        ;VERSCHIEBEN
49A4- E8        3230          INX
49A5- 88        3240          DEY
49A6- D0 F9     3250          BNE RE2
49A8- A9 00     3260          LDA #$00
49AA- 6A        3270          ROR A              ;LETZTES BIT IN AKU
49AB- 1D BD 02  3280          ORA BLOCK-3,X      ;UND AN ANFANG SETZEN
49AE- 9D BD 02  3290          STA BLOCK-3,X
49B1- E0 3F     3300          CPX #63
49B3- D0 E9     3310          BNE RE1
49B5- 60        3320          RTS                ;FERTIG
                3330          ;
                3340          ;
49B6- A2 3F     3350 LINKS    LDX #63            ;BYTEZAEHLER ABSOLUT
49B8- A0 03     3360 LI1      LDY #$03           ;BYTEZAEHLER IN ZEILE
49BA- 18        3370          CLC
49BB- 3E BF 02  3380 LI2      ROL BLOCK-1,X      ;VERSCHIEBEN
49BE- CA        3390          DEX
49BF- 88        3400          DEY
49C0- D0 F9     3410          BNE LI2
49C2- A9 00     3420          LDA #$00
49C4- 2A        3430          ROL A              ;LETZTES BIT IN AKU
49C5- 1D C2 02  3440          ORA BLOCK+2,X      ;UND ANS ENDE SETZEN
49C8- 9D C2 02  3450          STA BLOCK+2,X
49CB- 8A        3460          TXA
49CC- D0 EA     3470          BNE LI1
49CE- 60        3480          RTS                ;FERTIG
                3490          ;
                3500          ;
49CF- A2 3E     3510 OBEN     LDX #62            ;BYTEZAEHLER ABSOLUT
49D1- 86 02     3520          STX *$02
49D3- A0 15     3530 OB1      LDY #21            ;ZEILENZAEHLER
49D5- 84 03     3540          STY *$03
49D7- A6 02     3550          LDX *$02
49D9- BD 84 02  3560          LDA BLOCK-60,X     ;ERSTES BYTE HOLEN
49DC- 48        3570 OB2      PHA                ;IN ZWISCHENSPEICHER 1
49DD- BD C0 02  3580          LDA BLOCK,X        ;BYTE HOLEN
49E0- A8        3590          TAY                ;IN ZWISCHENSPEICHER 2
49E1- 68        3600          PLA                ;ZISCHENSPEICHER 1 HOLEN
49E2- 9D C0 02  3610          STA BLOCK,X        ;IN BYTE ABLEGEN
49E5- 98        3620          TYA                ;ZWISCHENSPEICHER 2 HOLEN
49E6- CA        3630          DEX
49E7- CA        3640          DEX
49E8- CA        3650          DEX                ;DARUEBER LIEGENDES BYTE
49E9- C6 03     3660          DEC *$03           ;NAECHSTE ZEILE
49EB- D0 EF     3670          BNE OB2
```

```
49ED- A6 02     3680          LDX *$02
49EF- CA        3690          DEX
49F0- 86 02     3700          STX *$02          ;NAECHSTE SPALTE
49F2- E0 3B     3710          CPX #59
49F4- D0 DD     3720          BNE OB1
49F6- 60        3730          RTS
                3740          ;
                3750          ;
49F7- A2 02     3760 UNTEN    LDX #02           ;BYTEZAEHLER ABSOLUT
49F9- 86 02     3770          STX *$02
49FB- A0 15     3780 UN1      LDY #21           ;ZEILENZAEHLER
49FD- 84 03     3790          STY *$03
49FF- A6 02     3800          LDX *$02
4A01- BD FC 02  3810          LDA BLOCK+60,X    ;LETZTES BYTE HOLEN
4A04- 48        3820 UN2      PHA               ;IN ZWISCHENSPEICHER 1
4A05- BD C0 02  3830          LDA BLOCK,X       ;BYTE HOLEN
4A08- A8        3840          TAY               ;IN ZWISCHENSPEICHER 2
4A09- 68        3850          PLA               ;ZISCHENSPEICHER 1 HOLEN
4A0A- 9D C0 02  3860          STA BLOCK,X       ;IN BYTE ABLEGEN
4A0D- 98        3870          TYA               ;ZWISCHENSPEICHER 2 HOLEN
4A0E- E8        3880          INX
4A0F- E8        3890          INX
4A10- E8        3900          INX               ;DARUNTER LIEGENDES BYTE
4A11- C6 03     3910          DEC *$03          ;NAECHSTE ZEILE
4A13- D0 EF     3920          BNE UN2
4A15- C6 02     3930          DEC *$02          ;NAECHSTE SPALTE
4A17- 10 E2     3940          BPL UN1
4A19- 60        3950          RTS
                3960          .EN
END OF ASSEMBLY!
```

--- LABEL FILE: --

```
B1 =491F              B2 =4928
BEFIDE =4918          BEFTAB =492C
BLOCK =02C0           BLOCKN =000B
CATALG =4946          CHROUT =FFD2
CONASS =0001          FNPAR =FDF9
FPAR =FE00            I1 =4804
IN1 =4990             INIT =4800
INVERS =498E          LADEN =4847
LAENGE =033C          LI1 =49B8
LI2 =49BB             LINKS =49B6
LOAD =FFD5            MODUS =0334
N0 =4889              N1 =48A1
N2 =489E              N3 =48B0
NETZ =4887            OB1 =49D3
OB2 =49DC             OBEN =49CF
P0 =48ED              P1 =48F9
P2 =48FD              P3 =48DD
PKTTAB =490C          PUNKT =48CE
PUNKT2 =48C8          RE1 =499E
RE2 =49A1             RECHTS =499C
SAVE =FFD8            SPEICH =4863
TASTE =0335           UN1 =49FB
UN2 =4A04             UNTEN =49F7
V =D000               XKOORD =0337
YKOORD =0336
//0000,4A1A,0A1A
```

An dieser Stelle sollte erwähnt werden, daß, wenn Sie keine Zeit oder Lust haben, dieses Programm von Hand einzugeben, eine Diskette mit allen in diesem Buch angeführten Programmen erhältlich ist. Sie sparen sich somit für wenig Geld viel Sorge und Ärger beim Eingeben und der nachherigen Fehlersuche.
Sofern Sie beim Eingeben keinen Fehler gemacht haben, meldet sich das vorliegende Programm wie folgt:



Dieser Aufbau gibt Ihnen ständig die aktuellen Informationen, die Sie über den Zustand Ihres Arbeitssprites benötigen.

24x21-Feld:
In diesem übersichtlichen Arbeitsfeld können Sie mit Hilfe der verschiedenen Funktionen (s.u.) und dem Feld-Cursor Ihr Zeichen herstellen. Die Ziffern am linken Feldrand geben die y-Koordinate bzw. die Reihe, die Ziffern am oberen Rand die x-Koordinate oder das zuständige Bit im jeweiligen Byte des Spritespeichers an. Jedes gesetzte Bit (Punkt) wird als rotes Kästchen dargestellt.

Originalfelder:
Diese Felder stellen Ihr aktuelles Sprite in den jeweils gewählten Farben (s. "F") in der mit "G" gewählten Größe dar, um Ihnen einen anschaulichen Überblick über das spätere Aussehen des Sprites zu ermöglichen. Feld 1 zeigt dabei ständig den momentanen Zustand des Objektes in Normalgröße, Feld 2 dagegen in x- und/oder y-Richtung vergrößert an.

Farbzuteilungsfeld:
Hir wird Ihnen ein Überblick über die den Grundfarben zugeordneten Farben gegeben, die Sie beliebig ändern können (s.u.)

Secundäroperationsfeld:
Hier werden weitere Informationen gegeben oder verlangt, sobald es dem Rechner notwendig erscheint.

---- Befehlssatz ----

Jeder Befehl besteht aus einem Tastendruck o h n e R e t u r n ! (Ausnahmen s.u.):

- B -
Einen kurzen Befehlsüberblick gibt Ihnen dieser Befehl. Nach Betätigung einer Taste kommen Sie zurück zur Spriteeingabe.

- <crsr> -
Mithilfe der Ihnen bekannten Cursorfunktionen können Sie den Feldcursor bewegen.

- 2/Q/W/A -
Zum leichteren Arbeiten besitzen diese Tasten ebenfalls Cursorfunktionen, deren Bewegungsrichtung aus der entsprechenden Tastaturposition resultiert.

- f1...f8 -
Mit diesen Tasten können Sie einen Punkt auf Ihr 24x21-Feld in der diesen Tasten (Grundfarben) zugeordneten Farbe zeichnen.

- F -
F versetzt Sie in die Lage, den Tasten f1-8 (Grundfarben) entsprechende Farben zuzuordnen. Die mit diesen Tasten gezeichneten Punkte werden nun entsprechend in ihrem Originalfeld andersfarbig.
Achtung! Taste f1/2 (=Grundfarbe 0) steht für die Hintergrundfarbe!

- G -
Durch Drücken dieser Taste können Sie die Größe des Sprites, das sich in dem Originalfeld 2 befindet wechseln.

- I -
Dieser Befehl ermöglicht Ihnen Ihr Sprite zu invertieren.

- V -
Sie können ihr Zeichen in Ihrem Feld beliebig verschieben. Dabei geht keine Information verloren, da die Ränder auf der anderen Seite wieder auftauchen.
Nach V drücken Sie R,L,O,U für Rechts, Links, Oben, Unten (Selbstverständlich läßt sich jede Operation direkt im Originalfeld beobachten).

- L -
Löscht Ihr gesamtes Sprite.

- C -
Bringt das Inhaltverzeichnis der Diskette (Direktory) auf den Bildschirm. Bei Drücken der ctrl.-Taste wird die Auflistung verlangsamt. Jede andere Taste bringt Sie wieder zurück.

- <ctrl>S / <ctrl>G -
Abspeichern (<ctrl>S) oder Laden (<ctrl>G) eines Sprites:
Geben Sie den Filenamen und - durch Semikolon(!) abgetrennt - die Gerätenummer an (bei Fehlen wird das zuletzt benutzte Gerät angesprochen). Nun drücken Sie <return>. Haben Sie diese Tasten versehentlich gedrückt, so genügt ein <return>, um wieder zurückzukehren (dies gilt für alle Secundär-operationen, wie V,F,...). Beim Speichern wird ein Programmfile erzeugt, das entweder direkt in den Speicher eingeladen, oder auf die oben beschriebene Weise quasi wie ein Sequentielles File gehandhabt werden kann.

- <ctrl>X -
Sind Sie fertig und wollen beenden, so beantworten Sie anch diesem Befehl die Frage "Beenden??" mit J oder JA, löschen die restliche Zeile und drücken <return>. Jede andere Eingabe bringt Sie wieder zurück.

Ich hoffe, Sie werden viel Spaß haben mit Ihrer neuen Spritecreation, die Ihren Bekannten die Blässe ins Gesicht treiben wird.

4.3.2. Programmierung der Spriteeigenschaften

Das theoretische Wissen über die Art und Weise, wie ein Sprite definiert, eingeschaltet und seine Eigenschaften verändert werden, sollte Ihnen der Paragraph 3.5 vermittelt haben. Jetzt ist es an der Zeit, diese Dinge auch von Basic aus anzuwenden. Dies soll anhand eines kleinen Beispielprogrammes geschehen, das sich fast aller Variationsmöglichkeiten bedient und schon einen kleinen Einblick in die typische Spriteanwendung zur Animation, also der bewegten Bilder, verschaffen soll. Dieses Thema soll zwar erst an späterer Stelle behandelt werden, doch bietet es sich hier geradezu an.
An diesem Beispielprogramm sollten Sie so viel ändern, wie sie wollen. Lediglich bei den Adressen der POKE-Befehle, also dem ersten Parameter, sollten Sie etwas vorsichtiger sein und zunächst einmal überlegen, welche Auswirkungen das haben könnte. Falls es Ihnen jedoch nichts ausmacht, den Rechner nach einem Absturz auszuschalten und das Programm neu zu laden, dann brauchen Sie sich aus dieser Warnung nichts zu machen. Doch hier zunächst einmal das Programm:

```
1000 REM ************************
1010 REM **                    **
1020 REM **  SPRITE-BEISPIEL   **
1030 REM **                    **
1040 REM ************************
1050 REM
1060 V = 53248 : REM BASISADRESSE VIDEOCONTROLLER
1070 POKE V+32,0 : POKE V+33,0 : REM RAHMEN UND HINTERGRUND =
SCHWARZ
1080 REM
1090 REM SPRITEDEFINITION IN BLOCK 13:
1100 REM ******************************
1110 A1 = 13*64 : REM ADRESSE BLOCK 13
```

```
1120 FOR X=0 TO 62
1130 READ DT : POKE A1+X,DT : REM DATEN LESEN UND IN BLOCK 13
POKEN
1140 NEXT X
1150 REM DATAS ERSTES SPRITE:
1160 DATA 000,007,000
1170 DATA 056,013,128
1180 DATA 025,031,224
1190 DATA 126,031,204
1200 DATA 025,012,252
1210 DATA 056,007,000
1220 DATA 000,014,000
1230 DATA 000,014,000
1240 DATA 000,015,128
1250 DATA 000,014,192
1260 DATA 000,030,096
1270 DATA 000,062,048
1280 DATA 000,046,000
1290 DATA 000,079,000
1300 DATA 000,155,128
1310 DATA 001,025,192
1320 DATA 002,024,096
1330 DATA 003,024,096
1340 DATA 127,024,120
1350 DATA 188,024,000
1360 DATA 036,030,000
1370 REM
1380 REM SPRITEDEFINITION IN BLOCK 14:
1390 REM ******************************
1400 A2 = 14*64 : REM ADRESSE BLOCK 14
1410 FOR X=0 TO 62
1420 READ DT : POKE A2+X,DT : REM DATEN LESEN UND IN BLOCK 13
POKEN
1430 NEXT X
1440 REM DATAS ZWEITES SPRITE:
1450 DATA 000,003,010
1460 DATA 000,013,132
1470 DATA 058,031,224
1480 DATA 124,031,170
1490 DATA 058,015,250
1500 DATA 000,007,000
```

```
1510 DATA 000,014,000
1520 DATA 000,014,000
1530 DATA 000,014,000
1540 DATA 000,015,000
1550 DATA 000,015,128
1560 DATA 000,014,160
1570 DATA 000,030,000
1580 DATA 000,047,000
1590 DATA 000,077,128
1600 DATA 000,141,128
1610 DATA 001,057,128
1620 DATA 003,097,128
1630 DATA 127,113,128
1640 DATA 220,025,128
1650 DATA 036,001,224
1660 POKE 2046, A1/64 : REM ZUNAECHST BLOCK 13 FUER SPRITE 6
1670 POKE V+21, 2^6 : REM SPRITE 6 EINSCHALTEN
1680 POKE V+39+6, 1 : REM FARBE SPRITE 6: WEISS
1690 POKE V+27,   0 : REM SPRITE VOR HINTERGRUND
1700 POKE V+28,   0 : REM NORMALFARBEN-SPRITE
1710 POKE V+23, 2^6 : REM SPRITE 6 IN Y-RICHTUNG VERGROESSERT
1720 POKE V+29, 2^6 : REM SPRITE 6 IN X-RICHTUNG VERGROESSERT
1730 POKE V+2*6+1, 101: REM SPRITE 6 Y-KOORDINATE FESTLEGEN
1735 POKE V+2*6  , 100: REM SPRITE 6 X-KOORDINATE ZUR
DEMONSTRATION
1740 POKE V+16,   0 : REM X-KOORDINATE HIGH-BIT LOESCHEN
1745 REM
1750 REM
1760 PRINT CHR$ (30) CHR$(147) : REM GRUEN + BILDSCHIRM
LOESCHEN
1770 FOR X=1 TO 10
1780 PRINT : REM 10 LEERZEILEN
1790 NEXT X
1800 PRINT CHR$(18); : REM RVS ON
1810 FOR X=1 TO 40
1820 PRINT " "; : REM 40 INVERSE, GRUENE LEERZEICHEN
1830 NEXT X
1840 REM
1850 REM
1860 REM **************
1870 REM **          **
```

```
1880 REM **  LAUFEN  **
1890 REM **          **
1900 REM ***************
1910 REM
1920 G = 10 : REM GESCHWINDIGKEIT
1930 F = 1 : REM FARBSTART
1940 FOR X=1 TO 400 STEP G
1950 F = F+.3: REM FARBE WECHSELN
1960 IF F=16 THEN F=1
1970 REM ---------------
1980 POKE V+39+6, F : REM FARBE WECHSELN
1990 POKE 2046, A1/64 : REM SPRITE 6 AUF BLOCK 13
2000 KO=X:IF X>255 THEN KO=X-256:POKE V+16,2^6:GOTO2020:REM
HIGH BIT X-KOORD. SETZEN
2010 POKE V+16, 0 : REM X-KOORDINATE - HIGH BIT LOESCHEN
2020 POKE V+2*6, KO : REM SPRITE BEWEGEN
2030 FOR Y=1 TO 40 : NEXT Y : REM WARTESCHLEIFE
2040 REM ---------------
2050 POKE 2046, A2/64 : REM SPRITE 6 AUF BLOCK 14
2060 KO=X+G/2:IF KO>255 THEN KO=KO-256:POKE V+16,2^6:GOTO
2080:REM HIGH BIT X-KOORD.
2070 POKE V+16, 0 : REM X-KOORDINATE - HIGH BIT LOESCHEN
2080 POKE V+2*6, KO : REM SPRITE BEWEGEN
2090 FOR Y=1 TO 40 : NEXT Y : REM WARTESCHLEIFE
2100 REM ---------------
2110 NEXT X
```

In der ersten Zeile (Z. 1060) wird zunächst wieder die Variable V definiert, eine Prozedur, die Sie wohl bereits kennen und die vor jedem Ihrer Sprite- oder Graphikprogramme durchgeführt werden sollte. Alsdann ändern wir die Rahmen- und die Hintergrundfarbe in schwarz.
Jetzt beginnt die eigentliche Arbeit:

a) Spritedefinition:
   Aus dem vorherigen Abschnitt kennen Sie bereits die ab Zeile 1100 folgende Routine: Die 63 Datenbytes, die ein Sprite definieren, werden aus nachstehenden DATA-Zeilen eingelesen und in den Speicher gePOKEt. Im ersten Fall wird die Definition in Block 13, im zweiten in Block 14 (ab Zeile 1400) eingeschrieben. Die Startadressen dieser

beiden Blöcke wurde zunächst durch Multiplikation mit 64 errechnet und in die Speicher A1 und A2 abgelegt.
Wir haben nun also zwei Sprites definiert, die, wie Sie sehen werden, recht ähnlich aussehen. Es handelt sich dabei in beiden Fällen um einen älteren Herrn mit Pfeife, der gerade seinen Hund ausführt und von einem Vogel begleitet wird. Die beiden Sprites zeigen lediglich zwei Phasen der Bewegung unseres laufenden Männchens, der rauchenden Pfeife und des flatternden Vogels (Sie können sich die beiden Sprite getrennt einmal ansehen, wenn Sie in die "Laufroutine" nacheinander in die Zeilen 2030 und 2090 ein STOP einfügen). Unsere Absicht ist es nicht etwa, diese zwei Sprites gleichzeitig auf den Bildschirm zu bringen, wir wollen vielmehr versuchen, die sich nur durch jene paar Veränderungen unterscheidenden Objekte stets abwechselnd so auf den Bildschirm zu bringen, daß sich aus den Unterschieden eine Bewegung entwickelt.

b) Blockvektor:
Wir verwenden für dieses Vorhaben Sprite 6 und legen als erstes den Block fest, aus dem die Spritedefinition für Sprite 6 stammen soll. Zu Beginn soll Block 13 die Definition liefern. Aus diesem Grunde müssen wir in das sechste der letzten 8 Bytes des Videoram, also als Spritedefinitionsvektor, eine 13 POKEn (s. # 3.5.3), was in Zeile 1660 geschieht.

c) Einschalten:
Doch damit haben wir längst noch nicht alles getan, um ein Sprite sichtbar zu machen. Wir müssen es zumindest noch einschalten. Dies wird in unserem Programm in Zeile 1670 unternommen. Hier wird in das Register 21 des VIC, das hierfür bekanntlich zuständig ist, der Wert 2hoch6 = 64 gePOKEt. Damit wird hier das 6. Bit gesetzt als Zeichen, daß nun Sprite 6 eingeschaltet ist.

d) Farbe:
Trotzdem werden wir höchstwahrscheinlich noch nicht viel zu sehen bekommen. Dies liegt meist daran, daß es noch außerhalb des Bildschirmfensters liegt. Darum werden wir uns später kümmern. Erst wollen wir die Farbe des Sprites

festlegen. Hierfür sind bekanntlich die Register 39-46 des VIC zuständig. In Zeile 1680 wird in das sechste dieser 8 Register (also in Register 39+6 = 45) der Wert 1 für weiß (s. Anhang) gebracht, um diese Festlegung für das Sprite 6 vorzunehmen.

e) Priorität:

Wir entscheiden uns nun in Zeile 1690, ob wir unser Sprite vor oder hinter den Hintergrundzeichen sehen wollen. Mit anderen Worten: wird unser Sprite von dem später gezeichneten grünen Strich verdeckt oder verdeckt es diesen? Wir haben uns für den letzteren Fall entschieden und das entsprechende Bit im VIC-Register 27 gelöscht. Mit POKE V+27, 2^6 jedoch können Sie diesen Sachverhalt ändern. Probieren Sie einmal! Lesen Sie hierzu auch # 3.5.4.3

f) Multicolor:

Wir haben unsere zwei Spritedefinitionen als Normalfarben-sprites geplant und entwickelt. Daher müssen wir ebenfalls diesen Modus einschalten. Dieses Kriterium entscheidet sich, wie Sie sehen, bereits sehr früh, schon bei der eigentlichen Konstruktion.

g) Vergrößerung:

Und noch eine Entscheidungen müssen wir treffen, bevor wir zur Tat schreiten können: Wollen wir das besagte Objekt in Originalgröße oder in irgendeine Richtung gedehnt anzeigen? Um die einzelnen Details der zwei Sprites besser erkennen zu können, wählten wir eine Vergrößerung des Objektes in x- und in y-Richtung und setzten in den Zeilen 1710/1720 die für Sprite 6 zuständigen Bits der Register 23 und 29 des VIC (s. # 3.5.4.2). Doch auch in Originalgröße sieht das Ganze recht lustig aus (erreichbar durch Ersetzen des Wertes 2hoch6 durch 0). Versuchen Sie doch einmal eine Vergrößerung nur in x- oder y-Richtung!

h) Positionierung:

Endlich ist es soweit! Jetzt schreiten wir zur Tat und bringen unser Sprite sichtbar auf den Bildschirm. Nun

nämlich legen wir die Koordinaten fest, bei denen sich das Objekt befinden soll. Später werden diese Angaben noch verändert, doch seien Sie hier schon der Vollständigkeit halber beigefügt. Wie Sie in # 3.5.4.1 erfahren haben, sind hierfür die Register 0-16 zuständig, speziell für Sprite 6 die Nummern 2*6 = 12 (Low-Byte x-Koordinate), 2*6+1 = 13 (y-Koordinate) und 16 Bit 6 (High-Bit x-Koordinate). Diese Register bzw. das 6.Bit von Register 16 werden bei uns in den Zeilen 1730-1740 belegt.

In diesem Stadium schon ist unser Sprite sichtbar. Wir haben also alles Notwendige getan, um ein Sprite zu definieren. All diese Dinge sollten daher in jedem Spriteprogramm auftauchen. Sie sehen, daß es trotz des Komforts gar nicht so leicht ist, Sprites zu bedienen. Doch mit der Zeit werden Sie Routine bekommen und auch Wege finden, wie man die verschiedenen Dinge abkürzen oder gar weglassen kann.
Doch weiter bei unserer Programmbesprechung. Es folgt nun der Teil des Programms (ab Zeile 1750), der die vorherigen Dinge anwendet und für den eigentlich sichbaren Verlauf zuständig ist:
Nach dem Bildschirmlöschen und dem Zeichnen einer grünen Linie startet eine Schleife, die insgesamt 400 mal durchläuft und den Wert X von 1 bis 400 ansteigen läßt (Zeile 1940). Dabei entscheidet G über die Schrittgröße pro Schleifendurchlauf.
In dieser Schleife passiert eine ganze Menge:
Zunächst wird der Speicher F um 0,3 erhöht (Zeile 1950). F wird verwendet, um die Farbe des Sprites in der Schleife zu verändern (Z. 1980). Da dabei nur ganze (integer-) Werte verwendet werden, ändert sich die Farbe nur dann, wenn sich der ganzzahlige Teil von F verändert. Dies geschieht somit etwa jede 3. Schleife.
Als Nächstes wird die Hauptaufgabe dieses Programmteils wahrgenommen: die Bewegung und der Definitionswechsel.
Zunächst zum Definitionswechsel:
Wie Sie wissen, ist für unser Sprite die Speicherstelle 2046 der Zeiger auf den jeweiligen Block, aus dem die Definition entnommen werden soll. Weiter oben hatten wir die beiden Objekte in den Blöcken 13 und 14 niedergelegt. Um nun eine Bewegung des Sprites an sich zu erhalten, müssen wir stets

zwischen den beiden Definitionen hin und her schalten. An dieser Stelle (Z. 1990) wird deswegen der Wert 13 in 2046 gePOKEt, um den Vektor dorthin zu legen, es wird also das erste Sprite angezeigt. Weiter unten schließlich (Z. 2050) wird nach Definition 14 gewechselt, und das zweite Sprite ist sichtbar.
Um eine kontinuierliche Bewegung unseres Objektes erscheinen zu lassen (in Wahrheit wird es ja stets ruckweise verschoben, unser Auge jedoch nimmt dies als gleichmäßige Bewegung wahr), versetzen wir es jeden Schleifendurchlauf um einen oder mehrere Punkte in x-Richtung (die y-Koordinate bleibt konstant). Hierfür wird der ansteigende Speicher X verwendet. Er wird zunächst in KO (für KOordinate) zwischengespeichert (Z. 2000). Jetzt wird geprüft, ob dieser Wert größer ist als 255. Ist das der Fall, so reicht ein Byte alleine nicht mehr aus und es muß gleichfalls noch das High-Bit der x-Koordinate in Register 16 gesetzt werden. Natürlich wird dabei der Speicher KO um 256 erniedrigt, um ein ILLEGAL QUANTITY ERROR zu verhindern. Nun wird auch das Low-Byte bestimmt (Z. 2020), und das Sprite ist versetzt. Im dritten Teil der Schleife wird diese Prozedur (nach einer Warteschleife) ein weiteres Mal ausgeführt (Z. 2060 ff.).

Im obigen Beispiel wurden Ihnen schon einige Techniken vermittelt, die Ihnen bei Ihrer Sprite-Programmierung behilflich sein werden. Doch einige Themen wurden noch gar nicht angeschnitten: Multicolor, Kollisionsbehandlung und das Arbeiten mit mehreren Sprites gleichzeitig.
Dies soll im nächsten Schritt vorgenommen werden. Sie sollten inzwischen wissen, daß in Multicolor insgesamt 4 Farben pro Sprite verwendet werden können, unter der Einschränkung allerdings, daß hierbei die x-Auflösung um die Hälfte abnimmt, die Gesamtgröße jedoch konstant bleibt und damit die Punktbreite mit dem Faktor 2 zunimmt. Im Speicher wird jeder Punkt durch 2 Bits bestimmt, die das Register angeben, aus dem die jeweilige Farbe entnommen wird, angeben. Diese Register sind die Multicolor-Register 0 und 1 (VIC-Register 37/38) und das für jedes Sprite eigene Farb-Register (VIC-Register 39-46). Sind die beiden Bits gelöscht, so ist diese Stelle des Sprites durchsichtig (s. ## 3.5.3 f.).
Es können zwei verschiedene Kollisionen festgestellt werden:

Sprite - Sprite (VIC-Register 30) und Sprite - Hintergrundzeichen (VIC-Register 31). In den jeweiligen Registern werden bei einer Berührung besagter Objekte die mit den Spritenummern korrespondierenden Bits gesetzt. Sollten Sie diese Dinge noch nicht kennen, so lesen Sie bitte unter # 3.5.4.4 nach. In unserem Fall wird getestet, ob sich zwei Sprite berühren.
Auch hier soll wieder ein Beispielprogramm zur Veranschaulichung des weiter unten gesagten dienen, dessen übertragung sich allein schon aufgrund der hübschen Spritedefinition lohnt:

```
100 REM ************************
110 REM **                    **
120 REM **  SPRITE-KOLLISION  **
130 REM **    (MULTICOLOR)    **
140 REM ************************
150 REM
160 V=53248 : REM VIC-BASISADRESSE
170 A1 = 11*64 : REM BLOCK 11
180 FOR X=0 TO 62
190 READ DT : POKE A1+X, DT : REM DEFINITION LESEN UND POKEN
200 NEXT X
210 POKE 2040, 11 : POKE 2042, 11 : REM BLOCKZEIGER SPRITE 0
UND 2 AUF 11
220 POKE V+28, 2^0 OR 2^2 : REM SPRITE 0 UND 2 AUF MULTICOLOR
230 POKE V+27, 2^0 : REM NUR SPRITE 2 HAT PRIORITAET VOR
HINTERGRUND
240 POKE V+29, 2^0 OR 2^2 : POKE V+23, 2^0 OR 2^2 : REM BEIDE
GROESSER
250 POKE V+21, 2^0 OR 2^2 : REM BEIDE SPRITES AN
260 POKE V+37, 7 : REM SPRITE-MULTICOLOR 0 = GELB
270 POKE V+38, 6 : REM SPRITE-MULTICOLOR 1 = BLAU
280 POKE V+39, 5 : POKE V+41, 8 : REM INDIVIDUALFARBEN SPRITE
0 UND 2
290 POKE V+16,0 : REM X-KOORD. HIGH-BITS LOESCHEN
300 POKE V+1,100 : POKE V+5,100 : REM Y-KOORDINATEN VORSETZEN
310 X2 = 255 : REM START-X-KOORDIANTE SPRITE 2
320 POKE V+0,0 : POKE V+4,X2 : REM X-KOORDIANTEN VORSETZEN
330 POKE V+30,0 : REM KOLLISION RUECKSETZEN
340 FOR X0=1 TO 255 : REM X-KOORD. SPRITE0
```

```
350 X2 = X2-1 : REM X-KOORD. SPRITE 2 ERNIEDRIGEN
360 POKE V+0, X0 : POKE V+4, X2 : REM X-KOORDIANTEN SPRITE
0/2 (LOW-BYTES)
370 POKE V+1, 40*SIN(X0/30)+100 : REM BEWEGEN AUF SINUSKURVE
380 POKE V+5, 40*COS(X0/30)+100 : REM BEWEGEN AUF
COSINUSKURVE
390 IF PEEK(V+30)=(2^0 OR 2^2) THEN GOTO 420
400 NEXT X0
410 REM
420 REM KOLLISIONSROUTINE:
430 REM
440 FOR X=1 TO 255
450 POKE V+39,X : POKE V+41,X : REM SPRITEFARBE WECHSELN
460 NEXT X
470 POKE V+0, 0 : REM SPRITES AUSEINANDERBRINGEN
480 POKE V+30,0 : REM KOLLISION RUECKSETZEN
490 REM
500 REM SPRITE-DATEN
510 REM
520 DATA 008,000,128
530 DATA 010,002,128
540 DATA 002,138,000
550 DATA 064,136,005
560 DATA 080,168,021
570 DATA 084,032,093
580 DATA 117,033,125
590 DATA 125,101,253
600 DATA 127,103,253
610 DATA 123,103,173
620 DATA 123,103,173
630 DATA 123,103,173
640 DATA 123,103,173
650 DATA 123,103,173
660 DATA 123,103,173
670 DATA 123,103,173
680 DATA 123,103,173
690 DATA 127,103,253
700 DATA 095,101,253
710 DATA 021,097,117
720 DATA 005,032,084
```

Zunächst werden hier wieder die üblichen Formalitäten zum Initialisieren der Sprites unternommen. In diesem Programm wird mit nur einer Spritedefinition gearbeitet, die in Block 11 untergebracht wird (Z. 170-200). Dafür aber sollen zwei Sprites gleichzeitig auf dem Bildschirm erscheinen. Wir wählen dafür Sprite 0 und 2. Beide Sprites besitzen jedoch dieselbe Form. Dies wird in Zeile 210 realisiert, in der beide Vektoren für Sprite 0 und 2 auf den Block 11 gerichtet werden. Beide Sprites also beziehen ihre Definition aus diesem Block. Trotzdem aber können Sie in den anderen Parametern weiterhin völlig unabhängig betrieben werden. Dies zeigen z.B. die Zeilen 230, in der beiden unterschiedliche Priorität vor den Hintergrundzeichen zugeschrieben wird, und 280, in der beide Sprite unterschiedliche Teilfarben erhalten. Wie gesagt wollen wir nun mit Multicolorsprites arbeiten. Die in den DATA-Zeilen angegebene Definition wurde deshalb als ein solches entworfen. Nun müssen wir dies gleichfalls dem VIC mitteilen, was in Zeile 220 geschieht. Hier haben wir auch gleich eine bisher noch nicht aufgetretene Schwierigkeit bewältigt. Bisher brauchten wir in allen Registern, die bitweise arbeiten lediglich ein Bit zu setzen. Hier aber wollen wir sowohl Sprite 0 als auch Sprite 2 zu Multicolorsprites erklären. Dies geschieht hier durch ODER- (OR-) Verknüpfung der beiden Einzelwerte 2hoch0 (Bit 0 gesetzt) und 2hoch2 (Bit 2 gesetzt). Sollten Sie damit noch Schwierigkeiten haben, so ziehen Sie sich noch einmal Kapitel 2 zu Gemüte.

Neues kommt auch in den Zeilen 260-280 auf uns zu. Hier werden die verschieden Farben der Multicolorsprites in die jeweiligen Register gelegt. Beachten Sie, daß nur Farbe 3 (Farbkanal 3) für alle Sprites unterschiedlich sein kann!

Unsere Sprites sind bereits eingeschaltet, aber wahrscheinlich noch nicht auf dem Bildschirm zu sehen. Wir werden sie also in das Bildschirmfenster hineinversetzen. Dies geschieht in den Zeilen 290-320. Eigentlich wäre diese Prozedur nicht notwendig, da alle Parameter (außer die High-Bits in Register 16) auch in der folgenden Bewegungsschleife gesetzt werden. Doch hier tritt noch eine weitere Schwierigkeit auf uns zu. Wir wollten die Sprites bei Ihrer Bewegung auf eventuelle Kollisionen überprüfen. Oft ist es aber so, daß sie sich bereits am Anfang berühren, was stets

nach dem Einschalten des Rechners der Fall ist, da alle Koordinaten auf 0 stehen und auch Kollisionen vermerkt werden, wenn die Sprites außerhalb des Sichtbereiches sind. Wenn dies aber der Fall ist, dann würde unser Programm ja bereits ganz am Anfang eine Kollision feststellen und entsprechend verzweigen (s.u.). Wir bringen also die Objekte zunächst einmal auseinander.
Danach - und das ist ebenfalls sehr wichtig - setzen wir das Sprite-Sprite - Kollisionsregister zurück, indem wir den Wert 0 hineinschreiben, da dessen Inhalt bekanntlich so lange erhalten bleibt (genau genommen nur die gesetzten Bits), bis dieses Löschen vorgenommen wurde, auch wenn die Berührung längst nicht mehr besteht.
Danach können wir loslegen. Auch hier (wie in dem vorherigen Beispiel) werden die Sprites wieder in einer Schleife bewegt. Dabei wird der Schleifenzähler (X0) als laufend ansteigender Wert zur Bestimmung der x-Koordinate von Sprite 0 verwendet. Ein von 255 bis 1 absteigender Wert (X2), der stets einmal pro Schleife in Zeile 350 erniedrigt wird, legt die x-Koordinate des Sprite 2 fest (Zeile 360). Um nun Sprite 0 entlang einer Sinus- und Sprite 2 gemäß einer Cosinuskurve "fliegen" zu lassen, werden die y-Koordianten in den Zeilen 360 und 370 durch eine entsprechende Formel errechnet. Verändern Sie hierbei ruhig einmal die verschiedenen Parameter! (Die 40 sorgt für die Größe des Ausschlages (Amplitude), die 30 für die Länge der Kurven (Wellenlänge) und die 100 für die Verschiebung der gesamten Kurve in y-Richtung).
Nun kommen wir wieder zu einem sehr interessanten Teil: Die Kollisionsabfrage. In Zeile 390 wird gefragt, ob Bits 0 und 2 des Registers 30, also des Sprite-Sprite - Kollisions-registers, gesetzt sind. Ist das der Fall, so verzweigt das Programm nach Zeile 440 in die Kollisionsroutine. Hier wird ein kleiner Farb-Effekt erzeugt und die beiden sich berührenden Sprites auseinander gebracht, um das Kollisions-register zu löschen. Weshalb wir die Sprite erst auseinander bringen müssen (wir hätten auch eines ausschalten können), ist weiter oben dargelegt. Damit hätten wir alles Notwendige, um Kollisionen zu bedienen.

Hier wurden Ihnen - ich hoffe recht anschaulich - die

Grundlagen der Spriteprogrammierung vermittelt. Jetzt ist es an Ihnen, zu probieren und zu programmieren. Es gibt tausende von Anwendungen, denen lediglich Ihre Phantasie Grenzen setzt. Selbstverständlich kann in diesem Buch nicht auch nur ein Bruchteil dieser Dinge vermittelt werden. Es bedarf schon einiger Eigeninitiative, um hier Fuß zu fassen. Das aber ist ja gerade das Interessante am Programmieren. Der ganze Reiz wäre fort, wenn alle Möglichkeiten bereits vorgekaut vor Ihnen lägen und Sie sie lediglich zusammenfügen müßten. So bleibt Ihnen genügend Freiraum, in dem Sie Ihrem Forschungsdrang Lauf lassen können.

4.4 Zeichensatz-programmierung

Ein bisher recht stiefmütterlich behandeltes Thema ist das der Zeichensatzveränderung. Enorme Möglichkeiten tun sich hier auf. Ich kenne nur sehr wenige mehr oder minder anspruchsvolle Spiele, die nicht auf diese Eigenschaft Ihres Rechners zurückgreifen oder gar den Kern Ihres Inhalts nicht hierhin verlagern. Hier entscheiden die großen graphischen Möglichkeiten, die fast an die hochauflösende Graphik heranreichen, bei einer 8-9 mal schnelleren Verarbeitungsgeschwindigkeit. Mit der Fähigkeit, einen eigenen Zeichensatz zu creieren wird der Commodore 64 ungeheuer anpassungsfähig. Der große Mangel vieler Computer, die sich mit den vielen verschiedenen Zeichensätzen oder Sonderzeichen in unterschiedlichen Ländern herumschlagen ist hier behoben. Nicht nur den amerikanischen, sondern auch den deutschen, schwedischen, ja sogar russischen, griechischen oder japanischen Zeichensatz kann Ihr Computer auf den Bildschirm bringen. Schriftarten können gewechselt werden und vieles mehr. Das bekannte Textomat beispielsweise, ein erfolgreiches Textverarbeitungsprogramm, bedient sich dieses inneren Schatzes. Sie sehen, diese Rechnereigenart ist nicht zu unterschätzen. Aus diesem Grunde widmen wir uns in diesem Kapitel der programmtechnischen Realisierung der Zeichensatz-

änderung. Diese ist nicht so einfach wie beispielsweise die Spriteprogrammierung und kommt zumindest an einer Stelle nicht ohne Maschinensprache aus, außer, Sie besitzen eine entsprechende Basicerweiterung. Deshalb sollten Computerneulinge zunächst einmal diesen Abschnitt überspringen. Gleichfalls ist es empfehlenswert vor der Lektüre der folgenden Ausführungen den Paragraphen 3.6 gelesen und verstanden zu haben.

Wie Sie in dem Abschnitt 3.6 erfahren haben, besteht eine Zeichendefinition aus 8 Bytes zu je 8 Bits, was eine 8x8-Punkte - Matrix pro Zeichen ergibt. Gleichzeitig sind 512 verschiedene Zeichen speicherbar, maximal jedoch nur 256 zur selben Zeit auf dem Bildschirm darzustellen. Der gesamte Satz von 512 Zeichen besitzt eine Länge von 4 K und liegt normalerweise bei $D000-$DFFF (53248-57343) im sogenannten Zeichensatz-ROM. Er ist verschiebbar durch Änderung der Register 24 des VIC und Register 0, Bits 0/1 der CIA 2 (Adressen: $D018 = 53272 und $DD00 = 56576). Soweit das Wichtigste noch einmal in Kürze.
Bei der Zeichensatzerstellung muß man zwei Fälle unterscheiden. Im ersten Fall sollen nur einige wenige Teile des Zeichensatzes (Sonderzeichen) verändert werden. Bei der anderen Möglichkeit wird der gesamte Satz ausgewechselt. Zunächst zur ersten:

4.4.1. Änderung einiger Zeichen

Wollen wir ein paar Zeichen des Zeichengenerators durch eigene Zeichenformen ersetzen, so müssen wir den originalen Satz zunächst einmal aus dem ROM-Bereich in einen uns angenehmen RAM-Bereich copieren. Anschließend ändern wir die Start - Adresse des Generators auf die in Abschnitt 3.3.2 (das Kapitel 3.3 sollten Sie, um alles richtig zu verstehen, bereits gelesen haben) erläuterte Art und Weise. Dann gehen wir hin und ändern die Zeichen, die wir ersetzen wollten.
Was hier so kurz und einfach erscheint, bedarf doch Einiges an Hintergrundwissen und Programmiertechnik. Grundsätzlich kann man den Zeichengenerator nicht von Basic aus auslesen bzw. copieren. Dies ist dadurch verursacht, daß der Ort von

$D000-$DFFF (53248-57343), in dem er sich befindet, normalerweise alle Eingabe / Ausgabe - Bereiche (E/A-Bereich) umfaßt und somit erst durch Ändern des Registers 1 der CPU (s. # 3.5), also der Speicherstelle 1 eingeschaltet werden muß. Damit ist aber der E/A-Bereich, der von der Interruptroutine des Betriebssystems verwendet wird, lahmgelegt und es käme in Basic zum Absturz des Computers. In Maschinensprache jedoch kann man den Interrupt durch Setzen des Interruptflags der CPU verhindern. Aus diesem Grunde geben wir Ihnen an dieser Stelle ein Maschinenprogramm an, das die Aufgabe des Copierens und des Verschiebens des Zeichengenerators für Sie übernimmt. Letzteres könnte zwar ebensogut von Basic aus geschehen, ist aber so einfacher.

```
 20:                ;
 30:                ;ZEICHENSATZVERSCHIEBUNG:
 40:                ;************************
 50:                ;
 60: C800                *=   $C800    ;STARTADRESSE (51200)
 70: C800           V    =    53248    ;BASISADRESSE VIDEOCONTROLLER
 80: C800           SATZ =    53248    ;BASISADRESSE ZEICHENSATZ
 90: C800           ZIEL =    $3000    ;BASISADRESSE COPIERADRESSE
100: C800 78       START SEI           ;INTERRUPT VERHINDERN
110: C801 A5 01          LDA  $01      ;CPU-REG. 1
120: C803 48             PHA           ;RETTEN
130: C804 29 FB          AND  #%11111011  ;BIT 2 LOESCHEN
140: C806 85 01          STA  $01      ;(ZEICHENGENERATOR AUSLESBAR)
150: C808 A9 D0          LDA  #>SATZ
160: C80A 85 03          STA  $03      ;QUELLADRESSE HIGH-BYTE
170: C80C A9 30          LDA  #>ZIEL
180: C80E 85 05          STA  $05      ;ZIELADRESSE HIGH-BYTE
190: C810 A0 00          LDY  #$00
200: C812 84 02          STY  $02
210: C814 84 04          STY  $04      ;LOW-BYTES = 00
220: C816 A2 20          LDX  #$20     ;ZAEHLER FUER 4 K-BYTE
230: C818 B1 02    COPIE LDA  ($02),Y ;BYTE LADEN
240: C81A 91 04          STA  ($04),Y ;UND COPIEREN
250: C81C C8             INY
260: C81D D0 F9          BNE  COPIE    ;256 MAL
270: C81F E6 03          INC  $03
280: C821 E6 05          INC  $05      ;HIGH-BYTES ERHOEHEN
```

```
C823 CA          DEX
C824 D0 F2       BNE  COPIE    ;4 K COPIEREN
C826 68          PLA
C827 85 01       STA  $01      ;ZEICHENGENERATOR AUS-E/A EIN
C829 AD 18 D0    LDA  V+24     ;ZEICHENGENERATORADRESSE
C82C 29 F1       AND  #%11110001
C82E 09 0C       ORA  #%00001100
C830 8D 18 D0    STA  V+24     ;AUF $3000 SETZEN
C833 58          CLI           ;INTERRUPT WIEDER ERLAUBEN
C834 60          RTS           ;ZURUECK NACH BASIC
```

Und hier der dazugehörige Basiclader:

```
100 FOR I = 51200 TO 51252
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 120,165,  1, 72, 41,251,133,  1,169,208,133,  3
130 DATA 169, 48,133,  5,160,  0,132,  2,132,  4,162, 32
140 DATA 177,  2,145,  4,200,208,249,230,  3,230,  5,202
150 DATA 208,242,104,133,  1,173, 24,208, 41,241,  9, 12
160 DATA 141, 24,208, 88, 96
170 IF S <> 5884 THEN PRINT "FEHLER IN DATAS !!" : END
180 PRINT "OK"
```

Wollen Sie also Teile Ihres Zeichensatzes ändern, so brauchen Sie lediglich diesen Lader einzuladen, RUN zu drücken und anschließend ein SYS 51200 einzutippen, und schon ist der originale Zeichensatz verschoben (Sie können, falls Sie einen Monitor besitzen, natürlich auch direkt das Maschinenprogramm eingeben, abspeichern und später mit LOAD "name",8,1 einladen).
Der ursprüngliche Grund für diese Exkursion war aber das Ziel, verschiedene Zeichen des Zeichengenerators zu verändern. Dies wollen wir jetzt unternehmen.
Aus dem Abschnitt 3.6 kennen wir bereits den 8x8-Aufbau (bzw. 4x8 bei Multicolor) eines Zeichens und seine Abspeicherung in 8 Bytes. Um nun eine Zeichendefiniton auszuwechseln, müssen wir uns zunächst eine eigene überlegen. Dies geschieht am besten auf die gleiche Weise, wie bei den Sprites mit Hilfe eines Zeichenentwurfsblattes, das Sie ebenfalls im Anhang finden. Die Anwendung kennen Sie bereits von den Sprites her. Jede Reihe dieses Zeichenentwurfes bildet dabei genau ein

Byte, jeder gesetzte Punkt ein Bit der Definition. Nun haben wir beispielsweise folgendes Gebilde creiert:

Bit:	7	6	5	4	3	2	1	0
Byte 0:	.	.	.	*	*	*	.	.
Byte 1:	.	*	*	*	.	.	.	.
Byte 2:	*	*	.	.	*	*	*	*
Byte 3:	*	*	.	.	.	.	.	.
Byte 4:	*	*	.	.	*	*	*	*
Byte 5:	.	*	*	*	.	.	.	.
Byte 6:	.	.	.	*	*	*	.	.
Byte 7:	.	.	.	.	.	.	.	.

Wir erhalten damit folgende 8 Bytes:

Byte 0: %0001 1100 = $1C = 028
Byte 1: %0111 0000 = $70 = 112
Byte 2: %1100 1111 = $CF = 207
Byte 3: %1100 0000 = $C0 = 192
Byte 4: %1100 1111 = $CF = 207
Byte 5: %0111 0000 = $70 = 112
Byte 6: %0001 1100 = $1C = 028
Byte 7: %0000 0000 = $00 = 000

Damit taucht auch gleich die nächste Frage auf: Welches Zeichen soll unsere Neuschöpfung ersetzen, und welchem der insgesamt 4 Zeichensätze (s. # 3.6) soll es angehören? In unserem Fall wollen wir statt des normalen Pfundzeichens, das im Groß/Graphikmodus erreichbar ist, nun diese Definition setzen.
Jetzt müssen wir feststellen, wo die entsprechenden 8 Bytes innerhalb des gesamten Zeichengenerators liegen. Dazu verwenden wir die im 3. Kapitel angegebene Formel:

adresse = basisadresse + 8 * bildschirmcode

Die Basisadresse des Zeichengenerators hängt grundsätzlich von dem Bereich ab, in den wir diesen verschoben haben. Da wir uns in unserem Beispiel innerhalb der Adressen $3000-$3FFF (12288-16383) befinden und wir nur den Großschrift Graphikzeichenmodus verändern wollen (die

Definitionen für dessen Zeichen liegen von $3000-$37FF (12288-14335)), liegt dieser erste Wert hier schon einmal fest. Bleibt lediglich der Bildschirmcode: Diesen erfahren wir durch Nachschlagen in der Bildschirmcodetabelle im Anhang. Er ergibt sich für das normale Pfundzeichen zu 28 ($1C) (das inverse Pfundzeichen hätte den Code: 128+28=156). Damit können wir die verschiedenen Werte in unsere Formel einsetzen:

adresse = 12288 + 8*28 = 12512 = $30E0

Damit wissen wir, daß die 8 Bytes von $30E0 bis $30E7 (12512-12519) die Definition für besagtes Pfundzeichen enthalten. Mit einem Speichern der obigen 8 Bytes unseres Sonderzeichens in diese Positionen ändern wir sofort das Aussehen aller Pfundzeichen auf dem Bildschirm. Dies geschieht etwa durch folgendes Basicprogramm (selbstverständlich, nachdem Sie zunächst mit obiger Routine den Zeichensatz copiert und verschoben haben):

```
10 REM ZEICHENAENDERUNG
20 FOR X=0 TO 7
30  READ DT : REM 8 DATEN LESEN
40  POKE 12288 + 8*28 + X, DT : REM UND EINPOKEN
50 NEXT X
60 DATA 28,112,207,192,207,112,28,0
```

Bevor Sie dieses Programm ablaufen lassen, sollten Sie ein paar Pfundzeichen auf den Bildschirm bringen, um die Wirkung direkt zu sehen. Wenn Sie nach dem Starten dann auf den Klein- / Großschrift - Modus umschalten (mit <shift><C=>), dann sehen Sie, daß das Pfundzeichen dieses Satzes aus einem anderen Speicherbereich stammt. Diese Operation können Sie natürlich mit allen anderen Zeichen genauso durchführen und sich damit ein ganzes Reservoir an Sonderzeichen schaffen, die bei der Erstellung von besonders schönen und abwechslungsreichen Graphiken oder beim kommerziellen Gebrauch Verwendung finden.

4.4.2. Änderung eines Zeichensatzes

Wollen wir einen ganzen Zeichensatz auswechseln, so brauchen wir den originalen nicht extra zu copieren. Damit kommt man völlig mit den Möglichkeiten, die in Basic bestehen aus. Man läd einfach den neuen Zeichensatz in einen bestimmten Speicherbereich ein und teilt dem VIC mit, daß er die Definition der verschiedenen Buchstaben etc. von nun ab von dort holen soll.
Doch zunächst einmal muß dieser neue Zeichensatz entstehen, der jetzt von mir aus alle griechischen, russischen, kyrillischen Zeichen oder nur Blocksatz, Elite oder Schreibschrift enthält. Dies ist normalerweise aufgrund der Vielzahl der verschiedenen Zeichen recht mühselig und mancher Programmierer würde ob dieser schier unbewältigbar erscheinenden Arbeit verzweifeln. Jedes Zeichen müßte auf einem Blatt entworfen, in Bytes übersetzt und schließlich entweder per Monitor direkt oder - was noch mühseliger wäre - von Basic aus durch POKEs in den Speicher eingegeben werden. Weil aus diesen Gründen ein vernünftiges Arbeiten mit ganzen Zeichensätzen kaum realisierbar erscheint, werden wir Ihnen an dieser Stelle ein Programm vorstellen - ähnlich dem Spriteeditor -, mit dem Sie auf einfachste Art und Weise solche Tätigkeiten vornehmen können. Auch hier erwartet Sie wieder einiges an Tiparbeit, die sich jedoch ernsthaft lohnt! Halten Sie dies jedoch für verlorene Zeit, so weise ich Sie auch hier wieder darauf hin, daß Sie exklusiv zu diesem Buch eine Diskette mit allen hier aufgeführten Programmen erhalten können. Speziell zum Zeichenformer ist dort auch eine Version für Multicolorzeichen und ein Programmteil "Erläuterungen" vorhanden.
In dem vorliegenden Programm wurde wieder viel Wert auf Dokumentation gelegt, damit Sie die Möglichkeit haben, dieses Programm auch zu verstehen.

```
10 REM **********************
20 REM **                    **
30 REM **  ZEICHENFORMER     **
40 REM **                    **
50 REM **********************
60 REM
70 REM INITIALISIERUNG:
80 REM *****************
90 GOSUB 10000 : REM MASCHINENROUTINEN EINLESEN
100 POKE 53280,0:POKE 53281,0:REM HINTERGRUND-/RAHMENFARBE
110 POKE 763,0:POKE 650,255:REM MODUS=0:ALLE ZEICHEN REPEAT
120 POKE 45,0:POKE 46,80:RUN 130:REM BASICENDE=$5000
130 REM
140 REM MASCHINENROUTINEN:
150 REM ******************
160 IN%=18432:REM INITROUTINE
170 PU%=18633:REM PUNKT EINZEICHNEN
180 NE%=18586:REM KOORDINATENSYSTEM
190 LA%=18487:REM ZEICHENSATZ LADEN
200 SP%=18515:REM ZEICHENSATZ SPEICHERN
210 CA%=18765:REM CATALOG
220 BE%=18689:REM BEFEHLSIDENTIFIZIERUNG
230 Q  =13048:REM TESTZEICHENADRESSE
240 REM
250 REM CONTROLZEICHEN:
260 REM ***************
270 C0$=CHR$(147):REM BILDSCHIRM LOESCHEN
280 C1$=CHR$( 19):REM HOME
290 C2$=CHR$(183):REM HOCHSTRICH
300 C3$=CHR$(117)+CHR$( 99)+CHR$(105):REM OBERER
ZEICHENFENSTERRAND
310 C4$=CHR$(106)+CHR$( 99)+CHR$(107):REM UNTERER RAND
320 C5$=CHR$( 98):REM MITTELSTRICH (SENKR)
330 C6$=CHR$( 18):REM RVS ON
340 C7$=CHR$(146):REM RVS OFF
350 NA%=704:REM FILENAMENLAENGE($02C0)
360 GA%=186:REM GERAETEADRESSE($BA)
370 MO%=763:REM MODUS
380 TA%=764:REM TASTE/BEFEHLSCODE
390 YK%=765:REM Y-KOORD
395 XK%=766:REM X-KOORD
```

```
400 REM
410 REM FARBEN DEFINIEREN:
420 REM ******************
430 DATA 144,  5, 28,159,156, 30, 31,158
440 DATA 129,149,150,151,152,153,154,155
450 DIM C$(16):FOR Y=0 TO 15:READ X:C$(Y)=CHR$(X):NEXT Y
460 N=1:F(0)=0:F(1)=1:V$=" ":SYS IN%:REM FARBEN/INIT
500 REM
510 REM LOESCHROUTINE (FELDAUFBAU):
520 REM ***************************
530 FOR Y=Q TO Q+7:POKE Y,0:NEXT Y:REM TESTZEICHEN LOESCHEN
540 PRINT C0$
550 PRINT C1$;SPC(10);C$(7);"ZEICHEN-NEU-CREATION"
560 PRINT SPC(11);C$(1);"(C) BY AXEL PLENGE"
570 PRINT C$(4);:FOR X=1 TO 40:PRINT C2$;:NEXT X:PRINT
C$(6):PRINT
580 PRINT "   76543210":SYS NE%
590 PRINT "   ";:FOR X=0 TO 7:PRINTC2$;:NEXT X
600 A=15:B=5:GOSUB 9000:REM POSITIONIEREN
610 PRINT C3$:PRINT TAB(15);C5$;CHR$(127);C5$:POKE
55552,F(1):REM TESTZEICHEN MIT FARBE
620 PRINT TAB(15);C4$:GOSUB 3000:X=0:Y=0:REM
STATUSFELD/X/Y-KOORD=0
700 REM
710 REM EINGABESCHLEIFE:
720 REM ****************
730 A=X+3:B=Y+6:GOSUB 9000:REM POSITIONIEREN
740 POKE XK%,X:POKE YK%,Y:F=0:REM KOORDINATEN UEBERGEBEN
750 PRINT C$(7);C6$;" ";CHR$(157);:REM BLINKPHASE AN
760 FOR S=1 TO 50:GETA$:IF A$<>"" THEN 800
770 NEXT S:SYS PU%:FOR S=1 TO 50:GET A$:IF A$="" THEN NEXT
S:GOTO 750:REM AUSSCHALTEN
800 REM
810 REM BEFEHLSERKENNUNG:
820 REM ******************
830 SYS PU%:C=ASC(A$):POKE TA%,C:SYS BE%:S=PEEK(TA%):REM
BEF-UEBERGABE/RUECKMELDUNG
840 REM VERTEILUNG:
850 ON S GOTO 1600,2800,2900,1060,1060,1080,1080,1100,1100
860 ON S-9 GOTO 1110,1110,3100,3100,3100,3100
870 ON S-15 GOTO 3100,3100,3100,3100,500,1700
```

```
880 ON S-21 GOTO 1800,1900,2000,2100,1200,3400
890 ON S-27 GOTO 1300,3200,1400,700
1000 REM
1010 REM BEFEHLSBEARBEITUNG:
1020 REM ********************
1030 REM
1040 REM CURSORBEWEGUNG:
1050 REM ****************
1060 X=X+1:IF X=8 THEN X=0:GOTO 1100
1070 GOTO 700:REM RECHTS
1080 X=X-1:IF X<0 THEN X=7:GOTO 1110
1090 GOTO 700:REM LINKS
1100 Y=(Y+1)AND7:GOTO 700:REM RUNTER
1110 Y=(Y-1)AND7:GOTO 700:REM HOCH
1200 REM
1210 REM BEENDEN:
1220 REM *********
1230 A=2:B=15:GOSUB 9000:REM POSITIONIEREN
1240 PRINT C6$;C$(7);"BEENDEN?";C7$;C$(6):INPUT T$
1250 IF T$="J" OR T$="JA" THEN SYS 64738:REM KALTSTART
1260 GOTO 540
1300 REM
1310 REM CATALOG:
1320 REM *********
1330 PRINT C0$:SYS CA%:GOSUB 9100:GOTO 540
1400 REM
1410 REM VERSCHIEBUNG:
1420 REM **************
1430 GOSUB 8999:PRINT C$(1);"VERSCHIEBUNG:":REM MELDEFELD
1440 PRINT "NACH: RECHTS(R), LINKS(L),":PRINT SPC(6)
"OBEN  (O), UNTEN(U):"
1450 GOSUB 9100:IF T$<>"R" THEN 1470
1460 FOR T=0 TO 7:R=PEEK(Q+T):POKE Q+T,(R/2) + (R AND
1)*128:NEXT T:REM RECHTS
1470 IF T$<>"L" THEN 1490
1480 FOR T=0 TO 7:R=PEEK(Q+T)*2:POKE Q+T,(R AND 255) +
R/256:NEXT T:REM LINKS
1490 S=1:IF T$="O" THEN 1510
1500 S=-1:IF T$<>"U" THEN 1520
1510
FORT=0TO7:P(T)=PEEK((7ANDT+S)+Q):NEXTT:FORT=0TO7:POKEQ+T,P(T)
```

```
:NEXTT:REM HOCH/RUNTER
1520 GOSUB 9200:GOTO 550
1600 REM
1620 REM MODUSWECHSEL:
1630 REM *************
1640 M=ABS(M-1):POKE MO%,M:GOSUB 3000:GOTO 700
1700 REM
1710 REM ZEICHEN DEFINIEREN:
1720 REM ********************
1730 GOSUB 8999:PRINT C6$;C$(5);"GEBEN SIE AN:";C7$
1740 PRINT C$(7);"ZUORDNUNG DES
ZEICHENS: ";C$(6);CHR$(127);C$(7)
1750 GOSUB 2500:IF F=1 THEN F=0:GOTO 540:REM
ADRESSENERR.+FEHLERABFR.
1760 FOR X=0 TO 7:POKE T+X,PEEK(Q+X):NEXT X:REM SPEICHERN
1770 FOR X=1 TO 1500:NEXT X:GOSUB 9200:GOTO550:REM
WARTEN+LOESCHEN
1800 REM
1810 REM ZEICHEN HOLEN:
1820 REM **************
1830 GOSUB 8999:PRINT
C$(1);"GEBEN SIE DAS ZU BEARBEITENDE":PRINT "ZEICHEN EIN:"
1840 GOSUB 2500:IF F=1 THEN F=0:GOTO 540:REM
ADRESSENERR.+FEHLERABFR.
1850 FOR Y=0 TO 7: POKE Q+Y,PEEK(T+Y):NEXT Y:GOSUB 9200:GOTO
550:REM LADEN
1900 REM
1910 REM ZEICHEN INVERTIEREN:
1920 REM ********************
1930 FOR B=0 TO 7:POKE Q+B,255-PEEK(Q+B):NEXT B:GOTO 550
2000 REM
2010 REM ZEICHENSATZ SPEICHERN:
2020 REM **********************
2030 GOSUB 8999:PRINT
C6$;C$(1);"ZEICHENSATZABSPEICHERUNG";C7$
2040 GOSUB 2300:IF F=1 THEN F=0:GOTO 2000:REM
EINGABE/FEHLERABFR.
2050 IF F=2 THEN F=0:GOTO 2150:REM FEHLER
2060 SYS SP%:GOTO 2200:REM SPEICHERN
2100 REM
2110 REM ZEICHENSATZ LADEN:
```

```
2120 REM ******************
2130 GOSUB 8999:PRINT C6$;C$(1);"ZEICHENSATZ LADEN:";C7$:
2140 GOSUB 2300:IF F=1 THEN F=0:GOTO 2100
2150 IF F=2 THEN F=0:GOTO 540
2160 SYS LA%
2200 REM FEHLERABFRAGE (NUR FUER DISK!):
2210 OPEN 1,8,15:INPUT#1,DS,DS$,DT,DB:CLOSE1
2220 IF DS<20 THEN 500:REM OK
2230 PRINT:T$=STR$(DS)+","+DS$+","+STR$(DT)+","+STR$(DB)
2240 GOSUB 9310:PRINT T$:FOR S=1 TO 2000:NEXT S:GOTO 500:REM
BLINKEN
2300 REM
2310 REM NAMENEINGABE:
2320 REM *************
2330 A$="":PRINT:PRINT "FILENAME"+C$(6);:INPUT A$:T=LEN(A$)
2340 S=VAL(RIGHT$(A$,1))
2350 IF S<>0 AND LEFT$(RIGHT$(A$,2),1)=";" THEN T=T-2:POKE
GA,S:REM GERAETEADR.
2360 IF T=0 THEN F=2:RETURN:REM KEIN NAME
2370 IF T>17 THEN 2390
2375 REM NAMEN AN MASCHINENROUTINEN (704=$02C0):
2380 POKE NA%,T:FOR S=1 TO T:POKE
NA%+S,ASC(MID$(A$,S,1)):NEXT S:RETURN
2390 PRINT CHR$(145);:GOSUB 970:T$=C6$+"LAENGE!"+C7$:GOSUB
915:REM FEHLERMELDUNG
2400 PRINT C$(6):F=1:RETURN
2500 REM
2510 REM ZEICHENADRESSE ERRECHNEN:
2520 REM *************************
2530 PRINT "ZEICHENSATZ(1-4): ";N:A$=""
2540 PRINT "TASTE(F3) ODER ASCII(F5)?":GOSUB 9100
2550 ON ABS(ASC(T$)-132) GOTO 2570,2570,2590,2590
2560 GOTO 9300
2570 INPUT "TASTE";A$:A$=LEFT$(A$,1):IF A$="" THEN 9300:REM
TASTENEINGABE
2580 T=ASC(A$):GOTO 2620
2590 INPUT "ASCII";A$:IF A$="" THEN 9300:REM ASCII-EINGABE
2600 T=VAL(A$)/255:T=INT((T-INT(T))*255):A$=CHR$(T)
2610 IF T<32 OR (T<160 AND T>127) THEN 9300
2620 IF T>191 THEN T=T-96:REM ASCII-UMWANDLUNG
2630 PRINT CHR$(145);"TASTE:";A$;" ASCII:";T;:IF B>8 THEN
```

```
V$=A$
2640 REM UMRECHNUNG (T IST DER NORMALE ASCII-WERT DES
ZEICHENS):
2650 IF T<64  THEN S=256:GOTO 2680
2660 IF T<128 THEN S=256*((32 AND T)/16)
2670 IF T>159 THEN S=256*(INT(T/32)-2)
2680 T=(N+47)*1024+S+(31 AND T)*8:RETURN
2800 REM
2810 REM ZEICHENSATZWECHSEL:
2820 REM ********************
2830 A=36:B=5:GOSUB 9000:PRINT C$(2);C6$;N;C7$;C$(6):REM
NUMMER INVERTIEREN
2840 GOSUB 9100:S=VAL(T$):IF S=0 OR S>4 THEN S=N:REM
S=NEUER/N=ALTER ZEICHENSATZ
2850 N=S:GOSUB 3000:GOTO 700
2900 REM
2910 REM ZEICHEN HOLEN (IN MODUS 1):
2920 REM ****************************
2930 IF C<32 OR (C>127 AND C<160) THEN 700
2940 T=C:R=N:V$=A$:GOSUB 3000:GOTO 1850
3000 REM
3010 REM STATUSFELD ERSTELLEN:
3020 REM **********************
3030 A=20:B=5:GOSUB 9000:PRINT
C$(7);"MODUS:";M;"/ SATZ:";N:A$=V$:T=ASC(A$)
3040 PRINT TAB(20);CHR$(17);CHR$(17);C$(6);
3050 GOSUB 2620:PRINT CHR$(157);" ":PRINT
3060 PRINT TAB(20);C$(7);"FARBZUTEILUNG:"
3070 PRINT TAB(20);C$(2);:FOR S=1 TO 14:PRINTCHR$(163);:NEXT
S:PRINT C$(6)
3080 FOR S=0 TO 1:PRINT TAB(20);"GRUNDFARBE";S;":";F(S):NEXT
S:RETURN
3100 REM
3110 REM PLOT:
3120 REM *****
3130 S=((S-12) AND 2)/2:REM PLOTFARBE FESTSTELLEN
3140 T=2^(7-X):POKE Q+Y,PEEK(Q+Y) AND (255-T) OR S*T:GOTO 700
3200 REM
3210 REM FARBENWAHL:
3220 REM ***********
3230 GOSUB 8999:PRINT
```

```
TAB(4);C$(1)"F"C$(2)"A"C$(3)"R"C$(4)"B"C$(5)"E"C$(6)"N";
3240 PRINT C$(7)"W"C$(4)"A"C$(6)"H"C$(2)"L"C$(7)":"
3250 PRINT TAB(4);C$(1);CHR$(172);:FOR S=1 TO 32:PRINT
CHR$(162);:NEXT S:PRINT CHR$(187)
3260 FOR S=1 TO 2:PRINT TAB(4);C6$;CHR$(161);
3270 FOR T=0 TO 15:PRINT C$(T);"  ";:NEXT T:PRINT
C$(1);C7$;CHR$(161):NEXT S
3280 PRINT
TAB(4);C6$;CHR$(161);" 0 1 2 3 4 5 6 7 8 9101112131415";C7$;C
HR$(161)
3290 PRINT:PRINT C$(6);"    FUER GRUNDFARBENNR.(F1/F3): ";
3300 GOSUB 9100:T=ASC(T$)-133:REM FUNKTIONSTASTE
3310 IF T<0 OR T>1 THEN GOSUB 9300:GOTO 540:REM FEHLER
3320 IF T>1 THEN T=T-4
3330 PRINT T:T$="":INPUT "    FARBE ";T$:S=ABS(INT(VAL(T$)))
3340 IF T$="" OR S>15 THEN GOSUB 9300:GOTO540:REM FEHLER
3350 F(T)=S:POKE 53281+T,S:REM FARBE SETZEN
3360 GOTO 540
3400 REM
3410 REM BEFEHLSSATZ:
3420 REM ************
3430 PRINT C0$;C6$;C$(2)"              ";C$(7);
3440 PRINT "BEFEHLSSATZ";C$(2);"               ";C7$;
3450 PRINT C$(4);:FOR S=1 TO 40:PRINT CHR$(184);:NEXT S:PRINT
3460 PRINT
C$(1)" NR.  "C6$"BEFEHL "C7$"-"C$(5)" FUNKTION"C$(4)
3470 FOR S=1 TO 10:PRINT "----";:NEXT S
3480 PRINT C$(1)" (1)  "C6$"(<>↑..)"C7$"-"C$(5)"
CURSORBEWEGUNGEN"
3490 PRINT C$(1)" (2)  "C6$"(<DEL>)"C7$"-"C$(5)" MODUSWECHSEL
"
3500 PRINT C$(1)" (3)  "C6$"(CTRL↑)"C7$"-"C$(5)" ZEICHENSATZW
ECHSEL"
3510 PRINT C$(1)" (4)  "C6$"(F1-F8)"C7$"-"C$(5)" PLOT IN FARB
EN 0-15"
3520 PRINT C$(1)" (5)  "C6$"(F)    "C7$"-"C$(5)" FARBEN 0-15
F. F1/3 DEF."
3530 PRINT C$(1)" (6)  "C6$"(B)    "C7$"-"C$(5)" BEFEHLSSATZ"
3540 PRINT C$(1)" (7)  "C6$"(D)    "C7$"-"C$(5)" ZEICHEN DEFI
NIEREN"
3550 PRINT C$(1)" (8)  "C6$"(H)    "C7$"-"C$(5)" ZEICHEN HOLEN"
```

```
3560 PRINT C$(1)" (9)  "C6$"(I)    "C7$"-"C$(5)" ZEICHEN INVE
RTIEREN"
3570 PRINT C$(1)"(10)  "C6$"(V)    "C7$"-"C$(5)" ZEICHEN VERS
CHIEBEN"
3580 PRINT C$(1)"(11)  "C6$"(L)    "C7$"-"C$(5)" ZEICHEN LOES
CHEN"
3590 PRINT C$(1)"(12)  "C6$"(CTRLG)"C7$"-"C$(5)" GET-ZEICHENS
. LADEN"
3600 PRINT C$(1)"(13)  "C6$"(CTRLS)"C7$"-"C$(5)" SAVE-ZEICHEN
S. SPEICHERN"
3610 PRINT C$(1)"(14)  "C6$"(C)    "C7$"-"C$(5)" DIREKTORY/CA
TALOG"
3620 PRINT C$(1)"(15)  "C6$"(CTRLX)"C7$"-"C$(5)" BEENDEN"
3630 GOSUB 9100:GOTO 540
8000 REM
8100 REM UNTERPROGRAMME:
8200 REM ***************
8300 REM
8400 REM POSITIONIERUNG:
8500 REM ***************
8999 A=0:B=16:REM MELDEFELD
9000 PRINT C1$;:FOR S=2 TO B:PRINT:NEXT S:PRINT
TAB(A);:RETURN
9050 REM
9060 REM TASTENEINGABE:
9070 REM ***************
9100 WAIT 198,255:GET T$:RETURN
9150 REM
9160 REM MELDEFELD LOESCHEN:
9170 REM ********************
9200 A=0:B=16:GOSUB 9000:FOR S=1 TO 5:GOSUB 9210:PRINT:NEXT
S:RETURN
9210 FOR T=1 TO 9:PRINT "     ";:NEXT T:RETURN
9250 REM
9260 REM FEHLERBLINKEN:
9270 REM ***************
9300 T$="UNZULAESSIG!"
9310 PRINT C$(1):FOR S=1 TO 9:PRINT T$:GOSUB 9330:PRINT
CHR$(145);
9320 GOSUB 9210:PRINT CHR$(145):GOSUB 9330:NEXT S:GOSUB
```

```
9200:F=1:RETURN
9330 FOR T=1 TO 75:NEXT T:RETURN:REM WARTESCHLEIFE
9890 REM
9900 REM ************************
9910 REM **                    **
9920 REM **  MASCHINENROUTINEN  **
9930 REM **                    **
9940 REM ************************
9950 REM
9960 REM DATAS WERDEN NACH DEM STARTEN GELOESCHT !!!
9970 REM
10000 FOR I = 1 TO 16 : READ X : NEXT I : REM VORDERE DATAS
UEBERSPRINGEN (FARBEN)
10005 FOR I = 18432 TO 18836
10010 READ X : POKE I,X : S=S+X : NEXT
10020 DATA 120,169, 51,133,  1,169, 48, 32, 26, 72,169,192
10030 DATA  32, 26, 72,169, 55,133,  1,169, 28,141, 24,208
10040 DATA  88, 96,133,  5,169,208,160,  0,132,  2,132,  4
10050 DATA 133,  3,162, 16,177,  2,145,  4,136,208,249,230
10060 DATA   5,230,  3,202,208,242, 96,173,192,  2,162,193
10070 DATA 160,  2, 32,249,253,169,  2,166,186,160,  0, 32
10080 DATA   0,254,169,  0,162,  0,160,192, 76,213,255,173
10090 DATA 192,  2,162,193,160,  2, 32,249,253,169,  2,166
10100 DATA 186,160,  0, 32,  0,254,169, 20,141, 24,208,169
10110 DATA  48,133,  5,169,192, 32, 30, 72,169,  0,133,  2
10120 DATA 169, 48,133,  3,169,  2,162,255,160, 63, 32,216
10130 DATA 255,120,169, 48,162, 51,134,  1, 32, 26, 72,169
10140 DATA  55,133,  1,169, 28,141, 24,208, 88, 96,160,  0
10150 DATA 169, 32, 32,210,255, 32,210,255,152,  9, 48, 32
10160 DATA 210,255,162,  0, 32,207, 72,169, 29, 32,210,255
10170 DATA 232,224,  8,208,243,169,165, 32,210,255,169, 13
10180 DATA  32,210,255,200,192,  8,208,212, 96,174,254,  2
10190 DATA 172,253,  2,152, 72,138, 72,169,  0, 56,106,202
10200 DATA  16,252, 57,248, 50,208,  3,160,  0, 44,160,  6
10210 DATA 162,  6,185,245, 72, 32,210,255,200,202,208,246
10220 DATA 104,170,104,168, 96,146, 31,111, 31,146,157, 18
10230 DATA  28, 32, 31,146,157,173,252,  2,162,  0,201, 20
10240 DATA 240, 35,201,148,240, 31,232,201, 30,240, 26,201
10250 DATA  94,208,  5,172,251,  2,240, 17,232,172,251,  2
10260 DATA 208, 11,160, 28,232,221, 47, 73,240,  3,136,208
10270 DATA 247,232,142,252,  2, 96, 87, 29, 81,157, 65, 17
```

```
10280 DATA  50,145,133,137,134,138,135,139,136,140, 76, 68
10290 DATA  72, 73, 19,  7, 24, 66, 67, 70, 86,169, 36,133
10300 DATA   2,169,  1,162,  2,160,  0, 32,249,253,169,  2
10310 DATA 166,186,160,  0, 32,  0,254,169,  0,162,  0,160
10320 DATA  64,134, 95,132, 96, 32,213,255,165, 95,164, 96
10330 DATA  32, 55,165,173,  0,  3, 72,173,  1,  3, 72,169
10340 DATA  61,141,  0,  3,169,227,141,  1,  3, 32,195,166
10350 DATA 104,141,  1,  3,104,141,  0,  3, 96
10360 IF S <> 46617 THEN PRINT "FEHLER IN DATAS !!" : END
10370 PRINT "OK" :RETURN
```

```
END OF ASSEMBLY!

                    0010          ;
                    0020          ;MASCHINENROUTINEN:
                    0030          ;******************
                    0040          ;
                    0050          ;
                    0060          .OS
                    0070          .BA $4800          ;STARTADRESSE
                    0080          .MC $0800
                    0090          ;
                    0100          ;SPRUNGADRESSEN UND REGISTER:
                    0110          ;****************************
                    0120          ;
                    0130 CHROUT   .DE $FFD2          ;ZEICHENAUSGABE
                    0140 FNPAR    .DE $FDF9          ;FILENAMENPARAMETER SETZEN
                    0150 FPAR     .DE $FE00          ;FILEPARAMETER SETZEN
                    0160 SAVE     .DE $FFD8          ;SPEICHERN AUF DISK/KASSETTE
                    0170 LOAD     .DE $FFD5          ;LADEN VON DISK/KASSETTE
                    0180 TESTCH   .DE $32F8          ;TESTZEICHEN (ANGEZEIGTES)
                    0190 LAENGE   .DE $02C0          ;FILENAMENLAENGE
                    0200 MODUS    .DE $02FB          ;MODUS
                    0210 TASTE    .DE $02FC          ;BEFEHLSTASTENDRUCK
                    0220 YKOORD   .DE $02FD          ;FELD-Y-KOORDINATE
                    0230 XKOORD   .DE $02FE          ;FELD-X-KOORDINATE
                    0240          ;
                    0250          ;ZEICHENSATZ COPIEREN:
                    0260          ;**********************
                    0270          ;
4800- 78            0280 INIT     SEI                ;KEIN INTERRUPT
4801- A9 33         0290          LDA #$33
4803- 85 01         0300          STA *$01           ;ZEICHENSATZ LESBAR MACHEN
4805- A9 30         0310          LDA #$30           ;VON $D000 NACH $3000
4807- 20 1A 48      0320          JSR MOVE           ;COPIEREN
480A- A9 C0         0330          LDA #$C0           ;VON $D000 NACH $C000
480C- 20 1A 48      0340          JSR MOVE           ;COPIEREN
480F- A9 37         0350          LDA #$37
4811- 85 01         0360          STA *$01           ;I/O AUSWAEHLEN
4813- A9 1C         0370          LDA #$1C
4815- 8D 18 D0      0380          STA $D018          ;ZEICHENSATZADRESSE NACH $3000
4818- 58            0390          CLI                ;INTERRUPT ERMOEGLICHEN
4819- 60            0400          RTS
                    0410          ;
                    0420          ;COPIEREN:
                    0430          ;*********
                    0440          ;
481A- 85 05         0450 MOVE     STA *$05           ;ZIELADRESSE HIGHBYTE
481C- A9 D0         0460          LDA #$D0           ;QUELLADRESSE HIGHBYTE
481E- A0 00         0470 M0       LDY #$00
4820- 84 02         0480          STY *$02           ;QUELLADRESSE LOWBYTE
4822- 84 04         0490          STY *$04           ;ZIELADRESSE LOWBYTE
4824- 85 03         0500          STA *$03
4826- A2 10         0510          LDX #$10
4828- B1 02         0520 M1       LDA ($02),Y        ;LADEN
482A- 91 04         0530          STA ($04),Y        ;SPEICHERN
482C- 88            0540          DEY
482D- D0 F9         0550          BNE M1
482F- E6 05         0560          INC *$05
4831- E6 03         0570          INC *$03
4833- CA            0580          DEX
4834- D0 F2         0590          BNE M1             ;NAECHSTE SEITE
4836- 60            0600          RTS
                    0610          ;
                    0620          ;ZEICHENSATZ LADEN:
```

```
                0630        ;******************
                0640        ;
4837- AD C0 02  0650 LADEN  LDA LAENGE        ;NAMENLAENGE
483A- A2 C1     0660        LDX #$C1          ;FILENAMENADRESSE LOW-
483C- A0 02     0670        LDY #$02          ;HIGHBYTE
483E- 20 F9 FD  0680        JSR FNPAR
4841- A9 02     0690        LDA #$02          ;LOGISCHE FILENUMMER
4843- A6 BA     0700        LDX *$BA          ;GERAETEADESSE
4845- A0 00     0710        LDY #$00          ;SECUNDAERADRESSE
4847- 20 00 FE  0720        JSR FPAR
484A- A9 00     0730        LDA #$00          ;LOAD/VERIFY-FLAG
484C- A2 00     0740        LDX #$00          ;STARTADRESSE (LOWBYTE)
484E- A0 C0     0750        LDY #$C0          ;STARTADRESSE (HIGHBYTE)
4850- 4C D5 FF  0760        JMP LOAD
                0770        ;
                0780        ;ZEICHENSATZ SPEICHERN:
                0790        ;**********************
                0800        ;
4853- AD C0 02  0810 SPEICH LDA LAENGE        ;FILENAMENLAENGE
4856- A2 C1     0820        LDX #$C1          ;FILENAMENADRESSE (LOW)
4858- A0 02     0830        LDY #$02          ;FILENAMENADRESSE (HIGH)
485A- 20 F9 FD  0840        JSR FNPAR
485D- A9 02     0850        LDA #$02          ;LOGISCHE FILENUMMER
485F- A6 BA     0860        LDX *$BA          ;GERAETEADRESSE
4861- A0 00     0870        LDY #$00          ;SECUNDAERADRESSE
4863- 20 00 FE  0880        JSR FPAR
4866- A9 14     0890        LDA #$14
4868- 8D 18 D0  0900        STA $D018         ;ORIGINALE ZEICHENSATZLAGE
486B- A9 30     0910        LDA #$30          ;ZIELADRESSE (HIGHBYTE)
486D- 85 05     0920        STA *$05
486F- A9 C0     0930        LDA #$C0          ;QUELLADRESSE (HIGHBYTE)
4871- 20 1E 48  0940        JSR M0            ;VON $C000 NACH $3000
4874- A9 00     0950        LDA #$00
4876- 85 02     0960        STA *$02          ;STARTADRESSE (LOWBYTE)
4878- A9 30     0970        LDA #$30
487A- 85 03     0980        STA *$03          ;STARTADRESSE (HIGHBYTE)
487C- A9 02     0990        LDA #$02          ;NULLSEITENADRESSE DER STARTADRESS
487E- A2 FF     1000        LDX #$FF          ;ENDADRESSE (LOWBYTE)
4880- A0 3F     1010        LDY #$3F          ;ENDADRESSE (HIGHBYTE)
4882- 20 D8 FF  1020        JSR SAVE          ;SPEICHERE ZS AB $3000
4885- 78        1030        SEI               ;INTERRUPT BLOCKIEREN
4886- A9 30     1040        LDA #$30          ;ZIELADRESSE (HIGHBYTE)
4888- A2 33     1050        LDX #$33
488A- 86 01     1060        STX *$01          ;ZEICHENSATZ LESBAR MACHEN
488C- 20 1A 48  1070        JSR MOVE          ;VON $D000 NACH $3000
488F- A9 37     1080        LDA #$37
4891- 85 01     1090        STA *$01          ;I/O AUSWAEHLEN
4893- A9 1C     1100        LDA #$1C
4895- 8D 18 D0  1110        STA $D018         ;ZEICHENSATZADRESSE NACH $3000
4898- 58        1120        CLI               ;INTERRUPT ERMOEGLICHEN
4899- 60        1130        RTS               ;ZURUECK NACH BASIC
                1140        ;
                1150        ;ARBEITSFELD HERSTELLEN:
                1160        ;***********************
                1170        ;
489A- A0 00     1180 NETZ   LDY #$00          ;ZEILENZAEHLER = 0
489C- A9 20     1190 N0     LDA #$20          ;" "
489E- 20 D2 FF  1200        JSR CHROUT
48A1- 20 D2 FF  1210        JSR CHROUT        ;2 LEERZEICHEN
48A4- 98        1220        TYA
48A5- 09 30     1230        ORA #$30          ;ZEILENZAEHLER IN ZIFFER WANDELN
48A7- 20 D2 FF  1240        JSR CHROUT        ;AUSGEBEN
48AA- A2 00     1250        LDX #$00
48AC- 20 CF 48  1260 N1     JSR PUNKT         ;EINEN PUNKT ZEICHNEN
48AF- A9 1D     1270        LDA #$1D          ;CURSOR RECHTS
48B1- 20 D2 FF  1280        JSR CHROUT
```

```
48B4- E8        1290         INX              ;NAECHSTER PUNKT
48B5- E0 08     1300         CPX #$08         ;ACHT PUNKTE PRO ZEILE
48B7- D0 F3     1310         BNE N1
48B9- A9 A5     1320         LDA #$A5         ;STRICH (CHR$(165))
48BB- 20 D2 FF  1330         JSR CHROUT
48BE- A9 0D     1340         LDA #$0D         ;CARRIGE RETURN
48C0- 20 D2 FF  1350         JSR CHROUT
48C3- C8        1360         INY              ;NAECHSTE ZEILE
48C4- C0 08     1370         CPY #$08         ;8 ZEILEN
48C6- D0 D4     1380         BNE N0
48C8- 60        1390         RTS
                1400         ;
                1410         ;EINE KOORDINATE ZEICHNEN:
                1420         ;*************************
                1430         ;
48C9- AE FE 02  1440 PUNKT2  LDX XKOORD       ;ANSTEUERUNG DURCH BASIC
48CC- AC FD 02  1450         LDY YKOORD       ;X/Y-KOORDINATE
48CF- 98        1460 PUNKT   TYA
48D0- 48        1470         PHA
48D1- 8A        1480         TXA
48D2- 48        1490         PHA              ;KOORDINATEN RETTEN
48D3- A9 00     1500         LDA #$00
48D5- 38        1510         SEC              ;EIN BIT SETZEN
48D6- 6A        1520 P0      ROR A            ;RICHTIGES BIT HERAUSSUCHEN
48D7- CA        1530         DEX
48D8- 10 FC     1540         BPL P0
48DA- 39 F8 32  1550         AND TESTCH,Y     ;ANDERE BITS DES TESTZEICHENS LOES
48DD- D0 03     1560         BNE P1           ;BIT (=PUNKT) GESETZT?
48DF- A0 00     1570         LDY #$00         ;NEIN
48E1- 2C        1580         .BY $2C          ;BIT-BEFEHL
48E2- A0 06     1590 P1      LDY #$06         ;BIT GESETZT!
48E4- A2 06     1600         LDX #$06
48E6- B9 F5 48  1610 P2      LDA PKTTAB,Y     ;ZEICHEN AUS PUNKTDARSTELLUNGSTABE
48E9- 20 D2 FF  1620         JSR CHROUT
48EC- C8        1630         INY
48ED- CA        1640         DEX
48EE- D0 F6     1650         BNE P2           ;NAECHSTES ZEICHEN
48F0- 68        1660         PLA
48F1- AA        1670         TAX
48F2- 68        1680         PLA
48F3- A8        1690         TAY              ;KOORDINATEN WIEDERHOLEN
48F4- 60        1700         RTS
                1710         ;
                1720         ;ZEICHEN FUER EINE KOORDINATE:
                1730         ;*****************************
                1740         ;
48F5- 92 1F 6F  1750 PKTTAB  .BY 146 031 111 031 146 157
48F8- 1F 92 9D
48FB- 12 1C 20  1760         .BY 018 028 032 031 146 157
48FE- 1F 92 9D
                1770         ;
                1780         ;TASTE ALS BEFEHL IDENTIFIZIEREN:
                1790         ;********************************
                1800         ;
4901- AD FC 02  1810 BEFIDE  LDA TASTE        ;TASTENCODE
4904- A2 00     1820         LDX #$00         ;BEFEHLSCODE
4906- C9 14     1830         CMP #$14         ;DEL (=MODUSWECHSEL)?
4908- F0 23     1840         BEQ B2           ;JA
490A- C9 94     1850         CMP #$94         ;INSERT (WIE DEL)?
490C- F0 1F     1860         BEQ B2           ;JA
490E- E8        1870         INX              ;BEFEHLSCODE + 1
490F- C9 1E     1880         CMP #$1E         ;CTRL. "PFEIL HOCH" ?
4911- F0 1A     1890         BEQ B2           ;JA
4913- C9 5E     1900         CMP #$5E         ;"PFEIL HOCH" ?
4915- D0 05     1910         BNE B0           ;NEIN
4917- AC FB 02  1920         LDY MODUS        ;NUR IN MODUS 0
```

```
491A- F0 11     1930          BEQ B2
491C- E8        1940 B0       INX                ;BEFEHLSCODE + 1
491D- AC FB 02  1950          LDY MODUS          ;MODUS 1?
4920- D0 0B     1960          BNE B2             ;JA, DANN KEINE WEITEREN BEFEHLE
4922- A0 1C     1970          LDY #$1C           ;27-1 BEFEHLE (ZAEHLER)
4924- E8        1980 B1       INX                ;BEFEHLSCODE ERHOEHEN
4925- DD 2F 49  1990          CMP BEFTAB-3,X     ;TASTE MIT TABELLE VERGLEICHEN
4928- F0 03     2000          BEQ B2             ;GEFUNDEN
492A- 88        2010          DEY                ;NAECHSTER BEFEHL
492B- D0 F7     2020          BNE B1             ;NOCH NICHT FERTIG
492D- E8        2030 B2       INX
492E- 8E FC 02  2040          STX TASTE          ;BEFEHLSCODE ALS RUECKMELDUNG
4931- 60        2050          RTS                ;ZURUECK ZU BASIC
                2060          ;
                2070          ;BEFEHLSTASTENTABELLE:
                2080          ;*********************
                2090          ;
                2100          ;    W/CRSR-RE/Q/CRSR-LI/A/CRSR-UN
4932- 57 1D 51  2110 BEFTAB   .BY 087 029  081  157  065  017
4935- 9D 41 11
                2120          ;    2/CRSR-OB/ F1/ F2/ F3/ F4
4938- 32 91 85  2130          .BY 050 145   133 137 134 138
493B- 89 86 8A
                2140          ;    F5/ F6/ F7/ F8/ L / D
493E- 87 8B 88  2150          .BY 135 139 136 140 076 068
4941- 8C 4C 44
                2160          ;    H / I/<HOME>/CTRL.G/CTRL.X/ B
4944- 48 49 13  2170          .BY 072 073  019   007     024  066
4947- 07 18 42
                2180          ;    C / F / V
494A- 43 46 56  2190          .BY 067 070 086
                2200          ;
                2210          ;DISKCATALOG:
                2220          ;************
                2230          ;
494D- A9 24     2240 CATALG   LDA #$24           ;"$" ALS FILENAME
494F- 85 02     2250          STA *$02
4951- A9 01     2260          LDA #$01
4953- A2 02     2270          LDX #$02
4955- A0 00     2280          LDY #$00           ;S.O.
4957- 20 F9 FD  2290          JSR FNPAR
495A- A9 02     2300          LDA #$02
495C- A6 BA     2310          LDX *$BA           ;GERAETEADRESSE
495E- A0 00     2320          LDY #$00
4960- 20 00 FE  2330          JSR FPAR
4963- A9 00     2340          LDA #$00           ;LOAD/VERIFY-FLAG
4965- A2 00     2350          LDX #$00
4967- A0 40     2360          LDY #$40           ;STARTADRESSE
4969- 86 5F     2370          STX *$5F
496B- 84 60     2380          STY *$60
496D- 20 D5 FF  2390          JSR LOAD           ;LADE CATALOG WIE BASICPROGRAMM
4970- A5 5F     2400          LDA *$5F           ;SIMULIERE:
4972- A4 60     2410          LDY *$60           ;BASIC-PROGRAMMSTARTADRESSE
4974- 20 37 A5  2420          JSR $A537          ;BASICZEILEN BINDEN
4977- AD 00 03  2430          LDA $0300
497A- 48        2440          PHA
497B- AD 01 03  2450          LDA $0301          ;ORIGINALEN WARMSTARTVEKTOR
497E- 48        2460          PHA                ; RETTEN
497F- A9 3D     2470          LDA #$3D
4981- 8D 00 03  2480          STA $0300
4984- A9 E3     2490          LDA #$E3
4986- 8D 01 03  2500          STA $0301          ;UND AUF RTS SETZEN
4989- 20 C3 A6  2510          JSR $A6C3          ;LISTBEFEHL AUSFUEHREN
498C- 68        2520          PLA
498D- 8D 01 03  2530          STA $0301
4990- 68        2540          PLA
```

```
4991- 8D 00 03   2550          STA $0300          ;ALTEN VEKTOR WIEDERHOLEN
4994- 60         2560          RTS                ;ZURUECK ZU BASIC
                 2570          .EN
END OF ASSEMBLY!


 ---  LABEL FILE:  --

B0 =491C                 B1 =4924
B2 =492D                 BEFIDE =4901
BEFTAB =4932             CATALG =494D
CHROUT =FFD2             FNPAR =FDF9
FPAR =FE00               INIT =4800
LADEN =4837              LAENGE =02C0
LOAD =FFD5               M0 =481E
M1 =4828                 MODUS =02FB
MOVE =481A               N0 =489C
N1 =48AC                 NETZ =489A
P0 =48D6                 P1 =48E2
P2 =48E6                 PKTTAB =48F5
PUNKT =48CF              PUNKT2 =48C9
SAVE =FFD8               SPEICH =4853
TASTE =02FC              TESTCH =32F8
XKOORD =02FE             YKOORD =02FD

//0000,4995,0995
Ü
```

Der Zeichenformer funktioniert analog zu dem Spriteeditor von Paragraph 4.3. Auch hier existieren ein Secundäroperations-, ein Farbzuteilungs-, ein Editor- (hier 8x8) und ein Originalfeld. Hinzu kommt noch die sogenannte Modus- und Satzangabe. Ihr Programm kennt zwei Modi:

- 0: Eingabemodus:
  In Modus 0 können Sie alle Befehle benutzen und Ihr eigenes Zeichen herstellen.

- 1: Abrufmodus:
  Modus 1 ermöglicht Ihnen einen bequemen Abruf bereits hergestellter Zeichen per Tastendruck aus dem Zeichenbuffer (Nach dem Starten Ihres Programmes enthält der zuständige Zeichenbuffer den kompletten originalen Zeichensatz).

Zur Satzangabe sind die 4 Zeichensätze (normal - Groß/Graphik // invers - Groß/Graphik // normal - Klein/Groß // invers - Klein/Groß) mit Ihren jeweils 128 Zeichen von 1-4 durchnumeriert. Der aktuelle Satz, der gerade behandelt wird, steht dann in der Satzangabe.

Grundsätzlich können alle Befehle des Spriteeditor (bis auf G) auch im Zeichenformer angewandt werden und sind dort nachzulesen. Zusätzlich stehen Ihnen hier noch folgende Funktionen zur Verfügung:

- <del> -
Moduswechsel (Modus 0-1)

- <ctrl>↑ -
Wollen Sie die Nummer des Zeichensatzes wechseln, so drücken Sie <ctrl>↑ und die Nr. (1-4) des Satzes ein (in Modus 0 genügt auch ↑).

- D/H -
Mit D können Sie Ihr Zeichen einem ASCII-Zeichen, bzw. einer Taste zuordnen und damit in Ihren Zeichensatz übernehmen.
Mit H holen Sie sich ein bereits existierendes Zeichen aus dem Zeichenbuffer in ihr Feld (analog zu Modus 1).

Bei beiden wird der aktuelle Zeichensatz (s.o.)(1-4) angesprochen. Nun entscheiden Sie, ob Sie das Zeichen als ASCII-Wert (f5/f6) oder Tastendruck (f1/f3) angeben wollen. Jetzt erfolgt die Eingabe der Taste oder des ASCII-Wertes mit <return>!

- <ctrl>S / <ctrl>G -
Abspeichern oder Laden eines ganzen Zeichensatzes. Näheres siehe Spriteeditor.

Haben Sie sich mit diesem komfortablen Zeicheneditor Ihren eigenen, persönlichen Zeichensatz hergestellt und auf Diskette gespeichert, dann können Sie Ihn nach dem Beenden des Editorprogramms ganz einfach mit LOAD "name",8,1 nach $3000 (12288) einladen (um ihn woanders hin zu laden, brauchen Sie einen Monitor oder ein Maschinensprache - Ladeprogramm). Als Nächstes müssen Sie mit dem folgenden einfachen Befehl den originalen Zeichensatzvektor verschieben, so daß der VIC sich die benötigten Definitionen von Stund an aus diesem Bereich holt:

```
POKE 53248+24, PEEK(53248) AND 241 OR 12
```

In diesem Befehl werden die drei Bits 1-3 des 24. VIC-Registers, die die Lage bzw. die Adressbits 11-13 des Zeichengenerators bestimmen, zunächst gelöscht (241 = %1111 0001) und dann die obersten beiden der drei Bits gesetzt (12 = %0000 1100). Schlagartig erscheint Ihre neue Zeichencreation auf dem Bildschirm. So einfach ist das.

4.5 Eingabe/Ausgabe von Graphik und Zeichensatz

Mitunter dauert es recht lange, bis wir mühselig ein Graphikbild ersonnen und auf den Bildschirm gebracht haben. Sollten wir dies jedesmal machen, wenn wir uns das Bild betrachten wollen, so ginge uns recht bald die Lust verloren. Doch es gibt einige Möglichkeiten, um diesen Prozess abzukürzen. Zum einen können wir unser Bild auf Diskette oder Kassette abspeichern und bei Bedarf wieder in den Speicher holen. Zum anderen, was das Anschauen später noch weiter verkürzt, sind wir in der Lage - sofern Sie einen Drucker besitzen, der gleichfalls Graphik ausgeben kann (z.B. durch Einzelnadelansteuerung) - sogenannte Hardcopies anzufertigen, also Bilder auf dem Blatt Papier. Die notwendigen Techniken für diese Unterfangen werden Ihnen hier vorgestellt. Es ist klar, daß wir keine Hardcopyroutinen für alle möglichen Drucker angeben können, da leider fast jeder Drucker seine eigene Art und Weise der Graphikausgabe besitzt. Deshalb sollten Sie einmal mit dem hier Gesagten versuchen, sich selbst eine eigene Routine für Ihren Drucker zu schreiben. Notfalls schauen Sie sich einmal in den entsprechenden Fachzeitschriften um, die ab und zu einmal verschiedene Artikel zu diesem Thema bringen. Ist Ihnen dies ebenfalls zu umständlich, so bleibt Ihnen nichts anderes übrig, als auf eine Graphikerweiterung zurückzugreifen, die Ihren Drucker betreiben kann.

4.5.1. Abspeichern/Laden

Das Problem bei der Ein- und Ausgabe von Graphiken oder eines Zeichensatzes auf Diskette oder Kassette ist das Abspeichern, da hierbei dem Computer Anfangs- und Endadresse des zu speichernden Bereiches angegeben werden muß und kein entsprechender Befehl hierzu existiert. Bei normalen Basic-programmen sind dem Rechner diese Dinge bekannt, Maschinen-programme oder sonstige Daten jedoch müssen auf etwas kompliziertere Art und Weise auf das Speichermedium übertragen werden.
Grundsätzlich kann dies durch die folgende Routine geschehen. Sie ist wieder so gehalten, daß Sie sich reibungslos in Ihre gesammelten Graphikroutinen einfügt:

```
10 REM ****************************
20 REM **                          **
30 REM **  GRAPHIKABSPEICHERUNG    **
40 REM **                          **
50 REM ****************************
60 REM
70 FI$="GRAPHIK" : GA = 8 : REM FILENAME UND GERAETEADRESSE
80 BE = 8192 : EN = 16192 : REM START- UND ENDADRESSE (HIER
GRAPHIK BEI $2000)
90 GOSUB 11200 : END : REM SPEICHERN
11200 REM
11210 REM   ********************
11220 REM   **  ABSPEICHERN   **
11230 REM   ********************
11240 REM
11250 SYS 57812 FI$,GA : REM DISKPARAMETERUEBERNAME (AN
$E1D4)
11260 X = EN/256
11270 POKE 175, INT(X) : REM HIGH-BYTE ENDADRESSE ($AF)
11280 POKE 174, (X-INT(X))*256 : REM LOW-BYTE ENDADRESSE
($AE)
11290 X = BE/256
11300 POKE 194, INT(X) : REM HIGH-BYTE STARTADRESSE ($C2)
11310 POKE 193, (X-INT(X))*256 : REM LOW-BYTE STARTADRESSE
($C1)
11320 SYS 62954 : REM SAVE-ROUTINE ($F5EA)
11330 RETURN
```

In diesem Programm werden in den Zeilen 70 und 80 der eigentlichen Speicherroutine die notwendigen Informationen zur Erzeugung eines Files übergeben. Zeile 11250 ruft dann einen Teil des normalen Basic SAVE-Befehls auf, der Filenamen und Geräteadresse (für Disk: SA=8; für Kassette: SA=1) übernimmt. Ihnen mag dieser eine Befehl etwas ungewohnt erscheinen, diese Form ist aber durchaus korrekt. Nachdem nämlich mit dem SYS 57812 die besagte Routine aufgerufen wurde, werden, wie beim SAVE-Befehl, die beiden Parameter verlangt. Aus diesem Grunde gibt es keinen SYNTAX ERROR.
Die folgenden Zeilen übergeben der in Zeile 11320 aufgerufenen eigentlichen SAVE-Routine in bestimmten Speicherstellen die Start- und die Endadresse des zu speichernden Bereiches.
Wir können dieses Programm für alle fraglichen Funktionen verwenden, die wir im Auge hatten. Sowohl Zeichensätze, wie Sprites, Graphik-, Text- und Farbspeicher können so auf Diskette oder Kassette gesichert werden. Ausschlaggebend ist dabei lediglich die Wahl der verschiedenen Parameter. Oben wurde das Beispiel zur Speicherung von Graphik gegeben, die in dem Bereich von $2000-$3F3F (8192-16192) liegt. Für unsere Zeichensätze, die wir in $3000-$3FFF (12288-16383) gespeichert hatten, müßten Sie die Zeile 80 wie folgt ändern:

```
80 BE = 12288 : EN = 16384
```

Wie Sie sehen, müssen Sie zur Endadresse stets 1 hinzuzählen. Möchten Sie vielleicht einmal Ihre gesamte Textseite auf Diskette bringen, die bekanntlich von $0400 bis $07E7 (1024-2023) geht, so schreiben Sie:

```
80 BE = 1024 : EN = 2024
```

Gleiches tippen Sie ein, wenn Sie den Farbteil Ihrer Graphik (sofern er dort liegt) auf Band oder Diskette bringen wollen. Vielleicht aber haben Sie auch eine Spritedefinition in Block 11 und wollen diese abspeichern, z.B. um sie dem Spriteeditor aus Paragraph 4.3 zugänglich zu machen (in diesem Fall muß die Definition in Block 11 liegen) oder einfach, um Sie

später wieder direkt hinein zu laden. Da Block 11 bei $02C0-$02FD (704-766) liegt, muß die entsprechende Passage lauten:

```
80 BE = 704 : EN = 767
```

Nebenbei können Sie damit auch jedes Maschinenprogramm speichern, ohne einen Monitor zu Rate ziehen zu müssen.

Wollen wir die verschiedenen Dinge wieder einladen, so genügt ein LOAD "name",8,1 (bei Floppy-Besitzern) oder für Kassettenrecorder: LOAD "name",1,1 und schon ist die Sache erledigt.

4.5.2. Hardcopy

Die Archillesferse aller kommerziellen Programme ist der Druckerbetrieb. Da es 'zig verschiedene Druckertypen gibt und natürlich jeder seine eigenen Ansteuerungen, ASCII-Codierung oder Steuerzeichen besitzt (besonders, was die Graphik betrifft), gibt es kaum Programme, die mehr als ein oder höchstens zwei Druckertypen bedienen können. Wenn Sie sich z.B. nach einer Graphikbefehlserweiterung oder einem anderen (Graphik-) Programm umschauen, so gebe ich Ihnen den Tip, unbedingt darauf zu achten, daß dieses Programm auch Ihren Drucker bedienen kann, und falls Sie noch keinen Drucker besitzen, besonderes Augenmerk auf die Programme zu legen, die möglichst viele verschiedene Druckertypen ansteuern können. Denn Sie werden sich, auch wenn Sie zur Zeit vielleicht noch keinen Wert darauf legen, garantiert nach einiger Zeit irgendeinen Drucker zulegen. Allein schon die Tatsache, daß etwas größere Programme ohne Druckerlisting völlig unüberschaubar werden, zwingt einen nach einiger Zeit zum Drucker. Falls Sie mit Graphik arbeiten, so ist unbedingt auf die Graphikfähigkeit eines solchen Gerätes zu achten (u.a. auch die Sauberkeit des Druckes, mit der ein Bild aufs Papier gebracht wird, die besonders bei Graphikausdrucken ins Auge fällt). Sie werden es sonst noch einmal bereuen!
Wie oben bereits gesagt, ist es hier unmöglich, alle Druckertypen mit den entsprechenden Routinen vorzustellen.

Aus diesem Grund werden wir uns hier mit dem wohl gängigsten Gerät, dem Seikosha GP 100 VC beschäftigen, auf dem eine graphische Hardcopy aufgrund des 7-Nadelkopfes wohl am kompliziertesten ist. Routinen für andere Drucker können Sie sich dabei gut davon ableiten. Angenommen wird wieder eine Graphikseite bei $2000-$3F3F (8192-16191).

```
11800 REM ****************************
11810 REM **         GRAPHIK-         **
11820 REM **  HARDCOPY - GP 100 VC    **
11830 REM **                          **
11840 REM ****************************
11850 REM
11860 SA = 8192
11870 OPEN1,4 : REM DRUCKERKANAL OEFFNEN
11880 Y=35:MK=255:ZZ=28:SG=6:REM Y-KOORD. / MASKE /
ZEILENZAHL / SPALTENGROESSE
11890 FOR FLAG=1 TO 2
11900  FOR ZE=1 TO ZZ : REM ZZ ZEILEN
11910   PRINT#1,CHR$(8)CHR$(27)CHR$(16)CHR$(0)CHR$(80);:REM
MITTENZENTRIERT+GRAPHIK EIN
11920   XK=0
11930   FOR SP=1 TO 40 : YK=Y : REM SPALTENZAEHLER
11940    FOR X=0 TO SG:GOSUB
12150:ZW(X)=PEEK(AD):YK=YK+1:NEXT X:REM 8*7 PUNKTE LADEN
11950    FOR X=0 TO 7 : REM 8 BYTES ZUM DRUCKER
11960     MS=2↑(7-X):B2=0
11970     FOR Z=0 TO SG:B1=-2*((ZW(Z) AND MS)>0):B2=B2 OR
(B1^Z+(Z+B1=0)):NEXT Z
11980     B2=(B2 AND MK) OR 128:PRINT#1,CHR$(B2);
11990    NEXT X : REM NAECHSTES DRUCKERBYTE
12000    XK=XK+8
12010   NEXT SP : REM NAECHSTE SPALTE
12020   PRINT#1 : REM RETURN
12030   Y=Y+7 : REM Y-KOORD.
12040  NEXT ZE : REM NAECHSTE ZEILE
12050  ZZ=1 : REM NUR LETZTE ZEILE
12060  MK = 16 : REM MASKE (OBERE 4 BITS ABSCHNEIDEN)
12070  SG=4 : REM SPALTENGROESSE
12080 NEXT FLAG : NOCH EINMAL FUER DIE LETZTE ZEILE
12090 CLOSE 1:END
```

```
12100 REM
12110 REM ************************
12120 REM **  PUNKTBERECHNUNG   **
12130 REM ************************
12140 REM
12150 AD = SA + 320 * INT(YK/8) + (YK AND 7) + 8 * INT(XK/8)
: RETURN
```

Auch diese Routine können Sie wieder in Ihren Unterprogrammschatz aufnehmen. Sie brauchen dann nur noch per GOSUB aufgerufen werden. Sie können jedoch auch Ihre Graphik bei SA=8192 ($2000) erstellen, dann dieses Programm hier einladen und laufen lassen. Da dabei die Graphik nicht zerstört wurde, erscheint eine originalgetreue Abbildung des Graphikspeichers auf dem Druckerpapier. Diese Routine funktioniert natürlich auch auf allen kompatiblen Druckern, wie beispielsweise Epson mit Data-Becker - Interface. Doch so eine Routine in Basic dauert sehr lange. Geben Sie also nicht gleich den Mut auf, wenn Sie etwas warten müssen. Sicher kann dieses Programm, wie auch alle anderen "frisiert" werden (wie das zu machen ist, wird im Anhang geschildert), doch hier wurde der Übersichtlichkeit halber geordnet vorgegangen. Wichtig zum Verständis dieser Routine ist die Tatsache, daß stets 7 untereinander liegende Punkt in einem Byte an den Drucker übergeben werden (jedes gesetzte Bit resultiert in einem gesetzten Punkt auf dem Papier), wobei das High-Bit stets gesetzt sein muß (dies erfordert der Drucker). Dadurch taucht eine kleine Schwierigkeit am Graphikende auf: Es bleiben nämlich 4 Reihen übrig, da 200 (die Zahl der Punkte in y-Richtung) nicht durch 7 teilbar ist. Diese werden in einem weiteren Durchlauf mit ein paar veränderten Parameter übersandt. In dem Hauptteil der Routine (Zeilen 11940-11990) werden zunächst 7 untereinander liegende Graphikbytes eingelesen (Z. 11940), die dann Spalte für Spalte an den Drucker gegeben werden (Z. 11950-11990). Die weiteren Einzelheiten zu besprechen, würde den Rahmen sprengen. Nur eine Kleinigkeit sollte hier noch erwähnt werden: Die Operation Z+B1=0, wie Sie in Zeile 11970 durchgeführt wird, ergibt -1, wenn Z+B1=0 ist und 0, wenn dies nicht der Fall ist. Aus 4=6 z.B. resultiert 0, aus 4=4 jedoch -1. Probieren Sie es aus. Dieser Trick kann eine wertvolle Hilfe in vielen

Anwendungen sein, in denen Entscheidungen gefordert werden, die Sie ungern mit IF oder ON...GOTO tätigen (z.B. aus Zeitgründen). In dem Fall hier wird damit die Tatsache umgangen, daß der Commodore 64 aus 0hoch0 1 herausbekommt, was falsche Resultate bringen würde. In dem Graphik-Aid weiter unten wird diese Hardcopy - Routine in Assembler angegeben.

4.6 IRQ-Handhabung

Aus Abschnitt 3.7 kennen Sie bereits die verschiedenen Interrupt- (Unterbrechungs-) möglichkeiten, die bei graphischen Anwendungen interessant sind. Dort wurde Ihnen diese Fähigkeit vorgestellt und alle theoretischen Grundlagen geliefert. Jetzt, da wir auch einigen praktischen Unterbau in Sachen Graphik besitzen, ist es an der Zeit, das wohl schwierigste aller Kapitel zu beginnen: Die Interruptprogrammierung. Schon die Tatsache, das Interrupts nur von Maschinensprache aus zu bedienen sind, macht das ganze für den Uneingeweihten nebulös und undurchsichtig. Aber auch erfahrene Assemblerprogrammierer müssen sich mit den völlig neuen Programmiertechniken vertraut machen. Doch haben Sie keine Angst. Sie können die angegebenen Beispiele ruhig ausprobieren. Sie werden Ihnen besonders hübsche Effekte zeitigen.
Die Interrupttechnik soll Ihnen anhand des Rasterzeilen-IRQs und des Lightpens gezeigt werden. Danach können Sie die erfahrenen Kenntnisse auf Ihre eigenen Programme mit z.B. Sprite-Kollisionen etc. anwenden. Bevor Sie sich jedoch an diesen Abschnitt begeben, sollten Sie zunächst Paragraph 3.7 gut durchgelesen haben.
Erinnern wir uns kurz an die wesentlichen Faktoren, die zur Bedienung von allgemeinen IRQs des VIC wissensnotwendig sind: Interrupts sind kontrollierte Unterbrechungen des Programmablaufs. Es gibt 4 verschiedene Ursachen, durch die der Videocontroller Interrupts auslösen kann: Rasterzeilen, Lightpen, Sprite-Sprite-, und Sprite-Hintergrundzeichen -

Kollisionen. Diese sind durch Setzen der entsprechenden Bits im Interrupt Mask Register (IMR), also VIC-Register 26, auszuwählen. Wird ein Interrupt ausgelöst, so erfolgt die Rückmeldung durch Setzen des korrespondieren Bits in Register 25 des VIC, das sogenannte Interrupt Request Register (IRR); das 7. Bit dieser Speicherstelle wird dabei als Kennzeichen ebenfalls gesetzt. Dieses Register ist nach jedem Interrupt durch Rückschreiben seines Inhalts zu löschen. Bei jedem IRQ wird eine sogenannte Interruptroutine aufgerufen, deren Adresse in den Speicherstellen $0314/$0315 (788/789) steht. Ein IRQ kann in Maschinensprache durch das Setzen des Interruptflags unterbunden werden.

4.6.1. Rasterzeilen-IRQ

Auch hier sollten wir uns zunächst an die notwendigen Dinge erinnern, die für Rasterzeilen - Interrupts nützlich sind:
Für diese Art von Unterbrechung sind jeweils Bits 0 der IRR und IMR zuständig. Aus den Registern 18 und 17/Bit 7 erfahren Sie die aktuelle Bildschirmzeile, die der VIC gerade aufbaut. Das vom VIC verwandte Koordinatensystem unterscheidet sich von dem normalen, uns bekannten (s. # 3.7). Werden diese beiden Register beschrieben, so geben Sie damit an, bei welcher Rasterzeile ein Interrupt stattfinden soll.
Mit diesen Informationen ist es uns jetzt möglich, eine eigene Interruptroutine zu erzeugen und damit direkt in das Geschehen einzugreifen. Dazu sei diese als Assemblerlisting angegeben:

```
100: C800                *=    $C800
110: C800          FARBE1 =    $FB
120: C800          FARBE2 =    $FC
130: C800          OBEN   =    $FD
140: C800          UNTEN  =    $FE
150: C800          IRQVECT =   $0314
160: C800          IRQALT =    $EA31
170: C800          RASTER =    $D012
180: C800          IRR    =    $D019
190: C800          IMR    =    $D01A
200: C800          RAHMEN =    $D020
```

```
210: C800           HGRUND  =    $D021
220:                 ;
230: C800            ;INITIALISIEREN:
240:                 ;***************
250:                 ;
260: C800 78        INIT    SEI          ;INTERRUPT VERHINDERN
270: C801 A9 1F             LDA #< IRQNEU
280: C803 8D 14 03          STA IRQVECT
290: C806 A9 C8             LDA #> IRQNEU
300: C808 8D 15 03          STA IRQVECT+1 ;IRQ-VEKTOR UMLEGEN
310: C80B A5 FD             LDA OBEN
320: C80D 8D 12 D0          STA RASTER  ;1.RASTERZ. FESTLEGEN
330: C810 AD 11 D0          LDA RASTER-1
340: C813 29 7F             AND #$7F
350: C815 8D 11 D0          STA RASTER-1  ;HIGH-BIT LOESCHEN
360: C818 A9 81             LDA #%10000001  ;MASKE
370: C81A 8D 1A D0          STA IMR      ;RASTERZ.-IRQ WAEHLEN
380: C81D 58                CLI          ;IRQ ERMOEGLICHEN
390: C81E 60                RTS
400:                 ;
410: C81F            ;INTERRUPTROUTINE:
420:                 ;******************
430:                 ;
440: C81F AD 19 D0 IRQNEU   LDA IRR      ;INTERRUPTREGISTER
450: C822 8D 19 D0          STA IRR      ;LOESCHEN
460: C825 30 07             BMI IRQRAS   ;RASTERZ.-IRQPRINT
470:                 ;NORMALER IRQ:
480: C827 AD 0D DC          LDA $DC0D    ;CIA 1-IRR LOESCHEN
490: C82A 58                CLI          ;INTERR. ERMOEGLICHEN
500: C82B 4C 31 EA          JMP IRQALT   ;ZUR ALTEN ROUTINE
510: C82E AD 12 D0 IRQRAS   LDA RASTER   ;RASTERPOSITION
520: C831 C5 FE             CMP UNTEN    ;UNTERER WERTPRINT
530: C833 B0 10             BCS ZWEITER  ;JA=>SPRUNG
540: C835 A5 FB             LDA FARBE1
550: C837 8D 20 D0          STA RAHMEN   ;RAHMEN- UND
560: C83A 8D 21 D0          STA HGRUND   ;HINTEGRUNDFARBE
570: C83D A5 FE             LDA UNTEN    ;UNTEREN WERT IN
580: C83F 8D 12 D0          STA RASTER   ;RASTERPOSITION
590: C842 4C BC FE          JMP $FEBC    ;ABSCHLIESSEN
600: C845 A5 FC     ZWEITER LDA FARBE2
610: C847 8D 20 D0          STA RAHMEN   ;RAHMEN- UND
```

```
620: C84A 8D 21 D0          STA HGRUND  ;HINTERGRUNDFARBE
630: C84D A5 FD             LDA OBEN    ;OBEREN WERT IN
640: C84F 8D 12 D0          STA RASTER  ;RASTERPOSITION
650: C852 4C BC FE          JMP $FEBC   ;ABSCHLIESSEN
```

Und hier das entsprechende Basic-Ladeprogramm mit einer kleinen hübschen Anwendung:

```
1000 FOR I = 51200 TO 51284
1010 READ X : POKE I,X : S=S+X : NEXT
1020 DATA 120,169, 31,141, 20,  3,169,200,141, 21,  3,165
1030 DATA 253,141, 18,208,173, 17,208, 41,127,141, 17,208
1040 DATA 169,129,141, 26,208, 88, 96,173, 25,208,141, 25
1050 DATA 208, 48,  7,173, 13,220, 88, 76, 49,234,173, 18
1060 DATA 208,197,254,176, 16,165,251,141, 32,208,141, 33
1070 DATA 208,165,254,141, 18,208, 76,188,254,165,252,141
1080 DATA  32,208,141, 33,208,165,253,141, 18,208, 76,188
1090 DATA 254
1100 IF S <> 11288 THEN PRINT "FEHLER IN DATAS !!" : END
1110 PRINT "OK"
1120 F1 = 7 : F2 = 6 : REM FARBEN 1 UND 2 (STREIFEN IN FARBE
1)
1130 OB = 60 : UN = 150 : REM OBERE UND UNTERE KANTE DES
STREIFENS
1140 POKE 251,F1:POKE 252,F2:POKE 253,OB:POKE 254,UN : REM
UEBERGEBEN
1150 SYS 51200 : REM ROUTINE EINSCHALTEN
1160 REM
1170 FOR X=1 TO 5000 : NEXT X : REM WARTESCHLEIFE
1180 REM
1190 REM BEWEGEN:
1200 REM ********
1210 FOR X=40 TO 240 : POKE 253,X : POKE 254, X+10 : NEXT X
1220 GET A$ : IF A$="" THEN 1190 : REM TASTE?
```

Doch zunächst zu unserer Maschinenroutine:
Unser Programm erzeugt einen quer über den Bildschirm gezogenen gelben Streifen (samt Rand) auf blauem Hintergrund. Dies ist möglich durch kontinuierliches Umschalten der Rahmen- und Hintergrundfarbe, jeweils genau dann, wenn sich der Videocontroller in bestimmten Rasterzeilen befindet.

Bevor aber unsere eigentliche Interruptroutine vom Prozessor aufgerufen wird, müssen wir erst einige Dinge erledigen. Dies geschieht in der Initialisierungsroutine. Als allererstes wird dort der IRQ-Vektor bei $0314/$0315 (788/789; s.o.), der normalerweise auf die originale Interruptroutine im ROM bei $EA31 (59953) zeigt, auf unser neues Unterbrechungs - Programm (hier bei $C81F (51231)) gerichtet. Für diesen Zweck werden Low- und High-Byte der neuen Adresse an diesen Vektor geschrieben (Zeilen 270-300). Doch damit nicht zwischendurch ein Interrupt ausgelöst wird, was ja bekanntlich alle 1/60 Sekunde geschieht, um die Tastatur abzufragen etc., müssen wir vorher durch ein SEI das Interruptflag des Prozessors setzen (Zeile 260).
Nun wird die Rasterzeile bestimmt, bei deren Strahldurchlauf der erste Rasterinterrupt stattfinden soll. Diese, es ist die obere Kante unseres gelben Streifens, steht in der nullseitigen Speicherstelle $FD (253), in die der gewünschte Wert durch das übergeordnete Basic-Programm eingeschrieben wurde. Der Wert wird jetzt in das Rasterzeilenregister (VIC-Register 18) eingeschrieben (Zeilen 310/320). Damit wird der erste Interrupt genau dann ausgelöst, wenn der Video-controller gerade diese Zeile aufbaut. Da wir das High-Bit hier nicht verwenden, wird es durch Zeilen 330-350 gelöscht.
Der nächste wichtige Schritt ist das Einschalten des Raster-IRQs durch Setzen des 0. Bits des IMR, also der Interrupt - Maske. Damit erst kann eine Unterbrechung ausgelöst werden, wenn das besagte Ereignis eintritt.
Am Schluß unserer Inititialisierung wird das I-Flag wieder gelöscht, sodaß die anstehenden Interrupts bedient werden können.

Das Problem bei der Unterbrechungsbehandlung ist, daß jetzt zwei Quellen existieren, durch die ein IRQ ausgelöst werden kann:

- Timer-IRQ
- Raster-IRQ

Ersterer dient zu oben genannten Zwecken und ist absolut notwendig für den normalen Betrieb des Computers. Denjenigen, die ihre Programme in Maschinensprache schreiben und auf die

Funktionen der Tastaturabfrage, Cursorblinken oder TI$-Uhr stellen verzichten können, ist natürlich freigestellt, ob Sie hier den Aufruf der normalen Interruptroutine zulassen wollen oder nicht. Doch sollten sich diejenigen, die damit manipulieren, gut im Betriebssystem Ihres Rechners auskennen. Grundsätzlich aber müssen wir unterscheiden, wodurch ein Interrupt ausgelöst wurde, da unsere Routine von Stund an auch dann aufgerufen wird, wenn der Timer der CIA 1 abgelaufen ist und damit die Tastaturabfrage fällig wäre. Zu diesem Zweck laden wir als allererstes das IRR und schreiben den erhaltenen Wert wieder zurück, um es zu löschen (ansonsten würde gleich nach Beenden unserer Routine wieder ein Interrupt ausgelöst usw.). Wurde ein Raster-IRQ ausgelöst, so sind dort u.a. das 0. und das 7. Bit gesetzt (s. # 3.7). Auf das 7. Bit wird nun geprüft (Zeile 460) und in besagtem Falle weiter zum zweiten Teil verzweigt (nach Zeile 510). Stammt der IRQ jedoch vom Timer, so müssen wir die alte (normale) Interruptroutine aufrufen, die bei $EA31 (59953) beginnt.

Doch hier taucht eine kleine Schwierigkeit auf. Bei jedem Interrupt wird automatisch (hardwaremäßig) die Rücksprungadresse und das Flag-Register der CPU auf den Stack gebracht und - was das hier wichtige ist - das Interruptflag gesetzt, so daß kein weiterer IRQ während unserer Interruptroutine ausgelöst werden kann, was ja auch vernünftig ist. Doch nehmen wir einmal an, wir verzweigen normal nach $EA31 und lassen dort die notwendigen Dinge ausführen. Dann kann es geschehen, daß während dieser Vorgänge ein Raster - Interrupt ausgelöst wird. Da aber das I-Flag gesetzt ist, wird dieser nicht bedient. Dieses Dilemma macht sich dann als kurzes Aufblitzen des Bildschirms bemerkbar, da nicht früh genug auf die andere Hintergrund- und Rahmenfarbe umgeschaltet wurde. Deshalb müssen wir den Rasterinterrupt während des Durchlaufs der Timerroutine zulassen und damit praktisch bei Bedarf die normale Interruptroutine durch einen Interrupt unterbrechen. Das klingt zwar sehr kompliziert und "gefährlich", ist aber durchaus statthaft und nach einigen Überlegungen auch logisch.

Wir könnten also einfach vor dem Aufruf der normalen IRQ-Routine das I-Flag löschen und alles wäre in Ordnung.

Doch wir haben wieder etwas vergessen. Bevor das Flag gelöscht wird müssen wir erst einmal - wie immer - die Ursache für den betreffenden Interrupt (in diesem Fall der Timer-Interrupt) beseitigen. Dies geschieht auch hier durch das Löschen eines IRR. Dieses IRR liegt dabei in dem Registersatz der CIA 1 und besitzt die Adresse $DC0D (56333). Das Löschen funktioniert dort jedoch etwas anders. Es genügt ein einfaches Lesen des Registers um diese Aufgabe zu erfüllen, was in Zeile 480 geschieht. Jetzt haben wir aber alles getan, um einen reibungslosen Ablauf zu gewährleisten und gehen weiter in unserer eigentlichen Interruptroutine (ab Zeile 510).

Da wir ja pro Bildaufbau insgesamt zweimal unsere Farben umschalten müssen, einmal um Farbe 2 auf Farbe 1 zu setzen, und einmal, um wieder von Farbe 1 auf Farbe 2 zurückzuschalten, müssen wir erst nachprüfen, in welcher Phase wir gerade sind. Dies geschieht bei uns durch Lesen der aktuellen Rasterzeile (Zeile 510), um sie mit dem Wert der unteren Kante unseres gelben Streifens zu vergleichen (Z. 520). Ist die aktuelle Zeile größer oder gleich dieses Wertes, so müssen wir auf die zweite Farbe umschalten, springen daher nach Zeile 600. Wir hätten rein theoretisch auch prüfen können, welche Farbe gerade Hintergrundfarbe ist, was ebenfalls seine Vorteile hätte. (Anmerkung: Da es eine ganze Weile dauert, bis das Programm nach dem IRQ an exakt diese Stelle kommt, ist der Rasterstrahl inzwischen einige Zeilen weiter gelaufen. Deshalb werden wir nie genau den Wert als aktuelle Zeile erhalten, bei der der Interrupt stattfand. Bedenken Sie, daß auch schon, bevor unsere Routine aufgerufen wird, einige Befehle im ROM abgearbeitet werden, die auch Zeit kosten. Das ist auch der Grund, warum die scheinbare(!) Rastereinteilung nicht ganz mit derjenigen übereinstimmt, die in Paragraph 3.7 angegeben wurde.)
Der Rest ist relativ einfach: Wir ändern die Rahmen- und Hintergrundfarbe und setzen den nächsten Rasterinterrupt auf den oberen bzw. unteren Zeilenwert. Als Abschluß springen wir in das reguläre Interruptende der normalen Routine, da noch einige Register wiedergeholt und ein RTI (return from Interrupt) ausgeführt werden müssen.

Wie Sie sehen, steht in den vier nullseitigen Speicherstellen die notwendige Information, die die Art des Streifens charakterisiert:

Adresse (Hex-Dez)	Inhalt
$FB - 251	Farbe 1
$FC - 252	Farbe 2
$FD - 253	Obere Kante
$FE - 254	Untere Kante

Durch Ändern der Inhalte dieser Adressen können Sie nun die Farbe des Streifens oder des restlichen Hintergrundes, die Position der oberen und der unteren Kante bestimmen. Dies geschieht am besten durch ein kleines Basicprogramm, das Ihnen oben ebenfalls angegeben wurde. Es lädt die Maschinenroutinen ein, legt die 4 Parameter fest und startet die Initialisierung und damit den Interrupt. Ab Zeile 1200 wird Ihnen ein Beispiel einer schönen Anwendung gegeben. Denken Sie sich doch einmal etwas aus. Vielleicht machen Sie einmal ein Programm, mit dem Sie -durch die Cursortasten steuerbar- jeweils einzelne Zeilen des Bildschirms hervorheben können, oder - falls Sie sich etwas in Maschinensprache auskennen - Sie wechseln statt der Hintergrundfarben immer zwischen Text- und Graphikmodus und erhalten so eine gemischte Anzeige. Haben Sie schon einmal 16 Sprites oder zwei Zeichensätze gleichzeitig auf dem Bildschirm gesehen? Es gibt unzählbar viele Möglichkeiten.

4.6.2. Lightpen

Bitte lesen Sie ruhig weiter, auch wenn Sie keinen Lightpen besitzen, es springt auch etwas für Sie ab!
Was ein Lightpen oder Lichtgriffel ist, wie er funktioniert und in den Rechnerablauf integriert wird, wissen Sie bereits aus dem Abschnitt 3.7.2. Fassen wir hier das wichtigste noch einmal kurz zusammen:
Der Lightpen ist ein Instrument, mit dem der Computer beliebige Positionen des Bildschirms identifizieren kann, auf die der Stift gerade zeigt. Die jeweiligen x- und y-Koordinaten dieses Punktes können durch Lesen der

VIC-Register 19 und 20 festgestellt werden. Das Koordinatenraster entspricht dem der Rasterzeilen (s. # 3.7). Auch der Lightpen kann einen Interrupt auslösen. In den IMR und IRR sind dafür jeweils die 3. Bits zuständig. Sein Eingang am Controlport 1 stimmt mit dem Eingang für den Feuerknopf eines Joysticks überein.
Doch wie ist ein Lightpen anzuwenden, wie wird er programmiert?
Die Lightpenabfrage kann auf zwei verschiedene Arten durchgeführt werden. Die einfachere ist die durch ein kleines Basicprogramm, das lediglich die beiden Register ausliest, die die Bildschirmkoordinaten enthalten. Ein Beispiel wäre etwa das Zeichnen eines Punktes an die Stelle, auf die der Lightpen gerichtet ist. Diese Funktion übernimmt das folgende kleine Demonstrationsprogramm, das nur zusammen mit den Graphikroutinen aus dem Abschnitt 4.2 funktionstüchtig ist!

```
100 REM ****************
110 REM **              **
120 REM **  LIGHTPEN    **
130 REM **              **
140 REM ****************
150 REM
160 V = 53248 : REM VIC BASISADRESSE
170 SA = 8192 : REM GRAPHIKSTARTADRESSE
180 GOSUB 10000:GOSUB 10200:FA=7*16+2:GOSUB 10400 : REM
GRAPHIK INITIALISIEREN
190 XP = PEEK(V+19) : REM LIGHTPEN-X-KOORDINATE
200 YP = PEEK(V+20) : REM LIGHTPEN-Y-KOORDINATE
210 XK = 2 * (XP - 40) : REM ERRECHNE X-KOORD.
220 YK = YP - 40 : REM ERRECHNE Y-KOORD.
230 IF XK>319 OR XK<0 OR YK>199 OR YK<0 THEN 190 : REM AUF
BEREICH TESTEN
240 GOSUB 10700 : REM PUNKT ZEICHNEN
250 GOTO 190
10000 REM INITIALISIERUNGS+PUNKTSETZ-ROUTINEN
10010 REM ......
10020 REM ...       (S. KAPITEL 4.2)
```

In diesem kleinen Programm werden - nach dem Einschalten und Löschen der Graphik - in den Zeilen 190 und 200 lediglich die

x- und y-Koordinaten des Punktes auf dem Bildschirm eingelesen, bei dem der Lightpen zuletzt auflag. Diese Bildschirmkoordinaten werden dann in unsere bekannten Graphikkoordinaten nach der Formel umgerechnet, die in Abschnitt 3.7.2 angegeben wurde (Z. 210/220). Alsdann muß das Ergebnis darauf geprüft werden, ob der Punkt, den der Lightpen anzeigte nicht außerhalb des Bildschimfensters liegt, was negative oder zu große Werte resultieren ließ. Erst dann kann die Punkt-Setz - Routine aufgerufen werden, um an der jeweiligen Stelle eben einen Punkt zu setzen. Sie können dieses Programm beliebig erweitern mit vielen Funktionen, so daß evt. ein ganzes Zeichenprogramm entsteht. Vielleicht legen Sie mit dem Lightpen die Eckpunkte von Linien oder den Mittelpunkt eines Kreises und seinen Radius fest, die auf Tastendruck dann gezeichnet werden, oder wählen per Lightpen bestimmte Menuefunktionen aus, die am Bildrand angezeigt werden.
Doch eine Sache stört dabei doch. Egal ob der Lightpen auf den Bildschirm zeigt oder nicht, an dieser Stelle wird ein Punkt gezeichnet. Hier mag das noch nicht so schlimme Folgen haben, bei anderen Anwendungen wird es sicher stören. Eine andere Möglichkeit der Bedienung ist die per Interrupt. Jedesmal, wenn der Lightpen einen Impuls sendet, also nur dann, wenn er auch tatsächlich auf den Bildschirm zeigt, könnte beispielsweise ein IRQ ausgelöst werden, der entweder einen Punkt zeichnet oder einfach in einer Speicherstelle ein kurzes Signal gibt, das nach einiger Zeit wieder gelöscht wird.
Eine andere Möglichkeit ist beispielsweise die Konstruktion einer Alarmanlage: Der Lightpen wird an die Zimmerlampe montiert, oder einfach auf sie gerichtet. Sobald irgendjemand diese Lampe beim Eintreten einschaltet, sendet der Lightpen einen Impuls an den Computer, der sofort mit einem Heulgetöse beginnt, das sämtliche Nachbarn aus dem Bett wirft (man könnte den Audioausgang vielleicht an eine Stereoanlage anschließen (geht mit einem einfachen Überspielkabel), die dem Ganzen noch etwas mehr "Power" gibt). Eine Variation dieser Alarmanlage wäre es z.B., wenn das Aufgehen der Türe einen Kontakt auslöste, der eine Lampe zum Leuchten brächte, was seinerseits wieder über den Lightpen zu einer Reaktion des Computers führte.

Wir wollen uns statt der Auslösung eines Alarms mit einem Farbwechsel des Rahmens zufrieden geben:

```
100: C800                *=  $C800
110: C800          FARBE  =   $FB
120: C800          IRQVECT=   $0314
130: C800          IRQALT =   $EA31
140: C800          IRR    =   $D019
150: C800          IMR    =   $D01A
160: C800          RAHMEN =   $D020
170:                ;
180:                ;INITIALISIEREN
190:                ;***************
200:                ;
210: C800 78       INIT   SEI          ;INTERRUPT VERHINDERN
220: C801 A9 16           LDA #< IRQNEU
230: C803 8D 14 03        STA IRQVECT
240: C806 A9 C8           LDA #> IRQNEU
250: C808 8D 15 03        STA IRQVECT+1 ;IRQ-VEKTOR UMLEGEN
260: C80B A9 00           LDA #$00
270: C80D 85 FB           STA FARBE   ;FARBE SCHWARZ
280: C80F A9 88           LDA #%10001000 ;MASKE
290: C811 8D 1A D0        STA IMR     ;LIGHTPEN-IRQ WAEHLEN
300: C814 58              CLI         ;IRQ ERMOEGLICHEN
310: C815 60              RTS
320:                ;
330:                ;INTERRUPTROUTINE
340:                ;*****************
350:                ;
360: C816 AD 19 D0 IRQNEU LDA IRR     ;INTERRUPTREGISTER
370: C819 8D 19 D0        STA IRR     ;LOESCHEN
380: C81C 30 07           BMI IRQRAS  ;RASTERZ.-IRQPRINT
390:               ;NORMALER IRQ
400: C81E AD 0D DC        LDA $DC0D   ;CIA 1-IRR LOESCHEN
410: C821 58              CLI         ;INTERRUPT ERMOEGLICHEN
420: C822 4C 31 EA        JMP IRQALT  ;ZUR ALTEN ROUTINE
430: C825 A5 FB    IRQRAS LDA FARBE
440: C827 8D 20 D0        STA RAHMEN  ;RAHMENFARBE
450: C82A E6 FB           INC FARBE   ;FARBE ERHOEHEN
460: C82C 4C BC FE        JMP $FEBC   ;ABSCHLIESSEN
```

Und hier das Ladeprogramm:

```
100 FOR I = 51200 TO 51246
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 120,169, 22,141, 20,  3,169,200,141, 21,  3,169
130 DATA   0,133,251,169,136,141, 26,208, 88, 96,173, 25
140 DATA 208,141, 25,208, 48,  7,173, 13,220, 88, 76, 49
150 DATA 234,165,251,141, 32,208,230,251, 76,188,254
160 IF S <> 5910 THEN PRINT "FEHLER IN DATAS !!" : END
170 PRINT "OK"
```

Sie starten wie immer mit einem

```
SYS 51200
```

Alle im folgenden genannten Zeilennummern beziehen sich auf das Assemblerlisting.
Die Initialisierungsroutine, die in Zeile 220 beginnt, sollte Ihnen inzwischen schon geläufig sein. Auch hier wird erst wieder der IRQ-Vektor umgelegt auf unsere eigene Routine und der entsprechende Interrupt diesmal durch Setzen des 3. Bits des IMR (Interrupt Mask Register) eingeschaltet. Nichts Besonderes bietet gleichfalls der Einstieg in unsere Interruptroutine (ab Zeile 360). Wieder wird auf die Art des Interrupts geprüft und entsprechend verzweigt. Dann aber kommt der Wechsel der Rahmenfarbe. Der Vorgang an sich ist leicht zu verstehen, wichtig hierbei ist allerdings, daß zum erstenmal eine reguläre Speicherstelle durch die Interruptroutine verändert wird. Dies bringt besondere Effekte, ist aber auch besonders gefährlich, wenn man nicht genau weiß, was mit dieser Stelle wann passiert, denn der Interrupt kann ja grundsätzlich zu jeder Zeit und in jeder Routine ausgelöst werden. Sie können aber von außen, also beispielsweise von Basic aus, den momentanen Zustand der IRQ-Routine abfragen oder andere Steuerfunktionen übernehmen. Dies wird z.B. vom Betriebssystem genutzt, indem die interne Unterbrechungsroutine jedesmal die TI$-Uhr verstellt, die dann von Basic aus abgefragt werden kann. Auch hier natürlich stehen Ihnen viele Möglichkeiten offen.
Bitte erinnern Sie sich, daß sie dieses Bildschirmblinken auch durch Betätigen des Feuerknopfes Ihres in Port 1 gesteckten Joysticks erzeugen können.

Sie sehen, der Interrupt ist ein schönes "Spielzeug", das enorme Welten entstehen läßt, wenn man es richtig zu nutzen weiß. Beschäftigen Sie sich ruhig ein wenig damit, es lohnt sich!

4.7 Ein kleines Graphik-Paket

Wir haben eine ganze Menge über Graphik und Ihre Programmierung gehört und gelesen. Viele Routinen sollten uns das Leben erleichtern und ein wenig Einblick in die phantastische Welt der Bilder vermitteln. Doch beim Ausprobieren all dieser Utilities und Basic - Programme, die wir gespannt eingetippt hatten, wurden wir zwar nicht durch das Ergebnis enttäuscht, dafür aber umso mehr durch den Weg dorthin. Bärte und graue Haare waren keine Seltenheit: Unsere Basicprogramme waren so langsam, auch wenn wir die gut gemeinten Tips im Anhang beherzigten, daß da kaum was von den Wogen der graphischen Dramatik hinüberschwappte. Kurz: Wir wissen zwar, wie es geht, doch die richtige Befriedigung machte sich kaum bemerkbar. Dem soll in diesem Abschnitt abgeholfen werden. Hier wird erst einmal anständig ausgepackt. Ein relativ umfangreiches Graphikpaket in Maschinensprache macht Ihre Graphik schnell und interessant.
Wenn Sie sich die nächsten Seiten betrachten, sollten Sie nicht kapitulieren. Es ist zwar viel Arbeit, die ganzen Maschinenroutinen abzuschreiben und auf Fehler zu kontrollieren, doch es lohnt sich! Wer wirklich ernsthaft mit Graphik arbeiten will, der sollte vor diesem Zeit- und Mußeaufwand nicht zurückschrecken, der braucht einfach das entsprechende Handwerkszeug.
Sie haben die Wahl:
1.) Entweder Sie bleiben bei den einfachen, aber langsamen Basicroutinen,
2.) Sie setzen sich einmal einen Tag hin und schreiben das Folgende ab.

3.) Sie kaufen sich eine Graphikerweiterung, die Ihnen viel Arbeit abnimmt, oder

4.) Sie besorgen sich die zu diesem Buch erhältliche Programmdiskette mit allen in diesem Buch befindlichen Routinen und Programmen (auch dem Graphikpaket).

Die Graphik völlig zu ignorieren kann ich Ihnen nicht empfehlen.

Doch nun zum Thema: Zunächst wird Ihnen das Assemblerlisting des Graphik - Paketes mit vielen Kommentaren vorgestellt, für diejenigen, die sich für die einzelnen Routinen interessieren. Sie werden sehen, daß einige Dinge aus Geschwindigkeitsgründen etwas anders organisiert sind, als wir es in den obigen Basicprogrammen getan haben. Dazu gehört z.B. die Routine, die beliebige Linien auf den Bildschirm zeichnet. Hier werden nicht direkt die Koordinaten eines jeden Punktes ausgerechnet, aus denen ja dann wieder die Speicheradresse errechnet werden müßte, sondern wir berechnen quasi das Verhältnis der Anzahl der zu gehenden Schritte nach oben/unten zu der Anzahl von Schritten nach rechts/links, um vom Ausgangspunkt zum Endpunkt zu gelangen, und gehen dann stets in einem bestimmten Rythmus, der diesem Verhältnis entspricht, eben in diese Richtungen. Dabei wird die Tatsache voll berücksichtigt, daß stets zwei Punkte über- bzw. nebeneinander liegen müssen, um eine zusammenhängende Linie zu erhalten. Sie werden staunen, wie schnell so etwas geht.

Nach diesem sogenannten Source-Listing wird Ihnen wieder ein Basic-Lader für diejenigen, die keinen Monitor besitzen, angeboten, um das Programm sicher und gut eintippen zu können. Doch hier das Listing:

```
END OF ASSEMBLY!

                    0010          .BA $C800
                    0020          .MC $0800
                    0030          .OS
                    0040          ;
                    0050          ; ************************
                    0060          ; **                    **
                    0070          ; **  HOCH AUFLOESENDES  **
                    0080          ; **   GRAPHIK - PAKET   **
                    0090          ; **                    **
                    0100          ; ************************
                    0110          ;
                    0120          ;
                    0130          ;
                    0140          ;ROM-SPRUNGADRESSEN:
                    0150          ;*******************
                    0160          ;
                    0170 GETBYT   .DE $B79E          ;HOLT BYTEWERT
                    0180 CHKCOM   .DE $AEFD          ;PRUEFT AUF KOMMA
                    0190 CHKGET   .DE $B7F1          ;CHKCOM+GETBYT
                    0200 GETCOR   .DE $B7EB          ;HOLT KOORDINATEN
                    0210 QERR     .DE $B248          ;ILLEGAL QUANTITY
                    0220 V        .DE $D000          ;VIDEOCONTROLLER
                    0230          ;
                    0240          ;NULLSEITIGE REGISTER:
                    0250          ;*********************
                    0260          ;
                    0270 OFFX     .DE $63
                    0280 MSK      .DE $AB            ;MASKE
                    0290 DIF0     .DE $69
                    0300 DIF1     .DE $6A
                    0310 DIF2     .DE $6B
                    0320 DIF3     .DE $6C
                    0330 DIF4     .DE $6D
                    0340 DIF5     .DE $6E
                    0350 ZWN      .DE $6F            ;XK/8 (HLINE)
                    0360 ZA       .DE $70            ;ZAEHLER (HLINE)
                    0370 A        .DE $AC            ;PUNKTADRESSE
                    0380 B        .DE $AC+1
                    0390 XK       .DE $14            ;X-KOORDINATE
                    0400 FLG      .DE $97            ;UNPLOT/PLOT-FLAG
                    0410 USE      .DE $FD            ;DIVERSES
                    0420          ;
                    0430          ;FESTE WERTE:
                    0440          ;************
                    0450          ;
                    0460 GRAPH    .DE $2000          ;GRAPHIKSTART
                    0470 VIDEO    .DE $0400          ;VIDEORAMSTART
                    0480 GSTART   .DE $21            ;GR-START+1-H-BYTE
                    0490 GEND     .DE $3E            ;GR-ENDE-1-H-BYTE
                    0500          ;
                    0510          ;
                    0520          ;ANSTEUERADRESSEN:
                    0530          ;*****************
                    0540          ;
C800- 4C 24 C8      0550          JMP INIT           ;GRAPHIK EIN
C803- 4C 41 C8      0560          JMP GOFF           ;GRAPHIK AUS
C806- 4C 12 C9      0570          JMP GCLEAR         ;GRAPHIK LOESCHEN
C809- 4C 31 C9      0580          JMP SCOLOR         ;FARBE SETZEN (LOESCHEN)
C80C- 4C 2A C9      0590          JMP PCOLOR         ;PLOTFARBE
C80F- 4C 58 C8      0600          JMP PLOT           ;PUNKT SETZEN
C812- 4C 55 C8      0610          JMP UNPLOT         ;PUNKT LOESCHEN
C815- 4C 6B C8      0620          JMP SLINE          ;LINIE ZEICHNEN
```

```
C818- 4C 68 C8  0630          JMP CLLINE        ;LINIE LOESCHEN
C81B- 4C 43 CA  0640          JMP GLOAD         ;GRAPHIK LADEN
C81E- 4C 52 CA  0650          JMP GSAVE         ;GRAPHIK SPEICHERN
C821- 4C 69 CA  0660          JMP HARDC         ;HARDCOPY (GP 100 VC ETC.)
                0670          ;
                0680          ;
                0690          ;GRAPHIK EINSCHALTEN:
                0700          ;*******************
                0710          ;
C824- EA        0720 INIT     NOP               ;KEINE BLOCKADE
C825- AD 11 D0  0730          LDA V+17
C828- 8D 1E CB  0740          STA STORE1
C82B- AD 18 D0  0750          LDA V+24
C82E- 8D 1F CB  0760          STA STORE2        ;ALTE INHALTE RETTEN
C831- A9 3B     0770          LDA #%00111011
C833- 8D 11 D0  0780          STA V+17          ;GRAPHIK EIN
C836- A9 18     0790          LDA #%00011000
C838- 8D 18 D0  0800          STA V+24          ;NACH $2000
C83B- A9 60     0810          LDA #$60          ;CODE FUER RTS ALS BLOCKADE
C83D- 8D 24 C8  0820          STA INIT          ;INIT WIRD BLOCKIERT
C840- 60        0830          RTS
                0840          ;
                0850          ;
                0860          ;GRAPHIK AUSSCHALTEN:
                0870          ;********************
                0880          ;
C841- AD 1E CB  0890 GOFF     LDA STORE1
C844- 8D 11 D0  0900          STA V+17
C847- AD 1F CB  0910          LDA STORE2
C84A- 8D 18 D0  0920          STA V+24          ;ALTE INHALTE RUECKHOLEN
C84D- A9 EA     0930          LDA #$EA          ;CODE FUER NOP FUER
C84F- 8D 24 C8  0940          STA INIT          ;BLOCKADE AUFHEBEN
C852- 4C 44 E5  0950          JMP $E544         ;BILDSCHIRM LOESCHEN
                0960          ;
                0970          ;
                0980          ;PUNKT LOESCHEN:
                0990          ;***************
                1000          ;
C855- A2 00     1010 UNPLOT   LDX #$00          ;LOESCH-FLAG
C857- 2C        1020          .BY $2C
                1030          ;
                1040          ;
                1050          ;PUNKT SETZEN:
                1060          ;*************
                1070          ;
C858- A2 80     1080 PLOT     LDX #$80          ;SETZ-FLAG
C85A- 86 97     1090 PL1      STX #FLG
C85C- 20 FD AE  1100          JSR CHKCOM
C85F- 20 79 C8  1110          JSR TESCOR        ;KOORDINATEN HOLEN
C862- 20 94 C8  1120          JSR HPOSN         ;ADRESSE ERRECHNEN
C865- 4C E2 C8  1130          JMP PLT           ;PUNKT SETZEN/LOESCHEN
                1140          ;
                1150          ;
                1160          ;LINIE LOESCHEN:
                1170          ;***************
                1180          ;
C868- A2 00     1190 CLLINE   LDX #$00          ;LOESCH-FLAG
C86A- 2C        1200          .BY $2C
                1210          ;
                1220          ;
                1230          ;LINIE ZEICHNEN:
                1240          ;***************
                1250          ;
C86B- A2 80     1260 SLINE    LDX #$80          ;SETZ-FLAG
C86D- 20 5A C8  1270          JSR PL1           ;ERSTEN PUNKT SETZEN
C870- 20 FD AE  1280          JSR CHKCOM
```

```
C873- 20 79 C8  1290          JSR TESCOR        ;ZWEITE KOORD. HOLEN
C876- 4C BB C9  1300          JMP HLINE         ;LINIE ZEICHNEN
                1310          ;
                1320          ;
                1330          ;KOORDINATEN TESTEN:
                1340          ;*******************
                1350          ;
C879- 20 EB B7  1360 TESCOR   JSR GETCOR        ;HOLEN
C87C- 8A        1370          TXA
C87D- A8        1380          TAY
C87E- A6 15     1390          LDX *XK+1
C880- C0 C8     1400          CPY #200          ;Y-KOORD. >= 200?
C882- B0 0D     1410          BCS ILLFF         ;JA!
C884- A5 14     1420          LDA *XK
C886- E0 01     1430          CPX #H,320        ;X-KOORD. >= 320?
C888- 90 06     1440          BCC T1            ;JA
C88A- D0 05     1450          BNE ILLFF         ;A:XK-LOW
C88C- C9 40     1460          CMP #L,320        ;X:XK-HIGH
C88E- B0 01     1470          BCS ILLFF         ;Y:YK
C890- 60        1480 T1       RTS
C891- 4C 48 B2  1490 ILLFF    JMP QERR          ;ILLEGAL QUANTITY
                1500          ;
                1510          ;
                1520          ;ADRESSE ERRECHNEN:
                1530          ;******************
                1540          ;
C894- 8C 1C CB  1550 HPOSN    STY YK            ;Y-K
C897- 8D 1A CB  1560          STA XKL           ;X-KL
C89A- 8E 1B CB  1570          STX XKH           ;X-KH (ZWISCHENSPEICHERN)
C89D- 85 14     1580          STA *XK
C89F- 86 15     1590          STX *XK+1
C8A1- 98        1600          TYA
C8A2- 4A        1610          LSR A
C8A3- 4A        1620          LSR A
C8A4- 4A        1630          LSR A             ;INT(Y/8)
C8A5- AA        1640          TAX
C8A6- BD 2F CB  1650          LDA MUL/H,X       ;320*INT(Y/8) (HIGH-BYTE)
C8A9- 85 AD     1660          STA *B
C8AB- 8A        1670          TXA
C8AC- 29 03     1680          AND #3            ;BITS 0+1 ISOLIEREN
C8AE- AA        1690          TAX
C8AF- BD 49 CB  1700          LDA MUL/L,X       ;320*INT(Y/8) (LOW-BYTE)
C8B2- 85 AC     1710          STA *A
C8B4- 98        1720          TYA               ;Y-KOORD.
C8B5- 29 07     1730          AND #7            ;(Y AND 7)
C8B7- 18        1740          CLC
C8B8- 65 AC     1750          ADC *A            ;OFFY=320*INT(Y/8)+(Y AND 7)
C8BA- 85 AC     1760          STA *A            ;**************************
C8BC- A5 14     1770          LDA *XK
C8BE- 29 F8     1780          AND #$F8          ;OFFX=8*INT(X/8)
C8C0- 85 63     1790          STA *OFFX         ;***************
C8C2- A9 20     1800          LDA #H,GRAPH      ;GRAPHIKSEITE
C8C4- 05 AD     1810          ORA *B
C8C6- 85 AD     1820          STA *B            ;+SA
C8C8- 18        1830          CLC
C8C9- A5 AC     1840          LDA *A
C8CB- 65 63     1850          ADC *OFFX         ;AD = OFFY + OFFX + SA
C8CD- 85 AC     1860          STA *A            ;**********************
C8CF- A5 AD     1870          LDA *B
C8D1- 65 15     1880          ADC *XK+1
C8D3- 85 AD     1890          STA *B
C8D5- A5 14     1900          LDA *XK
C8D7- 29 07     1910          AND #7
C8D9- 49 07     1920          EOR #7            ;7-(X AND 7)
C8DB- AA        1930          TAX
C8DC- BD 4D CB  1940          LDA MSKTAB,X      ;MASKENTABELLE
```

```
C8DF- 85 AB     1950         STA *MSK          ;2^(7-(X AND 7))
C8E1- 60        1960         RTS
                1970         ;
                1980         ;
                1990         ;PUNKT PLOTTEN:
                2000         ;**************
                2010         ;
C8E2- A0 00     2020 PLT     LDY #0
C8E4- 08        2030         PHP               ;CARRY-FLAG RETTEN
C8E5- A5 AB     2040         LDA *MSK          ;MASKE
C8E7- 24 97     2050         BIT *FLG          ;PLOT/UNPLOT-FLAG
C8E9- 30 05     2060         BMI PL2           ;PLOT
C8EB- 49 FF     2070         EOR #$FF          ;UNPLOT
C8ED- 31 AC     2080         AND (A),Y
C8EF- 2C        2090         .BY $2C           ;UEBERSPRINGE NAECHSTEN BEFEHL
C8F0- 11 AC     2100 PL2     ORA (A),Y         ;PLOT
C8F2- 91 AC     2110         STA (A),Y
C8F4- A5 AC     2120         LDA *A            ;FARBE SETZEN
C8F6- 85 FD     2130         STA *USE
C8F8- A5 AD     2140         LDA *B
C8FA- 4A        2150         LSR A
C8FB- 66 FD     2160         ROR *USE
C8FD- 4A        2170         LSR A
C8FE- 66 FD     2180         ROR *USE
C900- 4A        2190         LSR A
C901- 66 FD     2200         ROR *USE          ;ADRESSE/8
C903- 29 03     2210         AND #$03
C905- 09 04     2220         ORA #$04          ;VIDEORAM AB $0400
C907- 85 FE     2230         STA *USE+1
C909- AD 1D CB  2240         LDA COLOR         ;FARBE IN
C90C- 91 FD     2250         STA (USE),Y       ;VIDEORAM
C90E- 28        2260         PLP               ;CARRY-FLAG
C90F- A4 6F     2270         LDY *ZWN
C911- 60        2280         RTS
                2290         ;
                2300         ;
                2310         ;GRAPHIK LOESCHEN:
                2320         ;*****************
                2330         ;
C912- A9 20     2340 GCLEAR  LDA #H,GRAPH
C914- 85 FE     2350         STA *USE+1
C916- A0 00     2360         LDY #L,GRAPH      ;Y=0
C918- 84 FD     2370         STY *USE          ;GRAPHIK-STARTADRESSE
C91A- A2 20     2380         LDX #$20          ;LAENGE
C91C- 98        2390         TYA               ;Y=0!
C91D- 91 FD     2400 GC1     STA (USE),Y       ;LOESCHEN
C91F- C8        2410         INY
C920- D0 FB     2420         BNE GC1
C922- E6 FE     2430         INC *USE+1
C924- CA        2440         DEX
C925- D0 F6     2450         BNE GC1
C927- 4C 37 C9  2460         JMP SCOL1         ;VIDEORAM LOESCHEN
                2470         ;
                2480         ;
                2490         ;PUNKTFARBE DEFINIEREN:
                2500         ;**********************
                2510         ;
C92A- 20 F1 B7  2520 PCOLOR  JSR CHKGET        ;KOMMA+FARBE
C92D- 8E 1D CB  2530         STX COLOR
C930- 60        2540         RTS
                2550         ;
                2560         ;
                2570         ;FARBE SETZEN:
                2580         ;*************
                2590         ;
C931- 20 F1 B7  2600 SCOLOR  JSR CHKGET        ;KOMMA+FARBE
```

```
C934- 8E 1D CB  2610         STX COLOR
C937- A2 03     2620 SCOL1   LDX #3
C939- A9 04     2630         LDA #H,VIDEO
C93B- 85 FE     2640         STA *USE+1
C93D- A0 00     2650         LDY #L,VIDEO
C93F- 84 FD     2660         STY *USE          ;ADRESSE VIDEORAM
C941- 84 97     2670         STY *FLG          ;Y=0!
C943- AD 1D CB  2680         LDA COLOR         ;FARBE
C946- 91 FD     2690 GM9     STA (USE),Y       ;SETZEN
C948- C8        2700         INY
C949- C4 97     2710         CPY *FLG
C94B- D0 F9     2720         BNE GM9
C94D- E6 FE     2730         INC *USE+1
C94F- CA        2740         DEX
C950- F0 03     2750         BEQ GM9.
C952- 10 F2     2760         BPL GM9
C954- 60        2770         RTS
C955- A2 E8     2780 GM9.    LDX #$E8
C957- 86 97     2790         STX *FLG          ;SPRITEVEKTOREN.SCHUETZEN
C959- D0 EB     2800         BNE GM9           ;UNBEDINGT
                2810         ;
                2820         ;
                2830         ;VEKTORROUTINEN:
                2840         ;***************
                2850         ;
C95B- A5 AC     2860 UNTEN   LDA *A            ;PUNKTADRESSE-LOW
C95D- 29 07     2870         AND #7
C95F- C9 07     2880         CMP #7
C961- F0 05     2890         BEQ UN1           ;PACKENRAND!
C963- 38        2900         SEC
C964- A9 00     2910         LDA #0
C966- B0 04     2920         BCS UN2           ;ADDIERE 1 (C=1!)
C968- A9 38     2930 UN1     LDA #$38          ;ADDIERE 320-7=313
C96A- E6 AD     2940         INC *B            ;C=1!
C96C- 65 AC     2950 UN2     ADC *A
C96E- 85 AC     2960         STA *A
C970- A9 00     2970         LDA #0
C972- 65 AD     2980         ADC *B
C974- 85 AD     2990         STA *B
C976- 60        3000         RTS
                3010         ;
C977- 30 E2     3020 U/O     BMI UNTEN
                3030         ;
C979- A5 AC     3040 OBEN    LDA *A            ;ADRESSE
C97B- 29 07     3050         AND #7
C97D- F0 05     3060         BEQ OB1           ;PACKENRAND (OBEN)
C97F- 18        3070         CLC
C980- A9 FF     3080         LDA #$FF
C982- 90 04     3090         BCC OB2           ;SUBTRAHIERE 1
C984- A9 C7     3100 OB1     LDA #$C7          ;SUBTAHIERE 320+7=327
C986- C6 AD     3110         DEC *B            ;C=1!
C988- 65 AC     3120 OB2     ADC *A
C98A- 85 AC     3130         STA *A
C98C- A5 AD     3140         LDA *B
C98E- E9 00     3150         SBC #0
C990- 85 AD     3160         STA *B
C992- 60        3170         RTS
                3180         ;
                3190         ;
C993- 46 AB     3200 RECHTS  LSR *MSK          ;MASKE VERSCHIEBEN
C995- 90 0E     3210         BCC RE2           ;NOCH INNERHALB BYTE
C997- 66 AB     3220         ROR *MSK          ;AUSSERHALB BYTE
C999- A5 AC     3230         LDA *A
C99B- C8        3240         INY
C99C- 18        3250         CLC
C99D- 69 08     3260         ADC #8
```

```
C99F- 85 AC     3270         STA *A
C9A1- 90 02     3280         BCC RE2
C9A3- E6 AD     3290         INC *B          ;8 ADDIEREN
C9A5- 60        3300 RE2     RTS
                3310         ;
C9A6- 10 EB     3320 R/L     BPL RECHTS
                3330         ;
C9A8- 06 AB     3340 LINKS   ASL *MSK
C9AA- 90 0E     3350         BCC LI1
C9AC- 26 AB     3360         ROL *MSK
C9AE- A5 AC     3370         LDA *A
C9B0- 88        3380         DEY
C9B1- 38        3390         SEC
C9B2- E9 08     3400 LI3     SBC #8
C9B4- 85 AC     3410         STA *A
C9B6- B0 02     3420         BCS LI1
C9B8- C6 AD     3430         DEC *B          ;8 SUBTRAHIEREN
C9BA- 60        3440 LI1     RTS
                3450         ;
                3460         ;
                3470         ;LINIE ZEICHNEN:
                3480         ;***************
                3490         ;
C9BB- 48        3500 HLINE   PHA             ;A  : X2-LOW-BYTE
C9BC- AD 1B CB  3510         LDA XKH         ;X  : X2-HIGH-BYTE
C9BF- 4A        3520         LSR A           ;Y  : Y2
C9C0- AD 1A CB  3530         LDA XKL         ;XKL: X1-LOW-BYTE
C9C3- 6A        3540         ROR A           ;XKH: X1-LOW-BYTE
C9C4- 4A        3550         LSR A           ;YK : Y1
C9C5- 4A        3560         LSR A
C9C6- 85 6F     3570         STA *ZWN        ;X1 / 8 ZWISCHENSP.
C9C8- 68        3580         PLA
C9C9- 48        3590         PHA
C9CA- 38        3600         SEC
C9CB- ED 1A CB  3610         SBC XKL
C9CE- 48        3620         PHA
C9CF- 8A        3630         TXA
C9D0- ED 1B CB  3640         SBC XKH
C9D3- 85 6C     3650         STA *DIF3       ;X2-X1
C9D5- B0 0A     3660         BCS L3          ;NEGATIV?
C9D7- 68        3670         PLA             ;JA
C9D8- 49 FF     3680         EOR #$FF
C9DA- 69 01     3690         ADC #1
C9DC- 48        3700         PHA
C9DD- A9 00     3710         LDA #0
C9DF- E5 6C     3720         SBC *DIF3       ;VORZEICHENWECHSEL
C9E1- 85 6A     3730 L3      STA *DIF1
C9E3- 85 6E     3740         STA *DIF5       ;(X2-X1)-HIGH
C9E5- 68        3750         PLA
C9E6- 85 69     3760         STA *DIF0
C9E8- 85 6D     3770         STA *DIF4       ;(X2-X1)-LOW
C9EA- 68        3780         PLA
C9EB- 8D 1A CB  3790         STA XKL
C9EE- 8E 1B CB  3800         STX XKH
C9F1- 98        3810         TYA
C9F2- 18        3820         CLC
C9F3- ED 1C CB  3830         SBC YK          ;Y2-Y1
C9F6- 90 04     3840         BCC L4          ;NEGATIV?
C9F8- 49 FF     3850         EOR #$FF
C9FA- 69 FE     3860         ADC #$FE        ;VORZEICHENWECHSEL
C9FC- 85 6B     3870 L4      STA *DIF2       ;(Y2-Y1)
C9FE- 8C 1C CB  3880         STY YK
CA01- 66 6C     3890         ROR *DIF3       ;(X2-X1)/2
CA03- 38        3900         SEC
CA04- E5 69     3910         SBC *DIF0       ;(Y2-Y1)-(X2-X1)
CA06- AA        3920         TAX             ;LOW-BYTE NACH X-REG.
```

```
CA07- A9 FF     3930         LDA #$FF
CA09- E5 6A     3940         SBC *DIF1
CA0B- 85 70     3950         STA *ZA          ;HIGH-BYTE NACH ZA
CA0D- A4 6F     3960         LDY *ZWN
CA0F- B0 05     3970         BCS L5           ;UNBEDINGT
CA11- 0A        3980 L1      ASL A
CA12- 20 A6 C9  3990         JSR R/L          ;RECHTS/LINKS
CA15- 38        4000         SEC
CA16- A5 6D     4010 L5      LDA *DIF4
CA18- 65 6B     4020         ADC *DIF2
CA1A- 85 6D     4030         STA *DIF4
CA1C- A5 6E     4040         LDA *DIF5
CA1E- E9 00     4050         SBC #0           ;(X2-X1)-(Y2-Y1)NACH(X2-X1)
CA20- 85 6E     4060 L2      STA *DIF5
CA22- 84 6F     4070         STY *ZWN
CA24- 20 E2 C8  4080         JSR PLT          ;PUNKT ZEICHNEN
CA27- E8        4090         INX
CA28- D0 05     4100         BNE L6
CA2A- E6 70     4110         INC *ZA
CA2C- D0 01     4120         BNE L6           ;DEC ZAEHLER
CA2E- 60        4130         RTS
CA2F- A5 6C     4140 L6      LDA *DIF3
CA31- B0 DE     4150         BCS L1
CA33- 20 77 C9  4160         JSR U/O          ;UNTEN/OBEN
CA36- 18        4170         CLC
CA37- A5 6D     4180         LDA *DIF4
CA39- 65 69     4190         ADC *DIF0
CA3B- 85 6D     4200         STA *DIF4
CA3D- A5 6E     4210         LDA *DIF5
CA3F- 65 6A     4220         ADC *DIF1
CA41- 50 DD     4230         BVC L2           ;UNBEDINGT
                4240         ;
                4250         ;
                4260         ;GRAPHIK LADEN:
                4270         ;**************
                4280         ;
CA43- 20 FD AE  4290 GLOAD   JSR CHKCOM       ;KOMMA?
CA46- 20 D4 E1  4300         JSR $E1D4        ;PARAMETER HOLEN
CA49- A0 20     4310         LDY #H,GRAPH
CA4B- A2 00     4320         LDX #L,GRAPH     ;STARTADRESSE
CA4D- A9 00     4330         LDA #$00         ;LOAD-FLAG
CA4F- 4C D5 FF  4340         JMP $FFD5        ;LADEN
                4350         ;
                4360         ;
                4370         ;GRAPHIK SPEICHERN:
                4380         ;******************
                4390         ;
CA52- 20 FD AE  4400 GSAVE   JSR CHKCOM       ;KOMMA?
CA55- 20 D4 E1  4410         JSR $E1D4
                4420         ;ENDADRESSE
CA58- A2 3F     4430         LDX #H,GRAPH+8000
CA5A- A0 40     4440         LDY #L,GRAPH+8000
CA5C- A9 00     4450         LDA #L,GRAPH
CA5E- 85 FD     4460         STA *$FD
CA60- A9 20     4470         LDA #H,GRAPH
CA62- 85 FE     4480         STA *$FE         ;STARTADRESSE
CA64- A9 FD     4490         LDA #$FD         ;POINTER
CA66- 4C D8 FF  4500         JMP $FFD8        ;SPEICHERN
                4510         ;
                4520         ;
                4530         ;HARDCOPY SEIKOSHA GP 100 VC:
                4540         ;****************************
                4550         ;
CA69- 20 F1 B7  4560 HARDC   JSR CHKGET       ;HOLE LOG. FILENR.
CA6C- 86 67     4570         STX *$67
CA6E- 20 0F F3  4580         JSR $F30F        ;SUCHT LOG. FILENR.
```

```
CA71- 20 1F F3  4590          JSR $F31F        ;SETZT FILEPARAM.
CA74- A6 67     4600          LDX *$67
CA76- 20 C9 FF  4610          JSR $FFC9        ;KANAL OEFFNEN
CA79- A9 FF     4620          LDA #$FF
CA7B- 85 61     4630          STA *$61         ;MASKE
CA7D- A9 07     4640          LDA #7
CA7F- 85 FD     4650          STA *USE         ;PACKENGROESSE
CA81- A9 1C     4660          LDA #28
CA83- 85 97     4670          STA *FLG         ;ZEILENZAEHLER
CA85- A9 00     4680          LDA #0
CA87- 8D 20 CB  4690          STA ZWIS         ;YK-MERKER
CA8A- A9 28     4700 HA1      LDA #40
CA8C- 8D 21 CB  4710          STA FLG2         ;PACKENZAEHLER
CA8F- A2 04     4720          LDX #4
CA91- BD 2A CB  4730 HA1.     LDA HATAB,X      ;MITTENZENTRIERT
CA94- 20 D2 FF  4740          JSR $FFD2        ;AUSGABE
CA97- CA        4750          DEX
CA98- 10 F7     4760          BPL HA1.
CA9A- A9 00     4770          LDA #0
CA9C- 85 63     4780          STA *$63
CA9E- 85 64     4790          STA *$64         ;XK=0
CAA0- AD 20 CB  4800 HA2      LDA ZWIS
CAA3- 85 65     4810          STA *$65         ;YK
CAA5- A9 00     4820          LDA #0
CAA7- 85 FE     4830          STA *USE+1       ;BUFFERZEIGER
CAA9- A5 63     4840 HA3      LDA *$63         ;XK-L
CAAB- A6 64     4850          LDX *$64         ;XK-H
CAAD- A4 65     4860          LDY *$65         ;YK
CAAF- 20 94 C8  4870          JSR HPOSN        ;POSITION BERECHNEN
CAB2- A0 00     4880          LDY #0
CAB4- B1 AC     4890          LDA (A),Y        ;BYTE HOLEN
CAB6- A6 FE     4900          LDX *USE+1       ;BUFFERZEIGER
CAB8- 9D 22 CB  4910          STA BUFF,X       ;IN BUFFER
CABB- E6 65     4920          INC *$65         ;YK
CABD- E8        4930          INX
CABE- 86 FE     4940          STX *USE+1
CAC0- E4 FD     4950          CPX *USE
CAC2- D0 E5     4960          BNE HA3
CAC4- A9 00     4970          LDA #0
CAC6- A0 07     4980          LDY #7
CAC8- A6 FD     4990 HA4      LDX *USE
CACA- 1E 22 CB  5000 HA5      ASL BUFF,X       ;EIN BIT HOLEN
CACD- 2A        5010          ROL A            ;UND IN AKU SCHIEBEN
CACE- CA        5020          DEX              ;BUFFERZEIGER ERHOEHEN
CACF- 10 F9     5030          BPL HA5
CAD1- 25 61     5040          AND *$61         ;BEI LETZTER ZEILE=#$0F
CAD3- 09 80     5050          ORA #$80         ;HIGH-BYTE
CAD5- 20 D2 FF  5060          JSR $FFD2        ;SENDEN
CAD8- 88        5070          DEY              ;8 BYTES SENDEN
CAD9- 10 ED     5080          BPL HA4
CADB- A5 63     5090          LDA *$63         ;XK-L
CADD- 18        5100          CLC
CADE- 69 08     5110          ADC #8
CAE0- 85 63     5120          STA *$63         ;XK+8
CAE2- 90 02     5130          BCC HA6
CAE4- E6 64     5140          INC *$64         ;XK-H
CAE6- CE 21 CB  5150 HA6      DEC FLG2
CAE9- D0 B5     5160          BNE HA2
CAEB- A9 0D     5170          LDA #$0D         ;CARRIGE RETURN
CAED- 20 D2 FF  5180          JSR $FFD2        ;SENDEN
CAF0- AD 20 CB  5190          LDA ZWIS
CAF3- 18        5200          CLC
CAF4- 69 07     5210          ADC #7
CAF6- 8D 20 CB  5220          STA ZWIS         ;YK+7
CAF9- C6 97     5230          DEC *FLG
CAFB- F0 03     5240          BEQ HA8.
```

```
CAFD- 4C 8A CA   5250 HA8     JMP HA1
CB00- A9 04      5260 HA8.    LDA #4
CB02- C5 FD      5270         CMP *USE
CB04- F0 0C      5280         BEQ HA7
CB06- 85 FD      5290         STA *USE          ;LETZTE ZEILE
CB08- A9 01      5300         LDA #1
CB0A- 85 97      5310         STA *FLG          ;FLAG
CB0C- A9 0F      5320         LDA #$F
CB0E- 85 61      5330         STA *$61          ;MASKE
CB10- D0 EB      5340         BNE HA8
CB12- A9 0F      5350 HA7     LDA #15           ;NORMAL MODUS
CB14- 20 D2 FF   5360         JSR $FFD2
CB17- 4C CC FF   5370         JMP $FFCC         ;SCHLIESST KANAL
                 5380         ;
                 5390         ;
                 5400         ;INTERNE SPEICHER:
                 5410         ;*****************
                 5420         ;
CB1A-            5430 XKL     .DS 1             ;XK-LOW-ZWISCHENSPEICHER
CB1B-            5440 XKH     .DS 1             ;XK-HIGH-ZWISCHENSPEICHER
CB1C-            5450 YK      .DS 1             ;YK-ZWISCHENSPEICHER
CB1D-            5460 COLOR   .DS 1             ;FARBE
CB1E-            5470 STORE1  .DS 1             ;V+17-REG.-ZWISCHENSP.
CB1F-            5480 STORE2  .DS 1             ;V+24-REG.-ZWISCHENSP.
CB20-            5490 ZWIS    .DS 1             ;ZWISCHENSPEICHER
CB21-            5500 FLG2    .DS 1             ;ZWISCHENSPEICHER
CB22-            5510 BUFF    .DS 8             ;BUFFER FUER HARDCOPY
                 5520         ;
                 5530         ;
                 5540         ;TABELLEN:
                 5550         ;*********
                 5560         ;
                 5570         ;DRUCKERZEICHEN:
                 5580         ;(RUECKWAERTS)
                 5590         ;
                 5600         ;MITTENZENTRIERT/GRAPHIK EIN
                 5610         ;
CB2A- 50 00 10   5620 HATAB   .BY 80 0 16 27 8
CB2D- 1B 08
                 5630         ;
                 5640         ;
                 5650         ;MULTIPLIKATIONSTABELLE:
                 5660         ;(N*320 FUER N=0 BIS N=24)
                 5670         ;
                 5680         ;HIGH-BYTES
                 5690         ;
CB2F- 00 01 02   5700 MUL/H   .BY 0 1 2 3 5 6 7 8 10 11 12 13 15 16
CB32- 03 05 06
CB35- 07 08 0A
CB38- 0B 0C 0D
CB3B- 0F 10
CB3D- 11 12 14   5710         .BY 17 18 20 21 22 23 25 26 27 28 30 31
CB40- 15 16 17
CB43- 19 1A 1B
CB46- 1C 1E 1F
                 5720         ;
                 5730         ;LOW-BYTES
                 5740         ;
CB49- 00 40 80   5750 MUL/L   .BY $00 $40 $80 $C0
CB4C- C0
                 5760         ;
                 5770         ;
                 5780         ;MASKENTABELLE:
                 5790         ;
CB4D- 01 02 04   5800 MSKTAB  .BY %00000001 %00000010 %00000100 %00001000
CB50- 08
```

```
CB51- 10 20 40  5810          .BY %00010000 %00100000 %01000000 %10000000
CB54- 80
                5820          .EN
END OF ASSEMBLY!


 ---  LABEL FILE:  --

A =00AC                 B =00AD
BUFF =CB22              CHKCOM =AEFD
CHKGET =B7F1            CLLINE =C868
COLOR =CB1D             DIF0 =0069
DIF1 =006A              DIF2 =006B
DIF3 =006C              DIF4 =006D
DIF5 =006E              FLG =0097
FLG2 =CB21              GC1 =C91D
GCLEAR =C912            GEND =003E
GETBYT =B79E            GETCOR =B7EB
GLOAD =CA43             GM9 =C946
GM9. =C955              GOFF =C841
GRAPH =2000             GSAVE =CA52
GSTART =0021            HA1 =CA8A
HA1. =CA91              HA2 =CAA0
HA3 =CAA9               HA4 =CAC8
HA5 =CACA               HA6 =CAE6
HA7 =CB12               HA8 =CAFD
HA8. =CB00              HARDC =CA69
HATAB =CB2A             HLINE =C9BB
HPOSN =C894             ILLFF =C891
INIT =C824              L1 =CA11
L2 =CA20                L3 =C9E1
L4 =C9FC                L5 =CA16
L6 =CA2F                LI1 =C9BA
LI3 =C9B2               LINKS =C9A8
MSK =00AB               MSKTAB =CB4D
MUL/H =CB2F             MUL/L =CB49
OB1 =C984               OB2 =C988
OBEN =C979              OFFX =0063
PCOLOR =C92A            PL1 =C85A
PL2 =C8F0               PLOT =C858
PLT =C8E2               QERR =B248
R/L =C9A6               RE2 =C9A5
RECHTS =C993            SCOL1 =C937
SCOLOR =C931            SLINE =C86B
STORE1 =CB1E            STORE2 =CB1F
T1 =C890                TESCOR =C879
U/O =C977               UN1 =C968
UN2 =C96C               UNPLOT =C855
UNTEN =C95B             USE =00FD
V =D000                 VIDEO =0400
XK =0014                XKH =CB1B
XKL =CB1A               YK =CB1C
ZA =0070                ZWIS =CB20
ZWN =006F
//0000,CB55,0B55
```

Kommen wir zum Basiclader:

```
100 FOR I = 51200 TO 52054
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  76, 36,200,  76, 65,200,  76, 18,201,  76, 49,201
130 DATA  76, 42,201,  76, 88,200,  76, 85,200,  76,107,200
140 DATA  76,104,200,  76, 67,202,  76, 82,202,  76,105,202
150 DATA 234,173,  17,208,141,  30,203,173,  24,208,141,  31
160 DATA 203,169,  59,141,  17,208,169,  24,141,  24,208,169
170 DATA  96,141,  36,200,  96,173,  30,203,141,  17,208,173
180 DATA  31,203,141,  24,208,169,234,141,  36,200,  76,  68
190 DATA 229,162,   0,  44,162,128,134,151,  32,253,174,  32
200 DATA 121,200,  32,148,200,  76,226,200,162,   0,  44,162
210 DATA 128,  32,  90,200,  32,253,174,  32,121,200,  76,187
220 DATA 201,  32,235,183,138,168,166,  21,192,200,176,  13
230 DATA 165,  20,224,   1,144,   6,208,   5,201,  64,176,   1
240 DATA  96,  76,  72,178,140,  28,203,141,  26,203,142,  27
250 DATA 203,133,  20,134,  21,152,  74,  74,  74,170,189,  47
260 DATA 203,133,173,138,  41,   3,170,189,  73,203,133,172
270 DATA 152,  41,   7,  24,101,172,133,172,165,  20,  41,248
280 DATA 133,  99,169,  32,   5,173,133,173,  24,165,172,101
290 DATA  99,133,172,165,173,101,  21,133,173,165,  20,  41
300 DATA   7,  73,   7,170,189,  77,203,133,171,  96,160,   0
310 DATA   8,165,171,  36,151,  48,   5,  73,255,  49,172,  44
320 DATA  17,172,145,172,165,172,133,253,165,173,  74,102
330 DATA 253,  74,102,253,  74,102,253,  41,   3,   9,   4,133
340 DATA 254,173,  29,203,145,253,  40,164,111,  96,169,  32
350 DATA 133,254,160,   0,132,253,162,  32,152,145,253,200
360 DATA 208,251,230,254,202,208,246,  76,  55,201,  32,241
370 DATA 183,142,  29,203,  96,  32,241,183,142,  29,203,162
380 DATA   3,169,   4,133,254,160,   0,132,253,132,151,173
390 DATA  29,203,145,253,200,196,151,208,249,230,254,202
400 DATA 240,   3,  16,242,  96,162,232,134,151,208,235,165
410 DATA 172,  41,   7,201,   7,240,   5,  56,169,   0,176,   4
420 DATA 169,  56,230,173,101,172,133,172,169,   0,101,173
430 DATA 133,173,  96,  48,226,165,172,  41,   7,240,   5,  24
440 DATA 169,255,144,   4,169,199,198,173,101,172,133,172
450 DATA 165,173,233,   0,133,173,  96,  70,171,144,  14,102
460 DATA 171,165,172,200,  24,105,   8,133,172,144,   2,230
470 DATA 173,  96,  16,235,   6,171,144,  14,  38,171,165,172
480 DATA 136,  56,233,   8,133,172,176,   2,198,173,  96,  72
```

```
490 DATA 173, 27,203, 74,173, 26,203,106, 74, 74,133,111
500 DATA 104, 72, 56,237, 26,203, 72,138,237, 27,203,133
510 DATA 108,176, 10,104, 73,255,105,  1, 72,169,  0,229
520 DATA 108,133,106,133,110,104,133,105,133,109,104,141
530 DATA  26,203,142, 27,203,152, 24,237, 28,203,144,  4
540 DATA  73,255,105,254,133,107,140, 28,203,102,108, 56
550 DATA 229,105,170,169,255,229,106,133,112,164,111,176
560 DATA   5, 10, 32,166,201, 56,165,109,101,107,133,109
570 DATA 165,110,233,  0,133,110,132,111, 32,226,200,232
580 DATA 208,  5,230,112,208,  1, 96,165,108,176,222, 32
590 DATA 119,201, 24,165,109,101,105,133,109,165,110,101
600 DATA 106, 80,221, 32,253,174, 32,212,225,160, 32,162
610 DATA   0,169,  0, 76,213,255, 32,253,174, 32,212,225
620 DATA 162, 63,160, 64,169,  0,133,253,169, 32,133,254
630 DATA 169,253, 76,216,255, 32,241,183,134,103, 32, 15
640 DATA 243, 32, 31,243,166,103, 32,201,255,169,255,133
650 DATA  97,169,  7,133,253,169, 28,133,151,169,  0,141
660 DATA  32,203,169, 40,141, 33,203,162,  4,189, 42,203
670 DATA  32,210,255,202, 16,247,169,  0,133, 99,133,100
680 DATA 173, 32,203,133,101,169,  0,133,254,165, 99,166
690 DATA 100,164,101, 32,148,200,160,  0,177,172,166,254
700 DATA 157, 34,203,230,101,232,134,254,228,253,208,229
710 DATA 169,  0,160,  7,166,253, 30, 34,203, 42,202, 16
720 DATA 249, 37, 97,  9,128, 32,210,255,136, 16,237,165
730 DATA  99, 24,105,  8,133, 99,144,  2,230,100,206, 33
740 DATA 203,208,181,169, 13, 32,210,255,173, 32,203, 24
750 DATA 105,  7,141, 32,203,198,151,240,  3, 76,138,202
760 DATA 169,  4,197,253,240, 12,133,253,169,  1,133,151
770 DATA 169, 15,133, 97,208,235,169, 15, 32,210,255, 76
780 DATA 204,255, 48, 48, 48, 32, 58, 32,130, 32, 88, 32
790 DATA  58, 32,143, 32, 87, 65, 80,  0, 16, 27,  8,  0
800 DATA   1,  2,  3,  5,  6,  7,  8, 10, 11, 12, 13, 15
810 DATA  16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 30
820 DATA  31,  0, 64,128,192,  1,  2,  4,  8, 16, 32, 64
830 DATA 128, 76, 79
840 IF S <> 104883 THEN PRINT "FEHLER IN DATAS !!" : END
850 PRINT "OK"
```

Wie ist nun diese Graphik-Hilfsprogramm anzuwenden?
Der Aufruf sämtlicher 12 Befehle erfolgt über eine SYS-Anweisung, die in einigen Fällen noch durch einige Parameter

ergänzt wird. Das erscheint im ersten Moment etwas ungewöhnlich, da die Syntax des eigentlichen SYS-Befehls keine weiteren Parameter außer der Adresse zuläßt, Sie werden sich aber schnell daran gewöhnen. Am besten verfährt man, wie dies im Anwendungsbeispiel vorgeführt wird: Man definiert zunächst am Programmanfang zwölf Variablen mit den Sprungadressen zu den einzelnen Befehlen. Im Verlauf des übrigen Programms geschieht der Aufruf dann stets über diese Variablen, wobei bei Bedarf die notwendigen Parameter ergänzt werden. Eine kleine Tabelle informiert Sie über die Sprungadressen und die Syntax der verschiedenen Befehle:

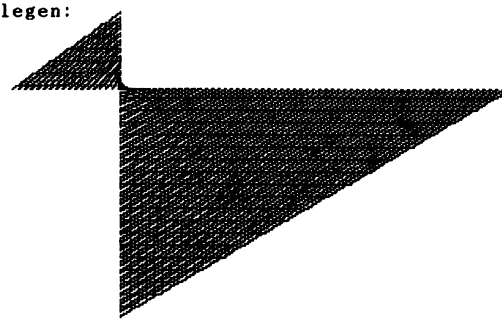
```
SYS 51200                - Graphik einschalten
SYS 51203                - Graphik ausschalten
SYS 51206                - Graphik löschen
SYS 51209,PF*16+HF       - Farbe setzen
SYS 51212,PF*16+HF       - Plotfarbe ändern
SYS 51215,X,Y            - Punkt setzen
SYS 51218,X,Y            - Punkt löschen
SYS 51221,X1,Y1,X2,Y2 - Linie zeichnen
SYS 51224,X1,Y1,X2,Y2 - Linie löschen
SYS 51227,"name",GA      - Graphik laden
SYS 51230,"name",GA      - Graphik speichern
SYS 51233,LF             - Hardcopy S.GP-100VC
```

Dabei bedeuten:

```
PF      - Farbe eines Graphikpunktes v. 0-15
HF      - Hintergrundfarbe von 0 bis 15
X       - X-Koordinate eines Punktes (0-319)
Y       - Y-Koordinate eines Punktes (0-199)
X1/2    - X-Koordinate des Start-/Endpunktes
Y1/2    - Y-Koordinate des Start-/Endpunktes
"name"  - Filename (auch als Stringspeicher)
GA      - Geräteadresse (1 oder 8)
LF      - logische Filenummer v. OPEN LF,GA
```

Bevor wir direkt in unser Demonstrationsprogramm einsteigen, sollten wir kurz die wesentlichsten Dinge besprechen, wobei Sie die Details am besten durch testen und probieren herausbekommen:

Die ersten drei Befehle dürften klar sein, wobei beachtet werden sollte, daß beim Einschalten der Graphik diese nicht, beim Ausschalten jedoch der Bildschirm gelöscht wird.
Beim Setzen der Farbe wird die Farbe aller Graphikpunkte und des Hintergrundes für das gesamte Graphikbild festgelegt. Wählen Sie also den Hintergrund grün (Farbcode: 5) und die Farbe der Punkte violett (Farbcode: 4), so setzen Sie PF=5 und HF=4. Bei jedem gezeichneten Punkt aber wird gleichfalls das entsprechende Farbbyte des Videoram gesetzt. Die jeweilige Farbe wird erstens durch den gerade besprochenen Befehl gesetzt (damit ändert sich die Farbe beim zeichnen nicht), und zweitens durch das folgende Kommando, das lediglich besagt, daß ab jetzt alle neu gezeichneten Figuren in dieser neuen Farbe gezeichnet werden.
Ihnen stehen zwei Befehle zur Verfügung, um einen Punkt zu zeichnen und um einen Punkt zu löschen. Gleichzeitig wird im Farbram die aktuelle Farbe gesetzt.
Wollen Sie eine Linie von den Koordinaten X1,Y1 nach X2,Y2 zeichnen bzw. löschen, so verwenden Sie die beiden nächsten Befehle. Was eben zu der Farbsetzung gesagt wurde, gilt hier natürlich genauso.
Die zwei Kommandos zum Laden und speichern von Graphik verwenden Sie bitte genauso, wie Sie Basicprogramme laden/speichern - unter Angabe des Filenamens und der Geräteadresse (GA=1 für Kassettenbetrieb, GA=8 für Diskette).
Bevor Sie eine Hardcopy auf dem Drucker Seikosha GP-100VC (oder Epson mit DATA BECKER - Interface) starten, müssen Sie z.B. mit OPEN 1,4 entsprechend den Druckerkanal öffnen. Anschließend folgt ein SYS HC,1 (da wir LF, die logische Filenummer als 1 gewählt haben) und ein CLOSE 1 (s. Demoprogramm).
Damit sollten die größten Unklarheiten beseitigt sein, und wir können loslegen:

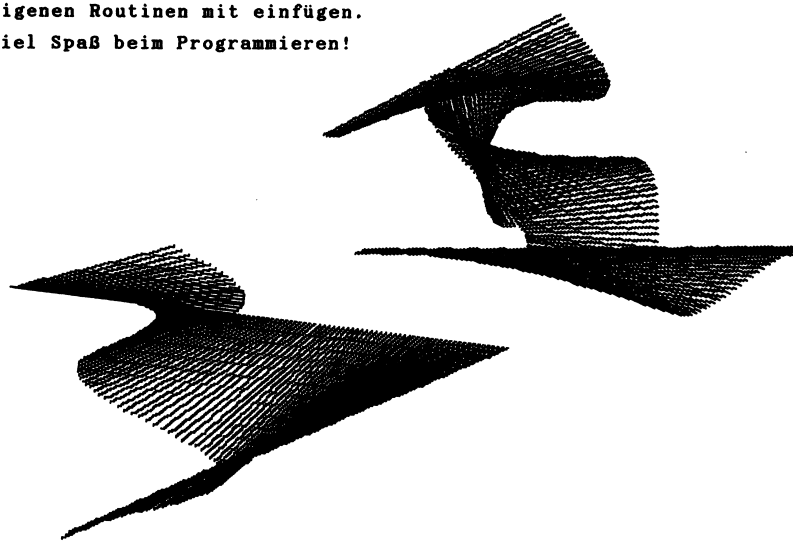


```
10 REM **********************
20 REM **                    **
30 REM **  GRAPHIK-PAKET     **
40 REM **                    **
50 REM **    - D E M O -     **
60 REM **                    **
70 REM **********************
80 REM
90 REM MASCHINENROUTINEN:
100 IN=51200 : OF=51203 :REM INIT      /GRAPHIK OFF
110 GC=51206 : SC=51209 :REM GCLEAR    /SET COLOR
120 PC=51212 : PL=51215 :REM PCOLOR    /PLOT
130 UP=51218 : SL=51221 :REM UNPLOT    /SET LINE
140 CL=51224 : GL=51227 :REM CLR LINE /GLOAD
150 GS=51230 : HC=51233 :REM GSAVE     /HARDCOPY
200 REM
210 REM BEISPIELE:
220 REM **********
230 REM
300 SYS IN : REM GRAPHIK EIN
310 SYS GC : REM GRAPHIK LOESCHEN
320 SYS SC,1*16+2 : REM FARBE SETZEN
330 SYS PC,7*16+2 : REM PLOTFARBE SETZEN
340 REM
350 REM FIGUR 1:
360 REM ********
370 REM
380 FOR X=1 TO 319 STEP 4
390 SYS SL,X,50,70,X/1.6 : REM LINIEN
400 NEXT X
410 REM
420 FOR X=1 TO 5000 : NEXT X : REM WARTESCHLEIFE
430 GOSUB 2000 : SYS GC : REM GRAPHIK LOESCHEN
440 REM
450 REM FIGUR 2:
460 REM ********
470 REM
480 FOR X=1 TO 319 STEP 3
490 SYS SL,X,40,50*SIN(X/30)+100,X/1.6
500 NEXT X
510 REM
```

```
520 FOR X=1 TO 5000 : NEXT X : REM WARTESCHLEIFE
530 GOSUB 2000 : SYS GC : REM GRAPHIK LOESCHEN
540 REM
550 REM FIGUR 3:
560 REM ********
570 REM
580 FOR X=1 TO 319 STEP 2
590 SYS SL,X,40*COS(X/20)+100,50*SIN(X/30)+100,X/1.6
600 NEXT X
610 REM
620 FOR X=1 TO 5000 : NEXT X : REM WARTESCHLEIFE
630 GOSUB 2000 : SYS GC : REM GRAPHIK LOESCHEN
640 REM
1000 WAIT 198,255 : REM AUF TASTE WARTEN
1100 SYS OF : END
2000 SYS OF : REM GRAPHIK AUS
2010 PRINT "WOLLEN SIE:":PRINT
2020 PRINT "(1) - GRAPHIK LADEN"
2030 PRINT "(2) - GRAPHIK SPEICHERN"
2040 PRINT "(3) - HARDCOPY"
2050 PRINT "(4) - WEITER" : PRINT
2060 POKE 198,0 : REM TASTEN LOESCHEN
2070 WAIT 198,255 : REM AUF TASTE WARTEN
2080 GET A$
2090 ON VAL(A$) GOTO 2200,2300,2400,2500
2100 GOTO 2000
2200 REM
2210 REM GRAPHIK LADEN:
2220 REM **************
2230 REM
2240 INPUT "FILENAME,GA";FI$,GA
2250 SYS GL,FI$,GA : REM LADEN
2260 SYS IN : REM GRAPHIK EIN
2270 SYS SC,16*3+9
2280 FOR X=1 TO 5000 : NEXT X : REM WARTEN
2290 GOTO 2000
2300 REM
2310 REM GRAPHIK SPEICHERN:
2320 REM ******************
2330 REM
2340 INPUT "FILENAME,GA";FI$,GA
```

```
2350 SYS GS,FI$,GA : REM SPEICHERN
2360 GOTO 2000
2400 REM
2410 REM HARDCOPY:
2420 REM *********
2430 REM
2440 PRINT "DRUCKER EINSCHALTEN UND TASTE DRUECKEN"
2450 POKE 198,0 : WAIT 198,255 : GET A$
2460 PRINT : PRINT "BITTE WARTEN!"
2470 OPEN 1,4 : REM DRUCKERKANAL OEFFNEN
2480 SYS HC,1 : REM HARDCOPY
2490 CLOSE 1 : REM SCHLIESSEN
2495 GOTO 2000
2500 REM
2510 REM WEITER:
2520 REM *******
2530 REM
2540 SYS IN : SYS SC,16*2+7 : RETURN
```

In diesem kleinen Demoprogramm wurden Ihnen einige Anwendungsbeispiele dargelegt. Sie können ohne Weiteres Ihre eigenen Routinen mit einfügen.
Viel Spaß beim Programmieren!



5. Kapitel
Anwendungen

Nachdem wir nun das große Kapitel der grundlegenden Graphikprogrammierung hinter uns haben, in dem uns die verschiedenen Möglichkeiten des Zeichnens von Punkten, Linien und Kreisen, der Sprite- oder Zeichensatz - Darstellung und -entwicklung, sowie die diversen Ausgabebedienungen und vieles mehr dargelegt wurden.
Doch was fangen wir mit diesem Wissen an? Wie verwendet man etwa Linien und Kreise, um unterschiedliche Sachverhalte darzustellen? Was bringen uns die Sprites? Und so weiter und sofort.
In diesem Kapitel sollen Ihnen einige Beispiele nahegebracht werden, die Ihnen die Arbeit mit diesen Themen schmackhaft machen sollen. Viele Tricks und Tips können Sie den einzelnen Paragraphen entnehmen, die Sie in Ihren Programmen gut verwenden können.
Drei große Abschnitte behandeln die verschiedenen Möglichkeiten der hochauflösenden Graphik oder einer schönen Anwendung der Sprites in Laufschriften, die besonders im kommerziellen Gebrauch Verwendung finden. Stellen Sie sich doch einmal die Repräsentation eines Produktes oder Sonderangebotes mit Hilfe der Sprites und den enormen Graphik- und Soundfähigkeiten des Commodore 64 vor. Sind das nicht verlockende Aussichten?
Als letzten Abschnitt zeigen wir Ihnen die Möglichkeiten, die sich z.B. bei Spielen ergeben, einige nützliche Routinen werden auch hier vorgestellt.

5.1 Graphikanwendungen

Lassen wir keine Müdigkeit aufkommen, gehen wir gleich zur Sache. Alle im folgenden angeführten Beispiele verwenden den Befehlssatz, der Ihnen durch das kleine Graphik - Paket in Kapitel 4 zur Verfügung gestellt wurde, um sie nicht allzu

langsam werden zu lassen. Haben Sie sich nicht die Mühe gemacht und unser Hilfsprogramm abgetippt, dann können Sie selbstverständlich auch die Basic - Unterprogramme verwenden, die Sie im vorherigen Kapitel kennengelernt haben. Sie brauchen lediglich die einzelnen Parameter in die entsprechenden Speicher zu geben und das jeweilige Unterprogramm aufzurufen. Vergessen Sie nicht, am Anfang ihres Programmes V und SA für die Basisadresse des Videocontrollers und den Graphikspeicher mit V=53248 und SA=8192 festzulegen.
Besitzen Sie eine Graphikbefehlserweiterung, die Sie sich vielleicht inzwischen zugelegt haben, dann können Sie die einzelnen SYS-Befehle ebenfalls durch die entsprechenden Graphikbefehle ersetzen. Die Programme sind extra so gehalten, daß derartige Veränderungen nicht besonders schwierig sind.
Vor allem die hochauflösende Graphik läßt sich gut für kommerzielle Zwecke, aber auch für hübsche Privatanwendungen (Schule, Beruf, Freizeit) verwenden. Arbeiten Sie einmal die folgenden Abschnitte durch.

5.1.1. Funktionendarstellung

Eine beliebte Art, die hochauflösende Graphik zu nutzen, ist das Zeichnen der Graphen verschiedener Funktionen. Dabei werden oft in Beispielprogrammen Sinus- oder Cosinuskurven verwendet (s. Beispielprogramm in Paragraph 4.2.2.1). Dies hat meist zwei Gründe: erstens sieht das Ganze recht eindrucksvoll aus, und zweitens - was wohl das Wichtigere ist - man hat nicht so große Probleme mit eventuellen Bereichsüberschreitungen. Letzteres ist die größte Schwierigkeit bei dem Zeichnen solcher Bilder. Um diesem Problem auf die Spur zu kommen, müssen wir uns wieder ein wenig mit der Mathematik beschäftigen.
Eine Funktion ist eine (eindeutige) Abbildung einer Menge auf eine andere, d.h. jedem Element der einen Menge ist genau ein Element der anderen zugeordnet (aber nicht unbedingt umgekehrt). Die Elemente der ersten Menge nennt man in algebraischen Funktionen auch x, die der zweiten Menge y. x und y sind in diesem Fall durch irgendeine mathematische

Formel bzw. Gleichung verbunden (in dem Fall der Linien aus Abschnitt 4.2.2.2 beispielsweise durch die Geradengleichung):

y = f(x) sprich: y gleich f von x

Will man z.B. ein Zuordnungspaar feststellen, so setzt man für x einen beliebigen Wert ein und kann so y errechnen. Die Beziehung zwischen x und y kann man gleichfalls graphisch in einem Koordinatensystem darstellen. Dabei stellt die horizontale Achse des Systems alle möglichen x- (Abszisse), die vertikale dagegen alle möglichen y-Werte (Ordinate) der Funktion dar. Kennen wir ein x/y-Paar, so fällen wir ein Lot auf den betreffenden x-Wert der x- und entsprechend auf den y-Wert der y-Achse. Am Schnittpunkt dieser beiden Senkrechten wird nun ein Punkt eingetragen. Um ein möglichst genaues Bild des entstehenden Graphen zu erhalten, müssen wir viele solcher Koordinatenpaare berechnen und in unser System eintragen. In der Computerpraxis sieht das dann so aus, daß schrittweise x-Koordinaten, die stetig von einem bestimmten Wert bis zu einem zweiten Wert ansteigen (durch eine FOR...NEXT-Schleife realisierbar), in die jeweilige Formel eingesetzt werden, und dadurch die fehlende y-Koordinate errechnet wird. Das Koordinatenpaar dient nun zum Zeichnen eines Punktes auf dem Bildschirm. Beim Zeichnen einer Linie oder eines Kreises im 4. Kapitel wurde das gleiche Prinzip angewendet. Das folgende Programm mag dies noch einmal veranschaulichen:

```
70 REM MASCHINENROUTINEN:
100 IN=51200 : OF=51203 :REM INIT    /GRAPHIK OFF
110 GC=51206 : SC=51209 :REM GCLEAR  /SET COLOR
120 PC=51212 : PL=51215 :REM PCOLOR  /PLOT
130 UP=51218 : SL=51221 :REM UNPLOT  /SET LINE
140 CL=51224 : GL=51227 :REM CLR LINE/GLOAD
150 GS=51230 : HC=51233 :REM GSAVE   /HARDCOPY
160 REM
170 REM ********************
180 REM
200 SYS IN : SYS GC : SYS SC,16*1+6 : REM GRAPHIK INIT
210 FOR X=0 TO 319
```

```
220 Y = X^2 : REM FUNKTIONSWERT ERRECHNEN
230 IF Y>199 THEN WAIT 198,255 : SYS OF : END : REM
BEREICHSUEBERSCHREITUNG
240 SYS PL,X,Y : REM PUNKT SETZEN
250 NEXT X
```

Wie Sie sehen wird hier in Zeile 230 geprüft, ob der y-Wert noch im Bereich des Bildschirms liegt, um ein ILLEGAL QUANTITY ERROR zu verhindern. Unter anderem darüber soll hier diskutiert werden.
Die in diesem Programm gewählte Funktion bereitet uns ein wenig Kopfzerbrechen, denn zum ersten kommt nur einer der erwarteten zwei Parabeläste auf das Bild und der auch noch verkehrt herum, dann ist die Funktion durchaus nicht bildfüllend und drittens wurden viel zu wenig Punkte berechnet, um einen schönen Kurvenverlauf zu erreichen. Dies hat die folgenden Gründe:
Auf unserem Computer besitzt unser Koordinatensystem eine festgelegte Ausdehnung. Wir können für x und y beispielsweise keine negativen oder gebrochenen Zahlen oder Werte einsetzen, die größer sind als 319 (für x) bzw. 199 (für y). Trotzdem gibt es Funktionen, die gerade in diesen Bereichen die interessanten Teile besitzen. So liefern einfache Winkelfunktionen wie:

```
y = sin(x)      oder    y = cos(x)
```

nur Werte zwischen -1 und 1 für y und besitzen eine Schwingungsperiode von 2*pi, also ca. 6, was direkt für unsere Anwendung nicht zu gebrauchen ist.
Es existieren nun Methoden, die diesem Dilemma Abhilfe verschaffen. Dazu gehören:

- Scalierung
- Verschiebung
- Verzerrung

a) <u>Scalierung:</u>
Unter Scalierung versteht man das Vergrößern oder Verkleinern der einzelnen Koordinatenwerte und damit der gesamten Kurve. Dies wird durch einfaches Multiplizieren

der x- und(!) y-Werte mit einem bestimmten Faktor erreicht. Dies kann auf zwei Arten geschehen:

1.) Es werden zunächst die normalen Werte für x zur Berechnung der unveränderten Funktion verwendet und nachher x- und y-Wert mit dem besagten Faktor multipliziert. Dabei hängt es vom Faktor (hier f genannt) ab, ob Sie damit den Graphen verkleinern oder vergrößern:

Faktor	Auswirkung
0<f<1	Verkleinerung
f>1	Vergrößerung

Wählen Sie zusätzlich f<0, so erscheint die Funktion spiegelverkehrt und kopfüber. Eine Basicroutine wäre hierzu etwa:

```
110 F=30 : REM VERGROESSERUNGSFAKTOR
120 FOR X=0 TO 10
130 Y = SIN(X) : REM WERT ERRECHNEN
140 Y = F*Y : X = F*X : REM SCALIEREN
150 IF Y>199 OR X>319 THEN WAIT 198,255 : SYS OF : END
160 SYS PL,X,Y : REM PUNKT ZEICHNEN
170 NEXT X
```

Dieses Programm läuft natürlich nur mit dem Graphik - Paket und der vorherigen Variablendefinierung. Außerdem liefert es noch negative Werte für y

2.) Sie bauen die Scalierung direkt in die Funktion mit ein. Dazu erzeugen Sie direkt scalierte Werte für x (durch die FOR...NEXT-Schleife), müssen diese in der Formel dann aber durch Dividieren von x durch f wieder rücksetzen. Die y-Scalierung erfolgt dann direkt in der Formel. Das könnte dann etwa so aussehen:

```
110 F=30 : REM VERGROESSERUNGSFAKTOR
120 FOR X=0*F TO 10*F
130 Y = F * SIN(X/F) : REM SCAL. WERT
140 IF Y>199 THEN WAIT 198,255 : SYS OF : END
150 SYS PL,X,Y : REM PUNKT ZEICHNEN
160 NEXT X
```

Diese Form hat insbesondere drei Vorteile: Erstens ist sie sichtbar kürzer und schneller, zweitens ist damit gleich von Anfang an überprüfbar, ob größere Werte für x gewählt werden, als erlaubt, denn wir wählen den Umfang der Schleife direkt danach, und drittens berechnen Sie schon hier statt der obigen 11 ganze 301 Werte, was die Genauigkeit der Kurve natürlich beträchtlich steigert. Trotzdem hat die ganze Sache noch einen Haken: y wird immer noch negativ.

b) Verschiebung:

Um diesen Punkt zu beheben, können wir den gesamten Graphen in dem Koordiantensystem in alle Richtungen verschieben. Wir können also die Sinuskurve aus dem negativen herunter in den positiven Bereich verschieben, sodaß y nur noch positive Werte annimmt. Oder wir verschieben den obigen Quadratfunktionsgraphen um einen bestimmten Betrag nach rechts, wodurch sein linker Ast sichtbar wird.
Dies funktioniert, indem zu den beiden Koordinaten jeweils bestimmte Werte hinzuaddiert werden. Diese Werte müssen nicht unbedingt gleich sein, wie bei der Scalierung, um das Aussehen des Graphen nicht zu beeinflussen. Addieren wir einen Wert b zu der y-Koordinate, so verschieben wir den Graphen nach oben bzw. unten, wenn b negativ ist (wir führen also eigentlich eine Subtraktion durch). Geschieht dies mit der x-Koordinate (a wird addiert), so resultiert eine Rechts- (für a>0) bzw. eine Linksverschiebung (a<0). Auch hier gibt es wieder die beiden Möglichkeiten, die völlig analog zu oben funktionieren:

1.) Der Wert der beiden Koordinaten wird errechnet und dann die beiden Summanden hinzuaddiert. Die Zeilen 130/140 in dem unter a)1.) stehenden Programm währen also so zu ersetzen (ohne Scalierung):

```
130 Y = SIN(X)
140 Y = Y+B : X = X+A
```

2.) Auch hier können wir das Ganze in eine Formel packen und erhalten (Zeilen 120/130 des unter a)2.) stehenden Programms):

```
120 FOR X=1 TO 10
130 Y = SIN(X-A) + B
```

Sie sehen, A wird von X abgezogen, statt addiert.
Unser Quadratprogramm vom Anfang könnten wir dann umschreiben, indem wir die Zeile 220 ersetzen durch z.B.:

```
220 Y = (X-13)^2 + 5
```

Schon erhalten wir ein ganz anderes Ergebnis. Hier wurde a=13 und b=5 gesetzt. Wählen wir für a z.B. größere Werte, so wird nichts gezeichnet, da y dann zu groß wird, was ja von unserer Abfrage in Zeile 230 abgefangen wird. Wie dieses Problem zu lösen ist, wird später erläutert.

c) Verzerrung:

Die Verzerrung ist einfach nur ein allgemeinerer Fall der Scalierung. Hier nämlich brauchen x und y nicht unbedingt mit dem gleichen Faktor multipliziert werden (wir nennen die beiden Faktoren jetzt einfach f1 und f2). Dadurch werden aber die Proportionen der Kurve verändert, d.h. es kommt zu einer Stauchung (für 0<f1/2<1) oder Streckung (f1/2>1) in x- (für f1) bzw. y-Richtung (für f2). Jetzt erst werden die meisten Kurven wirklich zeichenbar. Ersetzen Sie doch einmal Zeile 220 des Quadratfunktions - Programmes durch:

```
220 Y = 0.01 * ((X-139)/1)^2 + 5
```

Sie werden strahlen, das war es. Hier werden Verzerrung und Verschiebung gleichzeitig angewandt, was zu diesem hübschen Ergebnis führt. Die einzelnen Veränderungswerte lauten: f1=1; f2=0.01; a=139; b=5 (statt mit 0.01 zu multiplizieren, könnten wir der Einfachheit halber auch durch 100 dividieren).

Doch es sind noch ein paar Dinge zu klären. Unsere Parabel

steht immer noch auf dem Kopf und zweitens werden bei der Veränderung der verschiedenen Parameter immer noch ILLEGAL QUANTITY ERRORs produziert. Zunächst zum Ersten:
Dieses Phänomen taucht auf, da unser Koordinatensystem auf dem Kopf steht. Unser Nullpunkt liegt nicht unten, sondern oben links in der Ecke. Die y-Werte werden nach unten hin nicht immer kleiner, sondern größer. Diese Eigenart kann durch einen einfachen Trick behoben werden. Wir drehen die Kurve einfach um, indem wir das Vorzeichen aller y-Werte umkehren (s.o.). Da wir dann aber oft nur negative Werte für y bekommen, verschieben wir die Kurve gleichfalls noch um einen bestimmten Betrag nach unten (eigentlich oben), indem wir z.B. 195 addieren. Das sähe dann so aus:

```
220 Y = - 0.01 * ((X-139)/1) + 195
```

Das Ergebnis bestätigt das oben Gesagte.
Doch nun zu den Fehlermeldungen, die inzwischen immer häufiger werden. Die Ursache liegt einfach in unserer unvollkommenen Bereichsüberprüfung. Zum einen testen wir gar nicht, ob y negativ wird, zum anderen beenden wir gleich unser Programm, wenn es zu einer Überschreitung gekommen ist. Wenn Sie das obige "Quadrat" - Programm ab der Zeile 210 wie folgt ändern, dann werden Sie für alle Einsetzungen von a,b,f1 und f2 und für jede Funktion zufrieden sein, sofern sie überhaupt in dem gewählten Bereich liegt:

```
210 F1 = 1 : F2 = 0.1 : A = 160 : B = 100
220 FOR X=0 TO 319
230 Y = - F2 * ((X-A)/F1)^2 + B
240 IF Y>199 OR Y<0 THEN NEXT X : GOTO 270
250 SYS PL,X,Y : REM PUNKT SETZEN
260 NEXT X
270 WAIT 198,255 : SYS OF : END
```

Bevor der Computer etwas zeichnet, kann er schon eine ganze Menge Werte durchlaufen haben. Warten Sie also etwas, bevor Sie das Programm abbrechen. In diesem Programm wurde der Nullpunkt unseres verschobenen Koordinatensystems nach 160,100, also in die Mitte des Bildschirms gebracht. Wir könnten dort also auch z.B. zur Veranschaulichung die Achsen einzeichnen.

Eine weitere Möglichkeit ist beispielsweise die Niederlegung der Funktion in einer sogenannten Funktionsdefinition. Dies ist eine besondere Eigenschaft des Basic und wird durch den Befehl DEF FN realisiert. Mit diesem Befehl wird eine Funktion definiert, die irgendwo im Programm beliebig aufgerufen werden kann, ohne sie umständlich nieder zu schreiben. Sie müßten dazu lediglich folgende Änderungen vornehmen:

```
215 DEF FN F(X)= - F2 * ((X-A)/F1)^2 + B
230 Y = FN F(X)
```

Oder wenn Sie lediglich die reine Funktion in der Definitionszeile stehen lassen wollen:

```
215 DEF FN F(X)= X^2
230 Y = - F2 * FN F((X-A)/F1) + B
```

Der jeweilige Wert für die verschiedenen Variablen wird dabei stets in die Formel eingesetzt. So können Sie irgendwo die Funktion verändern, ohne in die Schleife eingreifen oder diese suchen zu müssen.

Das folgende Programm geht noch etwas weiter. Es zeichnet für Sie eine beliebige Funktion. Sie können in eine INPUT - Abfrage Ihre Funktion eintippen und das Programm entwickelt für Sie daraus durch EinPOKEn entsprechender Bytes in einen freien Basicbereich die Funktionsanweisung. Sie brauchen diese relativ komplizierte Routine nicht unbedingt zu verstehen, Sie demonstriert Ihnen aber den guten Nutzen des DEF FN - Befehls. Die Funktion wird ohne Verzerrungs- oder Verschiebungsparameter, also in "reiner" Form eingegeben. Diese werden später in der Hauptschleife in der Weise hinzugefügt, wie in den zuletzt gezeigten zwei Zeilen. Doch hier das Programm.

Achten Sie bitte unbedingt darauf, in dem ersten Teil bis zum Ende der DEF FN - Zeile (also bis einschließlich Zeile 350) genau den Wortlaut mit REMs und mit genau der gleichen Leerzeichenanzahl abzuschreiben, da das Programm damit rechnet! Beachten Sie das nicht, so kommt es im Zweifelsfall

zum Absturz des Programms. Vor dem ersten Start sollten Sie das Programm daher auf jeden Fall zunächst einmal abspeichern!

```
100 REM *************************
110 REM **                       **
120 REM **  FUNKTIONENPLOTTER    **
130 REM **                       **
140 REM *************************
150 REM
160 REM
170 REM DEF FN - ROUTINE:
180 REM ******************
190 REM
200 RESTORE:DIM A$(25):PRINT CHR$(147)
210 PRINT "GEBEN SIE EINE BELIEBIGE FUNKTION EIN
:":PRINT:PRINT:PRINT
220 GOSUB 500 : REM EINGABEROUTINE
230 AD=2684-1:REM POKE-START
240 FOR X=1 TO 25:READ A$(X):NEXT X:REM DATEN EINLESEN
250 FORX=1TOLEN(A$):AD=AD+1
260 Q=X:FORY=1TO25:B$=A$(Y):FORZ=1TOLEN(B$):C$=MID$(A$,X,1)
270 IFC$=" "THENAD=AD-1:NEXTX:GOTO310
280 IFC$=MID$(B$,Z,1)THENX=X+1:NEXTZ:POKEAD,Y+169:X=X-1:NEXTX
:GOTO310
290 X=Q:NEXTY
300 POKEAD,ASC(C$):NEXTX
310 POKEAD+1,58:POKEAD+2,143
340 REM FREIHALTER:
350 DEFFNF(X)=0000000000000000000000000000000000000000000000000
0000000000000000000
400 REM 25 FUNKTIONEN:
410 DATA +,-,*,/,^,AND,OR,<,=,>,SGN,INT,ABS,USR,FRE,POS,SQR,R
ND,LOG,EXP
420 DATA COS,SIN,TAN,ATN,PEEK
430 GOTO 600 : REM WEITER
500 INPUT "F(X)=";A$ : RETURN : REM EINGABEROUTINE(ZUM
AUSBAU)
540 REM
550 REM      ****************
560 REM **** GRAPHIK-ROUTINEN ****
```

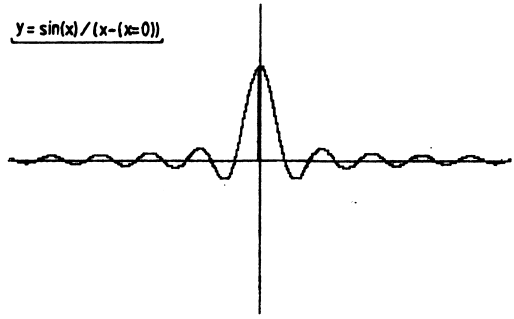
```
570 REM      ****************
580 REM
600 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
610 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
620 PC=51212 : PL=51215 :REM PCOLOR/PLOT
630 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
640 CL=51224 : GL=51227 :REM CLINE /GLOAD
650 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
655 POKE 2,0 : REM LOESCHFLAG
660 REM
670 REM       *************
680 REM  **** HAUPTPROGRAMM ****
690 REM       *************
700 REM
710 PRINT : PRINT "GEBEN SIE DIE FOLGENDEN PARAMETER EIN:" :
PRINT
720 INPUT "F1 (X-ZERRFAKTOR)  ";F1
730 INPUT "F2 (Y-ZERRFAKTOR)  ";F2
740 INPUT "A  (X-VERSCHIEBUNG)";A
750 INPUT "B  (Y-VERSCHIEBUNG)";B
1000 SYS IN : SYS SC,16*1+5 : REM INITIALISIEREN
1005 IF PEEK(2)=0 THEN SYS GC : REM GRAPHIK LOESCHEN
1007 POKE 2,0 : REM LOESCHFLAG
1010 REM
1020 REM ACHSEN ZEICHNEN:
1030 REM ****************
1040 IF A>=0 AND A<320 THEN SYS SL,A,0,A,199 : REM X-ACHSE
1050 IF B>=0 AND B<200 THEN SYS SL,0,B,319,B : REM Y-ACHSE
1060 REM
1070 REM ZEICHENROUTINE:
1080 REM ***************
1210 FL% = 1 : REM AUSSERHALB-FLAG
1220 FOR X=1 TO 319
1230 Y=-F2*FNF((X-A)/F1)+B
1240 IF Y<0 OR Y>199 THEN FL%=1:NEXT X:GOTO 1280
1250 IF FL%=0 THEN SYS SL,X1,Y1,X,Y
1260 FL%=0
1270 X1=X:Y1=Y:NEXT X : REM LETZTE KOORD. MERKEN
1280 POKE 198,0 : WAIT 198,255 : GET A$ : SYS OF : REM
GRAPHIK AUS
1290 REM *****  MENUE:  *****
```

```
1300 PRINT "WOLLEN SIE:" : PRINT : PRINT
1310 PRINT "(1) ANDERE PARAMETER"
1320 PRINT "(2) ANDERE FUNKTION"
1330 PRINT "(3) GRAPHIK NICHT LOESCHEN"
1340 PRINT "(4) GRAPHIK SPEICHERN"
1350 PRINT "(5) GRAPHIK LADEN"
1360 PRINT "(6) HARDCOPY"
1390 PRINT "(7) BEENDEN"
1450 WAIT 198,1 : GET A$
1460 ON VAL(A$) GOTO 700,1480,1490,1510,1600,1650,1500
1470 GOTO 1450 : REM FEHLEINGABE
1480 RUN : REM ANDERE FUNKTION
1490 POKE 2,1 : PRINT "OK" : PRINT : GOTO 1450 : REM FLAG
SETZEN
1500 END : REM BEENDEN
1510 REM
1520 REM GRAPHIK SPEICHERN:
1530 REM
1540 INPUT "FILENAME,GA";FI$,GA
1550 SYS GS,FI$,GA : REM SPEICHERN
1560 PRINT "OK" : GOTO 1450
1570 REM
1580 REM GRAPHIK LADEN:
1590 REM
1600 INPUT "FILENAME,GA";FI$,GA
1610 SYS GL,FI$,GA : REM LADEN
1620 SYS IN : SYS SC,16*1,5 : GOTO 1280
1630 REM
1640 REM HARDCOPY:
1645 REM
1650 PRINT "DRUCKER FERTIG MACHEN UND TASTE DRUECKEN"
1660 OPEN 1,4 : SYS HC,1 : CLOSE 1 : REM HARDCOPY
1670 PRINT "OK" : GOTO 1450
```

Beachten Sie bitte, daß auch dieses Programm für das Graphik-Paket aus # 4.7 geschrieben wurde und bei der Verwendung anderer Graphikroutinen erst auf die in der Einleitung (v. Kapitel 5) dargelegte Weise umgeschrieben werden muß.
Wie gesagt werden Sie zunächst aufgefordert, eine Funktion einzugeben. Das hierzu verwendete INPUT steht als Unter-

programm in Zeile 500 und kann so von Ihnen durch eine andere Eingabeschleife - z.B. mit GET - ersetzt werden. Die Verlegung nach Zeile 500 war notwendig, da Sie bis zur Zeile 350 ja keinerlei Änderungen vornehmen dürfen. Die Funktion steht dann in A$ und wird an die Übersetzerroutine bis Zeile 350 Übergeben. Diese formt daraus dann eine DEF FN - Zeile, d.h. Sie POKEt die notwendigen Bytes direkt in den Basic-speicher und verändert so die Zeile 350. Sie können sich ja einmal diese Zeile anschauen, nachdem Sie die erste Funktion eingegeben haben. Die Adresse, bei der die besagte Zeile beginnt, bzw. bei der das Programm beginnt, seine Funktion einzuschreiben, steht in Zeile 230 und kann von Programmierern geändert werden, die die Routine verstanden haben und in diesem Bereich Änderungen vornehmen wollen.
Grundsätzlich können Sie in Ihre Funktion jede der 25 Teilfunktionen aufnehmen, die Sie in den Zeilen 410/420 finden. Dabei ist die Syntax genau dieselbe, wie in Basic.
Nach den obligatorischen Sprungadressendefinitionen werden Sie aufgefordert, die beiden Verzerrfaktoren f1/f2 und die Verschiebesummanden a und b einzugeben (Z. 710-750). Hier können Sie das anwenden, was Sie in den obigen Ausführungen gelernt haben. Hier einige Richtwerte: f1 und f2 werden bei den meisten Funktionen größer als 1 (bei Winkelfunktionen ist z.B. der Bereich von 10-70 empfehlenswert). Wie Sie wissen ist f1 dafür zuständig, den Graphen nach links und rechts auseinander zu ziehen. f2 dagegen dehnt die Kurve nach oben und unten. Für einen ersten Überblick empfehlen sich für a und b die Werte a=160 und b=100, wodurch der Nullpunkt des Koordinatensystems genau in die Mitte des Bildschirms gebracht wird. Wollen Sie dann z.B. nur die für x positiven Bereiche, dann wählen Sie a=0 usw.
Nach dem Einschalten und eventuellen Löschen der Graphik (abhängig vom Löschflag) werden dann ab Zeile 1040 die Achsen des Koordinatensystems gemäß der gewählten Verschiebung gezeichnet. Wenn Sie wollen, können Sie hier noch Teilstriche als Einheiten entlang den Achsen zeichnen.
In Zeile 1210 beginnt nun die eigentliche Zeichenarbeit. Sie finden hier die altbekannte Routine vor, die wir oben entwickelt haben. Lediglich eine Kleinigit ist verändert. Um nicht nur die einzelnen Punkte der Kurve zu sehen, die berechnet wurden, werden diese hier durch Linien miteinander

verbunden. So entsteht ein schönes, angenähertes Bild. Man spricht hier auch von Aproximation. Die Schwierigkeit besteht darin, daß gerade dann nicht vom letzten Punkt zum gerade berechneten gezeichnet werden darf, wenn der Graph aus einem ungültigen Bereich kommt. Dafür sorgt das Flag FL% (Integervariable).
Ist die Kurve gezeichnet, so kommen Sie nach einem Tastendruck in das Hauptmenue. Hier können Sie verschiedene Optionen wählen. Sie können eine ganz neue Funktion wählen, nur die Parameter verändern, das alte Bild erhalten, während das neue gezeichnet wird (für Vergleiche besonders geeignet), Graphik speichern / laden, eine Hardcopy anfertigen oder einfach beenden.



Noch etwas zur Funktioneneingabe: Bitte achten Sie darauf, daß erstens die Syntax Ihrer Funktion richtig ist - andernfalls entsteht ein SYNTAX ERROR. Zweitens sollten Sie darauf achten, daß keine undefinierten Werte eingesetzt werden. Dies sind z.B.: allgemein zu hohe Werte, der Wert 0 in Nennern oder bei Logarithmus, negative Werte bei Wurzel oder Logarithmus.
Negative Werte können z.B. ausgeschlossen werden durch Verwendung der Absolut - Funktion, z.B.:

```
SQR(ABS(X))
```

Der Wert Null wird vermieden durch ersetzen von X durch den Term (X-(X=0)):

statt 1/X : 1/(X-(X=0)) oder
statt LOG(X): LOG(ABS(X-(X=0)))

Wird x gleich 0, so wird der Ausdruck X=0 im Commodore - Basic gleich -1, ansonsten bleibt er 0. Ähnlich können Größer- oder Kleinerzeichen (>,<) verwendet werden.
Ist ein Fehler aufgetaucht, so können Sie das Programm auch mit RUN 350 starten, wobei die Funktion erhalten bleibt.

5.1.2. 3-dimensionale Graphik

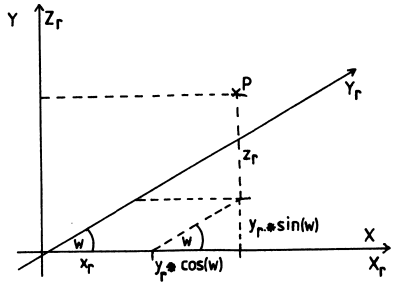
Kaum eine Computer-Werbeanzeige und zunehmend auch Anzeigen aus anderen Bereichen verzichten auf wunderschöne meist dreiimensionale Graphikbilder, um Ihre Produkte anzupreisen. Kein Wunder, die Zeichnungen strahlen eine enorme Ästhetik aus, durch die der Betrachter die unendlichen Weiten des Raumes mit dem jeweiligen Werbeobjekt assoziiert. Immer mehr Graphikdesigner besinnen sich auf das Hilfsmittel Computer und seine Fähigkeit auch komplizierte 3-dimensionale Funktionen und allgemein räumliche Bilder herzustellen. Wer hier nicht hinterher hinken will, der sollte sich einmal informieren.
Aber auch dem privaten Anwender bereitet es ungeheuren Spaß, sich mit diesen Dingen zu beschäftigen. Nicht umsonst ist meist eins der ersten Objekte des Informatikunterrichts in Schulen die Erzeugung von einfachen zweidimensionalen Funktionsgraphen (s. vorheriges Kapitel) bis hin zu den räumlichen Darstellungen.
Doch stößt Letzteres ohne Vorkenntnisse und ohne sachkundige Anleitung auf erhebliche Schwierigkeiten, da die Fachliteratur oft reichhaltiges Wissen voraussetzt und die eigene Ableitung der Algorithmen (Rechenvorschriften) relativ kompliziert ist. Aus diesem Grunde seinen hier einige Verfahren dargelegt, die sich auf die Kenntnis des in Abschnitt 5.1.1 Gesagten stützen.

5.1.2.1 Parallel-Projektion

Man unterscheidet grundsätzlich zwei Arten von Projektionen:

- parallele Projektion
- Zentralprojektion

Wir werden uns im folgenden mit der ersten einfachen Projektionsart beschäftigen. Weiter unten liefern wir auch das nötige Rüstzeug für die zweite, wirklichkeitsnahere.
Zunächst einmal müssen wir unser 2-dimensionales Koordinatensystem erweitern. Hierzu verwenden wir die Achsen xr,yr und zr (r steht für räumlich). Jeder Punkt eines 3-D-Objektes (z.B. ein Haus) hat damit drei Koordinaten (xr,yr,zr). Dieses Koordinatentripel muß zur Darstellung auf dem Bildschirm (denn hier kennen wir ja nur die Koordinaten x und y) in die ebenen Koordinaten x und y umgerechnet werden.
Bevor wir dies unternehmen, müssen wir uns erst einmal über das Aussehen unseres 3-D-Koordinatensystems einigen. Wir legen dafür hiermit folgendes fest (s.Abbildung):
Die xr- und zr-Achsen liegen in der Zeichenebene und stehen senkrecht aufeinander. Die durch die Zeichenebene gehende yr-Achse bildet mit der xr-Achse den Winkel w:



Wie aus der Zeichnung ersichtlich, ergeben sich damit folgende Formeln, die die Umrechnung der Raum- auf die Ebenenkoordianten vornehmen:

```
x = xr + yr*cos(w)
y = zr + yr*sin(w)
```

Der Winkel w bestimmt dabei die Perspektive, also den Winkel,

mit dem man das Objekt betrachtet. Wollen wir das Bild verschieben, so daß wir es z.B. mitten im Graphikfenster zu sehen bekommen oder das Objekt in weite Ferne zu rücken, können wir drei Summanden a,b und c zu der xr-,zr- und yr-Koordinate hinzuaddieren (s. # 5.1). Wir erhalten dann:

```
x = a+xr + (yr+c)*cos(w)
y = b+zr + (yr+c)*sin(w)
```

Wollen Sie die Figur lediglich in der Ebene verschieben, so wählen Sie c=0. Doch auch hier wird es oft dazu kommen, daß Teile des Bildes nicht zu sehen sind, da sie entweder aus dem Bildschirm herausragen oder viel zu klein sind, als daß die Auflösung ausreichte, sie zu erkennen. Aus diesem Grunde führen wir wieder unsere bekannten Verzerrfaktoren ein, die uns erlauben sollen, das Objekt einfach zu vergrößern oder Länge, Höhe und Breite zu verändern. Um alle drei Faktoren zu verändern, müssen wir wieder direkt die Raumkoordinaten mit den drei Faktoren f1,f2 und f3 für die xr-, zr- und yr-Koordinaten:

```
x = f1*(a+xr) + f3*(yr+c)*cos(w)
y = f2*(b+zr) + f3*(yr+c)*sin(w)
```

Wollen Sie die Figur nur vergrößern, ohne Sie zu verzerren, so wählen Sie f1=f2=f3. Für die drei Faktoren gilt wieder das in Abschnitt 5.1 Gesagte:

```
    f1/2/3 > 1 => Streckung (Vergrößerung)
0 < f1/2/3 < 1 => Stauchung (Verkleinerung)
```

Um die jeweilige Figur nun noch genau auf den Bildschirm zu bekommen, da der Nullpunkt der Raumkoordinaten nicht unbedingt immer mit dem Nullpunkt des Bildschirms übereinstimmen soll, der bekanntlich oben links in der Ecke liegt, wollen wir noch zwei Verschiebesummanden v1 und v2 einführen. v1 verschiebt das planare Koordinatensystem in +x-, v2 in -y-Richtung. Wollen Sie den Nullpunkt des Koordinatensystems beispielsweise genau in die Mitte des Bildschirms positionieren, so wählen Sie v1=160 und v2=100. Auch hier stehen unsere y-Ebenen-Koordinaten wieder auf dem Kopf, wir

müssen also ihr Vorzeichen ändern. Damit ergeben sich die beiden folgenden Formeln:

```
x =  f1*(a+xr) + f3*(yr+c)*cos(w) + v1
y = -f2*(b+zr) - f3*(yr+c)*sin(w) + v2
```

In dem folgenden Programm wird ein Haus räumlich dargestellt. Die dazu notwendigen Eckpunkte sind mit Ihren 3 räumlichen Koordinaten in DATA-Zeilen niedergelegt (Zeilen 1110-1140). Die 3-D-Koordinaten werden in die der Ebene umgerechnet und die einzelnen Punkte miteinander verbunden. Welche Punkte jeweils verbunden werden sollen, steht ebenfalls in dahinter liegenden DATA-Zeilen. Die Punkte werden der Reihe nach von 1 ab durchnumeriert und jeweils die Nummern zweier zu verbindender Punkte hintereinander in den Linien - DATA - Zeilen aufgeführt. Die Reihenfolge, in der die Linien gezeichnet werden, ist völlig egal, sie brauchen auch nicht zusammen zu hängen. Jeweils zu Anfang der Punkt- und der Linien-Daten steht die Anzahl der Punkte (Zeile 1100) und der Linien (Zeile 2100), die gezeichnet werden sollen. Wenn Sie sich an diese Datenform halten, können Sie sich beliebige eigene Figuren ausdenken und räumlich darstellen. Beispielsweise können Sie doch einmal Ihren Computer vermessen und die Daten in das Programm eingeben. Sie erhalten dann ein schönes räumliches Bild Ihres Gerätes.
Doch wofür sich die Umstände machen und die Koordinaten 3-dimensional eingeben? Dies hat einen besonderen Grund. Sie können Ihre einmal so erstellte Figur, wie in unserem Fall das Haus, beliebig vergrößern, verzerren, perspektivisch drehen oder verschieben. Dies erreichen Sie durch Veränderung der einzelnen Parameter a,b,c, f1,f2,f3, v1,v2 und w. Es wird eine Weile dauern, bis Sie die Auswirkungen dieser vielen verschiedenen Variablen überblicken. Bedenken Sie, daß die Zerrfaktoren f1,f2 und f3 die Einheiten der Achsen bestimmen, d.h. bei ihrer Veränderung scheinen sich die Objekte gleichfalls fort oder heran zu bewegen. Stört Sie das, so ändern Sie die einfach die obigen Formeln in jene:

```
x =  f1*a+xr + f3*yr*cos(w)+c + v1
y = -f2*b-zr - f3*yr*sin(w)-c + v2
```

Nun besitzen f1, f2 und f3 keinen Einfluß mehr auf den Abstand, obwohl diese Formeln mathematisch nicht ganz einwandfrei sind. Die Verzerrung bezieht sich nur auf das Objekt, nicht auf das Koordinatensystem. Wollten Sie also Einheiten an den Achsen abtragen, käme es zu Unstimmigkeiten. Eine sicher interessante Sache ist in den Zeilen 600-620 niedergelegt. Wir haben an dieser Stelle gerade die Raum-x- und die Raum-z- Achse gezeichnet. Um nun auch die in die Zeichenebene hineinragende y-Achse zu zeichnen, berechnen wir einfach den Punkt, bei dem die y-Ebenen - Koordinate 0 (bzw. 199) ist (unter Berücksichtigung des Winkels w) und zeichnen eine Linie vom Ursprung dorthin.

Wichtig bei allen Zeichenvorgängen ist die vorherige Überprüfung der Werte auf Bereichsüberschreitungen. Ein sehr gutes, professionelles Zeichenprogramm würde, falls ein oder zwei Eckpunkte außerhalb des Bildschirms liegen, trotzdem aber noch Teile der Linie zu sehen wären, die Schnittpunkte der Linie mit den Fensterränder berechnen und den sichtbaren Teil der Linie zeichnen. Unser einfaches Programm zeichnet diese Linien gar nicht. Sie könnten da ja Abhilfe schaffen.

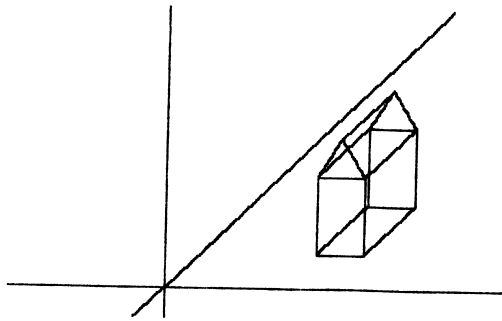
Gleichfalls werden die Linien, die eigentlich verdeckt sein sollten, ebenfalls gezeichnet. Dieses Problem der verdeckten Linien gehört zu den schwierigsten und langwierigsten Kapiteln der 3-D-Graphik. Große Institute tüfteln mit teilweise großem mathematischen Aufwand an solchen Algorithmen. Weiter unten wird Ihnen ein äußerst einfacher vorgestellt, der insbesondere bei 3-dimensionalen Funktionen Verwendung findet.

Vielleicht verändern Sie dieses Programm noch weiter. Sie könnten z.B. die jetzt in DATA-Zeilen abgelegten Daten in irgendeiner Form auf Diskette abspeichern und gegebenenfalls später wieder einladen. (DATA-Zeilen wären hierfür natürlich ungeeignet. Sie könnten alle Daten in Arrays unterbringen, die als sequentielles File auf Diskette gebracht werden können) Oder Sie bauen das Programm zu einem richtigen menuegesteuerten Zeichenprogramm aus; als kleines Beispiel dient Ihnen dazu schon das Programm aus Paragraph 5.1.

```
100 REM     ********************
110 REM     **                  **
120 REM     **  3-D-DESIGNER    **
130 REM     **                  **
140 REM     ********************
150 REM
200 REM
210 REM       ****************
220 REM **** GRAPHIK-ROUTINEN ****
230 REM       ****************
240 REM
250 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
260 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
270 PC=51212 : PL=51215 :REM PCOLOR/PLOT
280 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
290 CL=51224 : GL=51227 :REM CLINE /GLOAD
300 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
400 REM
410 REM          *********
420 REM    ****  PARAMETER  ***
430 REM          *********
440 REM
450 F1 = 5 : F2 = 5 : F3 = 3 : REM ZERRUNG
460 AR = 15 : BR =  0 : CR = 10 : REM VERSCHIEBUNG DER
RAUMKOORD.
470 W = 3.1415/4: REM SICHTWINKEL (IN RAD)
480 SI = SIN(W) : CO = COS(W) : REM KONSTANTEN (NICHT
VERAENDERN)
490 V1 = 100 : V2 = 180 : REM VERSCHIEBUNG DER EBENENKOORD.
500 REM
510 REM          **********
520 REM    ****  UMRECHNUNG  ***
530 REM          **********
540 REM
550 SYS IN : SYS GC : SYS SC,16*7+8 : REM GRAPHIK INIT
560 FG=0:IF V1<0 OR V1>319 THEN FG=1:GOTO 580 : REM FLAG
570 SYS SL,V1,0,V1,199 : REM RAUM-Z-ACHSE
580 IF V2<0 OR V2>200 THEN FG=1:GOTO 600
590 SYS SL,0,V2,319,V2 : REM RAUM-X-ACHSE
600 Z2 = V1 - (199-V2)/SI*CO : Z1 = V1 + V2/SI*CO
610 IF FG=0 AND Z1>=0 AND Z1<320 THEN SYS SL,V1,V2,Z1,0: REM
```

```
RAUM-Y-ACHSE OBEN
620 IF FG=0 AND Z2>=0 AND Z2<320 THEN SYS SL,V1,V2,Z2,199:
REM RAUM-Y-ACHSE UNTEN
630 READ AP : DIM X%(AP),Y%(AP) : REM ANZAHL PUNKTE
640 FOR ZA=1 TO AP : REM PUNKTEZAHL
650 READ XR,ZR,YR
660 X%(ZA) =  F1*(XR+AR) + F3*(YR+CR)*CO+V1
670 Y%(ZA) = -F2*(ZR+BR) - F3*(YR+CR)*SI+V2
680 NEXT ZA : REM NAECHSTER PUNKT
700 REM
710 REM          *******
720 REM    ****  STRICHE  ***
730 REM          *******
740 REM
750 READ AL : REM ANZAHL LINIEN
760 FOR ZA=1 TO AL
770 READ P1,P2 : REM PUNKTNUMMERN LESEN
775 IF X%(P1)<0 OR Y%(P1)<0 OR X%(P2)<0 OR Y%(P2)<0 THEN
790:REM AUSSERHALB
777 IF X%(P1)>319 OR Y%(P1)>199 OR X%(P2)>319 OR
Y%(P2)>199THEN790:REMAUSSERHALB
780 SYS SL,X%(P1),Y%(P1),X%(P2),Y%(P2) : REM VERBINDEN
790 NEXT ZA : REM NAECHSTE LINIE
900 POKE 198,0 : WAIT 198,255 : GET A$
910 SYS OF : LIST : REM GRAPHIK AUS
1000 REM
1010 REM       ***********
1020 REM ****  KOORDINATEN  ****
1030 REM       ***********
1100 DATA 10 : REM ANZAHL PUNKTE
1110 DATA  0, 0, 0,   6, 0, 0,   6,10, 0
1120 DATA  0,10, 0,   3,15, 0,   3,15,15
1130 DATA  6,10,15,   6, 0,15,   0, 0,15
1140 DATA  0,10,15
2000 REM
2010 REM       ************
2020 REM ****  VERBINDUNGEN  ****
2030 REM       ************
2100 DATA 17 : REM ANZAHL LINIEN
2110 DATA  1, 2,   2, 3,   3, 4,   4, 1
2120 DATA  4, 5,   5, 3,   5, 6,   6, 7
```

```
2130 DATA  7, 3,   7, 8,   8, 2,   8, 9
2140 DATA  9, 1,   9,10,  10, 4,  10, 6
2150 DATA 10, 7
```



Wenn Sie dieses Programm eingetippt und ausprobiert haben, werden Sie sehen, welchen Spaß es macht, mit ihm zu arbeiten. Speichern Sie ruhig einmal gelungene Bilder ab, oder erstellen Sie eine Hardcopy auf Ihrem Drucker.

Wollen wir die einzelnen Objekte zusätzlich im Raum drehen (nicht nur die Perspektive ändern), so müssen wir aus den ungedrehten Raumkoordinaten xr,yr,zr erst die gedrehten xr',yr',zr' errechnen, bevor wir sie in Ebenenkoordinaten umrechnen können. Dies geschieht mit Hilfe der folgenden Formeln:

Drehung um die xr-Achse:

```
xr' =  xr
yr' =  yr*cos(u) + zr*sin(u)
zr' = -yr*sin(u) + zr*cos(u)
```

Drehung um die yr-Achse:

```
xr' =  xr*cos(t) + zr*sin(t)
yr' =  yr
zr' = -xr*sin(t) + zr*cos(t)
```

Drehung um die zr-Achse:

```
xr' =  xr*cos(s) + yr*sin(s)
yr' = -xr*sin(s) + yr*cos(s)
zr' =  zr
```

dabei bedeuten:
u,s,t : Drehwinkel um die jeweiligen Achsen

Wollen Sie um zwei Achsen drehen, dann müssen diese Drehungen nacheinander ausgeführt werden: Sie berechnen erst die gedrehten Koordinaten für die Drehung um die erste Achse, dann setzen Sie die erhaltenen Werte in die Gleichungen ein, die für die Drehung um die zweite Achse zuständig sind. Entsprechendes gilt bei der Drehung um alle Achsen.
Die Erstellung dreidimensionaler Bilder findet in vielen Bereichen Anwendung. Da mit dem Computer auf die besprochene Art und Weise besonders gut reale Gegenstände oder Situationen maßstabsgetreu dargestellt und verändert werden können, eignen sich diese Techniken für die Konstruktion bzw. den Entwurf bestimmter Objekte wie Gebäude, Einrichtungen, Maschinen oder Maschinenteile. Dieses computerunterstützte Entwerfen (auch CAD = Computer Aided Design genannt), ist in vielen Teilen der Indrustrie und Ingenieurtätigkeit zu einem nicht mehr weg zu denkenden Werkzeug geworden. Autos, Flugzeuge werden konstruiert, technische Zeichnungen angefertigt oder Motoren etc. auf den Bildschirm und dann auf Papier gebracht. In diesem Zusammenhang eignet sich die 3-dimensionale Graphik hervorragend zur Simulation auch komplizierter Vorgänge. Diese Möglichkeit des Computers ist schon relativ früh entdeckt worden, zu einer Zeit, in der ein einfacher Taschenrechner noch ein Vermögen kostete, kompliziertere Geräte aber Unsummen verschlangen, was wohl ein Zeichen für die Wichtigkeit solcher Techniken ist.

5.1.2.2 Zentral-Projektion

Wenn wir Gegenstände betrachten, so können wir Ihre Entfernung unter anderem dadurch feststellen, daß sie, je entfernter sie stehen, desto kleiner sie werden. Das klassische Beispiel hierzu sind die langen Eisenbahnschienen, die immer schmaler werden und sich weit weg mit dem Horizont treffen. Tatsächlich ist es so, daß alle Linien sich auf einen virtuellen Punkt zuzubewegen scheinen, den

sogenannten Fluchtpunkt. Das Problem ist es nun, ein mathematisches Verfahren zu entwickeln, um unsere bisher parallelen Linien auf einen Punkt zu richten. Da die Ableitung der entsprechenden Formeln hier zu lange dauern würde, seien Sie hier mit Scalierung (fl/f2/f3) und Verschiebung (a/b/c) angegeben. Dabei stehen nun alle Achsen aufeinander senkrecht und die z-Achse ragt in die Bildebene hinein (eben war es die y-Achse!):

```
        x = (fl*xr+a)/q
        y = (f2*yr+b)/q
mit:  q = 1-(f3*zr+c)/fpz
```

dabei bedeuten nun:

```
xr,yr,zr: räumliche Koordinaten
fl,f2,f3: Verzerrung in xr-,yr-,zr-Richtung
 a, b, c: Verschiebung in xr-,yr-,zr-Richtung
fpz     : Entfernung (z-Koord.) des Fluchtpunktes
```

Sie müssen also erst q errechnen, um die beiden gesuchten Ebenen - Koordinaten zu errechnen. Wollen Sie gleichzeitig Ihr Objekt noch in alle drei Richtungen drehen, so wird die Sache sehr viel komplizierter:

```
        x = (fl*(A*xr + D*yr + G*zr)+a) / q
        y = (f2*(B*xr + E*yr + H*zr)+b) / q
mit:  q = 1-(f3*(C*xr + F*yr + I*zr)+c) / fpz
```

Das sieht schon kompliziert genug aus. Wenn Sie aber sehen, was Sie für A,B,C,...I einsetzen sollen, werden Ihnen die Augen ausgehen:

```
A =  cos(s)*cos(t)
B =  sin(s)*cos(t)
C = -sin(t)
D = -sin(s)*cos(u) + cos(s)*sin(t)*sin(u)
E =  cos(s)*cos(u) + sin(s)*sin(t)*sin(u)
F =  cos(t)*sin(u)
G =  sin(s)*sin(u) + cos(s)*sin(t)*cos(u)
H = -cos(s)*sin(u) + sin(s)*sin(t)*cos(u)
I =  cos(t)*cos(u)
```

dabei bedeuten:

s: Winkel der Rotation um die z-Achse
t: Winkel der Rotation um die y-Achse
u: Winkel der Rotation um die x-Achse

Solche Mammutaufgaben können selbstverständlich nur in Maschinensprache in relativ angemessener Zeit bewältigt werden. Selbst da gibt es einige Zeitprobleme (ein Tip am Rande: Die beste Möglichkeit wäre das Anlegen einer Tabelle von Sinus- und Cosinuswerten (jeweils in einem bestimmten Winkelabstand (z.B.1 Grad etc.)), in der das Programm bei Bedarf immer nachschlagen kann, ohne die Werte erst lange errechnen zu müssen).

5.1.2.3 3-D-Funktionen

Eine sehr reizvolle Anwendung der 3-D-Technik ist die Darstellung dreidimensionaler Funktionen. Doch nicht nur dem bloßen Vergnügen des Erstellers oder einiger Mathematiker dient diese weitere Möglichkeit. Auf diese Weise sind komliziertere Zusammenhänge, die sonst aus vielen unübersichtlichen Tabellen erahnt werden müssen, äußerst plastisch und informativ darstellbar. Sie kennen den Nutzen von zwei - dimensionalen Diagrammen, die beispielsweise die Entwicklung des jährlichen Umsatzes im Laufe der Jahre wiederspiegeln. Wollte man nun etwa gleichzeitig noch die Abhängigkeit des Umsatzes vom Preis eines bestimmten Produktes in den verschiedenen Jahren darstellen, so müßte man für jedes Jahr ein solches Diagramm erstellen, was auf die Dauer zu einer unübersehbaren Schar von Kurven führte. Doch dieser komlexe Zusammenhang kann anschaulich in einem einzigen 3-D-Diagramm vermittelt werden. So erhalten Sie einen schnellen und guten Überblick über die Dinge und können so gemäß den Trends den optimalen Preis Ihres Produktes im nächsten Jahr (Monat) ermitteln. Ich brauche Ihnen nicht erst zu sagen, welche Vorteile Ihnen dadurch erwachsen.
Hier soll Ihnen die Technik der Erstellung anhand von

mathematischen Funktionen vermittelt werden. In diesem Fall werden die einzelnen Werte, die notwendig sind, um das Bild zu erstellen, errechnet. Sie können diese selbstverständlich auch aus irgendwelchen Tabellen ermitteln, was den Anwendungsbereich der jetzt beschriebenen Vorgänge enorm erweitert.
Zu dem Graphen einer 3-D-Funktion kommen wir durch Verwendung einer sogenannten Wertematrix. Als Beispiel soll uns die Funktion

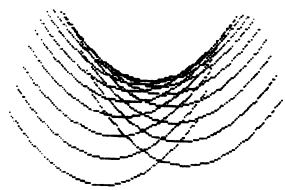
$z = x^2 + y^2$

dienen. Sie erinnern sich aus Abschnitt 5.1.1, daß dort die Funktionswerte f(x) in einer FOR ..NEXT - Schleife ermittelt wurden, in der die Variable x (die sogenannte Laufvariable) stets aufgezählt und die Variable y errechnet wurde. So erhielten wir beliebig viele Wertepaare, die uns als Koordinaten für den Graphen unserer Funktion dienten.
Bei drei-dimensionalen Funktionen -f(x,y)- dagegen besitzen wir zwei Laufvariablen (x und y), mit denen wir die dritte (z) berechnen müssen. Um diesem Umstand gerecht zu werden, verwenden wir zwei ineinander geschachtelte FOR...NEXT - Schleifen. In der ersten wird der x-Wert, in der zweiten der y-Wert hochgezählt. Das folgende Programm bringt Sie diesem Sachverhalt näher:

```
100 REM ************************
110 REM **                    **
120 REM **  3-D: Z=Y↑2-X↑2    **
130 REM **                    **
140 REM ************************
150 REM
230 REM **** GRAPHIK-ROUTINEN ****
240 REM      ****************
250 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
260 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
270 PC=51212 : PL=51215 :REM PCOLOR/PLOT
280 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
290 CL=51224 : GL=51227 :REM CLINE /GLOAD
300 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
320 REM
```

```
330 REM **** PARAMETER ****
340 REM      *********
400 W = 3.1415/8
410 A = 0 : B = 0 : C = 0
420 F1 = 20 : F2 = 5 : F3 = 8
430 V1 = 160 : V2 = 100
440 CO = COS(W) : SI = SIN(W)
520 REM
530 REM **** ZEICHNEN ****
540 REM      ********
600 SYS IN : SYS GC : SYS SC,16*1+4 : REM GRAPHIK INIT
610 FOR YR=3 TO -4 STEP -0.5
620 FOR XR=3 TO -3 STEP -0.05
630 ZR=YR*YR-XR*XR : REM FUNKTION
640 X=F1*(A+XR) + F3*(YR+C)*CO + V1
650 Y=F2*(B+ZR) + F3*(YR+C)*SI + V2
660 SYS PL,X,Y: REM PUNKT ZEICHNEN
670 NEXT XR,YR
1000 POKE 198,0:WAIT 198,255:SYS OF : REM GRAPHIK AUS
```

Die besagten FOR...NEXT-Schleifen überstrecken, wie Sie vielleicht sehen, die Zeilen 610 bis 670. In 630 wird dann aus den beiden Laufvariablen der zr-Wert berechnet, um dann diese Raumkoordinaten auf altbekannte Weise in Ebenenkoordinaten umzurechnen. Das einzige Problem ist hier, wie immer, die Wahl der Parameter und des xr-, yr-Wertebereichs. Bei 3-D-Funktionen tritt es besonders auf, da hier so viele Parameter zu bestimmen sind. Man hilft sich, indem man zunächst möglichst einfache Zahlen einsetzt (etwa: a,b,c=0, f1,2,3=1, w=3.1415/4, v1=160, v2=100) und dann das entstehende Bild entsprechend verändert.



Vernetzung:

Durch einen kleinen Trick können wir nun einen ganz besonders schönen Effekt mit hineinbringen. Bislang erhalten wir nur einzelne Kurven, die uns schrittweise den Verlauf in die dritte Dimension vermitteln. Oft sieht man aber quasi gebogene Gitternetze, die uns die Graphik noch ein ganzes Stück plastischer erscheinen lassen (Vernetzung oder cross-hatching). Dies wird einfach durch eine scheinbare Drehung der Kurve erreicht. Scheinbar deshalb, weil es sich eigentlich gar nicht um eine Drehung handelt, sondern nur eine Veränderung der Schrittweiten, in denen die xr-, bzw. yr-Werte vom Start- zum Endwert gelangen, also derjenigen Werte, die hinter den STEP-Komandos der beiden FOR...NEXT - Schleifen erscheinen.

Im obigen Beispiel wurde yr mit einer Schrittweite von -0.5, xr dagegen mit -0.05 aufgezählt. Dadurch erschien nicht etwa eine gleichmäßige Fläche, was passieren würde, wenn wir beide Schrittweiten gleich groß wählten, sondern jenes bekannte "Streifenmuster".

Wenn wir jetzt die Schrittweiten austauschen, so verläuft unser Streifenmuster genau senkrecht zum ersten. Das wird im nächsten Programm ausgenutzt:

```
100 REM ************************
110 REM **                    **
120 REM **  3-D: Z=Y↑2-X↑2    **
130 REM **     VERNETZUNG     **
140 REM ************************
150 REM
230 REM **** GRAPHIK-ROUTINEN ****
240 REM       ****************
250 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
260 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
270 PC=51212 : PL=51215 :REM PCOLOR/PLOT
280 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
290 CL=51224 : GL=51227 :REM CLINE /GLOAD
300 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
320 REM
330 REM **** PARAMETER ****
340 REM       *********
400 W = 3.1415/8
```

```
410 A = 0 : B = 0 : C = 0
420 F1 = 20 : F2 = 5 : F3 = 8
430 V1 = 160 : V2 = 100
440 CO = COS(W) : SI = SIN(W)
520 REM
530 REM **** ZEICHNEN ****
540 REM      ********
600 SYS IN : SYS GC : SYS SC,16*1+4 : REM GRAPHIK INIT
610 SY=-0.5 : SX=-0.03 : REM SCHRITTWEITEN (STEPS)
620 FOR ZA=1 TO 2 : REM ZAEHLER
630 FOR YR=3 TO -4 STEP SY
640 FOR XR=3 TO -3 STEP SX
650 ZR=YR*YR-XR*XR : REM FUNKTION
660 X=F1*(A+XR) + F3*(YR+C)*CO + V1
670 Y=F2*(B+ZR) + F3*(YR+C)*SI + V2
680 SYS PL,X,Y: REM PUNKT ZEICHNEN
690 NEXT XR,YR
700 IF ZA=1 THEN GOSUB 1100 : SYS GC : SY=-0.04 : SX=-0.5
710 NEXT ZA
720 FOR T=8192 TO 8192+8000:POKET,PEEK(T)ORPEEK(T+8192):NEXTT
:REM OREN
1000 POKE 198,0:WAIT 198,255:SYS OF:END: REM GRAPHIK AUS
1100 FOR T=8192 TO 8192+8000:POKET+8192,PEEK(T):NEXTT : REM
UEBERTRAGEN
1110 RETURN
```

Sie sehen, der Graph wird insgesamt zweimal auf zwei verschiedene Weisen gezeichnet. Nach dem ersten Mal wird die gesamte Graphik (bei $2000 = 8192) nach $4000 (8192+8192 = 16384) übertragen bzw. zwischengespeichert. Hernach zeichnen wir mit vertauschten STEP - Parametern. Zu guter Letzt werden die beiden Graphiken miteinander durch OR verknüpft, was zu einer Überlagerung führt. Haben Sie Geduld, der Prozess der Zwischenspeicherung und Überlagerung dauert recht lange, schalten Sie also nicht ab!
In unserem speziellen Beispiel wäre das Zwischenspeichern und nachherige ORen nicht nötig, wir könnten also direkt das zweite Bild über das erste zeichnen. Doch für unseren nächsten Schritt ist dieses Verfahren unabdingbar.
Vielleicht bringen Sie noch ein 3-D-Koordinatensystem in das Bild, wie es Ihnen das vorherige Programm zeigt.

Versteckte Linien:

Im diesem Teil werden wir ein einfaches Verfahren für das Auslöschen eigentlich von vorderen Flächen verdeckter Linien beim Zeichnen mathematischer und auch anderer Funktionen vorstellen.

Im täglichen Leben können wir normalerweise nicht durch die Dinge schauen, die wir betrachten. Wie Sie jedoch beim Zeichnen unserer Funktion sehen, ist dies hier der Fall. Unsere Linien durchdringen sich, obwohl die vorderen, die ja eine Fläche abstecken, die hinteren verdecken müßten.

Bei Funktionen gibt es nun unter anderem zwei Methoden, diese verdeckten Linien zu unterdrücken, bzw. nachträglich wieder zu löschen. Fangen wir bei der ersten an. Sie ist äußerst simpel, aber sehr effektiv und trickreich:

Beim Zeichnen unseres Graphen achten wir darauf, daß wir stets von hinten nach vorne, also mit abnehmendem yr zeichnen. Haben wir jetzt einen Punkt ganz normal ausgerechnet und auf den Bildschirm gesetzt, dann löschen wir einfach unter dem Punkt in einer Linie alles bis zum unteren Ende des Graphikfensters. Damit verdeckt jede neu gezeichnete Ebene alles hintere, vorausgesetzt, es liegt räumlich unter dem gezeichneten Punkt. So erhalten wir eine Aufsicht auf die graphische Struktur der Funktion. Wenn Sie in das obige Programm lediglich eine einzige Zeile hinzufügen, wird Ihnen der Effekt deutlich:

```
685 SYS CL,X,Y+1,X,199 : REM UNTER PUNKT LOESCHEN
```

Das einzige Problem sind eventuell die Randbereiche. Hier könnte man eigentlich die Sicht quasi von unten auf die Kurve zulassen. Doch mit unserem Algorithmus ist dies nicht möglich - wie Sie sehen, wenn Sie das Programm laufen lassen -, da auch die "Rückansicht" gelöscht wird. Um dieses Manko auszugleichen könnte man nun nicht bis zum unteren Bildrand, sondern lediglich zum darunterliegenden Punkt der nächsten Linie löschen. Damit wäre das Problem erledigt. Das Zeichnen würde mit dieser Methode jedoch um Einiges länger dauern. Vielleicht versuchen Sie einmal das obige Programm so umzuschreiben. Es gibt einige schöne Funktionen, die Sie ausprobieren sollten. Versuchen Sie es evt. 'mal mit jenen:

$z = x^2+y^2$

$z = 1/(1+x^2+y^2)$

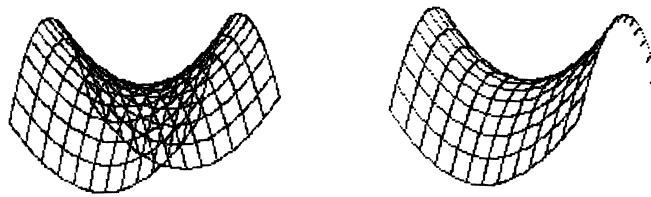
$z = SQR(1-x^2/4-y^2/9)$

$z = sin(x)/x+sin(y)/y$

$z = sin(1/x)/x+sin(1/y)/y$

Sie können diese Funktionen auch mischen, da der Teil, der x enthält den Verlauf in x-Richtung wiederspiegelt, der Teil mit y den in y-Richtung. Man käme dann beispielsweise auf so etwas (Mischung: letzte und vorletzte:):

$z = sin(x)/x+sin(1/y)/y$



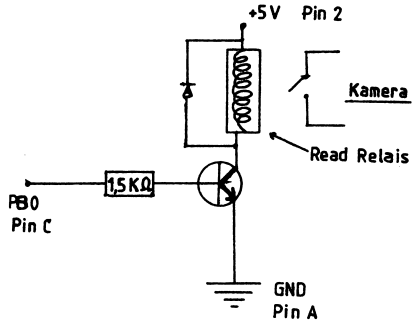
5.1.2.4. Bewegte Bilder in 3-D

In den letzten Paragraphen haben Sie die Methoden kennengelernt, die zur Erzeugung 3-dimensionaler Bilder notwendig sind. Wenn es nicht gerade kleine Objekte waren, dauerte die Erstellung recht lange. Wie sollten daraus laufende Vorgänge entstehen, die von uns als bewegt angesehen werden können? Nun es gibt im Prinzip hierbei zwei Möglichkeiten. Die erste ist relativ einfach: Sie erstellen ein Bild unsichtbar in einem im Moment nicht angezeigten Graphikbereich während Sie den Inhalt eines anderen Graphikbereiches auf dem Bildschirm darstellen. Ist das Bild vollendet, so schalten Sie einfach auf die neue Graphikseite um (s. # 3.3.2). Die nun unsichtbar gewordene erste Seite kann dann zur Erstellung des nächsten Bildes herangezogen werden usw.
Zwar ereichen Sie damit keine besonders schnell bewegten

Bilder, doch ist für uns trotzdem ein gewisser Bewegungseffekt vorhanden. Sehr schön sind Rotationen oder schrittweise Vergrößerungen der Objekte.

Die zweite Methode ist ein wenig aufwendiger. Sie benötigen dafür eine Filmkamera, die mit Einzelbildschaltung und einem Fernauslöser ausgerüstet ist. Rein theoretisch ist es auch möglich, eine Kamera ohne eine solche Einrichtung zu verwenden. Sie müssen dann zum Filmen einer Situation extrem kurz auf den Auslöser drücken, um möglichst wenige Bilder auf einmal zu knipsen. Mit etwas Übung erhalten Sie dann ebenfalls recht schöne Ergebnisse. Sind diese Voraussetzungen geschaffen, dann können Sie Ihre eigenen Computergraphikfilme produzieren. Was sonst nur im Kino oder Fernsehen in Science fiction - Filmen zu sehen ist, das bekommen Sie nun hautnah ins Haus.
Sie brauchen Ihre Kamera lediglich auf den Fernseher zu fixieren (am besten mit Stativ), die Einzelbildschaltung einzustellen und loszulegen: Sie programmieren dem Rechner eine Folge von Bildern ein. Die Rechen- bzw. Erstellungszeit ist dabei egal. Beispielsweise drehen Sie Ihr Objekt (Kurven oder Gegenstände) mit jedem Male ein Stück um eine Achse. Jedesmal, wenn ein Bild fertig gezeichnet ist, dann drücken Sie zum Festhalten eines Bildes einmal auf den Auslöser. Dies muß in sehr kleinen Schritten erfolgen, damit der Vorgang nachher auf dem Film nicht zu schnell erfolgt. Im Zweifelsfall photographieren Sie jede Szene mehrmals.
Eine noch elegantere Methode ist die Steuerung der Kamera durch den Computer. Dies ist dann möglich, wenn Ihre Kamera einen elektischen Fernauslöser besitzt. Haben Sie nur einen Drahtauslöser, dann gibt es entsprechende Adapter, die im Fachgeschäft erhältlich sind. Sie können dann Ihren Computer alleine laufen lassen, ohne selbst jedesmal die Kamera betätigen zu müssen. Gleichfalls können Sie, wenn Sie keine Einzelbildschaltung besitzen, extrem kurze Schaltzeiten erreichen und so nur wenige, oder gar nur ein Bild auf einmal filmen.
Zur Verwirklichung dieses Vorhabens müssen wir einen kleinen Abstecher in die Elektronik machen. Der Computer sendet jedesmal, wenn ein Bild geknipst werden soll ein Signal an den User-Port. Mittels der unten angegebenen Schaltung wird

dann die Kamera ausgelöst. Sie können sich diese folgende Schaltung selbst oder von kundigen Bekannten zusammenlöten lassen. Weiterhin benötigen Sie einen User-Port-Stecker und einen entsprechenden Stecker für den Fernauslöser Ihrer Kamera, die beide im Elektronik - Fachgeschäft erhältlich sind.



Diese Schaltung wurde eigenhändig ausprobiert und lieferte hervorragende Filmergebnisse. Mit den unten stehenden Befehlsfolgen können Sie die Kamera ansteuern:

```
10 C2=56576 : REM BASISADRESSE CIA 2 ($DD00)
20 POKE C2+2, PEEK(C2+2) OR 1 : REM PIN AUF AUSGANG
30 POKE C2,0 : REM KAMERA AUS
40 POKE C2,1 : REM KAMERA EIN
```

Zeile 20 braucht nur einmal im Verlaufe des Programms gegeben werden. Beachten Sie bitte, daß zwischen einem 'ein' und 'aus' genügend Zeit ist, daß das Relais und die Kamera reagieren können.

5.1.3. Graphische Statistik

Ein beliebtes Anwendungsgebiet der Graphik, besonders für den kommerziellen Gebrauch, stellt die Veranschaulichung komplizierter Tabellen in übersichtlichen Diagrammen dar. Hierzu werden verschiedene Darstellungsmethoden verwendet. Die wichtigsten unter ihnen sind:

- Kurvenstatistik
- Balkendiagramme
- Kuchendiagramme

a) Kurvenstatistik:

Besitzen wir viele "Meßwerte" in Abhängigkeit eines bestimmten Faktors (z.B. der Umsatz einer Firma in Abhängigkeit von dem jeweiligen Monat (d.h. in jeweils verschiedenen Monaten)), so verwenden wir allgemein die erste Form der Ausgabe. Hierbei werden die gleichen Techniken angewandt, wie dies bei der Darstellung von 2-dimensionalen Funktionen (s. # 5.1.1) der Fall ist. Der Unterschied liegt lediglich in der Herkunft der einzelnen Daten. Dort wurden sie errechnet, hier aus zwei-reihigen Tabellen gewonnen. Diesen Abschnitt sollten Sie sich also durchgelesen haben. Grundsätzlich gelten dieselben Regeln zur Verschiebung, Scalierung und Verzerrung der Kurven, wie in Abschnitt 5.1.1 dargelegt. Besonders hier spielen die Einheiten an den beiden Koordinatenachsen eine Rolle und sollten beachtet werden (s.u.).

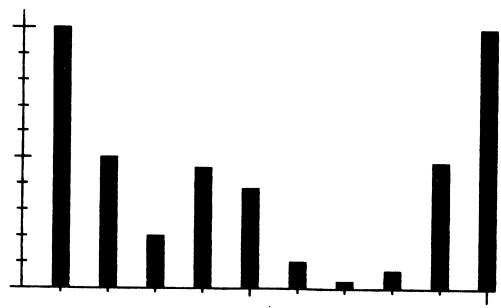
b) Balkendiagramme:

Bei den Balkendiagrammen wird die Sache schon etwas komplizierter, obwohl das Prinzip identisch ist. Sie verwendet man bei relativ kleinen Datenmengen, die optisch besser sichtbar gemacht werden sollen. Ein Wertepaar dient nun nicht zur Bestimmung der Lage eines Punktes auf dem Bildschirm, sondern der Höhe und Position eines Balkens, der sich von der x-Achse in die Höhe zieht. Das folgende kleine Programm soll Ihnen die Programmierung solcher Balken demonstrieren. Nehmen wir an, es sollen die Umsatzzahlen einer (den Marktschwankungen recht stark ausgelieferten) Firma über 10 Jahre mit Hilfe eines

Balkendiagramms dargestellt werden, um dem Leiter eine dringend nötige Übersicht zu vermitteln. Die jeweiligen Zahlen (in Tausend DM) werden dabei in DATA-Zeilen bereit gehalten. Sie könnten aber auch per INPUT solche Daten erfragen und evt. auf Diskette (Kasette) speichern und schon hätten Sie ein recht schönes Statistik-programm.

```
100 REM **********************
110 REM **                    **
120 REM **  BALKENDIAGRAMM    **
130 REM **                    **
140 REM **********************
150 REM
230 REM **** GRAPHIK-ROUTINEN ****
240 REM      *****************
250 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
260 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
270 PC=51212 : PL=51215 :REM PCOLOR/PLOT
280 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
290 CL=51224 : GL=51227 :REM CLINE /GLOAD
300 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
320 REM
330 REM **** ACHSEN ****
340 REM      ******
350 SYS IN : SYS GC : SYS SC,16*7+10 : REM GRAPHIK INIT
360 READ ZA : REM ZAHL DER WERTEPAARE
370 READ HI : REM HOECHSTER Y-WERT
380 XH = 300 : YH = 180 : REM ANZAHL DER PUNKTE IN
X-/Y-RICHTUNG (MAX.)
390 XE=INT(XH/ZA*1) : YE=INT(YH/HI*10) : REM BERECHNUNG
DER (1ER) 10ER EINHEITEN
400 SYS SL,10,10,10,190 : REM Y-ACHSE
410 SYS SL,10,190,310,190 : REM X-ACHSE
420 T=-1:FOR Y=190 TO 10 STEP -YE
430 T=T+1:IF T/5-INT(T/5)=0 THEN SYS SL,5,Y,15,Y : REM
GROSSER STRICH
435 IF T/10-INT(T/10)=0 THEN SYS SL,3,Y,17,Y : REM GROSSER
STRICH
440 SYS SL,7,Y,13,Y : REM EINHEITSMARKIERUNGEN
450 NEXT Y
```

```
460 T=0:BE=20+XE/2 : REM BEGINN ERSTER STRICH
470 FOR X=BE TO 310 STEP XE
480 T=T+1:IF T/5-INT(T/5)=0 THEN SYS SL,X,195,X,190 : REM
GROSSER STRICH
485 IF T/10-INT(T/10)=0 THEN SYS SL,X,198,X,190 : REM
GROSSER STRICH
490 SYS SL,X,193,X,190 : REM EINHEITSMARKIERUNGEN
500 NEXT X
600 REM
610 REM **** DIAGRAMM ****
620 REM      ********
630 BR=XE-20 : REM BALKENBREITE ERECHNEN
640 PO=BE-BR/2 : REM STARTPOSITION ERSTER BALKEN
650 FOR T=1 TO ZA
660 READ DA : REM DATEN LESEN
670 Y=190-DA*YE/10 : REM BALKENHOEHE (EINHEIT!)
680 FOR X=PO TO PO+BR : REM BALKENBREITE
690 SYS SL,X,Y,X,190 : REM BALKEN ZEICHNEN
700 NEXT X
710 PO=PO+XE : REM NEUE BALKENPOSITION
720 NEXT T
800 POKE198,0:WAIT 198,255:SYS OF:END
900 REM
910 REM **** DATEN ****
920 REM      *****
1000 DATA 10 : REM ANZAHL WERTE
1010 DATA 100 : REM HOECHSTER WERT
1100 DATA 100,50,20,46,38,10,2,6,48,99
```



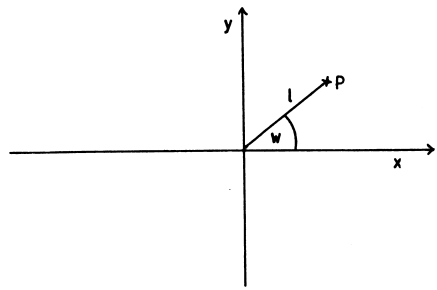
Das Ganze sieht zunächst recht kompliziert aus, da eine Menge von Formeln angewandt werden. Diese "Formeln" sind aber recht gut zu verstehen, da Sie lediglich die Formatierung der Achsen und Balken betreffen.
Wir haben uns in diesem Programm als Ziel gestellt, möglichst variabel bezüglich der Anzahl und der Größe der Daten zu sein, ohne Bereichsüberschreitungen oder viel zu kleine Graphiken zu erzeugen. Aus diesem Grunde werden vor die eigentlichen Daten zwei Werte gestellt: Die Anzahl der Daten und das größte Glied, die in den Zeilen 360/370 eingelesen werden. Weiterhin legen wir uns den Bereich fest, in dem die Balkendiagramme liegen sollen. Wir haben dafür ein 300x180 - Punktefeld reserviert (Z. 380).
Als erstes Problem stellt sich das Zeichnen der Achsen, da wir Einheiten eintragen wollen. Die Senkrechte (y-Achse) soll (so unsere Vereinbarung) jede 10. Einheit mit einem einfachen Strich anzeigen. Jede 50. Einheit steht ein etwa doppelt langer, jede 100. ein etwa dreifacher Strich. Auf der Waagerechten (x-Achse) wird die gleiche Einteilung unternommen, mit dem Unterschied, daß hier die 1., 5. und 10. Einheiten hervorgehoben werden.
Dazu berechnen wir zunächst die Anzahl der Punkte, die auf eine (bei der x-Achse) bzw. 10 (y-Achse) Einheiten fallen dürfen, so daß wir sowohl nach rechts, als auch nach oben nicht zu weit hinauskommen, aber auch die gesamte verwendbare Fläche ausnutzen (Z. 390).
Nun zeichnen wir erst einmal die bloßen Achsen (Z. 400/410). Dann folgt der Eintrag der Scalen (Z. 420-500). Dieser Vorgang sollte relativ leicht zu durchschauen sein. Dazu einige Bemerkungen: In den Zeilen 430, 435, 490 und 495 wird jeweils geprüft, ob gerade der 5. bzw. 10. Strich gezeichnet wird und entsprechend verlängert. Der dazu hinter dem IF stehende Ausdruck ist nichts weiter als das Abschneiden der Zahl vor dem Komma (auch Fraction genannt - das "Gegenteil" von INT). In Zeile 460 wird der Startpunkt der x-Scalierung errechnet. Die hier angeführte Formel hat etwas mit der Breite der Balken zu tun.
Jetzt erst werden die eigentlichen Balken gezeichnet. Dazu wird die Balkenbreite so bestimmt, daß zwischen zwei Balken genügend Platz ist (Z. 630). Die Formel für die Startpositon der Balken und Ihre Höhe sollten Sie

ebenfalls verstehen (bei letzterer Formel wird mit der Anzahl der Punkte pro Einheit multipliziert, um der Scala an der y-Achse gerecht zu werden).
Sie können beliebig die Daten ab Zeile 1000 verändern, sollten aber darauf achten, keine negativen und nicht zu viele zu verwenden. Bei extrem hohen Werten sollten Sie die Scaleneinheiten der Achsen ändern.
Damit haben Sie einen Einblick in die Programmiertechniken dieser Anwendung. Sicher fallen Ihnen noch viele Änderungen ein, die Sie an diesem Programm vornehmen können.

c) Kuchendiagramme:

Diese Art der graphischen Statistik ist geeignet zur übersichtlichen Darstellung von Ver- bzw. Aufteilungen von Mengen in Teilmengen und der Größenverhältnisse unter ihnen. Hierzu wird ein Kuchen (Kreis) gezeichnet, der die Gesamtheit aller zu betrachtenden Elemente darstellt (100 %) und der sich in verschieden große Teilstücke unterteilt. Die Größe der Teilstücke gibt dann den Anteil dieser Teilmenge an der Gesamtmenge an.
Oft sehen wir solche Graphiken bei Wahlen zur Darstellung z.B. der Sitzverteilung im Bundestag etc. oder im Geographieatlas, um beispielsweise die Anteile bestimmter Import- oder Exportwaren eines Landes am Gesamtumschlag zu demonstrieren usw.
Doch wie ist solches programmtechnisch zu verwirklichen? Schließlich haben wir es mit der ziemlich komplizierten Relation eines Kreises zu tun, der in bestimmte Winkelausschnitte unterteilt werden muß. In Paragraph 4.2.2.3 haben wir bereits die Erzeugung eines Kreises (Ellipse) kennengelernt. Doch die dort angegebene Kreisformel ist nicht dazu geeignet, lediglich Ausschnitte, die ja durch Angabe des jeweiligen Winkels definiert werden zu zeichnen. Aus diesem Grunde weisen wir Sie hier (wie schon in Abschnitt 4.2.2.3 angekündigt) in die Kreiserzeugung per Polarkoordinaten ein. Polarkoordinaten sind eine alternative Möglichkeit der Darstellung von Funktionen. Verwendet wird ein sogenanntes Polarkoordinatensystem, in dem nicht x und y als Achsenabschnitte, sondern w als Winkel zwischen der Verbindung von einem beliebigen,

gesuchten Punkt zu dem Nullpunkt und der Waagerechten und l für den Abstand des Nullpunktes und dem besagten Punkt angegeben werden. Veranschaulicht sähe das dann etwa so aus:



Ein Kreis ist dann leicht durch die Änderung des Winkels w bei konstant halten des Abstandes l zu kontruieren. Allgemein für eine beliebige Ellipse mit dem Mittelpunkt im Ursprung (0,0) gilt dann unter Berücksichtigung der Umrechnung von Polar- (l,w) in die uns bekannten sogenannten kartesischen Koordinaten (x,y):

```
x = a*cos(w)
y = b*sin(w)
```

wobei außer den genannten Parametern bedeuten:

a: Radius der Ellipse in x-Richtung
b: Radius der Ellipse in y-Richtung

Diese Zuordnung ist uns schon aus dem oben genannten Abschnitt bekannt. Mithilfe dieser beiden Formeln ist es uns nun möglich, einen bestimmten Randpunkt einer Ellipse durch Angabe des Winkels der Polarkoordinaten des Punktes zu bestimmen.
Eine Sache muß noch erläutert werden. Auch in den vorigen Abschnitten war öfter von Winkeln die Rede. Es gibt verschiedene Möglichkeiten, einen Winkel anzugeben:

- Angabe in Altgrad (0-360)
- Angabe in Neugrad (0-400)
- Angabe in Radiant (0-2*pi)

Die uns vertrauteste von allen ist wohl die erste. Doch unser Rechner rechnet nur mit der dritten, bei der 360 Altgrad dem Wert von 2*pi (pi=3.1415...) entsprechen. Dieser Wert entspricht der Länge eines Kreisbogens mit dem Radius 1 über den angegebenen Winkel. Wollen Sie Radiant in Altgrad oder umgekehrt umrechnen, so verwenden Sie diese Formel:

```
grad = 180*2*pi/rad        oder
rad  = 180*2*pi/grad
```

Damit besitzen wir das notwendige Rüstzeug, um das folgende Programm verstehen zu können:

```
100 REM ************************
110 REM **                    **
120 REM **  KUCHENDIAGRAMME   **
130 REM **                    **
140 REM ************************
150 REM
230 REM **** GRAPHIK-ROUTINEN ****
240 REM      *****************
250 IN=51200 : OF=51203 :REM INIT  /GRAPHIK OFF
260 GC=51206 : SC=51209 :REM GCLEAR/SET COLOR
270 PC=51212 : PL=51215 :REM PCOLOR/PLOT
280 UP=51218 : SL=51221 :REM UNPLOT/SET LINE
290 CL=51224 : GL=51227 :REM CLINE /GLOAD
300 GS=51230 : HC=51233 :REM GSAVE /HARDCOPY
320 REM
330 REM **** ELLIPSE ****
340 REM      *******
350 PI=3.1415
360 SYS IN : SYS GC : SYS SC,16*5+13 : REM GRAPHIK INIT
370 A = 100 : B = 60 : V1=160 : V2=80 : REM PARAMETER
380 W=0:GOSUB 930:X1=X:Y1=Y: REM X1 UND X2 VORBESTIMMEN
390 SP=7*PI/180 : REM STEP
400 BE=0:EN=2*PI : REM START- UND ENDWINKEL
410 GOSUB 800 : REM ELLIPSE ZEICHNEN
420 BE=0:EN=1.03*PI : REM ETWA 180 GRAD
430 V2=100 : REM TIEFER SETZEN
440 GOSUB 800 : REM ELLIPSE ZEICHNEN
```

```
500 REM
510 REM **** KUCHENSTUECKE ****
520 REM      **************
530 READ ZA : DIM T(ZA) : REM ANZAHL DER TEILE
540 FOR S=1 TO ZA
550 READ T(S) : REM DATEN LESEN
560 SU=SU+T(S) : REM SUMME BILDEN
570 NEXT S
580 W = 0 : REM STARTWINKEL
590 FOR S=1 TO ZA
600 PR=T(S)/SU : REM PROZENT AUSRECHNEN
610 WA=2*PI*PR : REM WINKEL DES AUSSCHNITTES
620 W=W+WA : REM WIRKLICHER WINKEL
630 V2=80:GOSUB 930:REM KOORDINATEN ERRECHNEN
640 SYS SL,V1,V2,X,Y : REM TRENNLINIE
650 IF W>PI THEN 690 : REM NUR AUF DER SICHTBAREN SEITE
660 X1=X:Y1=Y : REM MERKEN
670 V2=100:GOSUB 930:REM UNTERE KOORD. ERRECHNEN
680 SYS SL,X1,Y1,X,Y : REM SENKRECHTE ZUM UNTEREN BOGEN
690 NEXT S
799 WAIT 198,255:SYS OF:END
800 REM
810 REM **** ELLIPSENBOGEN ****
820 REM      **************
830 FOR W=BE TO EN+SP STEP SP : REM WINKEL BESTIMMEN
840 GOSUB 930 : REM KOORDINATEN BESTIMMEN
850 SYS SL,X1,Y1,X,Y : REM LINIE
860 X1=X:Y1=Y
870 NEXT W : RETURN
900 REM
910 REM **** PUNKT BERECHNEN ****
920 REM      ***************
930 X = A*COS(W) + V1
940 Y = B*SIN(W) + V2 : RETURN
1000 REM
1010 REM **** DATEN ****
1020 REM      *****
1100 DATA 6 : REM ANZAHL
1110 DATA 20,10,15,40,30,8
```

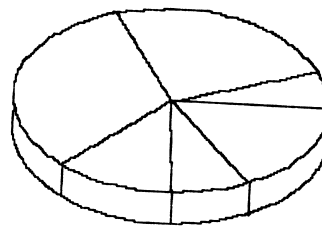
Wie Sie sehen, können Sie hier aus den Zeilen 800-940 für

Ihre eigenen Programme eine alternative Kreisformel entnehmen. Die Übergabeparameter sind im einzelnen:

BE: Startwinkel
EN: Endwinkel
SP: Schritteinheit
V1: Mittelpunkt-x-Koordinate
V2: Mittelpunkt-y-Koordinate
A : x-Radius
B : y-Radius

Zu der Schrittweite SP muß folgendes gesagt werden: Sie bestimmt, in welchem Winkelabstand die einzelnen Punkte der Ellipse berechnet werden, also die Genauigkeit. Diese Punkte werden dann in Zeile 850 durch eine Linie verbunden. Wählen Sie SP sehr groß, z.B. 30 oder 45, so erhalten Sie spezielle Effekte: Damit wird dann ein gleichseitiges Vieleck (8-Eck etc.) gezeichnet, was für andere Zeichnungen gut verwendet werden kann!
Eigentlich muß die Zeile 380 bereits in diesem Unterprogramm aufgeführt werden und sollte auch von Ihnen übernommen werden, sofern Sie lediglich die Ellipsenroutine verwenden wollen. Hier wurde sie herausgenommen, um einen speziellen Effekt zu erzielen (s.u.).
In dem obigen Programm wollen wir nicht einfach eine simple Scheibe zeichnen und dann entsprechende Abschnitte abtragen. Vielmehr soll das Ganze etwas Format haben und in 3-D gezeichnet sein. Es wird eine runde Platte mit einer gewissen Dicke dargestellt, die in die unterschiedlichen "Kuchen"stücke zerschnitten ist und von schräg oben gesehen werden soll. Dazu zeichnen wir zunächst einmal eine Ellipse als Oberfläche (Z. 410). Alsdann zeichnen wir die gleiche Figur nur um ein paar Punkte nach unten verschoben und als Halbellipse, um den unteren Rand der Platte darzustellen (Z. 440). Eigentlich wird bei dieser Konstruktion der Platte ein wenig "gemogelt", da wir die seitlichen Ränder als gerade Linien darstellen müßten (was rechts auch geschieht), dafür zeichnen wir die Ellipse ein klein wenig über 180 Grad (1*pi) hinaus, was wegen der Auflösung kaum auffällt.
Nun kommen wir zu den eigentlichen Kuchenschnitten:

Vorher sollten wir etwas über die verwendete Datenstruktur sagen. Am Anfang vor den eigentlichen Daten steht wie immer ihre Anzahl. Dann folgen beliebig viele und beliebig hohe Zahlen, die z.B. die Anzahl der Sitze der einzelnen Parteien oder allgemein die Größe der verschiedenen Teilmengen wiedergeben.
Im ersten Teil der ab Zeile 500 folgenden Routine werden alle durch ZA angegebenen Daten in ein eindimensionales Feld eingelesen und (das ist der eigentliche Grund für diese Schleife) die Summe aller Teilmengen gebildet (Z. 540-570).
In der nächsten FOR...NEXT - Schleife werden dann die Unterteilungen vorgenommen: Zeile 600 rechnet den prozentualen Anteil der jeweiligen Partei an der Gesamtheit aus. Zeile 610 bestimmt die daraus resultierende Größe des Kreis- (Ellipsen-) Ausschnittes in Radiant, die zu dem aktuellen Winkel W hinzugezählt wird (Z. 620). Nun, da der Winkel des betreffenden Schnittes bekannt ist, kann die Position des entsprechenden Randpunktes der Ellipse errechnet werden. Alsdann wird vom Mittelpunkt der Ellipse zu diesem Punkt eine Linie gezeichnet (Z.640).
Nun kommt es darauf an, ob sich dieser Schnitt bereits auf der hinteren Hälfte der Scheibe oder noch vorne befindet. Im ersten Fall ist die Sache erledigt und der nächste Wert kann bearbeitet werden. Im zweiten Fall jedoch muß der Schnitt noch sichtbar bis zur Unterkante gezogen werden. Dies geschieht, indem wir einfach die Position auf der verschobenen Kante berechnen und vom zuletzt gezeichneten Punkt bis hierhin ein Linie ziehen (Z. 650-680). Damit wäre der 3-dimensionalität Genüge getan.



Auch dieses Programm ist selbstverständlich voll ausbaufähig. Sie könnten beispielsweise die einzelnen Teile in verschiedenen Farben oder schraffiert zeichnen usw. Ihrer Phantasie sind keine Grenzen gesetzt.

5.2 Laufschriften

Eine verlockend einfache und gleichzeitig recht schöne Art und Weise, beliebig gestaltete und bewegliche Buchstaben auf den Bildschirm (auch in die hochauflösende Graphik) zu bringen, ist die Konstruktion von Buchstabensprites. Die Sprites besitzen eine geradezu phantastische Auflösung zur Erstellung von Schrift und können vergrößert, bewegt usw. werden.
Das einzige Problem bei diesem Unterfangen ist der begrenzte Speicherraum, der uns in Basic zur Verfügung steht. Mit den vier Blöcken, die uns in Basic zur völlig freien Verfügung stehen, kommen wir nicht aus. Um hier Abhilfe zu schaffen, können wir erstens den gesamten VIC-Adressbereich um 16, 32 oder gar 48 K nach oben verschieben (s. # 3.3.2) und so in Speicherebenen gelangen, die wir ohne weiteres nutzen können. Doch hier tritt eine kleine Schwierigkeit auf. Es verschieben sich ja nicht nur die Spriteblöcke, sondern gleich alle Bildschirmspeicher wie Videoram und Graphikspeicher, was die Handhabung erheblich erschwert.
Wir wollen uns die zweite Möglichkeit zu Nutze machen: Wir packen alle Sprite - Definitionen, die wir in unserem Programm verwenden, in den Bereich von $2000-$3FFF (8192-16383), wo wir normalerweise unsere Graphik beherbergen. Zwar können wir dann nicht mehr gleichzeitig Graphik anzeigen, aber man muß bereit sein, auch Kompromisse zu schließen. Damit verwenden wir die Blöcke 8192/64=128 bis 255 und haben Platz für insgesamt 127 Spritedefinitionen, was ausreichend sein sollte. Wir brauchen dann nur darauf zu achten, daß wir nicht unbedingt riesige Mammutprogramme oder solche mit einem großen Speicheraufwand schreiben und der Fall ist erledigt.
Kommen wir zu den Buchstabendefinitionen. Normalerweise ist es unnötiger Platzverbrauch, alle Spritedefinitionen als

DATA-Zeilen in unser Basicprogramm nieder zu legen. Alternative Methoden wurden Ihnen in Paragraph 4.3 vorgestellt. Hier allerdings wollen wir ausnahmsweise damit arbeiten, zumal wir nur wenige Buchstabendefinitionen vorstellen.
Die einzelnen Sprites können Sie mit dem Spriteformer in Abschnitt 4.3 erstellen und später z.B. direkt in den Speicher einlesen. Sie können sich so einen ganzen Vorrat an Zeichen erstellen und bei Bedarf abrufbereit halten.
Wir wollen aber nun zu der eigentlichen Aufgabe vorschreiten: der Programmierung von Laufschriften. Hierzu ein kleines Basicprogramm:

```
100 REM **********************
110 REM **                    **
120 REM **  LAUFSCHRIFTEN     **
130 REM **                    **
140 REM **********************
150 REM
160 V = 53248 : REM BASISADRESSE VIDEOCONTROLLER
170 POKE V+32,0 : POKE V+33,0 : REM RAHMEN UND HINTERGRUND =
SCHWARZ
175 PRINT CHR$(147)
180 REM
190 REM **** EINSPEICHERUNG ****
200 REM      ***************
210 FOR X=1 TO 9 : REM 9 BUCHSTABEN
220 READ A$ : REM NAME DES BUCHSTABEN
230 BK=ASC(A$)-32+128 : REM SPRITES NACH ASCII GEORDNET IM
SPEICHER (BLOCKNUMMER)
240 AD=BK*64 : REM ADRESSEN AB 8192
250 FOR Y=AD TO AD+62
260 READ DA : POKE Y,DA : REM DATEN LESEN UND SCHREIBEN
270 NEXT Y,X : REM 63 DATEN/8 BUCHSTABEN LESEN
300 REM
310 REM **** INITIALISIERUNG ****
320 REM      ****************
330 FOR X=0 TO 7
340 POKE V+39+X,X+1 : REM FARBEN SPRITES 0-7 FESTLEGEN
350 NEXT X
360 POKE V+23,255 : REM ALLE SPRITES GROSS (Y)
```

```
370 POKE V+29,255 : REM ALLE SPRITES GROSS (X)
380 POKE V+27,0 : REM PRIORITAET
390 POKE V+28,0 : REM NORMAL-FARBEN
400 SP=0 : REM START-SPRITENUMMER
410 ZA=8 : REM ANZAHL DER BUCHSTABEN AUF DEM BILD
(GLEICHZEITIG / MAX.:8)
420 AB=330/ZA : REM ABSTAND ZWEIER SPRITES
430 GE = 20 : REM GESCHWINDIGKEIT
500 REM
510 REM **** LAUFSCHRIFT ****
520 REM      ***********
530 TE$="DATA-BECKER---" : REM LAUFTEXT
540 FOR LA=1 TO 10 : REM 10 DURCHLAEUFE
550 FOR BU=1 TO LEN(TE$)
560 BU$=MID$(TE$,BU,1) : REM LAUFENDER BUCHSTABE
570 BK =ASC(BU$)-32 + 128 : REM BLOCKNUMMER DES BUCHSTABEN
580 POKE 2040+SP,BK : REM SPRITE AUF ENTSPRECHENDEN BLOCK
SETZEN
590 POKE V+SP*2,95 : REM SPRITE-X-KOORD.-LOW-BYTE
600 POKE V+SP*2+1,100 : REM SPRITE-Y-KOORD.
610 POKE V+16,PEEK(V+16) OR 2^SP : REM
SPRITE-X-KOORD.-HIGH-BIT
620 POKE V+21,PEEK(V+21) OR 2^SP : REM SPRITE EINSCHALTEN
630 SP=SP+1 : REM NAECHSTE SPRITENUMMER
635 IF SP=ZA THEN SP=0
640 FOR K=1 TO AB STEP GE : REM AB SCHRITTE ***  BEWEGEN  ***
650 FOR S=0 TO ZA-1 : REM ZA SPRITES BEWEGEN
660 AD=V+S*2:XL=PEEK(AD)-GE:XH=PEEK(V+16)AND2^S:REM
X-KOORDINATE DES SPRITES
670 IF XL<0 THEN XL=256+XL:POKE V+16,PEEK(V+16) AND
255-2^S:IFXH=0THENXL=0
680 POKE AD,XL
685 GOSUB 800
690 NEXT S,K
700 NEXT BU : REM NAECHSTEN BUCHSTABEN
710 NEXT LA : REM NAECHSTEN DURCHLAUF
800 POKE V+21,PEEK(V+21) AND 255-2^SP:RETURN : REM SPRITE AUS
1000 REM
1010 REM **** SPRITE-DATA ****
1020 REM      ***********
1090 REM
```

```
1100 DATA A : REM BUCHSTABE A
1110 DATA 000,000,000, 000,000,000, 003,255,192
1120 DATA 007,000,224, 006,000,096, 006,000,096
1130 DATA 006,000,096, 006,000,096, 006,000,096
1140 DATA 006,000,096, 007,255,224, 006,000,096
1150 DATA 006,000,096, 006,000,096, 006,000,096
1160 DATA 006,000,096, 006,000,096, 006,000,096
1170 DATA 006,000,096, 000,000,000, 000,000,000
1190 REM
1200 DATA B : REM BUCHSTABE B
1210 DATA 000,000,000, 000,000,000, 003,255,000
1220 DATA 003,001,128, 003,000,192, 003,000,192
1230 DATA 003,000,192, 003,000,192, 003,001,128
1240 DATA 003,255,000, 003,001,128, 003,000,192
1250 DATA 003,000,096, 003,000,096, 003,000,096
1260 DATA 003,000,096, 003,000,192, 003,001,128
1270 DATA 003,255,000, 000,000,000, 000,000,000
1290 REM
1300 DATA C : REM BUCHSTABE C
1310 DATA 000,000,000, 000,000,000, 001,255,192
1320 DATA 003,129,192, 003,000,000, 003,000,000
1330 DATA 003,000,000, 003,000,000, 003,000,000
1340 DATA 003,000,000, 003,000,000, 003,000,000
1350 DATA 003,000,000, 003,000,000, 003,000,000
1360 DATA 003,000,000, 003,000,000, 003,129,192
1370 DATA 001,255,192, 000,000,000, 000,000,000
1390 REM
1400 DATA D : REM BUCHSTABE D
1410 DATA 000,000,000, 000,000,000, 015,255,000
1420 DATA 012,001,128, 012,000,192, 012,000,192
1430 DATA 012,000,192, 012,000,192, 012,000,192
1440 DATA 012,000,192, 012,000,192, 012,000,192
1450 DATA 012,000,192, 012,000,192, 012,000,192
1460 DATA 012,000,192, 012,000,192, 012,001,128
1470 DATA 015,255,000, 000,000,000, 000,000,000
1500 DATA E : REM BUCHSTABE E
1510 DATA 000,000,000, 000,000,000, 003,255,192
1520 DATA 003,001,192, 003,000,000, 003,000,000
1530 DATA 003,000,000, 003,000,000, 003,006,000
1540 DATA 003,255,000, 003,006,000, 003,000,000
1550 DATA 003,000,000, 003,000,000, 003,000,000
```

```
1560 DATA 003,000,000, 003,000,000, 003,001,192
1570 DATA 003,255,192, 000,000,000, 000,000,000
1590 REM
1600 DATA K : REM BUCHSTABE K
1610 DATA 000,000,000, 000,000,000, 003,001,128
1620 DATA 003,003,000, 003,006,000, 003,012,000
1630 DATA 003,024,000, 003,048,000, 003,096,000
1640 DATA 003,192,000, 003,192,000, 003,096,000
1650 DATA 003,048,000, 003,024,000, 003,012,000
1660 DATA 003,006,000, 003,003,000, 003,001,000
1670 DATA 003,000,192, 000,000,000, 000,000,000
1690 REM
1700 DATA R : REM BUCHSTABE R
1710 DATA 000,000,000, 000,000,000, 001,255,000
1720 DATA 003,001,128, 003,000,192, 003,000,192
1730 DATA 003,000,192, 003,000,192, 003,001,128
1740 DATA 003,255,000, 003,192,000, 003,096,000
1750 DATA 003,048,000, 003,024,000, 003,012,000
1760 DATA 003,006,000, 003,003,000, 003,001,000
1770 DATA 003,000,192, 000,000,000, 000,000,000
1790 REM
1800 DATA T : REM BUCHSTABE T
1810 DATA 000,000,000, 000,000,000, 015,255,192
1820 DATA 012,048,192, 000,048,000, 000,048,000
1830 DATA 000,048,000, 000,048,000, 000,048,000
1840 DATA 000,048,000, 000,048,000, 000,048,000
1850 DATA 000,048,000, 000,048,000, 000,048,000
1860 DATA 000,048,000, 000,048,000, 000,048,000
1870 DATA 000,048,000, 000,000,000, 000,000,000
1890 REM
1900 DATA "-" : REM BINDESTRICH
1910 DATA 000,000,000, 000,000,000, 000,000,000
1920 DATA 000,000,000, 000,000,000, 000,000,000
1930 DATA 000,000,000, 000,000,000, 000,000,000
1940 DATA 000,000,000, 003,255,192, 000,000,000
1950 DATA 000,000,000, 000,000,000, 000,000,000
1960 DATA 000,000,000, 000,000,000, 000,000,000
1970 DATA 000,000,000, 000,000,000, 000,000,000
```

Dieses Programm kann Ihnen nur die Grundzüge der Laufschrift-technik vermitteln. Es liegt an Ihnen, die entsprechenden

Anwenderprogramme zu schreiben. Richtig schnell und ansehnlich wird das Ganze natürlich erst in Maschinensprache. Aber ich hoffe, Sie haben auch so Ihren Spaß daran. Wenn Sie sich das Programm durchschauen, so sollte Ihnen das Verständnis der Abläufe durch die vielen REM-Zeilen gut verständlich sein. Wie gesagt können Sie sich einen ganzen Zeichensatz zusammenstellen und schließlich beliebige Texte ausgeben. Viel Spaß!

5.3 Das Geheimnis der Spiele

Inzwischen gibt es wohl bereits annähernd tausend gute bis weniger gute Spiele für den Commodore 64, die in Computer - Shops zu kaufen sind. Wir lassen uns von ihrer Graphik, ihren Soundkaskaden berauschen und klopfen uns befriedigt auf die Schulter: "Hab' ich mir doch einen guten Rechner gekauft, was der alles kann!". In der Tat zeigen von allen Programmen ganz besonders die Spiele, welche Qualitäten ein Gerät besitzt, da diese meist bis an die Grenzen des Machbaren stoßen. Beim 64er sind diese Grenzen ziemlich hoch angesetzt und sogar die besten Spieleprogrammierer haben Probleme alle Möglichkeiten voll auszunutzen. Doch was nutzt einem ein guter Computer, mit dem alles möglich ist, wenn man selbst nicht weiß, wie es geht? Und ewig nur zuzuschauen, was andere programmiert haben, macht einen auch nicht satt.
In diesem Buch haben Sie bereits Vieles erfahren, das Ihnen helfen wird, Ihren Computer bezüglich Graphik, Sprites und allgemein der Bildschirmausgabe optimal zu nutzen. Natürlich konnte nicht annähernd alles, was der 64er bietet und im kapitel 3 (Hardwaregrundlagen) steht, auch zur Anwendung gebracht werden. Das ist auch gut so. Auf diese Weise bleibt noch genügend Freiraum für Ihren Forscherdrang.
Die meisten Spiele sind in Maschinensprache geschrieben, da das originale Basic einfach zu langsam ist. Es wurden Ihnen bereits einige Utilities in Assembler angeboten, die quasi als kleine Erweiterung des Basic - Befehlssatzes für die Aufbesserung der Geschwindigkeiten Ihrer Programme zur Verfügung stehen (z.B. das Graphik - Paket). Hier nun sollen Ihnen einige Techniken und Erweiterungen speziell für Spiele

vermittelt werden (Sie sind selbstverständlich auch für andere Anwendungen recht nützlich). So sind Sie in der Lage, auch in Basic (erst recht in Maschinensprache) schnelle und anspruchsvolle Spiele zu programmieren.

5.3.1. Animation

Unter Animation versteht man die Erzeugung bewegter Bilder auf dem Bildschirm. Natürlich "leben" die Spiele von der Animation, sofern nicht etwa Denkspiele wie Schach, Memorie etc. gemeint sind. Ohne Bewegung auf dem Bildschirm "läuft" nichts. Oft werden Spiele nur aufgrund der Qualität dieser Bewegung in die Reihe der Actionspiele oder "sonstige" Spiele eingereiht. Wir wollen uns deshalb zunächst mit diesem wichtigen Thema beschäftigen.
Man unterscheidet beim Commodore 64 fünf Arten von Animation:

- Sprite-interne Bewegung
- Spriteverschiebung
- Zeichen-interne Bewegung
- Zeichenverschiebung
- graphische Animation

Unter 'intern' verstehen wir hier das Bewegen des jeweiligen Objektes selbst, ohne daß es seine Position auf dem Bildschirm verändert.
Die ersten beiden Formen haben wir bereits in den zwei Beispielen zur Spriteprogrammierung im Abschnitt 4.3.2 ausführlich dargelegt und erläutert. Sie sollten sich diesen Paragraphen sowieso durchgelesen haben, da die Sprite-programmierung ein, wenn nicht gar das wichtigste Fundament der Spiele darstellt.
Auch den letzten Punkt wollen wir hier nicht abhandeln, da er sich aus den verschiedenen Graphikkapiteln ableitet und aus Geschwindigkeitsgründen nur selten oder gar nicht bei Spielen Verwendung findet.
Äußerst beliebt sind dagegen die beiden restlichen Punkte. Meist werden sie in Verbindung mit einer Zeichensatzänderung verwendet.

a) Zeichen-interne Bewegung:

Das Prinzip der Zeichen-internen Bewegung ist das gleiche, wie bei der Bewegung der Sprites. Eine aus einem oder mehreren Zeichen zusammengesetzte Figur wird durch den stetigen Wechsel von Zuständen bestimmter Teile des Objektes so verändert, daß daraus eine Bewegung entsteht. Im Klartext bedeutet das folgendes:

Angenommen wir wollen ein Männchen so steuern, daß es stets beide Arme und Beine auf und nieder bewegt. Wir setzen dieses Männchen dann aus mehreren Teilen zusammen, damit es nicht zu klein wird. Um nun die gewünschte Bewegung zu programmieren, legen wir uns jedoch zwei oder mehr Männchen bereit, die jeweils andere Phasen der Bewegung festhalten. Diese verschiedenen Figuren lassen wir dann in einem uns genehmen Takt abwechselnd an der gleichen Stelle auf dem Bildschirm erscheinen, und schon haben wir den gewünschten Effekt.

Um die diversen Zeichen auf eine bestimmte Position des Bildschirms zu bringen, verwenden wir die in Abschnitt 4.1 dargelegte Routine zur programminternen Cursorsteuerung. Besonders effektiv wird das Ganze natürlich mit einem veränderten Zeichensatz oder ein paar veränderten Zeichen. Wie dies auf einfache Weise gemacht wird, zeigt Ihnen Abschnitt 4.4. Besonders hier zeigen Multicolorzeichen Ihren Sinn. Ihr Objekt wird recht schön farbig. Sie sehen, beim Thema Spiele fließt alles Wissen und Können des Programmierers zusammen.

Das folgende Beispiel soll Ihnen den Nutzen auch des originalen Zeichensatzes für dieses Thema darlegen:

```
100 REM ********************
110 REM **                  **
120 REM **  ANIMATION-1     **
130 REM **                  **
140 REM ********************
150 REM
160 A$(0)=" "       +CHR$(119)
170 A$(1)=CHR$( 99)+CHR$(123)+CHR$( 99)
180 A$(2)=CHR$(167)+CHR$(183)+CHR$(165)
190 A$(3)=" "       +CHR$(113)
200 A$(4)=CHR$(173)+CHR$(123)+CHR$(189)
```

```
210 A$(5)=CHR$(183)+CHR$(183)+CHR$(183)
300 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
310 X=18 : REM SPALTENPOSITION
320 FOR ZA=0 TO 5 STEP 3
330 Y=12 : GOSUB 1000 : REM POSITIONIEREN
340 PRINT A$(ZA):GOSUB 1010 : REM KOPF
350 PRINT A$(ZA+1):GOSUB 1010 : REM RUMPF/ARME
360 PRINT A$(ZA+2) : REM BEINE
370 FOR S=1 TO 100 : NEXT S : REM WARTESCHLEIFE
380 NEXT ZA
390 GOTO 320 : REM MIT <RUN/STOP> UNTERBRECHEN
400 REM
410 REM **** POSITIONIEREN ****
420 REM      **************
1000 PRINT CHR$(19);:IF Y>0 THEN FOR T=1 TO Y:PRINT:NEXT T
1010 PRINT TAB(X);: RETURN
```

In diesem Programm werden zu Anfang (Zeilen 160-210) die Teile für die zwei Phasen der Bewegung des Männchens definiert. Insgesamt bauen wir es aus 7 Zeichen auf, die in 3 untereinander liegenden Reihen stehen sollen. Jede der drei Reihen wird in einen separaten Array-Speicher (A$(...)) abgelegt (Nr. 0-2 für die erste Phase und Nr. 3-5 für die zweite). Alsdann werden die Startkoordinaten des Männchens festgelegt und positioniert (Z. 330). Der Rest ist relativ einfach zu durchschauen: Die drei Reihen werden gezeichnet (1.Phase), wobei jeweils nur ein Teil der Positionierungsroutine aufgerufen wird. Im zweiten Durchlauf der FOR...NEXT - Schleife werden dann die Reihen der 2. Phase gezeichnet. Durch Veränderung der Länge der Warteschleife in Zeile 370 kann die Geschwindigkeit der Bewegung gesteuert werden. Versuchen Sie doch einmal, dieses Männchen durch eigene Zeichendefinitionen darzustellen (s. # 4.4). Dann ist es Ihnen gleichfalls möglich, mehr als zwei Bewegungsphasen nacheinander ablaufen zu lassen, was die Effektivität natürlich um Einiges steigert.

b) Zeichenverschiebung:

Wie bei den Sprites können wir auch unser Männchen über den Bildschirm bewegen. Dies geschieht auf ähnliche Weise,

wie in Kapitel 4.3 angegeben. Wir zeichnen unser Männchen an einer bestimmten Position auf den Bildschirm. Nach einiger Zeit (Warteschleife) löschen wir es wieder und setzen es dafür ein klein wenig verschoben weiter nach rechts, links, oben oder unten usw. Aus dieser stetigen Verschiebung resultiert dann eine kontinuierliche Bewegung, wie wir sie von den Sprites her kennen. Das einzige Problem dabei ist die Auflösung der Bewegung. Normalerweise können wir unser Objekt immer nur um minimal ein Zeichen verschieben. Dieses Manko können wir allerdings ein klein wenig dadurch ausgleichen, daß wir auch hier "Zwischenphasen" programmieren (das Objekt steht praktisch "zwischen" zwei Zeichen). Das geht natürlich nur unter Veränderung des Zeichensatzes.
Doch beschäftigen wir uns lieber erst einmal mit den Grundlagen. Das folgende Programm vereinigt die Technik der internen Bewegung mit der Zeichenverschiebung und vermittelt ein schon recht hübsches Bild. Wenn wir dies Männchen nun noch per Joystick steuern, bleibt (fast) kein Wunsch mehr offen.

```
100 REM ********************
110 REM **                  **
120 REM **  ANIMATION-2     **
130 REM **                  **
140 REM ********************
150 REM
160 A$(0)=" "+" "       +CHR$(119)+" "       +" "
170 A$(1)=" "+CHR$( 99)+CHR$(123)+CHR$( 99)+" "
180 A$(2)=" "+CHR$(167)+CHR$(183)+CHR$(165)+" "
190 A$(3)=" "+" "       +CHR$(113)+" "       +" "
200 A$(4)=" "+CHR$(173)+CHR$(123)+CHR$(189)+" "
210 A$(5)=" "+CHR$(183)+CHR$(183)+CHR$(183)+" "
300 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
305 SP=1:B=0:E=33
310 FOR X=B TO E STEP SP : REM SPALTENPOSITION
320 FOR ZA=0 TO 5 STEP 3
330 Y=12 : GOSUB 1000 : REM POSITIONIEREN
340 PRINT A$(ZA):GOSUB 1010 : REM KOPF
350 PRINT A$(ZA+1):GOSUB 1010 : REM RUMPF/ARME
360 PRINT A$(ZA+2) : REM BEINE
```

```
370 FOR S=1 TO 60 : NEXT S : REM WARTESCHLEIFE
380 X=X+SP
390 NEXT ZA
400 X=X-SP
410 NEXT X
420 SP=-SP : ZW=B : B=E : E=ZW : GOTO 310 : REM AUSTAUSCH
(BEWEGUNGSR. AENDERN)
500 REM
510 REM **** POSITIONIEREN ****
520 REM      **************
1000 PRINT CHR$(19);:IF Y>0 THEN FOR T=1 TO Y:PRINT:NEXT T
1010 PRINT TAB(X);: RETURN
```

Wir haben in diesem Programm um die bereits bekannte Bewegungsroutine eine weitere FOR...NEXT - Schleife gepackt. Weiterhin mußten wir die Reihendefinitionen in den Zeilen 160-210 jeweils um ein Leerzeichen vorne und hinten erweitern, um ein Löschen der zuletzt gezeichneten Figur sowohl beim Hin-, als auch beim Zurückgehen zu gewährleisten. Vielleicht versuchen Sie einmal hinter die Bedeutung der Speicher SP, B, E (und ZW) zu kommen. Sie dienen zur Bereitstellung der notwendigen Parameter für Rechts- und Linkslauf.
Damit haben wir Ihnen einige Tips gegeben, wie Sie neben den Sprites noch einige einfache Bewegungen mehr in Ihr Spiel bekommen. Kommen wir daher gleich zum nächsten Thema.

5.3.2. Scrolling

Wer hat nicht schon einmal 'Defender' oder ähnliche Actionspiele gesehen oder gar mit ihnen gespielt, in denen Sie Führer eines Raumschiffes sind, das mit einer Höllengeschwindigkeit durch den Weltraum fliegt. Oder Sie fahren mit einem Rennwagen unter Aufbringung aller Konzentration auf einer Piste, verfolgt von anderen Rivalen, die Sie abdrängen.
Doch wenn Sie genauer hinschauen, sind es nicht das Flugzeug oder das Auto, die sich vorwärts bewegen, sondern vielmehr der Hintergrund. D.h. der gesamte Bildschirm

(oder Teile) wird nach rechts, links, oben oder unten verschoben, während das jeweilige zu steuernde Objekt (ein Sprite) meist nur beschränkt bewegt werden kann. Nun ist ein solches Verschieben, Rollen oder Scrolling des Bildschirms eine sehr zeitaufwendige Sache und kann daher nur in Maschinensprache durchgeführt werden. Dabei verwendet man nicht etwa Graphik (hier müßten für einmal verschieben 8 K bewegt werden) sondern den Textmodus, was bei Spielen fast das Gleiche ist, da wir ja den Zeichensatz beliebig verändern können und so quasi hochauflösende Bilder erhalten. Bekanntlich müssen hier nur etwa 1 K Bytes bewegt werden, was die Arbeit gewaltig verringert. Im folgenden werden Ihnen entsprechende Assemblerroutinen angegeben, die Sie genauso wie die Befehle des Graphik - Paketes per SYS aufrufen können:

```
10: CC00                *= $CC00
 20:                    ;
 30:                    ;*****************
 40:                    ;**             **
 50:                    ;**  SCROLLING  **
 60:                    ;**             **
 70:                    ;*****************
 80:                    ;
110: CC00           CHKGET =  $B7F1
120: CC00           OB     =  2038
130: CC00           UN     =  2039
140: CC00           FLAG   =  $FD
150: CC00           ZAHL   =  $FE
160: CC00           ADRESS =  $61
200: CC00 20 F1 B7  START  JSR CHKGET  ;KOMMA + BYTE HOLEN
210: CC03 8A               TXA          ;RECHTS/LINKS-FLAG
220: CC04 4A               LSR A
230: CC05 08               PHP
280: CC06 20 F1 B7         JSR CHKGET
290: CC09 E0 19            CPX #25      ;OBERE ZEILE
300: CC0B 90 02            BCC S1
310: CC0D A2 18            LDX #24
320: CC0F 8E F6 07 S1      STX OB
330: CC12 20 F1 B7         JSR CHKGET   ;UNTERE ZEILE
340: CC15 E0 19            CPX #25
```

```
350: CC17 90 02          BCC S2
360: CC19 A2 18          LDX #24
370: CC1B 8E F7 07 S2    STX UN
380: CC1E 8A             TXA
390: CC1F AE F6 07       LDX OB
400: CC22 AC F7 07       LDY UN
410: CC25 38             SEC
420: CC26 ED F6 07       SBC OB       ;OBEN - UNTEN
430: CC29 B0 08          BCS S3
440: CC2B 49 FF          EOR #$FF     ;OBEN<UNTEN=>TAUSCH
450: CC2D AE F7 07       LDX UN
460: CC30 AC F6 07       LDY OB
470: CC33 85 FE    S3    STA ZAHL     ;ZAEHLER
480: CC35 28             PLP
490: CC36 08             PHP          ;RE/LI-FLAG
500: CC37 90 03          BCC S4
510: CC39 C8             INY          ;RECHTS:ZEILE WEITER
520: CC3A 98             TYA          ;UND UNTEN STARTEN
530: CC3B AA             TAX
540: CC3C BD CB CC S4    LDA MULH,X   ;HIGH BYTE ADRESSE
550: CC3F 85 62          STA ADRESS+1
560: CC41 BD E5 CC       LDA MULL,X   ;LOW-BYTE
570: CC44 85 61          STA ADRESS
580: CC46 28             PLP
590: CC47 08             PHP          ;RE/LI-FLAG
600: CC48 90 08          BCC MOVE
610: CC4A E9 01          SBC #1       ;BEI RECHTS  -1
620: CC4C 85 61          STA ADRESS
630: CC4E B0 02          BCS MOVE
640: CC50 C6 62          DEC ADRESS+1
650:                ;
660:                ;VERSCHIEBUNG
670:                ;*************
680:                ;
690: CC52 A5 62    MOVE  LDA ADRESS+1
700: CC54 29 03          AND #3
710: CC56 09 04          ORA #4       ;BASISADRESSE=$0400
720: CC58 28             PLP
730: CC59 08             PHP          ;FLAG
750: CC5A 20 86 CC       JSR MOVE1    ;VIDEORAM VERSCHIEBEN
760: CC5D 28             PLP
```

```
 770: CC5E 08              PHP          ;FLAG
 780: CC5F A5 61           LDA ADRESS
 790: CC61 90 0A           BCC M1
 800: CC63 69 27           ADC #39      ;C=1! / RECHTS
 810: CC65 85 61           STA ADRESS
 820: CC67 90 0C           BCC M2
 830: CC69 E6 62           INC ADRESS+1
 840: CC6B B0 08           BCS M2       ;UNBEDINGT
 850: CC6D E9 27    M1     SBC #39      ;C=0! / LINKS
 860: CC6F 85 61           STA ADRESS
 870: CC71 B0 02           BCS M2
 880: CC73 C6 62           DEC ADRESS+1
 890: CC75 A5 62    M2     LDA ADRESS+1
 900: CC77 29 03           AND #3
 910: CC79 09 D8           ORA #$D8     ;FARBRAM BEI $D800
 912: CC7B 28              PLP
 915: CC7C 08              PHP
 920: CC7D 20 86 CC        JSR MOVE1    ;FARBRAM VERSCHIEBEN
 930: CC80 C6 FE           DEC ZAHL
 940: CC82 10 CE           BPL MOVE
 950: CC84 28              PLP
 960: CC85 60              RTS
 970:                ;
 980:                ;VERTEILER
 990:                ;**********
1000:                ;
1010: CC86 85 62    MOVE1  STA ADRESS+1
1020: CC88 90 03           BCC LINKS
1030: CC8A 4C AB CC        JMP RECHTS
1040:                ;
1050:                ;LINKSVERSCHIEBUNG
1060:                ;******************
1070:                ;
1080: CC8D A0 00    LINKS  LDY #0
1090: CC8F B1 61           LDA (ADRESS),Y
1100: CC91 AA              TAX          ;ERSTES BYTE MERKEN
1140: CC92 A0 27           LDY #39
1150: CC94 B1 61    L2     LDA (ADRESS),Y
1160: CC96 48              PHA          ;MERKER 1
1170: CC97 8A              TXA          ;HOLE MERKER 2
1180: CC98 91 61           STA (ADRESS),Y
```

```
1190: CC9A 68              PLA          ;HOLE MERKER 1
1200: CC9B AA              TAX          ;IN MERKER 2
1210: CC9C 88              DEY
1220: CC9D 10 F5           BPL L2
1230: CC9F 18              CLC
1240: CCA0 A5 61           LDA ADRESS
1250: CCA2 69 28           ADC #40      ;NAECHSTE ZEILE
1260: CCA4 85 61           STA ADRESS
1270: CCA6 90 02           BCC L3
1280: CCA8 E6 62           INC ADRESS+1
1290: CCAA 60        L3    RTS
1300:                ;
1310:                ;RECHTSVERSCHIEBUNG
1320:                ;******************
1330:                ;
1340: CCAB 38        RECHTS SEC
1350: CCAC A5 61           LDA ADRESS
1360: CCAE E9 28           SBC #40
1370: CCB0 85 61           STA ADRESS  ;40 ABZIEHEN
1380: CCB2 B0 02           BCS R1
1390: CCB4 C6 62           DEC ADRESS+1
1400: CCB6 A0 28     R1    LDY #40
1410: CCB8 B1 61           LDA (ADRESS),Y ;LINKES BYTE HOLEN
1420: CCBA AA              TAX
1460: CCBB A0 01           LDY #1
1470: CCBD B1 61     R3    LDA (ADRESS),Y
1480: CCBF 48              PHA          ;MERKER 1
1490: CCC0 8A              TXA          ;MERKER 2 HOLEN
1500: CCC1 91 61           STA (ADRESS),Y
1510: CCC3 68              PLA          ;MERKER 1
1520: CCC4 AA              TAX          ;IN MERKER 2
1530: CCC5 C8              INY
1540: CCC6 C0 29           CPY #41
1550: CCC8 D0 F3           BNE R3
1560: CCCA 60              RTS
1570: CCCB 04 04 04 MULH   .BYTE4,4,4,4,4,4,5,5,5,5,5,5
1580: CCD8 06 06 06        .BYTE6,6,6,6,6,6,7,7,7,7,7,7
1590: CCE5 00 28 50 MULL   .BYTE$00,$28,$50,$78,$A0,$C8,$F0
1600: CCEC 18 40 68        .BYTE$18,$40,$68,$90,$B8,$E0
1610: CCF2 08 30 58        .BYTE$08,$30,$58,$80,$A8,$D0,$F8
1620: CCF9 20 48 70        .BYTE$20,$48,$70,$98,$C0,$E8
```

Auch hier wird Ihnen natürlich wieder ein entsprechender Basic-Lader angeboten:

```
100 FOR I = 52224 TO 52480
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,241,183,138, 74,  8, 32,241,183,224, 25,144
130 DATA   2,162, 24,142,246,  7, 32,241,183,224, 25,144
140 DATA   2,162, 24,142,247,  7,138,174,246,  7,172,247
150 DATA   7, 56,237,246,  7,176,  8, 73,255,174,247,  7
160 DATA 172,246,  7,133,254, 40,  8,144,  3,200,152,170
170 DATA 189,203,204,133, 98,189,229,204,133, 97, 40,  8
180 DATA 144,  8,233,  1,133, 97,176,  2,198, 98,165, 98
190 DATA  41,  3,  9,  4, 40,  8, 32,134,204, 40,  8,165
200 DATA  97,144, 10,105, 39,133, 97,144, 12,230, 98,176
210 DATA   8,233, 39,133, 97,176,  2,198, 98,165, 98, 41
220 DATA   3,  9,216, 40,  8, 32,134,204,198,254, 16,206
230 DATA  40, 96,133, 98,144,  3, 76,171,204,160,  0,177
240 DATA  97,170,160, 39,177, 97, 72,138,145, 97,104,170
250 DATA 136, 16,245, 24,165, 97,105, 40,133, 97,144,  2
260 DATA 230, 98, 96, 56,165, 97,233, 40,133, 97,176,  2
270 DATA 198, 98,160, 40,177, 97,170,160,  1,177, 97, 72
280 DATA 138,145, 97,104,170,200,192, 41,208,243, 96,  4
290 DATA   4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5
300 DATA   6,  6,  6,  6,  6,  6,  6,  7,  7,  7,  7,  7
310 DATA   7,  0, 40, 80,120,160,200,240, 24, 64,104,144
320 DATA 184,224,  8, 48, 88,128,168,208,248, 32, 72,112
330 DATA 152,192,232,  0,  0
340 IF S <> 27098 THEN PRINT "FEHLER IN DATAS !!" : END
350 PRINT "OK"
```

Dieses Programm harmonisiert mit dem Graphik - Paket aus Kapitel 4, d.h. beide Maschinenprogramme können sich gleichzeitig im Speicher befinden und auch verwendet werden. Der Aufruf erfolgt, wie gesagt, ebenfalls über SYS und zwar in der wie folgt angegebenen Art und Weise:

```
SYS 512224,r,a,e
```

Dabei bedeuten:

r: Richtung des Scrollens (0=links/1=rechts)

a: Anfangszeile und
e: Endzeile zwischen denen gescrollt wird

Die Anwendung dieser Basicerweiterung sollten Sie dem folgenden kleinen Demonstrationsprogramm entnehmen:

```
100 REM ******************
110 REM **              **
120 REM **  SCROLLING   **
130 REM **              **
140 REM ******************
150 REM
200 SR = 52224 : REM SCROLL-ADRESSE
210 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
220 PRINT:PRINT"  DER SCROLL-BEFEHL ERMOEGLICHT IHNEN,"
230 PRINT"     JEDE BELIEBIGE BILDSCHIRMZEILE"
240 PRINT"  UND BEI BEDARF AUCH MEHRERE GLEICH-"
250 PRINT"         ZEITIG ZU VERSCHIEBEN."
260 PRINT"               SEHEN SIE?"
270 FOR X=1 TO 5000 : NEXT X
280 FOR X=1 TO 40
290 FOR Y=1 TO 50 : NEXT Y
300 SYS SR,1,6,6
310 NEXT X
320 PRINT:PRINT "DAS GANZE GEHT NATUERLICH AUCH SCHNELLER"
330 FOR X=1 TO 4000 : NEXT X
340 FOR X=1 TO 200 : SYS SR,1,8,8 : NEXT X
350 PRINT"          UND ANDERS HERUM!"
360 FOR X=1 TO 4000 : NEXT X
370 FOR X=1 TO 400 : SYS SR,0,10,10 : NEXT X
380 PRINT : PRINT"  VIELLEICHT AUCH DER GANZE BILDSCHIRM"
390 FOR X=1 TO 4000 : NEXT X
400 FOR X=1 TO 200 : SYS SR,0,0,24 : NEXT X
410 PRINT:PRINT"           WOLLEN SIE EINMAL"
420 PRINT"       LAUFSCHRIFTEN ERSTELLEN?"
430 FOR X=1 TO 4000 : NEXT X
440 PRINT:PRINT"        ---- DATA BECKER ----"
450 FOR X=1 TO 400:SYS SR,0,17,17:FOR Y=X TO 150: NEXT Y,X
```

Wie Sie sehen, macht es richtig Spaß mit diesem schönen Befehl zu arbeiten. Denken Sie sich doch einmal andere Anwendungen aus - Sie werden es nicht bereuen!

Wir haben Ihnen nun die wichtigsten Grundlagen für die Programmierung der Spiele vermittelt. Nun ist es an Ihnen, diese in die Tat umzusetzen. Die Phantasie können wir Ihnen nicht abnehmen. Und wenn Sie einmal ein kleines Spiel fertig haben, dann laden Sie mich doch einmal ein.

6. Kapitel
Anhang

6.1 Programmoptimierung

In den einzelnen Kapiteln haben wir Ihnen viele Basicprogramme vorgestellt. Doch stört uns oft Ihre Langsamkeit, was nicht selten schöne Effekte verschleiert. Im folgenden werden Ihnen einige Tips gegeben, wie Sie Ihre Basicprogramme friesieren können.
Man unterscheidet grundsätzlich zwei Methoden der Geschwindigkeits - Aufbesserung:

1.) Das Optimieren des Basicprogrammes selbst
2.) Das Ersetzen von langwierigen Basicroutinen durch entsprechende Assemblerprogramme.

Zum ersten Punkt seien hier einige geraffte Informationen gegeben:

- Vermeiden Sie REM-Zeilen (zumindest oder besonders in oft zu durchlaufenden Schleifen).
- Vermeiden Sie zu viele Zeilen. Oft zu durchlaufende Schleifen etc. sollten in möglichst wenigen Zeilen untergebracht werden (Nutzen Sie die Befehlsabkürzungen, um möglichst viele Befehle in eine Zeile zu bekommen).
- Vermeiden Sie Leerzeichen zwischen oder in den Befehlen, die nicht syntax- oder programmnotwendig sind.
- Verringern Sie den Rechenaufwand in zeitkritischen Schleifen, indem Sie wenn dies möglich ist, Rechnungen oder Teilrechnungen bereits vor Aufruf der Schleife durchführen und die Ergebnisse in Zwischenspeichern bereithalten.
- Vermeiden Sie das direkte Rechnen mit Zahlen (Konstanten) in Schleifen; besser ist, wenn Sie diese vor Aufruf der Schleife in entsprechende Zischenspeicher packen, da Zahlen immer erst in das rechnerinterne Floatingpoint - Format umgerechnet werden müssen.

- Vermeiden Sie Unterprogrammaufrufe (GOSUBs) oder GOTOs in zeitrelevanten Schleifen.
- Konstruieren Sie Schleifen möglichst nur mit FOR...NEXT - Vermeiden Sie IF.
- Definieren Sie viel gebrauchte (vor allem in Schleifen verwendete) Speicher möglichst zuerst. Es genügt z.B. ein X=0 am Programmanfang, wenn Sie diese Variable im Laufe des Programmes sehr häufig verwenden.
- Packen Sie zusammengehörige Programmteile möglichst nahe beieinander, um langes Suchen des Rechners nach der Zeilennummer bei GOTO und GOSUB abzukürzen.
- Legen Sie DATA-Zeilen möglichst zusammen und ebenfalls in möglichst wenigen Zeilen an.

Leider vertragen sich (fast) alle diese Maßnahmen nicht mit der geforderten Übersicht über das Programm und sollten deshalb teilweise möglichst erst dann durchgeführt werden, wenn das Programm 'läuft'. Dieses 'speed up' Ihres Programms sollte also sein letzter Schliff sein.

Der zweite Punkt, das Ausführen von Maschinenprogrammen, ist natürlich etwas schwieriger, stellt aber bei vielen nicht arithmetischen (= nicht mathematischen) Prozessen eine sinnvolle und die effektivste Maßname dar, um Programme zu beschleunigen. Das Maschinenprogramm steht dazu in DATA-Zeilen und wird vor der Ausführung des eigentlichen Basicprogramms durch READ ausgelesen und in den jeweiligen Speicherbereich gePOKEt, in dem das Programm laufen soll, das später durch SYS aufgerufen wird (s. Zeichenformer, Spriteeditor und die vielen Beispiele, die von Assemblerprogrammen unterstützt werden).
Am eindrucksvollsten zeigt sich der Nutzen dieser Technik bei bestimmten graphischen Routinen (s. Graphik-Paket). Hier sollen die in Basic wohl zeitaufwendigsten Arbeiten mit der Graphik kurz durch entsprechende Assemblerroutinen ersetzt werden. Es sind dies: das Löschen der Graphik und die Farbgebung:

Graphik löschen

```
100 FOR I = 51200 TO 51221
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 32,133,254,160,  0,132,253,162, 32,152,145
130 DATA 253,200,208,251,230,254,202,208,246, 96
140 IF S <> 3772 THEN PRINT "FEHLER IN DATAS !!" : END
150 PRINT "OK"
```

Farbe setzen

```
100 FOR I = 51222 TO 51261
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,241,183,134,151,162,  3,169,  4,133,254,160
130 DATA   0,132,253,132,  2,165,151,145,253,200,196,  2
140 DATA 208,249,230,254,202,240,  3, 16,242, 96,162,232
150 DATA 134,  2,208,235
160 IF S <> 5970 THEN PRINT "FEHLER IN DATAS !!" : END
170 PRINT "OK"
```

Diese beiden Einleseroutinen können Sie in Ihre Programme einbauen, wenn Sie sich nicht die Mühe gemacht haben, das Graphik-Aid in Ihren Rechner zu tippen. Die Syntax der beiden neuen Graphikbefehle lautet:

```
SYS 51200 : REM GRAPHIK LOESCHEN
SYS 51222,16*PF+HF : REM FARBE SETZEN
```

Dabei bedeuten:

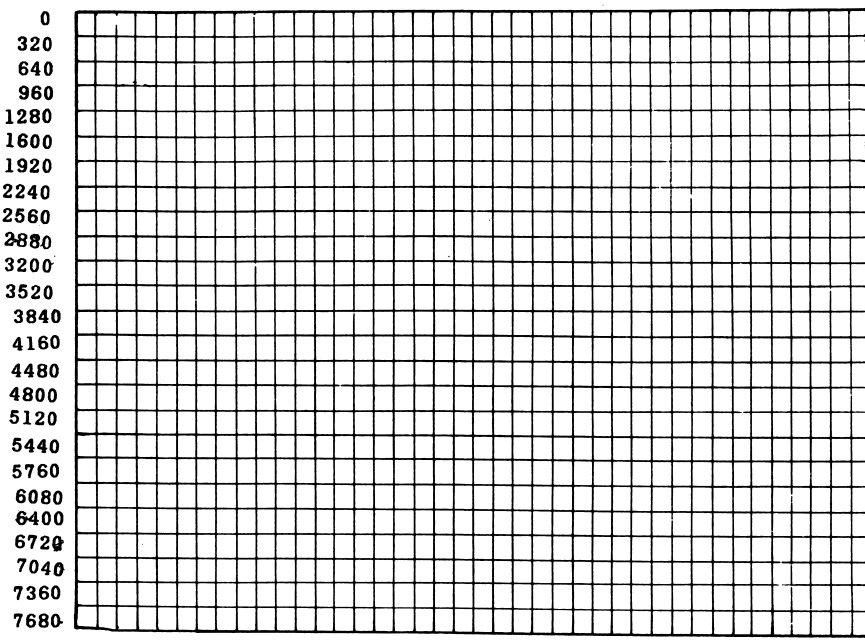
PF: Punktfarbe
HF: Hintergrundfarbe

Wie sie anzuwenden sind, zeigen Ihnen die Erläuterungen in Paragraph 4.7.

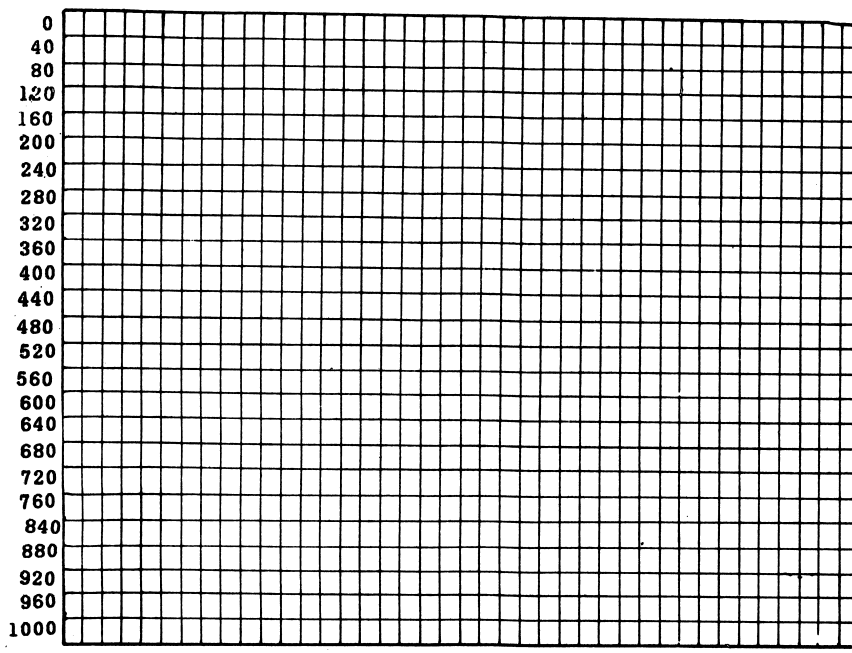
6.2 Graphikspeicheraufbau

Im folgenden wird Ihnen der Aufbau der drei zweit wichtigsten Graphik - Speicherfunktionen dargestellt: Videoram und Graphikspeicher. Zu den am Rand angegebenen relativen Adressen der jeweiligen Zeilenanfänge müssen Sie stets noch die Basisadresse hinzuzählen. Diese hängt von der Lage des Speichers ab und lautet nach dem Einschalten für den Videoram: 1024 ($0400). Der Farbram besitzt den gleichen Aufbau wie der Videoram und hat die Basisadresse: 55296 ($D800).
Bei dem Schaubild für den Graphikspeicher müssen Sie jeden Kasten noch einmal in 8x8 Kästchen unterteilen, um die einzelnen Punkte zu symbolisieren. Den genauen Aufbau entnehmen Sie bitte dem Kapitel 3.

6.2.1. Graphikspeicher



6.2.2. Videoram



6.3 Farbtabelle

Der Commodore 64 besitzt in allen verfügbaren Modi 16 Farben. Jeder dieser Farben ist ein sogenannter Farbcode zugeordnet, der in die entsprechenden Register gePOKEt wird, die für Farbangelegenheiten zuständig sind. Gleichzeitig können Sie aber auch alle 16 Farben im Textmodus für das Aussehen der Zeichen von Hand (Tastatur) aus anwählen. Dementsprechend gibt es für jede Farbe einen ASCII-Code, der die Bestimmung der Textfarbe auch von Programmen aus ermöglicht. In PRINT-Statements erscheinen die typischen Graphikzeichen. Die folgende Tabelle vereinigt alle Ansteuerungsmöglichkeiten übersichtlich zum schnellen Nachschlagen.

Taste	ASCII		Ausgabe	Codes		Farbe
	Dez	Hex		Dez	Hex	
<ctrl> 1	144	$90		00	$00	schwarz
<ctrl> 2	005	$05		01	$01	weiß
<ctrl> 3	028	$1C		02	$02	rot
<ctrl> 4	159	$9F		03	$03	zyanblau
<ctrl> 5	156	$9C		04	$04	violett
<ctrl> 6	030	$1E		05	$05	grün
<ctrl> 7	031	$1F		06	$06	blau
<ctrl> 8	158	$9E		07	$07	gelb
<C=> 1	129	$81		08	$08	orange
<C=> 2	149	$95		09	$09	braun
<C=> 3	150	$96		10	$0A	hellrot
<C=> 4	151	$97		11	$0B	dunkelgrau
<C=> 5	152	$98		12	$0C	mittelgrau
<C=> 6	153	$99		13	$0D	hellgrün
<C=> 7	154	$9A		14	$0E	hellblau
<C=> 8	155	$9B		15	$0F	hellgrau

6.4 Bildschirmcodes

Jedem Zeichen des Bildschirms ist ein bestimmter ASCII-Code, aber auch ein Bildschirmcode zugeordnet. Letzterer ist der Code, mit dem das Zeichen im Videoram abgespeichert wird. Wollen Sie also ein Zeichen direkt in diesen Textspeicher POKEn, so verwenden Sie jene Werte. Die Codes für die inversen Zeichen erhalten Sie, indem Sie 128 zu denen der normalen Zeichen hinzuaddieren.

Codes	ASCII	Zeichen	
		Satz1	Satz2
0	64	@	@
1	65	A	a
2	66	B	b
3	67	C	c
4	68	D	d
5	69	E	e
6	70	F	f
7	71	G	g
8	72	H	h
9	73	I	i
10	74	J	j
11	75	K	k
12	76	L	l
13	77	M	m
14	78	N	n
15	79	O	o
16	80	P	p
17	81	Q	q
18	82	R	r
19	83	S	s
20	84	T	t
21	85	U	u
22	86	V	v
23	87	W	w
24	88	X	x
25	89	Y	y
26	90	Z	z
27	91	[	[
28	92	£	£
29	93	]	]
30	94	↑	↑
31	95	←	←
32	32		
33	33	!	!
34	34	"	"
35	35	#	#
36	36	$	$
37	37	%	%
38	38	&	&
39	39	'	'

Codes	ASCII	Zeichen	
		Satz1	Satz2
40	40	(	(
41	41	)	)
42	42	*	*
43	43	+	+
44	44	,	,
45	45	-	-
46	46	.	.
47	47	/	/
48	48	0	0
49	49	1	1
50	50	2	2
51	51	3	3
52	52	4	4
53	53	5	5
54	54	6	6
55	55	7	7
56	56	8	8
57	57	9	9
58	58	:	:
59	59	;	;
60	60	<	<
61	61	=	=
62	62	>	>
63	63	?	?
64	96	─	─
65	97	♠	A
66	98	│	B
67	99	▁	C
68	100	▔	D
69	101	▏	E
70	102	▕	F
71	103	▁	G
72	104	▔	H
73	105	╱	I
74	106	╲	J
75	107	╮	K
76	108	╰	L
77	109	╯	M
78	110	╲	N
79	111	╭	O

80	112	┐	P
81	113	●	Q
82	114	│	R
83	115	●	S
84	116	─	T
85	117	─	U
86	118	╲	V
87	119	╳	W
88	120	o	X
89	121	↑	Y
90	122	─	Z
91	123	◆	+
92	124	+	▒
93	125	▒	─
94	126	▒	▒
95	127	◢	▒
96	160	▒	▒
97	161	■	▒
98	162	■	▒
99	163	▒	▒
100	164	▒	▒
101	165	▒	▒
102	166	▒	▒
103	167	▒	▒

104	168	▒	▒
105	169	◥	▒
106	170	─	▒
107	171	┼	▒
108	172	■	▒
109	173	┌	▒
110	174	┌	▒
111	175	■	▒
112	176	┐	▒
113	177	┼	▒
114	178	┼	▒
115	179	┼	▒
116	180	─	▒
117	181	─	▒
118	182	▒	▒
119	183	▒	▒
120	184	▒	▒
121	185	┐	▒
122	186	▪	▒
123	187	▪	▒
124	188	┐	▒
125	189	▪	▒
126	190	┐	▒
127	191	▪	▒

6.5 Dez/Hex/Dual-Konversion

		%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%
		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
		0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
%0000 0000	0	$00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
%0001 0000	16	$10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
%0010 0000	32	$20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
%0011 0000	48	$30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
%0100 0000	64	$40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
%0101 0000	80	$50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
%0110 0000	96	$60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
%0111 0000	112	$70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
%1000 0000	128	$80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
%1001 0000	144	$90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
%1010 0000	160	$A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
%1011 0000	176	$B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
%1100 0000	192	$C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
%1101 0000	208	$D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
%1110 0000	224	$E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
%1111 0000	240	$F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

6.6 Spriteentwurfsblatt

Mit Hilfe dieses Formblattes ist es Ihnen möglich auf einfache Art und Weise Sprites zu erstellen, indem Sie die einzelnen Kästchen, die jeweils einen Punkt darstellen, ausmalen und die oben stehenden Werte zu einem Byte (drei Bytes pro Zeile) zusammenzählen. Multicolorsprites erstellen Sie, indem Sie für einen Punkt zwei nebeneinander liegende Kästchen (möglichst die 3 verschiedenen Multicolorfarben auch in drei verschiedenen Farben zeichnen) ausmalen.

Bit:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
Wert:	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	2	1	0	Codes		
0																											
1																											
2																											
3																											
4																											
5																											
6																											
7																											
8																											
9																											
10																											
11																											
12																											
13																											
14																											
15																											
16																											
17																											
18																											
19																											
20																											

6.7 Zeichenentwurfsblatt

Ebenso wie beim Spriteentwurfsblatt können Sie auch mit der folgenden Tabelle Multicolorzeichen erstellt werden, indem Sie für einen Punkt jeweils zwei Kästchen ausmahlen.

Bit:	7	6	5	4	3	2	1	0			
Wert:	128	64	32	16	8	4	2	1	Codes		
7											
6											
5											
4											
3											
2											
1											
0											

6.6 VIC-Register-Übersicht

Register		Adresse		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dez	Hex	Dez	Hex	128	64	32	16	8	4	2	1
00	$00	53248	$D000	Sprite 0 --- x - Koordinate (Bits 0-7)(0-255)							
01	$01	53249	$D001	Sprite 0 --- y - Koordinate (0-255)							
02	$02	53250	$D002	Sprite 1 --- x - Koordinate (Bits 0-7)(0-255)							
03	$03	53251	$D003	Sprite 1 --- y - Koordinate (0-255)							
04	$04	53252	$D004	Sprite 2 --- x - Koordinate (Bits 0-7)(0-255)							
05	$05	53253	$D005	Sprite 2 --- y - Koordinate (0-255)							
06	$06	53254	$D006	Sprite 3 --- x - Koordinate (Bits 0-7)(0-255)							
07	$07	53255	$D007	Sprite 3 --- y - Koordinate (0-255)							
08	$08	53256	$D008	Sprite 4 --- x - Koordinate (Bits 0-7)(0-255)							
09	$09	53257	$D009	Sprite 4 --- y - Koordinate (0-255)							
10	$0A	53258	$D00A	Sprite 5 --- x - Koordinate (Bits 0-7)(0-255)							
11	$0B	53259	$D00B	Sprite 5 --- y - Koordinate (0-255)							
12	$0C	53260	$D00C	Sprite 6 --- x - Koordinate (Bits 0-7)(0-255)							
13	$0D	53261	$D00D	Sprite 6 --- y - Koordinate (0-255)							
14	$0E	53262	$D00E	Sprite 7 --- x - Koordinate (Bits 0-7)(0-255)							
15	$0F	53263	$D00F	Sprite 7 --- Y - Koordinate (0-255)							

Register		Adresse		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dez	Hex	Dez	Hex	128	64	32	16	8	4	2	1
16	$10	53264	$D010	Sprite 0-7 --- x - Koordinaten (Bit 8)(*256)							
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
17	$11	53265	$D011	Raster Bit 8	extend. Color	Hi-Res-Graphik	Bild aus	25/24 Zeilen	y-Scrolling		
18	$12	53266	$D012	Rasterzeile (aktuelle/Vorgabe)(Bits 0-7)(0-255)							
19	$13	53267	$D013	Lightpen --- x - Koordinate (0-255)							
20	$14	53268	$D014	Lightpen --- y - Koordinate (0-255)							
21	$15	53269	$D015		Sprite --- an / aus						
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
22	$16	53270	$D016	nicht benutzt			Multi-color	40/38 Spalten	x-Scrolling		
23	$17	53271	$D017	Sprite --- y - Vergrößerung							
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
24	$18	53272	$D018	Bildschirmspeicheradr. (*1024)				Zeichengen.adr.(*2048)			nicht benutzt
				Bit 13	Bit 12	Bit 11	Bit 10	Bit 13	Bit 12	Bit 11	
25	$19	53273	$D019	IRQ-Flag	nicht benutzt			Light-pen	Sp.-Sp. Kollis.	Sp.-Hg. Kollis.	Raster zeilen
26	$1A	53274	$D01A	nicht benutzt				Light-pen	Sp.-Sp. Kollis.	Sp.-Hg. Kollis.	Raster zeilen
27	$1B	53275	$D01B		Sprite - Hintergrund - Priorität						
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
28	$1C	53276	$D01C		Multicolor - Sprites						
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
29	$1D	53277	$D01D		Sprite --- x - Vergrößerung						
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0

Register		Adresse		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dez	Hex	Dez	Hex	128	64	32	16	8	4	2	1
30	$1E	53278	$D01E	Sprite - Sprite - Kollision							
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
31	$1F	53279	$D01F	Sprite - Hintergrund - Kollision							
				Sp. 7	Sp. 6	Sp. 5	Sp. 4	Sp. 3	Sp. 2	Sp. 1	Sp. 0
32	$20	53280	$D020	Rahmenfarbe (0-15)							
33	$21	53281	$D021	Hintergrundfarbe Nr. 0 (0-15)							
34	$22	53282	$D022	Hintergrundfarbe Nr. 1 (0-15)							
35	$23	53283	$D023	Hintergrundfarbe Nr. 2 (0-15)							
36	$24	53284	$D024	Hintergrundfarbe Nr. 3 (0-15)							
37	$25	53285	$D025	Gemeinsame Sprite-Farbe Nr. 0 in Multicolor (0-15)							
38	$26	53286	$D026	Gemeinsame Sprite-Farbe Nr. 1 in Multicolor (0-15)							
39	$27	53287	$D027	Farbe für Sprite Nr. 0 (0-15)							
40	$28	53288	$D028	Farbe für Sprite Nr. 1 (0-15)							
41	$29	53289	$D029	Farbe für Sprite Nr. 2 (0-15)							
42	$2A	53290	$D02A	Farbe für Sprite Nr. 3 (0-15)							
43	$2B	53291	$D02B	Farbe für Sprite Nr. 4 (0-15)							
44	$2C	53292	$D02C	Farbe für Sprite Nr. 5 (0-15)							
45	$2D	53293	$D02D	Farbe für Sprite Nr. 6 (0-15)							
46	$2E	53294	$D02E	Farbe für Sprite Nr. 7 (0-15)							

6.9 Literaturhinweise

Commodore 64:

Sally Greenwood Larsen
Sprite Graphics for the Commodore 64
Micro Text Publications
ISBN: 0-13-838144-5

Compute!'s First Book of Commodore 64
Compute! Books Publication
ISBN: 0-942386-20-5

C. Lorenz
Beherrschen Sie Ihren Commodore 64
Hofacker Verlag
ISBN: 3-88963-147-9

Stan Krute
Commodore 64 Graphics & Sound Programming
Tab Books Inc.
ISBN: 0-8306-0140-6

Shaffer
Commodore 64 Color Graphics: A Beginners Guide
The Book Company
ISBN:0-912003-06-5

Angerh./Brück./Eng./Gerits
64 intern
DATA BECKER GmbH
ISBN: 3-89011-000-2

Graphik-Fachbücher:

Faux/Pratt
Computational Geometry for Design and Manufacture
Ellis Horwood Limited
ISBN: 0-85312-114-1

Hearn/Baker
Computer Graphics for the IBM Personal Computer
Prentice-Hall, Inc.
ISBN: 0-13-164335-5

Encarnacao
Computer Aided Design-Modelling, Systems Engineering,
CAD-Systems
Springer-Verlag
ISBN: 0-387-10242-6

Encarncao
Computer-Graphics - Programmierung und Anwendung von
graphischen Systemen
R.Oldenbourg Verlag
ISBN: 3-486-34651-2

Brodlie
Mathematical Methods in Computer Graphics and Design
Academic Press
ISBN: 0-12-134880-6

Barnhill/Boehm
Surfaces in Computer Aided Geometric Design
North-Holland
ISBN: 0-444-86550-0

Myers
Microcomputer Graphics
Addison-Wesley Publishing Company
ISBN: 0-201-05092-7

6.10 Nachtrag zu Abschnitt 4.1

Im folgenden wird das in Abschnitt 4.1 aus technischen Gründen fehlende Basicprogramm nachgetragen:

```
100 REM *************************
110 REM **                     **
120 REM **  LOW-GRAPHIK/PRINT   **
130 REM **                     **
140 REM *************************
150 REM
160 PRINT CHR$(147) : PRINT : PRINT : REM BILDSCHIRM LOESCHEN/LEERZEILEN
170 PRINT"  \ /                       ": REM M,N
180 PRINT" \^            ~            ": REM M,U,I,U,K
190 PRINT"   v\                       ": REM J,K,M
200 PRINT" / \         ~              ": REM N,M,U,K
210 PRINT"             -              ": REM P
220 PRINT"          ___|_|___         ": REM P,P,P,L,L,P,P
230 PRINT"         /\       \         ": REM N,M,M
240 PRINT"        /  \      |         ": REM N,M,L
250 PRINT"       /    \     |         ": REM N,M,L
260 PRINT"       ┌────┐    /          ": REM O,Y,Y,Y,Y,P,N
270 PRINT"       | ┌┐┌┐|  /           ": REM H,A,S,N,N
280 PRINT"       | └┘└┘| /   ┌┐┌┐     ": REM H,Z,X,N,N,A,S,U,I
290 PRINT"       | ┌─┐ |/   _|_|_\    ": REM H,O,P,N,N,U,E,E,K,J,I
300 PRINT"       | ┌ | |/  --o--o-    ": REM H,O,N,N,N,-,E,W,C,C,W,E
310 PRINT"       ──┬────   ┌──────────": REM Y,Y,O,Y,Y,Y,O,Y,Y,...
320 PRINT"         └──────┘           ": REM L,P,P,P,P,P,@
```

PROFIMAT IN STICHWORTEN

PROFIMAT ist ein leistungsfähiges Programmpaket zur professionellen Maschinenprogrammierung mit dem COMMODORE 64

- PROFI-MAT
- PROFI-MON
- 2 K-Byte Maschinenprogramm
- Register- und Speicherinhalte anzeigen/ändern
- Maschinenprogramme laden, speichern, ausführen, disassemblieren
- Speicherbereiche durchsuchen, vergleichen, füllen
- PROFI-ASS
- 4 K-Byte Maschinenprogramm
- formatfreie Eingabe
- komplette Assemblerlistings
- ladbare Symboltabellen
- bedingte Assemblierung und Assemblerschleifen
- Verkettung von Quellprogrammen
- umfangreiche Pseudo-Opcodes
- vielfältige arithmetische und logische Operationen
- arbeitet mit beliebiger Gerätekonfiguration
- Diskettenprogramm
- mit ausführlichem Handbuch

PASCAL 64 IN STICHWORTEN

PASCAL 64, DAS IDEALE PASCAL FÜR EINSTEIGER ZUM COMMODORE 64

- erzeugt echte Maschinensprache
- ca. 22 K frei für Programm und Symboltabelle
- einfache Programmerstellung durch Bildschirmeditor
- enthält alle wichtigen Methoden zur Programmstrukturierung

Standard PASCAL-Features:
- BEGINN . . . END
- WHILE . . . DO
- REPEAT . . . UNTIL
- CASE
- PROCEDURE
- FUNKTION

Datentypen:
- CHAR
- INTEGER
- REAL
- ARRAY

Sonderfunktionen:
- PEEK
- SYS
- Graphikbefehle
- Behandlung sequentieller Dateien
- einbinden externer Prozeduren und Funktionen von Diskette
- lieferbar auf Diskette
- mit detailliertem Handbuch

SYNTHIMAT IN STICHWORTEN

SYNTHIMAT ist ein leistungsfähiger, polyphoner Synthesizer für den COMMODORE 64

- drei Oszillatoren (VCOs) mit 7 Fußlagen und 8 Wellenformen
- drei Hüllkurvengeneratoren (ADSRs)
- ein Filter (VCF) mit 8 Betriebsarten und Resonanzregulierung
- VCF mit Eingang für externe Signalquelle
- ein Verstärker (VCA)
- Ringmodulation mit allen drei VCOs
- 8 softwaremäßig realisierte Oszillatoren (LFOs)
- kräftiger Klang durch polyphones Spielen
- zwei Manuale (Solo und Begleitung)
- speichern von bis zu 256 Klangregistern
- schneller Registerwechsel
- speichern von 9 Registerdateien Diskette
- „Bandaufnahme" auf Diskette durch direktes Spielen
- keine lästige Noteneingabe
- speichern von bis zu 9 „Bandaufnahmen" je Diskette
- integrierte 24 Stunden-Echtzeituhr
- einstellbares PITCH-BENDING
- farblich gekennzeichnete, übersichtlich angeordnete Module
- umfangreiches Handbuch
- läuft mit einem Diskettenlaufwerk
- Diskettenprogramm

DISKOMAT IN STICHWORTEN

DISKOMAT, ein Programmpaket mit SUPERTWIN, DISK-MONITOR und DISK-BASIC zum COMMODORE 64.

- SUPERTWIN, ein Steuerprogramm, mit dem Sie zwei VC 1541 Diskettenlaufwerke wie ein Doppellaufwerk benutzen. Kopieren von Dateien und zwischen zwei Laufwerken Backup-Möglichkeit. Kompatibel zu Programmen, die für Doppellaufwerke gedacht sind.
- DISK-BASIC bietet Ihnen die komfortablen Diskettenbefehle des BASIC 4.0.

APPEND	BACKUP	CATALOG	COLLECT
CONCAT	COPY	DOPEN	DCLOSE
DLOAD	DSAVE	HEADER	RECORD
SCRATCH	DIRECTORY	RENAME	DS & DS $

- DISK-MONITOR
- Lesen eines Blocks von Diskette
- Anzeige und Ändern eines Blocks am Bildschirm
- Schreiben eines Blocks auf Diskette
- Senden von Diskettenbefehlen
- Anzeigen der Fehlermeldung der Diskette
- Diskettenprogramm mit ausführlichem deutschen Handbuch

SUPERGRAPHIK IN STICHWORTEN

SUPERGRAPHIK 64 ist ein umfassendes Graphikprogramm für den COMMODORE 64

— 2 unabhängige, hochauflösende Graphik-Seiten (320 x 200 Punkte)
— 1 Standard Low-Graphik-Seite (80 x 50 Punkte)
— Normalfarben-Graphik (20 x 200 Punkte)
— Multicolor-Graphik (160 x 200 Punkte)
— verdecktes Zeichnen (z. B. Text sichtbar, Graphikseite 2 wird erstellt)
— 183 Befehle und Befehlskombinationen:

1. Für jeden Befehl wählbare Zwischenmodi: Zeichnen, Löschen, Punktieren, Graphik-Cursor bewegen, Zeichnen mit/ohne Farbsetzung, Punkte zählen
2. Durch einfache Befehle zu steuernde Graphikfiguren: Punkt, Linie, Linienschar, Linie vom Graphik-Cursor, Kreise, Kreisbögen, Ellypse, Ellypsenbögen, selbstdeffinierbare Figuren, sortieren und vergrößern dieser Figuren, Rahmen, Feld, Text in Graphik
3. Weitere Graphik-Befehle: Graphikseiten- und Moduswechsel, Graphik löschen, Graphik invertieren, Scrolling von Text und Graphik, Wählen der Rahmen-, Hintergrund-, Zeichen- oder Punktfarbe

— Speichern, Laden von Graphik (auch verdeckt)
— Hardcopies für EPSON, CBM 1526, Seikosha GP 100 VC, Farb(!)drucker Seikosha GP 700 und andere mit DATA BECKER Interface
— 8 Sprites definiert durch einen Befehl!
— alle Sprite-Eigenschaften veränderbar
— Positionieren und Bewegen (!) von 8 Sprites gleichzeitig und unabhängig voneinander, während das übrige Programm weiterläuft (IRQ)
— Sprite-Kollisionsüberprüfung, Joystickunterstützung
— automatische Unterbrechung des BASIC-Programms bei Kollisionen (Interrupt), Sprung in Unterbrechungsroutine, dann Weiterführung des Hauptprogramms
— komfortable Soundprogrammierung mit Verstellung aller möglichen Sound-Parameter (Laufstärke, Klang, Filter, Tonhöhe, Tonlänge) ebenfalls unabhängig vom übrigen Programmablauf
— umfangreiche Anleitung
— Diskettenprogramm

DATAMAT IN STICHWORTEN

DATAMAT ist ein komfortables Dateiverwaltungsprogramm für den COMMODORE 64

— menuegesteuert, dadurch extrem einfach zu bedienen
— für jede Art von Daten zu gebrauchen
— völlig frei gestaltbare Eingabemaske
— 50 Felder pro Datensatz
— 253 Zeichen pro Datensatz
— je nach Umfang der Datensätze bis zu 2000 pro Datei
— Schnittstelle zu TEXTOMAT
— lieferbar als Diskettenprogramm mit ausführlichem deutschen Handbuch
— läuft mit ein oder zwei Floppies

Sie können:

— nach beliebigen Feldern selektieren
— nach allen Feldern gleichzeitig sortieren
— Listen in völlig freiem Format drucken
— Etiketten drucken
— mit jedem COMMODORE Drucker arbeiten
— nahezu jeden Fremddrucker verwenden (mit DATA BECKER Interface)

TEXTOMAT IN STICHWORTEN

TEXTOMAT, ein außergewöhnliches Textverarbeitungsprogramm für den COMMODORE 64

— Diskettenprogramm
— mit umfangreichem deutschen Handbuch
— durchgehend menuegesteuert
— deutscher Zeichensatz auch auf Commodore-Druckern
— Rechenfunktionen
— 24000 Zeichen pro Text
— beliebig lange Texte durch Verknüpfung
— frei programmierbare Steuerzeichen
— horizontales Scrolling für 80 Zeichen pro Zeile
— läuft mit ein oder zwei Floppies
— Formularsteuerung für Randeinstellung usw.
— komplette Bausteinverarbeitung
— Blockoperationen, Suchen und Ersetzen
— Serienbriefschreibung mit DATAMAT
— formatierte Ausgabe auf Bildschirm
— an fast jeden Drucker anpassbar

STRUKTO 64 IN STICHWORTEN

STRUKTO 64 ist eine fantastische neue Programmiersprache für strukturiertes Programmieren mit COMMODORE 64.

— Interpretersprache, die die Vorzüge von BASIC UND PASCAL vereint
— strukturiertes Programmieren
— übersichtliche Programme
— leichte Erlernbarkeit
— einfache Bedienung
— eingebautes Toolkit erleichtert das Eingeben und Verbessern von Programmen
— leichteres Arbeiten mit der Floppy
— Sprite-Editor ermöglicht das Einlesen der Sprite-Formen direkt vom Bildschirm
— Graphikbedienung wird mit gut durchdachten Befehlen unterstützt
— Abspielen von Musik ist unabhängig vom Programmablauf möglich
— ca. 80 neue Befehle
— lieferbar als Diskettenprogramm
— ausführliches, deutsches Handbuch

STRUKTO 64 ist für alle Programmierer geeignet, die den COMMODORE 64 als Allround-Computer einsetzen wollen und auf einfache Weise anspruchsvolle Programme erstellen wollen.

KONTOMAT IN STICHWORTEN

KONTOMAT ein Diskettenprogramm zur Einnahme-/Überschußrechnung

- professionelles Programm, angelehnt an eine echte Finanzbuchhaltung
- maximal 120 definierbare Konten
- 4 Mehrwert- und Vorsteuerkennzeichen
- intervallmäßige Belegeingabe
- 4 verschiedene Buchungsarten (SOLL/HABEN, HABEN/SOLL, SOLL, HABEN)
- Anzeige der SOLL- und HABEN-Summe bei mehrfachen Buchungssätzen
- komfortable Belegeingabe mit Datum, Belegnummer, Buchungstext, Steuerkennzeichen, Buchungsart und Betrag
- Druck des Journals mit der Belegeingabe
- Druck von Kontenblättern
- Druck einer Summen- und Saldenliste mit Monats- und Jahresumsatzsummen
- betriebswirtschaftliche Auswertung mit Druckausgabe
- Verzeichnis der AfA und der Sofortabschreibung
- ein oder zwei Laufwerke
- umfangreiches Handbuch

FAKTUMAT IN STICHWORTEN

FAKTUMAT ist ein Programm zur Fakturierung und Rechnungserstellung für den COMMODORE 64

- voll menuegesteuert
- Artikeldatei & Kundendatei parameterisiert
- läuft auf 1 oder 2 Floppies 1541
- Drucker: VC-1525, VC-1526, EPSON RX-/FX-80 mit DATA BECKER Interface
- Diskettenprogramm
- frei definierbarer Firmenkopf
- 4 wählbare Mehrwertsteuersätze
- Maximal 950 Artikel bei 50 Kunden oder 50 Artikel bei 950 Kunden. Die Summe von Kunden + Artikel kann maximal 1000 betragen
- Druck von Rechnung und Lieferschein möglich
- Lagerbuchführung ist integriert. Alle Artikel die über Rechnung/Lieferschein rausgegangen sind, werden in der Lagerdatei abgebucht.
- Rechnungsbeträge werden in der Kundendatei festgehalten
- deutsches, detailliertes Handbuch
- Sie arbeiten mit einer Diskette für Ihre Daten

PAINT-PIC IN STICHWORTEN

PAINT-PIC ist ein faszinierendes Malprogramm für den COMMODORE 64

- Programmsteuerung: Tastatur
- Steuerung des Stifts: Cursortasten und eckige Klammer (diag.) (Joystick kann benutzt werden)
- Routinen: Linien, Rechtecke, Dreiecke, Parallelogramme, Kreise, Kreisbögen, Ellipsen, Bestimmung v. Mittelpunkt, u. perspektivischer Linie, Kopieren u. drehen v. Teilbildern, Verdoppeln, halbieren u. spiegeln v. Teilbildern
- Modi: Malstiftmodus (schmale Linie) Pinselmodus (8 verschiedene Breiten) (Art der Linie selbst definierbar)
- Textmodus (Kompl. Zeichensatz Commodore) (Hoch-Tiefschrift)
- Speichern: Teilbilder (Blöcke) oder ganze Bilder
- Menue: 1 Hauptmenue mit 8 Untermenues
- mit ausführlichem deutschen Handbuch
- Diskettenprogramm
- Bilder kann man auf Diskette oder Cassette abspeichern

MASTER 64 IN STICHWORTEN

MASTER: Professionelles Programmentwicklungssystem für COMMODORE 64

- 70 zusätzliche Befehle
- Bildschirmmaskengenerator
- Definieren von Bildschirmzonen
- Eingabe aus Zonen
- Formatierte Ausgabe
- Abspeicherung von Bildschirminhalten
- Arbeiten mit mehreren Bildschirmmasken
- ISAM Dateiverwaltung
- Datensätze bis zu 254 Zeichen
- Schlüssellänge bis zu 30 Zeichen
- Dateigröße nur von Diskettenkapazität abhängig
- Zugriff über Schlüssel und Auswahlmasken
- Druckmaskengenerator
- Erstellung beliebiger Formulare und Ausgabemasken
- BASIC-Erweiterungen
- Toolkitfunktionen
- Mehrfachgenaue Arithmetik Rechnen mit 22 Stellen Genauigkeit