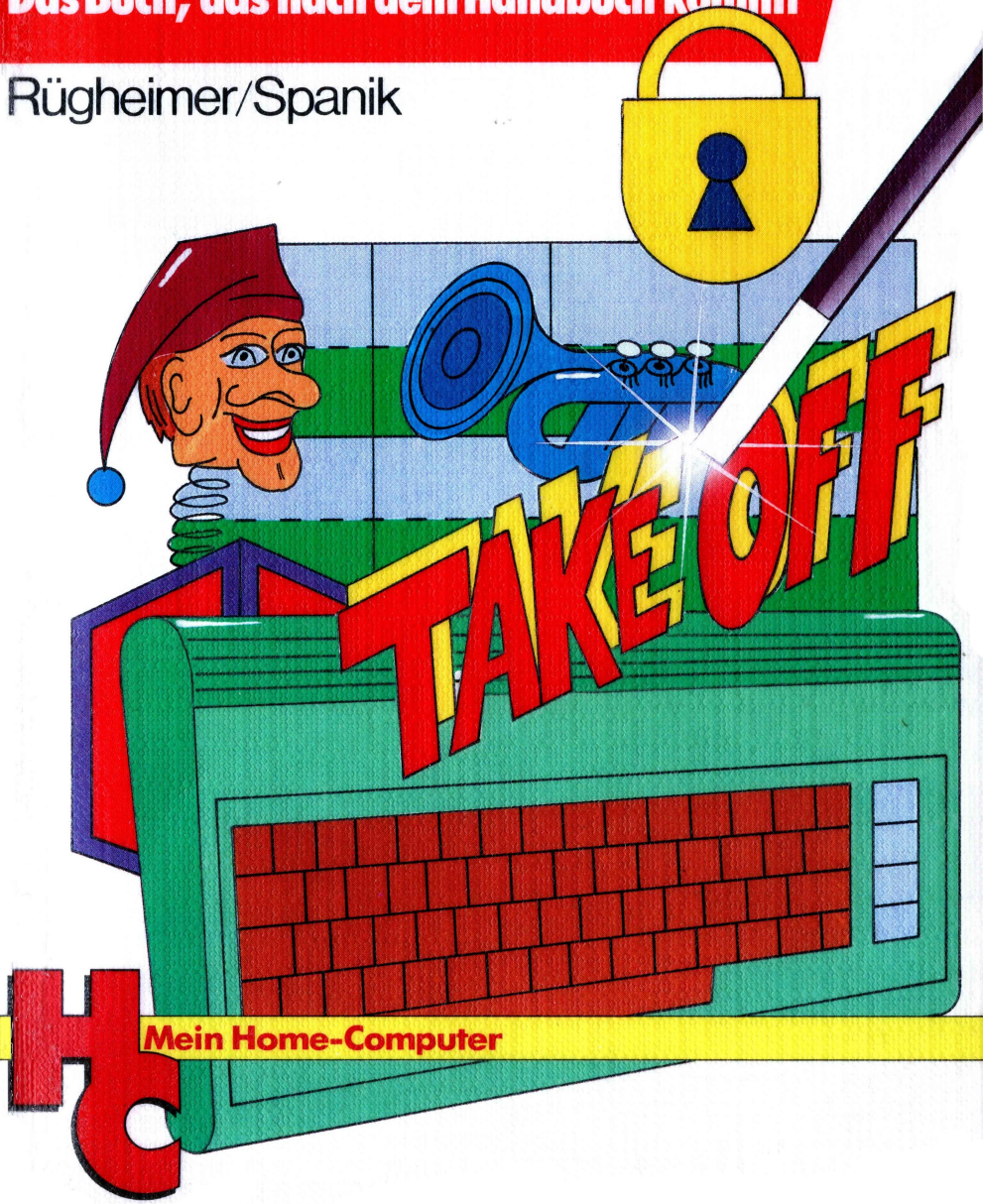
Mein zweites Commodore 64-Buch

Das Buch, das nach dem Handbuch kommt

Rügheimer/Spanik



Mein Home-Computer

Hannes Rügheimer, Christian Spanik
Mein zweites Commodore-64-Buch

Inhaltsverzeichnis

Vorwort

Einer der wichtigsten Punkte beim Schreiben dieses Buches war, daß es nicht nur informativ sein soll. Seine Aufgabe ist es auch, möglichst einfach und unterhaltsam die Fähigkeiten Ihres Commodore 64 zu erklären. Es soll Sie informieren, Ihnen Nützliches, Praktisches zeigen. In einem Satz: Es soll benutzerfreundlich sein. Diesen Effekt möchten wir mit zwei Dingen erreichen: dem Stil, in dem die folgenden Seiten abgefaßt sind, und den Beispielen, die Sie am Ende dieses Buches finden werden.

Wir haben alle Beispielprogramme so kurz wie möglich gehalten. Dies hat mehrere Gründe: Zum einen tippen nur wenige Leute gern lange Listings ab – außerdem will man ja möglichst schnell Ergebnisse vorweisen können. Zum anderen: Dies ist kein Tippbuch. Es gibt einfachere Wege, sich die Augen zu verderben, als nächtelanges Lesen kleingedruckter Listings. Uns ist es wichtig, daß Sie die Programmstruktur verstehen. Das ist sicher der beste Weg, den Umgang mit Ihrem Commodore 64 zu lernen. Unsere Beispiele sind auch nicht übermäßig komfortabel. Alle Schnörkel wurden weggelassen, damit die Programme möglichst übersichtlich bleiben. Und noch etwas: Wenn es Ihnen irgendwann zu lästig wird, dasselbe Programm immer wieder mit RUN zu starten, werden Sie von selbst anfangen, es zu verändern. Und genau das wollen Sie ja letztlich lernen: selbst programmieren. Sie sollen ja experimentieren, die Möglichkeiten des Computers selbst herausfinden. Denn eins kann Ihnen kein Buch ersetzen: die eigene Erfahrung mit Ihrem System. Natürlich haben wir die Programme – soweit möglich – etwas amüsant gestaltet. Daß das nicht immer gelingen kann, liegt in der Natur der Sache. Manche Dinge sind halt einfach nicht lustig.

Voraussetzung für die Arbeit mit diesem Buch ist, daß Sie das zum Commodore 64 mitgelieferte Handbuch zumindest durchgelesen und das

eine oder andere Beispiel ausprobiert haben. Unser Buch bietet zwar eine Einführung ins Commodore-BASIC, versteht sich aber nicht als BASIC-Kurs. Deshalb sollten Sie auch das Commodore-Handbuch möglichst immer in der Nähe haben. Ein paar Tips noch zu unserem Buch: Wir raten Ihnen, nicht zwischen einzelnen Kapiteln zu springen oder Kapitel auszulassen. Wichtig ist auch, die kurzen Zusammenfassungen jedes Kapitels im Anhang zu lesen. Diese Zusammenfassungen haben verschiedene Gründe. Sie sollen einerseits das gezielte Suchen nach Informationen vereinfachen und so helfen, Informationen, die schnell gebraucht werden, rasch wieder aufzufrischen. Vor allem aber wollen wir Sie in diesem «Buch im Buch» auf Nachfolgelektüre und die Fachzeitschriften vorbereiten. Denn diese Minikapitel sind in erster Linie technische Texte, die auch übliche Fachausdrücke verwenden.

Anhand des Stichwortverzeichnisses und des zugehörigen ausführlichen Kapitels können Sie sich die Textinhalte relativ einfach erschließen, sammeln so einen guten Schatz an Fachausdrücken und sind entsprechenden Texten nicht mehr hilflos ausgeliefert.

Wenn Sie jetzt das Gefühl haben, dieses Buch ist genau das, was Sie suchen: fein. Wenn nicht? Nun, dann hat es sich in jedem Fall gelohnt, diesen einführenden Abschnitt zu lesen. Wenn er auch – zugegeben – etwas trocken ist. Aber wir meinten, daß das so sein muß. In jedem Fall wünschen wir Ihnen viel Spaß mit Ihrem Commodore 64.

Die Macher

Okay, liebe zukünftige Programmierer. Es ist anzunehmen, daß wir jetzt unter uns sind. Deshalb sollten wir uns vielleicht erst einmal kurz bekannt machen. Jedem, der nicht daran interessiert ist, wer hinter diesem Buch steckt, sei empfohlen, diese Seiten zu überblättern. Oh – das ist nett von Ihnen, daß Sie weiterlesen.

Zwei Mann sind für dieses Buch verantwortlich.

Der eine ist der Commodore-Spezialist: Hannes Rügheimer – im folgenden einfach Hannes genannt – wurde 1965 in Oberndorf geboren. Der kleine Hannes entwickelte schon früh reges Interesse an der Elektronik. Mit acht Jahren jagte er deshalb den Experimentierkasten eines Freundes in die Luft. Die Freundschaft war nur von kurzer Dauer. Als die ersten Home-Computer auf den Markt kamen, fand Hannes damit auch ein neues Betätigungsfeld. Mittlerweile älter und reifer geworden, stellte er hocherfreut fest, daß diese schwarzen Kästen nicht sofort durchschmorten, wenn etwas nicht stimmte, sich dafür aber beharrlich mit SYNTAX ERROR meldeten.

Die allerersten mit Preisen um die 2000 DM waren meist unerschwinglich. Aber freundlicherweise lassen ja Kaufhäuser jeden an diesen Geräten herumprobieren. Stück für Stück arbeitete sich Hannes in die Geheimnisse von BASIC ein. Irgendwann war er es leid, ständig zu warten, bis die anderen fertiggespielt hatten.

Glücklicherweise kam zu diesem Zeitpunkt der VC 20 auf den Markt. Mit vereinten finanziellen Kräften wurde von der Familie dieser «Volkscomputer» gekauft. Der Rest ist schnell erzählt. Zuerst wurde der VC 20 aufgerüstet, ein Jahr später ein Commodore 64 gekauft; dazu ein Diskettenlaufwerk, ein Printer-Plotter usw. Abgesehen von der finanziellen Hilfe durch die Eltern konnte Hannes dank seines Wissens auch bald mit seinem Hobby etwas Geld

verdienen. Er erstellt Computergrafiken und hält Kurse über den Commodore 64. Außerdem berät er Kunden beim Computerkauf.

Damit sind wir beim Mittäter. Name: Christian Spanik, geboren 1963 in Salzburg. Beruf: Juniortexter in einer Frankfurter Werbeagentur. Eigentlich ist seine Entwicklung der von Hannes ziemlich ähnlich, außer daß es bei ihm das Schreiben war, das ihn von jeher interessierte. Um Sie aber nicht mit noch einer Kindheitsgeschichte zu langweilen (auch elektrische Schreibmaschinen gehen sehr leicht kaputt ...), kommen wir am besten gleich zum Kernpunkt.

1979 trafen wir beide zusammen. Wir waren in der gleichen Schule und in der gleichen Klasse. Schließlich fanden wir uns plötzlich auch in der gleichen Bank wieder. Da saßen wir dann auch die restlichen drei Jahre gemeinsam ab. Alle Unterbrechungen des Zusammensitzens sind auf höhere Gewalt zurückzuführen. Unsere Englischlehrerin hatte beispielsweise äußerst eigene Ansichten über die Sitzordnung. Schließlich ging Christian ins Berufsleben. Hannes blieb an der Schule, an der er zur Zeit noch ist.

Kaum im Beruf, hatte auch Christian, ehe er sich's versah, mit Computern zu tun. Obwohl er auf einem anderen System arbeitete, lernte er durch Hannes viel über Commodore. Aufgrund der Commodore-Kurse, die wir zusammen hielten, wurde das Wissen um diesen Computer logischerweise immer stärker. Seitdem arbeiten wir als Team am Commodore 64.

Schließlich kam, was kommen mußte. Durch einen Bekannten kamen wir mit dem Vogel-Buchverlag Würzburg in Verbindung und beschlossen, auch hier unsere Fähigkeiten gemeinsam einzusetzen. Das Ergebnis liegt vor Ihnen. Am besten, Sie urteilen selbst. Sollte es Ihnen gefallen, wäre es schön, wenn Sie uns das wissen ließen. Schreiben Sie uns aber auch, was Ihrer Meinung nach fehlt oder besser sein könnte. Für Anregungen und Tips danken wir schon jetzt.

Weil wir gerade dabei sind, möchten wir an dieser Stelle auch all denen danken, die bei der Entstehung dieses Buches mitgeholfen haben – nämlich den Herren Hugo E. und Christoph Martin, Würzburg, für das Überlassen des Arbeitsmaterials und die intensive Unterstützung, Brigitte für ihr Interesse an unserer Arbeit und ihre Ausdauer im Warten auf Christian, Hannes' Eltern für ihre Geduld und der Firma Lürzer, Conrad & Leo Burnett für das Verständnis, das sie insbesondere der Arbeit von Christian entgegenbrachte.

Zwischenspiel 1

Das Zwischenspiel werden Sie in diesem Buch insgesamt viermal finden. Es hat die Aufgabe, die Dinge aufzunehmen, die nicht direkt in den Ablauf der einzelnen Kapitel passen. Das erste Zwischenspiel enthält ein paar grundsätzliche Anmerkungen.

Alle im Text vorkommenden Wörter, die kursiv gedruckt sind, werden im Stichwortverzeichnis noch einmal erklärt. Die PEEKs und POKEs, die im Buch vorkommen, sind in einem weiteren Anhang zusammengefaßt. Außerdem finden Sie noch Literaturhinweise, Software-Empfehlungen und natürlich die Programmlistings. Die Listings haben wir deshalb in den hinteren Teil des Buches gepackt, weil Sie so die Möglichkeit haben, zuerst eine eigene Lösung zu versuchen, bevor Sie das Gedruckte abtippen. Grundsätzlich halten wir es für den besten Weg, das Buch von vorn nach hinten durchzuarbeiten. Einfach, weil wir sozusagen «aufsteigend» vorgehen. Wer schon mehr Erfahrungen hat, kann natürlich gleich die späteren Kapitel aufschlagen. Die anderen gelegentlich durchzulesen schadet sicherlich auch nicht.

Und jetzt noch ein paar ernste Worte zum Commodore 64: «Wir möchten Sie hiermit beglückwünschen ...» Das ist der erste Satz der Einleitung in der erstaunlich schmächtigen Anleitung, die mit dem 64er ins Haus kommt. Nun, in gewisser Hinsicht ist dieser Satz berechtigt. Es gibt wohl kaum einen Computer, der ein ähnlich gutes Preis-Leistungs-Verhältnis bietet. Da sind die Töne, die tolle Grafik usw. Aber wie programmiert man das eigentlich? Genau da wird es nämlich schwierig. Um dem Computer nur einen einzigen Ton zu entlocken, sind mindestens (!) fünf Speicherstellen anzusprechen. Es gibt keinen einzigen eigenen BASIC-Befehl, der Töne generiert. Genausowenig gibt es dergleichen, wenn es um Grafik geht. Wenn Sie Ihren Commodore schon länger besitzen, werden Sie diese Nachteile bereits kennengelernt

haben. Warum das so ist, war und ist uns völlig unbegreiflich. Denn seit dem ersten Commodore (PET) hat sich dieses BASIC praktisch nicht verändert. Der VC 20 und der Commodore 64 benutzen also immer noch das gleiche BASIC. Ein vergleichbares Beispiel wäre es, einen Käfer-Motor in einen Rolls-Royce einzubauen. Das war sicher eine herbe Enttäuschung für alle, die auf den neuen Commodore 64 umstiegen. Um eine hochauflösende Grafik zu erstellen, müssen Sie beim Commodore 64 jeden einzelnen Punkt mit einem entsprechenden POKE anschalten. Das ist sicherlich nicht der bequemste Weg, Linien zu ziehen.

Weil die Dinge nun mal so sind, wie sie sind, empfehlen wir Ihnen, möglichst bald eine BASIC-Erweiterung zu kaufen (siehe Software-Anhang).

Unser Buch geht dennoch den harten Weg. Denn viele haben eben kein solches Programm. Wir können nur hoffen und versuchen, mit unserer Kritik dazu beizutragen, daß Commodore diesen Zustand alsbald abstellt. Da es keinen anderen Computer gibt, der aufgrund seiner Möglichkeiten und Voraussetzungen besser für Einsteiger geeignet wäre, fänden wir es nur konsequent, das BASIC komfortabler und einstiegsfreundlicher zu machen.

1
Eine kleine Einführung ins Commodore-BASIC

IF no Ahnung THEN dies hier lesen

Wenn Sie die Überschrift hierher verleitet hat, dann haben Sie zwei Dinge bewiesen: Zum einen, daß Sie sich selbst gegenüber sehr ehrlich sind (und das ist eine gute Voraussetzung für das ganze Buch). Zum zweiten haben Sie damit gezeigt, daß Sie durchaus schon ein bißchen BASIC verstehen. Die Kapitelüberschrift ist nämlich ein BASIC-Befehl. Obwohl Ihr Commodore wenig erfreut sein dürfte, wenn Sie die Zeile oben eingeben und versuchen, ihm damit irgendeine Reaktion zu entlocken.

Warum das so ist, möchten wir kurz erklären. Diese Erklärung wird Ihnen vielleicht helfen, das restliche Kapitel und die Idee, die hinter BASIC steht, etwas leichter zu durchschauen.

Als Sie die Überschrift des Kapitels gelesen haben, ging Ihnen wahrscheinlich folgendes durch den Kopf: IF no Ahnung THEN dies hier lesen. Ohne großartig zu übersetzen, kam für Sie der Sinn heraus: Wenn ich noch gar nichts über meinen Commodore weiß, dann sollte ich diese Seiten lesen. Wenn wir Ihre Überlegung in lauter kleine Stückchen zerlegen, ergibt sich ungefähr folgender Monolog: IF ist englisch. Das heißt «wenn» – glaub' ich. No Ahnung: Logisch, das heißt «keine Ahnung». THEN ist ja auch bekannt: «dann» – ebenfalls aus dem Englischen. Dies hier lesen – aha. Gemeint ist: Wenn ich noch keine Ahnung habe, dann soll ich das hier lesen.

Habe ich eigentlich schon Ahnung? Na ja, so ein bißchen schon. Aber ein bißchen Auffrischung kann nicht schaden ... Eine Entscheidung ist gefallen. Sie haben sich entschlossen, dieses Kapitel zu lesen, weil die Bedingungen, das zu tun, für Sie gegeben sind. Aber es gibt Leute, die ebenfalls dieses Buch gekauft haben und genau die andere Entscheidung getroffen haben.

Wir wollen deren Weg nicht noch einmal so auseinanderklamüsern, wichtig war nur, es mal an einem Beispiel zu tun. Natürlich haben Sie Ihre Entscheidung in ein paar Sekunden gefällt, und deshalb werden auch diese ganzen Vorgänge gar nicht so bewußt. Aber sie finden statt. Es ging nur darum, daß Sie verstehen, was die Worte meinen. Denn auch wenn wir IF...THEN in unserer Überschrift verwendet haben – wäre der Rest beispielsweise in Kisuaheli geschrieben gewesen, hätten Sie den Text gar nicht oder höchstens nur teilweise verstehen können. (Selbstverständlich nehmen wir von dieser letzten Bemerkung all die Herrschaften der geschätzten Leserschaft aus, die Kisuaheli gelernt haben.)

Wenn unsere und damit auch Ihre Überlegungen erst einmal so weit gediehen sind, dann wird Ihnen auch schnell klar werden, warum Ihr Commodore nichts mit unserer Überschrift anfangen kann. Sie besteht zwar teilweise aus einem BASIC-Befehl, aber man kann mit an Sicherheit grenzender Wahrscheinlichkeit davon ausgehen, daß Ihr Computer mit Ausdrücken wie «no Ahnung» oder «dies hier lesen» nichts anfangen kann. (Womit übrigens auch schon der eigentliche BASIC-Befehl aus der Überschrift verraten wäre: Er heißt IF...THEN. Aber den schauen wir uns erst etwas später an.)

Langer Rede kurzer Sinn: Wir versuchten Ihnen nahezubringen, daß der Computer seine eigene Sprache hat. Natürlich gibt es immer ein paar Leute, die einwenden werden: «Ja, wenn der Kerl so dumm ist, daß er mich nicht versteht, soll er das doch erst mal lernen, bevor man ihn auf die Menschheit losläßt». Nun, diese Meinung ist sicherlich nicht ganz unberechtigt. Aber nehmen wir das Beispiel BASIC: Schon diese Sprache ist ein Zugeständnis an die Menschen, denn ein Computer wird nicht etwa mit BASIC als Muttersprache geboren. Was er kann, ist «Strom» oder «kein Strom» voneinander zu unterscheiden. Alles andere haben ihm Männer und Frauen in den Entwicklungsabteilungen der großen Computerfirmen und an Universitäten in mühevoller Arbeit beigebracht. Sie haben den Computer gebaut und ihm in einer besonderen Sprache namens Assembler oder Maschinensprache all die Dinge beigebracht, die es heute ermöglichen, daß wir Computer einfach programmieren können, daß sie uns bei unserer Arbeit helfen und uns mit Spielen unterhalten. Aber was auch immer die Experten getan haben, die BASIC entwickelten, das Lernen und Verstehen können sie uns nicht abnehmen. Erfreulicherweise.

Wir wollen Sie deshalb in diesem Kapitel einladen, mit uns einen Flug zu machen. Dieses Bild schien uns am besten zu passen. Wir wollen nämlich zu einer Art Insel, die im Meer der vielen Computersprachen und BASIC-Dialekte

liegt. Lassen Sie uns diese Insel «Commodore-BASIC» nennen. Unseren Flug dorthin und auch den Aufenthalt wollen Sie sich bitte als eine Art Studienreise vorstellen. Sie sollen zwar etwas dabei lernen, aber es soll vor allem Spaß machen.

Nachdem wir uns einig sind, dürfen wir unsere Passagiere an Bord unserer Maschine bitten. Stellen Sie das Rauchen ein, schnallen Sie sich an und – das wichtigste von allem – schalten Sie Ihren Commodore ein. Sie sollten alles, was wir jetzt besprechen, mitmachen. Wenn Sie bereit sind, kann unser Flug beginnen. Weil die BA (BASIC AIRWAYS) aus Kostengründen auf Stewardessen verzichtet, sollten Sie sich vielleicht schon vorher etwas zu trinken und zu knabbern holen.

Unterwegs zur Runway

Noch ein paar Anweisungen, während wir in Startposition rollen. Unsere Flugzeit hängt ganz von Ihnen ab. Aber es sollte nie allzulange dauern, damit Sie nicht zuviel vom Anfang vergessen. Notausgänge befinden sich überall in der Maschine und können jederzeit gern als Ort der Ruhe und des Nachdenkens benutzt werden. Scheuen Sie sich nicht, immer und jederzeit auszusteigen, um selbst auszuprobieren und zu üben. Glauben Sie uns, diesen Service bietet Ihnen keine andere Fluggesellschaft, denn bei uns können Sie auch jederzeit wieder zurückkommen. Die Flughöhe wird sich im Verlauf der Reise steigern. Sollten Sie noch nie mit uns geflogen sein (und wer ist das bisher schon ...?), nehmen Sie sich bitte noch die Sicherheitsinstruktionen zur Hand:

Darin finden Sie ein paar Dinge, die Sie wissen sollten. Die klingen zumeist sehr simpel (sind es auch), aber die eine oder andere Kleinigkeit, die man nicht beachtet hat, brachte schon allerhand Programmieranfänger zur Verzweiflung.

Zuerst einmal müssen Sie zwischen zwei Zuständen, in denen sich der Computer befinden kann, unterscheiden: Direktmodus oder Programmodus. Wenn Sie eingeben

```
PRINT "HALLO"
```

und dann ⟨RETURN⟩ drücken, wird Ihr Computer diesen Befehl sofort ausführen. Wenn Sie aber statt dessen eingeben:

```
10 PRINT "HALLO"
```

und danach ⟨RETURN⟩, wird erst einmal gar nichts passieren, zumindest nichts Sichtbares. Statt diese Zeile sofort auszuführen, speichert sie der Computer ab. In einem speziell dafür reservierten Bereich werden alle Befehle abgelegt, bis sie mit einem anderen Befehl aufgerufen werden. Damit wird auch der Sinn der Zeilennummer klar. Irgendwie muß ja geklärt sein, welcher der abgespeicherten Befehle zuerst ausgeführt werden soll.

Ein Beispiel aus dem berühmten «richtigen Leben» wäre ein Tagesablauf. Wenn Sie sich zum Beispiel folgendes kleine Programm für den 10. 3. zurechtgelegt hatten: Waschen, Anziehen, zur Arbeit gehen, nach Hause kommen, mit meiner Frau spazierengehen und sie dann zum Essen einladen, dann dürfte es ein ziemliches Durcheinander geben, wenn Sie den Programmteil «zur Arbeit gehen» vor allen anderen ausführen. Selbst wenn Sie einen sehr liberalen Arbeitgeber haben – spätestens Ihre Frau wird sich weigern, mit Ihnen im Pyjama spazieren oder gar essen zu gehen.

Das zum Thema Programmablauf. Eines noch: Sie werden sicherlich schon gemerkt haben, daß bei Programmausdrucken in Computerheften meist Zeilennummern in Zehnerschritten verwendet werden. Das ist natürlich nicht etwa eine Vorschrift. Sie können Zeilennummern frei zuweisen, solange Sie dabei die Reihenfolge einhalten, die das betreffende Programm vorschreibt.

Allerdings sind die Zehnerschritte schon sehr zu empfehlen. Denn wenn man mal etwas vergessen hat oder nachträglich einfügen möchte, ist das meist schwer, wenn nicht genug Platz zwischen den Zeilen ist. Viele Programme sind tatsächlich erst zwischen den Zeilen interessant geworden, weil einem halt mitten beim Programmieren immer mehr Ideen kommen als bei irgendwelchen Trockenübungen.

Jetzt noch ein paar Sätze zum Thema «Wie versteht mich mein Computer?». Es gibt drei «Dienstprogramme», die den Computer verständnisvoll machen. Zusammengefaßt nennt man diese drei Programme das *Betriebssystem* (Operating System). Da wäre zunächst einmal der *BASIC-Interpreter*. Er ist dafür verantwortlich, daß Ihre BASIC-Befehle in für den Computer verständliche Maschinensignale umgewandelt werden. Vielleicht haben Sie schon hier und da gehört, daß BASIC eine der langsamsten Computersprachen sein soll. Das stimmt. Denn wenn Sie direkt in der sogenannten Maschinensprache programmieren, sparen Sie Ihrem Computer den Weg über das Übersetzungsbüro – das heißt, der Dolmetscher (BASIC-Interpreter) wird nicht mehr gebraucht. Deshalb geht in Maschinensprache auch alles

schneller als in BASIC. Dafür ist sie schwerer zu erlernen, und selbst bei Profis dauert es lange, bis ein Maschinenprogramm richtig läuft, weil jede Anweisung in kleinste Schritte aufgeteilt werden muß und sich Fehler nicht nur schneller einschleichen, sondern auch schwieriger gefunden werden. BASIC wird deshalb auch als Interpretersprache bezeichnet.

Vielleicht haben Sie auch schon mal etwas von *Compilersprachen* gehört. PASCAL ist zum Beispiel eine solche. Compiler soll zum Ausdruck bringen, daß aus einem Programm in einer relativ einfach zu erlernenden Sprache ein der Maschinensprache sehr ähnlicher *Code* gemacht wird. Der ist dann zwar noch nicht ganz so schnell, aber doch schon ziemlich ebenbürtig.

Das nächste unserer Hauptprogramme ist das sogenannte Kernal. Es steuert alle Prozesse im Computer, die ständig stattfinden müssen – unabhängig von Ihrem laufenden Programm. Es führt die nötigen *Interrupts* aus. Das heißt zum Beispiel, daß es in kaum meßbaren Zeitabständen immer ein Programm losschickt, das nachschaut, ob Sie irgend etwas über die Tastatur eingegeben haben und ähnliches. Dann gibt es noch ein Programm, das für die Bildschirmausgabe sorgt. Es weist bestimmte Bausteine des Computers an, Ihre Eingaben am Bildschirm darzustellen oder eine Meldung des BASIC-Interpreters (etwa den unvermeidlichen SYNTAX ERROR) auszugeben, damit Sie wieder einmal erfahren, daß irgendwo in diesem M...-Programm ein Fehler steckt. Diese Programme sind – in einem speziellen Speicher gesichert – ständig vorhanden und überleben dort auch das Ausschalten des Computers. Mehr über die einzelnen Programme erfahren Sie noch – und auch einiges darüber, wie Sie sich deren Fähigkeiten zunutze machen können.

Zur Demonstration der Arbeit im Computer dient folgender Dialogauszug, frei übersetzt aus dem Stromischen ...

«Draußen vor dem Schirm, da steht ein User fein.»
«Nun, das kenn' ich schon – am besten laßt ihn ein.»
(Zitat von K. Ernal, am 25. 2. 1982 anläßlich einer Tastaturabfrage)

Der Dialog wird zwischen mehreren Programmen stattfinden. Zeit des Gesprächs ist etwa 22.30 Uhr. Anlaß ist der Versuch von Hannes Rügheimer, den Fehler in einem Programm zu beheben. Teilnehmer sind: der BASIC-Interpreter (B.I.), das Kernal (K.) und das Bildschirmprogramm (B.P.)

B.I.: «O nein!»
K.: «Doch!»

B.I.: «Er probiert es doch nicht etwa schon wieder?»
K.: «Weiß ich nicht. Er hat mir auf jeden Fall einen Befehl aus drei Buchstaben über die Tastatur mitgegeben. Was für einen, ist ja wohl dein Bier.»
B.I. (sich den mitgebrachten Befehl ansehend): «Tatsächlich. Schon wieder ein RUN. Ich soll sein ganzes Programm noch mal laufen lassen. Langsam müßte er doch kapiert haben, daß da ein Fehler drinsteckt. Das ist jetzt schon das fünftemal.»
K.: «Hartnäckig ist er ja. Das muß man ihm lassen.»
B.I. (ruft in Richtung des Mikroprozessors): «He, mach dich mal fertig. Der Typ da draußen will schon wieder einen Lauf.»
(Schweigen)

B.I. (dreht sich wieder um): «Was heißt hier hartnäckig! Als wir noch in diesem Laden rumstanden, da war erst was los. Da kommen dann so Knaben an mit Eingaben wie ‹Ich bin do...!›»
K.: «Entschuldige, wenn ich dich unterbrechen muß, aber ich höre gerade, daß ein Interrupt kommen soll. Also wartet mal 'n Moment.»
B.I.: «Es ist immer dasselbe mit euch Kernals, nie könnt ihr euch ruhig hinsetzen. Keine fünf Minuten quatschen kann man hier.»
K.: «Fünf Minuten! Bist du verrückt?»
B.P. (sich plötzlich einmischend): «Entschuldigung die Herrschaften, wenn ich mich einmische. Aber wenn ihr weniger schwätzen würdet, wäre ich euch sehr verbunden. So ein Interrupt ist nämlich gar nicht so schlecht. Dann kann ich vielleicht mal wieder meinen Bildschirm auf Vordermann bringen.»
B.I.: «Weißt du, was mich dein Bildschirm interessiert?»
B.P.: «Will ich gar nicht wissen. Aber unser User hätte schon ganz gern mal gewußt, ob der Herr BASIC-Interpreter in der Lage ist, dem Prozessor so eine einfache Nachricht wie RUN zu überbringen. Und mich würde das ebenfalls brennend interessieren, damit ich meinen blinkenden Cursor mal wegnehmen kann und der User merkt, daß hier auch was passiert.»
K.: «Ihr entschuldigt mich bitte. Ich müßte nämlich jetzt wirklich ...»
B.I.: «Typisch, kaum wird es brenzlig, haut er ab, diese Flasche.»
B.P.: «Pflichtbewußtsein nennt man das. Schon mal gehört?»
B.I. (resignierend): «Okay, okay. Nach dem Interrupt fangen wir an ...»

Wie gesagt, dies war eine freie Übersetzung. Übrigens: Die Dauer eines solchen Gesprächs liegt bei einer sechzigstel Sekunde. Dann kommt immer ein Interrupt.

Nachdem wir zugehört haben, können wir uns ins Geschehen begeben. Das heißt: Lassen wir die Jungs auf der anderen Seite vom Schirm doch mal ein bißchen für uns arbeiten. Jetzt widmen wir uns den Basics von BASIC.

Startposition erreicht. Ready for TAKE OFF

Noch eine Kleinigkeit, bevor wir uns in die BASIC-Höhen aufschwingen: Dieser Abschnitt ist kein BASIC-Lehrgang. Das will und kann er auch gar nicht sein. Wir wollen nur einen kleinen Überblick über die wichtigsten Befehle und ihre Anwendung geben. Damit sind Sie dann in der Lage, einfache Programme selbst zu schreiben. Später auftauchende Befehle, die nicht erklärt wurden, schlagen Sie bitte in Ihrem Commodore-Handbuch nach. Feinheiten des Programmierens lernen Sie noch in diesem Buch. Die beste Art zu lernen ist, selbst große oder kleine Programme zu schreiben.

Jetzt geht es los. Wir starten. Während die Maschine dabei ist, in unsere Reisehöhe zu klettern, können wir schon die ersten Überlegungen anstellen. Die erste Frage ist: Wie spricht man mit einem Computer während eines Programms? Unsere Frage gilt also dem Stichwort: Der Dialog.

Wenn Sie ein Programm schreiben, kann Ihr Computer zum Beispiel immer eine Zahl durch eine andere dividieren. Die Frage ist bloß, wie macht man das in BASIC-Programmen. Im Direktmodus würde es so aussehen:

```
PRINT 12/2
```

PRINT

Womit auch schon ein grundlegender Befehl genannt wäre. PRINT sagt dem Computer, daß er irgend etwas zeigen oder ausgeben soll. Mit Ausgeben kann hier natürlich nicht eine Runde in Ihrem Stammlokal gemeint sein, sondern nur ein Wert, der irgendwo errechnet oder eine Zahl, die abgespeichert wurde. PRINT wird immer dann verwendet, wenn dem Computer mitgeteilt werden soll, daß er irgend etwas anzeigen soll. Wenn Sie gerade die Zeile von oben ausprobiert haben, werden Sie sehen, daß nach dem ⟨RETURN⟩ das Ergebnis (hoffentlich die Zahl 6) ausgegeben wird. Wenn Sie aber statt dessen eingeben:

```
PRINT "12/2"
```

dann wird Ihnen der Computer als Ergebnis nicht sechs, sondern genau das liefern, was zwischen den Anführungszeichen steht, und zwar ohne das Wort PRINT und ohne die Anführungszeichen. Statt Zahlen können Sie auch Texte verwenden, der Computer gibt alles klaglos aus. Klaglos vor allem deshalb, weil er nicht versucht, das, was zwischen den Anführungszeichen steht, als Befehl auszulegen. Der Interpreter merkt durch diese Zeichen, daß er nur den Inhalt wiedergeben soll. Sie können auch versuchen,

```
PRINT HALLO
```

einzugeben. Nach diesem Versuch werden Sie als Antwort die berühmte Null-Lösung (bekannt aus Fernsehen und Politik) erhalten. Warum das so ist, erklären wir noch. Wenn Sie also einen Text ausgeben wollen, müssen Sie ihn zwischen Anführungszeichen setzen. Wenn Sie jetzt also innerhalb eines Programms Zahlen dividieren wollen, würde das Programm etwa heißen:

```
100 PRINT 12/2
```

Zugegebenermaßen ist dieses Programm nicht gerade anspruchsvoll. Um es etwas aufzupäppeln, könnte man ja nun dazufügen:

```
5 PRINT "DIVISIONSPROGRAMM VON OSKAR MUELLER"
100 PRINT 12/2
```

Sollten Sie zufällig nicht Oskar Müller heißen, setzen Sie bitte Ihren Namen ein.

Wenn Sie jetzt vorhaben, dieses Programm auf dem freien Markt anzubieten (etwa Freunden, Eltern, Kindern und Bekannten), werden Sie schnell merken, daß Ihr Programm bei diesen Anwendern nur begrenzten Erfolg haben wird. Erstens werden sie, selbst wenn es sich um miserable Kopfrechner handelt, nach zwei bis drei Programmläufen das Ergebnis im vorhinein (vor dem Computer!) wissen. Das einzige, was man daraus noch machen könnte, wäre ein Reaktionsspiel. Nach dem Motto: Wer sagt schneller sechs. Doch das wird irgendwann langweilig. Vor allem wird es dann peinlich, wenn einer der jüngsten Teilnehmer sagt: «Das kann mein Taschenrechner auch.»

Da hilft dann auch der Hinweis auf die oberste Zeile mit Ihrem Namen nichts mehr.

Außerdem: Wenn Sie eine andere Zahlenkombination ausprobieren wollen, müßten Sie immer erst das Programm listen und neue Zahlen eingeben. Überhaupt: Unser Stichwort lautete ja: Der Dialog. Aber wenn der Computer in einem Programm nur immer ausgibt und Sie nie eingeben, kann man

wirklich nicht von einem Dialog sprechen. Damit kommen wir an einen Punkt, an dem der Computer im Lauf eines Programms Eingaben annehmen können muß.

Wir möchten zum Beispiel gern die Zahlen, die dividiert werden, am Anfang des Programms eingeben. Dazu brauchen wir einen neuen Befehl:

INPUT

Der Befehl INPUT weist den Computer an, auf eine Eingabe zu warten. Und das tut er dann auch brav und geduldig. Aber INPUT allein reicht nicht aus, denn wenn Sie zwei Zahlen eingeben wollen, muß der Computer beim Dividieren zum Beispiel wissen, welche Zahl durch welche geteilt werden soll. Damit er jede Zahl von der anderen unterscheiden kann, müssen wir unserem Computer eine Hilfestellung geben. Weil sich die Zahl von einem Programmlauf zum nächsten verändern soll, also für den Computer immer eine Unbekannte ist, verwenden wir Variable. Vielleicht denken Sie jetzt gerade an all die schönen Gleichungen mit X und Y. So ähnlich ist es auch hier. Und weil es so schön an die Schule erinnert und wir auch zufällig zwei Platzhalter brauchen, nehmen wir doch gleich X und Y. Die beiden sind sowieso so etwas ähnliches wie Dick und Doof der Mathematik. Unser Programm würde jetzt also lauten:

```
5 PRINT "DIVISIONSPROGRAMM VON OSKAR MUELLER"
20 INPUT X
30 INPUT Y
100 PRINT X/Y
```

Daß sich die Zeile 100 verändern muß, ist Ihnen sicherlich klar. Sonst würde der Computer trotzdem – wie beim vorherigen Beispiel – 12 durch 2 dividieren. Wenn Sie dieses Programm mit RUN starten, erscheint zuerst ein Fragezeichen. Damit zeigt der Computer, daß er auf eine Eingabe wartet. Wenn Sie jetzt eine Zahl – na, nehmen wir zur Abwechslung mal 12 – eingeben, erscheint ein zweites Fragezeichen. Da geben Sie zum Beispiel 2 ein. Als Ergebnis erscheint sechs. (Irgendwoher kommt uns das so bekannt vor . . .)

Sie sehen: Ihr Computer nimmt jetzt Zahlen an und teilt sie. Damit Sie sehen, wie schlau er ist, geben Sie bitte als erste Zahl 120 und als zweite Zahl 0 ein. Bei dem einen oder anderen geht jetzt vielleicht eine in jahrelangen Mathematikstunden eingebaute Warnlampe an. Durch Null dividieren?

Wenn Sie es ausprobieren, merken Sie sehr schnell, daß das auch der Computer gelernt hat. Er sagt Ihnen nämlich, daß man nicht durch Null dividieren darf.

Bevor Sie nun voller Stolz Ihr neues Divisionsprogramm herzeigen, wollen wir Ihnen noch einen Tip wegen der Platzhalter geben. Da gibt es einige Unterschiede zwischen:

A, A$, AA, AA$

Es wäre doch auch ganz schön, wenn der Benutzer seinen Namen eingeben könnte. Wenn Sie das aber mit dem Platzhalter A versuchen, wird das nicht funktionieren. Wie in der Mathematik können diese Platzhalter nur für Zahlen stehen. Wenn Sie Zeichenketten zuweisen wollen, müssen Sie die Platzhalter für Zahlen und die für Buchstaben voneinander trennen. Aber das ist nicht weiter schwer. Damit der Computer weiß, daß der Platzhalter für eine Zeichenkette stehen soll, hängen Sie einfach ein $-Zeichen dran. Dieses Zeichen erinnert zwar extrem an die US-amerikanische Währung, wird aber üblicherweise nicht A-Dollar, sondern A-String ausgesprochen. Solche Strings können dann sowohl Buchstaben als auch Zahlen als Inhalt haben. Einfache Buchstaben können nur als Platzhalter für Zahlenwerte stehen. Ergebnis unserer Recherchen ist also: A nimmt nur Zahlen an und A$ Buchstaben und Zahlen.

Damit Ihnen auch ganz sicher nicht die Variablen ausgehen, obwohl man selten so viele braucht, kann der Computer sogar noch den Platzhalter A vom Platzhalter AA unterscheiden. Sie können also A einen anderen Wert zuweisen als AA, und Ihr Commodore weiß, daß A nicht gleich AA ist. Genauso ist es bei den Platzhaltern für Buchstaben. Sie haben so fast unendlich viele Platzhalter zur Verfügung. Wenn wir jetzt den Namen des augenblicklichen Benutzers abfragen und wieder ausgeben wollen, machen wir am besten folgendes:

```
5 PRINT "DIVISIONSPROGRAMM VON OSKAR MUELLER"
10 INPUT N$
20 INPUT X
30 INPUT Y
100 PRINT X/Y
```

Die Variable N$ haben wir übrigens nur deshalb benutzt, weil man sich dann leichter daran erinnern kann, daß N$ für NAME steht. Jetzt könnten Sie am Schluß des Programms den Namen mit einem kleinen Dankeschön fürs

Benutzen von Ihrem Commodore ausdrucken (gemeint ist auf den Bildschirm bringen) lassen. Zum Beispiel:

```
5 PRINT "DIVISIONSPROGRAMM VON OSKAR MUELLER"
10 INPUT N$
20 INPUT X
30 INPUT Y
100 PRINT X/Y
110 PRINT "VIELEN DANK FUERS RECHNEN MIT MIR, LIEBER "N$
```

Wenn Sie das Programm jetzt laufen lassen, werden Sie feststellen, daß sich der Computer höflich von seinem Benutzer verabschiedet. Das Leerzeichen in der PRINT-Zeile hat den Sinn, daß zwischen dem Namen und dem "... LIEBER" noch etwas Platz ist.

Um Text zu formatieren und dergleichen, gibt es zusätzlich einige besondere Zeichen, zum Beispiel ',' oder ';'. Wenn Sie diese Zeichen vor das N$ in Zeile 110 setzen, hat das interessante Effekte. Übrigens: Woher weiß eigentlich ein fremder Benutzer, daß er da seinen Namen und dort Zahlen eingeben soll? Nun, entweder findet er es heraus, weil der Computer nicht richtig reagiert, oder Sie geben ihm (dem Benutzer) Anweisungen. Nachdem Sie jetzt wissen, wie man mit PRINT umgehen muß, ist das kein Problem mehr.

Um das Stichwort Dialog zu rechtfertigen, fehlt aber noch ein Befehl.

GET

Mit GET werden immer nur einzelne Zeichen eingelesen. Sie müssen danach kein RETURN eingeben. Allerdings muß eins beachtet werden, wenn Sie mit GET einen Buchstaben oder eine Zahl abfragen wollen. Der Commodore hat einen Zeichenpuffer, der bis zu 10 Zeichen behalten kann. Genaueres dazu später. Weil aber dieser Zeichenpuffer immer aktiv ist und der GET-Befehl, sobald er ein Zeichen bekommen hat, weitermacht, muß man sicherstellen, daß der Computer nicht einfach weiterläuft, weil er vom Puffer einen Leerstring erhält. Deshalb muß eine GET-Abfrage folgendermaßen aussehen:

```
10 GET A$:IF A$="" THEN 10
```

Ab Zeile 20 kann dann in diesem Fall das kommen, was passieren soll, wenn die Eingabe stattgefunden hat.

Übrigens: Nur noch ein paar Minuten, dann haben wir unsere Reiseflug-

höhe erreicht. Die Zeit bis dahin benutzen wir am besten, noch den Befehl zu erklären, den Sie jetzt schon zweimal gehört oder gelesen haben.

IF...THEN

Übersetzt bedeutet er etwa WENN... DANN. Um ein Beispiel aus dem Leben zu nehmen, das zeigt, wann wir solche Entscheidungen gebrauchen, möchten wir gern auf Christians Schwester zurückgreifen. Wenn sie keine Lust zum Abwaschen hatte, sollte er das tun. In jedem Fall war ihr Standardsatz, wenn Christian und Geschirr zusammenkamen, immer: «WENN du irgend etwas runterschmeißt, DANN kannst du was erleben.»

Christian war sich immer sicher, daß sie das ernst meinte – das heißt, wenn diese Bemerkung erst einmal gefallen war, wußte er, daß sich Konsequenzen ergeben würden. Voraussetzung war, daß ein Teller oder irgendwelches Geschirr die Prozedur nicht überlebte.

Und genauso ist es mit dem Computer. Wenn Sie eine IF... THEN-Klausel einbauen, wird er sie ungerührt abfragen. Ist die Bedingung erfüllt, dann reagiert er so, wie es abgemacht war.

Wenn wir zum Beispiel unser kleines Rechenprogramm von vorhin nicht immer neu starten wollen, aber andererseits die Möglichkeit zum Abbruch nicht ganz ausschließen wollen, könnte man das so lösen:

```
5 PRINT "DIVISIONSPROGRAMM VON OSKAR MUELLER"
10 INPUT N$
20 INPUT X
30 INPUT Y
100 PRINT X/Y
102 PRINT "WOLLEN SIE JETZT WEITERMACHEN? (J/N)"
104 GET A$:IF A$="" THEN GOTO 104
106 IF A$="J" THEN RUN
108 IF A$="N" THEN GOTO 110
110 PRINT "VIELEN DANK FUERS RECHNEN MIT MIR, LIEBER "N$
```

Was passiert, ist wohl ziemlich klar. Zeile 102 schreibt dem Benutzer eine Anweisung auf den Schirm, die ihm sagt, wie er antworten soll, nämlich mit J oder N.

Zeile 104 wartet, bis eine Taste gedrückt wird. Solange das nicht geschieht bleibt das Programm bei dieser Abfrageroutine. Wenn dann die Taste gedrückt wurde, überprüft Zeile 106, ob es die Taste J war. War es J, dann

gibt sich das Programm selbst den Befehl RUN, fängt also wieder von vorn an. Zur Zeile 108 oder 110 kommt es dann gar nicht erst, weil die erste Bedingung erfüllt war. Ist die gedrückte Taste aber N, geht das Programm zu Zeile 110 und verabschiedet sich. Wenn Sie genau hinschauen, merken Sie zwei Dinge. Zum einen, warum es immer gut ist, in Zehnerabständen zu numerieren, und zum anderen, daß die Zeile 108 auch weggelassen werden könnte. Denn wenn die Bedingung in Zeile 106 nicht erfüllt wird, arbeitet der Computer sowieso die nächste Zeile ab. Und die ist ja die letzte Programmzeile. Sie sehen, Ihr Commodore ist in solchen Dingen genauso konsequent wie Christians Schwester.

10 000 Meter

Wir haben inzwischen die vorgesehene Reisehöhe erreicht und schon die BASIC-Befehle besprochen, die bereits ein paarmal im Verlauf des Textes aufgetaucht sind. Weil damit die drängendsten Fragen Ihrerseits wohl erledigt wären, gehen wir jetzt dazu über, die Befehle in alphabetischer Reihenfolge zu erklären, denn man kann nicht sagen, daß einer wichtiger sei als der andere. Man braucht sie alle. Wo es nötig ist, finden Sie im Text auch noch Hinweise auf die Aussprache – damit Sie das nächstemal im Computerladen auch mitreden können oder zumindest verstehen, was die da überhaupt reden.

ASC

Oft wüßte man gern, was für einen ASCII-Wert eine bestimmte Taste hat. Man kann natürlich das Programmierhandbuch aufschlagen und diese Werte suchen, aber weil das Buch nicht selten ganz zuunterst in einem Riesenhaufen von Papieren liegt, erweist sich dieses Vorgehen als unpraktikabel. Zumal man dieses Buch, sobald man danach sucht, sowieso nicht findet, selbst wenn man schwören könnte, daß es irgendwo in dem Stapel rechts von der Tür sein muß. Deshalb hat Commodore einen Befehl eingebaut, der es möglich macht, diesen Wert abzufragen. Der Befehl lautet ASC ($) (sprich Äski von einem String . . .). Wollen Sie den ASCII-Wert von X erfahren, geben Sie ein:

```
PRINT ASC ("X")
```

Als Antwort erhalten Sie genau die Zahl, die Sie gesucht haben.

CHR$

Wenn Sie erst einmal den ASCII-Code eines Zeichens haben, können Sie den sehr einfach wieder in ein Zeichen zurückverwandeln. Der Befehl CHR$ (N) (sprich Tscharstring von N) liefert Ihnen das Zeichen, das der Zahl N, die in Klammern steht, entspricht. Wenn Sie also gerade den ASCII-Wert von X herausgefunden haben, können Sie das gleich mal ausprobieren. Der Wert, den Sie bei X erhalten haben sollten, ist 88. Geben Sie jetzt ein:

```
PRINT CHR$ (88)
```

Der Lohn für Ihre Mühe wird ein X sein, das plötzlich auf dem Bildschirm auftaucht. Was Sie davon haben, daß Sie diese CHR$ kennen und benutzen können, werden wir im Kapitel Input/Output noch erklären.

CLOSE

Sie werden bald merken (spätestens im Kapitel über die Peripheriegeräte), daß der Commodore, wenn er mit angeschlossenen Geräten ein kleines Schwätzchen halten will, einen Kommandokanal eröffnen muß. Weil alle Geräte hintereinander an ihm hängen, hat jedes Gerät seine Nummer. Wenn der Commodore ein Gerät mit einer bestimmten Nummer sprechen will, ruft er laut und deutlich die Gerätenummer. Dann wissen alle anderen Geräte, daß sie jetzt weghören sollen. (Ob sie es tun, ist die andere Frage.) Weil im Alphabet der Buchstabe O (wie OPEN) nach dem Buchstaben C (wie CLOSE) kommt, ist CLOSE zwar von der Logik her der Befehl, den man erst später braucht, aber wir haben schließlich gesagt, wir machen es in alphabetischer Reihenfolge. Ist also irgendwo ein Kommandokanal mit OPEN eröffnet worden, wird er mit

```
CLOSE N
```

wieder geschlossen. Ist das passiert – sind also alle Kommandokanäle geschlossen –, dann brütet unser Commodore wieder in dumpfer Einsamkeit über seinen Rechenaufgaben. N steht hier wieder für eine Zahl.

CMD

CMD (sprich Command, wofür es auch die Abkürzung ist) ist ähnlich der CLOSE ein Peripheriebefehl, das heißt, dieser Befehl hat Auswirkungen auf die umliegenden (angeschlossenen!) Geräte. Der Fachmann umschreibt so

che Befehle auch gern mit dem Ausdruck «I/O-Befehle». Immer wenn Sie einen Fachmann auf freier Flur treffen – zum Beispiel in Computerläden oder Fachabteilungen der Kaufhäuser –, werden Sie bemerken, wie ein stolzes Lächeln um seine Lippen zuckt, wenn er sagt: «Nun, wahrscheinlich ist es ein I/O-Problem.» Nach diesem Satz fliegen dann oft noch Ausdrücke wie «Device» (sprich Diweis) oder «Floppy» durch die Gegend. Und wenn alle diese Wörter auch nach einer Stunde noch nicht geholfen haben, ein bestimmtes Problem zu lösen, fliegen meist nur noch relativ wenig fachliche Kraftausdrücke herum. Sie sehen, daß an dem Ausdruck «I/O» nichts Magisches hängt. Er ist nur eine Abkürzung für Input/Output, und das ist seinerseits wieder nur die englische Version der Wörter Eingabe/Ausgabe. Gemeint ist damit alles, was beim Computer mit der Ein- oder Ausgabe von Daten zu tun hat. So schnell verliert der Computer und die Sprache um ihn herum an Schrecken.

Der Vorteil von Abkürzungen ist aber unbestritten. Deshalb wollen wir ab jetzt auch I/O-Befehl sagen, wenn wir solche Dinge meinen. Wenn Sie also einen Kommandokanal zum Drucker aufgemacht haben, können Sie auch Kommandos zu ihm schicken, die er ausführen soll. Der Befehl müßte dann lauten:

CMD N: Befehl

N steht für das entsprechende Gerät, und statt Befehl müßten Sie eben einen BASIC-Befehl einsetzen – zum Beispiel LIST, um ein Programm vom Drucker listen zu lassen. Das geht jedoch erst, nachdem Sie einen Kommandokanal eröffnet haben.

CONT

CONT gehört ins normale BASIC-Repertoire. Natürlich ist es eine Abkürzung, die für Continue steht, was soviel heißt wie Weitermachen. Der Befehl wird immer dann verwendet, wenn ein Programm zum Beispiel mit ⟨RUN/STOP⟩ abgebrochen wurde, oder auch, wenn im Programm der Befehl STOP oder END angetroffen wurde. Natürlich ergibt CONT nur dann einen Sinn, wenn das Programm noch weitergeht. Wenn Sie aber während eines so erzwungenen Programmstopps eine Zeile listen und verändern oder ein SYNTAX ERROR auftritt, dann gibt Ihr Commodore ganz bescheiden zu, daß er leider gar nicht weitermachen kann. Der Befehl lautet einfach nur

CONT

Die Antwort kann entweder sein, daß er das Programm weiter ausführt oder aber auch ein

CAN'T CONTINUE ERROR

Mit CONT können Sie schon allerhand anfangen, zum Beispiel abfragen, ob das Ergebnis einer bestimmten Rechnung auch das richtige ist, und nur dann weiterrechnen lassen. Ihnen fällt bestimmt noch mehr ein.

DATA

DATA erinnert schon so deutlich an den Ausdruck «Daten». Das soll es auch. Die eigentliche Aufgabe für Computer ist ja sowieso, bestimmte Daten zu suchen, auszuwerten oder was immer damit zu machen. (Die Zeitungsschlagzeilen des letzten Jahres beweisen, daß den Leuten die unmöglichsten Dinge einfallen, die man mit Daten machen kann oder wofür man sie brauchen könnte ...) Wofür also ein eigener Befehl für Daten? Wir müssen beim Programmieren verschiedene Datenarten unterscheiden: die Programmdaten (das eigentliche Programm) oder die Daten der Variablen oder die Daten, die in DATA-Zeilen abgelegt werden.

DATA-Zeilen können überall im Programm stehen. Egal wo, der Computer wird sie immer der Reihe nach lesen, wenn man ihm sagt, daß er das tun soll. Eine DATA-Zeile in einem Programm kann ungefähr so aussehen:

DATA TARZAN,JANE,CHEETAH,3

Wie man so eine Datenzeile im Programm verwendet, folgt erst beim Befehl READ. Gedulden Sie sich noch ein wenig. Dann werden Sie auch verstehen, was diese komischen Ausdrücke sollen.

DIM

Das ist ein Befehl, der es möglich macht, sogenannte Felder zu DIMensionieren. Daher auch der Ausdruck. Sie können zum Beispiel sagen:

DIM A (100)

Dann können Sie der Variablen A bis zu hundert verschiedene Zahlen zuweisen. Jede Zahl bekommt einen sogenannten Index. Um dann die einzelnen Zahlen wieder hervorzuholen, geben Sie

PRINT A(0)

ein und erhalten dann den ersten Wert, den Sie A zugewiesen haben. 0 ist deshalb der erste Wert, weil der Computer fast immer bei Null zu zählen beginnt. Das ist eine seiner spleenigen Eigenarten. Im Grunde sorgt die DIM-Anweisung nur dafür, daß genügend Speicherplatz für Daten reserviert wird. Vergleichen könnte man das mit einem Restaurant. DIM A(5) wäre, wenn Sie dort anrufen und sagen: «Ich möchte bitte einen Tisch für fünf Personen.» Der Kellner reserviert für Sie den Tisch A mit fünf Plätzen. Es könnten sich aber Probleme ergeben. Wenn Sie zum Beispiel nochmals anrufen, um zu sagen, daß jetzt doch nur drei Leute kommen würden, wird der Kellner etwas unwirsch reagieren. Unser Commodore auch. Er wird Ihnen sagen:

REDIM' D ARRAY

was heißen soll, daß Sie einen bereits dimensionierten Raum noch einmal dimensionieren. (Das sollen Sie aber nicht.)

Zum Restaurant zurück. Wenn Sie dem Telefongespräch aus dem Weg gehen wollen und mit nur drei Leuten auftauchen, wird das dem Kellner zwar auch nicht recht sein, aber er kommt wenigstens nicht auf die Idee, Sie an einen kleineren Tisch zu setzen. Damit ist in manchen Gaststätten erst gewährleistet, daß jeder ausreichend Platz zum Essen hat. Wenn Sie dasselbe beim Commodore tun, ist ihm das ziemlich egal. Der Platz bleibt reserviert, und wenn Sie ihn nicht ausnutzen, verschenken Sie eben Speicher. So einfach ist das.

Wenn Sie – wieder im Restaurant – dem Ganzen aus dem Weg gehen wollen und erst gar nicht hingehen, obwohl Sie reserviert haben, wird sich der Kellner zwar ärgern, tun kann er aber gar nichts. Die Reaktion des Computers? Siehe oben.

Das zum Thema Reser ... – pardon, wir meinen natürlich Dimensionierungen.

END

Diesen Befehl hatten wir schon einmal kurz angesprochen. Damit kann man ein Programm (zum Beispiel bei einem bestimmten Ergebnis) abbrechen lassen. Um ehrlich zu sein, zu diesem Befehl gibt es nichts weiter zu sagen, so sehr wir uns auch anstrengen, es fällt uns nichts Lustiges dazu ein. Der Befehl ist eben wirklich nur

END

Es geht natürlich weiter.

FOR...NEXT

FOR...NEXT ist ein Befehl, der oft für einen anderen eingesetzt wird: X=X+1. Wenn Sie diesen Befehl in ein Programm bringen, weil aus irgendwelchen Gründen gezählt werden soll, würde das Programm so aussehen:

```
10 X=X+1
20 PRINT X
30 IF X=10 THEN END
40 GOTO 10
```

Der neue Befehl GOTO in diesem Programm wird Sie wohl nicht verwirren. Er heißt einfach nur GEHE ZU. (Erklärung folgt noch.)

Das Programm zählt von 1 bis 10. X ist, solange ihm kein Wert zugewiesen wird – genauso wie alle Platzhalter im Computer – eine Null. Dann wird dem Computer in Zeile 10 gesagt, daß er zu X eine 1 addieren soll. Somit ist X beim ersten Lauf eine 1. In Zeile 20 wird der augenblickliche X-Wert auf den Bildschirm gebracht. Dann wird festgestellt, ob X mittlerweile den Wert 10 hat. Der schlaue Commodore merkt, daß dem nicht so ist, und geht zur Zeile 40. Die macht ihm klar, daß er wieder zur Zeile 10 soll. Dort addiert er dann zum momentanen X-Wert eine 1 und ...

Den Rest können wir uns wohl schenken. Nun ist es etwas umständlich, vier Zeilen Programm zu brauchen, um die Leistung eines Erstkläßlers zu erreichen.

Mit FOR...NEXT geht es einfacher. Dasselbe Programm würde hier lauten:

```
10 FOR X=1TO10
20 PRINT X
30 NEXT X
```

Wenn X den Wert 10 erreicht hat, hört der Computer ganz von selbst auf zu zählen. Das Ganze kann man übrigens auch in einer Zeile zusammenfassen. Das würde dann folgendermaßen aussehen:

```
10 FOR X=1TO10:PRINT X:NEXT X
```

Ein Doppelpunkt trennt immer mehrere Anweisungen voneinander. Das hat allerdings den Nachteil, daß alles etwas unübersichtlicher wird und daß man natürlich das Programm immer nur zu einer bestimmten Zeilennummer und

nie nur zu einem Teil dieser Zeile schicken kann. Zurück zu unserer FOR-...NEXT-Anweisung. Die meisten Leute verwenden für diese Anweisung auch den Ausdruck «FOR...NEXT-Schleife», was aber nichts mit hübsch verpackten Geschenken zu tun hat. Nun hätte – denkt vielleicht der scharfe Beobachter – die andere Version gegenüber der FOR...NEXT-Schleife einen Vorteil, wenn man zum Beispiel nicht in Einer-, sondern in Zweierschritten zählen lassen will. Mit dem erwähnten Programm würde das ja kein Problem sein:

```
10 X=X+2
20 PRINT X
30 IF X=10 THEN END
40 GOTO 10
```

Man muß nur die erste Zeile verändern. Aber diesen Luxus kann man mit der FOR...NEXT-Schleife genauso einfach haben. Man muß nur die Schrittweite des Zählens angeben. Dazu dient der Befehl STEP (Schritt):

```
10 FOR X=1TO10 STEP 2
20 PRINT X
30 NEXT X
```

So einfach ist das!

FRE

Eine nicht ganz uninteressante Frage, die beim Programmieren auftaucht, ist die nach dem freien Speicherplatz. Auch hier kann der Computer mit Rat und Tat zur Seite stehen. Weil er den Speicher sowieso ständig im Überblick haben muß, war das Einbauen einer solchen Funktion nicht schwierig. Der Befehl FRE(0) (sprich Frie von Null) hilft weiter. Wenn Sie einem frisch eingeschalteten Commodore

```
PRINT FRE(0)
```

eingeben, gibt er Ihnen gern Auskunft. Sollte die Zahl, die er ausgibt, negativ sein, dann liegt das nicht etwa daran, daß er Sie auf den Arm nehmen will, sondern an internen Abläufen. Um die tatsächliche Zahl herauszufinden, müssen Sie noch 65536 dazuzählen. Also schreiben Sie in diesem Fall am besten:

```
PRINT FRE(0)+65536
```

Dann erhalten Sie den freien Speicherplatz. Zwei Dinge noch dazu: Wenn Sie glauben, daß Sie sich diese Zahl nicht merken können, warten Sie ab. Spätestens wenn Sie das Kapitel «Speicheraufteilung» gelesen haben, werden Sie von solchen Zahlen träumen.

Und: FRE(0) ist eine Funktion, die an praktisch jedem Computer funktioniert. Wenn Sie also Bekannte haben, die wenig über Computer wissen, kann man damit Eindruck machen. Eine kleine Anleitung dazu:

1. Man nehme einen oder mehrere Bekannte und führe sie wie zufällig in die Computerabteilung eines Kaufhauses.
2. Wenn man dann – genauso zufällig – an den kleinen geheimnisvollen Kästen entlangstreicht, über sie plaudert, bleibe man plötzlich an einem stehen.
3. Während sich alle – durch das plötzliche Stehenbleiben noch etwas verwirrt – erst einmal wieder fangen müssen, lesen Sie schnell den Namen des betreffenden Geräts.
4. Wenn Ihnen die ungeteilte Aufmerksamkeit wieder sicher ist, murmeln Sie halblaut: «Ach schau an, der neue XYZ.» (Für XYZ setzen Sie bitte den abgelesenen Namen ein.)
5. Dann gehen Sie zur Tastatur und tippen unser bekanntes PRINT FRE(0) ein. Dabei ist es egal, ob mit oder ohne der Addition von 65536.
6. Während alle vor Staunen erstarren, erhalten Sie als Ergebnis irgendeine Zahl.
7. Ab jetzt reichen halblaut hingeworfene Bemerkungen wie «Soso» oder «Dachte ich es mir doch» und ähnliches aus.
8. Sie gehen kommentarlos weiter.
9. Irgendwann wird einer Ihrer Begleiter (oft erst nach Minuten) fragen: «Was hast Du denn da gemacht?»
10. Und jetzt kommt das Beste. Als Antwort reicht hier völlig aus: «Oh, ich wollte nur mal feststellen, wie es so im System ausschaut.»
11. Sollten wider Erwarten doch noch Fragen kommen, sagen Sie einfach: «Das ist jetzt zu kompliziert zu erklären. Und ich will Euch damit nicht langweilen. Ach, dabei fällt mir ein ...»
12. Achtung! Probieren Sie diesen Trick nie, wenn Kinder in der Nähe sind. Das könnte schreckliche Folgen haben (von verhaltenem Grinsen bis zu lautem Lachen). Und achten Sie darauf, daß keiner Ihrer Bekannten diese Geschichte je erfährt ...

GOSUB

Dieser Befehl ist eine besondere Form des GOTO, das wir schon kennengelernt haben, aber es funktioniert etwas anders. Zuerst zur Namensklärung. GOSUB steht für GO SUBroutine, was (frei) übersetzt heißt: «Gehe zu einer Unterroutine.»

Unterroutine klingt zwar großartig, aber als solche bezeichnen Programmierer fast alle Programmteile, die mehr als einmal verwendet werden und bestimmte Dinge ausführen. Wenn Sie zum Beispiel oft in einem Programm eine Entscheidung des Programmbenutzers mit J oder N (für Ja oder Nein) anfordern, können Sie natürlich jedesmal diesen Programmteil schreiben, in dem diese Abfrage gemacht wird. Es reicht aber auch, wenn Sie das nur einmal machen. Das Programm würde dann zum Beispiel so aussehen:

```
10 PRINT "WOLLEN SIE ANFANGEN? (J/N)"
20 GOSUB 300
30 (Ab hier beginnt Ihr Programm)
100 PRINT "WOLLEN SIE WEITERMACHEN? (J/N)"
110 GOSUB 300
120 (Hier geht Ihr Programm weiter)
290 END
300 (Ab hier muß Ihre Abfrageroutine stehen)
350 (Wenn Ihre Unterroutine fertig ist, heißt der letzte Befehl:)
360 RETURN
```

Was passiert hier? Zuerst wird Ihre Anwenderinformation auf den Bildschirm gebracht, denn irgendwoher muß der ja wissen, was er tun soll. Dann geht das Programm ganz normal weiter zur nächsten Zeile. Hier trifft es dann die Anweisung GOSUB 300. Aha, denkt das Programm, ich soll zur Zeile 300. Aber weil da nicht GOTO, sondern GOSUB steht, merkt es sich im Vorbeihuschen noch schnell die Zeile, in der dieser Befehl steht. Bei 300 angelangt, führt es die Anweisungen dort so lange aus, bis es zum Rücksprungbefehl RETURN kommt. Der heißt zwar genauso wie die Taste auf dem Keyboard, hat aber damit nichts zu tun. Jetzt wird auch klar, warum die Zeilennummer mit der Sprungadresse so wichtig war. Ganz von selbst kehrt nämlich Ihr Programm jetzt wieder zu dem Befehl bzw. der Zeile zurück, die als nächste hätte ausgeführt werden sollen, wenn nicht das GOSUB dagewesen wäre. Dieses Spielchen läßt sich beliebig oft durchführen und so manche Tipperei einsparen. Noch ein Tip dazu: Vielleicht ist Ihnen aufgefallen, daß wir genau

eine Zeile vor der Unterroutine den Befehl END gesetzt haben. Der Grund dafür ist klar. Unterroutinen stehen immer ziemlich am Ende eines Programms. Wenn dann das Programm zu Ende ist, steht ja die Unterroutine immer noch da. Würden wir das Programm nicht aufhalten, würde es unsinnigerweise unsere Unterroutine ausführen.

GOTO

Zum Thema GOTO brauchen wir wohl nach obigen Erklärungen nicht mehr viel zu sagen. Damit wird das Programm verzweigt, wenn Sie beispielsweise ein sogenanntes Menü in Ihr Programm einbauen. Das hört sich zwar appetitlich an, ist es aber nicht unbedingt. Menü hat mit der Speisekarte eines Lokals nur eins gemeinsam: die Möglichkeit der Auswahl. Ein Computermenü könnte zum Beispiel so aussehen:

```
AUSWAHL:

SPIELEN...1
LERNEN...2
ENDE.....3

BITTE DRUECKEN SIE DIE ENTSPRECHENDE ZAHLENTASTE:?
```

Je nachdem, ob 1, 2 oder 3 gewählt wird, kann das Programm an eine bestimmte Stelle verzweigen, die das Gewünschte bereitstellt. Weil Sie aber in dieses Menü selten zurückmüssen, wird hier meist ein GOTO verwendet. Ein GOTO kann Ihnen aber auch aus der Patsche helfen, wenn zum Beispiel eine bis dahin benötigte Unterroutine plötzlich beim Programmieren im Weg ist weil diese Routine bei 300 anfängt, aber Ihre Programmzeilen bereits bis 29 fortgeschritten sind. Am elegantesten ist es in dem Fall, das Programm mi einem

```
GOTO 400
```

darüber hinwegzuheben, wenn die Unterroutine beispielsweise bis 390 ode so geht.

IF...GOTO

Wir hatten schon über IF...THEN gesprochen. Eine Abart davon ist da IF...GOTO. Was es tut, können Sie sich sicherlich erschließen. Man kann au

das Tippen von THEN verzichten und gleich eine Programmverzweigung veranlassen.

LET

LET ist natürlich auch englisch. Es kommt von to let, was auf deutsch einfach «lassen» heißt. LET allein ist der Imperativ, also die Befehlsform. Wenn man es übersetzt, heißt es schlicht «laß». Im Computer wird LET für Wertzuweisungen benutzt. Das sieht im Programm so aus:

```
LET D=5
```

Damit wird der Variablen D der Wert 5 zugewiesen. Ganz unter uns: Das LET kann man ruhig weglassen. Ein einfaches

```
D=5
```

reicht vollauf.

LIST

LIST ist ein Kommando, das Sie sehr oft brauchen werden – zwar selten in Programmen, aber um so öfter beim Programmieren.

Wenn Sie nämlich programmieren, wird Ihnen schnell auffallen, daß die Programmzeilen, die Sie eingeben, langsam, aber sicher dem oberen Bildschirmrand zustreben. Man nennt das auch *Scrolling*.

Wahrscheinlich haben Sie Vertrauen zur Firma Commodore und denken sich, daß man all das, was da oben verschwindet, sicherlich irgendwie wieder herkriegt. Das stimmt auch, und zwar eben mit dem Befehl LIST. Wenn Sie zum Beispiel Ihr Programm wiedersehen wollen, tippen Sie einfach

```
LIST
```

ein. Sogleich fängt der Commodore an, wie wild Ihre Programmzeilen aufzulisten. Damit Sie nicht denken, er hätte was verloren, tut er das sogar ganz hübsch schnell – meist etwas zu schnell zum Mitlesen. Deshalb drücken Sie, während er listet, einfach die ⟨CTRL⟩-Taste. Dann weiß er, daß Sie ihm vertrauen, und geht die Sache etwas beschaulicher an. Wenn Sie mal wollen, daß er ganz zu listen aufhört, weil Sie irgend etwas verbessern wollen, drücken Sie einfach ⟨RUN/STOP⟩. Weil aber niemand von Ihnen verlangen kann, daß Sie sich ein 2000 Zeilen langes Programm vom Anfang bis zum

augenblicklichen Ende anschauen, nur um an die 1979. Zeile heranzukommen, können Sie auch nur bestimmte Teile listen lassen:

LIST 1979

zeigt Ihnen zum Beispiel nur diese eine Zeile.

LIST 1979–

zeigt Ihnen alles ab 1979.

LIST –1979

zeigt Ihnen alles bis zur Zeile 1979.

LIST 1900–1980

zeigt Ihnen alles zwischen den Zeilen 1900 und 1980.

LIST kann natürlich auch innerhalb eines Programms verwendet werden Allerdings befindet sich der Computer danach automatisch im Direktmodus Das Programm muß dann neu gestartet werden.

LOAD

Dieser Befehl ist nur für Peripheriebesitzer interessant. Mit

LOAD

lassen sich Programme vom Kassettenrecorder laden. Mit

LOAD "NAME",8

kann man ein bestimmtes (NAME) Programm von der Diskettenstation lade lassen. Näheres zu diesem Thema im Kapitel über Peripheriegeräte.

NEW

Mit diesem Befehl sollten Sie sehr vorsichtig umgehen. Er löscht nämlich da momentan im Speicher befindliche Programm. Was weg ist, ist weg und fü den unbedarften Programmierer nicht mehr wiederzukriegen.

Viele Leute verwenden das NEW auch im Programm – zum Beispiel, wen es darum geht, ein laufendes Programm zu beenden. Dann hat nämlich d Programmierer, der damit gearbeitet hat, sofort einen freien Speicher. Dies Absicht ist löblich und sicherlich eine saubere Lösung. Wenn Sie es auch s

machen wollen, dann sollten Sie allerdings folgenden kleinen Ratschlag beherzigen:

Nehmen wir an, in Zeile 2350 Ihres Programms wurde der Anwender gefragt, ob er jetzt aufhören möchte. Nehmen wir weiter an, er hat das auch getan, und vermuten wir nun, Sie haben für diesen Fall ein NEW in Zeile 2360 vorgesehen. Bisher war alles prima. Doch gehen wir ein paar Tage in die Vergangenheit zurück. Es ist wieder 22.30 Uhr, und Sie sind gerade beim letzten Programmtest. Soeben haben Sie nach langen Mühen Ihr Programm um einen tollen Toneffekt und eine schöne Grafik bereichert. Jetzt wollen Sie nur noch schnell ausprobieren, ob sich das alles mit dem Programm verträgt. Sie geben Ihr RUN ein, alles läuft wunderbar. In ihrer grenzenlosen Freude über das gelungene Programm geben Sie auf die freundliche Frage: «Wollen Sie, lieber Hans-Peter, jetzt wirklich schon aufhören?» tatsächlich das Ja-Wort. Es folgt der tolle Ton, auf dem Bildschirm blinkt noch einmal ein kurzes «Tschüs, Hans-Peter», der Bildschirm wird plötzlich leer, und es ist vorbei.

«Ja», denken Sie noch, «so sieht das aus, so muß das sein.» Zufrieden legen Sie noch einmal Ihre Hände auf die Tastatur und tippen ein letztes Mal LIST ein.

Jedoch alles, was erscheint, ist die READY-Meldung. Sonst nichts. Aus. Keine Zeile des Zinseszinsprogramms ist mehr zu sehen. Sollte der Apparat, aber nein – natürlich –, das NEW. Was jetzt?

Meist setzt in solchen Fällen noch bis Mitternacht hektische Betriebsamkeit ein. Noch zwei, drei LIST-Versuche. Dann das Durchkramen von den abgespeicherten alten Versionen. Aber nein. Es ist vorbei. Nach einer halben Stunde wird dann der Entschluß immer stärker, jetzt den Computer auszuschalten. Man schaut noch einmal 15 Sekunden auf den Bildschirm, wo es gerade noch war, aber es ist zu spät. Kein Wunder geschieht. Das Programm ist weg.

Programmierers Freud und Programmierers Leid – wie nah liegen sie beisammen. Das Bild wird dunkel. Die Arbeit ist weg. Nur noch die alte Version – ohne Ton, ohne Grafik – ist da. Und die ist natürlich viel, viel langweiliger, als es die andere war.

Deshalb unser Tip: So rabiate, schicksalsschwangere Befehle wie NEW sollten wirklich erst ganz zum Schluß, wenn das Programm abgespeichert ist, eingefügt und ausprobiert werden. Wenn Sie es ausprobieren wollen, schreiben Sie statt dessen lieber eine Zeile wie:

```
2360 PRINT "HIER WUERDE EIGENTLICH EIN NEW STEHEN.
GOTT SEI DANK IST ES NOCH NICHT DA!"
```

Diese kleine Geschichte und das Beherzigen unseres einfachen Tips können Sie vor Verdruß bewahren. Selbst erfahrenen Programmierern ist so etwas schon passiert, auch wir beide können Lieder von dieser Problematik singen.

OPEN

Den Befehl OPEN (das ist ebenfalls einer der I/O-Befehle) haben wir ja schon kurz erwähnt. Er dient zum Öffnen eines Kommandokanals. Der Befehl muß heißen:

```
OPEN N,N,N
```

Die drei N stehen für Zahlen. Wann alle drei gebraucht werden und wann nicht, entnehmen Sie bitte dem späteren Kapitel «Peripheriegeräte».

PEEK

PEEK ist ein äußerst nützlicher Befehl. Er liest den Inhalt einer Speicherzelle aus. Man braucht ihn zum Beispiel bei Sprites oder zum Definieren eigener Grafikzeichen. Wie das genau geht, erfahren Sie in den folgenden Kapiteln. PEEKen können Sie in jede Speicherzelle zwischen 0 und 65 535. Sie bekommen dabei immer einen Wert zwischen 0 und 255 geliefert. Für beide Zahlen, die wir Ihnen hier genannt haben, kommen ebenfalls in den folgenden Kapiteln die Erklärungen. Ein Beispiel für das HerausPEEKen eines Wertes ist:

```
?PEEK(53280)
```

Damit erhalten Sie den Wert, der in der Speicherzelle 53280 steht. Bei einem frisch eingeschalteten Commodore müßte das Ergebnis 254 sein. Warum wir diese Speicherzelle genommen haben, wird Ihnen der nächste Befehl zeigen.

POKE

POKE ist das genaue Gegenteil von PEEK. Wenn Sie etwas POKEn, dann heißt das, daß Sie einen Wert in eine Speicherzelle schreiben. Das kann manchmal sehr erstaunliche Effekte haben. Probieren Sie folgendes:

```
10 POKE 53280,0: POKE 53281,0:POKE 646,0
20 FOR X=1TO 1000
30 PRINT "X";
40 NEXT
50 POKE 646,1
```

Zuerst sehen Sie, daß Ihr Bildschirm plötzlich schwarz wird, und zwar der ganze Bildschirm. Gleichzeitig verschwindet der eingegebene Text wie von Geisterhand nach oben. Obwohl Sie eine Zeile 30 haben, in der ausdrücklich steht, daß der Buchstabe X auf den Bildschirm gebracht werden soll, passiert nichts – zumindest scheinbar. Doch nach einiger Zeit steht ganz unten auf dem Bildschirm ein helles READY, obwohl der Bildschirm (wenn Sie sich an das erinnern, was wir bei der FOR... NEXT-Schleife gesagt haben) eigentlich voll mit lauter X sein müßte. An diesem kleinen Beispiel können Sie erkennen, was für seltsame Dinge POKEs auslösen können. Zum besseren Verständnis ändern Sie erst einmal die Zeile 10 ab, am besten so, daß sie dann lautet:

```
10 POKE 53280,0: POKE 53281,0:POKE 646,1
```

Den Rest des Programms lassen Sie so, wie er ist. Das Abändern ist ziemlich einfach. Sie müssen nur LIST 10 eingeben. Dann haben Sie die Zeile 10 vor sich. Mit den *Cursor*-Tasten – ganz rechts unten auf dem Keyboard – gehen Sie so lange nach oben, bis Sie auf der Höhe der gelisteten Zeile sind. Dann fahren Sie mit der anderen (äußersten) Cursortaste einfach über die Zeile hinweg. Erst bei der letzten 0 des POKE 646 halten Sie an. Wenn Ihr blinkender Cursor genau darüber ist, tippen Sie eine 1. Die 0 darunter verschwindet von selbst. Wenn Sie das gemacht haben, drücken Sie auf ⟨RETURN⟩. Dann können Sie wieder RUN eingeben. Jetzt können Sie sehen, wie lauter weiße X den Bildschirm auffüllen.

Zur Erklärung: Die Speicherzelle 646 ist für die Farbe der ausgegebenen Zeichen zuständig, Speicherzelle 53280 für die Rahmenfarbe und 53281 für die Hintergrundfarbe. Weil wir in die beiden 0 gePOKEd haben, wurde scheinbar der ganze Bildschirm schwarz. Beim ersten Programmlauf haben wir auch die Zeichenfarbe (Speicherzelle 646) auf Schwarz geschaltet. Deshalb haben wir nicht gesehen, daß unser Text von einem anderen (schwarzen) Text nach oben geschoben wurde. Erst die letzte Zeile hat wieder die Zeichenfarbe Weiß ausgelöst. Deshalb konnten wir das READY wieder sehen. Ganz einfach, oder? Wenn Sie jetzt Ihr Wissen von vorhin nutzen, können Sie sogar die Rahmenfarbe wieder auf Normal zurücksetzen. Wir haben ja vorhin den Wert herausgePEEKed, der nach dem Einschalten in dieser Speicherzelle steht.

Experimentieren Sie mit all diesen Zahlen ruhig ein bißchen herum. Es kann Ihrem Commodore 64 keinen dauerhaften Schaden zufügen. Das Schlimmste, was passieren kann, ist, daß der Rechner abstürzt. Und selbst, wenn das passiert, müssen Sie sich nicht bücken. Abstürzen heißt nur, daß er plötzlich

nicht mehr reagiert und nur noch durch Aus- und Wiedereinschalten zu erlösen ist. Deshalb sollten Sie, wenn Sie das POKEn ausprobieren, auch möglichst kein wichtiges, noch nicht auf Diskette oder Kassette abgespeichertes Programm im Speicher haben. Denn das hätte sich dann natürlich in Wohlgefallen aufgelöst.

READ

Jetzt sind wir endlich bei dem Befehl, von dem wir schon ziemlich am Anfang unseres Fluges gesprochen haben. READ ist der Befehl, der die DATA-Zeilen lesen kann. Wir zeigen Ihnen jetzt, was READ tut und wie dieser Befehl mit den DATAs in Programmen umgeht. Dazu geben Sie zunächst bitte folgendes Programm ein:

```
10 READ A$,B$,C$,D$
15 READ A
20 PRINT A$" UND "B$" LEBEN ZUSAMMEN IM WALD UND HABEN
   EINEN RIESEN"C$", DER "A;
30 PRINT " JAHRE ALT IST UND "D$" HEISST."
40 DATA TARZAN,JANE,AFFEN, CHEETAH,3
```

An diese DATA-Zeile können Sie sich vielleicht noch erinnern. Wenn Sie das Programm jetzt laufen lassen, bekommt auch der merkwürdige Satz, den Sie abgeschrieben haben, einen Sinn. Die Reihenfolge, in der der Computer die DATAs liest, hängt von der Reihenfolge ab, in der sie in der DATA-Zeile stehen. Also müssen DATAs immer in der Reihenfolge ihres Abrufs stehen. Egal ist allerdings, wo die DATA-Zeile steht. Wenn Sie ihr zum Beispiel die Zeilennummer 5 geben (sie also vor die READ-Anweisung legen), ändert das auch nichts. Die DATA-Zeile sucht sich der Computer von selbst, genauso wie er von selbst weiß, an welchem DATA er zu lesen aufgehört hat. Denn wenn er das nächste READ trifft, liest er genau da weiter, wo er zuletzt aufgehört hat – also auch mitten in einer DATA-Zeile.

REM

REM kommt natürlich auch aus dem Englischen. Es ist eine Abkürzung für REMark, was dem deutschen Wort Anmerkung sehr nahekommt. Ein REM in einem BASIC-Programm hat meistens den Sinn, ein Programm besser durch

schaubar zu machen. Aber bevor Sie sich jetzt falsche Hoffnungen machen, daß damit plötzlich alles viel leichter wird, wollen wir Sie besser gleich aufklären. Am besten zeigt Ihnen ein Beispiel, was wir meinen:

```
10 POKE 53280,1:REM RAHMENFARBE WEISS
20 POKE 53281,0:REM BILDSCHIRMFARBE SCHWARZ
```

Wie Sie sehen, ist bei diesem Programm auch noch nach drei Monaten gewährleistet, daß Sie wissen, was in den ersten beiden Zeilen passiert. Der REMark, die Anmerkung, erinnert Sie daran, daß die erste Zeile die Rahmenfarbe Weiß macht und die zweite Zeile die Bildschirmfarbe Schwarz.

Wenn aber vor diesen beiden Hinweisen nicht REM stünde, würde der Computer – genauer gesagt der BASIC-Interpreter – versuchen, die Ausdrücke Rahmenfarbe Weiß und Bildschirmfarbe Schwarz zu verstehen oder gar aus ihnen Befehle zu machen. Daß das in einem heillosen Durcheinander endet, merken Sie immer daran, daß der Computer ein hilfloses SYNTAX ERROR ausgibt. Sie können das gern mal ausprobieren, indem Sie einfach die beiden REMs in den Zeilen 10 und 20 herausnehmen, aber den Rest lassen, wie er ist.

REM bedeutet also für den Computer, daß er den Rest, der dahinter kommt, nicht mehr zu beachten braucht.

REM-Statements machen es also leichter, sich selbst nach längerer Zeit wieder in einem Programm zurechtzufinden. Allerdings haben sie zwei Nachteile.

Zum einen braucht natürlich jedes REM Speicherplatz. Deshalb sollte man doch etwas sparsam damit umgehen.

Zum anderen kostet es für den Computer jedesmal Zeit, ein REM zu erkennen, es zu übersetzen und zu verarbeiten.

Die Kunst bei REMs ist einfach in einen Satz zu bringen: Genug zum Zurechtfinden müssen es sein, aber wenig genug zum effektiven Programmieren sollen es sein.

Das ist sicher leichter gesagt als getan. Aber wir raten Ihnen, lieber ein paar REMs mehr zu benutzen. Wenn es bei einem Programm auf Schnelligkeit ankommt, machen Sie einfach eine Version, in der Sie alle REMs rausschmeißen. Die ist dann die Laufversion. Die andere benutzen Sie, um Änderungen zu machen. Dann kommen wieder alle REMs raus. Das ist zwar ziemlich zeitaufwendig, aber ein passabler Kompromiß.

RESTORE

Noch einmal kurz zu den DATA-Zeilen. Wie schon gesagt, weiß das laufende Programm immer genau, bei welchem DATA es aufgehört hat. Um nun ein Programm dazu zu bringen, wieder beim allerersten DATA zu lesen anzufangen, gibt es nur zwei Möglichkeiten. Entweder mit RUN das Programm neu zu starten oder eben den Befehl RESTORE. Das neuerliche Starten des Programms bringt ja nicht viel, deshalb ist tatsächlich RESTORE die einzige Möglichkeit, bei den DATAs wieder ganz von vorne anzufangen. Mittlerweile haben Sie sicherlich genug BASIC-Erfahrung, um sich selbst zu denken, wie die Befehlsform sein muß:

RESTORE

Nicht gerade schwierig oder?

RETURN

Das haben wir vorhin schon mal gehabt. RETURN ist eine Anweisung, die sein muß, um aus einer BASIC-Unterroutine zurück zum eigentlichen Programm zu springen. Wenn irgendwo im Programm ein GOSUB vorkommt, dann muß es auch ein dazugehöriges RETURN geben. RETURN steht immer am Ende der Unterroutine. Wenn der Computer auf ein RETURN trifft, ohne daß vorher mit einem GOSUB eine Unterroutine angesprungen wurde (wie der Fachmann sich das bildlich vorstellt, wissen wir zwar nicht, aber auf jeden Fall sagt er «anspringen» dazu), gibt das die Fehlermeldung

RETURN WITHOUT GOSUB ERROR

Nur damit Sie Bescheid wissen, falls das einmal auftritt.

RUN

Dieser Befehl müßte Ihnen mittlerweile hinlänglich bekannt sein. Wenn man ein Programm starten will, muß der Befehl

RUN

eingegeben werden. Sie können aber ein Programm auch erst von einer bestimmten Zeilennummer ab starten lassen. Dazu müßten Sie dann eingeben:

RUN N

wobei N irgendeine Zeile sein kann. Wenn die Zeile nicht im Programm vorkommt, bekommen Sie natürlich als Dankeschön eine kleine Fehlermeldung. Diesmal heißt sie zur Abwechslung

UNDEF' D STATEMENT

Um wieder ein Beispiel aus dem Leben zu bringen: Stellen Sie sich einen riesigen Wohnblock vor. Sie suchen einen alten Bekannten, der wahrscheinlich hier wohnt. Alles, was Sie an Lebendigem hier sehen, ist jemand, der das Treppenhaus bohnert. Es ist ein Mann, wahrscheinlich der Hausmeister. «Gut», denken Sie, «der sollte es ja wissen.» Sie gehen hin und fragen höflich: «Entschuldigung, ich suche die Familie Müller ...»

Der Mann schaut Sie kurz an und meint dann: «Die gibt's hier nicht.» Das Haus ist Ihr Programm, die Rolle des Hausmeisters übernimmt in diesem Fall der BASIC-Interpreter, der zwar den Namen versteht, aber sofort zurückgibt, daß es den hier nicht gibt. So ungefähr ist die Bedeutung dieser Fehlermeldung.

SAVE

Dieses ist eines der großartigen I/O-Kommandos. Mit SAVE können Sie ein Programm abspeichern, zum Beispiel auf Kassette oder Diskette. Das ist übrigens auch die einzige Art, mit der Sie Ihre Programme dauerhaft schützen können, denn ein BASIC-Programm, das Sie geschrieben haben, wird das Abschalten des Computers nicht überleben. Deshalb sollte man auch während des Programmierens immer wieder zwischenspeichern – einfach um für den Fall eines Stromausfalls oder anderer unglücklicher Umstände gerüstet zu sein. Denken Sie nur an kleine Kinder («Schau mal Papa, der Stecker da ...») oder große Eltern («Jetzt reicht's aber mit der Rumspielerei. Das Essen ist fertig.»), die in beiden Fällen zur Unterstützung ihrer Worte den Stecker aus der Dose ziehen.

Der Befehl selbst hängt in erster Linie vom Speichermedium ab:

SAVE "XXX"

speichert das Programm XXX auf Kassette – allerdings nur, wenn die beiden Tasten RECORD und PLAY gedrückt sind und der Recorder deutliche Anzeichen – sowohl akustische wie auch optische – von Bewegung von sich gibt. Übrigens kann Ihr Commodore nicht unterscheiden, ob am Recorder die richtigen Tasten gedrückt sind. Es kann Ihnen also durchaus passieren, daß

Sie, statt zu laden, aus Versehen löschen oder hin- und herspulen. Deshalb sollten Sie nach dem Recordergebrauch immer die ⟨STOP⟩-Taste drücken.

```
SAVE "XXX",8
```

speichert das Programm XXX auf der Diskette. Dazu muß die Diskettenstation allerdings eingeschaltet sein. Sonst gibt es einen

```
DEVICE NOT PRESENT ERROR
```

was soviel heißt wie: «Ist nicht da. Geht nicht.»

Bitte schnallen Sie sich wieder an ...

Merken Sie was? Wir sind jetzt über das Meer der Programmiersprachen hinweg. Vor uns taucht langsam die Insel Commodore-BASIC auf. Das langersehnte Ziel ist zum Greifen nahe. Schnallen Sie sich also wieder an. Während der Pilot langsam zum Landen ansetzt, sollten wir uns noch über ein paar kleine Befehle unterhalten.

STOP

Das hatten wir ja auch schon mal. Mit STOP können Sie ein Programm an praktisch jeder Stelle unterbrechen. Beim Programmieren ist das manchmal ganz nützlich, um kurz vor der Zeile, in der ständig ein Fehler auftritt, das Programm abzubrechen und zum Beispiel die Variablen zu überprüfen. Dann können Sie mit CONT weitermachen. Die Syntax für den Befehl STOP ist denkbar einfach. Irgendwo steht eine Zeile, die einfach den Befehl

```
STOP
```

hat. Der Ausdruck Syntax, den wir gerade verwendet haben, ist sicherlich dem einen oder anderen aus irgendwelchen Grammatikübungen noch bekannt. Die richtige Syntax heißt: der richtige Satzbau.

Beim Computer ist es genauso. Ein SYNTAX ERROR ist eine fehlerhafte, unverständliche Eingabe. Deshalb nennt man auch die richtige Befehlsform für eine bestimmte Aufgabe, die der Computer bearbeiten soll, die Syntax.

Sehen Sie, je weiter wir kommen, um so mehr können wir uns schon wie ausgefuchste Programmierer unterhalten.

SYS

Der Befehl SYS hat eine ganz besondere Aufgabe. Auch er greift (wie PEEK und POKE) direkt auf den interen Speicher des Commodore zurück. Er ruft zum Beispiel bestimmte Maschinenunterroutinen auf. So können Sie den Computer mit

```
SYS 64738
```

neu starten, ohne daß Sie ihn dazu ausschalten müssen. Wie Sie diesen Befehl nutzen können, was Sie davon haben und wo der Unterschied zwischen einer Maschinenunterroutine und einer BASIC-Unterroutine liegt, werden Sie am Ziel unserer Reise erfahren.

TAB

Mit diesem Befehl können Sie – wie bei einer Schreibmaschine – Tabulatoren setzen. Daher auch der Befehlsname. Ein TAB-Befehl wäre etwa:

```
10 PRINT TAB(20)"HALLO"
```

Wenn Sie das eingeben und starten, werden Sie gleich merken, was der Effekt ist. Das Hallo steht nicht mehr in der ersten Spalte, sondern beginnt in der zwanzigsten. Damit können Sie zum Beispiel Textausgaben immer genau untereinander setzen.

Langsam wird es jetzt Zeit, sich auf die Landung vorzubereiten. Apropos Zeit.

TIME

Geben Sie doch mal ein:

```
PRINT TI/60
```

Die Zahl, die Sie dann erhalten, sind die Sekunden, seit denen Ihr Commodore schon läuft. Sobald Sie ihn einschalten, wird auch eine innere Uhr angeschaltet, die alle sechzigstel Sekunde weiterzählt. Und weil wir gerade bei der Uhrzeit sind: Sie haben auch eine richtige Uhr in Ihrem Commodore. Wenn Sie eingeben:

```
PRINT TI$
```

erhalten Sie eine sechsstellige Zahl, die für Stunden, Minuten und Sekunden steht. Mit dem Befehl oben können Sie sehen, seit wie vielen Stunden,

Minuten und Sekunden Ihr Commodore jetzt schon läuft. Sie können die Uhr auch stellen, wenn Sie wollen. Zum Beispiel, wenn Sie eingeben:

```
TI$="224600"
```

und dann eine Minute später wieder fragen:

```
PRINT TI$
```

dann werden Sie als Antwort

```
224700
```

erhalten – zumindest dann, wenn Sie wirklich genau eine Minute später gefragt haben.

Es ist soweit: Bitte stellen Sie jetzt das Rauchen ein. Wir setzen in wenigen Minuten zur Landung an.

Sie müssen nur noch ein paar Minuten warten. Oh – warten. Stimmt ja. Das ist auch ein BASIC-Befehl ...

WAIT

Mit dem Befehl WAIT bringen Sie den Computer dazu, so lange zu warten, bis ein bestimmter interner Speicherzustand eintritt. Was das genau bedeutet, erfahren Sie noch. Nur ein Beispiel. Wenn Sie wollen, daß der Computer wartet, bis irgendeine Taste gedrückt wird, geben Sie ein

```
WAIT 198,1
```

Damit kommt man ganz elegant um ein GET- oder INPUT-Kommando herum, wenn es nur darum geht, irgendeine Taste zum Weitermachen zu drücken – zum Beispiel, wenn der Anwender Ihres Programms einen Text lesen und dann mit dem Drücken irgendeiner Taste dem Programm Bescheid geben sollte, daß er jetzt alles gelesen hat und weitermachen kann.

In genau dieser Situation sind jetzt auch Sie. Sie haben Ihr erstes Kapitel hinter sich. Die Maschine ist gelandet. Kapitän BASIC und seine Mannschaft verabschieden sich. Wir hoffen, der Flug hat Ihnen gefallen.

Nun ist es an Ihnen, mit uns weiter ins Landesinnere vorzudringen und das Innenleben Ihres Commodore zu erforschen. Die wichtigsten Grundlagen haben Sie während unseres gemeinsamen kurzen «Fluges» erhalten.

In diesem Sinne: Auf zu neuen Taten!

2
Die Commodore-64-Grafikzeichen

«Das steht halt so im Betriebssystem»

Der Commodore 64 gehört zu den wenigen Computern, mit denen man relativ schnell recht ansehnliche Grafiken aufbauen kann.

Das ganze Geheimnis liegt in den Tasten mit den Grafikzeichen. Daß Commodore bei diesen Zeichen nicht gerade das Seelenheil von Geschäftsleuten, also Verkaufsstatistiken oder Umsatzentwicklungen, im Kopf hatte, läßt sich recht schnell an den vorkommenden Zeichen erkennen, wenn auch Umsatzstatistiken durch Herzchen dargestellt sicherlich sympathischer wären. Die Grafiksymbole verführen eher zum Spielen als zur Statistik.

Um komplexere Dinge zumeist schematisch darstellen zu können, reicht ein Symbol allein nicht aus. Erst die Kombination von verschiedenen Zeichen läßt ein Bild ähnlich einem Mosaik entstehen. Dazu stehen allerhand Striche, Kreisstücke, Schräglinien und Dreiecke zur Verfügung. Probieren Sie es doch mal aus.

Ehe wir zu den Möglichkeiten kommen, die sich damit bieten, ein paar Vorbemerkungen:

Die Grafik- oder Sonderzeichen, die in ihrer ganzen Pracht auf den Tastenvorderseiten abgebildet sind, können mit der ⟨SHIFT⟩- oder der ⟨C=⟩-Taste erreicht werden: mit ⟨SHIFT⟩ die rechten, mit ⟨C=⟩ die linken.

Wenn Sie jetzt versucht haben, eine kleine Grafik zu erstellen, aber statt eines Männchens nur einen Buchstabensalat auf dem Bildschirm haben, hat das einen einfachen Grund: Um diese Grafikzeichen benutzen zu können, muß der Computer im «Großschrift-Modus» sein, also beim Schreiben nur Großbuchstaben zeigen.

Warum das so ist, wollen wir kurz erklären. Diese Erklärung ist weder

fundamental noch übermäßig wichtig. Es bleibt Ihnen überlassen, sie zu lesen oder nicht.

Dem Computer fällt es ja meist nicht ganz leicht, uns Menschen geistig zu folgen. Das heißt, egal ob Sie «Du Trottel» oder «Hallo» eintippen, der Computer antwortet immer nur lakonisch mit «SYNTAX ERROR».

Was lernen wir daraus? Er kann willkürliche Buchstabenkombinationen nicht verstehen.

Wer das nicht glaubt, sollte mal in die Computerabteilungen der Kaufhäuser gehen. Wenn der Abteilungsleiter – absichtlich oder unabsichtlich – vergessen hat, in einen der Computer ein Programm zu laden (oder mit anderen Worten: Ist es auf einem der Geräte nicht möglich, Außerirdische abzuknallen oder Bonbons zu futtern oder Frösche sicher über die Straße zu bringen), findet man dafür immer wieder Bildschirme mit seltsamen Nachrichten, zum Beispiel «MMMMMMM» oder «ZXZXZXZX 81» oder andere unsinnige Buchstabenkombinationen. Hat derjenige, der hier die Tastatur bzw. den Computer ausprobieren wollte, zufällig auch ein ⟨RETURN⟩ eingegeben, steht darunter noch ein «SYNTAX ERROR» und ein hilflos blinkender Cursor.

Irgendwann betritt dann ein Jüngling die Szene, lächelt breit ob der Dummheit seiner Mitmenschen, legt sein Kindergartentäschchen neben der Computer und zaubert ein kleines Programm auf den Bildschirm.

Entsprechend seiner Erfahrung ist das Programm. Meistens bleibt es bei einem Zweizeiler. Sehr beliebt ist zum Beispiel:

```
10 PRINT "WER DAS LIEST IST DOOF!"
20 GOTO 10
```

Trotz der offensichtlichen Frechheit führt der Computer dieses Programm klaglos aus.

Soweit unser kleiner Exkurs. Der Computer tut also nichts weiter, als bestimmte Buchstabenkombinationen irgendwo im Speicher abzulegen und bei Gelegenheit wieder hervorzukramen. Nur legt er eigentlich nicht Buchstaben ab, sondern Zahlen. Warum, erklären wir später. Diese Zahlen heißen *ASCII-Codes*. Jede Taste hat ihre eigene Zahl. Entsprechend dieser Zahl gibt der Computer das passende Zeichen aus. Er prüft es nicht auf Sinn und Gehalt (zumindest nicht, wie wir das tun würden), sondern er prüft es eigentlich gar nicht.

Und jetzt wird's spannend: Gibt der Computer nämlich, wenn Sie die ⟨SHIFT⟩-Taste und die ⟨S⟩-Taste gleichzeitig drücken, kein S aus, sondern

ein Herzchen, dann folgt daraus: Die Taste ⟨S⟩ hat zusammen mit ⟨SHIFT⟩ einen anderen Zahlenwert oder ASCII-Code. Und wenn Sie die ⟨C=⟩-Taste und ⟨S⟩ gleichzeitig drücken? Richtig. Genau dasselbe Spielchen. Damit sind wir wieder beim Großschreibmodus. Bei der Textverarbeitung, also wenn Groß- und Kleinschreibung benötigt werden, braucht man ⟨SHIFT⟩ wie bei der Schreibmaschine für die Großbuchstaben. Damit geht in diesem Modus eine Hälfte unserer Grafikzeichen verloren, weil die ASCII-Codes zum Unterscheiden von Groß- und Kleinbuchstaben gebraucht werden. Was bleibt, sind nur die Grafiksymbole, die zusammen mit der ⟨C=⟩-Taste zur Verfügung stehen. Wie gesagt: Das war alles nicht fundamental wichtig, hilft aber hoffentlich dabei, den Computer etwas besser zu verstehen. Also, Sonderzeichengrafik am besten nur im Großschriftmodus. Andernfalls geht ein Großteil der tatsächlichen Möglichkeiten verloren.

Zwei Möglichkeiten, Ihre Grafiken zu schützen

Sie können sich natürlich mit einer Keule neben Ihren Computer stellen, denn wenn Sie mittlerweile mit viel Fleiß und Mühe eine schöne Grafik gebastelt haben, wollen Sie selbstverständlich nicht, daß der großartige Eindruck nur deshalb Schaden nimmt, weil irgendeiner aus Versehen die falschen Tasten gedrückt hat (nämlich ⟨C=⟩ und ⟨SHIFT⟩ gleichzeitig). Sie können es zwar selbst gern einmal ausprobieren, aber wie gesagt: Der Eindruck leidet. Um dieses imagezerstörende Problem zu lösen, gibt es eine einfache Methode, nämlich den Befehl

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

Dadurch wird die Umschaltung blockiert. Damit Sie den Computer nicht ausschalten müssen, gibt es zwei Möglichkeiten, in den Normalzustand zurückzukommen:

Entweder ⟨RUN STOP⟩ und gleichzeitig ⟨RESTORE⟩ drücken, was allerdings sämtliche Sonderfunktionen, wie geänderte Farben und dergleichen, ebenfalls ausschaltet. Weil dabei aber auch dummerweise unsere Grafik gelöscht wird, gibt es, um die Blockade allein wieder aufzuheben, den Befehl

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

Bei der Gelegenheit noch zwei Tips: Derselbe Befehl mit CHR$(14) schaltet

den Computer z. B. innerhalb eines Programms von Groß- auf Kleinschreibung um. CHR$(142) tut das Gegenteil.

Fragen Sie aber bitte nicht, warum es ausgerechnet diese Zahlen sein müssen. Es gibt dafür wohl keinen besonderen Grund. Hannes' Standardantwort darauf war: «Das steht halt so im Betriebssystem.» Also nehmen Sie es hin.

Noch ein kleiner Tip am Rande: Einen ganzen, vollen Block erreicht man, indem man eine invertierte Leerstelle druckt. Klingt toll, was? Heißt aber nur, die Funktion RVS ON einschalten (⟨CTRL⟩ und ⟨9⟩), die Leertaste drücken und Reverse mit ⟨RVS OFF⟩ wieder ausschalten. Hier ein paar Beispiele, was mit Sonderzeichen alles möglich ist.

Bild 2.1 Beispiele für Grafikzeichen

Wie die Grafik laufen lernte

Jetzt kommen wir langsam zur Kernfrage: «Was soll ich damit und was hat das alles mit Spielen zu tun?» Ganz einfach. Alles, was wir jetzt die ganze Zeit direkt ausgeführt haben, läßt sich auch innerhalb eines Programms tun. Wie man das macht, wollen wir jetzt erklären.

Machen Sie doch mal spaßeshalber Anführungszeichen auf (⟨SHIFT⟩ und ⟨2⟩). Okay? Gut. Jetzt versuchen Sie bitte, Ihren Cursor mit den Steuertasten zu bewegen. Solange Sie ihn nach rechts steuern, scheint ja alles in Ordnung auch wenn er so komische invertierte Zeichen hinterläßt. Aber wenn Sie ihn nach oben, unten oder links bewegen wollen ...

Keine Sorge. Ihrem Commodore geht's gut. Bevor wir sagen, was das alles soll, drücken Sie doch bitte ⟨RETURN⟩. Den folgenden SYNTAX ERROR ignorieren Sie am besten. Das sagen Computer immer, wenn sie nicht weiterwissen.

Das ⟨RETURN⟩ sollten Sie eigentlich auch nur eingeben, um wieder aus diesem Modus herauszukommen. Sie müßten jetzt auf Ihrem Bildschirm allerhand invertierte Zeichen sehen.

Frage an Radio Eriwan: «Haben diese Zeichen irgendeinen Sinn?» Antwort: «Im Prinzip ja.» Nur, damit sie diesen Sinn auch erfüllen, fehlt noch eine Kleinigkeit, und zwar der BASIC-Befehl PRINT.

Probieren Sie es mal. Schreiben Sie PRINT, machen Sie Anführungszeichen auf, und versuchen Sie jetzt, den Bildschirm zu löschen (also: ⟨SHIFT⟩ + ⟨CLR HOME⟩). Das erste, was Ihnen auffallen wird, ist, daß jetzt genauso-wenig passiert wie vorhin bei der Cursorsteuerung. Bis auf eines: Gleich nach dem Anführungszeichen erscheint ein invertiertes Herzchen. Jetzt versuchen Sie wieder, den Cursor zu steuern. Am besten ein paarmal nach unten, ein bißchen nach rechts usw.

Stück für Stück füllt sich die Zeile mit immer mehr von diesen komischen Zeichen. Wenn Sie keine Lust mehr haben, können Sie jetzt aufhören. Mehr als zwei Zeilen sollten Sie aber auf keinen Fall auffüllen, denn länger darf eine BASIC-Eingabe nicht sein. Machen Sie dann die Anführungszeichen zu. Übrigens, Ihre Cursorsteuerung funktioniert wieder normal, aber probieren Sie es bitte noch nicht aus.

Bevor Sie ⟨RETURN⟩ drücken, wollen wir Ihnen sagen, was passiert ist. Sie wissen ja, daß der Computer bestimmte Befehle verstehen und ausführen kann. Und daß man diese Befehle in Programmen abspeichern und später ausführen lassen kann. Genau das haben wir getan – außer, daß wir vor unsere Programmzeile keine Zeilennummer geschrieben haben und der Computer sie in dem Moment ausführt, in dem wir ⟨RETURN⟩ drücken. Moment!

Schauen Sie sich Ihre Zeile noch einmal an: Zuerst steht da PRINT, also ein Befehl, der den Computer anweist, etwas auf dem Bildschirm darzustellen. Nur folgen diesmal dem Anführungszeichen keine Wörter oder Buchstaben, sondern unsere Zeichen. Und diese Zeichen kamen ja dadurch zustande, daß wir versucht haben, den Bildschirm zu löschen und den Cursor zu bewegen.

Wenn diese Zeichen jetzt auch noch Steuerzeichen heißen, dann kann man sich an fünf Fingern abzählen, was passieren wird. Der Computer wird alle diese Zeichen als Befehle interpretieren.

Das heißt, zuerst den Bildschirm löschen und dann den Cursor an die Stelle bewegen, die wir mit unseren Steuerzeichen angesteuert hatten. Drücken Sie also jetzt ⟨RETURN⟩. Wenn Sie alles richtig gemacht haben, müßte Ihr Bildschirm jetzt leer sein, irgendwo READY stehen und darunter der Cursor blinken.

Ein Nebeneffekt dabei, den man leicht übersieht, weil man ihn eben nicht sieht, ist, daß der Cursor nicht langsam dahin wandert, wo wir ihn haben wollten, sondern scheinbar sofort dort auftaucht. Aber eben nur scheinbar.

Denn tatsächlich führt der Computer alle Steuerzeichen eins nach dem anderen aus. Nur unglaublich schnell. Dieser Nebeneffekt führt uns zum zweiten Teil der Frage: Was hat das Ganze mit Spielen zu tun?

Spiele sind Animation. Computeranimation aber, also die scheinbare Bewegung von Figuren auf dem Bildschirm, wird ja nur dadurch erreicht, daß ständig Punkte gezeichnet und wieder gelöscht werden. So schnell, daß das menschliche Auge diesem Vorgang gar nicht folgen kann. Es läßt sich täuschen.

Mit den Steuerzeichen in einer Programmzeile können wir den Cursor genau an einer bestimmten Stelle zeichnen lassen. Wir können Reverse an- und ausschalten, den Bildschirm löschen usw.

Bevor wir Sie jetzt mit noch mehr Theorie quälen, ein bißchen Denksport: Wenn Sie also wissen, wie man Grafikzeichen auf den Bildschirm bringt, wie man diese Zeichen an die beabsichtigten Stellen bekommt, daß das alles extrem schnell geht und daß Computeranimation eigentlich nur das schnelle Zeichnen und Löschen von Grafiken an bestimmten Stellen ist, dann müßte Ihnen jetzt auch ansatzweise klar sein, was das Ganze mit Spielen zu tun hat. Denn mit unseren Grafiksymbolen und Steuerzeichen müßte sich doch eigentlich auch einfache Animation machen lassen. Sie müssen ja nichts weiter tun, als an derselben Stelle ein Grafikzeichen durch ein anderes zu ersetzen. Am besten versuchen Sie es erst einmal selbst. Sie können aber auch unser dreizeiliges (!) Programm «HUGO» benutzen, das im Listinganhang steht. Spätestens damit können Sie schon bei allerhand Leuten Eindruck schinden. Bei «HUGO» finden Sie übrigens noch einmal eine Erklärung der einzelnen Programmzeilen.

Wundern Sie sich nicht, wenn wir für das Hauptprogramm die etwas eigenwilligen Zeilen 1000 bis 1020 benutzt haben: Der Grund dafür liegt darin, daß wir Ihnen vorher und nachher genug Platz lassen wollten, Hugo zum Beispiel in einer hübschen Wohnung turnen oder sich andere zusätzliche Dinge einfallen zu lassen.

Wir schlagen vor, daß Sie sich jetzt erst einmal mit diesen ganzen Sonder- und Steuerzeichen ausgiebig beschäftigen, bevor Sie das nächste Kapitel lesen. Denn wenn wir davon ausgehen, daß Sie das Buch gegen 10 Uhr morgens gekauft haben, dann reicht die Zeit gerade noch so, bevor Sie zu Mittag essen sollten. Wenn Sie das Buch nachmittags gekauft haben, dann ist es auch nicht mehr weit zum Abendessen. In diesem Sinne viel Spaß. Und guten Appetit.

3
Der Speicheraufbau des Commodore 64

Also, mit SYS 61 320 bin ich eigentlich deutlich im Betriebssystem ... Oh!!

Na, lieber Leser? Alles gut verdaut? Alle Informationen und das Essen? (Egal, ob Mittag oder Abend ...)

Fein.

Wenn Sie bisher das Gefühl hatten, ein wenig unterfordert zu sein («Tasten drücken kann doch jeder!»), dann ist das folgende Kapitel sicherlich etwas für Sie. Denn jetzt wird es ein bißchen schwieriger. Vom technischen Verständnis her ist dieses dritte Kapitel sicherlich eins der anspruchsvollsten.

Es beschäftigt sich mit den komischen Dingern, von denen Programmierer ständig reden – egal, ob sie einer darum gebeten hat oder nicht: den Bits und Bytes.

Der Sinn dieses Kapitels soll sein, Ihr Verständnis für die Vorgänge im Computer zu schulen und Sie vor allerlei Überraschungen zu schützen, die vor allem beim POKEn und bei SYS-Aufrufen auf Sie warten. Das Zitat in der Überschrift beispielsweise enthält einen wichtigen Begriff. Das ist nicht etwa die Zahl 61320 oder der Ausdruck «Betriebssystem». Nein, es ist das «Oh!!!» am Schluß.

Es zeugt davon, welch erstaunliche Wirkung bestimmte Speicherzellen haben können. Beim zitierten «Oh» handelt es sich um den Ausdruck ungläubigen Erstaunens, als der Computer sich, ganz gegen Hannes' Absicht, sang- und klanglos von uns verabschiedete. Nicht einmal seinen üblichen «SYNTAX ERROR» brachte er noch heraus. Er war abgestürzt. Abstürzen heißt nichts anderes, als daß das System sich irgendwo verrennt und nicht mehr herauskommt. Übrigens, wenn Sie unseren SYS 61 320 vorhin ausprobiert haben, sollten Sie jetzt Ihren armen Commodore durch gnädiges Ausschalten erlösen.

Ein falscher POKE oder SYS kann den Computer wirklich «sprachlos» machen.

Dies ist zwar manchmal ganz lustig, aber wenn man gerade allerhand Programmzeilen eingegeben und beim Probelauf die falsche Speicherzelle angesprochen hat, kann das schon ärgerlich sein. Vor allem, wenn die einzige Möglichkeit, den Computer wieder hinzukriegen, der kleine Knopf auf der rechten Seite ist – ja, der zum Ein- und Ausschalten. Wenn Sie zumindest ungefähr wissen, wo Sie herumwerkeln, lassen sich damit schon allerhand Gefahren rechtzeitig abschätzen. Das Wissen um die Speicheraufteilung hat noch einen anderen Effekt. Vor allem Benutzer von Kassettenrecordern als Massenspeicher werden das zu schätzen wissen: Es ist nämlich möglich, mehrere Programme gleichzeitig im Speicher zu haben. Das erspart Wartezeiten. Auch für Floppy-Besitzer ist das praktisch:

Umrechnungsprogramme oder andere *Utilities* können abgelegt und abwechselnd aufgerufen werden.

Von Prozessoren, RAM, ROM und anderen Chips

Bevor wir uns solchen Programmiertechniken zuwenden, müssen wir uns ein etwas umfassenderes Wissen über den Speicher aneignen. In diesem Sinne: Stürzen wir uns in die Chips.

Apropos Chips. Das ist bereits der erste Begriff, über den man sich klar werden muß. Außer mit Paprika, gesalzen, ungarisch oder in der schlichten Kartoffelausführung gibt es diese Dinger auch im Computer. Prinzipiell lassen sich diese elektronischen Bauteile, die à la Lego auf eine Platine gesteckt werden, in drei Hauptgruppen unterteilen: Die drei Kategorien sind: Prozessoren, Speicherbausteine und Chips, die besondere Aufgaben haben und speziell dafür konstruiert wurden, wie zum Beispiel der Videochip oder der Soundchip des Commodore 64. Zuerst zu den Prozessoren. In diesem Baustein finden die Schaltvorgänge statt, die den Computer dazu bringen, auf jede (für ihn mehr oder weniger sinnvolle) Eingabe eine für uns (ebenso mehr oder weniger sinnvolle) Ausgabe folgen zu lassen. Er ist die sogenannte «Zentraleinheit».

Im Fall unseres Commodore 64 heißt dieser Chip 6510. Hinter diesem fantasievollen Namen stecken außer zahlreichen Transistorfunktionen auch 40 metallene Füßchen, die in etwas selbstherrlicher Weise den Computer despotisch beherrschen – frei nach Ludwig XIV.: «Der Schaltkreis bin ich.»

Na ja, zumindest glaubt dieser kleine Mikroprotz daran. Und ganz unrecht hat er ja nicht, da er in seiner Klasse sicherlich zu den Fähigsten gehört. Allerdings – ganz so allein, wie man meist annimmt, beherrscht er sein Mikroreich nicht.

Es gibt da noch andere Bausteine, die ihre eigenen und ganz speziellen Aufgaben haben, zum Beispiel den VIC, den Video-Interface-Chip, der die gesamte Bildschirmausgabe kontrolliert. Er ist kein Prozessor, davon gibt es nämlich nur einen in unserem Computer. Da aber im Ernstfall die Videoinformationen, also die Bildschirmdarstellung, vorgehen, ist der VIC sogar in der Lage, den Prozessor «warten» zu lassen.

Das also sind König und Kardinal.

Jetzt kommen wir zu den Untertanen: den Speicherbausteinen. Ihre Aufgabe besteht nur darin, Informationen zu speichern. Sie unterteilen sich in zwei Gruppen: die löschbaren und die nichtlöschbaren. Während man die einen mit Musikkassetten vergleichen kann, weil auf der Kassette die Informationen (Musik) gelöscht werden können, wenn man sie nicht mehr braucht, sind die anderen wie Schallplatten. Sie können nicht mehr gelöscht werden und spielen immer dasselbe Lied.

Weil Computer-Freaks ein Faible fürs Englische haben, heißt der erste Speichertyp «RAM» (Random Access Memory, was freier Zugriffspeicher bedeutet, besser aber mit Schreib-/Lese-Speicher umschrieben wird). Der RAM-Bereich wird bei jedem Ausschalten des Computers gelöscht. Der andere Typ ist das «ROM». Das hat nichts mit der gleichnamigen Stadt zu tun und wird auch kürzer gesprochen. Ausgeschrieben heißt «ROM» Read Only Memory oder Nur-Lese-Speicher. Im Gegensatz zum RAM wird das ROM beim Ausschalten nicht gelöscht, kann aber auch nicht beschrieben werden. Das heißt, daß alle Werte darin absolut feststehen, wenn sie erst einmal festgelegt wurden.

Strom oder nicht Strom, das ist hier die Frage

Aber egal ob ROM, RAM oder Prozessoren – alle diese *ICs* können nur zwei Zustände unterscheiden: Strom oder kein Strom. Und nur in dieser Art können sie Informationen verarbeiten und speichern. Natürlich stellt sich die Frage, wie es der Computer fertigbringt, sich beispielsweise den Namen Brigitte zu merken, und das mit seiner Strom-an-/Strom-aus-Methode.

Erinnern Sie sich noch an unsere ASCII-Codes vom letzten Kapitel – die Zahlenwerte für Buchstaben?

Na prima. Da hatten wir schon gesagt, daß der Computer sich keine Buchstaben merkt, sondern alle Zeichen als Zahlen abspeichert. So hat zum Beispiel der Buchstabe A den Wert 65.

Wenn Sie es nicht glauben, probieren Sie es aus. Geben Sie folgende Zeile ein:

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

Als Ergebnis werden Sie 65 bekommen. Das heißt, der Computer hat nirgends in seinem Speicher ein A rumliegen, sondern nur die Zahl 65. Wenn er die an geeigneter Stelle antrifft, dann fängt er an, so lange rumzuwerkeln, bis ganz zum Schluß ein A auf dem Bildschirm steht. Aber wahrscheinlich quält Sie schon seit Anfang dieses Absatzes die Frage, wie der Computer nun die Zahl 65 in Strom an/aus umsetzt. Also: Irgendwann kam ein schlauer Mensch darauf, wie man sich die Denkvorgänge im Innern eines Computers am besten vorstellen kann. Die Lösung bestand aus zwei Zahlen: 1 und 0.

Merken Sie was? 1 und 0, an und aus, Strom und kein Strom.

Man kann alle unsere *Dezimalzahlen* in dieses System umrechnen. Das Problem bei dieser Rechnerei sind bloß wir Menschen. Denn offensichtlich hat man bei unserer Konstruktion, zumindest bei den Händen, den technischen Fortschritt völlig außer acht gelassen. Wir haben nun mal an jeder Hand fünf Finger. Und weil wir zum Rechnen schon immer Hilfsmittel gebraucht haben ... Sie sehen also, widrige Umstände sind daran schuld, daß die meisten Menschen mit diesem Dualsystem nicht so zurechtkommen. Denn dank unserer zehn Finger wurde die Zehn für uns zu einer erhabenen Zahl.

Wenn man sich aber einmal wirklich mit diesem Dualsystem beschäftigt, stellt man fest, daß es zwar umständlicher als unser Dezimalsystem ist, aber keineswegs schwieriger.

Wie rechnet man mit einem Computer, der nicht bis 2 zählen kann?

Eine Stelle kann bei uns mit den Ziffern 0 bis 9 besetzt werden – egal, ob Einer, Zehner oder Hunderter. Wir machen schlicht und einfach bei allen Zehnerpotenzen eine neue Stelle auf. Also 1, 10, 100, 1000, was 10^0, 10^1, 10^2, 10^3 entspricht.

Der Computer behauptet aber, aus seiner Sicht mit Fug und Recht, daß 1 + 1 nicht 2, sondern 10 sei.

Sein Problem ist, daß er nicht bis zwei zählen kann. Oder besser gesagt, er kennt diese Zahl gar nicht.

Hätte der Mensch seit jeher nur zwei Finger, wäre dieses System für uns das Natürlichste der Welt. Zum Glück für alle Schreibmaschinenschulen ist es anders. Trotzdem, dank unseres Gehirns können wir uns mit dem Computer in seinem etwas eintönigen Ein/Aus-Dialekt unterhalten. Dazu rechnen wir in seinem System. Statt bei jeder Zehnerpotenz machen wir jetzt also bei jeder Zweierpotenz eine neue Stelle auf. 2^0 (= 1) ist also 1; 2^1 (= 2) ist 10; 2^4 (= 16) ist 10000 usw.

Womit auch unser kleines Rechenbeispiel von vorhin klar wäre. Während 1 + 1 für uns 2 ist, muß der Computer daraus 10 machen. Wir sollten zum besseren Verständnis nur nicht zehn, sondern eins-null sagen. Was ist dann also 3 im Dualsystem? Richtig, 11 (eins-eins). Denn wir haben ja bei 3 im Dualsystem noch eine Stelle frei und müssen auch keine neue aufmachen. Zwei ist 10 (eins-null). Eins ist auch dual 1. Also zusammen 11 (eins-eins). Voilà.

Wenn Sie das alles verstanden haben: großartig. Zur Belohnung noch eine kleine Aufgabe. Rechnen Sie doch mal schnell 10110110100110101010 1 ins Dezimalsystem um. Wenn Sie aber bis morgen früh etwas Konstruktiveres tun wollen, können Sie sich dazu auch ein Programm überlegen. Wenn Sie im Gedächtnis behalten, daß der Wert der ersten Stelle eins, der zweiten zwei, der dritten vier usw. ist, dann ist das gar nicht so schwer. Allerdings finden Sie im Anhang ein Listing für ein entsprechendes Programm. In jedem Fall würden wir Ihnen raten, sich noch etwas mit diesen Zahlen zu beschäftigen. Sie werden Ihnen bald sehr hilfreich sein. Haben Sie das Binärsystem erst einmal verstanden, ist die größte Hürde bei Bits und Bytes genommen. Jetzt machen Sie ruhig einmal Pause.

Dazu hier Tabelle 3.1

dezimal	dual
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Tabelle 3.1 Gegenüberstellung dezimal/dual

Über das Glück, einen 8-Bit-Prozessor zu besitzen, und vom Pech, 16 Bit zu brauchen

Schön, daß Sie wieder da sind. Startbereit zum nächsten großen Abschnitt? Wenn nicht, lesen Sie lieber den ersten Teil noch einmal. Das Buch nimmt Ihnen ja keiner mehr weg. (Oder stehen Sie etwa immer noch bei Ihrem Buchhändler?!) Im folgenden Abschnitt gehen wir in die Tiefe des Speichers.

Die Nullen und Einsen, von denen wir die ganze Zeit gesprochen haben, sind die berühmten Bits, die kleinste Informationseinheit, die ein Computer verarbeiten kann. Bit ist die Kurzform von binary digit. Und jetzt geht's los.

Wir haben in unserem Commodore nämlich einen 8-Bit-Prozessor. Das ist der kleine Despot von vorhin, der 6510. 8-Bit-Prozessor heißt aber nichts weiter, als daß er acht dieser kleinsten Informationen gleichzeitig verarbeiten kann. Das Kuriose dabei ist, daß unser Chip aber auch mit einer 16-Bit-

Leitung gekoppelt ist. Durch einen technischen Trick kann unser 6510 diese 16 Bit sogar ausnutzen. Aber wozu das Ganze? Diese Leitung, von der wir sprechen, heißt Adreßbus. Und genau dazu dient der ganze Aufwand auch: zu einer Art Adressenverwaltung. Gemeint sind die Adressen, um das Bild von vorhin wieder aufzugreifen, des gesamten Königreichs Computer. Der Prozessor als Quasi-König muß ja die Möglichkeit haben, mit jedem seiner Untertanen, also den RAMs und ROMs, Verbindung aufzunehmen. Immer, wenn Könige bisher auf diese Möglichkeit verzichteten, ging die Sache ins Auge. Die Geschichte beweist das. Sollte Ihnen das Bild von König und Reich zu infantil sein, der Fachmann sagt dazu: «Der Prozessor muß in der Lage sein, jedes Byte zu adressieren.»

Wenn Sie Ihren Commodore einschalten, erfahren Sie, daß er ein 64-KByte-RAM-System hat. 8 Bit entsprechen einem Byte. Damit läßt sich jetzt auch ausrechnen, wie viele Bytes Sie zur Verfügung haben sollten. Ein KByte heißt eigentlich ein Kilobyte. Das hat nichts damit zu tun, daß Speicherkapazität etwa pfundweise verkauft wird. Kilo ist nur ein traditioneller Ausdruck für 1000. Hier treffen allerdings moderne Technik und Tradition aufeinander. Der Computer ist ja, wie wir erfahren haben, ein absoluter Zweierfetischist. Tradition hin – Tradition her. 2^{10} ist halt nicht 1000, sondern 1024, und damit Schluß. Und weil er so dickköpfig ist, stimmt auch die Sache mit dem 64K-System nicht ganz. Denn 64mal ein Kilobyte (also 64 * 1024 Bytes) ist 65 535. (Eigentlich 65 536. Die Zählung der Speicherzellen beginnt aber mit der Zelle 0.) Damit haben wir auch die gesuchte Zahl. Was diese Zahl so wichtig macht? Eigentlich gar nichts, bis auf die Tatsache, daß der Prozessor sie gar nicht versteht. Und das stellt ihn eigentlich vor eine unlösbare Aufgabe. Sie erinnern sich: Ein Byte sind 8 Bit. Damit ist der höchste Wert, den ein Byte darstellen kann, logischerweise 11111111, und das ist 255. Rechnen Sie ruhig nach, wenn Sie es nicht glauben. Sie können es aber auch ausprobieren. Der höchste Wert, den man mit dem Befehl POKE in eine Speicherzelle schreiben kann, ist 255. Alles, was darüber liegt, quittiert der Computer mit einem kühlen "ILLEGAL QUANTITY ERROR".

Na, stimmt's?

Weil wir aber einen 8-Bit-Prozessor haben, ergibt sich hier das gleiche Problem. Die höchste Zahl, die er aufrufen kann, ist 255. Und genau deshalb kommt er in Schwierigkeiten. Wie soll er einen Wert aufrufen, der in der Speicherzelle 65 535 steht? Wo er doch diese Zahl gar nicht kennt. Aber genau deshalb gibt es den 16-Bit-Adreßbus. Denn die höchste Zahl, die man mit 16 Bit darstellen kann, ist 65 535. Was für ein Zufall ...

Damit lassen wir fürs erste diese ganze Ar-byte auf sich beruhen. Wir geben uns mit der Tatsache zufrieden, daß unser Prozessor die 16-Bit-Leitung benutzen und somit jeder Speicherzelle Bescheid sagen kann, daß sie ihren Inhalt vorbeischicken soll. Dieser Wert kommt dann – ganz legal – über eine andere, reguläre 8-Bit-Leitung zum Prozessor. Die 8-Bit-Leitung heißt übrigens Datenbus. Und weil jetzt alles so schön klar zu sein scheint, werden wir gleich wieder ein bißchen Unordnung reinbringen.

Wie einem ohne große Schwierigkeiten über 3000 Byte verlustig gehen

Die erste Meldung beim Einschalten Ihres Commodore 64 besagt, daß Sie stolzer Besitzer eines 64K-RAM-Systems sind. Schön, nicht? Nur ... Es stimmt leider nicht ganz. Aber nur die Ruhe! Was wir damit meinen, ist: Es ist zwar faktisch richtig, aber Sie haben keinen direkten Nutzen davon.

Zuerst muß zum besseren Verständnis gesagt werden, daß alle Programme, die den Computer zum Arbeiten bringen, unauslöschlich in ROM stehen. Auch diese Programme brauchen jedoch immer wieder Zwischenspeicher, wo sie Daten ablegen und aktualisieren können. Nehmen wir beispielsweise den Wert des *BASIC-Anfangs* (2048). Zwei Dinge müssen für diesen Wert möglich sein, damit der Computer damit arbeiten kann. Zuerst muß er das Ausschalten überleben. Wie wir wissen, geht das nur im ROM-Speicher. Andererseits soll er veränderbar bleiben. Aber das Ändern geht halt wieder nur im RAM. Einfachste Lösung dazu: den Wert, der an sich im ROM steht, beim Einschalten ins RAM zu kopieren. Fertig: damit hat man die berühmten zwei Fliegen mit der einen Klappe.

Natürlich braucht das Festlegen des BASIC-Anfangs nicht allein den halben Speicher. Es gibt noch einige andere, ähnlich gehandhabte Werte von der Cursorfarbe bis zur Länge des Kassettenpuffers. Das sind alles Dinge, die der Computer beim Einschalten von selbst erledigt, um sich häuslich einzurichten. Deshalb dauert es beim *Initialisieren* auch immer einige Sekunden, bis der Computer soweit ist. Ein anderer Speicherplatzbenutzer – außer dem Programmierer – ist der Bildschirm. In einem bestimmten Bereich des Speichers liegt eine «Kopie» Ihres Bildschirms. Also jeder Buchstabe, der bei Ihnen auf dem Fernseher erscheint, ist in einer Speicherzelle als Bildschirmcode abgelegt. Wenn Sie ⟨CLR HOME⟩ drücken, wird für Sie der Bildschirm gelöscht.

Tatsächlich wird aber dieser fast 1K große Bereich ausgeräumt. 1K deshalb, weil 1000 Zeichen auf dem Bildschirm dargestellt werden können, also müssen dafür 1000 Bytes herhalten. Und weil man gerade beim Speicherplatzklauen ist, werden gleich noch einmal so viele mitgenommen, um die Farbe jedes Zeichens anzugeben, da diese ja unterschiedlich sein könnte. Womit zusammengenommen weitere 2K futsch wären.

Weiterhin gibt es noch die sogenannte *Zeropage*. Diese «0-Seite» ist eine von 256, in die der ganze Speicher unterteilt werden kann. Jede *Seite* enthält dabei 256 Bytes. Unsere erste Seite geht also von 0 bis 255.

Wie das Inhaltsverzeichnis eines Buches ist diese Seite am schnellsten vom Computer zu erreichen. Das hat, zumindest in BASIC, zwei Gründe: Zum einen reicht bei 255 ohne große Tricks und Umwege die 8-Bit-Kapazität des Prozessors zum Adressieren aus. (Sie erinnern sich: Die höchste Zahl, die mit 8 Bits dargestellt werden kann, ist 255.) Der andere Grund ist, daß der Computer auf diese Seite zuerst stößt, wenn er sein internes «Speicherbuch» aufschlägt. Auf dieser Seite wird deshalb alles mögliche zwischengespeichert, was besonders oft und schnell gebraucht wird. Was hier so alles rumliegt, verrät Ihnen Ihr C-64-Handbuch ab Seite 160. In unserem PEEK&POKE-Anhang finden Sie dazu einige Tips für interessante Anwendungen.

Die Seiten 1, 2 und 3 (also 256 bis 1023) enthalten ebenfalls allerhand wichtige Informationen.

Aber trotz all unserer Erklärungen – bisher sind wir erst 3072 Bytes ärmer. Wo ist der ganze Rest geblieben?

Sag mir, wo die Bytes sind, wo sind sie geblieben?

Lassen Sie uns rekapitulieren: 38911 Bytes sind für BASIC frei – behauptet zumindest Ihr Commodore beim Einschalten. Von 3072 Bytes, also von satten 3K, haben wir uns im Verlauf der letzten Seiten trennen müssen. Daraus folgt, wir sind auf der Suche nach rund 23K, die dem Computer scheinbar irgendwo abhanden gekommen sind. Die erste wichtige Voraussetzung, um zu verstehen, wo das ganze Zeug geblieben ist, ist die Tatsache, daß der Prozessor des Commodore nur 65535 Bytes adressieren kann. Diese Zahl entspricht auch genau dem freien RAM-Bereich des Computers, und aus verschiedenen Gründen wollte Commodore auch keinesfalls auf diese 64K-RAM verzichten. Bis hierher ist das auch alles kein Problem – weder für uns beim Verstehen noch für den Prozessor beim Adressieren. Allerdings kann unser kleiner 6510

nun mal kein BASIC verstehen. Kunststück! Denn wenn er nur Strom oder kein Strom unterscheiden kann, ist klar, daß der Befehl PRINT einfach so für ihn ein böhmisches Dorf ist. Irgend etwas muß also unsere Befehle dem Computer übersetzen bzw. das PRINT interpretieren. Dafür gibt es ein eigenes Programm. Sein Name ist «BASIC-Interpreter». Dieses Programm steht unauslöschlich im ROM. Seine Aufgabe ist, kurz geschildert, BASIC-Befehle in Codezahlen, sogenannte Tokens, umzuwandeln. Damit kann der Computer den Befehl verstehen, ihn binär speichern und braucht so statt fünf Bytes für PRINT sogar nur ein Byte für das Token. Soweit ist alles klar, bis auf eins. Natürlich muß auch dieser ROM-Bereich irgendwie adressiert werden können. Denn wenn der BASIC-Interpreter gebraucht wird, muß der Prozessor ihn ja aufrufen können. Also muß er eine Adresse haben. Diese Adresse kann bekannterweise nicht höher als 65535 sein. Denn das könnte der Prozessor selbst mit der 16-Bit-Leitung nicht darstellen. Die einzige Möglichkeit, beide Dinge (RAM- und ROM-Bereich) unter einen Hut zu bringen, ist, beiden dieselbe Adresse zu geben. Weil der Computer aber nicht unterscheiden kann, für wen ein Aufruf z. B. mit der Adresse 41350 bestimmt ist, wird eine Entscheidung auf unsere Kosten getroffen. Ein 8K-RAM-Baustein wird abgeschaltet, das heißt, eigentlich ausgeblendet. Er ist zwar theoretisch funktionstüchtig, kann aber nicht eingesetzt werden, weil der Prozessor nur entweder RAM oder ROM adressieren, also aufrufen kann.

Nun gibt es außer dem BASIC-Interpreter noch das eine oder andere wichtige ROM-Programm.

Alles in allem hat der Commodore etwa 20K-ROM, zu denen auch der SID (der Tonchip) und der VIC (der Videochip) gehören, die auf diese Art und Weise mitverwaltet werden müssen. Entsprechend dieser ROM-Größe nimmt unser freier RAM-Bereich ab. Der Rest wird dann noch von einem 4K-RAM-Spezialspeicher benötigt, der zwischen BASIC und *I/O-Registern* liegt.

Daß es zwecks Speichererweiterung äußerst unpraktisch wäre, das BASIC-ROM und damit den Interpreter oder das Betriebssystem auszublenden, ist klar. Denn dann hätten Sie zwar mehr Speicher, aber leider sonst nichts mehr. Ohne BASIC und Betriebssystem ist es nämlich nichts mehr mit Programmieren. Der Computer stürzt ab und kann nicht einmal mehr mit ⟨RUN/STOP⟩ und ⟨RESTORE⟩ wieder in Ordnung gebracht werden. Wenn Sie das Ganze mal probieren wollen, die ROMs lassen sich mit

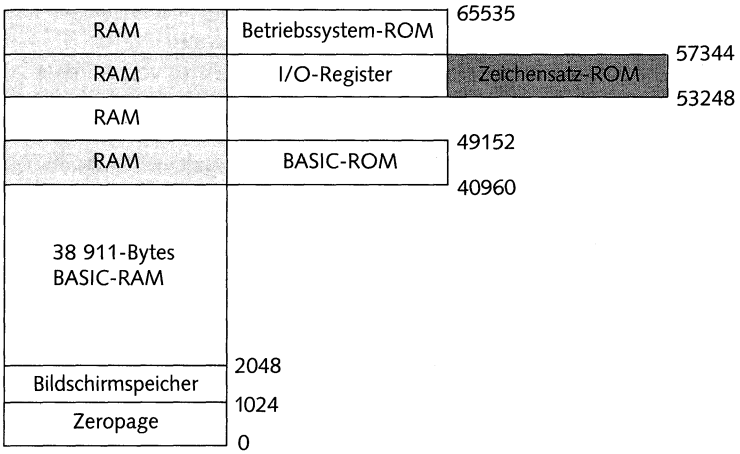
```
POKE 1,53
```

ausblenden. Das Ergebnis ist, daß Sie ohne Cursor und BASIC dastehen.

Noch einen letzten Satz zu den 64K-RAM. In Maschinensprache wäre es möglich, fast diesen ganzen RAM-Bereich zu benutzen. In BASIC müssen wir uns damit abfinden, daß ein gewisser Teil anderweitig gebraucht wird.

Aber keine Sorge. Bis Sie allein die restlichen 38911 Bytes aufgebraucht haben, dürften Sie schon allerhand zu tun haben. Zum besseren Verständnis des Ganzen dienen die folgenden Skizzen.

1. Normale BASIC-Konfiguration:



2. Mögliche Maschinensprachekonfiguration:

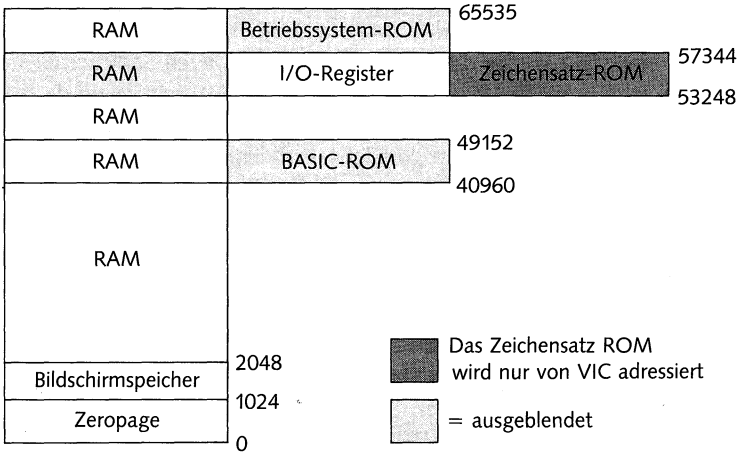


Bild 3.1 Speicherkonfigurationen

Wenn Bytes halbe-halbe machen

Bei normalen BASIC-Programmen kommt es nicht oft vor, daß man die ganzen 38911 Bytes dafür braucht. Mit einigen Tricks ist es aber möglich, genau diesen freien Speicherbereich auszunutzen – zum Beispiel, indem man mehrere Programme gleichzeitig im Speicher hat, aber immer nur eins arbeiten läßt.

Wie wir vorher gehört haben, beginnt der BASIC-Speicher bei 2048. Ab hier werden die bereits erwähnten Tokens abgespeichert.

Wenn Ihr Programm beispielsweise 2K lang ist, geht es von Adresse 2048 bis 4095 (BASIC-Anfang + 2mal 1024 Bytes). Ab 4096 legt der *Interpreter* dann die Inhalte der Variablen, wie A, A$ usw., ab. Damit sich der Computer das auch merkt, gibt es einige Adressen in der Zeropage, zum Beispiel von 43 bis 46. Dabei gilt, daß die Adressen 43 und 44 für den Anfang, 45 und 46 für das Ende des BASIC-Programms zuständig sind.

Warum jeweils zwei Adressen? Nach alledem, was wir in harter Arbeit über die Speicheraufteilung gelernt haben, können BASIC-Programme normalerweise nur innerhalb des Speicherbereichs von 2048 bis 40960 stehen. (Vergleichen Sie dazu auch die Skizze aus dem letzten Abschnitt.)

Damit sind wir wieder bei unserem alten Problem. Können Sie es sich denken? Genau. Es geht wieder mal ums Adressieren. 8 Bit sind maximal 255. Nichts ist's also mit 2048 bis 40960.

Aber wir haben ja zwei Speicherzellen. Nun, selbst damit erreichen wir nichts, denn die höchste Summe wäre dann 510, und das ist so gut wie nichts. Man könnte jetzt natürlich immer so viele Speicherzellen, wie nötig sind, addieren. Nur würde das einen wesentlich höheren Speicheraufwand bedeuten. Daß unser 6510, als die Adressierkapazität verteilt wurde, nicht gerade in der ersten Reihe stand, haben wir ja schon gemerkt.

Glücklicherweise ist auch hier wieder jemandem etwas eingefallen. Wie wäre es denn, wenn man 2 Bytes zur Darstellung benutzt, diese Bytes irgendwie unterscheidet und eins dann aber nicht mit den normalen 2^0 anfangen läßt, sondern mit 2^8 als erstem Wert? Das würde bedeuten, daß die erste Stelle des niedrigeren Bytes ganz normal den Wert 1 hat, die erste Stelle des höherwertigen Bytes aber bereits den Wert 256 darstellt. Damit kriegen wir wieder unsere magische Zahl 65535 zusammen. Da staunt der Fachmann, und der Laie wundert sich. Um jetzt in die ganze Konfusion wenigstens etwas System zu bringen, heißen die beiden Bytes schlicht und ergreifend Low Byte und High Byte.

Das alles schauen wir uns noch an einem Beispiel an.

Wie wir wissen, macht schon die Startadresse 2049 Schwierigkeiten. Binär sieht sie folgendermaßen aus:

100000000001

Dieser Wert wird nun in zwei 8-Bit-Werte geteilt. Weil unsere Binärzahl aber nicht aus 16, sondern nur aus 12 Stellen besteht, werden die fehlenden vorderen Stellen einfach mit Nullen besetzt. Damit ergibt sich

00001000 / 00000001

Das Low Byte, also der Wert, dessen erste Stelle 2^0 entspricht, kommt in die Speicherzelle 43.

Das High Byte, dessen erste Stelle 2^8 entspricht, kommt in die Speicherzelle 44.

Und jetzt noch mal dieses ganze Prinzip im Schema.

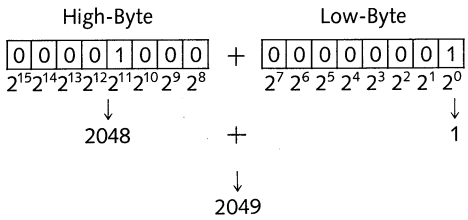


Bild 3.2 High-Byte-/Low-Byte-Prinzip

Sie können das ja mal ausprobieren: Mit

```
? PEEK (43)
? PEEK (44)
```

müßten Sie im ersten Fall 1, im zweiten Fall 8 als Ergebnis erhalten. Natürlich ergibt das nach einer Addition noch nicht die gesuchte Zahl 2049. Das liegt daran, daß Sie noch beide Bytes gleich behandelt haben. Das High Byte muß aber immer mit 256 multipliziert werden. Aus alledem ergibt sich die Formel

```
STARTADRESSE = 256 * PEEK (44) + PEEK (43)
  ENDADRESSE = 256 * PEEK (46) + PEEK (45)
```

Ein Tip: Fast immer ist die größte Speicherzelle (hier 44 bzw. 46) das High Byte. Wenn nicht, müßte es eigentlich angegeben sein.

Und noch etwas: In der «eigentlichen» Startadresse von BASIC muß immer 0 stehen. (Das ist eine Art Bestätigung, daß der Speicher wirklich frei ist.) Die BASIC-Tokens gehen dann eigentlich erst in der nächsthöheren Speicherzelle los.: Obwohl wir 2048 gemeinhin als «Start des BASIC-RAMs» bezeichnen, steht in dieser Zelle eine 0 (PEEKen Sie es doch einmal nach!). Der Zeiger in 43/44 zeigt tatsächlich erst auf 2049 (s. o.).

Im selben High-Byte/Low-Byte-Schema steht in den Speicherzellen 45/46 die Endadresse von BASIC. Sie muß übrigens immer mindestens 2 Bytes höher als der Anfang sein, selbst wenn kein Programm im Speicher ist. Das heißt, sie muß momentan als Grundwert 2051 enthalten.

Speicherzauberei

So, bis jetzt haben Sie nur gelesen. Jetzt geht's aber in die vollen! Wir wollen zwei BASIC-Programme gleichzeitig im Speicher haben. Zuerst setzen wir mit POKE 44,16 den BASIC-Anfang um 2K nach oben. Damit ist die BASIC-Startadresse 4097. Den Wert in 43 lassen wir unverändert 1.

Wir sollten aber unbedingt auch das Ende von BASIC nach oben setzen. Denn im Augenblick zeigt das Ende auf 2051, während der Anfang bei 4097 liegt. Diese Situation ist selbst für einen Computer etwas schwer durchschaubar. Also

```
POKE 45,3:POKE 46,16
```

Damit wäre dieses Problem gelöst. Wie bereits erwähnt, muß das BASIC-Speicherende 2 Bytes über dem Anfang liegen. Das wäre hiermit erledigt. Das Ende liegt jetzt nämlich bei 4099 (vgl. Formel für Endadresse!). Sie müssen aber hier auch die Zelle 43 verändern, denn der bereits erwähnte Interpreter verschiebt bei der Eingabe einer BASIC-Zeile das Ende des Programms automatisch nach oben.

Erinnern Sie sich noch an den kleinen Tip von vorhin?

Wenn Sie jetzt NEW eingeben oder LIST oder RUN, wird der Commodore diese Aktion deshalb mit einem SYNTAX ERROR quittieren, weil wir noch nicht in seinem neuen Zuhause (BASIC-Speicher) aufgeräumt haben. Ein

```
POKE 4096,0
```

käme hier einem Frühjahrsputz gleich.

Um wirklich absolut sicherzugehen, sollten Sie jetzt noch einmal NEW eingeben. Dieser Befehl hat die Eigenschaft, beide BASIC-Zeiger ins richtige Verhältnis zueinander zu bringen für den Fall, daß Sie sich doch verrechnet haben.

Jetzt können Sie endlich loslegen ... Schreiben Sie ein kleines Programm in den neuen Speicher, z. B.

```
10 PRINT"DIES IST DAS ZWEITE PROGRAMM!"
```

Nach dem Befehl RUN wird dieses Programm – wie üblich – ausgeführt. Jetzt aber setzen wir mit POKE 44,8 den Zeiger zurück auf 2048. Damit sind wir wieder im ursprünglichen BASIC-Bereich. Auch hier sollten Sie sicherheitshalber NEW eingeben. (Aufgeräumt hat der Computer beim Einschalten schon selbst.) Damit ist der Speicher bereit zur Eingabe eines neuen, anderen BASIC-Programms, weil der BASIC-Ende-Zeiger (in 45/46) automatisch richtig auf 2051 gesetzt wurde.

```
10 PRINT"DIES IST DAS ERSTE PROGRAMM!"
```

Was nach RUN passiert, brauchen wir Ihnen sicher nicht zu erklären. Doch jetzt kommt der praktische Nutzen der ganzen Mühe: Tippen Sie

```
POKE 44,16:RUN
```

Folge: Es läuft das zweite Programm

```
POKE 44,8:RUN
```

Und das wäre wieder Nummer eins.

Mit diesem POKE können Sie beliebig hin- und herschalten und das Programm aufrufen, das Sie gerade benötigen.

Aber gleich eine kleine Warnung: Programme, die Sie in dieser Art im Speicher haben, sollten Sie nicht mehr ändern oder abspeichern. Durch die Umschaltung kommen die Zeiger etwas durcheinander, und es kann zu recht seltsamen Reaktionen von seiten unseres 64ers kommen.

Nur wenn Sie diese Technik des Verschiebens und Umschaltens absolut sicher beherrschen, können Sie auch während des Programmierens gleichzeitig noch verschiedene Utilities im Speicher haben. Aber selbst dann sollten Sie etwas häufiger als sonst das Programm, an dem Sie gerade arbeiten, abspeichern, denn das Risiko, sich doch mal zu vertun, ist erheblich. Und das kann Sie Ihr Programm kosten.

Nach dem gezeigten Prinzip können Sie theoretisch beliebig viele Pro-

gramme gleichzeitig im Speicher halten. Lassen Sie aber immer genug «Abstand» zum nächsten Programm, und bereiten Sie die neuen Adressen erst mit den gezeigten POKEs und einem NEW vor.

Wenn Sie wissen, an welche Startadresse Sie z. B. Ihr zweites Programm legen wollen, können Sie unsere Formel genau umgekehrt anwenden: Teilen Sie die Adresse durch 256, und Sie haben den Wert, der in die Zellen 44 und 46 gePOKEd werden muß.

Vergessen Sie nicht, die mittlerweile bekannte 0 in die neue Startadresse zu POKEn.

Noch ein letztes Beispiel dazu: Sie wollen Ihr Programm ab der Zelle 10240 im Speicher haben:

Sie teilen 10240/256 = 40

Also wird gePOKEd:

```
POKE 43,1: POKE 44,40: POKE 45,3: POKE 46,40: POKE 10240,0: NEW
```

Auch bei diesem Thema heißt es natürlich zuerst probieren. Allerdings sollten Sie dabei ein bißchen Vorsicht walten lassen, da – wie gesagt – so verwaltete Programme nicht abgespeichert werden sollen. Arbeiten Sie also nur mit auf Diskette oder Kassette gesicherten Programmen. Gründliches Kennenlernen dieser Technik schützt Sie vor unangenehmen Überraschungen.

Das war's. Damit haben wir Ihnen so ziemlich alles über den Speicher verraten, was für Sie irgendwie wichtig ist.

Und damit kommen wir jetzt zu einem wesentlich amüsanteren Teil – zur eigenen Definition von Sonderzeichen.

4
Selbstdefinierte Zeichen

Warum der Computer Analphabet ist und was wir davon haben

Erst einmal möchten wir Ihnen gratulieren. Immerhin haben Sie sich durch die Wirrnisse und ständigen Adressierungsprobleme des Commodore-Speichers bis hierher durchgekämpft. Eine durchaus anerkennenswerte Leistung. Der Erfolg, den Sie davon haben, ist ein mittlerweile doch ganz ansehnliches Wissen über die internen Abläufe im Computer. Alles, was Sie so auf Ihrem Speicherrundgang gelernt haben, werden Sie in den folgenden Kapiteln und auch später bei Programmierproblemen immer wieder anwenden und gebrauchen können. Deshalb haben wir dieses Thema auch so ausführlich behandelt. Doch zurück zu den etwas erfreulicheren Dingen. Bevor wir Ihnen den Speicher erklärten, hatten wir uns mit Grafiken beschäftigt. Ja, sogar schon ein bißchen mit Animation. Dazu benutzten wir die fertigen Commodore-Grafikzeichen. Der Vorteil all dieser Zeichen ist, daß sie sich wie Text ganz einfach mit PRINT, Steuer- und TAB-Anweisungen über den Bildschirm bewegen lassen. Der einzige Nachteil bisher war, daß das Zeichen, das wir gerade brauchen könnten, leider nicht existiert. Wenn Sie dieses Kapitel gelesen und verstanden haben, sind Sie dieses Problem los. Ab dann können Sie sich alles, was Sie brauchen, selbst definieren.

Um dabei allerdings auch bald mit entsprechenden Erfolgen glänzen zu können, müssen wir noch schnell ein oder zwei Dinge besprechen, die hinter der Zeichendarstellung des Commodore 64 stecken.

Am wichtigsten ist zuerst einmal die Tatsache, daß alles, was mit Grafik zu tun hat – oder besser gesagt, alles, was mit Darstellungen auf dem Bildschirm verbunden ist – von einem Chip gesteuert wird: dem VIC-II-Videochip. VIC

ist dabei nicht die Abkürzung für den Namen Victor (das ist der Steward vom Traumschiff, also bitte nicht verwechseln ...), sondern für die Bezeichnung Video Interface Chip. Von diesem Baustein, im folgenden nennen wir ihn einfach VIC, haben wir schon im Speicheraufteilungskapitel gehört. Seine einzige Aufgabe besteht im Management der Bildschirmdarstellung. Damit hat er genug zu tun. Er ist nicht nur dafür verantwortlich, wann welches Zeichen wo auf Ihrem Fernsehschirm erscheint, sondern auch für die Farben des Hintergrundes, des Rahmens und der Zeichen. Damit er das alles tun kann, muß er sich allerhand Informationen holen, zum Beispiel aus dem Bildschirm-RAM. Dieser Bereich ist das Abbild dessen, was Sie auf dem Schirm sehen. Aus dem RAM kann VIC erkennen, welche Zeichen auf dem Schirm stehen sollen, und ein entsprechendes Fernsehbild aufbauen. Alles, was auf dem Bildschirm erscheint, ist hier in speziellen Codes abgelegt. Diese Codes sind übrigens nicht identisch mit den ASCII-Werten. Sie finden diese Bildschirm-Codes auf Seite 133 in Ihrem Commodore-Handbuch.

Ein anderer Informant für VIC ist das Farb-RAM. Hier kann er erkennen, in welchen Farben er jedes Zeichen ausgeben muß.

Die wichtigste Informationsquelle für unseren Chip ist aber sicherlich das Zeichen-ROM. Wie dieser Name schon sagt, ist hier der Zeichensatz des Commodore sicher aufbewahrt. Denn VIC, unser Kommunikationskünstler, hat ein kleines Problem. Er selbst ist Analphabet. Das Zeichensatz-ROM merkt sich wie ein guter Schüler auf ewig das Aussehen aller Zeichen. Dafür vergißt VIC mit schöner Regelmäßigkeit – wie ein schlechter Schüler – jedes Zeichen sofort wieder. Und deshalb braucht er – wie jeder normale Schüler – einen Spickzettel. Jedesmal, bevor er ein Zeichen ausgibt, schaut er deshalb im ROM nach.

Leider hat der ROM-Baustein für unsere Pläne, die Zeichensätze zu verändern, zwei Nachteile. Zum einen ist er so einfach zu erreichen wie der einzige Hundertmarkschein in einem Sparschwein. Er ist völlig zugebaut. Auf der gleichen Adresse liegen noch die Input/Output-Register und ein RAM-Baustein herum. Diese Speichersymbiosen sind uns ja mittlerweile bekannt.

Das andere Problem ist, daß ROM-Bausteine sowieso nicht beschrieben werden können. Deshalb müssen wir ihn also irgendwie ins RAM bekommen. Aber das macht uns seine ungünstige Lage im Speicher nicht ganz einfach.

Auf jeden Fall können wir von der Vergeßlichkeit VICs profitieren. Da er vor jeder Zeichenausgabe erst nachschaut, liegt die Vermutung nahe, daß es irgendeine Adresse geben muß, die ihm sagt, wo.

Und noch ein Vorteil, den VIC für uns bietet: Weil er sich gar nicht darum

kümmert, wie das Zeichen, das er ausgibt, eigentlich ausschaut, würde er es noch nicht einmal merken, ob er seine Zeichen aus dem ROM oder von sonstwoher liest. Ihm würde nicht auffallen, daß irgend etwas nicht stimmt, sondern er würde brav alles auf den Bildschirm bringen. Der Grund dafür, daß er alle Zeichen anerkennt, liegt darin, wie er seine Zeichen aus dem richtigen Speicher liest. Er sucht sich die Zeichen nämlich nicht nach festen Adressen zusammen.

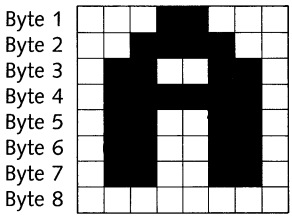
Wird zum Beispiel der Buchstabe B gebraucht, so findet VIC im Bildschirm-RAM den Wert 2. Dann beginnt er einfach von vorne, seine Speichereinheiten (jeweils 8 Bytes) abzuzählen. Er liest einfach das entsprechende Bitmuster aus dem angegebenen Speicher. In unserem Beispiel würde er bis zur Speichereinheit 2 zählen (0 = @; 1 = A; 2 = B) und dieses Muster auf den Schirm bringen.

Weil wir aber genau die Adresse, ab der VIC mit dieser Zählerei beginnt, ändern können, bringen wir ihn relativ einfach dazu, fast jeden Bereich zu lesen. Zum Beispiel aus dem RAM, wo wir ja bequem Zeichensätze hinlegen könnten.

Bevor wir damit anfangen, sollten wir erst noch ein bißchen in die Lehre gehen. Am besten bei dem ROM-Chip, den VIC normalerweise benutzt. Dort können wir am besten sehen, wie Buchstaben eigentlich abgespeichert sind und wie sie entstehen.

Natürlich müssen beim Commodore auch die Zeichen mit Strom an und Strom aus dargestellt werden.

Das Ganze geschieht in einer 8 * 8-Matrix. So bilden viele kleine Punkte ein Ganzes, vergleichbar einem Foto in der Zeitung.



Byte 1	=	0 0 0 **1** **1** 0 0 0	= 24
Byte 2	=	0 0 **1** **1** **1** **1** 0 0	= 60
Byte 3	=	0 **1** **1** 0 0 **1** **1** 0	= 102
Byte 4	=	0 **1** **1** **1** **1** **1** **1** 0	= 126
Byte 5	=	0 **1** **1** 0 0 **1** **1** 0	= 102
Byte 6	=	0 **1** **1** 0 0 **1** **1** 0	= 102
Byte 7	=	0 **1** **1** 0 0 **1** **1** 0	= 102
Byte 8	=	0 0 0 0 0 0 0 0	= 0

So sehen Sie das A auf dem Schirm

So steht's im Speicher

Das sind die Dezimalwerte der Bytes

Bild 4.1 Buchstabenmatrix A

Allein der Ausdruck 8 * 8 müßte Sie aber aufhorchen lassen, nachdem wir uns im letzten Kapitel ziemlich ausgiebig mit dieser Zahl beschäftigt haben.

Die Zeichen werden also in acht Zeilen zu jeweils acht Punkten abgelegt. Das bedeutet, daß man für jedes Zeichen achtmal acht Bits bzw. acht Bytes benötigt. Diese acht Bytes stellen für VIC eine Speichereinheit dar. Das war schon die ganze Kunst. Das kleine Schaubild soll das verdeutlichen.

Eine Umleitungsempfehlung für den VIC

Wie wir vermutet haben, gibt es tatsächlich einen Zeiger, der dem VIC klarmacht, wo er anfangen soll, die einzelnen Speichereinheiten abzuzählen und das richtige Bitmuster hervorzuholen. Normalerweise zeigt er auf das Zeichen-ROM. Normalerweise schon – aber wenn wir erst einmal da waren ... Wir können diesen Zeiger hinsetzen, wohin wir wollen. Und damit lassen sich einige äußerst kuriose Effekte erzielen. Die Adresse , die geändert werden muß, ist 53272. Je nach Wert versucht VIC jetzt, seine Zeichen zu lesen. Nur zur Demonstration ein kleiner Exkurs. Im Rahmen der Speicheraufteilung haben wir erklärt, daß in der Zeropage ständig Betrieb herrscht. Daten werden aktualisiert, Werte werden zwischengespeichert und so weiter. Das heißt, viele Bitmuster in den Speicherzellen 0 bis 255, eben der Zeropage, ändern sich ständig.

Probieren geht über Studieren.

```
POKE 53272, 16
```

Lustig, was? Je mehr «Zeichen» Sie auf dem Bildschirm haben, um so mehr ist los. Drücken Sie mal die Taste ⟨T⟩. Der kleine Block auf dem Bildschirm, der statt des Zeichens «T» erschienen ist, hat wohl mehr Ähnlichkeit mit einer lebenden Flohsammlung. Zur Erklärung: Wir sehen die Speicherzellen 200 bis 207. Unter anderem liegt hier der Zähler für den blinkenden Cursor. Wenn man bedenkt, daß der Cursor in zwei Sekunden sechsmal blinkt, wird klar, warum die da soviel zu tun haben.

Übrigens, es funktioniert noch alles wie gehabt – die Cursorsteuerung und Befehle. Denn wir dürfen nicht vergessen, daß die Zeichenwerte – sowohl die ASCII- als auch die Bildschirmcodes – noch immer dieselben sind. Nur die Darstellung läßt zu wünschen übrig.

Probieren Sie's ruhig aus.

Obwohl Sie beim Eintippen nichts erkennen können, sehen Sie, daß der Commodore schreibt.

```
POKE 53272, 21
```

Wenn Sie sich nicht vertippt haben, dann sind nach ⟨RETURN⟩ all die dubiosen Zeichen wieder ganz normal. Sie können das natürlich auch mit ⟨RUN/STOP⟩⟨RESTORE⟩ tun.

Wenn wir jetzt einen Zeichensatz irgendwo ablegen wollen, wäre die Zeropage aus naheliegenden Gründen nicht zu empfehlen. Natürlich bietet sich statt dessen freier RAM-Bereich an, allerdings nur bedingt. Das hat zwei Gründe. VIC kann maximal 16K verwalten. Das tut er üblicherweise von 0 bis 16384. In diesem Bereich müssen also alle Informationen erreichbar sein, die er zur Bildschirmdarstellung braucht – also die Zeichenfarben, das Aussehen der Zeichen und natürlich das Bildschirm-RAM.

Hier noch eine kleine Information am Rande: Der kleine VIC wird mächtig aufs Kreuz gelegt. Das Zeichen-ROM liegt tatsächlich ab 53248, das Farb-RAM ab 55296. Und das geht ja wohl sehr deutlich über die 16K, die der Arme tatsächlich kennt. Deshalb spricht er in Wirklichkeit immer nur mit einem Botschafter, der VICs Adressierung abfängt und so umwandelt, daß VIC die tatsächlich nötigen Bausteine erwischt. Warum das so kompliziert sein muß? Das läßt sich wohl nur mit der etwas eigenwilligen Systemarchitektur des Commodore 64 erklären ...

Doch zurück zum Thema: Weil VIC nur die 16K verwalten kann, ist es nicht möglich, den Zeichensatz einfach ans BASIC-Ende zu hängen. Denn da kann

0	Die Zeropage lassen Sie besser in Ruhe.
2048	Sofern Sie kein BASIC-Programm hier haben, okay. Sie könnten andernfalls den BASIC-Anfang auch nach oben setzen.
4096	Nicht verfügbar.
6144	VIC vermutet hier ROM (vgl. Text oben).
8192 10240 12288 14336	Wenn Sie hier hinter dem BASIC-Programm den Zeichensatz ablegen, dann darf das Programm nicht zu lang sein.

Tabelle 4.1 Mögliche Startadressen und was Sie davon halten sollten

VIC nicht suchen. Der andere Grund sind die BASIC-Programme selbst. Denn wenn Sie ein solches im Speicher haben und dann den Zeichensatz an Adresse 2048 ins RAM kopieren, würde das nicht gerade zur Verbesserung Ihres Programms beitragen. Bevor wir sagen, wo überhaupt und unter welchen Umständen am besten ein Zeichensatz im RAM sein könnte, in Tabelle 4.1 eine kleine Übersicht zu den möglichen Startadressen.
Wir schlagen folgende Lösung vor, weil sie den Programmierer nicht einschränkt wie die Lösungen, die den Zeichensatz hinter das BASIC setzen. Als erstes sollten Sie den BASIC-Anfang um 2K nach oben verschieben. Das haben wir ja bereits im Speicherkapitel gemacht. Damit Sie nicht zurückblättern müssen, hier noch einmal:

```
POKE44,16:POKE4096,0:NEW
```

Damit legen Sie den nötigen Speicher an und machen ihn sauber. Dann können Sie den Zeichensatz in diesen Bereich kopieren und «dahinter» ein ganz normales BASIC-Programm schreiben.

Wir werden jetzt bei den restlichen Erklärungen davon ausgehen, daß Sie diese Speicheraufteilung benutzen. Sollte es aber nötig sein, anders aufzuteilen, sehen Sie hier in Tabelle 4.2, wie Sie den entsprechenden Zeiger für Sonderzeichen auf andere Bereiche legen können.

Startadresse des Zeichensatzes	Zu POKEnder Wert in 53 272:
0	16
2048	18
4096	20 (nicht verfügbar – ROM)
6144	22 (nicht verfügbar – ROM)
8192	24
10 240	26
12 288	28
14 336	30

Tabelle 4.2 Startadressen des Zeichensatzes

Mit POKE 53 272, 18 sagen Sie VIC, daß er ab 2048 seinen Zeichensatz lesen soll. Wenn Sie das getan haben, werden Sie sehr schnell merken, daß Ihnen ab jetzt die Verständigung mit dem Computer etwas schwerfällt. Sie können nämlich plötzlich nichts mehr lesen. Was ja logisch ist, weil die «Zeichen», die VIC verwendet, zufällige Bitmuster sind. Ohne Hilfsmittel wäre es jetzt nicht gut möglich, neue Zeichen zu kreieren.

Warum aber nicht ausnutzen, was andere schon vorher erarbeitet haben?

Wie man Zeichen ein bitchen verändern kann

Nachdem Sie für Ihren Computer bezahlt haben, dürfen Sie auch kopieren – zumindest den Zeichensatz von ROM ins RAM. Allerdings ist das nicht ganz einfach. Denn wie bereits erwähnt, erweist es sich als etwas problematisch, gerade dieses ROM zu erreichen. Wenn Sie sich die Speicherskizze noch einmal ansehen, werden Sie bemerken, daß auf diesem Adreßbereich drei (!) verschiedene Chips liegen. Die I/O-Register, die ausgerechnet hier rumliegen müssen, machen natürlich Schwierigkeiten, und zwar deshalb, weil diese Register eine besondere Aufgabe haben. Sie werden für einen computerinternen Vorgang gebraucht, den sogenannten Interrupt. Dieser wird unabhängig vom laufenden Programm automatisch alle 1/60 Sekunde ausgeführt, um bestimmte Dinge zu erledigen. Alles weitere darüber finden Sie im Stichwortverzeichnis.

Damit wir aber nicht diese I/O-Register kopieren, weil sie ja dieselbe Adresse haben, sondern das Zeichen-ROM, müssen wir sie ausblenden. Wird aber dann ein Interrupt ausgeführt – und der kommt so sicher wie das Amen in der Kirche, nur schneller –, würde der Computer abstürzen, weil er sich statt auf die I/O-Register auf falsche Daten bezieht. Nun können diese Interrupts abgeschaltet werden, ohne daß der Computer dadurch in Schwierigkeiten kommt. Allerdings wird damit auch die Tastatur nicht mehr gelesen. Das heißt, der Computer nimmt keine Zeichen mehr an. Deshalb geht das Kopieren nur im Programmodus. Schreiben Sie also unbedingt vor jeden jetzt folgenden Befehl eine Zeilennummer. Programme, die den Interrupt abschalten, nutzen eigentlich bereits sehr fortgeschrittene Techniken und gehen dem Computer dabei doch sehr in die Eingeweide. Deshalb diese Vorsichtsmaßnahmen. Erster Programmschritt wäre es, die Interrupts zu unterbinden.

```
10 POKE 56334, 0
```

Danach können wir jetzt beruhigt das Zeichen-ROM ein- und die I/O-Register ausblenden. Dazu brauchen wir folgende Zeile:

```
20 POKE 1, 51
```

Jetzt wollen wir den Zeichensatz ins RAM kopieren. Dazu lesen wir die Werte mit PEEK aus dem Speicher und POKEn in die neue Adresse.

```
30 FORX=0TO2048:POKE2048+X,PEEK(53248+X):NEXTX
```

Das ist die eigentliche Kopierroutine. Die Variable X wird mit jedem Durchlauf auf einen Wert von 0 bis 2048 erhöht und deckt damit genau 2K Speicher ab. Dann wird in die gewünschte Adresse unseres RAMs der Inhalt der entsprechenden ROM-Zelle geschrieben. Um die Zeile besser zu verstehen, sollten Sie für zwei oder drei Werte das X selbst einsetzen. Damit wäre die Kopiererei erledigt.

Jetzt sollten wir freundlicherweise dem Computer seine I/O-Register zurückgeben.

```
40 POKE 1, 55
```

Weil er jetzt wieder in der glücklichen Lage ist, Interrupts durchführen zu können, kriegt er sie auch wieder.

```
50 POKE56334,1
```

Setzen Sie jetzt noch einmal den Zeichensatzzeiger auf 18 (also Zeichen lesen ab 2048).

```
POKE 53272, 18
```

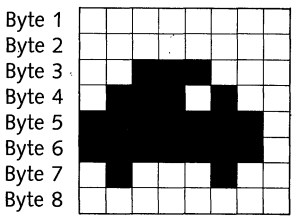
Jetzt tippen Sie möglichst viele Zeichen auf Ihren Bildschirm, und drücken Sie danach ⟨RETURN⟩. Dann können Sie unser Programm mit RUN starten.

Ganz eindrucksvoll, nicht wahr?

Dabei können Sie, wenn Sie genau hinsehen, auch erkennen, wie sich die Buchstaben von oben nach unten byte- bzw. zeilenweise aufbauen. Jetzt geben Sie gleich mal wieder direkt ein

```
POKE 2056,153
```

Na, fällt Ihnen nichts auf? Schauen Sie doch mal genau hin. Noch genauer. Ja, richtig: Aus allen A sind Ä geworden. Sie haben das Zeichen A geändert. Warum ausgerechnet der Wert 153 gePOKEd wurde? Vergleichen Sie dazu noch einmal die Buchstabenmatrix im ersten Abschnitt. Dort hatte die erste Zeile, also das erste Byte, bisher den Wert 24. Wir haben aber den Links- und



Byte 1	=	0 0 0 0 0 0 0 0	= 0
Byte 2	=	0 0 0 0 0 0 0 0	= 0
Byte 3	=	0 0 **1** **1** **1** 0 0 0	= 56
Byte 4	=	0 **1** **1** **1** 0 **1** 0 0	= 116
Byte 5	=	**1** **1** **1** **1** **1** **1** **1** 0	= 254
Byte 6	=	**1** **1** **1** **1** **1** **1** **1** 0	= 254
Byte 7	=	0 **1** 0 0 0 **1** 0 0	= 68
Byte 8	=	0 0 0 0 0 0 0 0	= 0

Bild 4.2 Zeichenmatrix «Kleines Auto»

den Rechtsaußen unserer Bit-Mannschaft angeschaltet. Damit ergibt sich nach dem berühmten Umrechnungsprinzip vom Dual- ins Dezimalsystem der Wert 153. Wenn Sie jetzt eigene Zeichen entwickeln wollen, gehen Sie am besten so vor.

Nehmen Sie ein kariertes Blatt, und entwerfen Sie das Muster auf einer «8 * 8-Matrix». Als nächstes stellen Sie dieses Muster in Binärzahlen dar. Jeder angeschaltete Punkt ist 1, jeder ausgeschaltete 0. Jetzt müssen Sie diese Binärzahl umrechnen (dazu können Sie ja das Umrechnungsprogramm benutzen) und die gewonnenen Dezimalzahlen an die entsprechende Zeichenadresse, die Sie ändern wollen, POKEn.

Die Zeichenadresse ist die Adresse im RAM, ab der die entsprechende Speichereinheit steht. Um ein bestimmtes Zeichen zu ändern, müssen Sie seinen Bildschirmcode kennen. Sie finden diese Codes (nicht die ASCII-Codes!) auf Seite 133 im Commodore-Handbuch.

Es gilt die folgende Formel:

Zeichenadresse = Startadresse des Zeichensatzes + Bildschirmcode * 8

Wenn wir unser kleines Auto also zum Beispiel anstelle des Zeichens C einsetzen wollen, wäre das Programm

```
10 FORX=0TO7:READA:POKE2048+3*8+X, A:NEXT
20 DATA 0,0,56,116,254,254,68,0
```

Wir lassen also den Computer bei 0 anfangen zu zählen, dann den ersten DATA-Wert lesen und POKEn diesen in die Zeichenadresse. A ist beim ersten Durchlauf also 0, beim zweiten auch 0, beim dritten 56 usw. bis zum achten Wert. Ganz zum Schluß haben wir dann ein neues Bitmuster in unserer 8-Byte-Speichereinheit.

Somit haben wir also unser erstes eigenes Sonderzeichen definiert. Ganz nebenbei: Haben Sie auch wirklich mitgetippt?

Gut. Wenn Sie jetzt das Prinzip verstanden haben, können Sie eigentlich loslegen.

Auf jeden Fall können Sie eine ganze Autobahn damit darstellen, auf der sich die Autos sogar bewegen – womit wieder bewiesen wäre, daß Computergrafik Illusion ist, denn von unseren Autobahnen kann man das ja nicht immer sagen ...

Zwischenspiel 2

Bevor wir an unser erstes gemeinsames Spiel gehen, erst noch ein paar Worte der Aufklärung: Unser Ziel bei allen Listings war, sie so kurz wie möglich zu halten. Die Gründe dafür haben wir ja schon anfangs erwähnt. Das gilt natürlich auch für die Spiele. So ergaben sich konsequenterweise keine neuen Softwarehits. Das war auch gar nicht unsere Absicht. Wir wollten Ihnen vielmehr die Möglichkeit bieten, ein Spiel Stück für Stück zu programmieren und damit Erfahrungen für Ihre eigenen zukünftigen Spiele zu sammeln. Wir hoffen, daß Ihnen das Ergebnis zum Schluß aber doch ein bißchen Freude macht.

Das andere, was wir gern noch losgeworden wären, ist eine Erklärung zu den Spielideen. Hier ging es uns darum, Ihnen zu zeigen, wie wichtig die Geschichte um ein Spiel herum ist, und Ihre Fantasie für eigene Verbesserungen an den besprochenen Spielen anzuregen. Außerdem konnten Sie so auch sehen, aus welchen unmöglichen Geschichten man Spielideen entwickeln kann. Und natürlich hoffen wir auch, daß diese Hintergrundgeschichten Ihnen ein bißchen Spaß beim Lesen bereiten. Zu beiden Texten sei hier noch erwähnt, daß sämtliche Ähnlichkeiten mit tatsächlich lebenden Personen äußerst zufällig sind.

5
Ein Spiel mit Sonderzeichengrafik

Rapunzel

Die Geschichte des Märchenreiches muß völlig neu geschrieben werden. Unser Reporter Gerd Heinzelmann hat die geheimen Tagebücher des Froschkönigs entdeckt. Nachdem sie gesäubert waren, stellte er fest, daß er einen Fund von unschätzbarem Wert gemacht hatte: Was niemand für möglich gehalten hätte, aufgrund der Freundschaft mit einem Tintenfisch hat der Froschkönig doch angefangen, Tagebuch zu führen. Lange Zeit waren die geheimen Tagebücher in einem Brunnen in der Nähe eines Königsschlosses vor den Augen der engagierten Märchenerzähler verborgen. Heinzelmann, der schon lange Zeit enge Kontakte zur Märchenwelt hatte («Schon als ich drei war, hat mir Großmutter vor dem Einschlafen immer vorgelesen ...»), fand die Tagebücher zufällig hinter einem Berg von goldenen Bällen und drei Lippenstiften. Und damit mußten viele Teile der Märchengeschichte, die als historisch gesichert galten, neu überdacht werden. Heinzelmann betrachtete das als seine Lebensaufgabe und veröffentlichte das Ergebnis seiner Arbeiten in einem bekannten deutschen Magazin. Mit freundlicher Genehmigung des Verlages benutzten wir diese Serie (die leider sehr früh wieder eingestellt werden mußte, weil man Heinzelmanns geheime Tagebücher entdeckt hatte) als Grundlage für unsere Spiele.

Eine der ersten veröffentlichten Arbeiten war die Richtigstellung des Märchenmotivs Rapunzel.

Im Gegensatz zur Lehrmeinung, spielte sich die Geschichte tatsächlich so ab: Königssöhne, die zwar im allgemeinen reich, aber dafür meistens dumm waren, sind aufgrund ihrer Eitelkeit oft der Grund für falsche Überlieferungen. Der Prinz bei Rapunzel, der sie regelmäßig zu besuchen pflegte, traf

nämlich eines Tages auf seinem Weg zum Turm eine böse Fee. Nun haben böse Feen die Angewohnheit, immer und überall zu beweisen, daß sie auch wirklich böse sind. Meistens tun sie das in Form von Verwandlungen. Nun traf unser Prinz eine Fee, die normalerweise im Märchenarbeitsamt beschäftigt ist und deshalb die Situation auf dem Arbeitsmarkt für Verwandelte sehr gut kannte. Um dem Prinzen eine langwierige Umschulung zu ersparen (Verwandlungen in Frösche, Esel oder Raben sind total überlaufen und werden nur noch im Losungsverfahren vergeben), beschloß die Fee, ihn in einen Floh zu verzaubern, und zog ihrer Wege.

Mühsam den Kolonnen arbeitsloser Frösche, Esel und Raben ausweichend, aber dennoch undankbar mit seinem Schicksal hadernd, machte sich unser Flohprinz auf den Weg in Richtung Turm.

Kaum dort angekommen, sah er sich einem mehrere Meter langen goldblonden Problem gegenübergestellt. Rapunzel hatte in Erwartung des Prinzen bereits ihr gülden Haar aus dem Fenster geworfen und haarte (pardon: harrte) der Dinge, die da kommen mochten. Glücklicherweise ist ein Floh ja auch Fachmann in bezug auf Haare und sehr begabt im Klettern. Wenngleich er noch nicht wußte, wie er Rapunzel seine Situation erklären sollte, machte er sich an den Aufstieg. Doch was er vorher als Prinz nie gemerkt hatte, konnte jetzt sein Verhängnis werden. Durch die ständigen Strapazen, die Rapunzels Haar im Lauf der Zeit mitgemacht hatte, war Haarspliß entstanden. Und so bestand nun die Aufgabe unseres unglücklichen Flohs darin, sich durch dieses Gewirr nach oben hin durchzuarbeiten. Das hatte zwei Konsequenzen: Zum einen, daß unser Floh immer mehr und mehr über Haarkuren nachzudenken begann, zum anderen, daß wir daraus ein Spiel gemacht haben. Ihre Aufgabe ist es nun, durch ein Labyrinth bis zu Rapunzel zu kommen und vor allem den Spielaufbau zu verstehen. Dazu wollen wir jetzt die einzelnen Programmschritte gemeinsam durchgehen.

Zeile 10: Bildschirmrahmen und Hintergrund werden schwarz. In die beiden Adressen 53 280 und 53 281 wird 0 gePOKEd.

Zeile 20: Da wir selbstdefinierte Sonderzeichen verwenden wollen, müssen wir den Zeichensatz ins RAM ab Adresse 10 240 legen. POKE 53272,2 erledigt das für uns.

Zeile 30: Diese FOR... NEXT-Schleife liest die dort abgelegten Bytes ein und POKEd sie für das Zeichen 0 (Klammeraffe @) ins RAM.

Zeile 40: Was wir noch brauchen, ist ein voller Cursorblock. Also POKE wir für den Code 1 ("A") lauter «volle» Bytes 255 ins RAM.

Zeile 50: Genauso benötigen wir eine Leerstelle. Damit Sie's nachher beim Eintippen des Labyrinths etwas einfacher haben, haben wir den Punkt (Bildschirmcode 46) dazu hergenommen. Da unser Programm ja während des Spiels keine normalen Buchstaben benötigt, brauchen wir sie auch nicht aus dem ROM zu kopieren. Außerdem können wir frei nach Belieben die ehemaligen Zeichen ändern. Um dem Punkt das Aussehen einer Leerstelle (⟨SPACE⟩) zu geben, müssen wir in seine Adressen im neuen Zeichensatz lauter 0 POKEn. Die Bytes, die dabei geändert werden sollen, sind (vom Anfang des Zeichenspeichers aus gesehen) die Bytes 46 * 8 (= 368) bis 46 * 8 + 7 (= 375). Die Schleife in dieser Zeile erledigt das für uns.

Zeile 60: In dieser DATA-Zeile ist unser kleiner Floh als Sonderzeichen abgelegt. Und so sieht er dann aus.

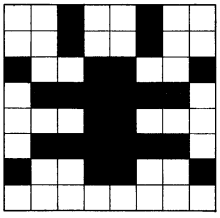


Bild 5.1 Zeichenmatrix «Floh»

Zeile 70: Mit GOSUB 350 wird das Unterprogramm aufgerufen, das den Bildschirmaufbau ausführt.

Zeile 80: Die nächsten Zeilen wählen eine zufällige Startposition für unseren Floh aus. Dafür kommen fünf Punkte in Frage. Also wird eine Zufallszahl zwischen 1 und 5 ermittelt: I = INT (5 * RND(1)) +1

Zeilen 90 bis 130: Je nachdem, welchen Wert I jetzt zufällig erhalten hat, wird die X-Position unseres Flohs festgelegt. Die X- und Y-Positionen entsprechen hier ganzen Zeichen. So kann X also von 0 bis 39 und Y von 0 bis 24 gehen. Die Startpositionen stimmen genau mit den untersten Punkten des Labyrinths überein.

Zeile 140: Die Zeitvariable TI$ wird auf ihren Grundwert zurückgesetzt. So kann die Zeit gemessen werden, die ein Spieler braucht, um durch das Labyrinth zu finden.

Zeile 150: Der Y-Wert unseres Flohs wird auf den tiefsten Punkt gesetzt:

24. Damit wären die Startkoordinaten also klar. Unser Floh sitzt irgendwo auf einem der fünf untersten Punkte im Labyrinth. Die Variablen X1 und Y1 werden jetzt erst einmal mit den «eigentlichen» Koordinaten gleichgesetzt.

Zeile 160: Mit diesem POKE, der zur Startadresse des Bildschirm-RAMs die Y-Koordinate * 40 (40 einzelne Zeichen hat ja eine Zeile) und die X-Koordinate addiert, wird der Floh jetzt endgültig auf den Schirm gebracht. Er ist hellblau (Farbcode 3) – weniger, weil das die bevorzugte Farbe der bösen Fee war, sondern mehr, weil sich diese Farbe gegen das Gelb der Haare sehr gut abhebt. Also wird der Farbcode in die entsprechende Adresse des Farb-RAMs gePOKEd.

Zeile 170: Hier findet die Tastaturabfrage statt. Das Programm wartet so lange, bis eine Taste gedrückt wird.

Zeile 180: Nun wird der Reihe nach abgefragt, ob eine bestimmte Taste gedrückt wurde. In dieser Zeile wäre die ⟨CRSR UP⟩ dran. Das Steuerzeichen können Sie eingeben, indem Sie ⟨SHIFT⟩-⟨CRSR UP/DOWN⟩ drücken. Wenn diese Taste wirklich die gedrückte sein sollte, wird Y1 der Wert von Y – 1 zugewiesen. X1 bleibt X. Welchen Sinn haben diese beiden Variablen? Nun, damit X und Y vorerst nicht verändert werden müssen (sie werden später noch gebraucht), dienen die beiden X1 und Y1 als Ersatz.

Zeile 190: Sollte ⟨CRSR DOWN⟩ gedrückt worden sein, wird die Y-Koordinate um 1 erhöht. Die X-Koordinate verändert sich nicht.

Zeile 200: Dasselbe Spielchen für ⟨CRSR LEFT⟩. Diesmal wird der X-Wert um 1 erniedrigt, und Y, also die Zeile, bleibt unverändert.

Zeile 210: Der ganze Spaß für ⟨CRSR RIGHT⟩.

Zeile 220: Der Variablen P (das deutet schon an, was mit ihr geschehen soll; der Inhalt von P wird gePEEKed werden) wird die nach dieser Eingabe errechnete Position im Bildschirm-RAM zugewiesen.

Zeile 230: Wenn dort irgendein anderes Zeichen als das "A", das wir ja zum Haar umdefiniert haben, steht, geht die Post gleich wieder zur Tastaturabfrage zurück, und nichts ändert sich. Der Floh darf das Haar ja schließlich nicht verlassen ...

Zeile 240: Nachdem hier offensichtlich Platz für unseren kleinen Floh ist, POKEn wir ihn halt mal dahin, wohin die Variable P zeigt. Auch die Farbe wird verändert. Das funktioniert mit folgendem kleinen Trick: Eine Adresse im Farb-RAM ist genau 54 272 Bytes von der dazugehörigen Adresse im Bildschirm-RAM entfernt. Wenn man also P + 54 272 ausrechnet, kommt man genau auf die gewünschte Adresse.

Zeile 250: Jetzt haben wir auch den Nutzen davon, daß wir die «alten» Koordinaten X und Y noch nicht verändert haben. Denn mit ihrer Hilfe können wir den alten Floh (gerade hatten wir ja für ganz kurze Zeit zwei davon auf dem Bildschirm. Haben Sie's gemerkt?) jetzt löschen bzw. durch ein «Stück Haar», einen Bildschirmcode 1, ersetzen. Die Farbe wird auch wieder hergerichtet, nämlich 7 (= gelb) fürs gülden Haar. Jetzt, wo es keinen «alten» Floh mehr gibt, können wir auch dessen Koordinaten X und Y löschen und durch die aktuellen X1 und Y1 ersetzen. Merken Sie schon, worauf das hinausläuft? Beim nächsten Durchlauf haben wir wieder eine neue Position, und das hier ist die alte. So wird der neue immer an seinen Platz gePOKEd und der alte gelöscht. Das Ganze ergibt dann ein bißchen den Effekt einer Animation.

Zeile 260: Hier wird nochmal schnell abgecheckt, ob der Spieler nicht schon lange gewonnen hat. Das Labyrinth ist nämlich so konstruiert, daß die Ausgänge (die übrigens zufällig genau gegenüber den entsprechenden Startpositionen liegen) allein in der ersten Bildschirmzeile liegen. Hat also ein Floh irgendwie die erste Zeile erreicht, hat er diese Runde gewonnen. Auf geht's dann zur nächsten Runde, der Floh ist ein Stück vorangekommen, aber oben ist er noch lange nicht. R steht für «Runde». Die nächste Runde ist also eingeläutet. Wenn Sie noch nicht die Nummer 3 ist, springt das Programm ins Unterprogramm und läßt sich dort einen Irrgarten zurechtzimmern. Wenn das soweit ist, springt das Programm in Zeile 150, wo die ganze Geschichte von vorn losgeht.

Zeile 270: Sollte aber die oberste Position erreicht worden sein und R = 3, die dritte Runde also erfolgreich bespielt wurde, dann hat der Floh ja wohl gewonnen. Das wird dann im einzelnen ab Zeile 290 gefeiert, resultiert hier aber erstmal in einem Bildschirmlöschen.

Zeile 280: Wenn der Floh natürlich noch nicht gewonnen hat, geht's zurück zur Tastaturabfrage in Zeile 170.

Zeile 290: Um dem Spieler mit wieder lesbaren Worten mitteilen zu können, daß er gewonnen hat, müssen wir den Zeichensatz wieder zurückschalten. POKE 53272,23 beschert uns Kleinbuchstaben.

Zeile 300: Da das Listing aber im Grafikmodus ausgedruckt wurde, was wegen der Steuerzeichen gewisse Vorteile hat, werden die Großbuchstaben in diesem Text als Grafikzeichen dargestellt. Das Steuerzeichen davor steht für Hellgrün. Sie tippen also PRINT "⟨CTRL⟩-⟨6⟩ Wieder ein glücklicher Floh!"

Zeile 310: «Sie haben in» (und jetzt werden aus dem bis jetzt immer weitergelaufenen TI$ die Minuten und Sekunden herausoperiert)" min: "(hier stehen die Sekunden)" s zu Ihrer"

Zeile 320: "Rapunzel gefunden!"

Zeile 330: Jetzt wird der Spieler aufgefordert, eine Taste zu drücken. Das tippen Sie dann mal so: PRINT"⟨ 3* CRSR DOWN⟩ ⟨RVS ON⟩ Drücken Sie eine Taste! ⟨RVS OFF⟩"

Zeile 340: Diese Zeile wartet, bis Sie eine Taste gedrückt haben. Wir werden Ihnen später im Input/Output-Kapitel erklären, was da im einzelnen vor sich geht. Wenn eine Taste gedrückt wurde, startet sich das Programm selbst mit RUN.

Zeile 350: Hier geht es jetzt also los, unser Labyrinth-Unterprogramm.

Dazu gibt es jetzt einiges zu sagen. Wir haben uns lange überlegt, was wir am geschicktesten mit dem Labyrinth anstellen. Natürlich sind Computer hervorrgend dazu geeignet, Labyrinthe zu berechnen und zu entwerfen. Aber als wir anfingen, selbst Experimente damit zu machen, und uns ansahen, wie andere Leute das gelöst hatten, wurden wir das Gefühl nicht los, daß die ganze Angelegenheit mit dem Spiel ins Uferlose wächst. Ein Labyrinth-Programm zieht sich meist über mehrere Seiten hin und ist gespickt von mathematischen Finessen. Das hätte nicht nur den Rahmen dieses Kapitels gesprengt, sondern auch in keinem Verhältnis zu dem Programm gestanden, das das Labyrinth dann benutzt. Kurz und gut, wir haben uns schließlich entschieden, ein «Fertig-Labyrinth» zu verwenden. Dabei handelt es sich um ein Labyrinth, das vollständig in PRINT-Zeilen abgelegt ist, in dem sich allerdings Löcher befinden, und zwar so, daß, wenn nur ein Loch geflickt wird, auch nur ein Weg frei wird.

Ein solches Labyrinth befindet sich in den Zeilen 350 bis 590. Wenn Sie aber schon mal ein solches «Fertig-Labyrinth» als Listing gesehen haben, waren Sie vielleicht von der Masse an Steuerzeichen schockiert, die da verwendet wurden. Revers an, ein Space, Revers aus, zwei rechts, ein Space usw.

Um das zu vermeiden, haben wir uns gedacht, wenn wir für den Floh eh' schon ein Sonderzeichen verwenden, können wir zwei bekannte Zeichen, wie "A" und "." auch noch umdefinieren.

So haben Sie jetzt den Vorteil, relativ leicht (so hoffen wir doch) unser Labyrinth in Ihren Commodore eingeben zu können. Die Steuerzeichen bedeuten übrigens: Invertiertes Herzchen für ⟨CLR/HOME⟩; invertiertes Pi

für Gelb (⟨CTRL⟩-⟨8⟩) und am Schluß der ganzen Reihe ⟨HOME⟩ für das invertierte S. Übersehen Sie auch nicht den Strichpunkt am Schluß der letzten Zeile. Der ist da, um das Scrolling zu verhindern, das stattfinden würde, wenn der Cursor beim PRINTen in die letzte Zeile rutscht.

Zeile 600: Hier ist jetzt der Teil, der ein Loch im Labyrinth auf Zufallsbasis flickt. In Zeile 600 wird eine Zufallszahl zwischen 1 und 5 ermittelt (vergleichen Sie bitte Zeile 80).

Zeilen 610 bis 650: In diesen Zeilen wird abgefragt, welche Zufallszahl jetzt herausgekommen ist, und aufgrund dieser dann eine der Positionen der Löcher ausgewählt.

Zeile 660: Hier wird das Loch mit einem Haarcode, also einem umdefinierten "A" (BS-Code = 1) zugebaut und im Farb-RAM mit der entsprechenden Farbe versehen.

Zeile 670: Dieses RETURN läßt das Programm dann wieder in den Hauptteil zurückkehren.

So, das war's. Wenn Sie jetzt mitgetippt haben oder irgendwann später unser Listing eintippen wollen, hier ein paar gute Tips dazu: Machen Sie auch schon während des Tippens Sicherungskopien. Es wäre wirklich schade, wenn ausgerechnet beim Eintippen der letzten Zeile irgendein Familienmitglied über das Stromkabel stolpert oder sonst eine Widrigkeit passiert.

Bevor Sie ein Programm wie das oben erklärte mit RUN starten, sollten Sie auf jeden Fall eine Version davon abSAVEn; auf Diskette oder Kassette, das ist dabei erst mal egal.

Wenn Sie nämlich das Programm starten, sind mit ziemlicher Sicherheit noch Fehler drin. Das ist auch ganz normal. Das Gegenteil wäre eher die Ausnahme. Aber wenn Sie sich ausgerechnet bei einem POKE vertippt haben oder dieser von einer falschen Adresse ausgeht, kann das böse Folgen haben.

Wenn Ihr Programm läuft, aber irgend etwas kommt Ihnen spanisch vor, weil's einfach komisch aussieht oder was auch immer, sollten Sie sich die Zeit nehmen, Ihr Programm noch mal mit unserem Listing zu vergleichen.

Wenn Sie das alles beherzigen, ist es wirklich nicht schwer, dieses Spiel zu spielen. Ihren Floh steuern Sie, wie gewohnt, mit den Cursortasten. Mehr gibt es eigentlich an Anleitung gar nicht zu sagen.

Also wünschen wir Ihnen viel Spaß mit unserem Spiel. Und experimentieren Sie auch mal selber damit! Denn dazu ist es ja in erster Linie da.

```
10 POKE53280,0:POKE53281,0
20 POKE53272,26
30 FORX=0TO7:READW:POKE10240+X,W:NEXT
40 FORX=8TO15:POKE10240+X,255:NEXT
50 FORX=368TO375:POKE10240+X,0:NEXT
60 DATA36,36,153,126,24,126,153,0
70 GOSUB350
80 I=INT(5*RND(1))+1
90 IFI=1THENX=3
100 IFI=2THENX=11
110 IFI=3THENX=20
120 IFI=4THENX=28
130 IFI=5THENX=34
140 TI$="000000"
150 Y=24:Y1=Y:X1=X
160 POKE1024+Y*40+X,0:POKE55296+Y*40+X,3
170 GETA$:IFA$=""THEN170
180 IFA$="□"THENY1=Y-1:X1=X
190 IFA$="■"THENY1=Y+1:X1=X
200 IFA$="■"THENX1=X-1:Y1=Y
210 IFA$="■"THENX1=X+1:Y1=Y
220 P=1024+Y1*40+X1
230 IFPEEK(P)<>1THEN170
240 POKEP,0:POKEP+54272,3
250 POKE1024+Y*40+X,1:POKE55296+Y*40+X,7:Y=Y1:X=X1
260 IFY=0THENR=R+1:IFR<3THENGOSUB350:GOTO150
270 IFY=0ANDR=3THENPRINT"□":GOTO290
280 GOTO170
290 POKE53272,23
300 PRINT"■DIEDER EIN GLUECKLICHER ←LOH!"
310 PRINT"↑IE HABEN IN "MID$(TI$,3,2)"
MIN:"RIGHT$(TI$,2)" SEC ZU ↖HRER"
320 PRINT"←APUNZEL GEFUNDEN!"
330 PRINT"■■■■←RUECKEN ↑IE EINE !ASTE!■"
340 POKE198,0:WAIT198,1:POKE198,0:RUN
350 PRINT"■■...A.......A.......A.......A.....A"
360 PRINT"...AAAAA.AAA.....AAAA.......A..AAAA..AA"
370 PRINT".......A.A.......A.........AA.A.....A"
380 PRINT".AAA..AA.AAAAAA..A.AAA.......A.AAAAAAA"
390 PRINT".A.AAAA.......A..AAA.A.AAAAA.A...A"
400 PRINT".........AA.AAAA......A.A...A.A...A"
410 PRINT".AAAAAAAA.........AAAA.AAA.AAA..AAA.AAA"
420 PRINT"...A....AAAAAAA...A......A......A...A.A"
430 PRINT"...A.........A...AAA.AA.AAAAAA.AAAAA.A"
440 PRINT"...AAAAAAAAAA.AAA......A......A.......A"
450 PRINT"...........A...A....AAAAAA.AAA...AAAAA"
```

```
460 PRINT"...AAAAAAAAAA.AAA....A...A........A"
470 PRINT"...A..........A......A...AAAAAAA..A"
480 PRINT"...A....AAAAAAAAAAAA.A.........A..AAAA"
490 PRINT"...A....A..........A.A.AAAAAA..A"
500 PRINT"...AAAA.A..........AAA.A....A..A.AAAAA"
510 PRINT"......A.A..........A...AAAA.A..A.A"
520 PRINT"......A.AAAAAAAA...A......A.AAAA.A..AAA"
530 PRINT"......A........A...AAAAAA.A......A..A.A"
540 PRINT".AAAAAA....AAA.A........A.AAAAAAAAAAA.A"
550 PRINT".A....A....A.A.A.AAAAAA.A......A......A"
560 PRINT".A....A....A.A.A.A....A.A.AAAAAA.AAAA.A"
570 PRINT".AAA........A.AAA.AAAA.AAA.A......A..AAA"
580 PRINT"...A........A.........A.....AAA....AA"
590 PRINT"...A........A........A.......A.....A";
600 I=INT(5*RND(1))+1
610 IFI=1THENL=1225
620 IFI=2THENL=1234
630 IFI=3THENL=1365
640 IFI=4THENL=1451
650 IFI=5THENL=1621
660 POKEL,1:POKEL+54272,7
670 RETURN
```

Zwischenspiel 3

Wir glauben, daß wir Ihnen noch eine Erklärung schuldig sind, auf die wir später immer wieder zurückgreifen werden. Und wenn Sie die erst einmal verstanden haben, wird Ihnen das – wie das Verständnis für Binärzahlen – den Umgang mit Bits und Bytes etwas vereinfachen. Was wir meinen, sind die Booleschen Operationen.

Aber keine Sorge, dazu ist weder Vollnarkose noch ein Skalpell nötig, sondern einfach nur ein bißchen Verständnis.

Der geistige Vater dieser Operationen ist ein gewisser George Boole, der im 19. Jahrhundert gelebt hat, also mit Computern nicht allzuviel zu schaffen gehabt haben konnte. Aber er war Mathematiker und hat ein ziemlich kompliziertes Modell entworfen, wie man mit Aussagen, Ereignissen und dgl. rechnen kann.

Vieles von dem, was er sich so ausgedacht hat, ist in erster Linie dazu geeignet, Schüler zu verwirren. Insofern unterscheidet ihn nichts von anderen Mathematikern. Doch einige Überbleibsel seiner Theorien leben noch immer in Ihrem Commodore fort, und zwar die BASIC-Befehle AND, OR und NOT.

Vielleicht sind sie Ihnen schon einmal aufgefallen, und Sie haben sie sogar schon ausprobiert. Die Ergebnisse kommen einem aber meist alles andere als logisch vor. Oder wieso bitteschön soll 1 und 2 gleich 0 sein? Sie glauben das nicht? Probieren Sie es selbst:

```
PRINT 1 AND 2
```

Naaaaa ...

Unser Computer scheint auch besondere Vorlieben für bestimmte Zahlen zu haben. Wenn Sie ihn (wie die Kinder bei Michael Schanze) vor die Entscheidung 1, 2 oder 3 stellen:

```
PRINT 1 OR 2 OR 3
```

entscheidet er sich prompt und stur für die 3. Nichts mit Plopp und Stop ...

Immerhin, wenn Sie ihm auftragen, nicht die 4 zu drucken, tut er es auch nicht:

```
PRINT NOT 4
```

Statt dessen druckt er −5. Da soll noch mal einer behaupten, Computer wären nicht folgsam.

Oder sollte hinter diesen Ereignissen nicht doch irgendein logisches System stecken?

Natürlich – die Lösung liegt, wie könnte es anders sein, in den Bits. Wir wollen Ihnen mal ein bißchen auf die Spur helfen.

Stellen Sie sich die Zahlen von oben mal als Bits vor, und schreiben Sie diese untereinander.

1 AND 2 wäre dann:

```
  01
AND 10
------
  00
```

Oder die andere Aufgabe.
1 OR 2 OR 3 würde ergeben:

```
  01
OR 10
OR 11
------
  11
```

Nachdem Sie jetzt sicher schon voller Spannung und Vorfreude sind, wollen wir das Geheimnis lüften. Mit solchen Operationen können Sie Bits nach ganz bestimmten Regeln vergleichen oder verknüpfen.

Der AND-Befehl

Wenn Sie zwei (Binär-)Zahlen mit AND verbinden, werden nur die Bits ins Ergebnis übernommen, die in beiden Zahlen «an» (= 1) sind.

BIT NR:	7	6	5	4	3	2	1	0	
	1	1	0	0	1	0	1	0	(= 202)
AND	1	0	1	1	1	0	0	0	(= 184)
ergeben	1	0	0	0	1	0	0	0	(= 136)

Zur Erklärung:
Wenn Sie

```
PRINT 202 AND 184
```

eingeben, erhalten Sie als Ergebnis 136. Denn es werden dann nur die gleichwertigen Bits (also Bit Nummer 3 der ersten Zahl und Bit Nummer 3 der zweiten Zahl oder Bit Nummer 7 der ersten und Bit Nummer 7 der zweiten Zahl) verglichen. Nur wenn beide Bits gesetzt oder «an» sind (also das der ersten Zahl UND (= AND) das der zweiten Zahl), wird das Bit auch im Ergebnis gesetzt, zum Beispiel bei den Bits Nr. 7.

Bei den Bits Nr. 6 ist es anders: Das erste ist zwar gesetzt, das zweite aber nicht. Deshalb ist Bit Nr. 6 im Ergebnis auch 0. Und so geht das weiter, bis alle Bits verglichen sind.

Wenn Sie also zwei Werte mit AND verbinden, ergibt sich folgendes Bild:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Der OR-Befehl

Bei OR geht das Ganze etwas anders. Hier genügt es, wenn eines der beiden Bits gesetzt ist.

BIT NR:	7	6	5	4	3	2	1	0	
	1	1	0	0	1	0	1	0	(= 202)
OR	1	0	1	1	1	0	0	0	(= 184)
ergibt	1	1	1	1	1	0	1	0	(= 250)

Wieder Beispiele zur Erklärung. In Bit Nr. 7 sind beide Bits gesetzt. Weil schon eines reichen würde, ist das entsprechende Ergebnisbit natürlich auch an. Bei Bit Nummer 6 ist das erste an, das zweite aus. Aber das Ergebnisbit bei OR ist natürlich 1. Denn wir sagten ja, es reicht, wenn eines der beiden «an» ist (also das erste ODER [= OR] das zweite Bit gesetzt ist). Entsprechend läuft das alles bei Bit Nr. 5 – nur, daß diesmal erst das zweite Bit gesetzt ist. Aber das macht ja nichts.

Zusammengefaßt ergibt das OR folgende Möglichkeiten:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

So einfach ist das.

Der NOT-Befehl

Der NOT-Befehl ist nur sehr schwer mit Bitoperationen zu erklären und für uns eigentlich auch uninteressant. Beim Programmieren braucht man ihn nicht sehr oft.

Aber wahrscheinlich hätten Sie jetzt gerne gewußt, was Sie davon haben, daß Sie sich das alles angehört haben.

Und genau der Beantwortung dieser Frage soll dieses Zwischenspiel dienen.

Boolesche Operationen sind immer dann notwendig und nützlich, wenn Sie einzelne Bits in einem Byte verändern wollen. Und da Sie beim Computer ja letztlich immer mit diesen kleinen Dingern zu tun haben, kann die Notwendigkeit dazu relativ häufig vorkommen. So gibt es speziell bei den Adressen von VIC einige Bits, die «allein» eine ganz besondere Aufgabe haben. Nehmen wir als Beispiel mal die uns ja gut bekannte Adresse 53 272. Bei ihr haben die verschiedenen Bits folgende Aufgaben:

53 272:

Bit	7 6 5 4	3 2 1	0
	Bildschirm RAM Adresse	Zeichen- satz	nicht belegt

Das bedeutet, Sie können durch die Bits 7 bis 4 festlegen, wo Ihr Bildschirm RAM losgeht. Den Wert dieser 4 Speicherzellen muß man dabei mit 102 multiplizieren. (Sie erinnern sich doch sicher noch: Mit 4 Bits kann man di Zahlen 0 bis 15 darstellen. 15 * 1024 = 15360. Das wäre also die höchst Startadresse für das Bildschirm-RAM. Und VIC kann ja auch bekanntlicl höchstens 16K adressieren.) Normalerweise ist also das Bit 4 an, die Bits 5 bi 7 sind aus. (Denn die normale Startadresse des Bildschirm-RAMs ist 1024 [= 1 * 1024].) Das ist auch der Grund, weshalb der niedrigste Wert, den wir be den Sonderzeichen in diese Adresse gePOKEd haben, 16 war. Also ist Bit immer an. Die Adresse des Zeichensatzes wird durch die Bits 1 bis 3 bestimmt Würden Sie POKE 53272,4 eingeben, wäre zwar das normale Zeichensatz ROM eingeschaltet, aber das Bildschirm-RAM läge jetzt in der Zeropage Probieren Sie es ruhig mal aus, der Effekt ist ganz lustig.

Üblicherweise benutzt man nach diesem Versuch ⟨RUN/STOP⟩ und ⟨RE STORE⟩.

Falls Sie übrigens ernste Absichten mit dem Bildschirm-RAM (bezüglicl Verschieben!) haben, dann finden Sie dazu Näheres im POKE-Anhang. Jetz haben wir also einen Fall, wo ein einfaches POKE 53272,18 nicht meh ausreicht, um den Zeichensatz zu verändern. Denn damit würden Sie ja aucl gleichzeitig den Bildschirm wieder auf seinen ursprünglichen Platz zurück schieben. Sie könnten jetzt natürlich die Bytes ausrechnen, die diesem neuei Zustand entsprechen. Auf die Dauer einfacher ist aber folgendes Prinzip: Wi wollen die Bits 4 bis 7 so schützen, daß sie nicht verändert werden können Die Bits 1 bis 3 sollen aber dafür einen bestimmten, neuen Wert erhalten.

Und jetzt können wir einsetzen, was wir gelernt haben. Um die oberen Bits quasi zu «konservieren», führen wir ein AND mit 1111 0000 (also 240 durch:

```
  00010100
AND 11110000
  00010000
```

Sie sehen: Nur die Bits, die innerhalb der obersten vier Stellen gesetzt sind werden in das Ergebnis übernommen. Um nun den neuen Wert für dei Zeichensatz (in unserem Fall 2) in unser Byte mit hineinzubekommen, setzei wir die entsprechenden Bits einfach dazu: "OR" mit 00000010

```
  00010000
OR 00000010
  00010010
```

Und damit hätten wir unseren Wert, den wir in die Adresse 53 272 POKEn können.

In BASIC sieht das dann folgendermaßen aus:

```
POKE 53272,(PEEK(53272)AND 240)OR2
```

Den Wert, den wir am Anfang bearbeiten wollen, müssen wir mit PEEK(53 272) aus der entsprechenden Speicherzelle lesen. Dann wird er verändert und zurückgePOKEd.

Derartige Formeln werden Sie oft finden, da sie den Vorteil haben, wirklich nur die Bits zu verändern, die nötig sind. Zu guter Letzt noch zwei Tips. Wenn Sie einzelne Bits einschalten wollen, geht das folgendermaßen:

```
POKE (Adresse),PEEK(Adresse) OR 2 ^ Bitnummer
```

Wenn Ihnen nicht ganz klar ist, was da passiert, nehmen Sie sich ein Blatt und schreiben Sie die zwei Werte untereinander ...

Für das Ausschalten eines Bits verwenden Sie folgenden Befehl:

```
POKE (Adresse),PEEK(Adresse) AND 255 - 2 ^ Bitnummer
```

Auch das sieht komplizierter aus, als es eigentlich ist. Der Ausdruck (255 − 2 ^ Bitnummer) ergibt ein Byte, in dem deshalb alle Bits (= 255) gesetzt sind, außer dem einen gewünschten (2 ^ Bitnummer). AND bewirkt dann folgendes:
Das vierte Bit soll gelöscht werden

```
  11011011
AND 11101111
  11001011
```

Was hiermit auch gelungen wäre. Und das war es auch schon zum Thema Boole. Die AND- und OR-Operationen sind, wie Sie vielleicht gesehen haben, oft unumgänglich. Wann man sie braucht, werden Sie noch im POKE-Anhang sehen.

Sollte noch irgend etwas nicht klar sein, lesen Sie dieses Zwischenspiel am besten noch einmal in Ruhe durch, und probieren Sie die verschiedenen Dinge auch aus. Dann werden Sie sehr schnell ausreichend Erfahrungen mit Herrn Boole und seinen Operationen haben.

6
Die hochauflösende Grafik des Commodore 64

Ein völlig neues Grafikgefühl

Hallo. Da sind Sie ja wieder. Naja, das Buch ist klein. Da trifft man sich schon ab und an, nicht?

Wir hoffen, Sie haben sich in bezug auf Sonderzeichen jetzt etwas ausgetobt.

Ach so, haben Sie eigentlich unser kleines Spiel von vorhin auch mal selbst so ein bißchen verändert? Wenn nicht, sollten Sie das nämlich jetzt tun. Damit erweitern Sie ja auch Ihre eigenen Programmierkenntnisse. Und außerdem: Wenn man erst mal das Grundprogramm hat, können schon kleine Änderungen ganz witzige Effekte haben. In jedem Fall könnten Sie ja mal die Hintergrundfarbe oder so ändern. Wenn Sie also in dieser Richtung bisher noch nichts gemacht haben, wäre jetzt die rechte Zeit dazu.

Die anderen Herrschaften dürften wir derweil in den nächsten Saal bitten. Die ausgestellten Stücke gehören zur Abteilung «Hochauflösende Grafik oder von der Schwierigkeit, einfache Bilder ohne BASIC-Befehle zu machen».

Oh, Moment noch.

He, Sie da. Ja, Sie meine ich. Haben Sie auch wirklich vorher mit dem Spielprogramm herumexperimentiert, oder schmuggeln Sie sich hier nur einfach so dazu?

Ein paar gutgemeinte böse Worte zur Einführung

Dann können wir jetzt also anfangen. Wir wollen uns über eine Sache unterhalten, die bei Einsteigern entweder völlig unbekannt ist oder nach den ersten Versuchen sehr schnell zu einem privaten Waterloo für die meisten Neulinge wurde. (Sie wissen ja: In Waterloo war die Geschichte mit diesem Franzosen, der offensichtlich auch schon bessere Schlachten gesehen hatte.)

Für jeden Commodore-Besitzer, der keine *BASIC-Erweiterung* hat, scheint der sogenannte *Hires-Modus* erst mal in etwa so ergiebig wie ein Fingerhut voll Waschmittel für die Wäsche einer Bundeswehrtruppe nach einer Feldübung im Regen. Man hat das Gefühl, es ist entweder völlig unsinnig, überhaupt erst anzufangen, oder der Aufwand lohnt die Mühe ja eh nicht. Dieser Eindruck ist leider nicht ganz unrichtig. Es spricht schon Bände, daß der C 64 nahezu das gleiche Betriebssystem hat wie der VC 20. Nur – der VC 20 hatte wenigstens auch einen wesentlich primitiveren Videochip, der nicht solche Möglichkeiten bot ...

Aber man kann trotzdem allerhand mit dieser Grafik tun. Um Ihnen dieses Unterfangen etwas zu erleichtern, haben wir auch einige Programme zusammengestellt, die einfache Dinge wie Kreise, Dreiecke oder Vierecke ermöglichen.

Alles in allem ist dieses Manko an Ihrem Computer trotzdem erträglich, weil man das meiste ja doch relativ einfach mit den Grafik- und Sonderzeichen hinkriegt. Aber wir finden: Wenn man schon eine Hires-Möglichkeit einbaut, hätte man zumindest einen Befehl zum Ziehen von Linien dazumachen können.

Der Bit(-tere) Weg zur Grafik

Es gibt nur eine Möglichkeit beim Commodore (ohne BASIC-Erweiterung), im Hires-Modus zu arbeiten: jeden gewünschten Punkt ein- bzw. ausschalten. Einzige Art, dies zu tun, ist der POKE-Befehl. Im Fall von Hires kann dieser Befehl für zwei verschiedene Dinge zuständig sein.

Entweder verändert er RAM-Speicher oder greift direkt auf VIC zu, um bestimmte Betriebszustände an- und abzuschalten. Um eine Grafik zu erstellen, muß man sich also der Methode des *Bitmappings* bedienen.

Bitmapping bedeutet eigentlich, daß ein bestimmter Speicherbereich (8K) als Hires-Seite von VIC gelesen wird. VIC geht diesen Speicherbereich Byte

für Byte durch. Jedes gesetzte Bit in diesem Speicher entspricht dabei einem Punkt auf dem Bildschirm. Dabei erreicht Ihr Commodore eine Auflösung von 320 Punkten horizontal und 200 Punkten vertikal.

Um das alles auf einen einfachen Nenner zu bringen: Beim Bitmapping sehen Sie immer 8K-Speicher auf dem Bildschirm. Ähnlich wie bei den selbstdefinierten Zeichen müssen die Werte, die dem gewünschten Bild entsprechen, in diesen Speicher gePOKEd werden. Bevor wir uns aber noch mehr in Theorie verstricken, schalten wir den Hires-Modus erst mal ein.

```
POKE 53265,59
```

Wenn Sie diesen Befehl direkt eingegeben haben, wird Ihr Bildschirm jetzt – na, sagen wir mal, etwas chaotisch aussehen.

Aber keine Sorge. Dieses Chaos ist logisch. Und zwar aus zwei Gründen. Zum ersten ist alles, was der Computer tut, logisch. Auch wenn es zugegebenermaßen nicht immer so aussieht. Der zweite Grund, das ist der ernstzunehmendere, ist folgender. In dem freien RAM-Bereich, den VIC ab jetzt als Parkplatz für Hires-Daten betrachtet, sind ja momentan lauter zufällige Werte. Daher sehen Sie auch lauter zufällige Bitmuster auf dem Schirm. Logisch, oder?

Weil wir uns vorstellen könnten, daß Ihnen dieses Bild auf die Dauer etwas langweilig wird, sagen wir Ihnen besser gleich, wie man aus der ganzen Story wieder rauskommt. (Bevor Sie auf das bewußte Knöpfchen an Ihrem Fernseher drücken, das es möglich macht, diese freundliche Familie da unten in Texas zu beobachten. Sie wissen schon, die mit dem Öl und dem blonden Gift.)

Zum Abschalten (des Hires-Modus, nicht des Fernsehers) können Sie wahlweise ⟨RUN/STOP⟩ + ⟨RESTORE⟩ drücken oder

```
POKE 53265,27
```

eingeben. Nachdem das geklärt wäre, können wir erst mal weitermachen. Wir haben ja schon einmal erwähnt, daß der gute VIC gleichzeitig nur maximal 16K-Speicher adressieren kann. Das heißt, seine Möglichkeiten bei Hofe (Sie erinnern sich doch noch an unser Bild vom Königreich?) sind begrenzt. Damit er überhaupt was auf die Matte (pardon: Mattscheibe) bringen kann, müssen seine wichtigsten Informanten innerhalb seines Einflußbereiches sein. Diese Informanten wären das Bildschirm-RAM (welche Zeichen sind gerade wo?), das Farb-RAM (was haben die da draußen eigentlich für eine Farbe ...) und der Zeichensatz (... und wie schauen sie überhaupt aus?)

Außerdem sind da noch eventuell vorhandene Sprite-Daten. Aber zu denen kommen wir erst im nächsten Kapitel. Unter den geschilderten Bedingungen gibt es eigentlich gar nicht mehr so viele Möglichkeiten, wo so ein Hires-Speicher liegen könnte.

Wir haben ja sicherlich auch schon mal erzählt, daß unser kleiner VIC von Commodore so richtig klassisch reingelegt wird. Das Zeichensatz-ROM hat die Startadresse 53 248. Weil diese Zahl aber zu hoch für VICs Adressierkapazität ist, hat man ihm bei Commodore einen anderen Chip vor die Nase (bzw. vor den Schaltkreis) gesetzt. Und dieser Chip manipuliert den Adreßbus von VIC so, daß VIC zwar die für ihn darstellbare Zahl 4096 aufruft, aber dabei in Wirklichkeit die Speicherzelle 53 248 anspricht.

Deshalb sind für VIC also offiziell die Adressen 4096 bis 8191 belegt. (Vgl. Sie dazu auch die Tabelle 4.2.)

Alles in allem bleibt zu guter Letzt nur noch eine Möglichkeit, diesen Hires-Speicher anzulegen: ab der freien RAM-Adresse 8192.

Noch eine Bemerkung hierzu. Natürlich gibt es doch noch eine andere Möglichkeit. Und zwar, indem man den *Adreßbereich* von VIC verschiebt. Das heißt im Grunde, VIC so zu manipulieren, daß er den Hires-Speicher nicht mehr von 0 bis 16 384 (also die ersten 8K), sondern von 16 384 bis 32 767 (also die zweiten 8K) liest. Über die finsteren Intrigen, die dazu bei Hofe nötig sind, klären wir Sie im POKE-Anhang auf.

Denn nach dem Motto «Ehrlich währt am längsten» gehen wir erst mal nur den legalen Weg. (Na, Sie werden doch nicht etwa zum POKE-Anhang weiterblättern wollen?!)

An dieser Stelle können wir übrigens auf Bekanntes zurückgreifen. Gemeint ist die Adresse, ab der VIC unsere Bitmap lesen soll. Wir kennen sie bereits von der Sonderzeichendefinition: 53 272. Wenn Sie eingeben

```
POKE 53265,59 : POKE53272,24
```

schalten Sie VIC in den Grafikmodus (erster POKE) und legen die Bitmap an die Stelle, die wir vorhin zusammen ausklamüsert haben: 8192. Wenn Sie sich gerade wundern, warum Sie beim zweiten POKE als Wert 24 eingeben sollten, bitten wir Sie, sich nochmals mit der Tabelle bei der Sonderzeichendefinition zu beschäftigen.

Vom Chaos zum Nichts

Was Sie jetzt auf dem Bildschirm sehen, entspricht wahrscheinlich immer noch nicht Ihren Vorstellungen von einer Hires-Grafik. Trösten Sie sich: unseren auch nicht. Je nachdem, welche Art von RAM-Chips Sie eingebaut haben, erscheinen auf Ihrem Bildschirm wieder mehr oder weniger regelmäßige Muster. Der Inhalt der jetzt bereitgestellten 8K-RAM, die bisher zufällige Werte oder auch ein BASIC-Programm enthielten, muß erst so verändert werden, daß er schließlich eine Grafik auf dem Bildschirm bilden kann.

Wenden wir doch am besten wieder dieselbe Methode an wie bei den Sonderzeichen. Lassen Sie uns zuerst einmal nachschauen, wie diese Grafikinformationen überhaupt gespeichert werden. Sie sollten dazu bitte zuerst den Grafikmodus verlassen. Am besten mit ⟨RUN/STOP⟩ + ⟨RESTORE⟩. Und dann löschen Sie bitte den Bildschirm. Wenn Sie das getan haben, wird uns folgendes kleines Programm weiterhelfen:

```
10 POKE 53265,59: POKE 53272,24
20 FOR X=8192 TO 16192: POKEX,0
30 FOR Y=1TO100: NEXT Y
40 NEXT X
```

Wenn Sie dieses Programm mit RUN starten, bemerken Sie zuerst, daß die Bitmap (also 8K-RAM) Stück für Stück gelöscht wird. Aber achten Sie doch mal darauf, wie das geschieht!

(Während Sie diese Erklärung lesen, können Sie das Programm ruhig weiterlaufen lassen.)

VIC ist es gewohnt, 8 Bytes als eine «Speichereinheit» zu lesen. Mit der Überlegung, daß auf den Bildschirm 1000 Zeichen passen (siehe Sonderzeichendefinition), kommen Sie nach einigem Rechnen auf runde 8000 Bytes. Stellen Sie sich die hochauflösende Grafik also in etwa als einen Bildschirm voll mit 1000 ganz verschiedenen Sonderzeichen vor.

(Genauso, wie diese 1000 Sonderzeichen auf dem Schirm ein Muster ergeben würden, ergibt auch der Hires-Speicher eines. 1000 Sonderzeichen gehen natürlich in Wirklichkeit nicht, weil höchstens 256 verschiedene Sonderzeichen gleichzeitig dargestellt werden können.)

Wenn Sie sich jetzt also an dieses Bild gewöhnt haben, dann ist der Rest eigentlich ganz simpel. Genauso, wie VIC 1000 Sonderzeichen aus dem Speicher lesen würde, also jedes als 8 * 8-Matrix, so liest er auch die Hires-Seite. Sollten Sie das alles noch nicht so ganz verstanden haben, dann

schauen Sie sich jetzt noch mal Ihren Bildschirm an. Wenn Sie genau beobachten, dann erkennen Sie, daß beim Löschen immer eine Art Block von oben nach unten gelöscht wird. Sollten Sie das nicht erkennen, könnte es daran liegen, daß Sie das Programm von vorhin unterbrochen haben oder es – wir wagen gar nicht, es auszusprechen – womöglich noch überhaupt nicht eingegeben haben.

Wenn wir jetzt also Ihre Kombinationsgabe und unsere Hinweise in einen Topf werfen, dann ergibt sich folgende Schlußfolgerung: VIC liest, wenn er Hires-Grafiken darstellt, immer einen Speicherbereich von je 8 Bytes und stellt diese, wie Sonderzeichen, untereinander dar. Hat er die erste Speichereinheit auf diese Weise erledigt, kommt die zweite, die auf dem Bildschirm neben der ersten angesetzt wird, und so weiter und so fort. Und weil er es beim Löschen genauso macht, können Sie das bei Ihrem laufenden Programm auch sehr schön beobachten.

So, jetzt haben wir hoffentlich alle Klarheiten beseitigt. Mittlerweile müßte Ihr Commodore in etwa ungefähr ein Viertel bis die Hälfte des Bildschirms gelöscht haben. Bevor er sich bei dieser atemberaubenden Geschwindigkeit eine Erkältung vom Zugwind holt, unterbrechen Sie das Programm am besten mit ⟨RUN/STOP⟩ + ⟨RESTORE⟩. Dann können Sie die Zeile 30 – von der Sie ja sicher gemerkt haben, daß Sie nur eine Warteschleife ist – rauswerfen. Und den Rest des Programms speichern Sie am besten ab. Das ist nämlich die *Routine* zum Löschen des Hires-Bildschirms. Sehr viel schneller geht das allerdings auch nicht. Aber ohne *Maschinensprache* oder BASIC-Erweiterung ist das die schnellste Möglichkeit.

(Wenn Sie den Hires-Modus schon eingeschaltet haben, fällt natürlich auch die Zeile 10 des Programms weg. Die Löschroutine ist nur die Zeile 20.)

Am besten, Sie starten das Programm jetzt nochmals, damit es, während wir noch beim Erklären sind, schon mal den Bildschirm frei macht. Ist das Programm erst mal fertig, dann werden Ihnen einige farbige Kästchen auffallen, die von Größe und Umfang her eine erstaunliche Ähnlichkeit mit dem *Cursor* haben.

Um jetzt sich eventuell anbahnende Theorien auch sofort zu untermauern: Tippen Sie ruhig mal ein bißchen auf der Tastatur herum.

Das geht selbstverständlich erst, wenn Ihr kleines Programm von vorhin fertig ist. Wenn nicht, gedulden Sie sich noch ein bißchen. Am besten, Sie entspannen sich jetzt mal etwas. Lehnen Sie sich zurück und genießen Sie das Gefühl, schneller als der Computer fertig gewesen zu sein.

Wenn Ihr Computer jetzt soweit ist, dann tippen Sie mal. Vielleicht fällt

Ihnen auf, daß immer, wenn Sie dieselbe Taste drücken, auch dieselbe Farbe auf dem Bildschirm erscheint. Auch hier rückt wieder eine krimimäßige Schlußfolgerung in greifbare Nähe. Die richtige Lösung wäre: Im Grafikmodus bedeuten Buchstaben offensichtlich Farben. Und das liegt daran, daß das normale Bildschirm-RAM (Sie erinnern sich vielleicht an die Speicheraufteilung, wo dieser Bereich erklärt wurde) jetzt zum Farbspeicher für Hires gemacht wird. Das Bildschirm-RAM wurde aus der einfachen Tatsache heraus verwendet, daß es ansonsten in Hires ziemlich unnütz im Speicher herumliegt. Nun keimt vielleicht im Unterbewußtsein die Frage heran, die da heißt: «Ein Farbspeicher? Wofür?» Ganz einfach. Die Punkte der Hires-Grafik können 16 verschiedene Farben annehmen. Dasselbe gilt auch für die Hintergrundfarbe.

Und jetzt kommt die große Stunde unseres kleinen Freundes VIC. Er bastelt nämlich völlig eigenmächtig an einer Art High-Byte/Low-Byte-Prinzip herum. Was der Prozessor davon denkt, interessiert unseren guten VIC dabei ungefähr so stark, wie wenn in der Madison Avenue in New York ein Fahrrad umfallen würde. VIC wendet einen simplen Trick an. Er nutzt die oberen vier Bits einer Speicherzelle (also sein selbst gedrechseltes «High Byte») als eigenen Speicher für die Zeichenfarbe und die unteren vier (also das genauso im Do-it-yourself-Verfahren entstandene «Low Byte») für die Hintergrundfarbe. Damit kann er sich wenigstens ein bißchen für die ständigen Intrigen gegen ihn rächen.

Sollte Sie also jetzt plötzlich der drängende Wunsch überkommen, eine Grafik in Dunkelblau auf hellblauem Hintergrund darzustellen, müßten Sie in jede Speicherzelle von 1024 bis 2024 den Wert 6 * 16 + 3 POKEn. Wie diese Zahlen zustandekommen? Das ist eigentlich recht einfach: 6 steht für das Dunkelblau und 3 für das Hellblau. Bliebe nur noch die Zahl 16 zu erklären. Wie gesagt, muß VIC jetzt eine Speicherzelle in zwei Werte unterteilen. Um dabei das eine halbe Byte vom andern halben Byte zu unterscheiden, multipliziert er den ersten Wert (also so ähnlich wie ein richtiges High Byte mit 256) mit 16. Wenn Sie das nicht verstehen, dann können Sie noch mal in der Speicheraufteilung unter der Überschrift «Wenn Bytes halbe-halbe machen» nachschauen. So ähnlich, wie es die Skizze dort zeigt, läuft das auch hier. Andererseits ist es nicht so wichtig. Sie können die ganze Sache einfach hinnehmen – aber nicht vergessen. Sollte Sie übrigens nicht der Wunsch nach ausgerechnet dieser erwähnten Farbkombination überfallen haben, so verweisen wir auf Seite 61 Ihres Commodore-Handbuchs, wo Sie eine Farbtabelle finden. Die ganze POKErei machen Sie mit einer kleinen Routine.

```
FOR X=1024 TO 2024: POKE X,99: NEXT
```

Es empfiehlt sich, diese und alle anderen Anweisungen, die die Grafik betreffen, in einem Programm abzuarbeiten. Dafür sprechen verschiedene Gründe.

Im Falle der Hintergrundfarbe zum Beispiel der, daß der Computer durch jedes «SYNTAX ERROR» oder «READY» die Farbeinstellung wieder kaputtmacht, weil er die Buchstaben ja als Signale für die Hintergrundfarbe ansieht. Zweitens ist es ja so, daß Sie das, was Sie schreiben, gar nicht sehen, wenn Sie im Grafikmodus sind und sich deshalb sehr leicht vertippen. Drittens werden Sie bei der Arbeit mit der hochauflösenden Grafik bestimmte Operationen immer wieder brauchen. Die können Sie dann als Unterprogramme ablegen.

Damit wären wir der ganzen Angelegenheit doch schon ein ganzes Stück auf das Bit gekommen. Das heißt, wir haben eigentlich alle Vorbereitungen getroffen, die nötig waren. Bliebe also nur noch eine Kleinigkeit, die wir vielleicht noch erklären sollten: Wie mache ich jetzt eigentlich eine Hires-Grafik?

Lehrer in der Schule haben auf solch fundamentale Sätze, vor allem, wenn Sie von Schülern einfach mitten ins Unterrichtsgeschehen geworfen werden, meist als erste Antwort: «Eine sehr gute Frage».

Wir möchten die Gelegenheit nutzen und uns hier an dieser Stelle zum erstenmal auf die Seite des Lehrkörpers stellen und mit ihm diese Formulierung teilen.

(Diesen kurzen Absatz widmen wir hiermit all denen, die sich in langen Schuljahren mit uns herumplagen mußten, beziehungsweise noch immer müssen. Ihre Arbeit war nicht umsonst!)

Von Autos, Bits und Sinuskurven

Bisher haben uns die Grafikbefehle, die der Commodore nicht hat, noch nicht sehr gefehlt. Aber jetzt wird's schon etwas schwieriger, so ganz ohne.

Nichtsdestotrotz wollen wir mutig (wie Orpheus in die Unterwelt) in die Hires-Page einsteigen.

Mittlerweile wissen wir, daß im Hires-Speicher ein Bit einem Punkt auf dem Bildschirm entspricht. Deshalb müssen wir uns jetzt eine gewisse Virtuosität im Jonglieren mit diesen kleinen Kerlchen aneignen.

Zuletzt haben wir bei den Sonderzeichen mit Bits gearbeitet. Im Grunde entspricht der Aufbau einer Hires-Grafik dem Basteln eines Sonderzeichens.

Um Ihnen das Gefühl zu geben, etwas bekanntes Terrain zu betreten, wollen wir unser kleines Auto aus dem Sonderzeichenkapitel noch einmal hernehmen.

Zuerst einmal sollten wir uns einen Parkplatz dafür aussuchen, auf dem wir keinen Strafzettel bekommen. Nehmen wir also mal die Adresse 12 184. Die liegt so ziemlich in der Mitte unseres Bildschirms. Wieder einmal folgt ein kleines Programm, das unser Auto dort einparkt.

```
10 POKE 53265,59:POKE53272,24
20 FOR X=0TO7:READ A:POKE12184 + X,A:NEXT
30 FOR Y=1024 TO 2024:POKEY,99:NEXT
40 DATA 0,0,56,116,254,254,68,0
```

Ist doch ganz hübsch, oder?

Natürlich hätten wir das gleiche mit Sonderzeichen auch haben können. Der Vorteil der hochauflösenden Grafik ist aber, daß Sie volle Kontrolle über 320 * 200, also über 64 000 einzelne Punkte haben. Und damit wird auch die Anwendung der Hires-Grafik deutlich: überall da, wo große detaillierte Figuren und Bilder gebraucht werden, wo viele einzelne Daten dargestellt werden sollen oder um Funktionsgraphen auf dem Computer darstellen zu können.

Will man aber zum Beispiel ein möglichst naturgetreues Gesicht auf den Schirm bringen, gelingt das am besten nur mit aufwendigen *Tools* oder gar mit *Hardware-Hilfen* wie einem sogenannten *Scanner*. Was kann man aber ohne solche Hilfsmittel machen?

Um das zu klären und auch zu zeigen, ist wieder mal ein kurzer Abstecher in die Binärrechnung nötig. Sie sehen schon: Diese Rechenart treffen wir so sicher immer wieder wie bestimmte Gallier bestimmte Piraten.

Wir wollen aber keine Seeschlacht, sondern eine Formel finden, der wir eine X- und eine Y-Koordinate übergeben und die uns dafür die entsprechende Adresse in der Bitmap liefert. Und da alle, denen Mathematik nichts gibt, die nächsten Seiten sowieso überschlagen werden, so sei es ihnen hiermit (seufz ...) auch erlaubt.

Aha! Da es keinen besonderen Mathematikergruß gibt, beglückwünschen wir Sie hiermit zu Ihrem Mut.

Allen, die sich, obwohl Sie nicht gut in Mathe sind, trotzdem hierher vorgewagt haben, sei noch einmal besonders Trost zugesprochen. Sie sind nicht allein! Nehmen wir uns also bei der Hand, gehen wir gemeinsam in das Reich der Zahlen ein.

Nach alledem, was wir jetzt über den Bildschirm und den Hires-Speicher wissen, kann die Y-Koordinate unseres Punktes nur zwischen 0 und 199 liegen. Alles, was darüber und darunter liegt, wäre nur noch auf der Tastatur, dem Zimmerboden oder der Decke zu lokalisieren. Die X-Koordinate unseres Punktes muß zwischen 0 und 319 liegen, sonst würde sie die Zimmerwände zieren.

Als erstes Etappenziel während unseres Marathons zur Grafik setzen wir uns am besten, zuerst mal die Spalte und die Speichereinheit (also den 8-Byte-Block) zu lokalisieren. Das ist noch verhältnismäßig einfach. Da sowohl horizontal wie auch vertikal ein Zeichen (also eine Speichereinheit) 8 Punkte hat, müssen wir unsere X- und Y-Werte nur in 8er-Schritten runden. Warum ausgerechnet 8er-Schritte? Ganz einfach: Eine Speichereinheit, also einer der Blocks, aus der Hires aufgebaut wird, besteht aus 8 * 8 Punkten. Will man einem Block eine Nummer geben, damit man ihn lokalisieren kann, muß man eben in 8er-Schritten zählen. Und damit es genaue 8er-Schritte werden und der Computer nicht rundet, muß der BASIC-Befehl INT dazu angegeben werden (siehe auch BASIC-Kapitel).

```
Zeile=INT(Y/8): Spalte=INT(X/8)
```

Wenn Sie also den 64. Punkt in der 1. Zeile suchen, liegt der im 8. Zeichen bzw. in der 8. Speichereinheit. Um auf die richtige Einheit zu kommen, müssen Sie allerdings in Computermanier bei Null anfangen zu zählen. Es ist also eigentlich das 9. Zeichen gemeint. Der 65. Punkt liegt immer noch im 8. Zeichen. Weil aber die Speichereinheiten, wie wir vorhin gesehen haben, von oben nach unten aufgebaut werden, also reihenweise, fehlt noch eine Information zur genauen Lokalisierung. Wir müßten noch rauskriegen, in welcher Reihe der Speichereinheit, d. h. in welchem Byte genau unser Y-Wert liegt. Dazu brauchen wir den Restwert der Rundung. Um den zu erhalten, schließt sich eine etwas kompliziertere Formel an.

```
REIHE=((X/8)-(INT(X/8)))*8
```

Wir ziehen also den gerundeten Wert vom Ganzen ab. Schreibfaulen und Binärfreunden (z. B. Computern) sei hier verraten, daß

```
REIHE=Y AND 7
```

genau denselben Effekt hat. Über diese Art der Binärrechnung und der *Booleschen Algebra* unterhielten wir uns im Zwischenspiel 3.

Ganz fertig sind wir aber immer noch nicht. Bleibt nämlich noch, das

richtige Bit zu treffen. Sollte Ihnen das alles etwas Schwierigkeiten machen, dann empfehlen wir, noch einmal die Grafik über die Darstellung der Zeichen im Sonderzeichenkapitel nachzuschlagen. Da läßt sich das auch mitverfolgen. In hartnäckigen Fällen von Nichtverstehen helfen am besten 2 Aspirin und etwas Ruhe.

Um unser Bit rauszupuzzeln, brauchen wir ebenfalls zunächst wieder den Rest der Rundung.

```
Bit=7-((X/8)-(INT(X/8))*8
```

Das Ergebnis dieses Ausdruckes (die «überzähligen» Bits) müssen wir von 7 abziehen. Denn der Computer hat nicht nur seine eigene Art, Bits zu zählen – natürlich nicht von 1 bis 8, sondern von 0 bis 7 –, nein, er muß sie auch noch rückwärts zählen. 1 Byte ist daher

```
1 Byte = 7  6  5  4  3  2  1  0
```

Wenn jetzt also das 4. Bit gesucht ist (Sie wissen schon: 0, 1, 2, 3, 4 – gemeint ist natürlich das 5. Bit ...), dann bringt uns die Formel 7 – 4 auf die Lösung. Das ergibt nämlich 3. Und das ist auch das richtige Bit.

Auch hier ist der einfachere Weg die Boolesche Algebra. Die Lösung würde lauten

```
Bit=7-(X AND 7)
```

Bevor Sie jetzt schluchzend zusammenbrechen, keine Sorge: Wir haben es gleich geschafft. Jetzt brauchen wir nur noch die Formel:

```
Byte=8192+(Y AND 7)+(8*INT(X/8))+320*(INT(Y/8))
```

Das also ist die kürzeste Formel zur Berechnung eines Punktes in Hires. Vielleicht verstehen Sie jetzt, was wir mit unkomfortabler Handhabung der Hires-Page meinten ...

Um jetzt ein Bit in diesem Byte zu setzen, brauchen wir nur noch den Befehl

```
POKE(BY),PEEK(BY) OR 2^(7-(X AND 7))
```

Das OR in diesem Befehl ist ebenfalls eine *Boolesche Verknüpfung*. Wir begrüßen übrigens auch die Leser aus nichtmathematischen Landen wieder unter uns. Für alle Abtrünnigen also ein kurzes Resümee: Das Ergebnis unserer Bemühungen ist die POKE-Zeile oben. Nicht erschrecken. Es sieht schlimmer aus, als es ist. Alles, was Sie außerdem noch brauchen, ist die Formel unmittelbar darüber, die die Adressen für die Bytes, also für den

Platzhalter «BY» in der POKE-Zeile errechnet. Wenn Sie diese beiden letzten Zeilen miteinander verbinden, können Sie ein kleines Programm schreiben, das für Sie immer die nötigen POKE-Werte errechnet. Andererseits steht dieser Programm-Selbstentwicklung natürlich wieder die Verlockung eines fertigen Listings im Anhang gegenüber. Was Sie jetzt tun, abschreiben oder selbermachen, das überlassen wir Ihnen.

Noch ein Hinweis für alle, die wacker mitgekämpft haben. Sollten Sie das alles nicht so recht verstanden haben, gibt es auch hier wieder eine einfache Methode, sich aus der Affäre zu ziehen: Nehmen Sie das alles erst einmal hin. Irgendwann später können Sie diesen Teil ja noch einmal lesen. Und dann verstehen Sie es bestimmt.

Und jetzt ist der große Augenblick gekommen. Wir wollen unsere Kenntnisse zum erstenmal anwenden, und zwar, indem wir eine Sinuskurve *plotten*. Wer sich das schon ohne Unterstützung zutraut, der sollte das jetzt tun. Alle anderen dürfen wir in den Listinganhang bitten. Hier finden Sie das notwendige Programm dazu. Da der Commodore eben keine Grafikunterstützung bietet, weichen wir hier ausnahmsweise etwas von unserem Konzept, kein Tippbuch zu sein, ab und bieten Ihnen im Listinganhang eine Reihe von Programmen, die die notwendigsten Grafikmöglichkeiten abdecken.

Wenn's Speicherprobleme gibt ...

Es könnte sein, daß die Bitmap manchmal ein BASIC-Programm stört oder umgekehrt, weil zufällig beide denselben Speicherbereich brauchen. In diesem Falle helfen uns die Kenntnisse über die Speicheraufteilung aus der Patsche.

Wir machen einfach auch hier das gelernte Spielchen. Gemeint ist, daß wir den BASIC-Anfang hochsetzen – am besten hinter die Bitmap, also auf die Adresse 16348. Eigentlich müßten Sie das ja schon selbst können. Aber für alle, die zumindest den Willen hatten, unserem Mathe-Exkurs zu folgen, wollen wir hier noch einmal einen kleinen Service bieten. In diesem Sinne:

```
POKE 44,64:POKE16348,0:NEW
```

Damit bleiben immer noch 24K für Ihr BASIC-Programm. Den Bereich von 2048 bis 8192 (also ca. 6K) könnten Sie anderweitig nutzen, beispielsweise für *Sprites*, dazu mehr im nächsten Kapitel, oder für kleinere BASIC-Programme (siehe Speicheraufteilung).

Noch ein Tip. Durch das Setzen des Anfangszeigers auf 8192 und des Endzeigers auf 16348 können Sie die ganze Bitmap auf Diskette oder Kassette speichern.

Wenn Sie gerade verzweifelt versuchen, POKE 44,8192 einzugeben, müssen wir Sie leider noch einmal auf das Kapitel Speicheraufteilung verweisen, wo Sie den richtigen POKE für diese Adresse finden.

Sie geben einfach ein

```
POKE 44,32
```

Ein NEW erübrigt sich in diesem Fall, da es die Bitmap verändern könnte. Jetzt geben Sie wie gewohnt ein SAVE ein. Dabei müssen die BASIC-Endzeiger auf ein BASIC-Programm über der Bitmap zeigen. Wollen Sie nur die Bitmap abspeichern, setzen Sie das Ende auf 16348 mit

```
POKE 46,64
```

Diese Art stellt sicherlich den schnellsten und elegantesten Weg dar, Hires-Grafiken abzuspeichern. Sie können dann auch wieder ganz normal mit LOAD geladen werden.

Commodore macht's so bunt, bunter geht's nicht

Also, das muß man ja sagen: Mit Farbmöglichkeiten hat Commodore wirklich nicht gegeizt. Zwei besondere Farbmodi gibt es noch im Zusammenhang mit der Hires-Grafik.

Da wäre zuerst einmal der Hintergrundfarbmodus (das ist eine freie Übersetzung – das englische Original heißt Extended Background Color). Der Zusammenhang dieses Modus mit der Hires-Grafik ist der, daß sich die beiden nicht riechen können. Das heißt, Hires und er, das geht nicht. Oder besser gesagt, er bringt im Hires-Modus soviel wie ein Blatt Zeitungspapier vor dem Druck.

Dafür erweist er sich im normalen Textmodus als hübsche Erweiterung. Eingeschaltet wird diese Hintergrundfarbe mit

```
POKE 53265,91
```

Haben Sie ihn erst mal an, heißt das auch Selbstbeschränkung. Denn dann gibt es nur noch die Zeichen mit den Bildschirmcodes von 0 bis 63. Vergleichen Sie dazu auch Ihr Commodore-Handbuch auf Seite 133.

Aber dafür können jetzt alle diese Zeichen vier verschiedene Hintergrundfarben haben, die gleichzeitig auf dem Bildschirm dargestellt werden können. Welche Farben das sind, können Sie frei von 0 bis 15 wählen. Bevor Sie jetzt anfangen, das alles zu verstehen, schauen Sie sich Tabelle 6.1 an.

Bildschirmcode	Farbadresse
0 bis 63 = (normale Hintergrundfarbe)	53281
64 bis 127 = (SHIFT)	53282
128 bis 191 = (REVERSE)	53283
192 bis 255 = (SHIFT & REVERSE)	53284

Tabelle 6.1 Hintergrundfarbmodus

Wahrscheinlich stellt sich jetzt bei Ihnen mehr und mehr die Frage: Was soll das alles? Keine Sorge – es ist alles gar nicht so kompliziert, wie es ausschaut. Erst einmal klären wir zum besseren Verständnis, was überhaupt passiert. Normalerweise können Sie beim Commodore ja vier verschiedene Schriftarten darstellen, als da sind: NORMAL, SHIFT, REVERSE und zu guter Letzt SHIFT & REVERSE gleichzeitig. Erinnern Sie sich noch ganz an den Anfang dieses Buches? Da haben wir eine eigentlich ganz simple Sache erklärt: die ASCII-Codes. Wenn Sie sich nicht mehr daran erinnern, blättern Sie noch mal in der Vergangenheit. In unserer Buchgeschichte wäre das die TASTATURZEIT, auch PRÄTASTIÄR genannt, so in der Gegend um 4. n. V. (das heißt vier nach Vorwort und heißt nach neuerer Kapitelrechnung 2).

Fein, wenn Sie jetzt also wieder wissen, was ASCII-Codes sind und wozu der Computer sie braucht, dann dürfte der Rest nicht mehr schwerfallen. Um die verschiedenen Schreibmodi voneinander zu unterscheiden, benutzt der Computer natürlich wieder nur Zahlen, und zwar genau jene ASCII-Codes, von denen wir gerade sprachen. Aber damit es nicht zu einfach wird, ist ein kleiner Stolperstein eingebaut. Es wäre ja jetzt wirklich zu einfach, wenn die ASCII-Codes den Zahlen entsprechen würden, die wir oben als erste Rubrik in unserer Tabelle haben. Dem ist nicht ganz so: Die Zahlen der Tabelle sind Bildschirmcodes. Diese Codes ergeben sich für den Computer aus den gedrückten Tasten bzw. aus den ASCII-Codes, die die Tasten zurückwerfen. Die ASCII-Codes werden umgewandelt und schließlich im Bildschirmspeicher abgelegt. Findet der Computer jetzt in diesem Bildschirmspeicher den Wert 0,

so stellt er ein bestimmtes Zeichen im Modus NORMAL dar. Findet er den Wert 64, so stellt er zwar dasselbe Zeichen dar, aber im SHIFT-Modus. Bei 128 ist's REVERSE und bei 192 SHIFT und REVERSE gleichzeitig. Übrigens sind zur Freude aller Beteiligten alle genannten Zahlen selbstverständlich im Speicher binär abgelegt.

Mittlerweile werden Sie – mit Recht – feststellen, daß das herzlich wenig mit Farben zu tun hat. Der Trick dabei ist aber folgender.

Anstatt diese Bildschirmcodes wie bisher zum Unterscheiden der Darstellungsarten zu benutzen, macht VIC daraus eben eine Art Farbspeicher. Er stellt die Buchstaben statt REVERSE mit einer anderen Hintergrundfarbe dar, die er in der Speicherzelle 53283 als Wert (von 0 bis 15) findet. Allerdings kann er ab jetzt nur noch Großbuchstaben darstellen.

Nachdem wir jetzt offensichtlich klargemacht haben, was passiert, wollen wir doch gleich mal diese positive Erfolgsstimmung ausnutzen, um zu erklären, was Sie machen müssen, damit Sie was davon haben. Sie könnten aber auch selbst darauf kommen und so Ihren eigenen Wissensstand ein bißchen überprüfen. Lesen Sie also nicht gleich weiter, sondern knobeln Sie erst mal etwas.

Gut, jetzt wollen wir es aber erklären. Hier folgt also die Auflösung: Sie können vier Farben gleichzeitig auf dem Bildschirm haben, weil es nur maximal vier verschiedene Arten gibt, das gleiche Zeichen darzustellen. Welche vier Farben das sind, können Sie aus 16 verschiedenen Angeboten aussuchen. Wenn Ihre Wahl getroffen ist, speichern Sie die Werte der Farben in vier Speicherzellen, und zwar in 53 281, 53 282, 53 283 und 53 284. Um dann die entsprechende Texthintergrundfarbe zu bekommen, drücken Sie einfach die ⟨SHIFT⟩-, die ⟨REVERSE ON⟩- oder die ⟨SHIFT/LOCK⟩- + ⟨REVERSE ON⟩-Tasten. Je nachdem, welchen Wert Sie in die einzelnen Speicherzellen gePOKEd haben, ändert sich dann auch die Hintergrundfarbe Ihres Textes. Aber bitte: Lesen Sie das nicht einfach alles durch. Probieren Sie es aus. Dann wird es am schnellsten klar.

Und jetzt noch zwei Tips dazu.

Erstens: Immer, wenn Sie den Wert einer dieser «Farbspeicherzellen» ändern, ändert sich auch alles auf dem Bildschirm, was in der alten Farbe dastand, in die neue Farbe. Damit lassen sich ganz hübsche Effekte erzielen, wenn man zum Beispiel den Text farbig blinken läßt. Wen es interessiert: Das liegt daran, daß der Bildschirm ständig neu aufgebaut wird, also das Bildschirm-RAM neu gelesen und eine aktuelle Version auf den Schirm gebracht wird.

Der zweite Tip: In Verbindung mit selbstdefinierbaren Zeichen ist dieser ganze Farbmodus deshalb manchmal ganz vorteilhaft, weil Sie dann automatisch nur die Codes 0 bis 63 kopieren und definieren müssen. Das geht schneller und spart allerhand Speicherplatz, der sonst für die zwar normalerweise mitkopierten, aber nicht gebrauchten Zeichen benötigt würde. Sie brauchen statt dessen für den Zeichensatz nur 504 Speicherzellen und können ab 2560 (also mit POKE 44,10) wieder BASIC-Speicher anlegen.

Der wichtigere Vorteil ist aber, daß, wenn Sie die SHIFT-Funktion oder REVERSE benutzen, trotzdem nicht zufällige Bitmuster auf den Schirm kommen, obwohl Sie diesen Teil der Zeichen nicht mitkopiert haben. Das würde ja normalerweise passieren, wenn Sie den Zeichensatz nur teilweise kopieren und VIC eine Lese-Start-Adresse im RAM angeben. Aber nicht vergessen: Das alles geht nur im Textmodus. In Hires haben Sie leider nichts davon. Deshalb gibt es auch (außer ⟨RUN/STOP⟩ + ⟨RESTORE⟩) zwei Möglichkeiten, den Extended-Background-Color-Modus auszuschalten. Entweder normal mit

```
POKE 53265,27
```

oder durch das Einschalten der Hires-Seite. Die beiden sind wirklich wie ein verkrachtes Ehepaar: «Kommst Du, dann geh' halt ich!»

Für Hires wesentlich interessanter ist da schon der *Mehrfarbmodus* – vor allem, weil er in gleicher Weise auf Zeichen, Hires und Sprites wirkt. Auch hier wollen wir zuerst verraten, wie man ihn einschaltet.

```
POKE 53270,216
```

Sollten Sie gerade damit beschäftigt sein, diesen POKE im Textmodus auszuprobieren, könnte es sein, daß Sie plötzlich Schwierigkeiten haben, den Text, der da steht, noch lesen zu können. Aber keine Panik. Bevor Sie den Augenarzt aufsuchen, schauen Sie noch mal genau hin. Mit einiger Mühe läßt sich doch Textähnliches erkennen. (Wenn Sie nichts erkennen, wäre das mit dem Augenarzt andererseits vielleicht doch nicht so schlecht ...)

Sollte überhaupt nichts passieren, so ist das andererseits kein Grund, Ihren Commodore in Reparatur zu geben. Wählen Sie statt dessen einfach eine Zeichenfarbe über 7 – also alle Farben, die mit der ⟨C=⟩-Taste gewählt werden.

So, jetzt ist aber hoffentlich bei allen der Text nur noch schwer zu entziffern. Was ist also passiert?

Na, vielleicht fangen wir mit dem Unangenehmen zuerst an. Sie haben jetzt

nur noch die halbe Auflösung. Das heißt, im Text nur mehr 4 * 8 Punkte pro Schreibstelle. Damit erklären sich auch Ihre Leseprobleme. In der Grafik haben Sie jetzt nur noch 160 * 200 Punkte zur Verfügung.

Jetzt das Angenehme. Sie können jetzt jedem einzelnen Punkt der Hires-Grafik oder auch der Sonderzeichen eine von drei verschiedenen Farben geben. Am besten erklären wir Ihnen gleich, wie das zustande kommt.

Normalerweise ist die niedrigste Auflösung immer ein Bit oder ein Punkt gewesen. Weil jetzt aber auch noch die Information mit rein muß, in welcher Farbe jeder Punkt dargestellt werden soll, und ein Bit nunmal nur 1 oder 0 sein kann und damit unmöglich alle Informationen aufnehmen kann, wurde auf die höchste Auflösungsstufe verzichtet. Damit besteht jetzt jeder «Punkt» dieser Grafik tatsächlich aus zwei Punkten. Dadurch können die Bits, wie in Tabelle 6.2 dargestellt, aussehen. Die Adressen, deren Inhalte geändert werden müssen, sehen Sie in Tabelle 6.3.

Punkt	Farbnummer
1, 00	0
2, 01	1
3, 10	2
4, 11	3

Tabelle 6.2 Bitmuster im Multicolormodus

Im Lores/Text-Modus ist zu ändern:
1, 53281
2, 53282
3, 53283
4, Farb-RAM ab 55296
und im Hires-Modus:
1, 53281
2, die Bits 7 bis 4 im Bildschirm-RAM
3, die Bits 3 bis 0 im Bildschirm-RAM
4, das Farb-RAM ab 55296

Tabelle 6.3 Farbadressen im Multicolormodus

Wenn Ihnen diese Zahlenkolonnen noch etwas dubios erscheinen, keine Sorge. Wir wollen uns die ganze Geschichte zunächst einmal im Textmodus ansehen. Zuerst geben Sie bitte ein

```
POKE53265,27: POKE53270,216
```

Im Textmodus – und das ist auch die Erklärung dafür, warum vielleicht bei Ihnen zunächst gar nichts passierte – ist der Mehrfarbmodus nur dann aktiv, wenn die Schriftfarben, die verwendet werden, größer als 7 sind – also, wie schon gesagt, alle Farbtasten, die mit der ⟨C=⟩-Taste erreicht werden können. Wenn Sie einige Buchstaben schreiben, am besten auch ein paar in REVERSE, werden Sie am schnellsten erkennen, worum es eigentlich geht. Für die Schriftfarbe gibt es jetzt sowieso nur noch 8 Möglichkeiten. Wenn Sie Farben mit der ⟨CTRL⟩-Taste anwählen, scheinen sie normal. Wenn Sie ⟨C=⟩ + eine Farbtaste drücken, sind Sie im Mehrfarbmodus. Die weiteren Kombinationen probieren Sie am besten durch POKEs in die Adressen 53 281 bis 53 283 aus. Bei der Verwendung von selbstdefinierten Sonderzeichen lassen sich so tolle Effekte erzielen. Was wir darunter verstehen, probieren Sie am besten mit einem Listing im Anhang aus. Bedenken Sie aber bei Ihren eigenen Entwürfen die vier möglichen Bit-Kombinationen.

Und jetzt zur Hires-Grafik. Wie Sie an der Tabelle erkennen können, läuft es hier etwas anders. Geben Sie bitte folgendes kleine Beispielprogramm ein

```
10 POKE 53265,59: POKE 53272,24: POKE 53270,216
15 POKE 53281,6
20 FOR X=8192 TO 12352 STEP 8
30 FOR Y=0 TO 3: POKE X+Y,250:NEXT
40 FOR Y=4 TO 7: POKE X+Y,80:NEXT
50 NEXT X
60 FOR X=0 TO 511: POKE 1024+X,5: POKE 55296+X,7
70 NEXT
```

Das Programm schaltet zuerst in Hires- und Multicolormodus. Dann stellt es einen halben Bildschirm voll Zeichen dar.
Als nächstes werden die Farben 0 (= Schwarz) und 5 (= Grün) nach folgender Formel in das Bildschirm-RAM gePOKEd:

```
0*16+5=5
```

Also 0 in die obere Hälfte (damit der Computer das auch weiß, wird die Null ja mit 16 multipliziert) und 5 in die untere Hälfte. Die 7 (= gelb) kommt dann

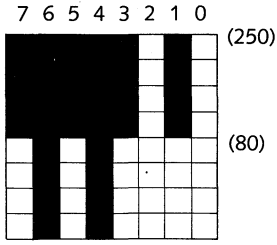


Bild 6.1 Zeichenmatrix Multicolor-Beispiel

ins Farb-RAM. Der Bildschirm ist bei alledem blau. Sie sehen jetzt auf einer Schreibstelle vier Farben: Schwarz, Grün, Gelb, Blau. Benutzen Sie das Programm zum Testen anderer Farbkombinationen. Die Zeilen 20 bis 50 können Sie, um Zeit zu sparen, löschen. Aber speichern Sie das Programm zuerst! Noch ein Beispiel: Wenn Sie die Farben Blau, Rot, Weiß und Orange gleichzeitig auf einer Schreibstelle haben wollen, müssen Sie ins Bildschirm-RAM 6 * 16 + 2 (Blau + Rot) = 98, ins Farb-RAM 1 (= Weiß) und in die Adresse 53 281 den Wert 8 (= Orange) POKEn. Selbst auf die Gefahr hin, daß wir Sie damit nerven: Auch hier gilt, nur lesen hat keinen Sinn. Probieren Sie es aus!

Und noch einmal zur Erinnerung: Wenn Sie eigene, hochauflösende Mehrfarbgrafiken probieren wollen, denken Sie an die vier möglichen Bitkombinationen.

Das war's dann auch erst mal von unserer Seite zum Thema Hires-Grafik. Natürlich sind da jetzt ganze Berge von POKEs und Informationen auf Sie eingestürzt. Also lassen Sie sich jetzt mal ruhig etwas Zeit mit dem nächsten Kapitel. Ausprobieren und üben ist der beste und sicherste Weg, schnell zu lernen. Benutzen Sie auch die Programme zum Erstellen von Formen und Figuren in Hires. Spielen Sie einfach ein bißchen. Hier gilt – wie überall im Buch – das, was wir schon ganz am Anfang sagten. Eines kann Ihnen kein Buch ersetzen: Die Erfahrung mit Ihrem System.

Und nach diesem ehernen Schlußwort vielleicht noch ein kleiner Tip. Sollten Ihnen die Farbmöglichkeiten des Commodore nicht so viel Spaß machen, denken Sie doch mal über die Anschaffung eines Farbfernsehers nach ...

Vergessen Sie trotz allem nicht, in ein paar Wochen Ihr Buch wieder rauszuholen und weiterzulesen.

Zwischenspiel 4

Da die letzten Kapitel doch ziemlich anstrengend waren, möchten wir Sie hier zu einem kleinen Abenteuerurlaub einladen. Es bietet sich jetzt nämlich die Gelegenheit, mit dem bisherigen Wissen und einer kleinen Portion Forschergeist an einer recht interessanten Sache teilzunehmen: Exklusiv – eine Expedition durch den Dschungel des Commodore-Speichers.

Worum geht's?

In diesem Zwischenspiel sollen zwei Themen, die wir bisher behandelt haben, zusammengebracht werden: Die Speicheraufteilung und die hochauflösende Grafik.

Im letzten Kapitel haben Sie gelernt, wo und wie man die Bitmap der Hires-Grafik anlegen kann. Eine Möglichkeit haben wir dabei gleich wieder verworfen, und zwar die, die Bitmap auf die Zeropage und den daran angrenzenden Speicherbereich zu legen. Aus gutem Grund – denn es gibt Dinge zwischen Tastatur und Platine, die man besser nicht versuchen sollte. Und dazu gehört eben auch das Überschreiben der Zeropage. Da hier außerdem der Bildschirmspeicher und – normalerweise – Ihr BASIC-Programm liegt, ist der Bereich von 0 bis 8192 für eine Hires-Bitmap tabu.

Andererseits hindert uns niemand daran, sich in diesem scheinbaren Dschungel etwas umzusehen. Solange wir keine Rodungen hier veranstalten, kann uns auch gar nichts passieren. Was also durchaus möglich ist, wäre, die Hires-Grafik einzuschalten und die Bitmap von 0 bis 8000 anzulegen. Denn solange wir uns nach der Museumstaktik verhalten (Anschauen erlaubt – Anfassen nicht), ist damit plötzlich die Möglichkeit eines Fensters in den Commodore-Speicher da.

Mit

```
POKE 53265,59:POKE 53272,16
```

läßt sich das alles bewerkstelligen. Und dann bringt uns unser guter Geist VIC den Speicher als Hires-Grafik auf den Bildschirm. Jedes Bit im Speicher entspricht dann einem Punkt dieser Grafik. Und schon haben wir die Möglichkeit, unsere an und für sich sehr scheuen Freunde, die Bits, einmal auf freier Wildbahn zu beobachten. Und das, ohne – wie die meisten Förster – schon morgens um 3 Uhr aufstehen zu müssen. Aber vergessen Sie nicht: Wir befinden uns in einem Wildbit-Park. Ehrensache, daß es bei einer Fotosafari bleibt? Okay, dann alles einsteigen und ab in Richtung Urwaldexpedition. Was einem zuerst auffällt, ist wohl, daß hier gerade Herbst zu sein scheint. Obwohl das eigentlich gegen die Gewohnheit von Urwäldern ist. Der Grund dafür ist recht einfach: Wie Sie ja wissen, wird im Hires-Modus der Bildschirmspeicher zum Farbspeicher umfunktioniert. Dabei dienen die oberen vier Bits als Speicher für Vordergrund-, die unteren vier für die Hintergrundfarbe. Da momentan unser Bildschirm-RAM leer ist, also mit lauter Leerstellen (= Spaces) gefüllt ist, und diese Leerstellen den Code 32 haben, sieht eine solche Zeile folgendermaßen aus:

BIT NR.	7	6	5	4	3	2	1	0
	0	0	1	0	0	0	0	0

Damit sind die ersten vier Bits gleich dezimal 2 und die hinteren vier Bits gleich dezimal 0. Daraus folgt, daß die Hintergrundfarbe Schwarz ist (es wird Nacht über unserem Speicher ...) und der Vordergrund Rot. Falls Sie eine andere Farbe entdecken, haben Sie irgendeinen Text auf dem Bildschirm. Wenn Sie gar keine Farbe entdecken, haben Sie wahrscheinlich einen Schwarzweißfernseher.

Die Tastenkombination ⟨SHIFT⟩ + ⟨CLR⟩ macht dann alles wieder gut (bei Farbfernsehern natürlich nur!).

Sie können den Bildschirmspeicher übrigens beobachten. Er liegt ungefähr im zweiten Achtel Ihrer Hires-Grafik und besteht aus senkrechten Strichen. (Das sind die 32er-Bits von vorhin, die immer an diese Stelle kommen, wenn der Text-Bildschirm leer ist ...) Wenn Sie einige Tasten drücken, sehen Sie, wie die Bitmuster sich ändern. Aber danach wieder aufräumen!

Unmittelbar vor diesem Speicherbereich liegt die Schlucht namens Kassettenpuffer. Dieser Teil des Schirms ist gewöhnlich gähnend leer, also schwarz. Es sei denn, Sie haben unmittelbar vorher mit der Datasette gearbeitet.

Jetzt wollen wir einen kleinen Abstecher in die linke obere Ecke machen.

Auf dem Weg dorthin können wir übrigens einige Eingeborene beobachten.

Es sind die Bits von den Stämmen «Zeitvariable» und «Cursorsteuerung», die hier eine Art Tanz aufführen. Ihre Götter sind der Timestring oder der Blinkzähler des Cursors. Und um diese bei guter Laune (bzw. beim aktuellen Wert) zu halten, sind diese kleinen Bitianer ständig am Rumwuseln.

Ganz links oben aber, da blitzt und funkelt es nur so. Das sind die Schätze des Königs 6510. Aber Vorsicht, wenn Sie nur etwas davon wegnehmen, bricht das Reich zusammen. Dann vergeht lange Zeit, bis die Erlösung in Form des «Großen Stromaus» kommt. (Bevor wir uns aber endgültig in das Reich der Fantasy-Literatur verlieren: Technisch gesprochen sehen Sie hier die Speicherzelle 1, die für die Verteilung von RAM- und ROM-Bausteinen unter der gleichen Adresse zuständig ist. Weil VIC alle paar Millisekunden Zugriff auf das Charakter-ROM und dergleichen haben muß, ist hier ständig Betrieb.)

Im ersten Viertel lagen also Zeropage und Bildschirmspeicher. Für die, die keinen Kompaß haben, sei an dieser Stelle vermerkt, daß wir uns etwa bei der Adresse 2048 befinden.

Danach entdecken Sie jetzt die Wüste RAMI. Eine riesige Landschaft, die von Touristen auch BASIC-Speicher genannt wird. Erkennen kann man diese Landschaft immer an relativ gleichmäßigen Mustern im zweiten Viertel des Schirms. Beim Einschalten haben nämlich alle RAM-Chips mehr oder weniger gleiche Bit-Muster. Es kann natürlich auch sein, daß ein Bautrupp der Zivilisation in Form eines BASIC-Programms mittlerweile da war. Dann kann diese Gegend natürlich ziemlich anders ausschauen. Auf jeden Fall wird es erst mal chaotisch wirken, wie alle moderen Bauten in einer altehrwürdigen Umgebung das halt tun.

In der zweiten Hälfte des Bildschirms steht – der Zeichensatz? Also ab Adresse 4096. Aber das wurde doch vorhin anders erklärt, oder?

Richtig. Der tatsächliche Zeichensatz liegt noch runde 50 KB entfernt. Aber was Sie hier sehen, sehen Sie ja mit den Augen von VIC. Und wir haben ja schon mehrmals darauf hingewiesen, mit welchen Methoden er reingelegt wird. Statt eines Blicks in die freie Natur, muß VIC hier auf den Zeichensatz schauen. Dank des Botschafters glaubt VIC nämlich, hier – ab 4096 – liegt sein Zeichen-ROM. Nutzen Sie die Chance und sehen Sie sich die verschiedenen Zeichen an. Vielleicht entdecken Sie dabei ja einige Anregungen für Ihre selbstdefinierten. Wenn Sie diesen Anblick genossen haben, bitten wir Sie, wieder in den Bus einzusteigen. Durch Drücken von ⟨RUN/STOP⟩ & ⟨RESTORE⟩ treten wir wieder die Heimfahrt an.

7
Sprites auf dem Commodore 64

Die Riesen-Super-Sonderzeichen

Es wäre mal wieder soweit: ein neues Kapitel, ein neues Glück. Mittlerweile haben wir ja zusammen schon eine ganze Menge gemacht. Sie kennen sich inzwischen ganz gut mit der Tastatur Ihres Commodore aus, alle, die es vorher noch nicht konnten, haben noch ein Stück BASIC dazugelernt. Sie wissen, wo was im Commodore-Speicher ist und was man damit alles anfangen kann. Sie können eigene Sonderzeichen definieren und so bereits bewegte Grafik machen. Wir haben zusammen ein Spiel mit eigenen Sonderzeichen programmiert, und auch die Hires-Möglichkeiten des Commodore 64 sind Ihnen mittlerweile kein Geheimnis mehr. Das ist doch insgesamt gesehen ein ganzes Stück Arbeit, das wir da geschafft haben, nicht wahr?

Halt, halt, es hat keiner was von gemütlich zurücklehnen gesagt. Vor uns liegen noch die Sprites, ein hübsches Spiel, Musik und Geräusche, Antworten zu allen Fragen in bezug auf die Steuerung von Figuren mittels Joystick, Paddles und Tastatur und die Handhabung von Diskettenstationen oder anderer Peripherie.

Gehen wir also zuerst ans Nächstliegende. Das wären in unserem Fall die Sprites. Als der Commodore auf den Markt kam, bot diese Möglichkeit keiner der anderen günstigen Home-Computer. Erst jetzt kommen die Dinger in Mode.

Normalerweise ist es bei Computern, die keine Sprites haben, gar nicht einfach, bewegte Grafiken zu machen, die größer sind als ein Zeichen. Entweder wurde das mit verschiedenen, kombinierten Sonderzeichen (und gleichzeitig eindeutigen Zugeständnissen an die Erkennbarkeit) erreicht oder es wurden munter drauflos Sonderzeichen definiert, die mittels ständigem

Löschen und Neudrucken «bewegt» wurden. Und wenn keine dieser Möglichkeiten funktionierte – nicht jeder Computer hat Grafikzeichen, und schon gar nicht bei jedem Computer kann man sie so einfach verändern –, dann blieb nur noch eins: Ein Objekt in Hires-Grafik zu entwerfen und dann ebenfalls durch Löschen, Neuzeichnen, Löschen, Neuzeichnen usw. zu bewegen – also eigentlich eine kleine Hires-Grafik in einer großen Hires-Grafik zu bewegen. Mal abgesehen von einer nicht ganz unkomplizierten Handhabung der genannten Methoden, kommen noch einige ganz handfeste Nachteile dazu. Erstens haben alle drei Arten die Eigenschaft, beim Löschen der Sonderzeichen oder Grafiken auch gleich den Hintergrund mitzulöschen. Sie lassen es nicht zu, auf zwei Ebenen zu arbeiten. Also geht es nicht, eine Hintergrundebene und eine Spielfigurebene zu definieren und dann die Figur unabhängig davon zu bewegen.

Nur verliert nun mal jeder Spielhintergrund an Wirkung, wenn der Spieler bei seiner Wanderung ein Riesenloch hinterläßt. Um etwas gegen diese schwarzen Löcher zu unternehmen, braucht man also eine geeignete Methode, die dann auch noch richtig ausgeführt werden muß. Die Lösung kann eigentlich nur darin bestehen, den Hintergrund wieder neu zu zeichnen. Aber das kostet wieder Zeit – genauso, wie das Zeichnen der Grafik, je komplizierter sie ist, entsprechend mehr Zeit kostet. Und je mehr Zeit das kostet, um so schwieriger ist es, die Illusion, die hinter aller Computeranimation steht, aufrechtzuerhalten. Denn, wenn der Spieler sieht, wie sich ein Zeichen nach dem anderen erst aufbaut, wird es verständlicherweise etwas schwieriger für ihn, sich als Kommandant eines Raumschiffes zu fühlen, von dessen geschickter Verteidigung es abhängt, ob die gute alte Mutter Erde noch ein paar Drehungen vor sich hat oder nicht. Vor allem ist es natürlich so, daß jede außerirdische Invasion entschieden an Schrecken verliert, wenn sich die feindlichen Raumschiffe mit der Geschwindigkeit einer gut ausgebildeten, besonders talentierten Rennschnecke über den Bildschirm bewegen.

Damit wären die meisten, die «nur» BASIC können, auch schon wieder aus dem Rennen, wenn es um tolle Spielideen geht. Einfach weil diese Sprache unter normalen Umständen zu langsam für ein richtig fixes Spiel ist. Und genau in diesem Problembereich setzen die Sprites an. Mit ihnen wird das alles nämlich plötzlich wunderschön leicht. Vorausgesetzt, man versteht, wie die Sprites funktionieren. Genau das sollen Ihnen unsere nächsten Abschnitte vermitteln.

Ab hier können Ihnen bis zu 8 Steine vom Herzen fallen

Warum ausgerechnet 8 Steine? Nun, das liegt daran, daß Sie mit Ihrem Commodore achtmal um so aufwendige Methoden zum Entwickeln eines Computerspiels herumkommen, weil Ihnen nämlich 8 Sprites zur Verfügung stehen, die gleichzeitig auf dem Bildschirm sein können. Und mit einer Kombination aus Sprites, Grafik- und Sonderzeichen müßten Sie eigentlich jede Spielidee verwirklichen können. Wofür sind sie nun aber da, diese freidefinierbaren Figuren, wie schauen sie aus und wie sind sie aufgebaut? Also gut – Sie wollten es ja wissen: Vorhang auf. Die Charaktere sollen vorgestellt werden. Hauptdarsteller und Held dieses Stückes ist: der Sprite.

Er sieht nicht gerade aus wie Clark Gable (sollten Sie ein männlicher Leser sein, setzen Sie statt Clark Gable bitte Marylin Monroe oder so ein), die Figur ist eher etwas untersetzt und besteht aus 24 * 21 Punkten. Eigentlich ist er ein kleiner Hires-Bildschirm, den Sie auf dem Bildschirm bewegen können. Natürlich können Sie auch den großen Bildschirm bewegen. Vorausgesetzt, Sie sind kräftig genug und lassen ihn nicht fallen. Vergessen Sie aber nicht das Wörtchen «frei definierbar», darin liegt's nämlich. Solange Sie ihm kein Leben eingehaucht haben, ist er im wahrsten Sinne des Wortes ein etwas farbloser Geselle. Und wenn Sie ihn auf die Röhre, die für ihn die Welt bedeutet (sprich Ihren Bildschirm), holen, ohne ihn vorher geformt zu haben, dann sehen Sie folglich auch nichts. Deshalb können Sie so ziemlich alles aus ihm machen, was in die 24 * 21 Punkte paßt. Der Sprite ist eine kleine Hires-Grafik, die Sie einfach bewegen können. Bevor wir uns aber jetzt unserem geplanten Drama zuwenden, sollten wir vielleicht noch kurz sagen, wie er auf dem Bildschirm dargestellt wird. Unser kleiner Star besteht aus 63 Bytes. Am besten stellen Sie sich das so vor, wie auf Bild 7.1 zu sehen.

Eine solche Skizze sollten Sie immer in ausreichender Menge greifbar haben, wenn Sie mit Sprites arbeiten. Da kann man sie nämlich am besten entwerfen. Um Ihnen das etwas zu erleichtern, finden Sie im Listinganhang eine Seite, auf der diese Skizze leer dargestellt ist. Am besten, Sie kopieren sich diese Seite entsprechend dem Bedarf. (Übrigens wären wir Ihnen dankbar, wenn das alles bliebe, was Sie kopieren. Alles andere ist geschützt und darf, selbst auszugsweise, nur mit der schriftlichen Genehmigung des Verlages vervielfältigt werden. Ganz egal, wofür. Okay? Na prima.) Wie Sie dann mit dieser Skizze arbeiten sollen, ist wahrscheinlich klar: Jeder Punkt, der erscheint bzw. an ist, sollte hier eingetragen werden. Daraus ergeben sich dann die (wer hätte das gedacht?) zu POKEnden Werte für den Sprite.

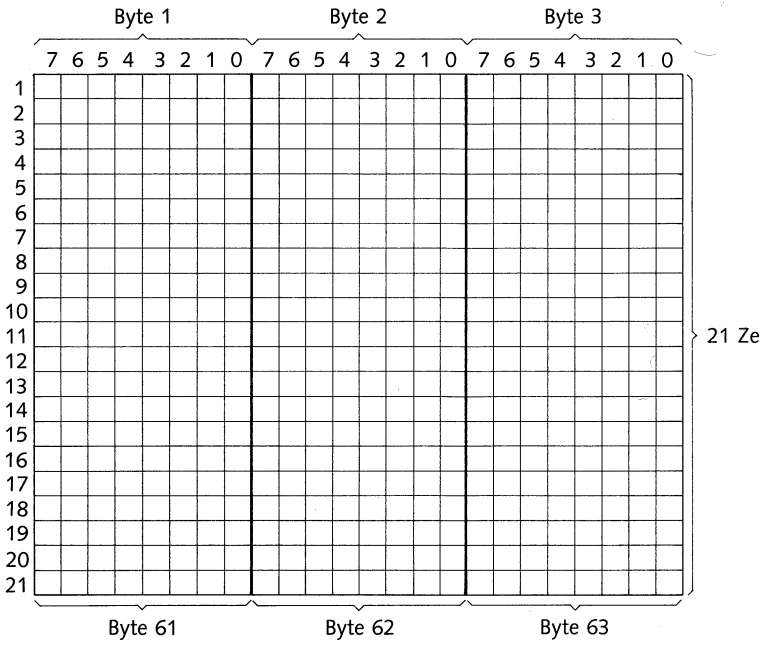


Bild 7.1 Leere Spritematrix

Haben wir erst mal einen Sprite definiert, übernimmt der gute VIC für u all die lästigen Arbeiten des Zeichnens und Löschens. Scheint doch ga brauchbar zu sein, oder?

Natürlich ist klar, daß die 63 Bytes, aus denen ein Sprite besteht, au irgendwo ein Zuhause haben müssen. Denn außer in Frankreich ist es wo nirgends romantisch, ein Obdachloser zu sein. Und weil wir ja die geistig Eltern der Dinge sind, die wir im Speicher unseres Computers aufbau haben wir uns gefälligst auch um geeignete Übernachtungsmöglichkeiten kümmern.

Aber das war jetzt genug an Vorwissen. Das Spiel möge beginnen.

1. Akt: Wie macht man einen Star

Mit Star meinen wir natürlich unseren Sprite. Anhand der Skizze von vorhin und der Punkt-an-/Punkt-aus-Erklärung haben Sie bestimmt schon selbst gemerkt, daß ein Sprite durchaus mit den selbstdefinierten Zeichen zu vergleichen ist. Einer der großen Unterschiede ist allerdings: Wir haben nicht mehr nur 8 * 8, sondern 24 * 21 Punkte zur Definition.

Haben wir einst bei den Sonderzeichen ein kleines Auto (im Zeichen der Zeit wahrscheinlich einen Kleinwagen aus Turin) zusammengebaut, könnten wir ja jetzt die Fertigung auf ein größeres Modell umstellen – vielleicht eines der Prachtstücke aus Stuttgart, Untertürkheim. In jedem Fall soll es ein mustergültiger Luxuswagen werden, wie ihn die Bosse der großen Werbeagenturen immer fahren. Es könnte beispielsweise so aussehen, wie auf Bild 7.2.

Auch hier sind wir, wie im Frankfurter Westend, natürlich erst mal dazu gezwungen, einen geeigneten Parkplatz zu finden. Und da wie dort sind die meisten Plätze schon besetzt. Steht der Platz einmal fest, ist klar, wie wir

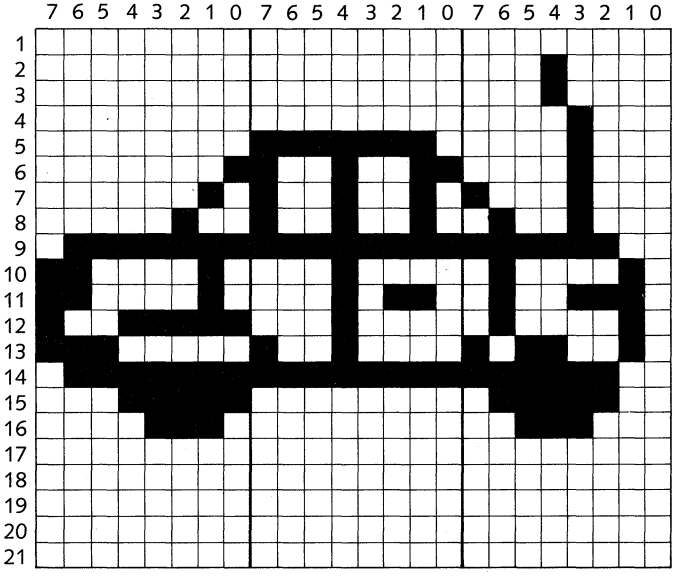


Bild 7.2 Spritematrix «Großes Auto»

dahin kommen: Sie müssen die Bitmuster ausrechnen und byteweise in den Speicher POKEn. Wir raten hier, am besten das Umrechnungsprogramm zu benutzen, das Sie im Listinganhang schon beim Thema Speicheraufteilung fanden. Aber bevor wir einparken, brauchen wir einen Parkplatz.

Wir haben im Laufe des Buches immer wieder darauf hingewiesen, daß VIC nur 16K-Speicher gleichzeitig adressieren kann. Wohin also innerhalb dieses Bereiches mit den Daten für die Sprites? Die Zeropage zu benutzen, wäre ungefähr so, als ob Sie mitten im Halteverbot vor der Polizeiwache parken würden. Ehe Sie sich's richtig versehen, sind Sie schon abgeschleppt worden, und es gibt hinterher Riesenärger. Denn genauso wie die Polizei würde es uns der 6510 sehr übelnehmen, wenn wir seine Autorität nicht berücksichtigen würden.

Aber in der Nähe der Zeropage gibt es tatsächlich ein paar freie Adressen, die wir nutzen könnten. Da für einen Sprite immer 63 Byte gebraucht werden, bieten sich die Unterkunftsmöglichkeiten an, die Sie aus Tabelle 7.1 ersehen können.

Speicherbereich	Spritenummer	Anmerkung
704 bis 766	11	Freie Bytes
832 bis 894	13	Kassettenpuffer
896 bis 958	14	(ist frei, wenn
960 bis 1022	15	Datasette nicht benutzt wird)
2048 bis 4095	32 bis 63	BASIC-RAM
8182 bis 16383	126 bis 255	(Sie können BASIC ja verschieben!)

Tabelle 7.1 Speicherbereich für Sprites

Womit auch unsere Parkplatzsuche gelöst wäre, als ob wir eben ein völlig neu erbautes Parkhaus entdeckt hätten. Die Speicherzellen 4096 bis 8191 sind nicht für Sprites verfügbar. Das hängt wieder einmal damit zusammen, daß unser kleiner VIC belogen und betrogen wird und deshalb denkt, hier sei etwas ganz anderes im Speicher.

Wahrscheinlich werden Sie gerade verwundert die Tabelle oben anschauen und sich fragen, warum wir die ganze Zeit von 8 Sprites reden, aber

Spritenummern von 0 bis 255 angeben. Nein, wir müssen Sie leider enttäuschen. Es bleibt bei den acht Sprites – zumindest bei den acht Sprites, die gleichzeitig auf dem Bildschirm sichtbar sind. Allerdings hindert Sie niemand (nicht einmal der 6510, der sonst eifrigst darauf bedacht ist, kein bißchen Boden an VIC zu verlieren) daran, bis zu 255 Sprites im Speicher zu haben. Sie können jeden Sprite, den Sie im Rahmen des Spieles nicht mehr (oder auch gerade nicht) brauchen, durch einen anderen ersetzen und dann den anderen wieder zurückholen oder ihn wieder durch einen neuen ersetzen – so lange, bis Sie alle 255 durchhaben.

Die nächste Frage, die wir beantworten sollten, ist wahrscheinlich die nach dem Sinn der angegebenen Spritenummern. Die Antwort ist ganz einfach: Ein Sprite braucht eigentlich nur 63 Bytes. Nun hat der Computer ja die Angewohnheit, notgedrungen, alles in Zweierpotenzen darzustellen. 63 ist zwar eine rundum schöne Zahl, aber leider keine Zweierpotenz. Weil unser Commodore aber in dieser Hinsicht kein Pardon kennt, macht er kurzerhand 64 daraus und schenkt sich dann das eine Byte. Schon ist die Welt wieder in Ordnung. (Ein System, das sich auch in der Politik großer Beliebtheit erfreut.)

Aus dieser Rechnerei ergibt sich dann, daß Sprite 0 von 0 bis 62 liegen würde, Nummer 1 von 64 bis 126, Nummer 3 von 128 bis 190 usw. Sprite Nummer 255 liegt demnach von Speicherzelle 16320 bis 16382. Wenn Sie mal die Startadresse eines Sprites suchen (und wir könnten uns vorstellen, daß das nach Ende dieses Kapitels des öfteren der Fall sein wird), dann gilt folgende kleine Formel:

Startadresse = Spritenummer * 64

Womit wir Sie auch über den Sinn der Spritenummern aufgeklärt hätten. (Merken Sie was? Wir kommen ganz schön voran.)

So, wenn wir also unser Auto unter der Spritenummer 13 einparken wollen, steht dem jetzt nichts mehr im Wege. Nach der Formel liegt die Adresse ab 832. Da sich hier der Kassettenpuffer befindet, ist unser Auto auch einigermaßen sicher. Das heißt, es wird nicht mit BASIC zusammenstoßen und ziemlich sicher ohne größere Dellen wieder auf dem Bildschirm erscheinen. Das folgende BASIC-Programm baut unser Auto zusammen:

```
10 FOR X=1TO62:READA:POKE 832+X,A:NEXT X
20 DATA 0,0,0
30 DATA 0,0,16
40 DATA 0,0,16
```

```
50 DATA 0,0,8
60 DATA 0,254,8
70 DATA 1,147,8
80 DATA 2,146,136
90 DATA 4,146,72
100 DATA 127,255,252
110 DATA 194,16,66
120 DATA 194,22,78
130 DATA 159,16,78
140 DATA 224,144,178
150 DATA 127,255,252
160 DATA 31,0,124
170 DATA 14,0,56
180 DATA 0,0,0
190 DATA 0,0,0
200 DATA 0,0,0
210 DATA 0,0,0
220 DATA 0,0,0
```

Na, geschafft? Prima. Ganz hübsche Tipperei, was? Aber das Programm sieht umfangreicher aus, als es ist. Das liegt aber nur daran, daß wir für jede Zeile des Sprites eine neue DATA-Zeile aufgemacht haben. So erschien es uns ein bißchen übersichtlicher. Sie können dann die einzelnen DATA-Werte nachrechnen und überprüfen. Dabei merken Sie auch gleich, wie sicher Sie noch beim Umrechnen von Binär in Dezimal sind. Wenn also für Ihren Geschmack zu viele Zeilen im Programm sind, dann können Sie ruhig immer so viele DATAs zusammennehmen, wie in eine Programmzeile passen.

Jetzt ist der große Moment gekommen: Unser mühsam erarbeiteter Star soll die Bühne betreten. Wir haben ihn gemacht, ihn zu seiner Vollendung geführt.

Wir starten also das Programm mit RUN.

Wir sehen, daß wir nichts sehen. Geht auch nicht. Denn jetzt folgt erst der zweite Akt. Unser Star ist zwar jetzt da (das hat das RUN bewirkt), aber er kann noch nicht die Bühne betreten. Es fehlt noch etwas Entscheidendes.

2. Akt: Wie bringt man einen Star auf die Bühne?

Wenn Sie vorhin RUN eingegeben haben, ist Ihnen vielleicht aufgefallen, daß es einen kurzen Augenblick dauerte, bevor die READY-Meldung erschien. Daraus ergibt sich natürlich die naheliegende Vermutung, daß unser Commodore in dieser Zeit nicht etwa dasaß und Däumchen drehte, sondern offensichtlich mit irgend etwas Ernsthafterem beschäftigt war – womit, können Sie sich eigentlich denken: Er hat unseren Star zusammengebastelt und dann irgendwo hinter dem Vorhang abgestellt. Und das ist auch der Grund, warum wir ihn noch nicht sehen konnten. Denn in diesem Speicherbereich, wo er jetzt ist, würde er bis in die Puppen bleiben, wenn wir ihm nicht beibringen, wie so ein richtiger Star wie ein Sprite auf die Bühne zu kommen hat. Dazu bereitet man am besten zuerst die Bühne vor. Das heißt für uns, wir setzen die *Register* von VIC. Da ist zunächst einmal die Adresse 53269. Sie ist dafür zuständig, welche Sprites gerade an- und welche gerade ausgeschaltet sind. VIC kann bis zu 8 Sprites gleichzeitig darstellen. Also entspricht jedes Bit in der Speicherzelle 53269 genau einem Sprite.

53269 = Sprite 7 6 5 4 3 2 1 0

Wenn ein Bit gesetzt ist, ist der entsprechende Sprite eingeschaltet. Das heißt, unser Star kriegt in der Garderobe die Meldung: «Sprite Nummer 5 fertig zum Auftritt.» Mit folgendem Befehl können Sie Sprites einschalten:

POKE 53269,PEEK (53269) OR2∧SPRITE

Wobei SPRITE eine Zahl von 0 bis 7 ist. Um unser Auto einzuschalten, reicht – weil noch kein anderer Sprite eingeschaltet ist, der sonst aus Versehen gelöscht werden könnte – eine etwas einfachere Version:

POKE 53269,1

Damit wird Sprite 1 eingeschaltet. Warum das untere leichter als das obere aussieht, liegt größtenteils an einem Herrn mit Namen Boole. Sollte Ihnen das kein Begriff sein, haben Sie wahrscheinlich unser Zwischenspiel 3 überschlagen, wo wir den Versuch unternommen haben, die mathematischen Künste von Herrn Boole dem Publikum etwas näherzubringen. Sie können gern noch mal zurückblättern! Ausschalten können Sie das Ganze übrigens folgendermaßen:

POKE 53269,PEEK(53269) AND (255−2∧SPRITE)

Auch hier ist SPRITE wieder ein Wert zwischen 0 und 7.

Sie sollten das aber nicht unbedingt gleich ausprobieren!

Wir gehen also weiterhin davon aus, daß Sie Ihren Sprite eingeschaltet haben und daß Sie immer noch nichts von ihm sehen können. Das ist auch ganz richtig so. Als nächstes will VIC nämlich wissen, ab wo er Ihren Sprite lesen soll. Die Zeiger, die ihm das sagen, liegen recht raffiniert. Sie erinnern sich ja vielleicht daran, daß das Bildschirm-RAM genau 1K groß ist. 1K ist gleich 1024 Zeichen. Aber ein voller Bildschirm faßt nur 1000 Zeichen. Womit 24 Bytes übrigbleiben, die geschickterweise auch noch genau in diesem von VIC adressierbaren Bereich liegen. Ab Adresse 2040 liegen also die Künstlergarderoben. Ab hier kann VIC die einzelnen Sprites aufrufen. Der Zeiger für einen ganz bestimmten Sprite kann mit der Formel:

Zeiger = 2040 + Sprite

errechnet werden. Für unser Auto brauchen wir die erste Adresse, also 2040. Unser Sprite hatte die Nummer 13, liegt also im Speicher ab Adresse 832. (Beachten Sie dazu auch die Tabelle 7.1.) Deshalb ist unsere Eingabe:

```
POKE 2040,13
```

Womit der Zeiger auf die richtige Adresse gesetzt wäre.

So, jetzt haben wir es aber bald geschafft. Wir sind nämlich bei der letzten Information angelangt, die VIC braucht, um unseren Star auf seinen Auftritt vorzubereiten. Und zwar ist das die Stelle, an der er erscheinen soll. Probieren Sie am besten mal die Gegend von 100 bis 150 auf dem Bildschirm:

```
POKE 53248,100:POKE 53249,150
```

Und jetzt ist es soweit.

3. Akt: A Star is born

Jetzt müßten Sie ein weißes Auto auf dem Schirm sehen. Wenn nicht, prüfen Sie noch einmal genau nach, ob Sie auch alles so gemacht haben, wie Sie es gelesen haben. Wenn das alles stimmt, aber Sie trotzdem nichts erkennen können, dann sollten Sie ernsthaft anfangen, sich Gedanken über Ihren Commodore oder Ihren Optiker zu machen – allerdings nur unter der Voraussetzung, daß Ihr Monitor eingeschaltet war ...

Wenn jetzt alles stimmt, können wir uns dem nächsten Teil zuwenden: der bühnengerechten Bewegung unseres Stars. Stellen wir uns mal vor, wir möchten gern, daß unser Star (momentan ist er ja ein Auto) in irgendeine Richtung losfährt.

Sollten Sie sich übrigens nicht daran gewöhnen können, daß wir die ganze Zeit von Star reden, aber eigentlich ein Auto meinen, dann denken Sie mal an Walt Disney's verrückten Käfer Herbie. Der war ja auch ein Star unter den Autos.

Und bei Sprites ist noch etwas zu bemerken. Es gibt nur wenige wirklich gute Stars, bei uns genau 8 Stück. Und diese Stars werden immer so geschminkt, daß sie praktisch jede Rolle spielen können. Was wir damit meinen, ist einfach, daß der Sprite in Ihrem Computer tatsächlich immer einer der acht vorhandenen ist. Er transportiert nur eben mal das eine, dann das andere Bitmuster über Ihren Schirm. Gut. Zurück zum Thema. Folgende kleine Zeile macht unser Auto mobil:

```
10 FOR X=1TO255:POKE53248,X:NEXT
```

Wenn Sie das Programm starten, werden Sie auch schnell sehen, mit welchen einfachen Mitteln man durch Sprites bereits sehr eindrucksvolle Effekte erzielen kann. Und was lernen wir aus der letzten Programmzeile? Richtig, die Speicherzelle 53248 scheint für die horizontale Bewegung zuständig zu sein. Wenn Sie diese Zahl in Ihrem Programm durch die nächsthöhere ersetzen (also 53249), sehen Sie, daß sich die Sache sofort ändert. Plötzlich geht es nicht mehr von links nach rechts, sondern von oben nach unten.

Sollte Ihr Auto plötzlich vom Schirm verschwunden sein, dann ist das ganz normal. Mit

```
POKE 53249,150
```

holen Sie es wieder zurück.

Noch etwas: Sobald Sie ⟨RUN/STOP⟩ + ⟨RESTORE⟩ drücken, schicken Sie Ihren Sprite wieder in die Künstlergarderobe. Das heißt, er wird dadurch ausgeschaltet.

Um ihn dann wieder herzuholen, müssen Sie die Register von VIC wie gerade beschrieben neu setzen.

Jetzt sollten wir Ihnen aber vielleicht die Bühne noch ein bißchen besser erklären. Wie ist das jetzt ganz genau mit diesen Bewegungskoordinaten?

Sollten wir es noch nicht erwähnt haben: Ihr Sprite ist eine kleine Hires-Grafik aus 24 * 21 Punkten. Unser ganzer Hires-Schirm besteht aber aus 320 * 200 Punkten. Mit den POKEs 53248 für die X- und 53249 für die Y-

Koordinaten können Sie den Sprite auf Ihren Bildschirm bringen, und zwar aufs i-Pünktchen genau. Allerdings entsprechen die Werte, die gePOKEd werden, nicht genau den Bildschirmkoordinaten. Das wäre ja auch wirklich zu einfach gewesen.

Das heißt also, daß 0,0 nicht genau die linke obere Ecke ist, sondern für den Sprite bereits «außerhalb» des Bildschirms. Tatsächlich entsprechen die Werte 24 und 50 der Grundposition. Mit kleineren Zahlen können Sie Sprites auch über den Rand hinaus darstellen. Und damit Ihr Fernseher von diesen Versuchen keine Dellen kriegt, ist man bei Commodore dazu übergegangen, den Sprite dann einfach «verschwinden» zu lassen. Das heißt, Sie sehen nur noch so viel von ihm, wie die Koordinaten, seine Größe und der Bildschirmrand es zulassen.

Der untere Rand entspricht bei alledem übrigens einem Y-Wert von 229. Aber wie auch immer Sie Ihren Sprite setzen wollen, ein kleines Problem bleibt.

Vielleicht können Sie es sich schon denken. Wenn der Bildschirm eine horizontale Auflösung von 320 Punkten hat und eine Speicherzelle für diese Position zuständig sein soll, dann werden Sie auf Schwierigkeiten bei der Koordinatenangabe stoßen. Na, wissen Sie jetzt, was wir meinen?

Die größte Zahl, die ein Byte darstellen kann, ist ja bekanntlich 255, weil nämlich 8 Bits «an» (also maximal 8mal eine 1 in einem Byte) umgerechnet nur 255 ergeben kann, nicht mehr und nicht weniger. Was aber ist mit den Koordinaten, die darüber liegen?

Auch hier gibt es einen Trick. Allerdings gilt für ihn genau das, was auch für die meisten anderen Tricks zutrifft: Er macht alles etwas komplizierter.

Na, wir wollen mal versuchen, es zu erklären.

Im Grunde bleibt alles beim alten. Maximum für ein Byte sind 8 Bit. Deshalb muß das 9. Bit ja auch in eine eigene Speicherzelle. Und wenn das 9. Bit unserer Speicherzelle, die für einen bestimmten Sprite verantwortlich ist, «an» ist, dann weiß der Computer: «Aha, ich muß nicht ganz vorne, sondern bei 255 zu zählen anfangen.»

Das 9. Bit wirkt also so ähnlich wie die ⟨SHIFT⟩-Taste an der Tastatur. Wenn Sie ⟨SHIFT⟩ und ⟨A⟩ gleichzeitig drücken, dann ergibt sich daraus ein anderer ASCII-Wert, als wenn Sie nur ⟨A⟩ drücken. Sie können es aber auch mit dem High-Byte/Low-Byte-Prinzip vergleichen, das wir Ihnen an anderer Stelle erklärt haben. Jetzt ist natürlich noch interessant, wo dieses 9. Bit eigentlich ist. Die Speicherzelle ist 53264.

Adresse	X-Koordinate	Sprite-Nr.
53248	X-Koordinate	0
53250	X-Koordinate	1
53252	X-Koordinate	2
53254	X-Koordinate	3
53256	X-Koordinate	4
53258	X-Koordinate	5
53260	X-Koordinate	6
53262	X-Koordinate	7
53264	9. Bit der X-Werte für alle Sprites	

Adresse	Y-Koordinate	Sprite-Nr.
53249	Y-Koordinate	0
53251	Y-Koordinate	1
53253	Y-Koordinate	2
53255	Y-Koordinate	3
53257	Y-Koordinate	4
53259	Y-Koordinate	5
53261	Y-Koordinate	6
53263	Y-Koordinate	7

Tabelle 7.2 Spritekoordinaten

Mit

```
POKE 53264,1
```

verschieben Sie deshalb die X-Koordinaten um 255 Positionen nach rechts. Und nun können Sie ab dieser Stelle normale X-Werte POKEn. Um aber bei dieser ganzen Operation auch noch Platz zu sparen, wurden alle 9er-Bits aller Sprites in einem Byte zusammengefaßt. Das spricht mal wieder für die Findigkeit der Commodore-Leute.

Speicherzelle 53264:

9. Bit der X-Koordinaten von Sprite 7 6 5 4 3 2 1 0

Um unser Auto also über den ganzen Bildschirm zu bewegen, wäre folgendes Programm nötig:

```
5 POKE 53269,1:POKE 53249,100
10 POKE 53264,0
20 FOR I=0 TO 350
30 X=I
40 IF I>255 THEN X=I-255:POKE 53264,1
50 POKE 53248,X
60 NEXT I
```

Die Koordinaten der anderen 7 Sprites liegen hinter denen des 1. Sprites. In Tabelle 7.2 noch einmal eine kleine Übersicht über die Speicherzellen, die für unsere Sprites wichtig sind.
So, jetzt dürften Sie eigentlich keine Probleme mehr haben, mit Sprites auf dem Bildschirm zu spielen. Und weil das jetzt so schön war, schauen wir uns die anderen Funktionen auch gleich an. Das gehört jetzt aber schon zum nächsten Akt unter das Thema Kosmetik und Maske.

4. Akt: Der Star und sein Kostüm

Jeder Sprite hat beispielsweise eine eigene Farbadresse. So ist es kein Problem, zum Beispiel aus zwei erst mal äußerlich völlig gleichen Sprites bei einem Spiel ein grünes (gutes) Waldmännchen und ein rotes (böses) Feuermännchen zu machen. Diese Farbadressen finden Sie übrigens ab der Adresse 53287. Damit Sie wissen, wohin Sie Ihre Farbpost schicken sollen: Mit einem

POKE53287 + (Sprite 0–7),Farbnummer

können Sie sich alles farblich so zusammenstellen, wie es Ihnen gefällt.

Wenn wir beispielsweise unseren Straßenkreuzer geklaut hätten, täten wir gut daran, ihn möglichst bald umzulackieren. War er vorher weiß, so macht ein

POKE 53287,10

ihn rot. (Aber nicht etwa vor Scham, wegen der Klauerei ...)

Übrigens: Da wir gerade beim Auto-Tuning sind, versuchen Sie doch gerade mal

POKE 53271,1

Wie Sie sehen, können Sprites horizontal vergrößert werden. Sollten Sie

nichts sehen, könnte es daran liegen, daß Ihr Commodore defekt ist – oder daß Sie vielleicht gar nicht mittippen?

Mit einem

```
POKE 53277,1
```

geht dasselbe in vertikaler Richtung.

Sprites können also in einer oder zwei Richtungen gleichzeitig vergrößert werden.

Diese Geschichte funktioniert natürlich auch bei dem Rest der Bande. Nur weil diesmal die zuständigen Bits etwas weiter hinten im Byte liegen, müssen wir wieder einmal unseren Meister Boole mit ins Boot ziehen:

```
POKE 53271,PEEK(53271) OR 2∧(Sprite)
POKE 53277,PEEK(53277) OR 2∧(Sprite)
```

Nur noch mal zur Erinnerung: Der erste POKE vergrößert horizontal, der zweite vertikal.

Und diese beiden Funktionen lassen sich auch für jeden Sprite getrennt wieder ausschalten:

```
POKE53271,PEEK(53271) AND (255−2∧Sprite)
POKE53277,PEEK(53277) AND (255−2∧Sprite)
```

Auch hier wieder: POKE eins für das Ausschalten der horizontalen Vergrößerung, zwei für die vertikale.

Langsam nähern wir uns dem Höhepunkt unseres Dramas.

Wie Ihnen wahrscheinlich beim bisherigen Experimentieren mit den Sprites aufgefallen sein dürfte, werden die Sprites über dem Text auf dem Bildschirm dargestellt. Und wie das bei echten Stars so üblich ist, scheren sie sich auch keinen Deut darum, was um sie herum passiert.

So erklärt sich auch, daß ein Sprite auf dem Bildschirm nicht vom *Scrolling*, also vom Rollen des Textes auf dem Bildschirm, betroffen wird. Diese stoische Ruhe eines oder mehrerer Sprites ist bei allerhand Spielen sicherlich sehr praktisch.

Aber wenn es Ihnen Spaß macht (oder Ihre Spielidee es verlangt), kann unser Star auch mal ein bißchen zurückstecken. Mit

```
POKE 53275,1
```

bringen Sie ihn dazu, daß er auch mal mit einer Rolle als Statist hinter dem Text oder irgendwelchen Grafik- und Sonderzeichen zufrieden ist.

Da dies für jeden Sprite einzeln einstellbar ist, sind sogar mit einiger Experimentierfreude dreidimensionale Darstellungen möglich. Einschalten kann man mit

```
POKE 53275,PEEK(53275) OR 2∧(Sprite)
```

Ausschalten läßt sich's mit

```
POKE 53275,PEEK(53275) AND (255−2∧Sprite)
```

Sollten Sie jetzt mehrere Sprites auf dem Bildschirm haben, wäre es vielleicht nicht schlecht, wenn Sie etwas mehr Überblick hätten. Zumindest sollten Sie merken, wenn es irgendwo kracht – also, wenn zum Beispiel ein Sprite in die Kulissen rennt, weil er blind wie ein Maulwurf ist (oder der Spieler zu langsam reagiert?), oder ob zwei Sprites ineinanderdonnern. VIC agiert in solchen Fällen wie ein guter Polizist. Er geht davon aus, das Sie als Vater Staat die Augen nicht überall haben können, und deshalb paßt er auf.

Mittels eines sogenannten Kollisionsdetektors kann VIC feststellen, ob sich da Sprites zueinander oder auch zum Hintergrund besonders hingezogen fühlen.

Das ist natürlich bei selbstprogrammierten Spielen sehr nützlich. Hier gilt dann für alle Mitspieler die Devise: Achtung! Big VIC is watching you.

Dieser Kollisionsdetektor liegt auf der Adresse 53278. Und je nachdem, wer da mit wem, ändert sich der Wert dieser Speicherzelle bzw. wieder die einzelnen Bits der Zelle. Das heißt mit anderen Worten, daß nach einer Kollision alle die Bits in 53278 gesetzt (bzw. an) sind, die den zusammengestoßenen Sprites entsprechen. Und diese Bits bleiben auch so lange gesetzt, bis sie vom richtigen Mann ausgelesen wurden, also die Zelle mit

```
PRINT PEEK (53278)
```

aufgerufen und kontrolliert wurde. (Erinnert doch irgendwie an das Vorgehen dieser Punktesammlung in Flensburg, oder?)

Man kann sie aber auch einfach durch

```
POKE 53278,0
```

zurücksetzen.

Das sollte man am Anfang eines Programms, das diesen Detektor benutzt, übrigens sowieso tun, um sicherzugehen, daß alles mit rechten Dingen zugeht. Sonst kriegen Sie womöglich die Punkte vom Mitspieler ab. Übrigens: Denken Sie daran, daß unsere kleinen Stars natürlich auch hinter der Bühne

zusammenstoßen können. (Die Gänge zu den Garderoben sind aber auch wirklich schlecht beleuchtet . . .)

Die Adresse 53279 ist für die Kollision mit Zeichen bzw. einer Hires-Grafik zuständig. Sie wird ebenfalls mit dem Befehl PEEK ausgelesen und kann genauso mit

POKE 53279,0

zurückgesetzt werden.

Hier bedeutet ein bestimmtes gesetztes Bit, daß der dazugehörige Sprite irgendwo einen Unfall mit dem Hintergrund hatte.

Hier ein kleines Programm zur Verdeutlichung.

```
10 POKE2040,13: POKE2041,13: POKE53269,3: POKE 53277,3: POKE
   53281,0
20 X0=30:X1=120
30 POKE 53248,X0: POKE 53250,X1: POKE53249,150: POKE 53251,150:
   POKE53278,0
40 FOR I=0TO200
50 X0=X0+1.4:X1=X1+1
60 POKE 53248,X0:POKE 53250,X1
70 IF PEEK(53278)〈〉0 THEN FOR X=0TO34:POKE 53288,X:NEXT:END
80 NEXT I
```

Zuerst wird dieses Programm unsere beiden Autos auf den Schirm bringen. Dann fahren beide los. Nur das eine ist etwas schneller (siehe Zeile 50), und so kommt, was kommen muß. Es fährt auf seinen Vordermann auf. Besonders aufmerksam machen wollen wir Sie nur auf die Zeilen 30 und 70. In der ersteren wird das Kollisionsregister zurückgesetzt, in Zeile 70 wird das Register abgefragt. Wenn es einen Wert enthält, der nicht 0 ist, dann reagiert der Computer. Was all die anderen Zeilen tun, diese Frage geben wir galant an Sie zurück. Es sind alles Ihnen mittlerweile bekannte POKEs. Sie müssen sich das Programm nur einmal aufmerksam durchlesen. Das ist eine gute Übung. Jetzt ganz zum Schluß wollen wir noch etwas mehr Farbe in die ganze Sache bringen. Wir haben ja bereits, als wir im Hires-Kapitel über Farbe sprachen, vom Multicolormodus gesprochen. Da wurde ja unter anderem gesagt, daß dieser Modus auch bei den Sprites funktioniert. Und weil wir nie vergessen, was wir einmal gesagt haben, wollen wir das jetzt auch erklären.

Was wollten wir doch gleich erklären?

Ach ja. Der Multicolormodus bietet die Möglichkeit, bis zu 3 Farben in

einem Sprite darzustellen. Dafür bleibt aber nur die halbe Auflösung. Wenn Sie das schon bei der Konzeption Ihres Sprites berücksichtigen, lassen sich damit auch sehr hübsche Effekte erzielen. Wenn Sie sich daran nicht mehr so recht erinnern können, sollten Sie noch mal das Hires-Kapitel aufschlagen.

Bei der Frage, welche Farbe dargestellt wird, liefern wieder einmal Bitkombinationen die Antwort (Tabelle 7.3).

Bitkombination	Farbe	Adresse
00	Hintergrund	53281
01	Sprite Multic. Nr. 0	53285
10	Sprite Farbe	53287 + (Sprite)
11	Sprite Multic. Nr. 1	53286

Tabelle 7.3 Farbadressen für Multicolorsprites

Der Ausdruck «(Sprite)» in der Tabelle steht natürlich wieder für eine Zahl von 0 bis 7.

Wie Sie sehen, müssen die Multicolorfarben für alle Sprites zusammen festgelegt werden. Dazu dienen die Register 53285 und 53286.

Die Spritefarbe selbst bleibt, wie gehabt, individuell.

Der Multicolormodus kann aber gezielt für einzelne Sprites eingeschaltet werden. Und zwar mit

```
POKE53276,PEEK(53276) OR 2∧(Sprite)
```

Ausschalten können Sie genauso mit

```
POKE5327,PEEK(53276) AND (255−2∧Sprite)
```

Tja, das war dann auch schon unser kleines Star-Lexikon auf dem Commodore 64.

Alles weitere, was es hier noch zu zeigen gäbe, können wir jetzt getrost Ihrer Fantasie überlassen. Probieren Sie also ruhig erst mal ein bißchen, bevor Sie das nächste Kapitel aufschlagen. Da geht es dann um Töne und Geräusche auf dem Commodore 64. Danach kommt ein Spiel mit Sprites und vielem mehr.

Wie immer beim Commodore 64, wird auch die Freude an den Sprites durch die Unmengen an POKEs etwas getrübt. Aber tragen Sie es mit Fassung. Sollten Sie noch etwas Übung brauchen, um sich Zahlen zu merken, dann üben Sie ruhig das erst noch ein bißchen, zum Beispiel, indem Sie das Frankfurter Telefonbuch auswendig lernen. Na ja, wenn Sie wollen, können Sie auch das Ihres eigenen Ortes nehmen – aber nur, wenn Sie mindestens 150 Anschlüsse haben. Und denken Sie zum Trost auch daran, daß es einen Anhang gibt, wo die wichtigsten PEEKs und POKEs zusammengefaßt sind. Also nicht verzweifeln. Wir lassen Sie jetzt erst mal mit Ihren neuerworbenen Kenntnissen etwas allein. Wir haben nämlich noch zu tun. Das Frankfurter Telefonbuch ist dick, und wir sind gerade erst beim zweiten Drittel «Müller» angelangt.

Bis später also. Und viel Spaß!

Ende der Vorstellung. Der Vorhang fällt. Applaus?

8
Töne und Geräusche auf dem Commodore 64

Der Ton macht die Musik

Wenn man sein erstes selbstgeschriebenes Spielchen auf dem Commodore laufen hat, überflutet einen schon schnell der Vaterstolz. Trotzdem, wenn man dann zwecks Information mal so einen dieser Spielautomaten bewundert, vielleicht sogar eine Studienreise in die nächstgelegene Spielhalle unternimmt, stellt man doch fest, daß dem eigenen Spiel noch irgend etwas an Faszination fehlt. Die Raumschiffe können es nicht sein. Die hat man ja auch. Die Farbe? Nein, die ist auch da. Was aber macht Donkey Kong oder Pac-Man oder Frogger oder ... egal, was macht sie alle so stark? Bevor Sie jetzt der Idee verfallen, an die Seite Ihres Commodore einen kleinen Kasten zu bauen, der immer dann Strom gibt, wenn einer eine Mark hineinwirft, und sich damit heillos in die Welt der Hardware verrennen, lesen Sie dieses Kapitel. Wir können Ihnen versichern, daß der Vorteil der anderen Spiele nicht etwa das Geldhineinwerfen ist, sondern das, was Sie vielleicht bei Ihrem ersten Spielhöllenbesuch eher als lästig empfunden haben: der Krach, den sie verursachen. Denn jede Invasion wird erst schön, wenn der Feind auch so richtig schöne Invasionsgeräusche von sich gibt. Und da die Leute, die an solchen Apparaten stehen, meist noch keine Invasion mitgemacht haben – weder eine irdische, geschweige denn eine außerirdische –, fällt ihnen zumeist auch nicht auf, ob die Geräusche sehr lebensecht klingen. Und wenn die Geräuschexperten der großen Spielehersteller gar nicht mehr weiterwissen, dann versuchen sie es mit Musik.

Mit welchen Geräuschen sollte man auch um Himmels willen Frogger unterlegen? Dieses Spiel vollzieht in erster Linie das Leben eines Frosches in der Nähe der B 27 nach. Diese Straße hat für das Leben eines Frosches einen

entscheidenden Nachteil: nämlich den, daß Autos darauf fahren. Und weil nun unser Frosch ausgerechnet hier sein Quartier aufschlagen mußte, steht er bei seinem täglichen Wunsch nach einem Bad vor einem rund 8 Meter breiten Problem. Denn geschickterweise fiel den Spielstrategen gerade noch rechtzeitig ein, daß unser Frosch sozusagen «drüben» wohnt. Will er also auf die andere Seite (zum Fluß), dann muß er über die Straße. So weit, so gut. Nur stellen Sie sich mal vor, Sie müßten jeden Morgen auf dem Weg in Ihr Badezimmer eine Bundesstraße überqueren – für jeden, der morgens noch halb schlafend ins Bad tappt, eine großartige Chance, seine Lebensversicherung bereits sehr früh an Verwandte verteilen zu können. Unserem Frosch ergeht es nicht besser. Wenn er flink genug ist – prima. Nur, wenn nicht, ereilt ihn ein schnelles Schicksal in Form von mit wohlklingenden Namen bedachten Reifenfabrikaten. Wie aber drückt man das in Geräuschen aus? Man entschied sich in erster Linie für Musik und ein neutrales Tröten, wenn ein Frosch die ganze Sache nicht so hinkriegt. Nachdem diese Wohnungen neben der B 27 offensichtlich sehr günstig, nicht spekulationsgefährdet sind und vielleicht sogar von der Bundesregierung subventioniert werden, kommen immer neue Frösche. Und so gibt es jeden Abend in Deutschlands Wohnzimmern eine fröhliche Froschhatz. Wohlgemerkt: Es ist die Computerindustrie, die zum Halali bläst. Denn Deutschlands Kinder, Väter und manchmal auch Mütter sind der arme kleine Frosch. Womit man deutlich sieht, daß auch auf dem Videoschirm der Kampf der Grünen gegen die Industrie andauert.

Soweit zum Geleit. Geräusche können in jedem Fall, wenn sie etwas fantasievoll eingesetzt werden, aus einem mittleren Spiel ein passables machen, und bei einem sehr guten Spiel sind sie das Tüpfelchen auf dem i – deshalb auch dieses Kapitel. Die lange Vorrede war eigentlich nur dazu da, Ihnen klarzumachen, daß Geräusche – genauso wie manche Spiele an sich – sehr vom Geschmack abhängig sind. Vielleicht hätten Sie Frogger ganz anders vertont. Vielleicht hätten Sie Frogger auch nie geschrieben – zum Beispiel, weil Sie Mitglied im örtlichen Tierschutzverein sind. Wie auch immer: Der eine findet jenes Geräusch gut, der andere sagt, es sei schlicht und ergreifend schwachsinnig. Weil wir uns aus dieser ganzen Streiterei etwas raushalten wollten, beschlossen wir, uns gerade in diesem Kapitel auf unsere hauptsächliche Aufgabe zu besinnen und Ihnen in erster Linie zu erklären, wie man es anstellt, dem Commodore mehr als ein dünnes Piepsen zu entlocken. Was Sie dann mit Ihrem Wissen wiederum anstellen, überlassen wir Ihnen. Aus alledem resultieren zwei Dinge: Zum einen, daß in diesem Kapitel in

erster Linie von bestimmten Speicherzellen die Rede sein wird, die wir erklären. Das heißt im Grunde, daß das ganze Kapitel Ihnen im Vergleich zu den anderen eher etwas theoretisch vorkommen wird. Das ist leider auch so. Nur haben wir keinen anderen Weg gefunden, um Töne und Geräusche zu erklären.

Natürlich werden wir Ihnen sagen, wie man das alles im Computer am besten zum Arbeiten bringt. Aber praktische Beispiele werden aus den dargestellten Gründen etwas spärlich sein.

Zum anderen heißt das aber für Sie, daß Sie gerade hier noch mehr auf das Selbstprobieren angewiesen sind. Und das wiederum ist durchaus nicht schlecht. Insgesamt empfehlen der Herr Doktor, dieses Kapitel erst einmal durchzulesen. Wenn dann die Verständniskurve etwas angestiegen ist, könnten Sie Ihren Commodore konsultieren. Bliebe nur noch eines: anzufangen.

Der kleine Schwarze mit dem lauten Ton

Mittlerweile kennen wir so ziemlich den ganzen Hofklüngel, der sich so um unseren 6510 herumschart. Der 6510 selbst, die RAMs und ROMs, den guten VIC. Jetzt wollen wir einen neuen Vertreter kennenlernen. Während die RAMs und ROMs höchstens von sich behaupten könnten, sie seien Hoflieferanten, ist unser Neuer so eine Art Pressesprecher der Regierung. Deshalb hat er auch eine besonders laute, fein zu nuancierende Stimme, die kraftvoll den Raum durchdringt. Meist so kraftvoll, daß bei den ersten Versuchen mit dem Tonchip des Commodore das ganze Haus zusammengelaufen kommt, um die vermeintliche Katze aus Ihren ebenso vermeintlich brutalen Händen zu entreißen oder um sich ein Arbeitszimmer nach einer Explosion anzusehen oder zu welchen Vermutungen auch immer das verursachte Geräusch Grund gab. Diesen Effekt können Sie übrigens verstärken, indem Sie Ihren Commodore an die Stereoanlage anschließen. Das geht mit einem fünfpoligen DIN-Stecker. Näheres dazu finden Sie in Ihrem Commodore-Handbuch. Wenn Sie das gemacht haben, können Sie damit auch prima die zu verwendenden Geräusche ausprobieren. Wenn Sie beispielsweise an einer Gasexplosion herumbasteln, den Ton entsprechend laut gedreht haben und das Testprogramm laufen lassen, können Sie leicht an der Reaktion der Umwelt erkennen, ob Sie das Geräusch halbwegs authentisch hinbekommen haben – je nachdem, ob die Mitbewohner mit Beilen, Feuerlöschern oder einem Krankenpfleger vor der Tür stehen ...

Aber wenn Sie das alles nicht wollen, dann sollten Sie Ihren Fernseher doch besser während des Probierens leiser stellen, vielleicht sogar noch die Tür zumachen.

Noch einmal kurz zu unserem Pressesprecher. Sein Name ist SID. Und weil es so schön romantisch ist, hat er auch eine Nummer gekriegt: SID 6581. Da wir aber gute Bekannte werden wollen, wollen wir uns beim Vornamen nennen. Deshalb sagen wir einfach SID. Das steht für «Sound Interface Device». Wenn man das so hört, staunt man doch immer wieder über die fantasievollen Ausdrücke, die die Computerindustrie so geprägt hat. Unser Synthesizer-Chip ist nun tatsächlich sehr leistungsfähig. Er hat drei voneinander unabhängige Stimmen. Und für jede dieser Stimmen läßt sich eine eigene ADSR-Hüllkurve programmieren. Was das genau heißt, kommt noch. Grundsätzlich unterscheidet man bei diesen Schwingungen vier verschiedene Arten: Dreieck, Rechteck, Sägezahn und Rauschen. Diese Ausdrücke kommen in erster Linie vom Aussehen der zugehörigen Töne in einem Diagramm. Dazu gibt es noch allerhand Filter und natürlich einen softwaremäßigen Lautstärkeregler.

Und was muß man tun, um diesem scheinbar so ausgeklügelten System einen Ton zu entlocken? Nun, dasselbe, was man so oft bei Commodore muß: POKEn. Und das heißt wieder einmal, sich Adressen und die richtigen Werte dazu zu merken. Deshalb wollen wir gleich wieder ganz sachlich werden.

Die Startadresse unseres Soundchips liegt bei 54272. Das erinnert Sie vielleicht irgendwie an die Kapitel über Speicheraufteilung und Sonderzeichendefinition. Stimmt. Diese Adresse liegt in den I/O-Registern. Wenn Sie die Kenntnisse von damals noch mal auffrischen wollen, dann können Sie das jetzt tun. Wenn nicht, machen wir einfach weiter.

Unser SID hat verschiedene *Register* – so ähnlich, wie das auch bei VIC der Fall war. Diese Register sind verantwortlich für alle seine Funktionen und Fähigkeiten, die er ausführt. Bevor wir auf sie im einzelnen eingehen, sollten Sie folgendes wissen: Die Töne und Geräusche, die wir hören, sind lediglich Schwingungen in der Luft, die in unserem Ohr zu verständlichen Signalen umgewandelt werden. (Wenn sie unverständlich sind, kann es daran liegen, daß jemand eine andere Sprache spricht, daß er undeutlich spricht, daß er beides zusammen tut oder daß er Politiker ist.)

Für diese Schwingungen in der Luft gibt es bestimmte charakteristische Bezeichnungen. Wir meinen hier aber nicht so landläufige Ausdrücke wie Geschwafel, sondern wir meinen mit Toncharakter eigentlich eher, wie

jemand diese oder jene Eigenschaft eines Tones hört. Deshalb nochmals unsere Bitte, mitzuprobieren. Nur so können Sie hören, was sich an einem Ton ändert oder welchen Charakter er im Laufe der Änderungen annimmt. Denn wir müssen zu unserer Schande gestehen, daß es uns sehr schwerfällt, einen Ton zu beschreiben.

Die Art der Wellen, die unser Ohr aufnimmt, kann sich stark unterscheiden. Ihr Soundchip kann vier verschiedene Wellenformen erzeugen; zum ersten die Sägezahnschwingung, die steigt und dann sofort wieder abfällt. Ein Diagramm einer solchen Schwingung würde etwa so aussehen.

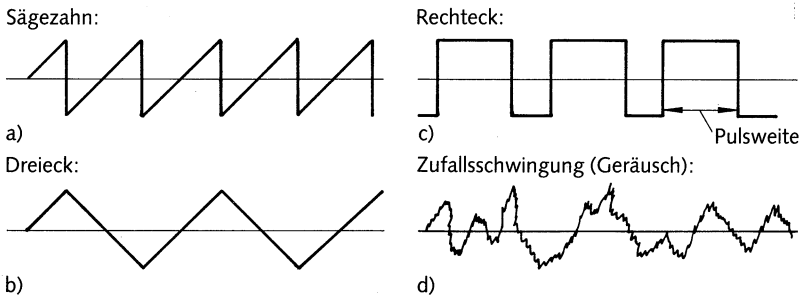


Bild 8.1 Schwingungsdiagramme

Etwas flüssiger im Verhältnis von Anstieg zu Abstieg und nicht so abrupt ist die Dreieckschwingung.
Dann gibt es noch die Rechteckschwingung. Bei ihr läßt sich die «Breite» des Rechtecks sehr gut programmieren.
Bleibt zum Schluß eine Art Zufallsschwingung, durch die das Rauschen entsteht. Bei diesem Diagramm müssen wir ehrlich zugeben, daß es sich nicht um authentisches Material, sondern eher um das zufällige Auf und Ab von Hannes' Hand handelt.

Diese ganzen Wellenformen lassen sich nun, wie schon erwähnt, für jede Stimme einzeln einstellen. Dazu dienen die Adressen in Tabelle 8.1.
In diese Speicherzellen können folgende Werte gePOKEd werden: 17 für Dreieckswellen, 33 für Sägezahn, 65 für Rechteck und 129 für Rauschen.

Zur Demonstration ein kleines Beispiel:

```
POKE 54276,17
```

Stimme	Adresse
1	54276
2	54283
3	54290

Tabelle 8.1 Wellenformen für die Stimmen 1 bis 3

schaltet für die erste Stimme die Wellenform Dreieck ein. Wenn Sie sich wundern, daß Sie noch nichts hören, keine Sorge: können Sie auch noch gar nicht. Vorher sind noch einige andere Parameter notwendig. Aber zuerst noch etwas zur Rechteckschwingung. Hier müssen Sie zusätzlich noch die Breite des Rechtecks angeben. Der Ausdruck Breite ist zwar technisch nicht ganz richtig, trifft aber am besten das Gemeinte. (Wir hoffen, die Techniker unter Ihnen verzeihen uns das.) Für diesen Breitenwert kann eine Zahl zwischen 0 und 4095 angegeben werden. Weil diese Zahl natürlich nicht in eine Speicherzelle paßt (Sie erinnern sich doch hoffentlich noch an das, was wir über das Fassungsvermögen einzelner Speicherzellen gelernt haben?), wird sie nach dem bekannten High-Byte/Low-Byte-Prinzip umgewandelt. Dadurch werden insgesamt sechs Speicherzellen benötigt, für jede Stimme zwei (Tabelle 8.2).
Hier sind übrigens immer die ersten Speicherzellen, also die mit der niedrigeren Nummer, diejenigen, in denen sich das Low Byte wiederfindet.

Wieder ein Beispiel:

Sie wollen die Rechteckschwingung auf der Stimme 1 auf eine Länge von 2049 programmieren.

```
POKE 54274,1:POKE 54275,8
```

Stimme	Speicherzellen
1	54274 & 54275
2	54281 & 54282
3	54288 & 54289

Tabelle 8.2 Breitenwert für die Stimmen 1 bis 3

Wie sich solche Änderungen der Pulslänge (das ist der technisch treffendere Ausdruck) akustisch auswirken, überlassen wir wieder Ihnen. Probieren Sie es später mit den ersten Tönen aus. Generell läßt sich sagen, daß ein Ton mit abnehmender Pulslänge immer dumpfer klingt.

Und gleich noch etwas zu den Wellenformen: In diesen Speicherzellen legte Commodore etwas Westernmentalität an den Tag: In den drei Wellenformregistern gibt es ein sogenanntes Key-Bit, das wie der Abzug von John Waynes Revolver funktioniert: Wenn es gesetzt wird, klingt der Ton an, schwillt dann ab und wird auf dem Sustain-Wert gehalten. Um den Ton ausklingen zu lassen, löschen Sie das Bit wieder. Die Wellenform muß aber erhalten werden: POKEn Sie zum Ausklingen des Tons einen der Werte 16, 32, 64 oder 128 – je nach Wellenform. Wenn Sie das beherzigen, können Sie auch ein flottes Knallen programmieren.

Und jetzt wären wir bei der Lautstärke angelangt. Sie gilt für alle drei Stimmen gleichzeitig. Die Adresse 54296 wird dafür benutzt. In diese Speicherzelle POKEn Sie am besten nur Werte von 0 (kein Ton) bis 15 (viel Ton).

Aber freuen Sie sich: Das ist natürlich noch nicht alles, um unseren SID zum Sprechen zu bringen. (Sie sehen, auch hier ist er wie ein Regierungssprecher. Obwohl das völlig gegen seinen Namen spricht, ist es nicht ganz einfach, ihn zum Reden zu bringen. Aber Sie werden lachen, genau das gekonnte Schweigen macht einen Regierungssprecher erst richtig gut. Paradox, nicht?)

Die letzte Formalität, die wir noch brauchen, ist das Wissen um die sogenannten Hüllkurven. Denn genau solche Hüllkurven müssen wir programmieren. Aber keine Angst, das hört sich nur so schlimm an. In Wirklichkeit geht es mit einiger Übung ganz fix.

Bei einer Hüllkurve geht man von folgender Überlegung aus: Ein Ton ändert in der Zeitspanne seines kurzen, bescheidenen Lebens ständig seine Lautstärke bzw. seine Intensität. Bei einer ADSR-Hüllkurve versucht man diesen Tonverlauf in vier Abschnitte zu unterteilen. ADSR ist die Abkürzung für Attack/Decay/Sustain/Release, was übersetzt etwa heißen würde: Anschlag/Abschwellen/Halten/Ausklingen. Bildlich vorstellen kann man sich das in etwa folgendermaßen (Bild 8.2).

Um diesen Kurvenverlauf in den Computer zu bringen, bedient man sich bei Commodore natürlich wieder des POKE-Befehls. Für jeden Abschnitt können Werte von 0 bis 15 eingegeben werden. Um Ihnen ein ungefähres Gefühl davon zu geben, was die Werte in den einzelnen Abschnitten bewirken, haben wir eine kleine Liste zusammengestellt, die zeigt, was der jeweils niedrigste und der jeweils höchste Wert bedeuten.

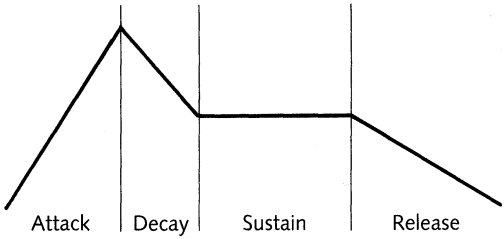


Bild 8.2 ADSR-Hüllkurve

Beim Anschlag stehen die Werte für die Intensität, das heißt dafür, wie hart bzw. stark man den Ton anschlägt. Der höchste Anschlagwert ist 0 (also hart), der niedrigste deshalb 15 (also weich).

Das Abschwellen der Töne kann ebenfalls stark oder schwach sein. Zum Beispiel beim Klavier. Hier schwillt der Ton langsamer ab, wenn der Spieler kein Pedal betätigt. Wenn er es aber tut, schwillt der Ton sehr schnell ab. Auch hier geht es also um eine Intensität. 0 wäre hartes, abruptes Abschwellen, 15 ein weiches, langsames Abschwellen.

Ein Ton kann bei einer bestimmten Lautstärke länger oder kürzer gehalten werden. Das ist auch der nächste wichtige Wert. 0 heißt hier, daß er dann gehalten wird, wenn er leise ist, 15 heißt, daß er bereits gehalten wird, wenn er noch laut ist.

Bliebe noch das Ausklingen. Das Ausklingen eines Tones kann sehr schnell gehen oder auch sehr langsam. Und genau dafür stehen auch die Werte: 0 bedeutet schnelles Ausklingen, 15 langsames Ausklingen.

Was bedeutet das alles? Nun, ein Ton kann mehr oder weniger intensiv angeschlagen werden, dann mehr oder weniger intensiv auf seine Normal- oder Grundlautstärke abschwellen, auf diesem Grundwert mehr oder weniger laut gehalten werden und dann mehr oder weniger schnell ausklingen. Das Kreuz mit diesen ganzen Daten ist, daß man sie schlecht in Worte fassen kann. Wenn Ihnen jetzt alle diese komischen Werte mehr oder weniger gleich vorkommen, dann gedulden Sie sich noch einen Augenblick, bis wir den ersten Ton ausprobieren. Dann wird das alles etwas klarer.

Vielleicht fragen Sie sich gerade, warum wir nicht, wie sonst auch, die Speicherzellen bereits genannt haben. Nun, das liegt daran, daß es sich auch mit denen etwas komplizierter verhält. Haben Sie schon mal was von einem Nibble gehört? Nein, auch wenn es sich so ähnlich anhört, das hat nichts mit

dem Nippel zu tun, den man durch die Lasche ziehen muß. Ein Nibble ist in der Sprache der Computerleute ein Halbbyte. Wenn Ihnen jetzt in Erinnerung an Karl May der Ausdruck Halbblut einfällt, dann ist das gar nicht so weit weg von dem, was wir meinen. Halbbyte ist ein 4-Bit-Wert, also ein halbes Byte. (Genauso, wie ein Halbblut nur ein halber Indianer ist.) Und weil unsere Werte , die wir POKEn wollen, immer nur von 0 bis 15 gehen, reicht ein Byte auch für zwei Werte – oder ein halbes Byte für einen Wert oder ein Halbbyte für einen Wert oder, um es fachmännisch auszudrücken, ein Nibble für einen Wert. Und deshalb wurden immer zwei Einstellungen in einem Byte zusammengefaßt. Auf diese Art und Weise wurde wieder einmal Speicherplatz gespart.

Sicherlich kommt es Ihnen mehr und mehr so vor, als sei das Bauen eines Computers eine Sache, bei der ständig in die Trickkiste gegriffen werden muß. Mit dieser Vermutung kommen Sie der Wahrheit sehr nahe. Und im Grunde ist es sogar so, daß sich oft die Ingenieure selbst wundern, daß so ein Kasten schließlich doch läuft.

Aber jetzt ist es mal wieder soweit. Eine Liste der Speicherzellen, in die gePOKEd wird, finden Sie in Tabelle 8.3.
Fein, werden Sie jetzt sagen, und wie bringe ich das jetzt in den Computer, damit er auch weiß, was wohin gehört? Eine kleine Formel soll Ihnen hier helfen.

Im ersten Fall, also für Attack und Decay, würde die Formel lauten:

Byte Wert = 16 * Attack + Decay

Attack/Decay	in Stimme
54277	1
54284	2
54291	3
Sustain/Release	in Stimme
54278	1
54285	2
54292	3

Tabelle 8.3 Adressen für ADSR-Hüllkurven

Im zweiten Fall, also für Sustain und Release:

```
Byte Wert = 16 * Sustain + Release
```

Eigentlich gar nicht so schwer, oder?

Noch ein Beispiel dazu: Für einen Ton sollen in Stimme 2 die Werte Attack gleich 15, Decay gleich 12, Sustain gleich 10 und Release gleich 3 sein. Überlegen Sie ruhig erst mal selbst, am besten auf einem Blatt Papier.

Die Lösung wäre:

```
POKE 54284, 16 * 15 + 12
POKE 54285, 16 * 10 +  3
```

Übrigens können Sie beim Experimentieren mit den Geräuschen die Hüllkurven ruhig zuerst einmal unberücksichtigt lassen. Man hat zwar die Möglichkeit, mit Ihnen sogar die Schwingungscharakteristik bestimmter Instrumente oder Geräusche relativ genau zu simulieren, aber der Anfänger sollte erst einmal ein Gefühl für die richtige Wellenform, die Lautstärke (auch die kann man ja während des Geräusches verändern) und die Frequenz bekommen.

Die Frequenz ist übrigens auch der letzte Wert, den Sie noch kennenlernen müssen.

Ein Ton oder Geräusch – also eine Schwingung – ist nämlich außer durch die Wellenform auch sehr stark durch die Frequenz bestimmt, mit der diese Welle schwingt. Und diese Frequenz kann man dem Commodore natürlich angeben und damit die Höhe des Tones, der herauskommen soll, bestimmen.

Die Frequenz geht bis etwa 4000 Hz (sprich 4000 Hertz). Der höchste Ton entspricht einem Wert von 65 535. Und weil das wieder eine Zahl ist, die nicht in eine Speicherzelle paßt, wird das High-Byte/Low-Byte-Prinzip verwendet. Wie das funktioniert, müßte mittlerweile ziemlich klar sein. Es geht nur noch darum, welche Speicherzellen angePOKEd werden müssen (Tabelle 8.4).

Stimme	High	Low
1	54272	54273
2	54279	54280
3	54286	54287

Tabelle 8.4 Frequenzadressen für Stimmen 1 bis 3

Sollten Ihnen die nötigen Werte dazu nicht mehr so klar sein, würden wir Sie bitten, das Errechnen von High-Byte- und Low-Byte-Werten im Abschnitt Speicheraufteilung unter der Überschrift «Wenn Bytes halbe-halbe machen» nachzuschlagen.

Wir wagen es: Ein Beispiel

Grundsätzlich ist es egal, in welcher Reihenfolge Sie die einzelnen POKEs eingeben. Sie müssen sich nur angewöhnen, die Wellenform als letztes zu POKEn. Denn wenn in diesen Zellen ein Wert steht, ist das für den Computer sozusagen der Startschuß. Ein Beispiel dafür wäre der Start einer Raumfähre. Egal, welcher Astronaut zuerst einsteigt – wichtig ist nur, daß alle drinnen sind, sobald der Countdown bei 0 angelangt ist. (Wie allerdings die Erfahrung aus der ersten Mondlandung zeigt, ist es wiederum keineswegs so egal, wer zuerst aussteigt ...)

Ist erst einmal die Wellenform gePOKEd, dann wird automatisch auch der Startvorgang ausgelöst, also das «Key-Bit» gesetzt.

Nun zu unserem Beispiel. Zuerst stellen wir die Lautstärke relativ laut ein. Damit ist natürlich die softwaremäßige Lautstärke gemeint. Ihren Fernseher sollten Sie höchstens auf Dallas-Lautstärke einstellen. Wenn Sie Dallas nicht mögen, darf es auch Denver-Clan-Lautstärke sein.

```
POKE 54296,12
```

Um eine halbwegs gutklingende Frequenz zu erhalten, geben Sie

```
POKE 54273,20
```

ein. Hier benutzen wir übrigens nur das High-Byte, weil es in unserem Fall voll ausreicht.

Jetzt versuchen wir noch, eine möglichst «flache» (sprich ausgewogene) Hüllkurve zu erreichen:

```
POKE 54277,140
POKE 54278,140
```

und jetzt kommt's: 5 ... 4 ... 3 ... 2 ... 1 ... 0

```
POKE 54276,17
```

schaltet unseren Ton ein – natürlich erst, wenn sie ⟨RETURN⟩ dazu eingegeben haben. Sodann hören Sie die «Dreieckige» des berühmten Tonkomponisten Grinaldo Spagati.

Und jetzt?

Nun können wir Sie nur noch einmal bitten, zu experimentieren. Mit all Ihrem Wissen sollte es Ihnen auch durchaus möglich sein, ein kleines Tontestprogramm zu schreiben, mit dem Sie die verschiedensten Töne ausprobieren können. Im Listinganhang finden Sie auch ein entsprechendes Programm dazu. Aber Sie wissen ja: Selbst ist der Programmierer. Nur noch ein kleiner Tip: Ändern Sie mal die Wellenform oder mit FOR...NEXT-Schleifen die Lautstärke oder die Frequenz.

Sie werden schnell merken, was für erstaunliche Effekte kleine Änderungen haben können. Und noch etwas, versuchen Sie mal, unser Programm mit den beiden aufeinanderfahrenden Autos zu vertonen, wenn Sie sich ein bißchen zurechtgefunden haben. Das ist eine gute Übung. Viel Spaß. Und vielleicht sollten Sie demnächst, zumindest für die ersten paar Tage Geräuschtest, den restlichen Familienmitgliedern eine Großpackung Oropax kaufen.

Warnung: Sämtliche Schäden, die sich im Zusammenhang mit diesem Kapitel und seinem Ausprobieren nach und vor 22 Uhr ereignen (zum Beispiel Fliegeralarm oder 100maliges Schreiben des Satzes «Ich darf nicht nachts meine Eltern mit meinem Computer belästigen»), entziehen sich der Haftung durch die Autoren oder des Verlages.

9
Ein Spiel mit Sprites

Schneewittchen und die sieben Zwerge

Und wieder ein Stück Märchengeschichte. Eine der nächsten Veröffentlichungen von Gerd Heinzelmann war die Enthüllung der tatsächlichen Vorgänge um ein junges Mädchen namens Schneewittchen. Wie so oft, war auch hier die Realität ganz anders, als die Überlieferung uns glauben machen will. Und auch diesmal ist es ein Prinz, der für die geänderte Märchenschreibung verantwortlich zu machen ist. Nachdem einer dieser Königssöhne am Sonntagmorgen bei der Lektüre eines bekannten Massenblattes erfahren hatte, daß eben jenes bewußte Mädchen hinter den sieben Bergen an Lebensmittelvergiftung litt (sie hatte von ihrer Stiefmutter einen vergifteten Apfel bekommen), machte er sich auf den Weg, sie zu retten. Da er sowieso gerade Krach mit seinem Vater und einen Erste-Hilfe-Kurs bestanden hatte, erschien dies auch nur angebracht. (Übrigens waren auch die Umstände seines Bestehens ein Zeichen für den mangelnden Intellekt seiner Zunft. Denn er bestand erst nach dem vierten Versuch, obwohl damals alles, was er können mußte, erfolgreiches Wachküssen war.) Während der Prinz sich auf den Weg machte, las Schneewittchens böse Stiefmutter die Geschichte in der Zeitung und fand, daß der Apfeltrick gar keine schlechte Idee war. Die Zeitung hatte die Story nämlich eigentlich nur erfunden, um ein hübsches Mädchen auf dem Titelblatt zu haben. Nachdem die Stiefmutter aber viel näher bei Schneewittchen wohnte, war sie auch schneller da als der Prinz, womit bewiesen wäre, daß Zeitungen nicht lügen. Während die Stiefmutter ihr Ziel erreichte, kann man das vom Prinzen nicht gerade behaupten. Als er nämlich so seines Weges wandelte, kam er an einem Knusperhäuschen vorbei, vor dem auch ein Junge und ein Mädchen standen. Die beiden sprachen ihn an,

ob er nicht etwas Kleingeld hätte. Um seine edle Abstammung zu dokumentieren, zog der Prinz einen Beutel Silbermünzen heraus. Damit war sein Schicksal besiegelt. In dem Knusperhäuschen befanden sich nämlich Videospielautomaten. Anstatt sich selbst einen Heimcomputer zu kaufen, gab der Prinz sein ganzes Geld beim Videospielen aus. Die Geschichte mit Schneewittchen beschloß er auf später zu verschieben. Als er aber tatsächlich keinen Groschen mehr besaß, kam ein junger gutaussehener Mann auf ihn zu und offenbarte ihm, daß er auf Modellsuche für eine Werbeagentur sei. Was man dort bräuchte, sei ein junger Prinz für eine gewisse Keksrolle. Weil auch das Gehalt nicht schlecht war, sagte der Prinz freudig zu, was die Rettung von Schneewittchen wieder deutlich verzögerte.

Was aber war in der Zwischenzeit hinter den sieben Bergen geschehen? Schneewittchen, die von all dem in der Zeitung gelesen und in der Hoffnung auf die Rettung durch den Prinzen den Apfel gegessen hatte, wurde des Wartens langsam überdrüssig. Als sie dann auch noch eine Keksrolle mit seinem Bild geschenkt bekam (unvergiftet!), beschlossen sie und die Zwerge, ihr Schicksal selbst in die Hände zu nehmen. Die Zwerge erzählten ihr von dem Zauberkristall, nach dem sie schon seit Jahren buddelten. Aus bisher ungeklärten Gründen sahen die Zwerge plötzlich eine Möglichkeit, das, was sie seit Jahren nicht geschafft hatten, in wenigen Minuten zu vollbringen. Allerdings nur unter der Voraussetzung, daß Schneewittchen sie begleitete und ihnen ständig aus einem der wenigen nicht von arbeitslosen Fröschen bewohnten Brunnen in der Nähe Wasser schöpfte und sie so schneller arbeiten konnten.

Und genau das ist jetzt auch Ihre Aufgabe. Aber erst, nachdem Sie das Spiel eingegeben haben, über das wir uns jetzt noch ein bißchen unterhalten wollen.

Zeile 10: Hier werden zunächst mit POKE 53280,11 und POKE 53281,11 die Hintergrund- und Rahmenfarbe auf Hellgrau (Farbnr. 11) gesetzt. Durch POKE 53265,91 wird der Hintergrundfarbmodus aktiviert. Das hat zur Folge, daß der Titel, der gleich gedruckt werden soll, mit gelber Schrift auf einem roten Balken dargestellt wird, was gleich viel besser aussieht als normale Schrift. Mit dem POKE 53283,2 (2 ist ja Rot) wird die Farbe dieses Balkens festgelegt. Der letzte POKE 53272,21 setzt schließlich den Zeichensatz auf die normale Adresse zurück, was nötig ist, da das Spiel mit umdefinierten Zeichen arbeitet, sich aber später von selbst wieder startet und dann für den Titel die normalen Buchstaben verwendet.

Zeile 20: Durch PRINT CHR$(8) wird die Umschaltung zwischen Groß- und Kleinbuchstaben blockiert, was empfehlenswert ist, da wir mit selbstdefinierten Zeichen arbeiten werden, aber keine Kleinbuchstaben definiert haben ...

Zeile 30: Jetzt wird endlich der Titel " SCHNEEWITTCHEN UND DIE SIEBEN ZWERGE " gedruckt. Die Steuerzeichen bedeuten im einzelnen folgendes: Das invertierte Herzchen steht für ⟨SHIFT⟩-⟨CLR/HOME⟩, also Bildschirm löschen. Mit den zehn folgenden invertierten Q's wird der Cursor von der HOME-Position aus zehn Zeilen nach unten bewegt. Sie tippen zehnmal ⟨CRSR DOWN⟩. Das invertierte Pi steht für gelbe Farbe (⟨CTRL⟩-⟨8⟩). Wir wollen unseren Titel ja gelb auf rot schreiben. Damit wir bei den Leerstellen auch einen Balken bekommen, drucken wir noch ⟨RVS ON⟩. Das invertierte R steht dafür.

Zeilen 40 bis 70: Hier werden die Sprites aus den DATA-Zeilen gelesen und in den Speicher gePOKEd. Dabei werden die Speicherzellen ab 704 (Spriteadresse 11), 832 (= 13), 896 (= 14) und 960 (= 15) belegt. Für jeden Sprite werden 63 Bytes benötigt. Deshalb FOR X=0 TO 62 ...

Zeile 80: Mit POKE 53269,255 werden alle 8 Sprites aktiviert. Sprite Nr. 1 (Schneewittchen) wird in Y-Richtung vergrößert. Deshalb POKE 53271,1. Mit POKE 53276,255 werden alle Sprites in den Multicolormodus geschaltet. Als Multicolorfarben dienen 8 (Orange, als Hautfarbe) und 0 (Schwarz, beispielsweise für Schuhe und Haare ...). GePOKEd werden diese Farben in die Adressen 53285 und 53286.

Zeile 90: Schneewittchens Kleid wird mit POKE 53287,3 hellblau gemacht (das ist ja die Farbadresse für Sprite 0). Diese Farbe kann auch in Multicolor für jeden Sprite einzeln angegeben werden. Daher bekommen die Zwerge mit der dann folgenden Schleife auch rote (Farbnr. 2) Mäntel.

Zeile 100: Den einzelnen Sprites werden die Bitmuster zugeteilt. Sprite 0 (Schneewittchen) hat die Spritenummer 13, also POKE 2040,13. Die sieben Zwerge bekommen zunächst die Nummer 14.

Zeilen 110 bis 130: Hier sind nun die Bitmuster der Sprites. Geben Sie beim Eintippen ganz besonders acht, daß die Zahlen stimmen. Nachher zeigen wir Ihnen noch, wie Sie Ihre Eingaben überprüfen können. Von Zeile 110 bis 130 (später hat dieses Muster die Spritenummer 11) stehen die Daten eines Zwerges, der etwas erschöpft in der Gegend sitzt, weil er kein Wasser bekommen hat. Das sieht dann so aus (Bild 9.1).

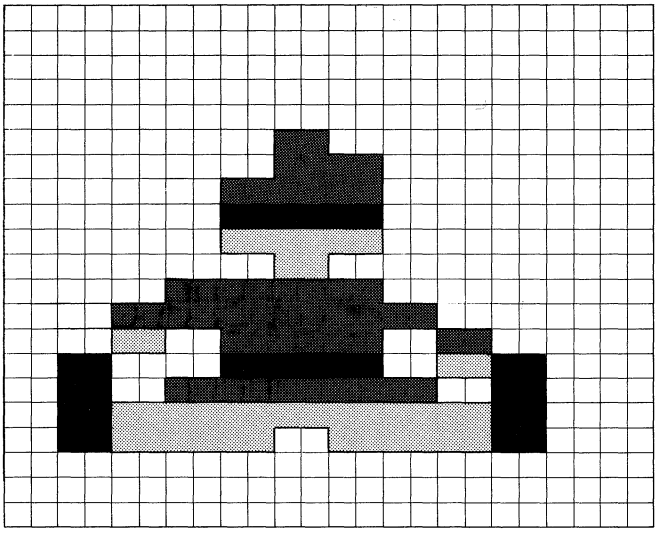


Bild 9.1 Sitzender Zwerg

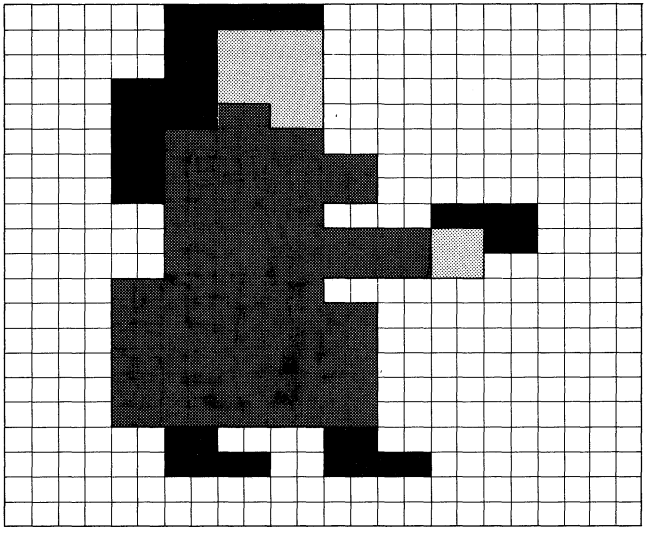


Bild 9.2 Schneewittchen

Zeilen 140 bis 170: Das ist unser Schneewittchen. Es wird die Spritenummer 13 haben (Bild 9.2).

Zeilen 180 bis 210: Das ist ein Zwerg in der ersten Bewegungsphase. Um das «Laufen» der Zwerge besser aussehen zu lassen, haben wir es in zwei verschiedenen Phasen abgespeichert. Wie bei einem Zeichentrickfilm entsteht dann durch das rasche Abwechseln der beiden «Teilbilder» die Illusion einer Bewegung. Diese Phase wird als Spritenummer 14 abgelegt (Bild 9.3).

Zeilen 220 bis 250: Das ist jetzt die andere Bewegungsphase. Sie wird in Nummer 15 gespeichert (Bild 9.4).

Zeile 260: Hier werden die verschiedenen Sonderzeichen gelesen und in den Speicher gePOKEd, die wir verwendet haben. Dabei belegen wir die Buchstaben «A» bis «I» (Bildschirmcodes 1 bis 9) neu. Die Buchstaben «A» bis «C» sind Teilstücke, aus denen wir in Lego-Art unser Gebirge aufbauen, aus den umdefinierten Buchstaben «D» bis «I» entsteht der Brunnen. Unseren selbstdefinierten Zeichensatz legen wir ab der Adresse 14336 im Speicher ab. Das ist die größtmögliche Adresse für einen eigenen Zeichensatz. Da wir die Zeichen 1 bis 9 verändern, müssen wir deren neue Bitmuster ab dem 8. Byte (erstes Byte des Zeichens «A» [1 * 8]) bis zum 79. Byte (letztes Byte des Zeichens «I» [9 * 8 + 7]) POKEn. Das tut unsere Schleife FOR X=8 TO 79...

Zeile 270: Da wir die Zeit und die Stärke von Schneewittchen darstellen wollen, brauchen wir einige Zeichen aus dem Charakter-ROM. Um diese kopieren zu können, schalten wir mit POKE 56334,0 die Interrupts ab und blenden mit POKE 1,51 das Charakter-ROM in den Adreßbereich des 6510. Nun übernehmen wir die Zeichen «0» (Bildschirmcode 48) bis «9» (Bildschirmcode 57) sowie ":" (BS-Code 58). Also kopieren wir die Bytes (vom Anfang des Zeichensatzes aus gesehen) 384 (48 * 8) bis 471 (58 * 8 + 7). Genau das tut unsere Schleife FOR X=384 TO 471..

Zeile 280: Hier kopieren wir den Buchstaben «S» (BS-Code 19), der uns später als Abkürzung für «Stärke» dienen soll. Kopiert werden die Bytes 152 (19 * 8) bis 159 (19 * 8 + 7).

Zeile 290: Dasselbe geschieht mit dem «Z», das wir für «Zeit» brauchen. Es hat den Bildschirmcode 26, also kopieren wir die Bytes 208 (26 * 8) bis 215 (26 * 8 + 7).

Zeile 300: Das Zeichen «%» brauchen wir für die Angabe der Stärke Schneewittchens in Prozent. Es hat den BS-Code 37. Also werden die Bytes 296 (37 * 8) bis 303 (37 * 8 + 7) kopiert.

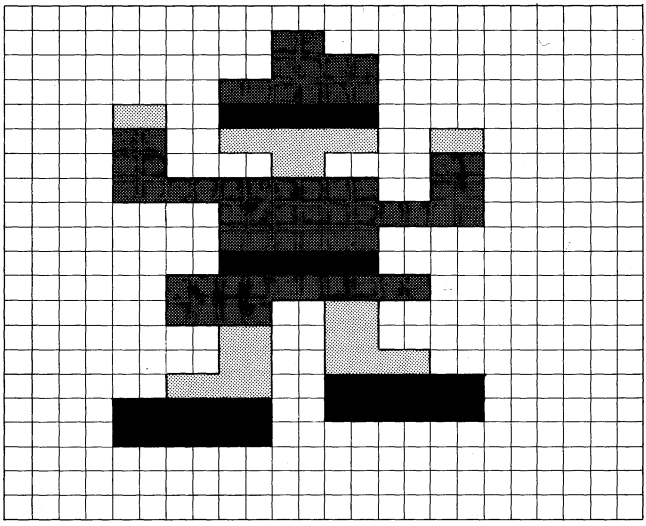


Bild 9.3 Gehender Zwerg (Teilbild 1)

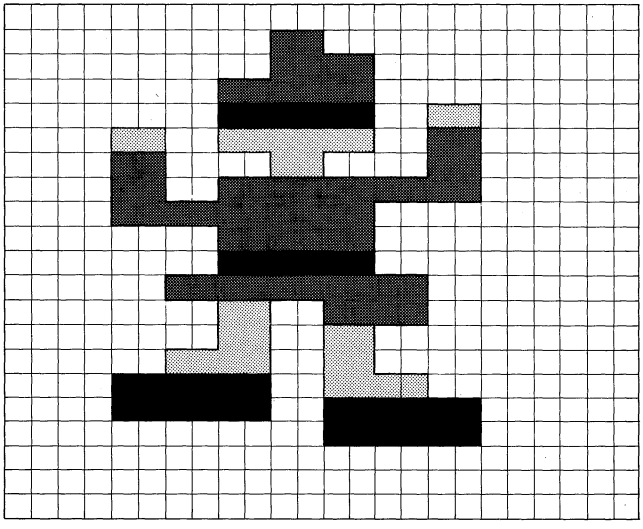


Bild 9.4 Gehender Zwerg (Teilbild 2)

Zeile 310: Da wir einen leeren Hintergrund brauchen, müssen wir auch das Zeichen " " (Leertaste oder engl. ⟨SPACE⟩) übernehmen. Es hat den Bildschirmcode 32. Dieses Zeichen zu kopieren wird gern vergessen, ist aber unbedingt nötig. Denn an der Stelle, wo im Charakter-ROM lauter 0-Bits stehen, befinden sich im RAM ja zufällige Werte. Nun besteht das Zeichen " " aber wirklich nur aus leeren Bytes. Daher brauchen wir nicht das Charakter-ROM zu lesen, sondern können direkt sieben Nullen in die Bytes 224 (32 * 8) bis 231 (32 * 8 + 7) POKEn.

Zeile 320: Wir sind fertig mit dem Kopieren. Also blenden wir die I/O-Register mit POKE 1,55 wieder ein und schalten die Interrupts mit POKE 56334,1 wieder an. Bild 9.5 zeigt die Muster umdefinierter Zeichen.

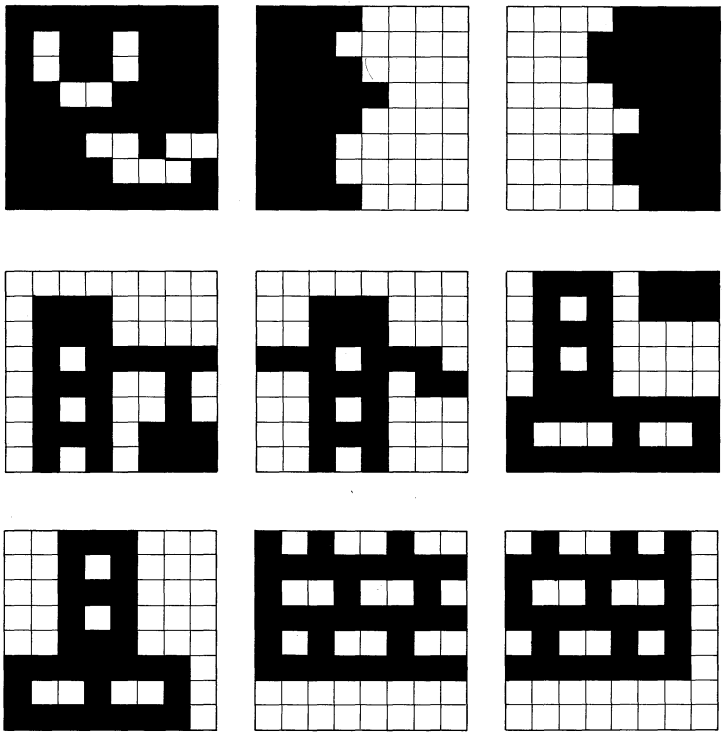


Bild 9.5 Muster umdefinierter Zeichen

Wenn Sie das Programm jetzt bis zu dieser Stelle eingegeben haben, kommen Ihnen vielleicht Zweifel, ob Ihre DATA-Zeilen auch mit unseren übereinstimmen. Das können Sie aber ziemlich einfach nachprüfen. Geben Sie mal folgende Zeile ein:

```
500 FOR X=1 TO 324: READA: W=W+A: NEXT X: IF W<>21465 THEN PRINT "FEHLER!!!"
```

und starten Sie sie mit RUN 500. Diese Zeile zählt alle DATA-Werte zusammen und vergleicht sie mit unserer Summe (auch «Prüfsumme» genannt). Wenn alles stimmt, meldet sich der Computer wieder mit «READY». Wenn aber nicht, druckt er «FEHLER!!!». Dann sollten Sie Ihre Eingaben noch mal mit dem Listing vergleichen und Fehler ausbessern. Wenn schließlich kein «FEHLER!!!» mehr auftritt, können Sie ziemlich sicher sein, daß Ihre DATAs und somit Ihre Sprites und Sonderzeichen stimmen. Bevor Sie dann weitermachen, sollten Sie die Zeile 500 löschen. Dazu tippen Sie einfach 500 ein und drücken ⟨RETURN⟩.

Zeile 420: Nachdem wir jetzt so fleißig alle Zeichen umdefiniert haben, wollen wir doch auf unseren neuen Zeichensatz ab 14336 umschalten: POKE 53272,27

Zeile 430: Jetzt sehen Sie auch den tieferen Sinn, warum wir ausgerechnet die Zeichen «A», «B» und «C» umdefiniert haben. In dieser Zeile wird der obere Teil des Berges auf den Bildschirm gedruckt. Bei vielen Programmlistings müssen Sie sich bei solchen Gelegenheiten durch einen unübersichtlichen Dschungel von Steuer- und Grafikzeichen kämpfen und nach dem System «Dieses Zeichen könnte gemeint sein» vorgehen. Wir haben Ihnen aber das Abtippen einfacher gemacht, indem wir normale Buchstaben verwendet haben, die wir vorher umdefinierten. So können Sie ganz einfach («einmal C, fünfmal A...») diese Zeilen eintippen. Die Steuerzeichen vorn bedeuten: invertiertes Herzchen für Bildschirm löschen (⟨SHIFT⟩-⟨CLR/HOME⟩) und das invertierte Kreuz für Dunkelgrau (⟨C=⟩-⟨5⟩). Vergessen Sie den Strichpunkt nicht, damit kein Zeilenvorschub erfolgt.

Zeile 440: Die sieben Gänge, in denen die Zwerge arbeiten, werden jetzt gedruckt. Dazu verwenden wir dasselbe Muster 16 Zeilen lang. Also kommt die Schleife FOR X= 1 TO 16

Zeile 450: Dieses Muster aus unseren Sonderzeichen «A», «B» und «C» stellt wieder ein Stück Berg dar, diesmal von sieben Gängen unterbrochen.

Zeile 460: NEXT X zu der Schleife von Zeile 440

Zeile 470: Hier drucken wir den Brunnen auf den Schirm. Dazu verwenden wir Steuerzeichen. Ab der aktuellen Cursorposition (sozusagen am Fuß des Berges) gehen wir drei Zeilen runter (also drei invertierte Q's, was für Sie dreimaliges Drücken von ⟨CRSR DOWN⟩ bedeutet), eine Spalte nach rechts (wofür die invertierte Klammer steht – ⟨CRSR RIGHT⟩) und drucken dort den ersten Teil unseres Brunnens, bestehend aus den Zeichen «D» und «E». Alsdann gehen wir mit dem Cursor eine Zeile tiefer (inv. Q – ⟨CRSR DOWN⟩) und zwei Positionen zurück. Das ergibt ein Steuerzeichen, das aussieht wie ein invertierter Strich. Drücken Sie zweimal ⟨CRSR LEFT⟩. Dann kommen die Zeichen «F» und «G», die die zweite Zeile unseres Brunnens darstellen. Die Prozedur von oben wiederholt sich (⟨CRSR DOWN⟩ ⟨2 * CRSR LEFT⟩), und wir können den letzten Teil des Brunnens drucken. «H» und «I».

Zeile 480: Die Y-Koordinaten der sieben Zwerge (Sprites 1 bis 7) werden allesamt auf 54 gesetzt.

Zeile 490: Die X-Koordinaten der Zwerge mußten wir beim Programmieren mehr oder weniger von Hand austarieren. Daher haben wir die Werte, die die Sprites genau so positionieren, daß sie jeweils in der Mitte ihres Ganges sind, in der DATA-Zeile 500 abgelegt. Sie werden mit READ einzeln eingelesen und dann in die entsprechenden Register gePOKEd. Beachten Sie, daß wir für die Sprites 6 und 7 noch ein zusätzliches 9. Bit im Register 53264 setzen müssen, da deren tatsächliche X-Koordinaten größer als 255 sind. Der POKE 53264,192 ($2^6 + 2^7 = 128 + 64 = 192$) erledigt das.

Zeile 500: In dieser DATA-Zeile sind die X-Koordinaten abgelegt, die wir austariert haben.

Zeile 510: Schließlich bekommt auch noch Schneewittchen eine Position. Ihre X-Koordinate ist 30, Y ist 200.

Zeile 520: Die Variable S steht ab jetzt für «Stärke». Hier ist der prozentuale Wert ihrer Kraft abgelegt. Momentan beträgt er 100%, also ist S = 1. Der Detektor für Sprite-Sprite-Kollisionen wird mit POKE 53278,0 zurückgesetzt. Das sollte man am Anfang eines Programms immer tun, da die Sprites beim Aufbau ja schon zusammengestoßen sein könnten. Die Variable Z steht ab jetzt für die Anzahl der noch arbeitsfähigen Zwerge. Momentan ist sie 7. Die Zeitvariable TI$, die wir später zur Zeitmessung benötigen, wird mit TI$="000000" zurückgesetzt. R ist die Spielzeit in Sekunden. Da wir 4 Minuten spielen wollen, bekommt sie den Wert 240.

Zeile 530: Hier werden die Werte für den Ton gesetzt, den die Zwerge beim Arbeiten machen. Vergleichen Sie sie damit, was Sie in unserem Kapitel «Musik und Geräusche» gelesen haben. Die Gesamtlautstärke wird auf 9 gesetzt (POKE 54296,9). Die folgenden Einstellungen gelten für Stimme #1: Die Frequenz des Geräusches hat den Dezimalwert 2209. Das sind rund 125 Hertz. Diese wird nach High Byte/Low Byte in die Register 53272 und 53273 gePOKEd. Die nächsten beiden POKEs legen die ADSR-Hüllkurve fest: Attack = 1, Decay = 0, Sustain = 3, Release = 7. So entsteht ein Ton, der sehr hart angeschlagen wird, sofort auf seine Normallautstärke zurückfällt und mittelmäßig schnell ausklingt. Das Ganze soll das Hacken der Zwerge auf Stein untermalen.

Zeile 540: Da wir gerade beim Musikmachen sind, legen wir gleich die Hüllkurve für die Stimme #2 fest. Diese Stimme werden wir immer dann gebrauchen, wenn ein Zwerg erfolgreich Wasser bekommen hat (freudiges Geräusch) oder vor Erschöpfung umfällt (weniger freudiges Geräusch ...). Die Hüllkurve sieht folgendermaßen aus: A = 5, D = 10, S = 14, R = 12. Alles in allem ist dies ein relativ träger Ton, der nicht besonders intensiv anklingt, langsam auf sein Grundniveau zurückfällt und ebenfalls langsam ausklingt.

Zeile 550: Hier beginnt das eigentliche Steuerprogramm. Der Variablen T wird der Inhalt der Adresse 203 zugewiesen. In dieser Adresse steht der Code der gerade gedrückten Taste. Vergleichen Sie dazu auch Bild 10.1.

Zeile 560: A = 1 − A. Diese kleine Formel ist ziemlich praktisch. Sie liefert als Ergebnis abwechselnd die Werte 0 und 1. Schauen wir uns das mal genauer an. Zuerst ist A = 0. Also wird durch diese Formel A auf 1 gesetzt. Denn 1 − 0 = 1 und dieser Wert wird der Variablen A zugewiesen. Das nächstemal, wenn das Programm an dieser Stelle vorbeikommt, ist A = 1. Die Formel macht daraus 1 − 1 = 0. A ist wieder 0. Beim nächstenmal wird aus der 0 wieder 1 usw. Was hat das aber nun für einen Zweck? Unsere Zwerge sollen mächtig aktiv, also ständig in Bewegung sein. Um das zu erreichen, haben wir ja oben zwei Bewegungsphasen programmiert. Doch dazu gleich. Außerdem sollen sie aber auch kräftig Lärm machen. Also spielen wir jedesmal, wenn das Programm hier ankommt, das Geräusch an, das die Zwerge beim Arbeiten von sich geben. Der POKE 54276 dient dazu, den Ton auf Stimme #1 einzuschalten. Aber jetzt zu unseren Bewegungen. Alle Zwerge bewegen sich im Gleichschritt. Das fällt im Spiel gar nicht weiter auf.

Die Schleife FOR X=1 TO 7 ... sorgt für diese Bewegung, indem sie für jeden der Sprites 1 bis 7 abwechselnd (abhängig von A) den zugehörigen Zeiger auf Spritenummer 14 oder 15 setzt. Jetzt könnte es aber sein, daß einige Zwerge schon außer Gefecht sind. Die dürfen sich natürlich nicht bewegen. Um das zu erreichen, gibt es das Feld Z(I), also eine Variable für Z(1), eine für Z(2) usw. bis Z(7). So ist jedem Zwerg – wie könnte es auch anders sein? – ein Platz in diesem Feld zugeordnet. Dabei gilt, was zugegebenermaßen etwas unüblich, aber einfacher ist, folgende Belegung: Wenn A(I) = 0, dann ist der Zwerg Nr. I noch im Rennen. Ist aber Z(I) = 1, dann ist der Zwerg umgefallen. Abhängig von dieser Eintragung wird der entsprechende Zwerg dann bewegt oder eben nicht.

Zeile 570: Da der letzte Befehl der Zeile 560 ein IF... war, wird er nicht immer ausgeführt. Deshalb muß das NEXT unserer «Zwergenbewegungsschleife» in der nächsten Zeile stehen. So trifft es der Computer – egal, ob die Bedingung erfüllt war oder nicht. Was jetzt kommt, ist sozusagen das Gift im Programm. Die Variable S, die ja für Schneewittchens Stärke steht, wird (wenig, aber immerhin ...) verkleinert. Der Wert 0.0013 hat sich als ganz vernünftig erwiesen. Wenn Sie das Spiel einfacher machen wollen, setzen Sie einen kleineren Wert ein, wenn Sie es schwieriger haben wollen, einen höheren. B ist eine Variable, die für «Bewegung» steht. Sie wird gleich für die Fortbewegung von Schneewittchen gebraucht. Damit Schneewittchen nicht von allein weiterläuft, wenn der Spieler keine Taste drückt, wird dieser Bewegungsfaktor jedesmal wieder gelöscht, wenn das Programm hier vorbeikommt. Nachdem diese ganzen Aktualisierungen erledigt sind (der Computer braucht weniger als eine Sekunde dazu), können wir auch den Ton mit POKE 54276,0 wieder abschalten. So ergibt sich ein typisches «Hack»-Geräusch. Das letzte in dieser Zeile ist noch eine kleine Aufräumarbeit: Wenn die Stärke S, nach der Verminderung von gerade eben, kleiner als 0 ist (was ja keinen Sinn haben würde, was soll eine «negative Stärke» sein?), soll sie doch gleich 0 bleiben.

Zeile 580: Das ist der eigentliche Steuerungsteil in diesem Programm. Wenn die ⟨CRSR UP/DOWN⟩-Taste gedrückt worden ist (sie hat den Code 7, die Variable T gibt darüber Aufschluß), dann soll sich Schneewittchen nach hinten, also in Richtung Brunnen bewegen. Dazu wird B entsprechend negativ. S ist ja die Stärke Schneewittchens. Wenn diese 100% beträgt, ist S = 1. Das bedeutet also, daß im Bestfall ein Schritt von Schneewittchen 7 Bildschirmpunkte beträgt. Wenn sie schwächer wird, verkleinern sich ihre

Schritte entsprechend. Ist ihre Stärke 0, dann kann sie sich überhaupt nicht mehr bewegen.

Zeile 590: Hier wird dasselbe für die Taste ⟨CRSR LEFT/RIGHT⟩ (der Code ist 2) gemacht. Nach vorn bewegt sich Schneewittchen etwas schneller. Ein Schritt in diese Richtung beträgt im Bestfall 8 Bildschirmpunkte.

Zeile 600: Diese Zeile stellt den neuen X-Wert für Sprite 0 fest. Dafür wird die augenblickliche X-Koordinate mit PEEK (53248) ausgelesen. Dazu kommt B, also die Bewegung, die oben definiert worden ist. Wenn PEEK (53264) = 193, dann bedeutet das, daß Schneewittchen sich rechts von der Koordinate 255 befindet. Sie wissen ja, für die X-Richtung gibt es ein Bit Nr. 8. Allerdings sind diese Bits für die Zwerge 7 und 6 schon gesetzt (siehe Zeile 490). Demnach ist der Wert von vornherein 192. Wenn jetzt Sprite 0 (Schneewittchen) dazukommt, ist der Wert 128 + 64 + 1 = 193. Und dann – lange Zeile, kurzer Sinn – muß die X-Koordinate um 256 erhöht werden.

Zeile 610: In dieser Zeile wird abgefragt, ob Schneewittchen etwa abhauen, also hinter dem Bildschirmrand verschwinden will. Das wäre der Fall, wenn die X-Koordinate kleiner als 22 würde. Ist das der Fall, soll sie schön bei 22 bleiben.

Zeile 620: Anhand der neuen X-Koordinate wird erst einmal festgestellt, ob das 9. Bit für Schneewittchen gesetzt werden muß. Wenn dem nicht so ist, weil die Koordinate kleiner als 256 ist, dann wird in das entsprechende Register der «Normalwert» 192 gePOKEd. (Sie wissen ja: die beiden Rechtsaußen unter den Zwergen.) Die Variable V, die nachher in das X-Register des Schneewittchen-Sprites gePOKEd wird, kann also bedenkenlos den X-Wert übernehmen.

Zeile 630: Sollte X aber größer oder gleich 256 sein (und dafür kann man schreiben: IF X⟩255...), dann wird unser 9. Bit gesetzt und die Variable V, die ja gePOKEd werden soll, also gar nicht größer als 255 sein darf, entsprechend verringert. Sollte V jetzt noch größer als 70 sein (also der X-Wert größer als 256 + 70 und damit größer als 326), muß Schneewittchens Flucht nach vorn auch entsprechend verhindert werden. Also wird V in diesem Fall zurück auf 70 gesetzt.

Zeile 640: Jetzt kommt der Lohn für alle unsere Mühe. Der Wert V, unsere entsprechend behandelte X-Koordinate, kommt ins X-Register von Sprite 0 (Schneewittchen).

Zeile 650: In dieser Zeile wird abgefragt, ob Schneewittchen den Brunnen berührt. Sobald eine Kollision zwischen dem Schneewittchen-Sprite und dem

Hintergrund festgestellt wird (und das einzige Objekt im Hintergrund, mit dem der Sprite 0 kollidieren könnte, ist der Brunnen), hat Schneewittchen wieder Wasser. Zuerst wird das Kollisionsregister gelöscht. Dann wird es abgefragt. Wenn sich Sprite 0 gerade über dem Brunnen befindet, ist der Inhalt dieser Adresse gleich 1 und somit die IF...THEN-Bedingung erfüllt. Also wird die Variable WA, die natürlich für Wasser steht, auf 1 gesetzt. Solange sie das ist, hat Schneewittchen Wasser, das sie einem Zwerg zu trinken geben kann. Damit der Spieler irgendwie die beiden Zustände «Wasser» und «kein Wasser» unterscheiden kann, wird der Rahmen blau gePOKEd.

Zeile 660: Hier wird die Kollision zweier Sprites überprüft. Da die Zwerge voneinander durch den Felsen getrennt sind, kann eine Kollision nur mit Schneewittchen stattfinden. Wieder wird zunächst das Kollisionsregister gelöscht. Wenn sich dann der PEEK(53278) von 0 unterscheidet (was bedeutet, daß eine Kollision stattgefunden hat) und gleichzeitig die Wasser-Variable WA = 1 ist (was bedeutet, Schneewittchen hat Wasser), dann passiert der Reihe nach folgendes: WA = 0, also Schneewittchen hat kein Wasser mehr und muß neues holen. POKE53281,11 – der Rahmen wird wieder grau, so daß auch der Spieler merkt, daß das Wasser futsch ist. OK = 1. Diese OK-Variable ist immer dann 1, wenn der Zwerg, der gerade unterwegs ist (da kommen wir gleich drauf), sein Wasser erhalten hat, also weiterarbeiten kann. Mit GOSUB 1000 wird in ein Tonunterprogramm gesprungen, das diese erfreuliche Begebenheit durch ein entsprechendes Geräusch dokumentiert.

Zeile 670: Die Variable ZW stellt auch kein Rätsel mehr für uns dar. Sie steht für «Zwerg». Aber man sollte das schon noch etwas genauer festlegen: ZW ist immer dann 1, wenn gerade ein Zwerg unterwegs zu Schneewittchen ist, um Wasser zu holen. Solange diese Variable 1 ist, wird der Rest dieser Programmzeile nicht ausgeführt. Ist ZW aber 0, was bedeutet, daß kein Zwerg unterwegs ist, dann passiert folgendes: Mit INT(7*RND(1))+1 wird eine Zufallszahl zwischen 1 und 7 ausgewählt. Diese Zahl heißt ab jetzt ZN, was als Abkürzung für Zwergennummer gedacht ist. Wozu diese Zwergennummer? Sie ist die Nummer des Zwerges (wir haben die Zwerge natürlich ganz lapidar von 1 bis 7 durchnumeriert), der gerade unterwegs ist. Die Variable W ist die Bewegungsrichtung dieses Zwerges. Sie ist momentan +5, das bedeutet, der Zwerg bewegt sich mit steigenden Y-Werten, also nach unten. Die Variable OK wird gelöscht, denn der Zwerg, der da jetzt losmarschiert, hat ja noch Durst. Sollte jedoch der Zwerg, der auf Zufallsbasis

ausgewählt wurde, überhaupt nicht mehr verfügbar sein (also vor Ermattung irgendwo im Bergwerk sitzen und ausruhen), dann war die ganze Aktion für diesmal umsonst. ZW, die Variable, die anzeigt, ob einer unterwegs ist, wird wieder 0, und mit einem GOTO 550 springt das Programm wieder zur Tastaturabfrage, denn die Zeilen, die jetzt noch kommen, dienen lediglich der Bewegung des Zwerges.

Zeile 680: Und genau das tun wir dann auch gleich. Die Y-Koordinate des auserwählten Zwerges ZN wird um den Wert W vergrößert. Sollte W später einmal negativ sein, bewegt sich der Sprite wieder nach oben, da die Koordinate dann in Wirklichkeit immer kleiner wird. Der aktuelle Stand des Zwerges wird mit PEEK aus dem zuständigen Register gelesen und, nachdem er aktualisiert wurde, mit POKE dorthin zurückgebracht.

Zeile 690: Sollte die Y-Koordinate unseres Zwerges ZN größer als 200 sein (dann ist er nämlich auf der Höhe von Schneewittchen, wo er ja mit ihr zusammenstoßen und somit Wasser erhalten kann), darf unser kleiner Freund umkehren. Seine Bewegung wird also negativ (W = −5), und gleichzeitig wird die Variable D auf 1 gesetzt. D steht ungefähr für «Rückzug». Sie sehen, daß uns langsam die symbolträchtigen Buchstaben ausgingen ...

Zeile 700: Ist unser kleiner ZN wirklich auf dem Rückzug (D = 1) und seine Y-Koordinate kleiner als 55, dann ist er wieder im Bergwerk. Sein Rückzug ist also beendet (D = 0). Mit ZW = 0 wird gleich festgehalten, daß keiner mehr unterwegs ist. Aber jetzt kommt's drauf an. Wenn der Arme nämlich kein Wasser bekommen hat, dann ist wieder ein Zwerg weniger im Rennen. Im Feld Z(ZN) wird er gelöscht, das heißt, in diesem Fall auf 1 gesetzt. Vergleichen Sie dazu die Zeile 560.

Zeile 710: Sollte die Anzahl der «gesunden» Zwerge = 0 sein, ist also kein Zwerg mehr in der Lage, seine Arbeit zu verrichten, dann ist das Spiel zu Ende. Schneewittchen hat verloren. Das Programm springt zu Zeile 900.

Zeile 720: Hier wird festgestellt, wie lange Schneewittchen noch spielen muß. Die Zeitvariable TI$ haben wir ja am Anfang zurückgesetzt. Also war auch TI (das ist die Variable, die in 60stel-Sekunden zählt) = 0. Nun wird abgefragt, wie viele Sekunden seit Anfang vergangen sind. Ist die Differenz zwischen R (der Zeit, die gespielt werden muß) und der Spielzeit kleiner als 1 Sekunde, dann ist Schluß. Da noch mindestens ein Zwerg übrig sein muß (sonst wäre das Programm in Zeile 710 zum «Ende-Programm» ab Zeile 900 gesprungen), hat Schneewittchen gewonnen – also auf zur Zeile 800, wo der Sieg dann auch gebührend gefeiert wird.

Zeile 730: Hier wird ein Maschinenunterprogramm des Betriebssystems benutzt. Das heißt, wir nutzen die Arbeit, die Commodore schon geleistet hat, um den Computer etwas machen zu lassen, was wir sonst selbst programmieren müßten. Dieses Maschinenprogramm setzt den Cursor an die Position, die wir in den Adressen 211 und 214 angegeben haben. Mit SYS 58732 wird es aufgerufen. Wenn es fertig ist, springt es – wie ein gutes Unterprogramm – dahin, wo es herkam. So ersparen wir uns komplizierte Steuerzeichen und können direkt im rechten unteren Eck hinPRINTen, was wir dort hinschreiben wollen. Das wäre zunächst einmal die noch verbleibende Spielzeit P und Schneewittchens Stärke S. Damit das alles einigermaßen gut aussieht, verwenden wir folgende Steuerzeichen: Der invertierte Pfeil nach oben steht für ⟨CTRL⟩-⟨6⟩, also Grün. Es folgt der Buchstabe «Z:» für «Zeit». Die eigentliche Zeitvariable muß außerhalb der Anführungszeichen stehen, damit auch ihr Wert gedruckt wird und nicht bloß ein zweiter Buchstabe «Z». Die dann folgenden Steuerzeichen für Gelb (inv. Pi, ⟨CTRL⟩-⟨8⟩) und Cursor nach links (inv. Strich, ⟨CRSR LEFT⟩) müssen aber wieder innerhalb der Anführungszeichen stehen. Der Cursor kommt wieder eine Position nach links, weil der Commodore nach einer Zahl (wie z. B. dem Wert von P) grundsätzlich eine Position frei läßt. Sodann folgt der Buchstabe «S:» für «Stärke» und die Angabe dieser Stärke (die Variable hieß ja auch S) in Prozent. Dazu muß der Wert von S, der ja irgendwo zwischen 0 und 1 liegt, mit 100 multipliziert werden. Schließlich kommt noch ein ⟨CRSR LEFT⟩ wegen der Leerstelle, die nach der Zahl gedruckt wird, und das %-Zeichen. Um den Cursor von dem kritischen Bildschirmeck wegzubekommen, wo er bei ungünstigen Fällen ein Scrolling, also ein Hochschieben unserer Grafik, bewirken könnte, kommt er mit ⟨HOME⟩ (Steuerzeichen dafür ist das invertierte S) wieder in die HOME-Position – bis zum nächstenmal.

Zeile 740: Nachdem wir jetzt endlich alles erledigt haben, springen wir zurück zur Tastaturabfrage in Zeile 550, auf daß alles, was Sie jetzt eingegeben oder zumindest gelesen haben, noch mal passiert. Und noch mal. Und noch mal. Und ...

Zeile 800: Hier gehen jetzt unsere Unterprogramme los, die wir so anspringen können. In dieser Zeile ist quasi die Siegesfeier festgelegt. Hier springt das Programm nämlich hin, wenn Schneewittchen gewonnen hat. Diese gute Nachricht und die Wirkung des eben freigelegten Zauberkristalls wirken wahre Wunder: Alle Zwerge erwachen zu neuen Kräften und rennen zu Schneewittchen. Ihre Y-Koordinate wird also in einer Schleife von 54 auf

200 hochgezählt. Ein Schritt entspricht dabei fünf Bildpunkten. Abwechselnd (das A=1−A kennen Sie ja schon ...) wird eine von zwei Bewegungsphasen gedruckt. Das Prinzip ist dasselbe wie in Zeile 560.

Zeile 810: Das Ganze wird mit einem entsprechenden Ton versehen. Er läuft über Stimme #2, die wir ja für solche Anlässe schon in Zeile 540 vorbereitet haben. Seine Wellenform ist Dreieck, und so wird 33 in die Adresse 54283 gePOKEd, womit der Ton auch gleichzeitig eingeschaltet wird. Nun lassen wir die Zwerge vor Freude springen. Acht Sprünge sind angesagt. Die J-Schleife («J» für engl. jump) wird sie zählen. Die X-Schleife wird vier Töne mit den Frequenzwerten 24 * 256 (= 6144; ca. 350 Hertz), 27 * 256 (= 6912; ca. 400 Hertz), 30 * 256 (= 7680; ca. 450 Hertz) und 33 * 256 (= 8448; ca. 500 Hertz) erzeugen. Musikfreunde mögen uns beim Blick auf die Frequenzen verzeihen. Diese Freudenhüpfer erheben keinen Anspruch auf Perfektion im Sinne eines Dreiklangs in Dur. Wir haben übrigens bloß die High Bytes gePOKEd, die Low Bytes setzen wir auf 0 und können sie so vernachlässigen.

Zeile 820: In der Z-Schleife wird den sieben Zwergen je eine neue Y-Koordinate zugewiesen. So entsteht ein Sprung nach oben. Nächster Zwerg: nächste Koordinate: nächster Sprung – oder auf BASIC: NEXT Z: NEXT X: NEXT J. Ton aus. Zwerge ab von der Bühne (POKE 54283,0: POKE 53269,0).

Zeile 830: Um den Spieler auch textlich entsprechend belohnen zu können, brauchen wir mit POKE 53272,23 unseren Text-Zeichensatz aus dem ROM. Die Steuerzeichen, die nun gedruckt werden, stehen für ⟨SHIFT⟩-⟨CLR/HOME⟩, ⟨C=⟩-⟨6⟩ (Hellgrün) und dreimal ⟨CRSR DOWN⟩. Der nun folgende Text ist im Listing im Grafikmodus abgedruckt, so daß Großbuchstaben als Grafikzeichen dargestellt werden. Das schien uns dennoch vernünftiger, als das ganze Listing in Kleinschrift zu drucken, wobei z. B. völlig andere Steuerzeichen rausgekommen wären. Der Text hier heißt: «Bravo! Es ist Ihnen gelungen,»

Zeile 840: «Schneewittchen zu retten!!!»

Zeile 850: «⟨CRSR DOWN⟩ Neues Spiel gefaellig?»

Zeile 860: «⟨2 * CRSR DOWN⟩ ⟨RVS ON⟩ ⟨CTRL⟩-⟨4⟩ (Hellblau). Druecken Sie eine Taste! ⟨RVS OFF⟩»

Zeile 870: Diese Zeile wartet, bis eine Taste gedrückt wird. Die Funktion der Adresse 198 finden Sie ausführlich im Input/Output-Kapitel. Wenn eine Taste gedrückt wurde, fängt das Spiel mit RUN neu an.

Zeile 900: Wenn das Programm allerdings hierher kommt, gibt es weniger zu feiern. Schneewittchen hat verloren. Das wird zunächst mit einer zum traurigen Anlaß passenden Bildschirmfarbe unterstrichen (POKE53280,0: POKE 53281,0 = Schwarz). Der Bildschirm wird gelöscht (invertiertes Herzchen), alle Sprites außer dem todkranken Schneewittchen verschwinden vom Schirm, und Schneewittchen wird mit den Koordinaten 235 und 150 auf den Schirm gesetzt. Mit ihren schwarzen Haaren auf schwarzem Grund sieht sie schon gespenstisch aus ...

Zeile 910: Für einen Trauermarsch war hier leider kein Platz, aber einen Ton des Beileids kann sich SID einfach nicht verkneifen. Dazu wird die Wellenform der Stimme #2 auf Dreieck geschaltet, der Ton aktiviert und mit einer FOR...NEXT-Schleife die Frequenzen von 8192 bis 4096 (entspricht ca. 500 bis 250 Hertz) abwärts durchlaufen. Das Low Byte schenken wir uns wieder. Danach wird mit POKE 54283,0 der Ton abgeschaltet, und mit POKE 53272,23 wird der Commodore in den Textmodus zurückgesetzt.

Zeile 920: Auch diesmal erhalten wir eine Mitteilung über unser Können. Wie ab Zeile 800 gibt es hier verschiedene Steuerzeichen. Der Text im Listing ist im Grafikmodus gedruckt. Sie geben einfach ein: PRINT «⟨CTRL⟩-⟨6⟩ Es ist Ihnen leider auch nicht gelungen,»

Zeile 930: «Schneewittchen zu retten.»

Zeile 940: «⟨CRSR DOWN⟩ Jetzt koennen wir nur noch auf den»

Zeile 950: «⟨CRSR DOWN⟩ Prinzen hoffen ...»

Zeile 960: «⟨3 * CRSR DOWN⟩ Wollen Sie's nochmal versuchen?»

Zeile 970: «⟨3 * CRSR DOWN⟩ ⟨RVS ON⟩ ⟨CTRL⟩-⟨3⟩ (Hellblau) Druecken Sie eine Taste!»

Zeile 980: Auch diese Zeile wartet, wie Zeile 870, auf einen Tastendruck und startet dann mit RUN das Programm neu.

Zeile 1000: Dies ist das erfreulichere Ton-Unterprogramm. Wenn eine «Wasserübergabe» erfolgreich verlaufen ist, springt das Programm hierher (aus Zeile 660 ...). Stimme #1 wird abgeschaltet, Stimme #2 auf Dreiecksschwingung und auf «Ein». Dann wird in der Schleife die Frequenz von 23 * 256 (= 5888; ca. 350 Hertz) bis 32 * 256 (= 8192; ca. 500 Hertz) durchlaufen. Danach wird Stimme #2 wieder abgeschaltet.

Zeile 1010: Das RETURN veranlaßt den Computer, aus diesem Unterprogramm zurückzuspringen.

Zeile 1100: Dieses andere Unterprogramm ist weniger erfreulich. Es wird angesprungen, wenn ein Zwerg erschöpft zusammenbricht. Man kann hier nicht von einem reinen Ton-Unterprogramm reden, da hier auch noch einige Änderungen vorgenommen werden, die für die Steuerung wichtig sind. Erinnern Sie sich noch? Der Zwerg ZN war derjenige, der unterwegs war. Wenn er kein Wasser bekommen hat, fällt er jetzt um. Dazu sind folgende Schritte nötig: Der Spritezeiger des entsprechenden Sprites zeigt auf Spritenummer 11. Da liegt unser Bitmuster für einen erschöpften Zwerg. Die Anzahl der aktiven Zwerge Z wird um 1 vermindert. Da die anderen nun länger schuften müssen, erhöht sich die Zeit, die Schneewittchen durchhalten muß. R wird proportional zur Anzahl der erschöpften Sprites vermehrt.

Zeile 1110: Hier gibt nun noch SID seinen Kommentar zum Geschehen. Er schaltet Stimme #1 aus und #2 mit einer Dreiecksschwingung an. Die Frequenz wird abwärts durchlaufen, von 32 * 256 (= 81921; ca. 500 Hertz) bis 16 * 256 (= 4096; ca. 250 Hertz). So entsteht ein entsprechend mitleidiger Ton. Schließlich wird Stimme #2 abgeschaltet, und in ...

Zeile 1120: erfolgt der Rücksprung ins Hauptprogramm.

Damit hätten wir auch dieses Spiel geschafft. Fürs Eintippen gilt dasselbe, was wir auch schon beim ersten Spiel gesagt haben. Da dieses Programm besonders viele POKEs verwendet, ist hier beim Testen ganz besondere Vorsicht geboten.

Ansonsten ist auch hier die Anleitung zum Spielen nicht sonderlich schwer: Schneewittchen wird mit den beiden ⟨CRSR⟩-Tasten bewegt. Sie bekommt Wasser, wenn sie den Brunnen berührt, man erkennt das an der blauen Rahmenfarbe. Ein Zwerg bekommt Wasser, indem er Schneewittchen berührt. Das Ganze muß durchgestanden werden, bis die Zeit, unten rechts angezeigt, abgelaufen ist.

Viel Spaß beim Spielen!

```
10 POKE53280,11:POKE53281,11:POKE53265,91:POKE53283,2:POKE53272,21
20 PRINTCHR$(8)
30 PRINT"■■■■■■■■■■■■■■  SCHNEEWITTCHEN UND DIE SIEBEN ZWERGE  "
40 FORX=0TO62:READW:POKE704+X,W:NEXT
50 FORX=0TO62:READW:POKE832+X,W:NEXT
60 FORX=0TO62:READW:POKE896+X,W:NEXT
70 FORX=0TO62:READW:POKE960+X,W:NEXT
80 POKE53269,255:POKE53271,1:POKE53276,255:POKE53285,8:POKE53286,0
90 POKE53287,3:FORX=53288TO53294:POKEX,2:NEXT
100 POKE2040,13:FORX=2041TO2047:POKEX,14:NEXT
110 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,32,0,0,40,0,0,168,0,0
120 DATA252,0,0,84,0,0,16,0,2,168,0,10,170,0,4,168,128,48,252
130 DATA112, 50,170,48,53,85,112,53,69,112,0,0,0,0,0,0,0,0,0
140 DATA 3,240,0,3,80,0,3,80,0,15,80,0,15,144,0,14,160
150 DATA 0,14,168,0,14,168,0,2,160,240,2,170,112,2,170,64
160 DATA 10,160,0,10,168,0,10,168,0,10,168,0,10,168,0,10
170 DATA168,0,3,12,0,3,207,0,0,0,0,0,0,0
180 DATA 0,0,0,0,32,0,0,40,0,0,168,0,4,252,0,8,84,64
190 DATA 8,16,128,10,168,128,0,170,128,0,168,0,0,252,0,2
200 DATA170,0,2,132,0,0,68,0,0,69,0,1,79,192,15,207
210 DATA192,15,192,0,0,0,0,0,0,0,0,0,0
220 DATA0,0,0,0,32,0,0,40,0,0,168,0,0,252,64,4,84,128
230 DATA8,16,128,8,170,128,10,168,0,0,168,0,0,252,0,2
240 DATA170,0,0,74,0,0,68,0,1,68,0,15,197,0,15,207,192
250 DATA0,15,192,0,0,0,0,0,0,0,0,0,0
260 FORX=8TO79:READW:POKE14336+X,W:NEXT
270 POKE56334,0:POKE1,51:FORX=384TO471:POKE14336+X,PEEK(53248+X):NEXT
280 FORX=152TO159:POKE14336+X,PEEK(53248+X):NEXT
290 FORX=208TO215:POKE14336+X,PEEK(53248+X):NEXT
300 FORX=296TO303:POKE14336+X,PEEK(53248+X):NEXT
310 FORX=256TO263:POKE14336+X,0:NEXT
```

```
320 POKE1,55:POKE56334,1:POKE53272,31
330 DATA255,183,183,207,255,228,241,255
340 DATA240,224,240,248,240,224,224,240
350 DATA15,31,31,15,7,15,15,7
360 DATA0,112,112,95,114,82,119,87
370 DATA0,56,56,238,59,40,56,40
380 DATA119,87,112,80,112,255,137,255
390 DATA56,40,56,40,56,254,146,254
400 DATA164,255,146,255,164,255,0,0
410 DATA74,254,146,254,74,254,0,0
420 POKE53265,27
430 PRINT"■■CAAAAABCAABCAAAAAAAAABCAAAABCAAAAAAAAAAAB";
440 FORX=1TO16
450 PRINT"AAAAB   CB  CAB  CAB   CB  CAB  CAB  CAA";
460 NEXTX
470 PRINT"■■■■DE■■■FG■■■HI"
480 FORX=1TO7:POKE53249+X*2,54:NEXT
490 FORX=1TO7:READW:POKE53248+X*2,W:NEXT:POKE53264,192
500 DATA64,101,141,184,220,5,45
510 POKE53248,30:POKE53249,200
520 S=1:POKE53278,0:Z=7:TI$="000000":R=240
530 POKE54296,9:POKE54272,161:POKE53273,8:POKE54277,16:POKE54278,55
540 POKE54284,90:POKE54285,236
550 T=PEEK(203)
560 A=1-A:POKE54276,129:FORI=1TO7:IFZ(I)=0THENPOKE2040+I,14+A
570 NEXT:S=S-.0013:B=0:POKE54276,0:IFS<0THENS=0
580 IFT=7THENB=-7*S
590 IFT=2THENB=8*S
600 X=INT(PEEK(53248)+B):IFPEEK(53264)=193THENX=X+256
610 IFX<22THENX=22
620 IFX<256THENPOKE53264,192:V=X
```

```
630 IFX>255THENPOKE53264,193:V=X-256:IFV>70THENV=70
640 POKE53248,V
650 POKE53279,0:IFPEEK(53279)=1THENWA=1:POKE53280,6
660 POKE53278,0:IFPEEK(53278)<>0ANDWA=1THENWA=0:POKE53280,11:OK=1:GOSUB1000
670 IFZW=0THENZN=INT(7*RND(1))+1:ZW=1:W=5:OK=0:IFZ(ZN)=1THENZW=0:GOTO550
680 POKE53249+ZN*2,PEEK(53249+ZN*2)+W
690 IFPEEK(53249+ZN*2)>200THENW=-5:D=1
700 IFD=1ANDPEEK(53249+ZN*2)<55THEND=0:ZW=0:IFOK=0THENZ(ZN)=1:GOSUB1100
710 IFZ=0THEN900
720 P=INT(R-(TI/60)):IFP<1THEN800
730 POKE211,24:POKE214,24:SYS58732:PRINT"■Z:"P"■■ S:"INT(S*100)"■■ % ■";
740 GOTO550
800 FORX=54TO200STEP5:A=1-A:FORZ=1TO7:POKE53249+Z*2,X:POKE2040+Z,14+A:NEXT:NEXT
810 POKE54283,33:FORJ=1TO8:FORX=1TO10STEP3:POKE54280,23+X
820 FORZ=1TO7:POKE53249+Z*2,200-X:NEXTZ:NEXTX:NEXTJ:POKE54283,0:POKE53269,0
830 POKE53272,23:PRINT"■■■■■IRAVO! ¯S IST ¬HNEN GELUNGEN,"
840 PRINT"♦CHNEEWITTCHEN ZU RETTEN!!!"
850 PRINT"■/EUES ♦PIEL GEFAELLIG?"
860 PRINT"■■■▙¯RUECKEN ♦IE EINE |ASTE!■"
870 POKE198,0:WAIT198,1:POKE198,0:RUN
900 POKE53280,0:POKE53281,0:PRINT"■":POKE53269,1:POKE53248,235:POKE53249,150
910 POKE54283,33:FORX=32TO16STEP-.1:POKE54280,X:NEXT:POKE54283,0:POKE53272,23
920 PRINT"■¯S IST ¬HNEN LEIDER AUCH NICHT GELUNGEN,"
930 PRINT"♦CHNEEWITTCHEN ZU RETTEN."
940 PRINT"■´ETZT KOENNEN WIR NUR NOCH AUF DEN"
950 PRINT"■¯RINZEN HOFFEN..."
960 PRINT"■■■OLLEN ♦IE´S NOCH MAL VERSUCHEN?"
970 PRINT"■■■▙¯RUECKEN ♦IE EINE |ASTE!"
980 POKE198,0:WAIT198,1:POKE198,0:RUN
```

```
1000 POKE54276,0:POKE54283,33:FORX=23TO32STEP.2:POKE54280,X:NEXT:POKE54283,0
1010 RETURN
1100 POKE2040+ZN,11:Z=Z-1:R=R+(7-Z)*10
1110 POKE54276,0:POKE54283,33:FORX=32TO16STEP-.3:POKE54280,X:NEXT:POKE54283,0
1120 RETURN

READY.
```

10
Input/Output beim Commodore 64

Rein in die Kartoffeln, raus aus den Kartoffeln

Stellen wir uns vor, wir haben unser Spiel fertiggeschrieben. Stellen wir uns weiterhin vor, wir hätten alles vorbereitet, um daraus einen großartigen Familienabend zu machen. Es ist jetzt all das da, was ein tolles Spiel so haben muß. Und schließlich hat man alle, die wollen (oder auch nicht), zusammengetrommelt, um eine richtige Spielorgie abhalten zu können und dabei die bewundernden Worte der bisherigen Kritiker («Was denn, Du sitzt ja immer noch an diesem Kasten ...» oder «Wir wollten nur höflich darauf aufmerksam machen, daß es halb zwölf ist und Du morgen in die Schule solltest») zu genießen.

Obwohl Ihre Bemühungen bereits so weit fortgeschritten sind, könnte es sein, daß Sie einen wichtigen Umstand vergessen haben. So etwas passiert meistens dann, wenn man sieht, daß alles läuft, und in dieser ersten Euphorie nicht bemerkt, daß man es eben nur sieht. Das Schöne an Spielen ist ja, daß man damit spielen kann. Spielen ist aber, das lernt man schon in der Grundschule, ein Tunwort. Ja, es gehört sogar in die Klasse der Verben. Das lernt man allerdings meist später. Der erste Ausdruck, der wahrscheinlich jedem Gymnasiallehrer oder auch Buchlektor die Haare zu Berge stehen lassen wird, hat einen Vorteil. Er beinhaltet genau das Wort, auf das es ankommt. Spielen ist etwas, das man tut. Deshalb fehlte bisher für unser Spiel, aber auch für andere Anwendungen wie Dateien oder Textverarbeitung, dieses Kapitel, das I/O-Kapitel. I/O, darüber haben wir uns schon einmal unterhalten, als wir über BASIC sprachen, ist die englische Abkürzung, die für die deutsche Abkürzung E/A steht, und das ist die Abkürzung für Eingabe/Ausgabe. Weil wir aber schon erfahrene Programmieranfänger sind

(und weil es so schön klingt), einigen wir uns auf I/O oder bestenfalls auf Input/Output.

Was aber heißt das eigentlich genau? Nun, für Sie war es bisher immer die selbstverständlichste Sache der Welt, daß Sie irgend etwas eingegeben haben und der Commodore dafür irgend etwas ausgab. Wir sind ja schon mal in einem früheren Kapitel darauf eingegangen, daß dazu einige Programme ständig im Computer parat stehen, die dafür sorgen, daß es so ist. Gut, sagen Sie jetzt, das weiß ich ja.

Schon, aber wenn Sie schon mal ein Spiel auf Ihrem Commodore haben laufen lassen, dann haben Sie ja vielleicht gemerkt, daß die Eingabe nicht immer nur über die Tastatur kommt.

Na klar, werden Sie jetzt forsch antworten, weiß ich auch schon. Richtig, aber haben Sie bei diesen Spielen dann schon mal auf die Ausgabe, den Output geachtet?

Wahrscheinlich erreicht die Diskussion dann einen Punkt, an dem Sie etwas unsicher werden.

Ausgabe, nee. Warum Ausgabe?

Sehen Sie, das dachten wir uns. Aber wer, glauben Sie, bewegt das Raumschiff, mit dem Sie Feinde abschießen?

Ich, oder?

Nein, eigentlich nicht. Sie bewegen nur ein Eingabegerät, auch Input-Device genannt. Die Ausgabe, das ist übrigens das sich bewegende Raumschiff, macht der Computer.

Was wir mit dieser kleinen Verunsicherung bezwecken wollten, war eigentlich nur, Ihnen klarzumachen, daß Input/Output sich keineswegs nur auf Text beschränkt und genausowenig nur über die Tastatur stattfinden muß. Auch der Output muß nicht immer über den Bildschirm erfolgen. Es kann ja auch der Drucker sein oder der Kassettenrecorder und natürlich auch das Diskettenlaufwerk. Diese Geräte werden dann auch als Output-Device bezeichnet.

Deshalb auch das eventuelle Problem mit Ihrem geplanten Spieltestabend. Das Spielen wird allen Beteiligten mehr Spaß machen, wenn sie auch was damit zu tun haben. Deshalb sollten Sie sich mit diesem Kapitel ausgiebig beschäftigen. Eine gute I/O-Routine in einem Spiel macht schon den halben Erfolg. Wenn sie nämlich sehr langsam ist, diese Routine, dann hat das den Nachteil, daß auch das Spiel sehr langsam wird. Wir werden uns also im folgenden mit mehreren Input-Gerätschaften befassen, zuerst mit dem Keyboard. Mit ihm sind sie ja (hoffentlich) im Verlauf des Buches schon sehr oft umgegangen. Weiterhin erwarten Sie in diesem Kapitel noch Geräte, die man

meist mit Händen und Füßen traktieren kann, um dem Computer klarzumachen, was Sie von ihm wollen. Nur damit Sie die Namen schon mal gehört haben, hier ein kurzer Überblick mit ein bißchen Information zum Kauf von solchen Dingen.

Ein kleiner Leitfaden zum Einkauf

Da wären mal die Joysticks. Wie der Name schon sagt, sollen sie Freude (= engl. joy) bereiten. Die Menge dieser Freude ist direkt abhängig von der Leichtigkeit, mit der man diese Steuerknüppel handhaben kann. Deshalb sollten Sie sie grundsätzlich, bevor Sie sich zum Kauf eines solchen entschließen, ein bißchen testen. Am besten wäre es, einen neuen und einen bereits im Gebrauch befindlichen zu testen. Daran läßt sich nämlich zum Beispiel sehr gut erkennen, wie es um die Qualität eines Joysticks steht. Ist der gebrauchte schon ziemlich ausgeleiert, sollte man lieber noch einen anderen testen. Denn in den Geschäften, wo meist ein Computer zum Spielen herumsteht, kann man davon ausgehen, daß der Joystick wirklich hart getestet wird. Andererseits sollte der Joystick auch nicht zu hart sein, sonst ermüdet die Hand sehr schnell. Das Spiel macht keinen Spaß mehr, weil man sich mehr auf den Joystick als auf den Feind im Weltraum konzentrieren muß. Wichtig ist auch, wie die Druckknöpfe reagieren. Arbeiten sie zuverlässig? Kann man gut spüren, daß der Kontakt da war? (Das heißt: Merkt man an irgend etwas, daß der Knopf auch tief genug gedrückt wurde?)

Am besten nehmen Sie sich eines Ihrer Lieblingsspiele mit, wenn Sie einen Joystick kaufen, und testen ihn damit. Natürlich ist klar, daß Sie sich zuerst an die neue Form gewöhnen müssen. Deshalb werden Sie nicht sofort einen neuen Rekord aufstellen, aber Sie werden schnell merken, ob Ihnen der neue im wahrsten Sinne des Wortes liegt. Sollte Ihnen das alles ein bißchen übertrieben vorkommen: mag sein. Aber ein passabler Joystick kostet mindestens 35 DM. Und das ist auch Geld. Deshalb ruhig testen. Ein guter Computerladen wird Ihnen sogar selbst dazu raten. Denn ein verärgerter Kunde bringt auch nicht viel.

Das zweite Gerät, das wir besprechen wollen, sind die sogenannten Paddles. Diese «Steuerräder» haben bei manchen Spielen oder auch anderen Anwendungen gegenüber Joysticks doch einige Vorteile. Auch bei den Paddles sollten Sie übrigens vor einem Test nicht zurückschrecken. Hier kommt es in erster Linie darauf an, daß Sie den höchsten Wert, in den der

Computer ihre Stromspannung umwandeln kann (das wäre die Zahl 255), auch mühelos erreichen. Wenn Sie dieses Kapitel gelesen haben, werden Sie sich sehr einfach ein kleines Programm schreiben können, das Ihnen die ausgelesenen Werte anzeigt. Achten sollten Sie auch auf die Druckknöpfe. Sie sollten nicht zu groß und nicht zu klein (meist für den Daumen) sein, und sie müssen, wie die Feuerknöpfe der Joysticks, sauber reagieren. Soviel zum Thema. Zuerst wollen wir uns aber mit der Tastatur, auch Keyboard genannt, beschäftigen. Und wenn Sie jetzt sagen, was kann da schon noch kommen, dann werden Sie aber ganz schön staunen.

Tipp-tipp hurra

«Wir wollen uns jetzt mal mit dem Keyboard beschäftigen ...» Die erste Reaktion bei diesem Satz während unserer Kurse über den Commodore war eigentlich immer die Antwort, daß man ja sehr wohl wisse, wie man mit diesen Tasten umzugehen hätte. Warum also dann eine eigene Unterweisung? Die Antwort ist, daß es schon den einen oder anderen Trick gibt, der Ihnen allerhand Arbeit ersparen kann. Die BASIC-Befehle INPUT und GET sollten Sie bereits kennen. Wenn nicht, hier noch einmal kurz: Mit INPUT lassen sich ganze Zeichenketten (Namen, Telefonnummern u. ä.) abfragen, mit GET dagegen nur einzelne Tasten. Allerdings schert sich das GET einen feuchten Kehricht darum, ob Sie eine Taste gedrückt haben oder nicht. Deshalb muß eine typische GET-Abfrage bisher meist so aussehen:

```
10 PRINT "DRUECKEN SIE EINE TASTE!"
20 GET A$:IF A$="" THEN 20
```

In Zeile 20 lassen Sie also Ihr Programm prüfen, ob sich GET bereits zurückziehen kann oder ob es gefälligst noch warten soll, weil noch gar keine Eingabe stattgefunden hat. Wenn dann eine Taste gedrückt würde, dann erst macht Ihr Programm weiter.

Nach dem Motto «Gefahr erkannt, Gefahr gebannt» machen wir uns erst mal daran, das Problem auszubaldowern.

Schwierigkeit, dein Name ist Tastaturpuffer.

Die Tastaturabfrage wird ja bekanntlich alle ⅙₀ Sekunde während des Interrupts durchgeführt. Bis allerdings das Zeichen oder die Funktion, die zur entsprechenden Taste gehört, ausgeführt wird, vergeht einiges an Zeit, in der unser Commodore rechnet, vergleicht, nachschaut usw. Natürlich geht das

alles verhältnismäßig schnell, aber bei der Konzeption seiner Computer hat Commodore wohl einen Test mit einer ausgebildeten Sekretärin gemacht und festgestellt, daß diese Dame mit dem Tippen ziemlich fix war, der Commodore mit der Zeichendarstellung allerdings ein paar Hundertstel hinterherhinkte.

Ergebnis dieses Vorgangs (von dem wir leider nicht prüfen konnten, ob er so stattfand) war offensichtlich das, was wir jetzt unter dem Namen Tastaturpuffer besprechen. Dort werden bis zu 10 Zeichen, die schneller eingetippt werden, als der Computer schreiben kann, zwischengespeichert. So gehen sicher keine Buchstaben verloren. Nun ist dieser Tastaturpuffer aber nicht etwa nur dann in Betrieb, wenn der Commodore um Hilfe brüllt, sondern ständig bereit, alles, was auf der Tastatur passiert, zwischenzuspeichern, und wenn es soweit ist, dem Prozessor Bescheid zu stoßen, daß da wohl Post für ihn wäre. Die Geschehnisse eines Tastendrucks lassen sich also in etwa folgendermaßen rekonstruieren: Während des Interrupts stellt der Computer fest, daß eine Taste gedrückt wurde. Den Code dieser Taste legt er im Tastaturpuffer ab. Aber eine Taste kommt selten allein, also wird alles, was sonst noch anfällt, ebenfalls im Tastaturpuffer abgelegt. Solange genug Zeit ist, kann sich das Betriebssystem darum kümmern, was mit dem nächsten Zeichen geschehen soll. Deshalb liest es zunächst den ersten Wert des Tastaturpuffers. Und das ist ja auch die Taste, die zuerst gedrückt wurde. Die Taste wird übersetzt und entsprechend der Bedeutung weitergeleitet. Hier verlieren sich unsere Beobachtungen der Taste erst mal in den Tiefen des Speichers.

Nun kann man, um nachzuprüfen, ob das alles auch wirklich so vor sich geht, folgende kleine Zeile im Direktmodus (also keine Zeilennummer davor) eingeben:

```
FOR X=1TO5000:NEXT
```

Diese Zeile ist nichts weiter als eine ordinäre Warteschleife. Aber sie hat einen Sinn. Sie beschäftigt unseren Commodore genauso wie uns ein spannendes Buch. Das liegt daran, daß der Computer sowieso nur Spannung im Strom finden kann. Wenn Sie jetzt ein READY auf dem Bildschirm sehen, dann ist der Computer bereits fertig – während Sie diese Zeilen gelesen haben. Sollte das so sein, geben Sie die Schleife bitte noch mal ein. Und gleich, nachdem Sie ⟨RETURN⟩ getippt haben, geben Sie noch irgend etwas ein, z. B. Ihren Namen. Sie haben rund 5 Sekunden Zeit. Dalli – Dalli ...

Wenn der Computer jetzt sein READY von sich gegeben hat, müßten Sie noch einige andere Zeichen darunter sehen, die – mehr oder weniger

verstümmelt – Ihren Text, den Sie tippten, widerspiegeln sollten. Wenn Sie uns geglaubt haben, was wir Ihnen erzählten, ist das auch nur logisch. Wenn er sie auch nicht sofort ausgeben konnte – er war ja beschäftigt –, Ihr Commodore hat das Drücken der Tasten sehr wohl bemerkt. Und kaum hat er seine stupide Zählerei fertig gehabt, hat er das Versäumte schleunigst nachgeholt.

Und von diesem Wissen können wir jetzt profitieren. Dazu erst einmal wieder ein paar POKE-Adressen. (Wenn Sie in letzter Zeit mit dem Gedanken spielten, sich ein Adreßbuch für Ihren Computer zu kaufen, dann lassen Sie es bleiben und denken Sie getrost an unseren PEEK&POKE-Anhang. Er enthält noch einmal alle wichtigen Adressen.)

Im PEEK (649) steht die Anzahl der Zeichen, die der Tastaturpuffer maximal fassen kann. Ein

```
PRINT PEEK (649)
```

ergibt konsequenterweise die Zahl 10. Diesen Wert können Sie verändern. Bevor Sie jetzt aber gleich POKE-Werte um 255 herum eingeben und versuchen, eine Adressensammlung im Tastaturpuffer anzulegen, lassen Sie sich gewarnt sein. Der Puffer liegt nämlich noch sehr nahe an der Zeropage. Und die vorhergehenden Kapitel müßten Ihnen klargemacht haben, was passiert, wenn Sie hier wichtige Speicherzellen mit Tastaturcodes überschreiben. Sie können jetzt, am besten unter der Verwendung der Warteschleife von vorhin, mit diesen Werten etwas herumexperimentieren.

Wenn Sie aber

```
POKE 649,0
```

eingeben, dann gibt es gar keinen Tastaturpuffer mehr. Das Ergebnis können Sie sich wahrscheinlich vorstellen. Wenn Sie es trotzdem ausprobieren: Die Kombination ⟨RUN/STOP⟩ + ⟨RESTORE⟩ befreit Sie und Ihren Commodore aus dieser mißlichen Lage.

Wollen Sie also zum Beispiel im Rahmen eines Spiels die möglichen Tasten, die zwischengespeichert werden können, auf eine bestimmte Zahl begrenzen – jetzt wissen Sie ja, wie es geht.

Aber noch viel interessanter ist die nächste Adresse: PEEK(198) enthält die Anzahl der Tasten, die tatsächlich gerade im Tastaturpuffer stehen. Wenn Sie, während das kleine Programm

```
FOR X=1TO2000:PRINT PEEK(198):NEXT
```

läuft, einige Tasten drücken, werden Sie sehen, wie dieser Wert ansteigt

Natürlich können Sie diesen Wert auch jederzeit auf 0 zurücksetzen. Dann wird der Tastaturpuffer gelöscht:

```
FOR X=1TO5000:NEXT:POKE198,0
```

Versuchen Sie jetzt mal nach der Methode von vorhin, Text in den Computer zu kriegen ... Unsere GET-Abfrage von vorhin ließe sich damit auch etwas eleganter programmieren:

```
10 PRINT "DRUECKEN SIE EINE TASTE!":POKE198,0
20 IF PEEK(198)=0 THEN 20
```

Solange der Tastaturpuffer leer ist, bleibt das Programm in der Zeile 20. Wenn Sie jedoch irgendeine Taste drücken, geht das Programm über zum nächsten Schritt. In diesem Fall würde es sich mit READY melden. Die Taste, die Sie gedrückt haben, steht aber dennoch im Tastaturpuffer. Es hat sie ja bisher noch keiner dort rausgeholt, und sie erscheint schließlich auf dem Schirm.

Soll also beispielsweise eine ganz bestimmte Taste gedrückt werden, so läßt sich das mit Hilfe des WAIT-Befehls besonders schön machen:

```
10 POKE 198,0:WAIT 198,1:GET A$
20 IF A$<>"A" THEN 10
```

Das Programm fährt nur dann fort, wenn Sie die Taste ⟨A⟩ drücken. Zuerst einmal wartet es allerdings in Zeile 10 so lange, bis eine Taste gedrückt wird – also bis in der Speicherzelle 198 der Wert 1 auftaucht. Durch den Befehl GET bekommt dann die Variable A$ das Zeichen der entsprechenden Taste zugewiesen. Zeile 20 überprüft dann, ob die gedrückte Taste ein A war. Wenn nicht, dann geht das Programm wieder zur Zeile 10 und wartet. Sie können das gern ausprobieren – vielleicht, indem Sie in der Zwischenzeit einen Kaffee trinken gehen. Natürlich, werden nun die kritischen Leute sagen, das kann man mit GET allein aber auch machen. Stimmt. Zwar nicht so elegant, aber es stimmt. Doch abwarten. Der Tastaturpuffer hat noch mehr Überraschungen auf Lager.

Um die kennenzulernen, müssen wir Ihnen zuerst verraten, wo der Puffer eigentlich liegt. Er befindet sich sehr nahe an der Zeropage. Das hat den Vorteil, daß das Betriebssystem sehr schnell an ihn rankommt. Und das ist ja wichtig. Die Speicherzellen gehen von 631 bis 640. Aber da es sich hierbei um ganz normalen RAM-Bereich handelt, hindert Sie niemand daran, hier auch Werte hineinzuPOKEn. Probieren Sie es aus.

```
10 FOR X=0TO9:POKE631+X,211:NEXT
20 POKE 198,10
```

In Zeile 10 schreiben wir den Code 211 in den Tastaturpuffer. Welches Zeichen das ist, werden Sie sehr bald merken, wenn Sie unser Beispiel ausprobieren. Zeile 20 teilt Ihrem Computer dann mit, wie viele Zeichen im Puffer stehen. Da wir ihn ganz aufgefüllt haben, POKEn wir 10.

Diese kleine Spielerei hat Ihnen gezeigt, wie man den Computer 10 Zeichen «vorprogrammieren» kann, während er beschäftigt ist. Alles, was wir ihm auf diese Art und Weise vorher eingeben, führt er dann im Direktmodus aus. So kann man ganze neue Programmzeilen von einem Programm selbst einfügen lassen.

```
10 PRINT"■■■■20REM DIES IST ZEILE 20"
30 POKE631,13:POKE198,1:PRINT"■■";

READY.
```

Zeile 20 wird gedruckt und durch den Code 13 (das entspricht der Taste ⟨RETURN⟩) dann auch wirklich eingegeben. Die Codes, die Sie POKEn müssen, um solche oder ähnliche Effekte zu erzielen, finden Sie wieder in Ihrem Commodore-Handbuch auf der Seite 135. Es handelt sich um die ASCII-Codes.

Passen Sie übrigens bei dem oberen Beispiel auf, daß die Cursorzeichen (= Steuerzeichen) genau stimmen: sie sorgen dafür, daß der Cursor auf der Zeile 20 steht, wenn das ⟨RETURN⟩ durchgeführt wird. Das wird erreicht durch die Zeichen «HOME» und «CURSOR DOWN» (invertiertes S und invertiertes Q).

Auf dieselbe Art können Sie auch mehrere Zeilen eingeben und/oder dann im Programm weitermachen, indem Sie zum Beispiel ein RUN oder GOTO drucken und durch ein ⟨RETURN⟩ im Puffer ausführen lassen. Jetzt sollten Sie erst mal ein bißchen mit diesen neuerworbenen Kenntnissen experimentieren, bevor Sie weiterlesen. Dann fällt meist auch das Verstehen leichter.

Manchmal könnte es ja sein, daß Sie genau das Gegenteil von dem wollen, was wir jetzt besprochen haben: Eine Taste soll nicht zwischengespeichert werden, sondern es soll sofort geprüft werden, welche es war. Auch dafür gibt es eine Speicherzelle – nein, sogar zwei Speicherzellen. Sowohl in der Adresse 197 wie auch in der Adresse 203 steht genau dasselbe.

Lassen Sie zu Demonstrationszwecken das folgende Miniprogramm über längere Zeit laufen, und drücken Sie dabei einige Tasten. Sie werden sehen, wie sich die Werte ändern.

```
10 PRINT PEEK(197):RUN
```

Sie werden andauernd neue Zahlenwerte auf den Bildschirm bekommen. Solange Sie als Wert 64 erhalten, heißt das, daß Sie gar keine Taste drücken (obwohl wir doch sagten, daß Sie das sollen!!). Alle anderen Tasten, außer ⟨SHIFT⟩, ⟨C=⟩ und ⟨CTRL⟩, haben offensichtlich einen ganz bestimmten Wert. Diese Werte können Sie zum Beispiel durch Ausprobieren herausbekommen, wir haben Ihnen aber eine Zeichnung der Tastatur angefertigt, auf der alle Werte stehen.

Wenn Sie diese Speicherzelle überprüfen, können Sie jederzeit feststellen, welche Taste gedrückt wurde, unabhängig vom Tastataturpuffer. Und jetzt kommen wir langsam zu den Spielen: Sie können zum Beispiel eine Spielfigur mit den Tasten Z und C steuern lassen. Und die Abfrageroutine dazu kann den BASIC-Befehl GET umgehen, ist sehr schnell und einfach zu programmieren. Wenn der PEEK(197) = 12 ist, dann bewegen Sie die Figur nach links. Ist der PEEK(197) = 20, dann eben nach rechts. Sie können also einzelnen Tasten, wie beim Joystick bestimmte Stellungen, ganz bestimmte Richtungen zuweisen. Wie das beim Joystick geht, kommt noch.

Kommen wir schließlich zum letzten Punkt. Und weil bekanntlich das, was nicht so auf Anhieb geht, das Interessanteste ist, wollen wir Ihnen zeigen, wie man ⟨CTRL⟩, ⟨SHIFT⟩ oder ⟨C=⟩ doch abfragen kann. Diese drei Tasten haben nämlich, aufgrund ihrer Sonderfunktion, eine eigene Speicherzelle bekommen, und zwar die Nummer 653. Dabei entsprechen folgende Werte in dieser Zelle folgenden Tasten (Tabelle 10.1).
Tastenkombinationen ergeben dann die Summen der Einzelwerte, ⟨SHIFT⟩ & ⟨C=⟩ zum Beispiel 3 oder alle zusammen 7. (Wer diese Zahl erreicht, muß aber schon sehr aufpassen, daß er sich nicht die Finger verrenkt.)

Wert	Taste
1	⟨SHIFT⟩
2	⟨C=⟩
4	⟨CTRL⟩

Tabelle 10.1 Werte in Speicherzelle 653

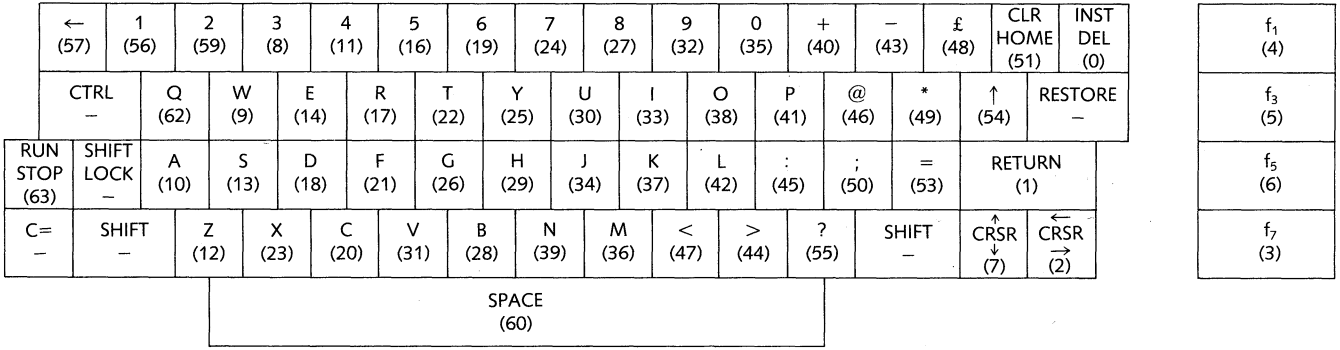


Bild 10.1 Tastatur und Codes in PEEK (197)

Wenn Sie also alles, was Sie bis jetzt gelernt haben, berücksichtigen und eine Programmzeile schreiben, die wartet, bis jemand ⟨CTRL⟩ & ⟨SPACE⟩ gleichzeitig drückt, sollte das etwa so aussehen:

```
10 IF PEEK(653)<>4  OR PEEK(197)<>60 THEN 10
```

oder auch kürzer

```
10 WAIT 653,4:WAIT 197,60
```

So, damit hätten wir so im großen und ganzen alles, was man mit der Tastatur (legal!!!) machen kann. Bedenken Sie aber beim Programmieren, daß nicht alles, was machbar ist, auch unbedingt sein muß. Wenn Sie zum Beispiel ein Textverarbeitungsprogramm schreiben, gereicht Ihnen eine Tastenkombination ⟨CTRL⟩ & ⟨SHIFT⟩ & ⟨S⟩ & ⟨RETURN⟩ als «Abkürzung» für den Befehl «Text schreiben» sicherlich nicht zur Ehre.

Jetzt kommt es knüppeldick

Irgendwann kamen einige schlaue Leute darauf, daß es auf die Dauer reichlich mühselig ist, immer «E» für Raumschiff hoch und «X» für Raumschiff runter zu drücken. Dann setzten sich diese Leute in ein kleines Hinterzimmer und bastelten und werkelten, bis sie voller Stolz den ersten Joystick gebaut hatten. Diese Leute wurden zwar nicht berühmt (es gibt niemanden, der Herr Joystick heißt und nach dem diese Dinger benannt worden sind), aber dafür konnten sie sich bald ein schönes, großes Auto kaufen, weil so ein Steuerknüppel auch anderen Leuten, die nicht so geschickt im Basteln waren, gefiel und sie sich deshalb einen bauen ließen. Bald erkannten viele Leute, die im Basteln geschickt waren, daß sich mit der Herstellung von Joysticks eine Familie ganz gut ernähren läßt. Das Ergebnis waren Joystickfabriken. Damit aber auch jeder genau den Joystick kaufte, den man selbst verkaufen wollte, machte man furchtbar viele Formen, bei denen jeder behaupten konnte, seine sei die beste. Kurz und gut: Es gibt heute unheimlich viele Arten von Joysticks. Und an dem Gerede von der besseren Form ist auch sicher was dran. Warum aber alle nur über die Form reden, liegt daran, daß praktisch alle Joysticks nach demselben Prinzip funktionieren. Und weil man in der Werbung schlecht über das reden kann, was der andere auch hat, macht man eine neue Form, und schwups hat man etwas, worüber man reden kann.

Ehrlicherweise muß man aber tatsächlich zugeben, daß von der Form dieser Geräte schon allerhand abhängt. Für das allgemeine Hobbyabend-Telespiel reichen die relativ einfachen Joysticks aus. Sie kosten zwischen 30 und 50 DM. Das ist zwar auch nicht sooo wenig, aber immer noch billiger, als die Tastatur nach ein paar Stunden Pac-Man zu ersetzen.

Bei Ihrem Commodore 64 kann man gleichzeitig zwei davon anschließen (Joysticks, nicht Pac-Man). Dazu befinden sich an der rechten Seite zwei Stecker, die den hübschen Namen «Control Port 1» und «Control Port 2» tragen. Wie zwei eineiige Zwillinge glänzen die beiden da an der Seite. Der Vorteil ist natürlich, daß man zwei Joysticks anschließen kann. Der Nachteil ist allerdings, daß man zwei Joysticks anschließen kann. Das heißt eigentlich, daß man denselben Joystick an jeden der beiden Control Ports anschließen kann. Leider haben sich weder die Hobbyprogrammierer noch die Softwarehäuser für einen der beiden entschließen können. Die einzige allgemeingültige Regel, die man aufstellen kann, ist die: Schließt man einen Joystick an einen Port an und startet das Spiel, so wäre mit 90prozentiger Sicherheit der andere Port der richtige gewesen.

Deshalb wollen wir die günstige Gelegenheit nutzen und hier zu zweit an die Programmierer der Welt plädieren: Einigt euch, im Namen der Single-Joystick-Besitzer, auf einen Port.

Und nach diesem Aufruf können wir zum eigentlichen Thema kommen: Wenn man selbst programmiert, wie nutzt man dann die Joysticks? Erst einmal gibt es (selbstverständlich) eine Adresse, aus der man den Wert, den ein Joystick gerade durch seine Position in den Computer einspeist, herauslesen kann. Es sind die Adressen 56320 für den Port 2 und 56321 für den Port 1.

Wenn Sie einen Joystick haben, geben Sie bitte folgendes Programm ein:

```
10 PRINT PEEK(56321):RUN
```

Wenn Sie jetzt verschiedene Joystickstellungen durchprobieren, werden Sie sehen, wie sich die Werte auf dem Bildschirm ändern. Dazu gleich eine Anmerkung: Wenn Sie Ihren Joystick nach links drücken, werden Sie die Zahl 251 sehen, gleichzeitig werden die Zahlen langsamer über den Bildschirm rollen (man nennt diesen Vorgang des Rollens auch *Scrolling).* Wenn Sie sich erinnern können, haben wir einmal erwähnt, daß derselbe Effekt auch dann auftritt, wenn man die ⟨CTRL⟩-Taste gedrückt hält.

Probieren Sie doch auch das jetzt einmal aus. Sie werden sehen, daß sich die Zahl wieder ändert, auch wenn Sie den Joystick in seine Ruhestellung

zurückkehren lassen. Wir wollen Ihnen kurz erklären, warum das so ist: Der Joystick in Port 1 läuft über den gleichen I/O-Baustein, an dem auch die Tastatur angeschlossen ist. Deshalb gibt es einige Tasten, die den gleichen Effekt haben wie ein angeschlossener Joystick. Umgekehrt ist es deshalb auch nicht zu empfehlen, den Joystick 1 während des Programmierens zu betätigen. In Programmen, die Sie schreiben und die sowohl den Joystick 1 als auch die Tastatur benutzen, empfiehlt es sich, vor jedem INPUT oder GET mit POKE 198,0 den Tastaturpuffer zu löschen.

Beim Joystick 2 gibt es dieses Problem nicht. Näheres zu der ganzen Sache werden Sie dann noch erfahren, wenn wir uns über die Paddles unterhalten.

Wie sind jetzt aber die PEEK-Werte, die Sie wahrscheinlich immer noch auf Ihrem Bildschirm sehen, zu verstehen?

Wenn Sie Ihrem Joystick eine Ruhepause gönnen und ihn völlig loslassen (was natürlich nicht meint, daß Sie den Armen auf den Boden fallen lassen sollen!!), erscheint die Zahl 255. Und die kennen wir ja mittlerweile zur Genüge. Sie bedeutet, daß alle Bits in der Speicherzelle an sind.

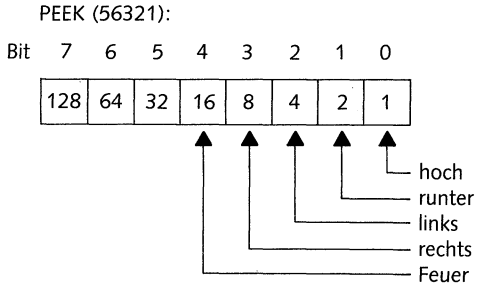
Sollten Sie wider Erwarten doch noch ein gestörtes Verhältnis zu unserer 255 haben, schauen Sie noch mal in der Speicheraufteilung nach. Ergebnis unserer bisherigen Nachforschungen ist also, daß, wenn nichts gedrückt, geschoben oder sonstwie verändert wird, alle Bits an sind. Drücken Sie jetzt den Joystick nach vorne (bzw. nach oben, das kommt ganz darauf an, wie Sie das sehen). Sie sehen, die Zahl ändert sich in 254. Aus der Sicht unseres Commodore heißt das, daß ein einzelnes Bit (nämlich Nummer 0) gelöscht oder «aus» ist.

Wenn Sie jetzt den Knüppel selbst wieder zurück in die Ruheposition lassen, aber dafür den Feuerknopf drücken, erscheint 239. Das heißt für den Computer, daß Bit Nummer 4 (= 16) gelöscht ist.

Und wenn Sie jetzt gleichzeitig noch den Knüppel wieder nach vorne drücken (oder nach oben – ganz wie Sie meinen), erscheint aufs neue eine andere Zahl: 238. Das heißt die Bits 4 und 1 sind soeben gelöscht worden (255 – 16 – 1).

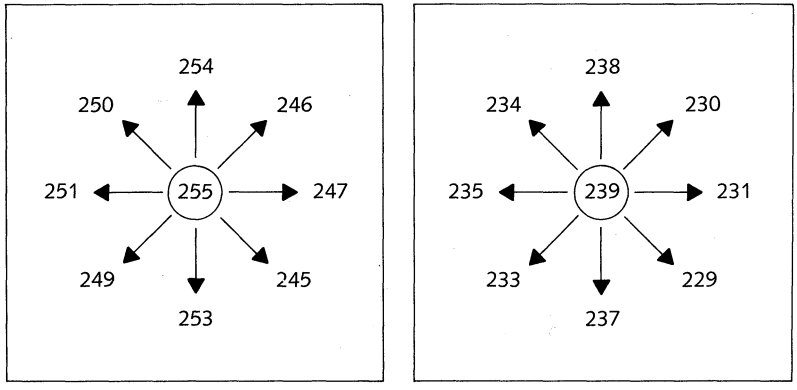
Ganz schön informativ, so eine Knüppelei mit dem eigenen Computer, nicht wahr?

Und damit haben Sie das Prinzip der Joystickabfrage eigentlich schon verstanden. In bestimmten Positionen des Joysticks oder wenn der Feuerknopf gedrückt wird, werden ganz bestimmte Bits gelöscht. Dadurch ergeben sich auch ganz bestimmte Werte in unseren Speicherzellen. Um es anders zu sagen, den einzelnen Bits in der entsprechenden Speicheradresse ist jeweils



Das entsprechende Bit ist jeweils aus!

Joystick 1 (PEEK (56321)):



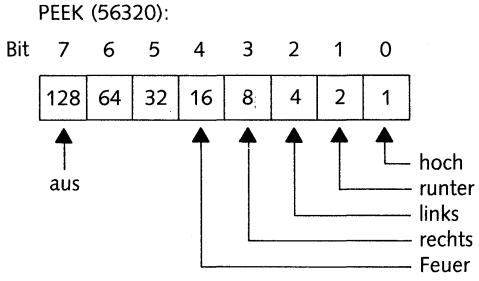
ohne Feuerknopf mit gedrücktem Feuerknopf

Bild 10.2 Joystick 1, Bitbelegung in (56321)

ein «Schalter» im Joystick für vor, zurück, links, rechts und Feuerknopf zugeordnet. Und aus nichts anderem als aus fünf Schaltern besteht ein Joystick. (Verstehen Sie jetzt, warum die immer von der Form sprechen?)

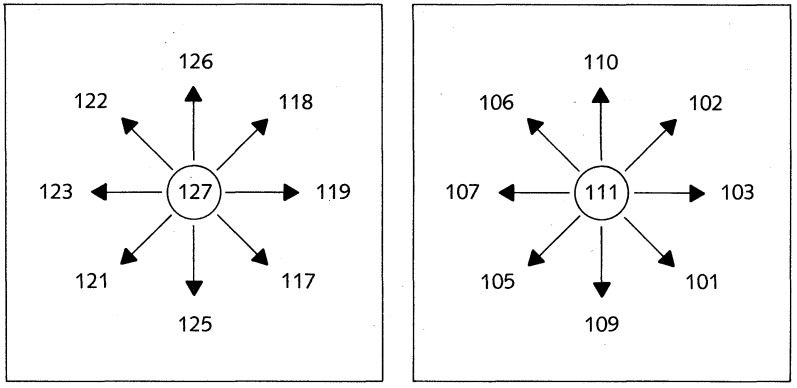
Zwei kleine Skizzen sollen Ihnen helfen, das besser zu verstehen und gleichzeitig eine Wertetabelle für bestimmte Joystickpositionen darstellen. Diese Werte erhalten Sie mit PEEK(56321). Nebenbei, in PEEK(145) steht noch mal dasselbe.

Zur Demonstration noch einmal ein Blick in Byte 56321 (Bild 10.2).



Das entsprechende Bit ist jeweils aus!

Joystick 2 (PEEK (56320))



ohne Feuerknopf mit gedrücktem Feuerknopf

Bild 10.3 Joystick 2, Bitbelegung in (56320)

Fein, denken Sie jetzt, Ende der Fahnenstange, das war es zum Thema Joystick. Der zweite geht ja wohl genauso.

Aber erstens kommt es anders, zweitens als man denkt. So einfach hat es uns Commodore nicht gemacht. Sehen Sie selbst.

```
10 PRINT PEEK(56320):RUN
```

Der Wert für die Grundstellung ist hier 127. Aber keine Angst, im Prinzip ist alles geblieben wie vorher. Der einzige Unterschied ist, daß Bit Nummer 7

(= 128) von vornherein ausgeschaltet bleibt. Und damit kann der höchste Wert in dieser Speicherzelle nicht 255 (wie bei der anderen) sein, sondern 255 – Bit 7. Das heißt 255 – 128. Und das ergibt genau 127. Toll, oder? Unser Joystickmodell sieht hier also folgendermaßen aus (Bild 10.3).
Übrigens, gestatten Sie uns erneut den Hinweis: Wer mit all diesen Bits noch nicht so recht klarkommt, sollte noch einmal das Kapitel Speicheraufteilung auf diesen Teil hin durchlesen.

Stellt sich nur noch die Frage, wie man all das beim Programmieren gut ausnützen kann.

Quo vadis, Joystick?

Quo vadis ist zweierlei. Zum einen eine der bekannten Hollywood-Mammutproduktion, und zum anderen lateinisch. An das erstere erinnert man sich eigentlich ganz gern, an das zweite – na ja. Hier schweigt des Sängers Höflichkeit ...

Trotzdem reicht unser Latein gerade noch so, um diesen Ausdruck zu verwenden. Quo vadis heißt nämlich «Wohin gehst Du». Jeder, der Asterix liest, kann sich vielleicht daran erinnern, daß es da auch vorkommt - allerdings nicht so häufig wie das berühmte «Alea jacta est», was soviel heißt wie «Verdammt, jetzt isses aber zu spät ...». Und genau diese Frage, jetzt reden wir wieder von Quo vadis, stellt natürlich unser Commodore dem Joystick. Schließlich muß er – zwecks Bildschirmdarstellung – wissen, wohin die Reise gehen soll. Um diese Frage a) von einem BASIC-Programm aus stellen zu können und b) auch noch zu beantworten, müssen wir uns wieder der Mithilfe von Herrn Boole versichern (Zwischenspiel 3).

WERT	SCHALTER
1	HOCH
2	RUNTER
4	LINKS
8	RECHTS
16	FEUER

Tabelle 10.2 Joystickwerte

Die Abfrage (dieses Wort erinnert auch immer so extrem an Latein ...) muß im Programm

IF NOT PEEK (Joystick) AND WERT THEN entsprechender Schalter an

heißen. Wir würden Ihnen natürlich nicht raten, das so einzutippen. Sie sollten es eher interpretieren. Das heißt: Für JOYSTICK setzen Sie bitte 56320 oder 56321 (je nachdem ...) ein.

WERT sollte aus Tabelle 10.2 stammen.

Hinter dem THEN sollte der Teil des Programms stehen, der ausgeführt werden soll, wenn der Joystick entsprechend bewegt wurde. Also beispielsweise:

```
10 IF NOT PEEK(56321) AND 16 THEN PRINT "FEUERKNOPF AM
   JOYSTICK 1"
20 RUN
```

Natürlich muß nicht alles, was bei einer bestimmten Position zu machen ist, direkt hinter dem THEN stehen. Hier kann genausogut ein GOTO oder GOSUB stehen. Und wenn wir schon beim Programmieren sind, noch ein kleiner Tip: Angenommen, Sie wollen einen Sprite oder irgendein Zeichen mit dem Joystick steuern. Natürlich sollte das Objekt auch nach oben, wenn Sie den Joystick nach oben bewegen. Um solche oder ähnliche Aufgaben zu bewältigen, benutzen Sie am besten ein Koordinatensystem. Bei den Sprites müssen Sie das ja sowieso. Also legen Sie einen X- und einen Y-Wert fest, den Sie je nach Position des Joysticks erhöhen oder verringern. Weil das alles sehr theoretisch klingt, haben wir ein kleines Beispielprogramm gemacht. Es geht davon aus, daß Sie ab Adresse 832 einen Sprite im Speicher haben. Beispielsweise unser Auto aus dem Sprite-Kapitel:

```
10 POKE2040,13: POKE53269,1: X=174: Y=135: POKE53284,X:
   POKE53249,Y
20 IF NOT PEEK(56321) AND 1 THEN Y=Y-1
30 IF NOT PEEK(56321) AND 2 THEN Y=Y+1
40 IF NOT PEEK(56321) AND 4 THEN X=X-1
50 IF NOT PEEK(56321) AND 8 THEN X=X+1
60 POKE 53248,X:POKE 53249,Y
70 GOTO 20
```

Ist doch gar nicht so schwer, oder? Wenn Ihnen die Bewegung zu langsam ist, ersetzen Sie die Zahl 1 in den Zeilen 20 bis 50 einfach durch eine höhere. Sie könnten aber auch den Feuerknopf als Gaspedal einsetzen ...

Sie sehen, schon wieder hat sich Ihrer Experimentierfreude ein weites Feld erschlossen...

Die Widerstandsbewegung beim Commodore 64

Paddles sind nicht ganz so verbreitet wie Joysticks. Ein bißchen zu Unrecht allerdings, wie wir meinen. Denn Sie können einiges, von dem Joysticks noch nicht einmal träumen.

Der Hauptunterschied zwischen beiden ist der, daß bei den einen das Signal, das sie geben, in guter alter Computermanier einfach nur aus Schalter an oder Schalter aus (dem berühmten 1 und 0) besteht.

Die Paddles dagegen haben ihre eigene Ansicht darüber. Sie kommen sozusagen schon zur Widerstandsbewegung geboren auf die Welt – natürlich nur theoretisch.

Das heißt, sie haben einen Widerstand eingebaut, der jeden Wert von 0 bis 255 zurückgeben kann. Das klingt jetzt zwar völlig unproblematisch, weil 255 ja nun eine alte Bekannte ist, hat aber trotzdem seine Haken und Ösen.

Die Paddles geben natürlich keine Zahlen ins Chipgehirn, sondern verschiedene Stromstärken.

Um aber jetzt diese Stromstärken in computergerechte 0- und 1-Snacks zu zerteilen, bedarf es einiger Hilfsmittel.

Ein solches Hilfsmittel ist ein *A/D-Wandler*. Diese Abkürzung steht selbstverständlich wieder für einen der sinnigen Computerfachausdrücke, und zwar für Analog/Digital-Wandler. Wie der Name schon sagt, wandelt der Wandler Analoges in Digitales. Vereinfacht ausgedrückt, heißt das: Er bekommt freundlicherweise auf einer Seite Strom, der durch den Widerstand im Paddle mehr oder weniger geschwächt wurde, und gibt diesen Strom, weil er ja kein Egoist ist, auf der anderen Seite schön mundgerecht an eine 8-Bit-Leitung weiter. Der Witz dabei ist, daß die Stromstärke auf der einen Seite einem Bitwert auf der anderen entspricht. So kann der Computer mit den Paddleinformationen arbeiten.

Wie wurde das jetzt technisch gelöst?

Also: Ein Paddle hat einen drehbaren Knopf auf der Oberseite. Der ist mit einem Schiebewiderstand verbunden, und je nachdem, wie weit rechts der Knopf oben gedreht ist, um so größer wird dieser Widerstand. So kann man den durchfließenden Strom schwächen. Der Vorteil von Paddles ist klar: Anstatt irgendwelche Koordinaten zu vergrößern oder zu verringern, kann man hier die Stellung einer Figur direkt vom Ausschlag des Paddles abhängig machen.

Paddles werden immer als Paar geliefert, also eines für die X-, eines für die Y-Richtung bzw. eines für den ersten, eines für den zweiten Spieler. So

können Sie also vier Paddles an Ihren Commodore anschließen – zwei an Port 1 und zwei an Port 2.

Aber gleich ein Wermutstropfen. A/D-Wandler sind ziemlich teuer, und da Sparen großgeschrieben wird, haben die Commodore-Leute in die Trickkiste gegriffen. Unser kleiner SID, von dem wir gerade im vorigen Kapitel gehört haben, hat – Pech für ihn – bereits zwei solcher A/D-Wandler eingebaut. Also werden die Paddles von einem CIA-Baustein auf diese A/D-Wandler umgeleitet. (CIA hat in diesem Fall ausnahmsweise nichts mit dem gleichnamigen amerikanischen Geheimdienst zu tun, obwohl die Vermutung gar nicht so weit hergeholt wäre, sondern heißt einfach Complex Interface Adapter.)

Solche falschen Botschafter wie der CIA (Baustein!!) sind uns ja schon bekannt. Man vergleiche es nur mit dem Schicksal von VIC. Aber so ist das halt in diesen Königreichen. Doch mit den nötigen Bestechungs-POKEs kann man trotzdem 4 Paddles gleichzeitig auslesen. Wie, zeigen wir gleich. Zunächst erst mal die beiden SID-Adressen:

PEEK(54297) = Paddle 1
PEEK(54298) = Paddle 2

Die Paddles müssen zunächst an Port 1 angeschlossen werden. Sollten Sie die Adressen gleich ausprobieren wollen, noch einige kurze Bemerkungen: Wir sagten, die Werte können von 0 bis 255 liegen, sie müssen aber nicht ...

Viele Paddles kommen selbst bei Vollausschlag nicht so weit. Deshalb auch unsere Einkaufstips.

Zweitens werden die Werte selten ganz ruhig stehen wegen der hohen Auflösung des Stroms. Sie müssen bedenken, daß jeder Strom, der in den A/D-Wandler kommt, in 256 Einzelstufen zerlegt wird. Deshalb werden die digitalen Werte des ursprünglich analogen Stromflusses um einige Stufen nach oben und unten differieren. Ansonsten steht allerdings einem Programm

```
10 PRINT PEEK(54297);PEEK(54298):RUN
```

nichts mehr im Wege.

Wenn Sie die Paddles angeschlossen haben, wird Ihnen auffallen, daß sie an der Seite je einen Feuerknopf haben. Den muß man natürlich auch abfragen können. Die Adresse, mit der man das bewerkstelligen kann, kennen wir schon: 56321.

Allerdings haben die Bits diesmal eine andere Belegung.

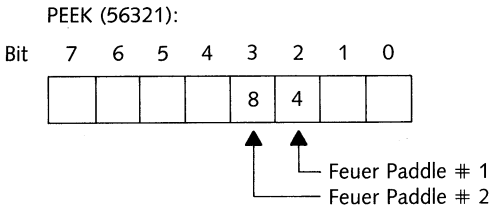


Bild 10.4 Paddles Port 1, Bitbelegung in (56321)

Abfragen lassen sich die Paddle-Feuerknöpfe folgendermaßen:

```
PORT 1
IF NOT PEEK(56321) AND 4 THEN 1. PADDLE
IF NOT PEEK(56321) AND 8 THEN 2. PADDLE
```

Über die sinnvolle Anwendung eines zweiten Feuerknopfes machen Sie sich am besten Ihre eigenen Gedanken.

Oh, beinahe hätten wir es vergessen. Da war ja noch eine Intrige zu bereinigen. Wie ist es möglich, gegen den CIA die Paddles zu lesen, die am Port 2 hängen? Nun, die beste Art, den Feind zu schlagen, ist manchmal, sich mit ihm zu verbünden. Also nehmen wir erst mal mit dem CIA-Typen Kontakt auf. Er liegt gerade so ab der Adresse 56320 herum. (Schau an ...)

Hier erfüllt er verschiedene Aufgaben. Zum Beispiel läuft über ihn die ganze Tastaturabfrage. Und er ist auch für die beiden Game Ports zuständig. Sie sehen, er ist überall da, wo Informationen fließen. Und deshalb ist es auch er, der die Signale an den SID weiterleitet. Die obersten beiden Bits der Speicherzelle 56320 (Sie erinnern sich vielleicht noch: auch Joystick 2 hängt hier daran. Dieser beeinflußt aber nur die unteren Bits); diese obersten beiden Bits also sind dafür verantwortlich, welcher Port weitergeleitet wird.

Wenn wir nun dem CIA-Typen auf diese Weise eine fingierte Botschaft überbringen, daß statt Port 1 der andere gelesen werden soll – schon haben wir, was wir wollen. Allerdings müssen wir vorher noch die Tastaturabfrage abschalten, denn während des Lesens von Paddle 2 darf sie nicht stattfinden. Sonst merkt unser kleiner Agent, was hier gespielt wird. Und man kennt diese Typen ja ...

Eine kurze Zwischenbemerkung, solange wir uns moralisch auf unseren kleinen Anschlag vorbereiten: Wenn Sie sowieso nur ein Paddlepaar haben, schließen Sie es der Einfachheit halber gleich an Port 1 an. Okay, aber jetzt

geht's los. Wir schalten zuerst die Interrupts ab (wie bei der Sonderzeichendefinition mit POKE 56334,0). Arbeiten Sie dabei, wie gehabt, mit dem nötigen Respekt. Wenn das geschehen ist, können Sie die Bits in 56320 ohne Skrupel ändern. Dabei gilt:

POKE	Paddlesatz
56320,64	I
56320,128	II

Jetzt können wir uns die Werte der beiden Paddles beim SID abholen. Danach sollten Sie den Interrupt wieder einschalten, damit die Tastatur wieder gelesen werden kann (POKE 56334,1).

Und da 56320 im allgemeinen eine sehr vielseitige Adresse ist, kann man hier auch die Feuerknöpfe der Paddles von Port 2 lesen.

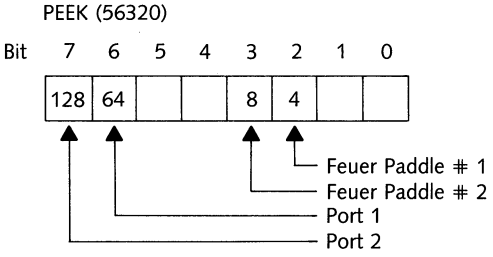


Bild 10.5 Paddles Port 2, Bitbelegung in (56320)

Wie beim Port 1 sind die Bits Nummer 2 und 3 dafür zuständig.

```
PORT 2
IF NOT PEEK(56320) AND 4 THEN 1. PADDLE
IF NOT PEEK(56320) AND 8 THEN 2. PADDLE
```

Zur Abfrage der Feuerknöpfe wie auch der Joysticks dürfen die Interrupts nicht ausgeschaltet sein. Ein Unterprogramm, das alle Paddleabfragen durchführt, sieht ungefähr so aus:

```
10 POKE56334,0
20 POKE56320,128
30 X2=PEEK(54297):Y2=PEEK(54298)
```

```
40 POKE 56320,64
50 X1=PEEK(54297):Y1=PEEK(54298)
60 POKE 56334,1
```

Dabei entsprechen die Variablen folgenden Werten:

X1 = WERT PADDLE 1, PORT 1
Y1 = WERT PADDLE 2, PORT 1
X2 = WERT PADDLE 1, PORT 2
Y2 = WERT PADDLE 2, PORT 2

Das war es dann für heute. Bleibt Ihnen nur noch die Kleinigkeit, ein wahnsinnig tolles Softwarepaket um 2 oder 4 Paddles herum zu schreiben – vielleicht eines, in dem es um Agenten vom CIA geht ...

11
Peripheriegeräte

Der Commodore bekommt Gesellschaft

Irgendwann kommt der Tag, an dem reicht der kleine Kassettenrecorder am Commodore einfach nicht mehr aus. Das kann sein, weil man es satt hat, ständig hin- und herzuspulen, oder weil man ein Programm, das man gern möchte, nur auf Diskette bekommt, oder auch, weil man selbst ein Programm schreiben will, das nur mit Diskette funktioniert.

Warum auch immer, der Tag kommt so gut wie sicher, an dem man sich zu der Entscheidung gedrängt sieht, ob man eine Diskettenstation anschaffen sollte oder nicht.

Dieses Kapitel ist natürlich in erster Linie für diejenigen geschrieben, die schon eine Floppy (oder einen Drucker, über den wir auch kurz sprechen wollen) besitzen. Aber auch die anderen sollten es lesen. Zum einen, um informiert zu sein, was solche Erweiterungen bringen und ob sich ihre Anschaffung zum gegenwärtigen Zeitpunkt lohnt. Zum anderen, um eine solche Erweiterung schon jetzt beim Programmieren berücksichtigen zu können. Schon manch einer, der seine Datenverwaltung vorher mit Kassette erledigt hat, war nach dem Kauf einer Floppy wochenlang damit beschäftigt, seine Programme umzuschreiben. Dabei hätte es bei entsprechender Programmierung schon gereicht, einige Adressen zu ändern (nicht die in der Datei, sondern die im Programm!)

Jetzt aber genug der Vorrede. Wir wollen uns zunächst um diese runden schwarzen Scheiben kümmern. Nein, gemeint sind nicht angebrannte Pfannkuchen, sondern die Disketten selbst.

Eine Scheibe mit Format

Meist das erste Zusatzgerät, das nach dem Commodore nebst Kassettenrecorder gekauft wird, ist die Diskettenstation, abgekürzt auch gern Floppy genannt. Schon dieser Spitzname klingt sehr lebendig, im Sinne von schnell. Und so sind dann auch meist folgende Vorteile gegenüber dem Kassettenrecorder ausschlaggebend für die Anschaffung: Die Floppy speichert und lädt Daten schneller als der Recorder. Sie tut das aber auch sicherer. Ein LOAD ERROR kommt bei Kassetten schon öfter mal vor, bei einer Floppy fast nie. Und die Floppy ist komfortabler zu bedienen. Außerdem gibt es, wie schon oben erwähnt – hauptsächlich für professionelle Anwendungen wie Textverarbeitung –, Programme, die man eh nur auf Diskette bekommt. Meist machen sie auch nur da Sinn.

Vielleicht überlegen Sie sich jetzt, warum das so ist. Denn schließlich speichern doch beide Geräte einfach nur magnetische Signale.

Zur Klärung schauen wir uns am besten mal kurz an, wie die Geschichte des Speicherns ablief.

Bekannt ist ja, daß ein Programm zumeist im RAM steht. Da steht es auch gut, solange Strom durchfließt. Doch wird der Computer ausgeschaltet, dann verschwindet das Programm. Nun könnte man den Computer auch die ganze Nacht anlassen oder so eine Art Notstromspeicher in Form einer Batterie einbauen. Gut, sicher ein Weg. Aber dann bräuchten Sie für jede Anwendung einen eigenen Computer. Oder Sie tippen jedesmal, nachdem Sie was anderes gemacht haben, das gewünschte Programm wieder ab. Unpraktisch, nicht? Ein Programm muß also irgendwie, möglichst unabhängig vom Strom, konserviert werden können und zwar so, daß der Computer es später auch wieder einlesen kann. Zu diesem Zwecke mußten wieder unsere kleinen Freunde, die Bits, herhalten. Denn sie kann man am einfachsten haltbar machen. Aber bis zur Speicherung auf Floppy-Disks war es auch nach dieser Erkenntnis noch ein weiter Weg.

Lange Zeit wurden Lochkarten zum Speichern verwendet. Sie können sich es wohl schon denken. Loch = Strom an; kein Loch = Strom aus. Allerdings hatte dieses System einige entscheidende Nachteile. Komplexere Programme nahmen auf diese Weise sehr schnell das Ausmaß von ganzen Karteikästen an. Dem stand natürlich wieder der Vorteil gegenüber, daß die Programmierer zu Fasching mit genügend Konfetti versorgt waren ...

Spätestens mit der Verbreitung des Magnetbands wurde das Speichern aber schon wesentlich vereinfacht. Die Bits wurden zu Impulsen auf Band

umgewandelt und genau wie Töne der Nachwelt erhalten. Und damit sind wir der Entwicklung immerhin schon bis zum Kassettenrecorder nachgeeilt, den man auch an Ihren Commodore hängen kann.

Doch ein entscheidender Nachteil blieb. Suchte man ein Programm, das am anderen Ende des Bandes gespeichert wurde, mußte man eben spulen. Und das dauerte natürlich. Kurz und gut: Ein schneller Datenzugriff war immer noch nicht möglich.

Also besannen sich die Techniker auf die (ältere) Technik der Schallplatte. Hier kann der Tonarm ja auch, durch Hin- und Herfahren, ziemlich jede Position blitzschnell erreichen. Zu empfehlen ist allerdings, ihn vorher von der Platte zu heben.

Und somit stand der Floppy-Disk eigentlich nichts mehr im Wege – außer vielleicht die Kleinigkeit der technischen Entwicklung ...

In der Tat ist eine Diskette einer Schallplatte gar nicht so unähnlich. Nur, daß es bei der Diskette keine Rillen mehr gibt, sondern Spuren, und daß die auch nicht einfach «gekratzt», sondern rein magnetisch aufgezeichnet werden.

Heute gibt es im großen und ganzen zwei Arten von Disketten: Die großen, etwas klobigen 8-Zoll-Disketten, die hauptsächlich bei Großcomputern eingesetzt werden, und die Fünfeinviertel-Zoll-Disketten, wie sie auch die VC-1541-Floppy für Ihren Commodore verwendet. Womit das Kind auch seinen Namen hätte. Und genau diese Diskettenart wollen wir uns jetzt näher ansehen.

Wenn wir beschreiben, wie die Diskette aussieht, wollen wir uns nicht bei Äußerlichkeiten aufhalten. Wahrscheinlich hat jeder von Ihnen schon einmal so eine Scheibe gesehen. Viel interessanter ist, wie so oft, was man nicht sehen kann.

Damit hier keine Probleme aufkommen, die eigentliche Diskettenscheibe ist natürlich rund, nicht eckig. Sie kann sich in der eckigen Hülle frei drehen. Allerdings sollten Sie das nicht mit den Händen ausprobieren. Denn die Oberfläche der Diskette, die aus mikrofeinen Partikeln besteht, ist sehr empfindlich gegen Hautfett und Schweiß. Die Diskette fassen Sie am besten immer nur an der Schutzhülle an – da, wo zum Beispiel auch der Markenaufkleber ist.

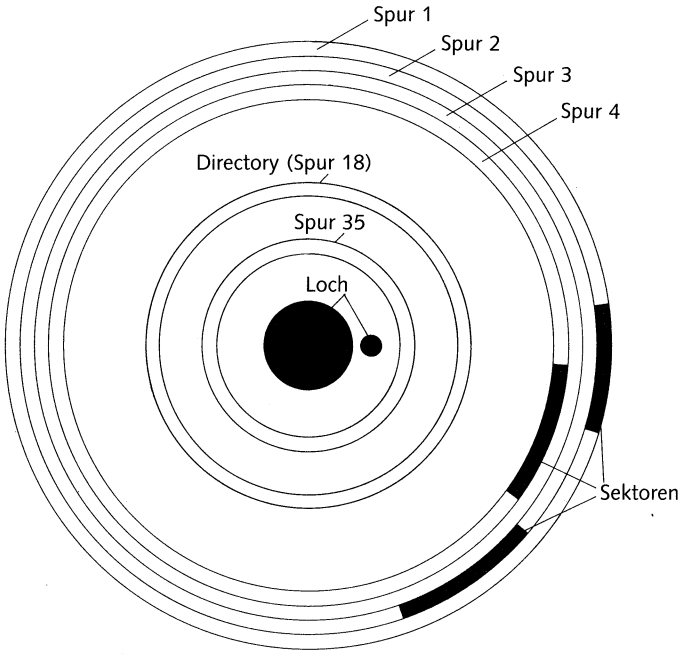
Wie gesagt, die Informationen befinden sich in Form magnetischer Signale auf der Oberfläche der Diskette. Dazu werden auf der Diskette konzentrische *Spuren* angelegt. Die Anzahl, Breite und Lage dieser Spuren ist von Firma zu Firma verschieden. Wohlgemerkt: nicht von Diskettenfirma zu Disketten-

firma, sondern von Computerfirma zu Computerfirma. Wenn Sie Disketten kaufen, dann sind sie sozusagen erst mal frei von jeglichen Regeln. Das *Format* wird beim ersten Einsatz aufgezeichnet. Man nennt das auch *Formatieren*. Wie das in unserem Fall geht, zeigen wir Ihnen noch. Während des Formatierens werden in jedem Fall konzentrische, ineinanderliegende Kreise magnetisch vorgezeichnet. Zum Schluß sieht die Diskette dann, natürlich nur magnetisch betrachtet, aus wie eine Schießscheibe beim Schützenverein. Bei der 1541 werden 35 solcher Spuren gezogen.

Eine Spur wird dann ihrerseits wieder in mehrere *Sektoren* unterteilt. Ein Sektor ist also ein Ausschnitt aus einer Spur. In einem solchen Sektor können 256 Bytes Informationen gespeichert werden. Wenn Sie sich noch an die Speicheraufteilung erinnern: 256 Bytes sind für den Computer genau eine Speicherseite. Insgesamt befinden sich auf Ihrer Diskette 683 solcher Sektoren. Damit die Floppy aber bei dieser Menge von Sektoren (auch *Blocks* genannt) auch immer weiß, wo sich welche Informationen befinden, gibt es ziemlich in der Mitte der Diskette, nämlich auf der Spur 18, ein Inhaltsverzeichnis – die sogenannte Directory. Auf dieser Spur sind Informationen wie der Name und die Nummer der Diskette gespeichert, außerdem die Namen der Programme und der Datensätze, die sich auf ihr befinden, und auch ein Zeiger, das ist so eine Art interner Wegweiser, der angibt, auf welcher Spur und in welchem Sektor der Anfang dieser Aufzeichnung zu suchen ist. Natürlich belegt die Directory die Spur 18 für sich, und so können hier keine anderen Daten mehr gespeichert werden. Damit bleiben zum Schluß nur noch 664 Blocks frei. Und das entspricht nun runden 170K Speicher.

Wenn Ihnen das jetzt alles etwas verwirrend vorkam, hier noch einmal eine kleine Skizze und eine Übersicht über die wichtigsten Daten Ihrer Disketten. Da zur Speicherung von 256 Bytes ein gewisser Platz nötig ist, der Platz auf jeder Spur aber, wie die Skizze zeigt, zur Mitte hin abnimmt, ist auch einsichtig, daß sich auf den äußeren Spuren mehr Sektoren befinden als auf den inneren. Im einzelnen sieht das dann so, wie in Tabelle 11.1 dargestellt, aus.

Jetzt wissen wir doch schon allerhand über das Format der Disketten. Nur noch eine kleine Anmerkung: Sie wundern sich vielleicht über das kleine Loch in der Diskette. Nein, das ist nicht das Versehen eines Programmierers, der noch Lochkarten gewohnt ist. Aber ein Relikt aus vergangenen Tagen ist es trotzdem. Wie Sie sich vorstellen können, ist es unbedingt notwendig, daß sich die Diskette mit der absolut gleichen Geschwindigkeit dreht. Sonst würde der Schreib-/Lese-Kopf des Laufwerks die Daten nicht mehr erkennen kön-



Spuren: 35
Sektoren: 683
Bytes je Sektor: 256
Directory: Spur 18
freie Sektoren: 664
Kapazität: ~ 170K

Bild 11.1 Diskette

Spur	Anzahl der Sektoren
1 bis 17	21
18 bis 24	19
25 bis 30	18
31 bis 35	17

Tabelle 11.1 Anzahl der Sektoren pro Spur

nen. Stellen Sie sich nur mal vor, irgend jemand würde eine Ihrer Lieblingsplatten zu schnell oder zu langsam spielen. Dann würden Sie auch Probleme haben, den Sänger wiederzuerkennen. Selbst Langspielplatten mit Wagners «Ring der Nibelungen» würden an Ausdruckskraft verlieren, wenn sie in 45er-Geschwindigkeit abliefen.

Deshalb verwendete man früher dieses kleine Loch in der Hülle und in der Magnetscheibe. Einmal pro Umdrehung sind die beiden Löcher ja deckungsgleich, und Licht kann hindurchscheinen. Eine Fotozelle auf der anderen Seite zählte dann die Lichtstöße je Sekunde und regelte auf dieser Basis die Geschwindigkeit. Bei moderneren Floppys geschieht dies durch eine magnetische Synchronmarkierung, die vom Schreib-/Lese-Kopf erkannt wird.

Und noch ein Tip zur Diskettenpflege: Die Seite, auf der aufgezeichnet wird, ist entgegen der allgemeinen Vermutung nicht oben, wo das Etikett ist, sondern unten. Deshalb ist es nicht zu empfehlen, die Disketten nur von oben gegen Staub zu schützen und sie mit der Unterseite irgendwo herumliegen zu lassen. Der beste Platz für Disketten ist die Schutzhülle. Und der beste Platz für Disketten in der Schutzhülle ist ein Diskettenkasten.

Nach all diesen theoretischen Vorbemerkungen ist es nun soweit. Wir nehmen Kontakt mit der Floppy-Disk auf.

Das Königreich von nebenan – die VC 1541

Sie erinnern sich vielleicht noch an unser Bild vom 6510, dem Prozessor Ihres Commodore 64. Nun, dieses Bild wollen wir hier fortsetzen. Damit lassen sich nämlich viele Dinge ganz anschaulich erklären.

Wenn man also die Floppy als ein eigenes Reich betrachtet, und das kann man durchaus, dann zeigt dieses Reich zuerst die typischen Eigenschaften eines Agrarstaates, denn schließlich sind die dort alle damit beschäftigt, irgendwelche Furchen auf der Diskette anzulegen und dort massenweise Bits anzupflanzen.

Auf der anderen Seite sind auch deutliche Merkmale eines Industriestaates zu erkennen: Die Floppy ist in hohem Maße von Import und Export abhängig. Über dieses komplizierte Staatsgefüge herrscht ein etwas älterer, aber weiser und gemütlicher König: Sein Name ist 6502 (auf das Beiwerk «der Erste» verzichtete man, weil auch so schon genug Zahlen im Namen vorkommen). Er ist übrigens der Vorläufer des jungen 6510. Der 6502 ist zwar etwas langsamer als der 6510, aber dafür hat er seine Ruhe. Er ist absoluter

Herrscher in seinem Reich. Allerdings finden wir als Besucher einige alte Bekannte hier in der Floppy wieder: RAMs und ROMs zum Beispiel. In den ROMs steht das Betriebssystem der Diskette, DOS *(Diskette Operating System)* genannt. Auch 2K-RAM finden wir hier. Teils dienen sie als Zeropage für das DOS und teils als Pufferspeicher. Was das genau ist, erklären wir später noch.

Schließlich gibt es auch ein paar I/O-Bausteine zur Kontrolle der Motorgeschwindigkeit, des Schreib-/Lese-Kopfes und zur Kommunikation mit dem Computer. Und damit wäre auch eine wichtige Sache von vornherein geklärt: Die Floppy ist ein eigener, autonomer Computer, allerdings mit anderen Aufgaben, als sie der Commodore hat. Das ist übrigens keineswegs bei allen Computern so.

Die Floppy braucht zum Beispiel keinen Bildschirm, dafür aber verschiedene Elektromotoren zur Bewegung der Diskette und des Schreib-/Lese-Kopfes. Dieser Name ist jetzt schon einige Male gefallen. Was ist eigentlich ein Schreib-/Lese-Kopf? Nun, er ist in erster Linie für die Datenspeicherung verantwortlich. Im Prinzip ist er nichts weiter als ein Elektromagnet. Wenn er schreibt, wandelt er elektrische Impulse in ein Magnetfeld um, das die Partikel auf der Diskettenoberfläche eben magnetisiert – oder nicht (nach dem bekannten Prinzip: Strom oder kein Strom). Wenn er liest, dann wandelt er die magnetischen Impulse auf der Diskettenoberfläche wieder um: In Strom an/Strom aus. Allerdings denkt er sich nicht viel dabei. Den Rest überläßt er der Regierung – wie ein braver Bürger eben.

Wenn wir also von der Floppy irgend etwas wollen, müssen wir schon sozusagen beim König vorsprechen. Aber das funktioniert nur, wie das halt so ist bei Monarchen, über ein sehr strenges Protokoll. Wenn Sie das noch nicht getan haben, dann schalten Sie Ihre Floppy jetzt ein.

Voraussetzung dafür, daß das Kommando funktioniert, ist natürlich, daß die Floppy ordnungsgemäß an Ihren Commodore angeschlossen ist. Sonst tritt derselbe Effekt ein, wie bei den Western, wo die Banditen genau in dem Moment die Telegrafenleitung kappen, in dem Old Shatterhand den Gouverneur in Santa Fé alarmieren will.

Wenn das geklärt ist, sprechen wir doch mal beim Haushofmeister vor: Dazu müssen wir einen *Kommandokanal* zur Floppy eröffnen.

```
OPEN 1,8,15
```

erledigt das für uns.

Die Zahlen bedeuten im einzelnen folgendes:

1 ist die Kanalnummer. Sie kann völlig frei von 0 bis 255 gewählt werden. Wir leben ja in einer Monarchie, also fast völlig frei: Erstens empfiehlt es sich, üblicherweise nur Nummern unter 127 zu verwenden. Der Sinn dieser Vorschrift wird allerdings erst beim Drucker erklärt. Außerdem sollte man möglichst dieselbe Kanalnummer verwenden wie die Gerätenummer. Das hat den Vorteil, daß, wenn man mehrere Geräte angeschlossen hat, immer eine ganz gute Übersicht behalten werden kann.

8 ist die zweite Zahl. Sie ist die *Gerätenummer* der Floppy. Man sollte sie einfach hinnehmen. Man kann sie zwar softwaremäßig (bis zum nächsten Ausschalten) oder auch hardwaremäßig (also für die fernere Zukunft) ändern. Das empfiehlt sich jedoch nur, wenn man mehrere Floppys gleichzeitig verwenden will. Denn die meisten Programme, die sich auf das Diskettenlaufwerk beziehen, gehen davon aus, daß die Gerätenummer 8 ist.

15, als letzte Zahl, stellt schließlich die *Sekundäradresse* dar. Sie spricht verschiedene Betriebsmodi an, über die wir uns später noch unterhalten wollen. In unserem Fall steht die 15 auf jeden Fall für den Modus: «Über diesen Kanal werden Befehle und Meldungen hin- und hergeleitet.» Förmlich ausgedrückt, entspricht das bei Hof: «Wir bitten untertänigst um die Gnade einer Audienz. Wenn Sire 6502, hochwohlprogrammiert, bitte anhören würden, was wir vorzubringen gedenken.»

Falls unser Ersuchen nicht abgelehnt wurde, zum Beispiel durch einen SYNTAX ERROR oder einen ILLEGAL QUANTITY ERROR oder andere diplomatische Verwicklungen, dann steht die Verbindung. So, spätestens jetzt sollten wir uns überlegen, was wir eigentlich vorzubringen gedenken. Denn Majestäten sind ja nun sehr beschäftigte Leute.

Bevor die Floppy allerdings irgend etwas tun kann, braucht sie erst mal Futter. Schieben Sie ihr also am besten die dem Gerät beigefügte «TEST/DEMO»-Diskette in den Schlund. Dann machen Sie die Klappe zu. Achten Sie darauf, daß das Etikett beim Einschieben immer oben ist und auf Sie zuzeigt – oder andersrum, daß die ovale Aussparung an der Diskette in Richtung des Laufwerks zeigt, wenn Sie sie einschieben. Denn nur über diese Öffnung kann der Schreib-/Lese-Kopf direkt auf die magnetische Oberfläche zugreifen. Nachdem nun wirklich alle Vorbereitungen getroffen wären, wollen wir frisch heraus unsere Forderungen stellen: «Sire, leset die Identity in Ihro Arbeitsspeicher». Oder, etwas moderner ausgedrückt:

```
PRINT#1,"I"
```

Wenn jetzt das Laufwerk zu surren anfängt und die rote LED-Anzeige kurz

aufleuchtet, zeigt uns der Hof sein Verständnis. Aber ... was haben wir ihm eigentlich gesagt?

Nun, nachdem Sie ja den Ausdruck Identity gelesen haben, müssen wir Sie erst einmal von einer unter Umständen falschen Spur abbringen. Dieser Ausdruck hat keineswegs damit zu tun, daß unser kleines Reich unter einer Identitätskrise leidet. Eigentlich war der Befehl, den wir gegeben haben, sogar etwas unnötig, und zwar aus folgendem Grund: Beim Formatieren, wir werden das nachher noch zusammen ausprobieren, wird jeder Diskette ein zweistelliger Identifikations-Code («Identity») zugeordnet. Anhand dieses Codes erkennt die Floppy, um welche Diskette es sich handelt. Oder einfacher gesagt, wenn sich dieser Code von dem der vorhergehenden Diskette unterscheidet, ist alles in Butter. Dann wird nämlich alles, was der «I»-Befehl (von Initialize) bewirkt, automatisch erledigt. Wenn aber nicht – und diese Möglichkeit besteht, falls Sie nicht wirklich für jede Diskette eine andere Nummer verwendet haben –, dann gibt es Probleme. Und genau deshalb muß der Floppy dann mitgeteilt werden, daß jetzt eine neue Diskette eingelegt wurde. Das tut der «I»-Befehl. Deshalb ist er auch eher als eine erste schüchterne Kontaktaufnahme als ein richtiger Befehl zu sehen. Warum interessiert die Floppy, welche Diskette eingelegt wurde? Um das zu verstehen, muß man sich etwas genauer über die Vorgänge bei Hofe informieren.

Das Land, das dort ständig bebaut wird (die Diskettenoberfläche), stellt mit seinen 683 Blocks ja schon ein relativ großes Terrain dar. Und damit keine Querelen entstehen, wem nun welches Stück Land gehört, ist beim König eine Landkarte hinterlegt, auf der genau verzeichnet ist, ob ein Block schon belegt ist. Das ist in erster Linie wichtig, wenn Programme oder Daten geschrieben werden sollen. Dann muß die Floppy nämlich wissen, in welchem Block schon Daten stehen. Dazu wird ein solcher Block auf der Karte (sie heißt BAM = *Block Availability Map*) als «belegt» gekennzeichnet. Diese BAM wird zwar auf die Diskette draufgeschrieben, aber um schneller damit arbeiten zu können, wird sie ins RAM kopiert. Würde man jetzt die Diskette wechseln, ohne daß es die Floppy weiß, dann würde sie sich auf die flache BAM in RAM beziehen. So schön sich das auch reimt, so katastrophale Folgen kann es jedoch haben. Denn hier sind ja ganz andere Blocks frei und belegt als auf der Diskette. Die Auswirkungen wären wohl eine Art Flurbereinigung, gegen Ihren Willen. Und das würde Ihren Programmen gar nicht guttun, weil belegte Blocks eben überschrieben werden. Um genau solche Hausrevolutionen zu verhindern, ist die ID-Nummer da. Denn vor jeder Schreiboperation überprüft die Floppy diesen Code. Hat er sich geändert, wird die neue BAM

eingelesen – kein Problem. Wenn nicht, geht die Floppy, nicht ganz zu Unrecht, davon aus, daß alles beim alten geblieben ist, und schreibt deshalb munter drauflos – mit allen Konsequenzen, die wir oben beschrieben haben.

Für Sie sollte das jetzt konsequenterweise heißen: unterschiedliche ID-Codes verwenden! Oder nach jedem Diskettenwechsel einen «I»-Befehl zur Floppy schicken. Die Lösung mit der ID-Nummer ist wohl die bessere. Im übrigen sollten Sie sich dazu ein eigenes System zurechtlegen, damit Nummernwiederholungen ausgeschlossen werden. Bevor wir jetzt aber anfangen, unsere erste eigene Diskette anzulegen, wollen wir uns ein paar Dinge von der «TEST/DEMO»-Diskette anschauen. Dazu bitten wir zunächst unseren kleinen König, die Audienz zu beenden:

```
CLOSE 1
```

Dafür wollen wir uns jetzt das Inhaltsverzeichnis der «TEST/DEMO»-Diskette mal genauer ansehen. Da taucht aber auch gleich wieder ein Problem auf: Das Commodore-64-BASIC V2 ist auf Diskettenbefehle etwa so großartig vorbereitet wie der Durchschnittsschüler auf eine Latein-Kurzarbeit, das heißt also wenig bis gar nicht. Bei den meisten Computern gibt es einen bestimmten Befehl, um sich dieses Inhaltsverzeichnis anzuschauen. Schließlich ist es ja nicht uninteressant, zu wissen, was auf einer Diskette abgespeichert wurde. Beim Commodore gibt es leider erst mal nur einen kleinen Umweg: Die Directory wird als Pseudo-BASIC-Programm geladen und kann dann mit LIST angesehen werden. Der Befehl zum Laden ist:

```
LOAD "$",8
```

Wenn Sie diesen Befehl mit dem BASIC-Kapitel vergleichen, wird Ihnen auffallen, daß hier eigentlich ein Programm ganz normal von der Diskette geladen wird. Nur, daß es den ominösen Namen «$» trägt. Dieser Name ist für ein Programm nun tatsächlich etwas ungewöhnlich. Das merkt auch unser DOS und gibt dem Computer statt dessen das Inhaltsverzeichnis. Mit einem einfachen

```
LIST
```

sehen Sie dann das Gewünschte. Jetzt wollen wir kurz erklären, was Sie nun noch außer Programmnamen auf dem Bildschirm sehen.

In der ersten Zeile steht in invertierter Darstellung (also dunkle Buchstaben auf hellem Grund) der Name der Diskette. In diesem Fall: «1541 TEST/DEMO», gefolgt von der Identity, die hier «ZX» heißt. Nun haben Sie diesen

Namen vielleicht schon einmal gehört, es ist einer der Commodore-Konkurrenten auf dem Markt der Home-Computer. Aber wir gehen von der Vermutung aus, daß es sich hier nur um einen Zufall handelt. Ähnlichkeiten mit tatsächlich existierenden oder noch zu erfindenden Computern wären rein zufällig und völlig unbeabsichtigt. Als letzte Information in dieser Zeile findet sich das Kürzel «2A». Es steht für die DOS-Version, unter der die Diskette formatiert wurde.

Aber die interessiert uns momentan weniger.

In den nächsten Zeilen stehen die Namen der Programme, die auf dieser Diskette gespeichert sind. Vor ihnen finden Sie jeweils eine Zahl. Das ist die Zahl der Blocks oder Sektoren, die das jeweilige Programm dahinter belegt. Daraus läßt sich auch in etwa die Größe eines Programms errechnen. Als Faustregel gilt

Zahl der Blocks/4 = Länge des Programms in KByte.

Hinter den Namen können Sie an einem Kürzel noch erkennen, um welche Art von Aufzeichnung es sich handelt.

PRG steht für Programm und ist meist das übliche.

SEQ steht für sequentielle Datei.

REL steht für relative Datei.

Ganz am Ende steht noch die Anzahl der freien Blocks, also der Blocks, die noch nicht belegt sind. In unserem Fall müßten das rund 558 sein. Wenn Sie jetzt feststellen, daß diese Werte und Namen relativ wenig mit Ihrer «TEST/DEMO»-Diskette zu tun haben, dann machen Sie sich keine Sorgen. Genauso wie manche Leute alle paar Monate einen Rappel bekommen und ihre gesamten Möbel in der Wohnung hin- und herschieben, so scheint es auch Commodore zu gehen. Immer mal wieder überraschen sie Käufer mit neuen «TEST/DEMO»-Versionen. Übrigens geht das Gerücht um, daß ähnliches auch mit dem Betriebssystem der Fall sein soll. Nur nicht so oft ...

Bisher erschien Ihnen das aber alles wohl recht unproblematisch. Das liegt zum einen sicherlich daran, daß Sie gar nichts anderes gewöhnt sind. Es gibt zum Beispiel einen mit einer Fruchtsorte verwandten Computer, der dasselbe einfach mit dem Befehl CATALOG erledigt. Und da ist er beileibe nicht der einzige. Zum anderen ist Ihnen sicherlich auch noch die Konsequenz des Ladens eines BASIC-Programms nicht ganz klar. Wenn man das nämlich tut, dann löscht man damit immer das bereits im Speicher Befindliche. Das ist nun an und für sich eine sehr gute Einrichtung, um ein heilloses Durcheinander von Programmen zu verhindern. Aber stellen Sie sich jetzt mal vor, Sie haben

gerade ein ziemlich langes BASIC-Programm eingegeben und wollen nun, nach getaner Müh', dieses Programm auf Diskette speichern. Da Sie aber nicht genau wissen, ob noch genug Platz auf der Diskette übrig ist, wollen Sie eben nachschauen. Aber wir würden Ihnen davon sehr abraten. Denn in dem Augenblick, da Sie das Directory als Quasi-BASIC-Programm laden, können Sie Ihr eigentliches BASIC-Programm nur noch als freudige Erinnerung im Gedächtnis behalten. Denn jedes neue Programm löscht das vorhergehende. Also heißt es nur: entweder Programm abspeichern oder auf der Diskette nachschauen.

Aber bevor Sie sich jetzt darüber ständig den Kopf zerbrechen, sei Ihnen hier schon verraten, daß Commodore dafür glücklicherweise eine Lösung hat. Nur weil alles schön der Reihe nach gehen soll, dachten wir uns, wir formatieren erst mal eine Diskette zusammen.

Mit all unserem Wissen wollen wir jetzt mutig Neuland betreten – so, wie die Siedler mit ihren Trecks in den rauhen Westen zogen und dort die Prärie urbar machten. Unsere Prärie besteht aus einer unformatierten, leeren Diskette. Wenn Sie eine solche momentan nicht besitzen, sollten Sie sich schnell welche kaufen. Auf keinen Fall sollten Sie auf die Idee verfallen, Ihre «TEST/DEMO»-Diskette neu zu formatieren. Denn das bedeutet auch den Verlust sämtlicher bereits sowieso nur spärlich vorhandener Programme. Normalerweise dürfte aber der Formatierungsbefehl mit Ihrer «TEST/DEMO»-Diskette gar nicht funktionieren. Aber man weiß ja nie.

Warum er nicht funktionieren würde? Nun schauen Sie sich diese Diskette doch mal genau an. An der rechten Seite, wo bei anderen Disketten meist ein kleiner viereckiger Ausschnitt zu sehen ist, befindet sich in Ihrem Fall ein silberner Aufkleber. (Die Farbe kann differieren, die Funktion bleibt aber die gleiche.) Dieses Stück undurchsichtiges Klebeband, um nichts anderes handelt es sich im Grunde, wird auch als «Schreibschutzaufkleber» bezeichnet. Solange der kleine, viereckige Ausschnitt damit zugeklebt ist, kann auf die Diskette nichts geschrieben werden. Solche Aufkleber befinden sich meist in den Diskettenpackungen, so daß Sie auch selbst Ihre Disketten vor dem Beschreiben schützen können. Ist der Aufkleber allerdings weg, ist Ihre Diskette allen Schreibangriffen schutzlos ausgesetzt. Und weil Formatieren wirklich ein sehr drastischer Vorgang ist, sollten Sie ihn eigentlich nur bei frisch gekauften Disketten anwenden oder bei Disketten, bei denen Sie sicher sind, daß entweder alle Programme irgendwo anders nochmals abgespeichert sind oder sowieso nichts taugen und deshalb ruhig gelöscht werden können.

Neue Disketten aber müssen erst mal, wie schon erwähnt, formatiert werden. Was passiert dabei?

Ganz einfach. Ihre Commodore-Floppy wird sich Spuren und Sektoren, die sie als passend empfindet, und die Directory anlegen. Übrigens liegt die Betonung hier auf «Ihre Floppy». Es kann manchmal schwierig sein, wenn zwei Leute Programme austauschen. Wenn nämlich der Schreib-/Lese-Kopf einer Floppy nur ein bißchen falsch justiert ist, dann kann es sein, daß die andere Floppy die Diskette nicht lesen kann, obwohl sie auf der ersten einwandfrei gelaufen ist. In so einem Fall sollte der Schreib-/Lese-Kopf vom Fachhändler nachjustiert werden.

Jetzt aber zum Formatieren: Wenn Sie Ihre neue Diskette erst mal eingelegt haben, besteht das einzige Problem noch darin, dem König mitzuteilen, daß sich seine Arbeiter um das Neuland kümmern sollen. Also fangen wir wieder mit einem höflichen Türklopfen an:

```
OPEN 1,8,15
```

Wenn Sie das getan haben, müssen Sie sich über zwei Punkte klarwerden: Wie soll die Diskette heißen und welche ID-Nummer bekommt sie? Letzteres dürfte für Sie kein Problem sein. Um Ihren (hoffentlich) guten Vorsätzen treu zu bleiben, dürfen Sie jede zweistellige Kombination wählen, außer «ZX», denn das ist ja schon auf der «TEST/DEMO»-Disk verbraucht worden. Wie gesagt, am besten legen Sie sich für diesen ganzen ID-Kram ein bestimmtes System zurecht. Das hilft Ihnen, gleiche IDs zu vermeiden.

Zum ersten Punkt: Sie können jeder Diskette einen Namen geben. Dieser Name kann bis zu 16 Zeichen lang sein und sollte natürlich irgendeinen Bezug zum Inhalt der Diskette haben. Diskettennamen wie «SPIELE 1» oder «UTILITIES» sind sicher besser zum späteren Wiedererkennen geeignet als Namen wie «FRITZ» oder «C-64 DISK». Diese ganzen persönlichen Wünsche teilen Sie am besten Ihrer Majestät selbst mit.

```
PRINT#1,"N: TEST,01"
```

Wir haben uns also für den Namen TEST und die ID-Nummer 01 entschieden. Es steht Ihnen natürlich völlig frei, hier einzusetzen, was auch immer Ihnen Spaß oder Sinn zu machen scheint – mit einer Einschränkung. Es dürfen nicht die beiden Zeichen «?» und «*» vorkommen. Die werden nämlich noch anderweitig gebraucht. Stößt ansonsten unser Ansuchen bei Hofe auf Verständnis, dann können wir gleich nach dem ⟨RETURN⟩ mit einer Antwort rechnen. Sie müßte sich in einigem Rumoren und dem Leuchten der roten LED äußern. Während das alles passiert, wird die Diskette neu formatiert. Etwa 80 Sekunden herrscht in der Floppy derweil Hochbetrieb. Und nun

kommt ein Vorteil zum Tragen, den wir vorhin schon kurz erwähnten: Nämlich der, daß die Floppy ein eigener Computer ist. Ihr Commodore ist nämlich während der ganzen Zeit verfügbar. Sie können damit fast alles machen wie bisher, nur nicht auf die Floppy zugreifen. Die hat ja zu tun. Wenn die rote LED während des Formatierens zu blinken anfängt, dann ist irgendein Fehler aufgetreten. Prüfen Sie Ihre Eingabe zuerst einmal auf Tippfehler. In jedem Fall sollten Sie aber einen zweiten Anlauf nehmen. Wenn es dann immer noch nichts ist, sollten Sie das Ganze noch einmal mit einer anderen Diskette probieren. Erst wenn das alles nichts fruchtet, bringen Sie Ihre Floppy oder auch die verwendeten Disketten zum Fachhändler.

Ist aber alles richtig gelaufen, hört die rote LED auf zu leuchten. Wir haben jetzt eine leere formatierte Diskette.

Das können Sie auch gleich mit

```
LOAD"$",8
```

nachprüfen. Ein nachfolgendes LIST müßte in etwa folgendes Bild ergeben.

```
0 "TEST                 " 01 2A
664 BLOCKS FREE.

READY.
```

In der ersten Zeile steht, wieder invertiert, TEST oder wie auch immer Sie Ihre Diskette genannt haben. Dann kommt die ID-Nummer, bei uns die 01, und natürlich die obligatorische DOS-Version-Angabe 2A.

Darunter sollte stehen: 664 BLOCKS FREE

Sollte sich Ihr Ergebnis von unserem unterscheiden, das heißt im Klartext, wenn Sie irgendwelche ominösen Zeichen, Symbole und ungeordnete Buchstaben anstatt der aufgezählten Zeilen vorfinden, ist ein Fehler beim Formatieren aufgetreten. Das kann verschiedene Gründe haben: Oft liegt es an den verwendeten Disketten. Die Commodore-Floppys sind in bezug auf ihr Futter wirklich sehr wählerisch. Es kann aber auch sein, daß die Disketten wirklich einfach schlecht gemacht sind. Auch die besten Landarbeiter können einem ausgelaugten Boden nichts entlocken. Versuchen Sie es auch hier im Zweifelsfall noch einmal. Ansonsten wäre wieder der Gang zum Händler notwendig.

Ein Tip am Rande: Fragen Sie den Händler, ob er eine bestimmte Diskette empfehlen würde. Das ist meist ein brauchbarer Rat. Und versichern Sie sich, daß Sie umtauschen können, wenn Sie das erste Mal Disketten kaufen. Dann

können Sie gut probieren, ob Diskette und Floppy sich verstehen. Und scheuen Sie sich auch nicht davor, mangelhafte Disketten, die ohne Ihr Verschulden einfach schlecht sind (wellig oder dergleichen), zurückzubringen und umzutauschen.

Wenn aber alles reibungslos geklappt hat, dann hat soeben eine neue Diskette das Licht der elektronischen Welt erblickt. (Allerdings wird kleinen Disketten auch gern die Geschichte vom Klapperbit erzählt, das sie gebracht haben soll ...)

Und weil wir jetzt eine schöne, freie Diskette haben, wollen wir doch gleich mal ausprobieren, wie das mit dem Laden und Speichern vor sich geht.

Geben Sie also ein kleines BASIC-Programm ein. Vorher sollten Sie aber mit

```
NEW
```

den Speicher löschen.

Dann können Sie Ihrer Fantasie freien Lauf lassen. Wie gesagt, es muß nichts Großartiges sein, vielleicht ein Spiel oder ein Textverarbeitungsprogramm. Im Notfall tut es auch ein BASIC-Zweizeiler.

Jetzt wollen wir dieses neue Programm auf unserer Diskette speichern. Dazu geben Sie einfach ein:

```
SAVE"PROGRAMM",8
```

Wie Sie sehen, haben wir unser Programm sinnigerweise PROGRAMM genannt. (Gut, gut, wir geben's ja zu, daß das nicht gerade ein Geniestreich ist. Sie können es ja gerne anders machen.) Nach dieser Eingabe sollte die Floppy aktiv werden. Wie üblich, erkennt man das am Geräusch und der roten LED.

Wenn die LED erlischt und auf Ihrem Bildschirm die Meldung «READY» erscheint, ist das Programm abgespeichert. Alle, die vorher nur den Kassettenrecorder kannten, dürften von dieser Geschwindigkeit schon etwas beeindruckt sein.

Nun, wir wollen mal kontrollieren: «Die Programmausweise, bitte schön ...»

```
LOAD "$",8
```

und

```
LIST
```

Aha, jetzt steht in der Directory der Programmname, den Sie vorhin beim Abspeichern benutzt haben. Davor müßte je nach Programmlänge die Anzahl

der Blocks stehen, die davon belegt wurden. Gut, spielen wir also mit diesem soeben angelegten Programm. Löschen wir zunächst den Speicher mit einem resoluten

```
NEW
```

Übrigens haben wir gerade eben nicht etwa unser Programm gelöscht, sondern die mittlerweile nachgeladene Directory – nur zur Erinnerung. Jetzt holen wir unser kleines Programm wieder von der Diskette herunter.

```
LOAD "PROGRAMM",8
```

Natürlich müssen Sie Ihren Programmnamen verwenden. Das hat man halt davon, wenn man so eigenwillig ist.

```
LIST
```

zeigt Ihnen, daß das Beispielprogramm immer noch da ist. Sie können es also so oft von der Diskette in den Speicher holen, wie Sie wollen. Es ist nicht nötig, es wieder zurückzuschreiben – so, wie man zum Beispiel etwas aus einem Regal im Kaufhaus nimmt und es wieder zurücklegt, um dann doch etwas anderes zu kaufen. Hätten Sie es aber doch mitgenommen, wäre es jetzt halt weg. Wenn Sie aber von der Diskette etwas nehmen, dann haben Sie eigentlich nur eine Kopie des Programms in Ihrem Speicher. Das Programm bleibt auf der Diskette.

Dieses kleine Beispiel zeigt auch den Vorteil einer Floppy: Sie können Programme direkt aufrufen, ohne zu spulen oder zu suchen.

Was aber, wenn Sie einige Programme auf der Diskette haben, aber nicht mehr genau wissen, wie ein bestimmtes hieß, sich vielleicht nur noch an den Anfang erinnern können? Oder wenn Sie einfach keine Lust haben, beim Laden jedesmal den ganzen Namen einzugeben? Dann genügt auch ein

```
LOAD"PRO*",8
```

Dieses Symbol wird auch als Joker bezeichnet – so eine Art Hofnarr, der alles mit sich machen läßt. Allerdings kann es natürlich sein, daß hier Namensgleichheiten entstehen, wenn zum Beispiel auf der gleichen Diskette ein Programm namens «PROFIT» ist. In diesen Fällen gilt folgende Regelung: Geladen wird grundsätzlich das Programm, das am weitesten vorne in der Directory steht. Das ist auch der Grund dafür, warum man mit dem Befehl

```
LOAD"*",8
```

direkt nach dem Einschalten immer das erste Programm auf der Diskette laden kann. Warum nur direkt nach dem Einschalten? Ganz einfach: Wenn Sie erst einmal irgendein Programm geladen haben, dann ruft « * » allein immer dieses Programm auf. Das heißt, mit

```
LOAD"*",8
```

wird normalerweise immer das zuletzt geladene oder gespeicherte Programm noch einmal geladen. Wichtig ist allerdings in diesem Zusammenhang, daß mit jedem Ein- und Ausschalten des Commodore auch die Floppy neu anfängt. Man nennt diesen Vorgang Initialisieren. Sie merken das – wie immer – an der aufleuchtenden LED und am Geräusch. Nun haben wir aber noch ein Jokersymbol, nämlich das «?» (sprich Fragezeichen).

Das Fragezeichen steht nun seit Donald Duck für Nichtwissen. Und so auch hier. Ein Beispiel:

```
LOAD "P?STE",8
```

würde Programme wie «PUSTE», «PASTE», «PISTE» und dergleichen laden.

Wenn wieder viele Programme zur Auswahl stehen, gilt: Das erste Programm, das dem gesuchten Code entspricht, wird geladen. Sie können also den Joker überall da hinsetzen, wo Ihnen bestimmte Buchstaben nicht einfallen.

Beim Abspeichern funktioniert das natürlich nicht. Denn wie soll denn die Floppy wissen, welches Programm Sie abspeichern wollen, wenn Sie es selbst nicht wissen? Und auch im Diskettennamen und in der ID haben die Joker nichts verloren. Grundsätzlich gilt für die drei letztgenannten Anwendungen:

«Für Hofnarren und Bedienstete verboten!»

Sollten Sie versuchen, dieses Verbot zu umgehen, dann wird das die königliche Syntaxwache sofort merken und Alarm schlagen. Vielleicht ist Ihnen das auch schon passiert: Wenn ein fehlerhafter Befehl zur Floppy geschickt wird, fängt die rote LED an, ganz aufgeregt zu blinken. Wie soll man sich bei solchem Rot-Alarm nun verhalten? Am besten, wir fragen den König, was ihm nicht gefällt.

Das Problem ist einfach folgendes: Nachdem die Floppy ein eigener Computer ist, hat sie auch ihre eigenen Fehlermeldungen. Das heißt, ein Floppyfehler wird nicht wie unser bekannter «SYNTAX ERROR» (den haben Sie sicherlich im Laufe der Zeit sehr gut kennengelernt . . .) auf dem Bildschirm ausgegeben. (Der gehört ja zum Commodore, und so gut sind die nachbar-

schaftlichen Beziehungen nun auch wieder nicht.) Wie erfährt man aber dann, was man falsch gemacht hat? Nun, das Protokoll kennen Sie ja. Diesmal müssen wir nur ein kleines Programm daraus machen. Denn um einen Fehler von der Floppy einzulesen, brauchen wir eine spezielle Art des INPUT-Befehls. Und der kann ja nun nicht direkt ausgeführt werden. Unser Befehl lautet INPUT# (dieses Zeichen # wird meist File ausgesprochen und zeigt hier an, daß eine Eingabe von einem bestimmten Peripheriegerät kommen soll).

Um also den *Fehlerkanal* der Floppy abzufragen, brauchen Sie folgendes kleine Programm:

```
10 OPEN 1,8,15
20 INPUT#1,A,A$,B,C
30 PRINT A;A$;B;C
40 CLOSE 1
```

Was tut dieses Programm nun im einzelnen?

Immer, wenn die Floppy einen Fehler erkennt, blinkt die rote LED, und auf dem Fehlerkanal werden folgende Meldungen zum besseren Verständnis des Fehlers ausgegeben: seine Nummer, der Fehlertext, die Spur und der Sektor, wo der Fehler aufgetaucht ist. Diese vier Meldungen sind wie Eisenbahnwaggons, die bereitstehen, um abgeholt zu werden. Das oben gezeigte Beispielprogramm weist nun jeder dieser Fehlerinformationen eine Variable zu: A ist die Fehlernummer, A$ der Fehlertext, B die Spur und C der Sektor. Wenn Sie das jetzt ausprobieren, werden Sie wahrscheinlich durch Zeile 30 folgende Nachricht ausgedruckt bekommen:

```
0 OK 0 0
```

Das bedeutet, seine Hoheit sind mit uns zufrieden.

Wenn Sie die Floppy gerade erst eingeschaltet haben, kann der Text auch so aussehen:

```
73 CBM DOS V2.6 1541 0 0
```

Das ist eine Art königliches «Guten Morgen», ähnlich der Einschaltmeldung Ihres Commodore 64. Sollte irgendeine andere Fehlermeldung kommen, so liegt noch ein Fehler von vorhin auf dem Kanal. Was die einzelnen Fehlercodes genau bedeuten, entnehmen Sie bitte Ihrem Floppy-Handbuch.

Nun stellt sich aber wieder das bereits bekannte Problem ein. Sie haben noch immer Ihr Superprogramm (Sie wissen schon: das, was Sie vorhin

geschrieben haben) im Speicher, und beim Abspeichern desselben meldet die Floppy einen Fehler. Um den aber auslesen zu können, brauchen Sie das kleine Programm von vorhin. Um das aber mit Ihrem bereits vorhandenen Programm unter einen Hut zu bringen, muß Ihnen schon was Schlaues einfallen. Eine beliebte Verzweiflungstat war beispielsweise der Versuch, das Fehlerprogramm noch an den Anfang oder das Ende des vorhandenen Programms zu quetschen und dann gezielt aufzurufen.

Aber Sie sehen schon, allmählich wachsen die fehlenden Diskettenbefehle sich zu einem ernsthaften Problem aus. Und weil das jetzt so wirklich nicht mehr weitergehen kann, schaffen wir jetzt endgültig Abhilfe.

Commodore hat nämlich, offensichtlich im Bewußtsein dieser Schwierigkeiten, ein Programm auf die «TEST/DEMO»-Diskette gepackt, das sich als sehr hilfreich erweisen wird: das DOS 5.1.

Wir wollen es jetzt laden und fortan davon profitieren. Aber weiterhin gilt natürlich, daß alle Befehle auch nach der alten Zeremonie (also OPEN 1,8,15:PRINT#1," usw.) an die Floppy geschickt werden können. Aber wahrscheinlich werden Sie sich so an das DOS 5.1 gewöhnen, daß Sie es während des Programmierens ständig im Speicher haben werden. Dieses Programm ist sozusagen eine ständige Vertretung des FFS (Freien Floppy Staates), der uns so ziemlich alle Audienzen beim König abnimmt. Und weil es mit den dortigen Gepflogenheiten wesentlich besser vertraut ist und als Maschinenprogramm auch ein enger Vertrauter des 6502 ist, beschleunigt es die ganze Sache natürlich ungemein.

Und nun zum Bau des Botschaftsgebäudes: Auf Ihrer «TEST/DEMO»-Diskette müßte sich eigentlich ein Programm befinden, das «C-64 WEDGE» heißt. Das soll nur heißen, daß das Programm dem Commodore 64 angepaßt wird. Wenn Sie dieses Programm laufen lassen, dann lädt und startet es das DOS 1.5 ganz von alleine. Sie brauchen nur

```
LOAD"C-64 WEDGE",8
```

und danach RUN einzugeben. Damit Sie später auch eigene Programme schreiben können, die das DOS 5.1 automatisch nachladen, wollen wir uns dieses WEDGE-Programm mal genauer ansehen.

```
10 IF A=0THENA=1:LOAD"DOS 5.1",8,1
20 IF A=1THEN SYS12*4096+12*256
30 NEW
```

Grob erklärt, passiert folgendes. In Zeile 10 wird das eigentliche DOS 5.1

geladen. Wenn Sie es auf Ihrer Diskette suchen, es ist 4 Blocks lang. Da es sich um ein Maschinenprogramm handelt, das nicht am BASIC-Anfang liegt, muß es mit «8,1» *absolut* geladen werden. In Zeile 20 wird es dann gestartet. Die Startadresse ist 12 * 4096 + 12 * 256, also 52224. Damit liegt es genau in dem 4K-RAM-Block, der von 49152 bis 53248 reicht. Somit wurde also auch für diesen Speicherbereich noch eine sinnvolle Anwendung gefunden. (Vergleichen Sie dazu auch das Speicherkapitel.)

In Zeile 30 wird dann der BASIC-Speicher durch des NEW wieder gelöscht, das heißt, das Programm räumt sich sozusagen selbst auf. Ab hier könnte aber genausogut ein normales BASIC-Programm stehen, beispielsweise eine Auswahltabelle für alle Programme, die sich auf dieser Diskette befinden. Was Sie jetzt noch beschäftigen dürfte, ist der tiefere Sinn der Variablen A. Dazu muß man folgendes wissen: Wenn ein LOAD-Befehl innerhalb eines Programms auftaucht, geht der Computer davon aus, daß ein neues BASIC-Programm geladen wird. Also springt er nach dem Laden an den Anfang des – vermeintlich neuen – Programms und führt es aus. Wenn nun aber, wie in unserem Fall, ein Maschinenprogramm geladen wird, ändert sich ja nichts an dem BASIC-Programm, das heißt, es bleibt im Speicher, wie es war. (In diesem Fall spricht man von einem LOADER-Programm – es lädt ein anderes Programm.) Danach springt der Computer an den Anfang des Programms und startet es wieder in dem sicheren Glauben, es handelte sich um ein neues Programm. Also würde noch mal geladen werden und noch mal und noch mal und immer so weiter, bis zum Tag des jüngsten Stromausfalls.

Durch den Trick mit der Variablen läßt sich das vermeiden. Denn die Variablen werden bei diesem Vorgang nicht gelöscht. Man spricht dabei übrigens von einem *Warmstart*.

Deshalb ist A am Anfang 0. Die IF...THEN-Bedingung ist erfüllt, also wird A auf 1 gesetzt und dann der LOAD-Befehl ausgeführt. Treuherzig wie er ist, springt der Computer wieder zum Anfang des Programms zurück (bildlich natürlich nur!) und will wieder anfangen. Aber ... aha oder besser gesagt A(aah). Er merkt nämlich plötzlich, daß A nicht mehr 0 ist. Also gilt die Bedingung in Zeile 10 nicht mehr, und in seiner Not blickt sich der Commodore suchend nach Hilfe um. Die findet er dann auch in Form der Zeile 20, die ihm sagt, daß er einen SYS-Aufruf machen soll (also ein im Speicher befindliches Maschinenprogramm aufrufen soll). Und mit diesem Aufruf wird das DOS 5.1 gestartet. Während sich unser Commodore noch ganz verwundert umblickt und die Betriebsamkeit, die hinter ihm ausgebrochen ist, anstarrt, rennt er vorne in Zeile 30 in das NEW. Und damit vergißt er schlagartig, daß

er sich eben noch gewundert hat, und läßt dafür das DOS 5.1 laufen. Gar nicht so dumm, das alles, was?

Wenn Sie jetzt also das RUN eingegeben haben und damit Ihren Computer durch die Höhen und Tiefen seines Daseins geschickt haben, werden Sie einen entsprechenden Text finden, der Ihnen mitteilt, wer der Autor ist und wer infolgedessen auch Copyright auf dieses Programm hat. Wir sollten jetzt wirklich eine Schweigeminute für diesen Mr. Fairbairn halten und ihm für dieses Programm danken.

Geben Sie am besten gleich mal ein

```
>$
```

Also ein «Größer als»- und ein Dollarzeichen, dann das obligatorische ⟨RETURN⟩.

Wie Sie jetzt sehen, zeigt uns der Computer nun den Inhalt der Diskette, und zwar, ohne sich weiter um das Programm im Speicher zu kümmern. Das ist doch schon ganz vorteilhaft.

Aber es geht noch weiter.

Durch alleinige Eingabe von

```
>
```

und ⟨RETURN⟩ können Sie jederzeit den Fehlerkanal abfragen. Auch für die Lade- und Speicheroperationen gibt es jetzt Kürzel. Ein Schrägstrich, gefolgt von einem Programmnamen, wirkt wie unser bisheriges LOAD "PROGRAMMNAME",8.

Wenn Sie statt dessen das Prozentzeichen (%) voranstellen, wirkt dies wie LOAD"PROGRAMMNAME",8,1. Das brauchen Sie immer dann, wenn Sie ein Maschinenprogramm laden wollen.

Mit einem Pfeil nach oben (↑) als erstem Zeichen, gefolgt von dem Programmnamen, wird das Programm nicht nur geladen, sondern auch automatisch gestartet.

Ein Pfeil nach links (←) steht für das SAVE-Kommando.

Aber jetzt kommt das Beste: Nach einem ">" (Größer-als-Zeichen) kann ein beliebiger Floppy-Befehl stehen. Wenn Sie jetzt also sagen wollen: «Herr Botschafter, teilen Sie Ihrem König bitte mit, daß die eingelegte Diskette formatiert werden soll», so geht das einfach so:

```
>N:TEST,01
```

Sie erinnern sich, die ausführlichere Version von früher war:

```
OPEN 1,8,15:PRINT#1,"N:TEST,01":CLOSE15
```

Sie werden uns jetzt wahrscheinlich zustimmen, wenn wir sagen, daß das DOS 5.1 die ganze Sache schon etwas einfacher und komfortabler macht.

Mit diesem starken Verbündeten wollen wir uns jetzt auch noch an die restlichen Befehle für die Floppy machen.

Legen Sie wieder die Diskette ein, die wir ganz zu Anfang formatiert haben. Ein

```
)$
```

wird Ihnen zeigen, daß sich bisher nur ein Programm auf dieser Diskette befindet, und zwar das, das wir vorhin abgespeichert haben. Dem kann abgeholfen werden. Der COPY-Befehl kopiert Programme oder Dateien (allgemeiner Ausdruck dafür ist *Files*) innerhalb einer Diskette. Dazu müssen sie nur einen neuen Namen angeben.

```
)C:KOPIE=PROGRAMM
```

Nun wird ein zweites Programm auf die Diskette geschrieben, das sich in nichts von dem ersten PROGRAMM unterscheidet. Das sehen Sie zum Beispiel an der exakt gleichen Anzahl der belegten Blocks in der Directory. Der König hat also seine Untertanen angewiesen, zwei völlig identische Felder anzulegen. Und die – gewohnt, zu gehorchen – fragen nicht lange, sondern tun es. Bevor wir uns aber jetzt auf den Weg zur Monokultur begeben, wollen wir noch ein bißchen was ändern. Es könnte ja sein, daß uns plötzlich einfällt, daß KOPIE doch kein angemessener Name für den neuen Zögling ist. Wir wollen ihn umtaufen. Kein Problem. Dafür gibt es den RENAME-Befehl. Wenn das Programm KOPIE ab sofort PROGRAMM 2 heißen soll, wenden wir uns mit unserem Anliegen einfach an die Floppysche Botschaft.

```
)R:PROGRAMM 2=KOPIE
```

Wenn die Floppy mit dieser kurzen Aktion fertig ist und Sie sich mit

```
)$
```

das Inhaltsverzeichnis ansehen, werden Sie feststellen, daß das Programm KOPIE verschwunden ist. Dafür gibt es jetzt ein PROGRAMM 2. Wenn Sie sich darüber wundern, wie schnell das gegangen ist, hier noch ein kurzes Wort zur Klärung. Bei dem letzten Befehl wird nicht etwa KOPIE in den Speicher gelesen und dann als PROGRAMM 2 wieder auf die Diskette

geschrieben. Alles, was die Floppy tut, ist, den neuen Namen über den alten im Directory (also auf der Spur 18) zu schreiben. Das Programm selbst bleibt davon unberührt.

Weil wir jetzt aber zweimal dasselbe Programm auf der Diskette haben, was ja wirklich nicht sein muß, können wir auch das alte PROGRAMM wieder löschen.

Dazu gibt es den SCRATCH-Befehl. Aber Vorsicht. Zum einen ist dieser Befehl sehr endgültig und zum anderen: Verwenden Sie hier keine Jokersymbole. Sie würden nämlich nicht nur das erste der in Frage kommenden Files, sondern alle in Frage kommenden Files löschen. Aber zurück zum Löschen: Der Befehl heißt

```
)S:PROGRAMM
```

Damit wird das Programm gelöscht. Das heißt, die Blocks, die es ursprünglich belegte, werden wieder als frei gekennzeichnet und können beim nächsten SAVE wieder neu beschrieben werden.

In jedem Fall sollte man diesen Befehl nur nach zweimaligem Nachdenken anwenden.

Jetzt fehlt uns nur noch ein wichtiger Floppy-Befehl: VALIDATE. Wir haben Ihnen ja schon von der BAM erzählt, der Karte, auf der alle belegten Blocks gekennzeichnet sind. Leider ist ja nun unser Monarch nicht mehr der allerjüngste. Und auch seine Berater und Bediensteten nehmen es nicht immer allzu genau. Deshalb kommt es vor, daß hie und da mal ein Block als belegt dazugeschmuggelt wird, der es gar nicht ist. Deshalb sollte man von Zeit zu Zeit mal eine Kontrolle durchführen.

Ähnlich, wie das der Bundesrechnungshof macht, geschieht das auch durch den VALIDATE-Befehl.

```
)V
```

ist alles, was man dabei eingeben muß. Der Rest wird von ganz allein durchgeführt. Wenn man das alle paar Wochen macht, dann kommt schon der eine oder andere Block dabei heraus.

Damit wäre unser kleiner Kurs im Vokabular der Floppy eigentlich beendet. Es gibt zwar noch einige andere Befehle. Aber die würden wirklich den Rahmen dieses Kapitels sprengen. Außerdem braucht man die selten schon am Anfang. Wenn es soweit ist, sollten Sie sich alles dazu Nötige im Floppy-Handbuch durchlesen.

Bleibt nur noch eine Kleinigkeit: der Klammeraffe. Den finden Sie hier

ausnahmsweise nicht im Zoo, sondern auf der Tastatur. Gemeint ist damit eine Art Kringel. Darstellen tut er aber ein a im Kringel. Das Ganze sieht dann so aus: @.

Ursprünglich ist er ein amerikanisches Symbol für «at». Aber dafür haben sich Programmierer nie sonderlich interessiert. Nur damit das Kind – pardon, der Affe – einen Namen hat, kam man eben auf Klammeraffe. Nachdem wir beide noch nie einen echten Klammeraffen gesehen haben, können wir leider nicht feststellen, ob er tatsächlich so aussieht. Wenn aber doch, dann täte es uns leid für ihn.

Von der freien Wildbahn wieder zurück ins traute Wohnzimmer. Hier kann uns der Klammeraffe zwei gute Dienste leisten. Er kann zum Beispiel beim DOS 5.1 als Ersatz für das "⟩" verwendet werden.

Wichtiger aber ist, daß er dafür sorgt, daß ein File unter seinem eigenen Namen überschrieben werden kann.

Probieren Sie doch mal, irgendein Programm unter dem Namen «PROGRAMM 2» auf die Diskette von vorhin zu schreiben. Man wird Ihnen sehr bald mitteilen, daß ...

```
63, FILE EXISTS, 00, 00
```

daß also ein Programm mit diesem Namen bereits auf der Diskette steht. Der Vorteil des Ganzen ist eigentlich klar: So kann man verhindern, daß man ein Programm aus Versehen überschreibt.

Bei Dateien kann das aber äußerst unangenehm sein. Denn wenn eine Datei aktualisiert wird oder auch ein Programm verbessert wurde und deshalb neu auf Diskette geschrieben werden soll, muß ja erst das alte Programm gelöscht werden. Damit man das nicht immer vorher machen muß, kann der Klammeraffe angewendet werden, und zwar, indem man ihn folgendermaßen einsetzt:

```
SAVE "@:PROGRAMM 2",8
```

oder

```
←@:PROGRAMM 2
```

Aus verschiedenen Gründen, die durch die Arbeitsweise des DOS bedingt sind, kann es bei diesem Modus aber zu Fehlern kommen. Die sicherere Methode ist die, das alte File mit SCRATCH zu löschen, und dann das neue normal abzuspeichern. Durch den dann frei gewordenen Platz wird das DOS dieses Programm auch «über» das alte schreiben.

Nachdem Sie jetzt das DOS 5.1 als freundlichen Helfer kennengelernt haben, kann es sein, daß Sie es gern auch auf Ihren anderen Disketten hätten, um nicht immer von der «TEST/DEMO»-Diskette abhängig zu sein. Wenn das so ist, müssen Sie es kopieren. Nun haben Sie vielleicht entdeckt, daß sich auf der «TEST/DEMO»-Diskette auch ein Kopierprogramm mit Namen COPY ALL befindet. Und tatsächlich, dieses Programm würde unser DOS 5.1 auch kopieren – vorausgesetzt, wir haben zwei Floppylaufwerke. Wenn Sie es nämlich starten, stellen Sie fest, daß das Programm mit einem gar nicht funktioniert. Nun bringt das vielleicht für Commodore ein Umsatzplus, uns aber erst mal nur Verdruß. Weil das DOS 5.1 ein Maschinenprogramm ist, läßt es sich nicht einfach mit einer LOAD&SAVE-Kombination kopieren.

Aber es geht auch anders. Wir wollen Ihnen hier kurz zeigen, wie man nun das DOS 5.1 doch dahin kriegt, wo man es haben will. Voraussetzung für diese Methode ist allerdings, daß sich DOS 5.1 schon im Speicher befindet.

```
10 OPEN 1,8,2,"DOS 5.1,P,W"
20 PRINT#1,CHR$(0);
30 PRINT#1,CHR$(204);
40 FOR X=0TO870
50 PRINT#1,CHR$(PEEK(52224+X));
60 NEXT X:CLOSE1
```

An diesem Programm läßt sich auch sehr schön erkennen, wie Programme auf der Floppy aufgezeichnet werden.

In Zeile 10 wird ein File zum Schreiben geöffnet. P steht für Programm, W für WRITE (= schreiben). Darüber werden wir uns etwas später noch mal genauer unterhalten.

In Zeile 20 und 30 wird der Floppy die Anfangsadresse des Programms mitgeteilt, und zwar in Low Byte (0) und High Byte (204). Denn 204 * 256 ist ja 52 224.

Wozu das? Wenn ein Programm absolut, also mit "...,8,1" geladen wird, muß der Computer ja erfahren, wo er das Programm hinpacken soll. Diese beiden Bytes entsprechen etwa dem Schild «Ich will nach München», das man einem kleineren Kind in der Eisenbahn um den Hals hängt, damit es auch dort ankommt. Oder auch dem Schild «Ich will nach München», das man bei größeren Kindern in der Nähe der Autobahnausfahrt öfter antrifft.

Zeile 40 zählt dann von 0 bis 870, denn so lang ist unser Programm (das DOS 5.1) eben: 870 Bytes.

In Zeile 50 werden die Bytes, die in den Zellen 52224 bis 53094 stehen,

hintereinander auf Diskette geschrieben. In Zeile 60 schließlich wird das File ordnungsgemäß geschlossen. Das ist absolut wichtig, denn erst nach diesem Befehl werden alle wichtigen Informationen in die Directory und die BAM übernommen. Also immer daran denken!

Wie Ihnen vielleicht aufgefallen ist, werden alle Bytes als CHR$ (sprich Charstrings) zur Floppy geschickt. Das ist bei der Übergabe von Daten an Peripheriegeräte allgemein so üblich und auch zweckmäßig, denn so können die Bits gleichzeitig übergeben werden und nicht erst die 2, dann die 0 und schließlich die 4 für den Wert 204.

Im Grunde genommen haben die CHR$ auf Peripheriegeräte genau dieselbe Funktion wie die CHR$ auf dem Bildschirm. Sie stehen nur für den entsprechenden Code. (Bei der Bildschirmausgabe macht erst VIC ein lesbares Zeichen daraus.)

Und damit die Werte schön hintereinander reinkommen, steht der ";" hinter jedem Wert. Wenn Sie das Programm mit

```
RUN
```

starten, wird das DOS 5.1 auf die Diskette geschrieben, die sich gerade im Laufwerk befindet. Von dort kann es dann, wie vorhin gezeigt, geladen werden. Zum DOS 5.1 noch zum Abschluß zwei Bemerkungen. Es ist so programmiert, daß es sich ständig auf die Gerätenummer 8 bezieht. Sollten Sie jedoch mehrere Floppys einsetzen wollen, können Sie mit

```
〉# Gerätenummer
```

dem Botschafter ein anderes Königreich (sprich Ihre zweite Floppy) unterjubeln.

```
〉#9
```

schaltet DOS 5.1 auf Geräteadresse Nummer 9.

```
〉#8
```

schaltet wieder zurück.

Sie können DOS-5.1-Befehle auch innerhalb Ihrer Programme verwenden. Allerdings mit einer kleinen Änderung: Hinter dem "〉" müssen die Befehle nun in Anführungszeichen stehen, um Verwechslungen, denen der BASIC-Interpreter sonst erliegen könnte, zu vermeiden. Beispiele hierzu:

```
10 〉"$"
```

zeigt die Directory während eines Programms an.

```
10 )
```

fragt den Fehlerkanal ab

```
10 )"I"
```

liest die BAM über den Befehl «Initialize» ein.

Und noch ein letzter Tip: Sie können, wann immer Sie diplomatische Verwicklungen mit anderen Maschinenprogrammen befürchten, die Kontrolle über die Botschaft abbrechen.

```
)Q
```

für Quit (= verlassen) ist dann der notwendige Befehl. Das heißt aber noch lange nicht, daß Sie damit den Botschafter des Landes verweisen. Genauso wie in Camp David können Sie die Verhandlungen immer wieder aufnehmen. Mit

```
SYS 52224
```

Schön, oder?

Ein Umsteigebahnhof für Bits

Im nun folgenden erfahren Sie so ziemlich das Letzte ...

Wir haben im Verlauf des letzten Abschnittes mehrmals den OPEN-Befehl mit verschiedenen Sekundäradressen verwendet und Sie dabei immer auf später vertröstet oder uns mit dem Ausdruck «Einstellen von verschiedenen Betriebsmodi» vor einer Erklärung gedrückt. Das wollen wir jetzt nachholen.

Zunächst einmal grundsätzlich: Mit OPEN wird ein sogenannter Kanal eröffnet. Das hat nichts mit Rhein-Main-Donau oder so zu tun, sondern meint eine Art Standleitung von und/oder zur Floppy. Eine bestimmte Leitung benutzen Sie zum Beispiel ständig: den Fehlerkanal. Aber es gibt noch andere. Einen solchen haben wir zum Beispiel bei unserem DOS-Kopierprogramm verwendet. Aber die Anzahl der möglichen Kanäle ist begrenzt. Denn jeder Kanal wird mit einem sogenannten Pufferspeicher verbunden. Der Name sagt eigentlich schon, was diese Dinger tun. Dasselbe, was auch Puffer bei der Eisenbahn tun. Sie fangen auf – in unserem Fall Bits, bei der Bundesbahn Stöße. Das hat einen einfachen Grund: Bei der Bundesbahn verhindert das, daß regelmäßig Waggons zu Schrott gefahren werden, und bei uns dient es der Schnelligkeit. Das Kabel, mit dem die Floppy an den

Commodore angeschlossen ist, hat zwar 6 Leitungen, aber nur eine davon dient tatsächlich der Datenübertragung. Die anderen führen irgendwelche Strom- und Masseleitungen oder die «Aufgepaßt, es kommen Daten!»-Leitungen. Da wir also nur eine Leitung zur Datenübertragung haben, müssen die Bits seriell, soll heißen hintereinander, übertragen werden. Die Straßen in Richtung Floppy sind also so eng, daß sich Reiter und Fußvolk nur hintereinander fortbewegen können. Das ist auch der Grund, warum die VC-Floppys im Vergleich zu vielen anderen Laufwerken so langsam sind. Die anderen arbeiten nämlich mit einer breiten achtspurigen Autobahn, auch Parallelbus genannt.

Da die Bits also eher tröpfchenweise eintrudeln, wäre es viel zu aufwendig, jedes einzelne zum Schreib-/Lese-Kopf zu schicken. Statt dessen warten sie im Pufferspeicher sozusagen auf ihren Anschlußzug in Richtung Diskette – wie in einem Umsteigebahnhof.

Die Floppy hat aber nun mal nur eine begrenzte Anzahl solcher Pufferspeicher. Es sind genau fünf. Davon werden 2 vom DOS ständig reserviert: einer für Programme lesen, einer für Programme schreiben. So bleiben nur noch drei für den Programmierer übrig. Und die dritte Zahl beim OPEN-Befehl gibt eben an, welcher Puffer verwendet werden soll.

```
OPEN 1,8,X,...
```

Wie gesagt, zwei Puffer sind reserviert: nämlich 0 und 1. Die Puffer 2 bis 4 sind frei. Man kann zwar für das X von oben auch die Zahlen 5 bis 14 einsetzen, dann werden die Puffer von der Floppy der Reihenfolge nach angelegt. Solche Zahlen sind aber unpraktisch, weil man so schneller die Übersicht verliert, welcher Puffer zu welchem Kanal gehört und wie viele Puffer noch frei sind. Die Anzahl ist nämlich immer auf drei begrenzt.

Hinter dem Puffer wird in den Anführungszeichen noch angegeben, um welchen Dateityp es sich handelt und ob gelesen oder geschrieben werden soll.

Für den Dateityp gibt es verschiedene Möglichkeiten:

P für Programm
S für sequentielle Datei
R für relative Datei
U für User-File

Darauf können wir aber an dieser Stelle nicht weiter eingehen. Diese Kürzel entsprechen dann den in der Directory angegebenen Dateitypen, also

PRG
SEQ
REL
USR

Wir teilen dem König mit, welche speziellen Arten an Daten wir anbauen lassen möchten und welche Datenfelder deshalb aufgemacht werden müssen. Es muß auch bekannt sein, ob gesät oder geerntet werden soll bzw. geschrieben oder gelesen. Deshalb steht R für READ und W für WRITE.

Am besten, wir sehen uns einige Beispiele an:

```
OPEN 1,8,2,"HALLO,P,R"
```

Das Programm (dafür steht P) mit Namen «HALLO» wird über Puffer 2 (siehe letzte Zahl) zum Lesen (dafür steht R) geöffnet.

```
OPEN 1,8,3,"@:DOS 5.1,P,W"
```

Ein (evtl. schon existierendes) Programm «DOS 5.1» wird über Puffer 3 zum Schreiben (W) geöffnet. Man kann also, wie beim DOS, ein Programmfile auch künstlich zum Lesen oder Schreiben eröffnen, es muß nicht immer über LOAD oder SAVE gehen. Wie das geht, haben wir ja schon im DOS-Kopierprogramm erklärt.

```
OPEN 2,8,4,"ADRESSEN,S,R"
```

Die sequentielle Datei «ADRESSEN» wird über Puffer 4 zum Lesen geöffnet.

Was wir jetzt noch kurz zeigen wollen, ist, wie man eine sequentielle Datei verwenden kann.

Wie Sie sicher schon einmal gehört haben, kann man bei einem Computer nicht nur Programme abspeichern, sondern auch Daten, zum Beispiel Adressen, Titel von Büchern und Schallplatten und so weiter. Eine Ansammlung solcher Daten nennt man Datei. Nun gibt es natürlich verschiedene Möglichkeiten, solche Daten abzuspeichern. Die einfachste und damit am weitesten verbreitete Art ist die sequentielle Datei. Bei ihr werden alle Daten einfach hintereinander auf die Diskette geschrieben – ähnlich einer Gänsefamilie, kommt eines nach dem anderen.

Man kann diese Art der Datenspeicherung auch mit einem Telegramm vergleichen:

Vorname – stop – Nachname – stop – Straße – stop – usw.

Die Funktion des Stop, also das Trennen einzelner Daten, übernimmt in diesem Fall der Code für ⟨RETURN⟩, das heißt CHR$ (13).

Das ist noch ein Überbleibsel aus der Zeit, wo hauptsächlich Drucker angesprochen wurden: Da war CHR$ (13) das Zeichen für «Line Feed». Wenn der Drucker diesen Code entdeckte, wußte er, daß jetzt die Zeile zu Ende ist.

Genauso weiß die Floppy, daß bei einem CHR$ (13) die entsprechende Eintragung in die Datei zu Ende ist.

Es gibt nun verschiedene Möglichkeiten, wie so ein CHR$ (13) gesendet werden kann. Darauf wollen wir aber erst beim Drucker genauer eingehen. Hier sei Ihnen erst einmal gesagt, daß der Befehl

PRINT# N

wenn dem Text, der danach kommt, kein ";" oder "," folgt, automatisch einen CHR$ (13) sendet.

Nachdem wir alle mal klein angefangen haben, wollen wir jetzt auch eine ganz kleine Adreßdatei mit einer einzigen Adresse anlegen.

```
10 OPEN 1,8,2,"ADRESSDATEI,S,W"
20 PRINT#1,"MUELLER"
30 PRINT#1,"HANS OTTO"
40 PRINT#1,"ROSENWEG 27"
50 PRINT#1,"8001 BAYERNHAUSEN"
60 PRINT#1,88811277
70 CLOSE1
```

Wir haben also alle Daten in unsere Adreßdatei geschrieben. Beachten Sie, daß die (fiktive!!) Telefonnummer in Zeile 60 kein String, wie die anderen Daten, sondern eine *numerische* Variable ist. Das müssen wir beim Auslesen auch wieder berücksichtigen. Sonst gibt es einen

FILE DATA ERROR

Ein Programm, das unsere Adresse wieder liest, könnte ungefähr so aussehen:

```
10 OPEN 1,8,3, "ADRESSDATEI,S,R"
20 FOR X=1TO4
30 INPUT#1,A$
40 PRINT A$
```

```
50 NEXT X
60 INPUT#1,A
70 PRINT A
80 CLOSE 1
```

Wir eröffnen also unsere Datei zum Lesen (Zeile 10), lesen mit der FOR... NEXT-Schleife von 20 bis 40 Daten ein und drucken sie dann auf den Bildschirm. Zeilen 60 und 70 machen dasselbe mit der Telefonnummer. Ein kleiner Tip: Einfacher geht's, wenn Sie auch Zahlen wie Strings behandeln. Allerdings können Sie dann die Stringvariablen mit Zahlen nicht zum Rechnen verwenden.

In Zeile 80 wird die Datei wieder geschlossen.

Probieren Sie es aus. Sie könnten jetzt eigentlich ein Programm schreiben, das mit mehreren Adressen und komfortablerer Eingabe arbeitet. Sie müssen ja einfach nur statt des Namens aus dem ersten Listing eine Variable nehmen, die dann vorher mittels INPUT einer Adresse zugewiesen wird.

Wenn Sie dasselbe mit dem Kassettenrecorder machen wollen, müssen Sie nur Gerätenummer 1 verwenden und als Sekundäradresse 0 für Lesen und 1 für Schreiben. Zusätze wie «S,W» oder «S,R» entfallen. Lesen Sie zu einer genaueren Beschreibung das Kapitel «Fortgeschrittene Kassettenoperation» auf Seite 109 in Ihrem Commodore-Handbuch.

Und damit wären wir – zumindest, was die Floppy betrifft – so ziemlich am Ende.

Wie immer gilt, nur Übung macht den Meister. Und Sie haben ja hoffentlich gesehen, daß an SEQ-Dateien überhaupt nichts Geheimnisvolles ist.

Also, auf geht's. Die Adreßverwaltung wartet ...

Der Commodore lernt schreiben

Jetzt wollen wir noch einen Blick auf ein anderes wichtiges Peripheriegerät werfen – den Drucker. Davon gibt es ja erst mal eine ganze Menge, zum einen die, die Commodore selbst anbietet. Dann auch andere Firmen, die Commodore-*kompatible* Drucker herstellen. Und schließlich gibt es noch verschiedene Interfaces, *Schnittstellen*, mit denen man auch noch viele andere Fabrikate anschließen kann. Bei diesen Interfaces kommt es darauf an, die sequentiellen Signale der Commodore-Schnittstelle so umzumodeln, daß der entsprechende Drucker sie auch versteht. Die meisten Drucker haben, wie

die Floppy, eine eigene Regierung, also einen eigenen Prozessor mit eigenem Betriebssystem, der auf ganz unterschiedliche Befehle reagiert. Für richtige, spezifische Informationen müssen wir Sie deshalb auch an das entsprechende mitgelieferte Druckerhandbuch verweisen.

Wir möchten hier nur einige ganz allgemeine Anwendungen und Vorgänge aufzeigen und den einen oder anderen Trick verraten.

Am häufigsten wird ein Drucker wohl dann eingesetzt, wenn es um das Auslisten von Programmen geht. Wie geht das?

Es gibt einen Befehl, der die Ausgabe vom Bildschirm auf ein Peripheriegerät umlenkt. Das ist der

CMD

-Befehl. Um ihn anwenden zu können, müssen wir wieder diplomatisch tätig werden. Unser rotes Telefon geht diesmal zum Gerät Nummer 4. Manchmal haben Drucker aber auch die Nummern 5 oder 6. Schauen Sie im Handbuch nach, oder fragen Sie gegebenenfalls Ihren Händler.

Mit diesem Befehl leiten wir die Bildschirmausgabe um und listen das Programm aus. Die ganze Syntax sieht dann so aus:

```
OPEN 4,4: CMD4: LIST
```

Wenn der Drucker fertig ist, müssen wir ihm natürlich mitteilen, daß er jetzt erst mal keine Daten mehr bekommen wird. Gleichzeitig schalten wir die Ausgabe zurück auf den Bildschirm und schließen unseren Kanal wieder.

```
PRINT#4: CLOSE4
```

Das «leere» PRINT#4 lenkt die Ausgabe wieder auf den Bildschirm. Bei den meisten Druckern dürfte das so funktioniert haben.

Vielleicht hatten Sie aber auch Pech. Dann sieht das Listing entsprechend der Länge des Programms aus, als ob eine Horde wilder Ameisen auf engstem Raum das Papier überquert hätte – und alle mit schmutzigen Füßen. Die gleiche Zeile wurde immer wieder überdruckt, bis das Listing zu Ende war.

Der Grund ist folgender: Ihrem Drucker hat der Zeilenvorschubcode gefehlt. Deshalb druckte er das ganze Listing in eine Zeile. Das spart zwar auf Dauer immens Papier, hilft aber nicht gerade bei der Programmdokumentation. Das alles kommt nur daher, daß einige Drucker, wenn ein Zeilenendcode (CHR$ (13)) kommt, automatisch einen Zeilenvorschubcode (CHR$ (10)) anfügen. Das heißt, wenn diese Drucker merken, daß die Zeile zu Ende ist, schieben sie von selbst das Papier weiter. Nur Ihr Drucker tut das scheinbar

nicht. Ihm muß unser Commodore sagen, was Sache ist. Und das ist auch gar kein Problem, wenn Sie eine Kanalnummer verwenden, die höher als 127 ist. Dann wird das am Ende jeder Zeile automatisch gemacht. Probieren Sie also in diesem Fall:

```
OPEN 128,4: CMD 128: LIST
```

Das müßte das erhoffte Listing auf den Drucker bringen. Denken Sie aber bei Listings auch daran, daß Fremddrucker meist gar nicht in der Lage sind, die Steuerzeichen richtig zu drucken, also invertierte Herzchen und dergleichen.

Wenn bei Ihnen schon beim ersten Versuch alles geklappt hat, aber Sie nicht der Versuchung widerstehen konnten, das obige dennoch auszuprobieren, dann haben Sie halt jetzt doppelten Zeilenabstand. Auch nicht schlecht, oder?

Auf jeden Fall sollten Sie nach dem Listing eingeben:

```
PRINT#128: CLOSE128
```

Die meisten Commodore-kompatiblen Drucker haben zwei Druckmodi: Den Grafikmodus, den wir schon von den Grafikzeichen kennen (also nur Großbuchstaben- und Grafikzeichendarstellung) und den Textmodus (Groß- und Kleinbuchstaben). Je nachdem, was und wie Sie drucken wollen, müssen Sie einen dieser beiden Modi wählen. Dazu dient meist die Sekundäradresse

```
OPEN 4,4,7
```

Damit wählen Sie den Textmodus. Ab jetzt müßte alles in Groß- und Kleinbuchstaben gedruckt werden.

```
OPEN 4,4,0
```

wählt dagegen den Grafikmodus mit Großbuchstaben und Grafikzeichen. Wenn das nicht so ist, müßte in Ihrem Druckerhandbuch unter diesen Stichworten etwas stehen.

Wenn Sie jetzt aber keine Listings, sondern Daten und Texte von einem Programm ausdrucken wollen, müssen Sie sich an eine ähnliche Zeremonie halten wie bei den Verhandlungen mit der Floppy.

```
10 OPEN 4,4
20 PRINT#4,"DIES IST EIN DRUCKERTEST."
```

Wenn nötig, sollten Sie ab jetzt immer selbständig eine Kanalnummer über 127 wählen.

Wenn alles klappt, druckt das Programm den obenstehenden Text aus. Wenn Sie jetzt den automatischen Zeilenende-/Zeilenvorschubcode unterdrücken wollen, verwenden Sie einfach einen Strichpunkt als Trennungszeichen, wie Sie das ja schon von den PRINT-Befehlen her kennen:

```
10 PRINT#4,"DIES IST EIN DRUCKERTEST";
20 PRINT#4," FUER IHREN DRUCKER"
```

Ähnlich wie bei den Steuerzeichen für Farben und Cursorbewegung bei Ihrem Commodore 64, gibt es auch Steuerzeichen für Ihren Drucker.

```
CHR$ (14)
```

steht zum Beispiel meistens für Breitschrift.

Solche Steuerzeichen werden folgendermaßen an den Drucker geschickt:

```
20 PRINT#4, CHR$ (14);"BREITSCHRIFT";
```

Zum Ausschalten könnte zum Beispiel CHR$ (15) dienen:

```
25 PRINT#4, CHR$ (15)
```

Je nachdem, welchen Drucker Sie haben, eröffnet sich ein großes Feld für neue Experimente. Dieser Abschnitt sollte Ihnen nur zeigen, was man so alles machen kann, wie die Commodore-Syntax zur Druckersteuerung ist, und im letzten Teil dafür sorgen, daß Sie verstehen, was die Anleitungsbücher mit CHR$ und dergleichen meinen. Wer weiß, vielleicht schreiben Sie ja schon in den nächsten Tagen an Ihrem ganz persönlichen Textverarbeitungsprogramm. Wir drücken Ihnen in jedem Fall alle vier Daumen dazu und hoffen, daß Sie viel Spaß dabei haben werden.

12
Anhang

Kapitelzusammenfassungen

«In der Kürze liegt die Würze». Diese Zusammenfassungen sollen noch mal stark in geraffter Form zeigen, was an einem Kapitel besonders wichtig war. So finden Sie später schnell die Informationen, die Sie brauchen. Außerdem richten sich diese Kapitel in Sprache und Aufbau mehr nach Fachbüchern und Fachzeitschriften. Dadurch werden Sie schon ein bißchen auf die Lektüre solcher Werke vorbereitet.

Die Grafikzeichen

Der Commodore 64 hat zwei alternative Zeichensätze fest eingebaut: Im sogenannten Grafikmodus stehen Großbuchstaben und die Grafikzeichen zur Verfügung. Im Textmodus gibt es Groß- und Kleinbuchstaben. Mit den Tasten ⟨C=⟩ und ⟨SHIFT⟩ können Sie zwischen diesen Modis wechseln.

Wenn innerhalb eines Programms zwischen den Zeichensätzen umgeschaltet werden soll, dienen dazu die folgenden Befehle:

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

schaltet den Textmodus ein.

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

schaltet den Grafikmodus ein.

Es kann aber nur immer ein Modus aktiv sein. Auf dem Bildschirm können sich also nicht gleichzeitig Zeichen befinden, die in den beiden Modis

unterschiedlich belegt sind (zum Beispiel Grafikzeichen und Kleinbuchstaben ...).

Die verwendbaren Grafikzeichen sind auf den Vorderseiten der Tasten dargestellt. Um sie zu erreichen, muß die ⟨SHIFT⟩- oder die ⟨C=⟩-Taste gedrückt werden. Dabei gilt: Die Symbole rechts werden mit ⟨SHIFT⟩ erreicht, die Symbole links mit ⟨C=⟩.

Im Textmodus muß für Großbuchstaben ⟨SHIFT⟩ gedrückt werden. Die Symbole, die mit ⟨C=⟩ angesprochen werden, sind auch im Textmodus verwendbar. Es handelt sich dabei vor allem um Symbole, die für die Darstellung von Tabellen usw. interessant sind.

Um die Umschaltung zu blockieren, können Sie den Befehl

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

verwenden. Zur Aufhebung der Blockierung dient

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

Welche Symbole in welchem Modus erreichbar sind, läßt sich auch im Anhang E des Commodore-64-Handbuchs finden (Seite 132).

Die Steuerzeichen

Die Tasten, die zur Steuerung auf dem Bildschirm dienen (also die ⟨CRSR⟩-Tasten, die Farbtasten, ⟨CLR/HOME⟩ und ⟨INST/DEL⟩, und verschiedene Codes, die durch ⟨CTRL⟩ erreicht werden), können auch programmiert werden.

Dazu wird der PRINT-Befehl verwendet. Der Steuerzeichenmodus wird durch ein Anführungszeichen (") oder ⟨INST⟩ aktiviert. Er stellt dann für alle oben aufgeführten Tasten ein besonderes Steuerzeichen dar. Abgeschaltet wird er durch ein weiteres Anführungszeichen oder ⟨RETURN⟩.

Bei der Programmausführung wirkt ein solcher PRINT-Befehl dann so, als ob die Tasten direkt gedrückt wurden.

Mit diesen Steuerzeichen können durch Löschen und Überdrucken auch bewegte Grafiken erzeugt werden.

Die Speicheraufteilung

Der Commodore 64 besitzt 64K-RAM und 20K-ROM. Außerdem werden noch Adressen durch andere Bausteine, wie den VIC-II oder den SID, benötigt.

Da der Mikroprozessor 6510 mit seinem 16-Bit-Adreßbus aber nur 64K gleichzeitig adressieren kann, muß er zwischen den verschiedenen Bausteinen, die auf einer Adresse liegen, umschalten können. Dazu werden jeweils 8K zu Blocks zusammengefaßt, die der 6510 dann ein- oder ausblendet. Zusätzlich greift der Videochip VIC auch noch auf das Charakter-ROM zu und benutzt dabei den Datenbus. Also müssen sich 6510 und VIC den Bus teilen. Dazu steuert VIC die Interrupts und benutzt genau dann den Datenbus, wenn der Prozessor ihn nicht benötigt.

Man kann aber einen Teil der ROM-Programme, nämlich den BASIC-Interpreter und das Betriebssystem (Kernal), in die RAM-Bausteine kopieren, die auf der gleichen Adresse liegen, und dann die ROMs ausblenden. So lassen sich dann im RAM Veränderungen am Betriebssystem vornehmen.

Der Adreßbereich von 65535 Bytes wird üblicherweise in 256 «pages» (Seiten) zu je 256 Bytes unterteilt. Die Seite Nr. 0 von 0 bis 255 heißt Zeropage. Hier legt der Rechner wichtige Informationen ab. Einige nützliche Adressen in diesem Speicherbereich werden im PEEK&POKE-Anhang beschrieben.

Auch die nächsten 3 Pages werden für interne Zwecke benötigt. Insbesondere wären da der Stack, eine Art Zwischenspeicher für den Prozessor, der Kassettenpuffer und das Bildschirm-RAM zu nennen.

Dann folgen 38911 Bytes RAM für BASIC-Programme.

Von Adresse 40960 bis 49151 ist dann normalerweise ROM eingeblendet: Hier befindet sich der BASIC-Interpreter.

Es folgt von 49152 bis 53247 ein besonderer RAM-Bereich für Maschinensprache-Programme, Befehlserweiterungen usw. Hier liegt beispielsweise auch das DOS 5.1.

Der nächste Adreßbereich ist sogar dreifach belegt: Hier liegen die I/O-Register (von VIC, SID und den Interface-Bausteinen). Außerdem befindet sich hier das Charakter-ROM, das aber im Normalfall vom Prozessor nicht gelesen werden muß. Schließlich liegen hier auch noch 4K-RAM.

In den letzten 8K von 57334 bis 65535 befindet sich das Kernal im ROM. Mit der Speicherzelle 1 läßt sich steuern, ob RAM oder ROM für einen bestimmten Speicherbereich eingeblendet ist.

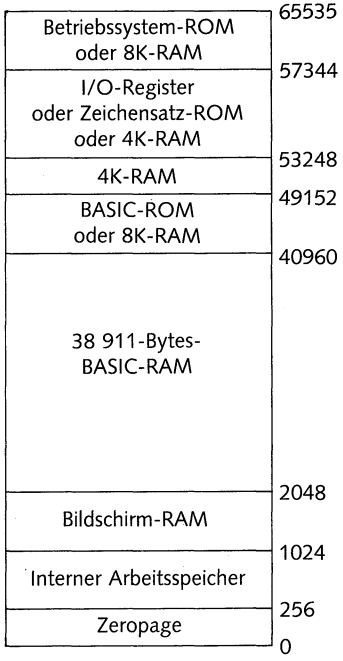


Bild 12.1 Speicheraufteilung

Dabei sind folgende Bits für die genannten Bereiche verantwortlich:

Bit	Bereich	Inhalt
0	40960 bis bis 49151	BASIC-ROM oder RAM
1	57344 bis 65535	KERNAL(ROM) oder RAM
2	53248 bis 57343	I/O-Register oder Charakter-ROM

Der jeweils zweite Bereich ist eingeblendet, wenn das entsprechende Bit aus ist. Die restlichen Bits dienen zur Steuerung des Kassettenrecorders und sollten nicht verändert werden.

Mit den Zeropageadressen 43 bis 46 und anderen läßt sich der Speicherbereich für BASIC-Programme verschieben. Näheres dazu finden Sie im PEEK- & POKE-Anhang. Diese Adressen benutzen das High-Byte/Low-Byte-Prinzip, das bei einem 8-Bit-Prozessor und einem 16-Bit-Adreßbus ständig verwendet

wird. Dazu wird eine 16-Bit-Zahl in der Mitte geteilt und so in zwei Pseudo-8-Bit-Zahlen umgewandelt.

So wird z. B. aus 2048 (00001000 00000000) das High Byte 8 (00001000) und das Low Byte 0 (00000000).

Ein Umrechnungsprogramm zwischen Binär- und Dezimalzahlen befindet sich im Listinganhang.

Selbstdefinierbare Zeichen

In der 8*8-Matrix, in der VIC die Zeichen auf dem Bildschirm darstellt, lassen sich auch eigene Zeichen entwickeln.

Dazu muß zuerst der Zeichensatz aus dem Charakter-ROM ins RAM, also in den Arbeitsspeicher kopiert werden. Dies bedeutet aber, die I/O-Register auszublenden und dafür das Charakter-ROM in den Adreßbereich des Prozessors zu übernehmen. Um das gefahrlos tun zu können, müssen während des Kopiervorgangs die Interrupts abgeschaltet werden.

Das folgende Programm übernimmt alle diese Aufgaben:

```
10 POKE56334,0
20 POKE1,51
30 FORX=0TO2048: POKE (Startadresse) + X,PEEK (53248 + X) :NEXT
40 POKE1,55
50 POKE56334,1
```

Der Zeichensatz befindet sich nun im RAM ab der angegebenen Startadresse. Um Verwicklungen mit einem BASIC-Programm zu vermeiden, sollte BASIC entsprechend verschoben werden (vgl. auch PEEK&POKE-Anhang ...)

Um auf den neuen Zeichensatz umschalten zu können, muß die Adresse 53272 entsprechend verändert werden:

```
POKE 53272, (PEEK(53272) AND 240))) OR (Zeiger)
```

Die möglichen Startadressen und die Zeiger dorthin entnehmen Sie bitte Tabelle 12.1.

VIC kann gleichzeitig nur 16K verwalten. Wenn Sie (wie im PEEK&POKE-Anhang erklärt) seinen Adreßbereich verschieben, müssen Sie die Startadresse dieser Speicherbank zu den oben aufgeführten Adressen addieren. Die Zeiger bleiben. Beachten Sie dabei aber, daß sich die Adresse des Bildschirm-RAMs evtl. auch verändert.

Startadresse	Zeiger
2048	2
8192	8
10240	10
12288	12
14336	14

Tabelle 12.1 Startadresse für Zeichensatz

In dem RAM-Bereich, wo sich der Zeichensatz nun befindet, kann er beliebig abgeändert werden. Ein Hilfsprogramm dazu befindet sich im Listinganhang.

Die Hires-Grafik

Im Hires-Modus benutzt VIC einen Speicherbereich von 8K als Bitmap. So erreicht er eine Auflösung von 320 * 200 Punkten.

Dabei wird das Bildschirm-RAM als Farbspeicher benutzt. Sie können natürlich diese Speicherbereiche wieder verschieben, aber der Normalfall dürfte folgende Aufteilung sein:

Farbspeicher	(Bildschirm-RAM)	1024 bis 2024
Bitmap	(Arbeitsspeicher)	8192 bis 16192

Diese Aufteilung und das Einschalten des Hires-Modus wird durch folgende Befehle erreicht:

```
POKE53272,24: POKE53265,59
```

Bei anderer Aufteilung muß der Inhalt der Adresse 53 272 entsprechend geändert werden. Nähere Informationen hierzu finden Sie im PEEK&POKE-Anhang.

Zum Abschalten geben Sie ein:

```
POKE53272,21: POKE53265,27
```

Die Hires-Grafik muß jetzt in die Bitmap gePOKEd werden. Einige Hilfsroutinen dazu finden Sie im Listinganhang.

Im Normalmodus kann für einen Block von 8 * 8 Punkten (entsprechend den normalen Zeichen) jeweils eine Vorder- und eine Hintergrundfarbe angegeben werden. Dazu POKEn Sie die beiden Farben nach folgender Formel in die entsprechende Adresse im Farb-RAM:

POKE (Adresse), (Vordergrundfarbe) * 16 + (Hintergrundfarbe)

Es gibt auch einen Mehrfarbmodus (Multicolormodus). Er wird eingeschaltet durch:

POKE53270,216

Wenn er aktiv ist, verringert sich die horizontale Auflösung auf 160 Punkte.

Je zwei Bits wird nun eine Farbe zugeordnet, abhängig von der Kombination der beiden: (Tabelle 12.2).

Der Multicolormodus läßt sich auch im Textmodus anwenden. Insbesondere wird dies interessant in Verbindung mit selbstdefinierten Zeichen. Die entsprechenden Adressen sind in Tabelle 12.3 zu ersehen.

Bitmuster	Farbadresse
00	53281
01	Bits 7 bis 4 im Bildschirm-RAM
10	Bits 3 bis 0 im Bildschirm-RAM
11	Bits 3 bis 0 im Farb-RAM (55296 bis 56296)

Tabelle 12.2 Bitmuster für Multicolormodus

Bitmuster	Farbadresse
00	53281
01	53282
10	53283
11	Bits 3 bis 0 im Farb-RAM (55296 bis 56296)

Tabelle 12.3 Adressen für Multicolortexte

Sie können mit

```
POKE 53270,200
```

den Multicolormodus abschalten.

Im Textmodus läßt sich außerdem noch der Hintergrundfarbmodus (Extended Background Color Mode) anwenden: Er wird mit

```
POKE 53265,91
```

eingeschaltet. In diesem Modus können Sie die Hintergrundfarbe eines einzelnen Zeichens bestimmen. Es bestehen insgesamt vier verschiedene Möglichkeiten für den Hintergrund.

Um diese Hintergrundfarben zu unterscheiden, werden die Bits 7 und 6 des Bildschirmcodes herangezogen. Es existieren also nur noch die Zeichen 0 bis 63, diese dafür aber mit vier verschiedenen Hintergründen.

Der Zusammenhang zwischen Bildschirmcode und Tastatur ist wie folgt (Tabelle 12.4).

In folgende Adressen können Sie die Hintergrundfarben POKEn (Tabelle 12.5).

BS-Code	Zeichen
0 bis 63	«normale» Zeichen
64 bis 127	⟨SHIFT⟩ + Zeichen
128 bis 191	⟨RVS ON⟩, Zeichen
192 bis 255	⟨RVS ON⟩, ⟨SHIFT⟩ + Zeichen

Tabelle 12.4 Hintergrundfarben

BS-Code	Farbadresse
0 bis 63	53281
64 bis 127	53282
128 bis 191	53283
192 bis 255	53284

Tabelle 12.5 Adressen für Hintergrundfarben

Um diesen Modus abzuschalten, verwenden Sie folgenden POKE:

```
POKE53265,27
```

Der Hintergrundfarbmodus ist im Hires-Modus sinnlos und kann daher dort nicht angewendet werden.

Sprites

Sprites sind kleine Grafikblocks, die unabhängig vom Hintergrund bewegt werden können. Sie sind 24 * 21 Punkte groß, und maximal acht von ihnen können gleichzeitig auf dem Bildschirm dargestellt werden. Ein Sprite wird als Bitmuster in einem Speicherbereich von 63 Bytes abgespeichert. Diese Sprites können dann mit Hilfe von X- und Y-Koordinaten auf dem Bildschirm bewegt werden. Da sie unabhängig vom Bildschirmhintergrund sind, kann dieser sowohl der normale Textbildschirm sein als auch eine Hires-Grafik. Ein weiterer Vorteil ist, daß sie den Hintergrund in keiner Weise beeinflussen, also nicht verändern oder löschen. Weitere Möglichkeiten sind: Vergrößerung, Kollisionsabfrage, Wechsel der Hintergrund-Priorität und Mehrfarbdarstellung.

Alle diese Funktionen werden vom VIC-Videochip durch einzelne Register gesteuert.

Zunächst muß ihm mitgeteilt werden, wo im Speicher sich das Bitmuster für einen bestimmten Sprite befindet. Dazu werden acht bisher freie Adressen am Ende des Bildschirm-RAMs verwendet: Die Zeiger auf die Startadresse eines Sprites liegen in den Speicherzellen 2040 bis 2047. Dieser Zeiger gibt jeweils den 64-Byte-Block an, um den es sich handelt. Wenn hier beispielsweise 13 steht, ist die Startadresse 13 * 64, also 832.

Allerdings sind einige Speicherbereiche für Spritemuster ungeeignet, beispielsweise die Zeropage, das Bildschirm-RAM und der BASIC-Speicher. Welche Adressen frei sind, erkennen Sie in der Tabelle 7.1

Wenn nun die Zeiger auf die entsprechenden Speicherbereiche gesetzt sind und die Bitmuster der Sprites dorthin gePOKEd wurden, erfolgen alle weiteren Steuerungen über Register von VIC.

Im einzelnen wären das:

```
Sprite einschalten: 53269
```

Diese Speicherzelle ist dafür verantwortlich, welcher Sprite gerade aktiv ist,

also auf dem Bildschirm erscheint. Jedem der acht Sprites ist ein Bit in dieser Speicherzelle zugeordnet, so daß ein Sprite mit folgendem Befehl eingeschaltet werden kann:

POKE 53269, PEEK (53269) OR 2 ∧ Spritenummer

Die Spritenummern reichen entsprechend den Bits von 0 bis 7. Um einen Sprite auszuschalten, verwenden Sie

POKE 53269, PEEK (53269) AND (255−2 ∧ Spritenummer)

Die X- und Y-Koordinaten für die verschiedenen Sprites stehen in den Adressen 53248 bis 53263 also (Tabelle 12.6).
Diese Koordinaten reichen in den Rahmen hinein, so daß sich für die echte Startposition (linkes oberes Bildschirmeck) die Koordinaten (X = 24; Y = 50) ergeben.

Da die Bildschirmbreite größer als 255 ist, muß für die X-Koordinaten noch ein 9. Bit bereitgestellt werden. So kann also für X ein Wert bis 511 angegeben werden, was in jedem Fall ausreicht.

Diese 9. Bits sind für alle Sprites in einem Register zusammengefaßt:

53264: 9. Bits der Sprites 7 6 5 4 3 2 1 0

Um Sprites zu vergrößern, gibt es zwei Register:

53271 für Vergrößerung in Y-Richtung,
53277 für Vergrößerung in X-Richtung.

Wenn ein Bit in einer oder beiden Speicherzellen gesetzt ist, wird der zugehörige Sprite in der jeweiligen Richtung auf die doppelte Größe gestreckt.

Die Register, in denen die Farben für die Sprites angegeben werden, liegen von 53287 bis 53294. Für jeden Sprite kann eine Farbe von 0 bis 15 angegeben werden (Tabelle 12.7).
Ein Sprite kann auch im Multicolormodus dargestellt werden. Dieser wird für jeden Sprite einzeln gewählt. Die zuständige Speicherzelle ist 53276. Wenn hier ein Bit gesetzt ist, wird der zugehörige Sprite als Multicolorgrafik dargestellt.

Die Farbadressen für die vier möglichen Bitkombinationen sind (Tabelle 12.8).
Weiterhin kann die sogenannte Sprite-Hintergrund-Priorität angegeben werden. Das bedeutet, es kann für jeden Sprite einzeln festgelegt werden, ob er

X-Koordinate von Sprite #0	53248
Y-Koordinate von Sprite #0	53249
X-Koordinate von Sprite #1	53250
Y-Koordinate von Sprite #1	53251
...	
X-Koordinate von Sprite #7	53262
Y-Koordinate von Sprite #7	53263

Tabelle 12.6 Spritekoordinaten

Farbe von Sprite #0	53287
Farbe von Sprite #1	53288
...	
Farbe von Sprite #7	53294

Tabelle 12.7 Farbadressen für Sprites

Bits	Adresse
00	53281
01	53285
10	Farbregister des jeweiligen Sprites
11	53286

Tabelle 12.8 Farbadressen für Multicolorsprites

vor oder hinter dem Hintergrund dargestellt wird, also vor oder hinter dem Text bzw. vor oder hinter den Punkten der Hires-Grafik. Das geschieht mit Adresse 53275. Ein gesetztes Bit bedeutet, der zugehörige Sprite wird hinter dem Hintergrund dargestellt.

Schließlich gibt es noch zwei Kollisionsdetektoren: Hier kann festgestellt werden, ob die Sprites untereinander oder mit dem Hintergrund zusammengestoßen sind.

Das Register 53278 ist für die Kollisionen untereinander zuständig: Die Bits

der beteiligten Sprites sind gesetzt und bleiben gesetzt, bis sie mit PEEK (53278) ausgelesen wurden. Da dieses System nicht hundertprozentig funktioniert, sollte nach einer Abfrage der Inhalt durch POKE 53278,0 gelöscht werden.

Nach genau demselben Prinzip arbeitet die Adresse 53 279. Sie ist für die Kollisionen mit dem Hintergrund verantwortlich.

Musik und Geräusche

Der SID 6581 hat drei voneinander unabhängige Stimmen. Für jede dieser Stimmen müssen verschiedene Parameter programmiert werden, die die Charakteristik, Dauer und Lautstärke des zu erzeugenden Tons beschreiben.

Zur Einstellung dieser Werte dienen wieder Register, diesmal die des SID. Sie liegen ab der Adresse 54 272.

Der erste Parameter ist die sogenannte Wellenform. Je nachdem, wie die Schwingungen, die vom menschlichen Ohr als Töne wahrgenommen werden, gebaut sind, klingt der Ton.

Beim SID können Sie zwischen vier verschiedenen Wellenformen wählen:

Sägezahn (17)
Dreieck (33)
Rechteck (65)
Rauschen (129)

Eine dieser Wellenformen muß nun für jede Stimme angegeben werden. Dazu POKEn Sie die entsprechende Zahl (oben angegeben) in das Wellenform-Register der jeweiligen Stimme (Tabelle 12.9).

Stimme 1	54276
Stimme 2	54283
Stimme 3	54290

Tabelle 12.9 Wellenformen für Stimmen 1 bis 3

Bei der Programmierung eines Tons sollte diese Aufgabe als letzte erfolgen, da die Wellenform-Register das sogenannte «Key-Bit» enthalten (nämlich Bit

#0), das den Ton letztlich einschaltet (bzw. ausschaltet. Deshalb können Sie auch durch Löschen von Bit#0 im jeweiligen Wellenform-Register den gesamten Ton beenden).

Genausogut können Sie natürlich zuerst die Wellenform angeben und dann am Schluß durch

POKE (Adresse), PEEK (Adresse) OR 1

den Ton starten.

Wenn Sie die Wellenform «Rechteck» wählen, muß zusätzlich noch das «Tastverhältnis» (bzw. die «Pulsbreite») angegeben werden. Dies ist eine Zahl, die die Breite des Rechtecks beschreibt. Sie wird als 12-Bit-Wert dargestellt, kann also von 0 bis 4095 reichen. Dieser Wert wird für jede Stimme in zwei aufeinanderfolgenden Registern nach dem High-Byte/Low-Byte-Prinzip abgelegt (Tabelle 12.10).

Stimme 1	54274 und 54275
Stimme 2	54281 und 54282
Stimme 3	54288 und 54289

Tabelle 12.10 Pulsbreiten für Stimmen 1 bis 3

Der Verlauf des Tons wird beim SID nach dem ADSR-Prinzip beschrieben. Das bedeutet, durch die vier Parameter Attack, Decay, Sustain und Release können Änderungen der Intensität programmiert werden. Jeder Parameter wird als 4-Bit-Wert (also 0 bis 15) dargestellt. Dabei können diese Parameter folgendermaßen umschrieben werden: Attack (Anschlag) ist die Intensität, mit der ein Ton anklingt. Decay (abschwellen) ist das Maß, die Geschwindigkeit, mit der ein Ton auf seine Grundlautstärke zurückfällt. Sustain (aushalten) ist diese Grundlautstärke. Release (ausklingen) ist die Geschwindigkeit, mit der der Ton verklingt.

Da jeweils vier Bit für eine dieser Einstellungen zuständig sind, können zwei Parameter (als Nibble) in einem Byte untergebracht werden (Tabelle 12.11). Die wichtigste Einstellung schließlich ist die Frequenz. Sie kann von 0 bis 65 535 gehen, benötigt also zu ihrer Darstellung 16 Bit. So wird auch sie wieder in High Byte/Low Byte zerlegt (Tabelle 12.12).

Stimme	Attack/Decay	Sustain/Release
1	54277	54278
2	54284	54285
3	54292	54293

Tabelle 12.11 Adressen für ADSR-Hüllkurven

Stimme	High Byte	Low Byte
1	54272	54273
2	54279	54280
3	54286	54287

Tabelle 12.12 Frequenzadressen für Stimmen 1 bis 3

Damit der Ton nun auch erklingt, muß nur noch die Lautstärke angegeben werden. Sie ist für alle Stimmen einheitlich und liegt in der Adresse 54296.

Vergessen Sie nicht, bei Ihren Angaben die Wellenform als letztes zu POKEn!

Input/Output

1. Tastatur

Gedrückte Tasten werden in einem sogenannten «Tastaturpuffer» zwischengespeichert, unabhängig davon, wann sie ausgelesen werden. Im Normalfall wird das sofort sein (also Tippen eines Programms oder einer Eingabe). Wenn während eines Programmlaufs oder während einer Warteschleife Tasten gedrückt werden, warten diese im Tastaturpuffer so lange, bis sie aufgrund eines INPUT- oder GET-Befehls abgefragt werden oder bis das Programm zu Ende ist und der Inhalt des Tastaturpuffers auf den Bildschirm geschrieben werden kann.

Die maximale Größe des Tastaturpuffers steht in der Adresse 649. Der normale Wert ist 10. Er sollte nicht erhöht werden. Man kann aber die Anzahl der erlaubten gedrückten Tasten so auf eine Zahl, die kleiner als 10 ist, beschränken. Wenn der Inhalt dieser Adresse 0 ist, ist die Tastatur blockiert.

In der Zelle 198 dagegen steht die Anzahl der Tasten, die sich tatsächlich im Tastaturpuffer befinden. Diese läßt sich mit PEEK (198) abfragen, um beispielsweise die Anzahl der bisher gedrückten Tasten zu prüfen. Sie kann auch auf 0 gesetzt werden, um bisher gedrückte Tasten zu unterdrücken. Oder sie kann im Zusammenhang mit der simulierten Tastatur verwendet werden.

Dabei werden in den Tastaturpuffer (631 bis 640) die Codes der Tasten gePOKEd, die ausgeführt werden sollen. Dann wird der Adresse 198 die Anzahl dieser Tasten übergeben. Damit wurde der Tastaturpuffer künstlich gefüllt und wird nun bei der nächsten Abfrage (meist durch Rückkehr in den Direktmodus) die vorprogrammierten Tasten ausgeben. In den Zellen 197 bzw. 204 steht der Tastaturcode der Taste, die gerade gedrückt wird. Die Werte, die sich in dieser Zelle befinden, sind auf Bild 10.1 zu sehen.

Ob eine der Tasten ⟨SHIFT⟩, ⟨C=⟩ oder ⟨CTRL⟩ gedrückt wurde, läßt sich mit PEEK (653) feststellen. Dabei erhalten Sie folgende Werte:

1 für ⟨SHIFT⟩
2 für ⟨C=⟩
4 für ⟨CTRL⟩

Wenn mehrere dieser Tasten gehalten werden, befindet sich in der genannten Speicherzelle die Summe der oben aufgelisteten Werte.

2. Joysticks

Die Joysticks werden an die beiden Control-Ports an der rechten Geräteseite angeschlossen. Sie können mit PEEK (56321) und PEEK (56320) abgefragt werden.

Vergleichen Sie die Bedeutung einzelner Bits im PEEK&POKE-Anhang.

3. Paddles

Zum Auslesen von Paddles benötigt man A/D-Wandler. Zwei solche Wandler sind bereits im SID eingebaut, so daß diese zum Auslesen der Paddles verwendet werden.

Den aktuellen Wert der Paddles kann man also durch

PEEK (54297) für Paddle 1
PEEK (54298) für Paddle 2

feststellen. Das gilt zunächst für die Paddles, die an Port 1 angeschlossen sind.

Ob die Feuerknöpfe der Paddles gedrückt werden, erkennt man an den Bits 2 und 3 in der Adresse 56321.

Sie können folgendermaßen abgefragt werden:

```
IF NOT PEEK(56321) AND 4 THEN 1. Paddle
IF NOT PEEK(56321) AND 8 THEN 2. Paddle
```

Wenn Paddles am 2. Port betrieben werden sollen, erfordert dies eine etwas andere Programmierung: Da über den gleichen Baustein auch die Tastaturabfrage stattfindet, müssen zuerst die Interrupts abgeschaltet werden. Dazu dient POKE 56334,0 (nur innerhalb eines Programms verwenden und danach wieder einschalten!).

Dann muß dem Interfacebaustein (CIA) über einen entsprechenden POKE mitgeteilt werden, welche Paddlesignale an den SID weitergegeben werden sollen:

```
POKE 56320,64 für Port 1
POKE 56320,128 für Port 2
```

Die Feuerknöpfe des zweiten Paddlesatzes können in den Bits 2 und 3 der Adresse 56320 abgefragt werden:

```
IF NOT PEEK(56320) AND 4 THEN 1. Paddle
IF NOT PEEK(56320) AND 8 THEN 2. Paddle
```

Noch mal: Vergessen Sie nicht, nach der Paddleabfrage die Interrupts und somit die Tastatur wieder zu aktivieren!

Peripheriegeräte

Die Floppy VC 1541

Die Floppy zum Commodore 64 verwendet 5¼-Zoll-Disketten. Die Daten einer Diskette sind in Tabelle 12.13 dargestellt.

Vor dem ersten Verwenden muß eine frisch gekaufte Diskette «formatiert» werden. Dabei werden die Spuren und Sektoren magnetisch vorgezeichnet. Die Spuren auf einer Diskette sind konzentrisch um das Loch in der Mitte angeordnet.

Die Anzahl der Blocks je Spur ist unterschiedlich (Tabelle 12.14).

Die Floppy ist ein «intelligentes» Peripheriegerät: Sie hat einen eigenen Prozessor und ihr eigenes Betriebssystem (DOS) in ROMs eingebaut.

Spuren (Tracks): 35
Sektoren (Blocks): 683
Bytes je Block: 256
Directory: Spur 18
freie Sektoren: 664
Speicherkapazität: 170 KByte

Tabelle 12.13 Daten einer Diskette

Spur:	Sektoren:
1–17	21
18–24	19
25–30	18
31–35	17

Tabelle 12.14 Sektoren je Spur

Um Floppy-Funktionen auszuführen, muß deshalb ein entsprechender Befehl an die Diskettenstation geschickt werden.

Zum LOADen und SAVEn (also zum Laden und Abspeichern) von Programmen muß eine Gerätenummer angegeben werden:

```
LOAD"Programmname",8
SAVE"Programmname",8
```

Normalerweise hat die Floppy die Gerätenummer 8. Wenn gleichzeitig mehrere Floppys verwendet werden sollen, kann diese Gerätenummer geändert werden.

Um das Inhaltsverzeichnis einer Diskette (Directory) zu lesen, muß dieses im normalen BASIC als Programm geladen werden:

```
LOAD"$",8
```

Das hat allerdings zur Folge, daß das Programm, das sich gerade im Speicher befindet, gelöscht wird.

Um diesen Nachteil zu umgehen, sollte man das Programm DOS 5.1

verwenden, das sich auf der mitgelieferten «TEST/DEMO»-Diskette befindet. Es wird mit LOAD"DOS 5.1",8,1 geladen. Aufgerufen wird es mit SYS 52224:NEW.

Nun können Sie für die Anzeige des Directory den Befehl

```
⟩$
```

verwenden.

Zur Abfrage des Fehlerkanals verwenden Sie in BASIC normalerweise:

```
10 OPEN 1,8,15 : INPUT#1, A,A$,B,C : PRINT A;A$;B;C : CLOSE 1
```

Diese ganze Prozedur läßt sich beim DOS 5.1 durch

```
⟩
```

ersetzen.

Zur Übermittlung eines Befehls an die Floppy benötigen Sie in BASIC folgende Syntax:

```
OPEN1,8,15: PRINT#1,Befehl: CLOSE1
```

Das DOS 5.1 stellt Ihnen folgende Kurzform zur Verfügung:

```
⟩Befehl
```

Anstelle von "⟩" kann bei DOS-5.1-Befehlen auch "@" verwendet werden. Eine Gegenüberstellung weiterer Befehle:

DOS 5.1	BASIC V.2
/Programm	LOAD "Programm",8
↑ Programm	LOAD "Programm",8 RUN
%Programm	LOAD "Programm",8,1
←Programm	SAVE "Programm",8

Außer diesen Befehlen gibt es Befehle, die direkt an das DOS in der Floppy übermittelt werden. Dabei ist es im Prinzip egal, ob das mit Hilfe des DOS 5.1 oder über Standard-BASIC geschieht. Es genügt generell, wenn der erste Buchstabe eines solchen Befehls gesendet wird. Also zum Beispiel «I» anstelle von «INITIALIZE». Die wichtigsten Befehle an die Floppy sind:

INITIALIZE

Dieser Befehl weist das DOS an, die BAM (Block Availability Map) in den Arbeitsspeicher der Floppy einzulesen.

NEW

Dieser NEW-Befehl an die Floppy (nicht zu verwechseln mit dem BASIC-Befehl NEW) formatiert eine Diskette neu. Dazu müssen der Diskettenname und die Identity-Nummer angegeben werden:

N:Diskname,Id

COPY

Dieser Befehl kopiert ein File innerhalb einer Diskette. Das kopierte Programm muß einen anderen Namen als den des Originalprogramms erhalten.

C:Name der Kopie = Name des Originals

RENAME

Mit diesem Befehl kann ein File auf Diskette umbenannt werden:

R:Neuer Name = Alter Name

SCRATCH

Damit werden Files auf einer Diskette gelöscht.

S:Programmname

VALIDATE

Dieser Befehl vergleicht die belegten Blocks in der BAM mit der aktuellen Belegung und korrigiert gegebenenfalls Falscheinträge. Die Jokersymbole:

* und ? können als Jokersymbole verwendet werden.

? ersetzt einen unbekannten Buchstaben in einem Programmnamen. So steht H?LLO genauso für HALLO wie für HELLO.

* beendet den Namen vorzeitig. EINK* kann für EINKAUF, EINKUENFTE oder EINKOMMENSTEUERERKLAERUNG stehen.

Wird bei einem LOAD-Befehl ein Joker verwendet, wird grundsätzlich das erste Programm geladen, auf das das Muster paßt.

Bei den Floppy-Befehlen werden alle Programme betroffen, die passen. Ein «S:PRO*» löscht also alle Programme, die mit «PRO» anfangen. Bei SAVE darf kein Joker verwendet werden. Das erzeugt einen SYNTAX ERROR, ebenso bei Diskettennamen und ähnlichem.

Druckerhandhabung

Um ein Listing auf einem Drucker auszugeben, dient folgende Syntax:

```
OPEN 4,4: CMD 4: LIST
PRINT# 4: CLOSE 4
```

Wenn ein Fremddrucker verwendet wird, der dem Carriage Return-Code (CHR$(13)) keinen automatischen Line Feed (CHR$(10)) beifügt, muß eine Kanalnummer größer als 127 verwendet werden.

Zur Ausgabe von Daten auf den Drucker dient der PRINT#-Befehl:

```
OPEN4,4: PRINT# 4,"DIES IST EIN TEST"
```

Die Trennzeichen ";" und "," wirken wie beim normalen PRINT-Befehl. Bei jedem Drucker können durch Steuercodes oder Sekundäradressen bestimmte Betriebsmodi gewählt werden. Vergleichen Sie dazu aber bitte Ihr Druckerhandbuch.

Listings

In diesem Anhang finden Sie die Programmlistings, die wir Ihnen im Lauf des Textes immer wieder versprochen haben. Vorher aber noch einige Anmerkungen: Erklärtes Ziel dieser Listings ist es, Ihnen Denkanstöße zu vermitteln. Das gilt speziell für die verschiedenen Utilities. Wir haben Ihnen zwar lauffähige Programme abgedruckt, verstehen diese aber nicht als fertige Lösungen. Jedes der abgedruckten Programme sollte komfortabler sein. Das zu programmieren, liegt an Ihnen.

Die abgedruckten Programme sind auch nicht gegen Fehlbedienung geschützt. Wenn Sie eine sinnlose oder fehlerhafte Eingabe machen, wird

früher oder später ein ERROR oder (bei POKEs!!!) ein Programmabsturz eintreten.

Dennoch sollten Sie diese Programme abtippen und als Hilfe benutzen. Speziell das Grafikutility und die Zeichen- bzw. Spriteeditoren werden Ihnen wohl gute Dienste erweisen können.

Wir haben uns entschieden, diese Programme nicht so ausführlich zu dokumentieren, wie Sie es von unseren beiden Spielen gewohnt sind. Das hat zwei Gründe: Erstens würde es sich bei den meisten kurzen Programmen sowieso nicht lohnen, und zweitens sollen Sie durch Experimente und Beobachtungen selbst aus diesen Programmen lernen. Die Beschreibungen zu den einzelnen Programmen wollen nur kurz aufzeigen, welche Ideen hinter den einzelnen Beispielen stecken.

Hugo

Dieses vierzeilige Programm zeigt, mit welch einfachen Mitteln schon ein bißchen Animation möglich ist. In den Zeilen 20 und 30 befinden sich genau dieselben Steuerzeichen (aus der ⟨HOME⟩-Position, zweimal ⟨CRSR DOWN⟩, zweimal ⟨CRSR RIGHT⟩, Kopf zeichnen ⟨CRSR DOWN⟩, zweimal ⟨CRSR LEFT⟩, Arm, ⟨RVS ON⟩ für den Rumpf, Leerstelle, ⟨RVS OFF⟩, anderer Arm, ⟨CRSR DOWN⟩, dreimal ⟨CRSR LEFT⟩, Bein, Bauch, Bein. Die beiden Bewegungsphasen müssen sich aber in den Zeichen für Arme und Beine unterscheiden. Der Trick dabei ist, beim PRINTen immer in der HOME-Position anzufangen, damit die Zeichen auch richtig überdruckt werden. Mit GOTO 20 erreicht man eine Endlosschleife und somit einen ständigen Wechsel der beiden Phasen. Zeile 10 löscht einmal zum Anfang den Bildschirm, um störende Texte zu entfernen.

```
HUGO


10 PRINT"[illegible]"
20 PRINT"[illegible]"
30 PRINT"[illegible]"
40 GOTO20

READY.
```

Zeicheneditor

Dieses Programm braucht am Anfang ein wenig Zeit, um den Zeichensatz von ROM ins RAM zu kopieren. Dann wird ein Feld aus 8 * 8 Punkten aufgebaut, in dem man sich mit dem Cursor frei bewegen kann. Ein Druck auf die Taste "." setzt einen Punkt, ⟨SPACE⟩(also die Leertaste) löscht ihn wieder. In der Matrix wird auf diese Weise ein Zeichen aufgebaut. Nach Eingabe von ⟨RETURN⟩ fragt der Computer, «welches Zeichen» so umdefiniert werden soll. Drücken Sie einfach die entsprechende Taste. Das Zeichen wird umdefiniert, und der Computer druckt unter der Bezeichnung DATA die Dezimalwerte aus, die für das entwickelte Zeichen in den Speicher gePOKEd werden müssen. Die Punkte sind in einem Datenfeld von 8 * 8 Positionen gespeichert (A%(X,Y)). Zur Errechnung der POKE-Werte dient ab Zeile 290 eine einfache Umrechnungsroutine von Dual in Dezimal. Wenn Sie ⟨CLR HOME⟩ drücken, können Sie jederzeit neu anfangen, wobei Sie allerdings auch die Matrix löschen.

```
ZEICHEN DEF.


10 PRINT"■BITTE WARTEN"
20 POKE56334,0:POKE1,51
30 FORX=0TO2047:POKE14336+X,PEEK(53248+X):NEXTX
40 POKE1,55:POKE56334,1:POKE53272,30
50 PRINT"■■ ZEICHENSATZ-EDITOR"
60 PRINT
70 FORX=1TO8
80 PRINT"   ........"
90 NEXT
100 X=1:Y=1
110 GETA$
120 IFA$="■"THENY=Y-1:IFY=0THENY=8
130 IFA$="■"THENY=Y+1:IFY=9THENY=1
140 IFA$="■"THENX=X-1:IFX=0THENX=8
150 IFA$="■"THENCLR:GOTO50
160 IFA$="■"THENX=X+1:IFX=9THENX=1
170 IFA$="."THENA%(X,Y)=1
180 IFA$=" "THENA%(X,Y)=0
190 IFA$=CHR$(13)THEN260
200 IFA%(X,Y)=1THENA=160
210 IFA%(X,Y)=0THENA=46
220 POKE1065+X+Y*40,102
230 IFA$=""THENFORI=1TO100:NEXTI
```

```
240 POKE1065+X+Y*40,A
250 GOTO110
260 PRINT"■■■■■■■■■■■■■■WELCHES ZEICHEN?";:POKE204
    ,0:WAIT198,1:POKE207,0:POKE204,1
270 GETA$:PRINTA$:PRINT:PRINT"DATA";
280 A=PEEK(1480)
290 FORY=1TO8
300 FORX=1TO8
310 IFA%(X,Y)=1THENW=W+2↑(8-X)
320 NEXTX
330 POKE14336+A*8+Y-1,W:PRINTW",";:W=0
340 NEXTY
350 GOTO100

READY.
```

Sinus

Dieses Programm erklärt sich weitgehend aus dem Hires-Kapitel. Um genauere oder weniger genaue Plots zu erhalten, können Sie den STEP-Wert in Zeile 40 beliebig abändern. Anmerkung: Im Gegensatz zu einem mathematischen Koordinatensystem ist beim Computer der Punkt (0,0) oben links. Dadurch ist die dargestellte Kurve eigentlich «spiegelverkehrt».

```
SINUS


10 POKE53265,59:POKE53272,24
20 FORX=8192TO16192:POKEX,0:NEXT
30 FORX=1024TO2024:POKEX,99:NEXTX
40 FORX=0TO319STEP1
50 Y=SIN(X/30):Y=100-(Y*80)
60 BY=8192+(YAND7)+(8*INT(X/8))+(320*(INT(Y/8)))
70 POKEBY,PEEK(BY)OR2↑(7-(XAND7))
80 NEXTX

READY.
```

Dual 〉 Dezimal

Dieses Programm fragt Sie nach einer Dualzahl. Wenn Sie andere Zeichen als 0 oder 1 eingeben, wird das in Zeile 30 erkannt, und das Programm springt erneut zur Abfrage. Nach der Umrechung können Sie eine beliebige Taste drücken. Mit ⟨E⟩ können Sie abbrechen.

```
DUAL > DEZIMAL


10 INPUT"DUALZAHL";Z$
20 FORX=1TOLEN(Z$)
30 IFMID$(Z$,X,1)<>"1"ANDMID$(Z$,X,1)<>"0"THENPRINT
   "FEHLER IN DER ZAHL!":GOTO10
40 NEXTX
50 FORX=1TOLEN(Z$)
60 IFMID$(Z$,X,1)="1"THENZ=Z+2↑(LEN(Z$)-X)
70 NEXTX
80 PRINTZ$"="Z:PRINT:PRINT"<E>=ENDE":PRINT
90 GETA$:IFA$=""THEN90
100 IFA$="E"THENEND
110 RUN

READY.
```

Dezimal 〉 Dual

Sie können eine beliebige ganzzahlige Dezimalzahl eingeben. In Zeile 20 wird die höchste Zweierpotenz gesucht, von Zeile 40 bis 70 liegt die eigentliche Umrechnungsroutine. Drücken Sie irgendeine Taste, um weiterzumachen, 〈E〉 für Ende.

```
DEZIMAL > DUAL


10 INPUT"DEZIMALZAHL";Z
20 D=D+1:IFZ/(2↑D)>=1THEN20
30 D=D-1:A=Z
40 FORX=DTO0STEP-1
50 IFA/(2↑X)>=1THENZ$=Z$+"1":A=A-2↑X:GOTO70
60 Z$=Z$+"0"
70 NEXTX
80 PRINTZ"="Z$:PRINT
90 PRINT"<E>=ENDE":PRINT
100 GETA$:IFA$=""THEN100
110 IFA$="E"THENEND
120 RUN

READY.
```

Grafikutility

Dieses Programm haben wir für all diejenigen geschrieben, die noch keine BASIC-Erweiterung haben oder die sich selbst mit Grafikprogrammierung beschäftigen wollen. Dieses «Utility» besteht aus vier Unterprogrammen, die Sie mit GOSUB aufrufen. Laden Sie die abgedruckten Programmzeilen, und schreiben Sie dann Ihr Grafikprogramm in den Zeilen 0 bis 59998. Folgende Funktionen stehen Ihnen zur Verfügung.

GOSUB 60000

Dieses Unterprogramm schaltet die Grafik ein und löscht die Bitmap. Sie können diesem Programm in den Variablen ZF (Zeichenfarbe) und HF (Hintergrundfarbe) Zahlen zwischen 0 und 15 übergeben, die dann für die entsprechenden Farben verwendet werden.

GOSUB 61000

Diesem Unterprogramm übergeben Sie die X- und Y-Koordinate eines Punktes, der gesetzt werden soll.

GOSUB 62000

Mit diesem Unterprogramm können Sie eine Linie zwischen zwei Punkten ziehen. Sie übergeben ihm die Anfangs- und Endpunkte (X1,Y1) und (X2,Y2). Die Genauigkeit (die bei steilen Linien sehr viel höher sein muß als bei flachen) können Sie in den Zeilen 62030 und 62040 angeben. Anstelle des Wertes 320 können Sie jeden Wert zwischen 1 und 320 verwenden.

GOSUB 63000

Diesem Programm übergeben Sie den Mittelpunkt eines Kreises (X,Y) und den Radius R. Das Programm zieht dann einen Kreis um diesen Punkt, wobei an den äußersten Stellen Löcher auftreten, weil eine höhere Genauigkeit außen bei den anderen Punkten zu viel Zeit kosten würde.

Beispiele, die Sie ausprobieren sollten:

```
10 HF=6: ZF=1: GOSUB 60000
20 X=159: Y=99: FOR R= 90 TO 10 STEP -10: GOSUB 63000: NEXT
```

```
10 HF=2: ZF=0: GOSUB 60000
20 X1=159: Y1= 99: Y2=180: FOR X2=10 TO 310 STEP 5: GOSUB
   62000: NEXT
```

GRAFIK UTILITY

```
59999 END
60000 REM GRAFIKINIT (LOESCHEN, ZF= ZEICHENFARBE, HF=HINTERGRUNDFARBE)
60010 POKE53265,59:POKE53272,24
60020 FORI=1024TO2023:POKEI,ZF*16+HF:NEXT
60030 FORI=8192TO16191:POKEI,0:NEXT
60040 RETURN
61000 REM POINT X,Y
61010 I=8192+(YAND7)+(8*INT(X/8))+320*(INT(Y/8)):POKEI,PEEK(I)OR2↑(7-(XAND7))
61020 RETURN
62000 REM LINE (X1,Y1) TO (X2,Y2)
62010 IFX1=X2THENX2=X2+1
62020 Y=Y1
62030 FORJ=X1TOX2STEP(SGN(X2-X1)/ABS(320/(X2-X1)))
62040 X=J:Y=Y+(ABS((Y2-Y1)/(X2-X1))*SGN(Y2-Y1))/ABS(320/(X2-X1))
62050 I=8192+(YAND7)+(8*INT(X/8))+320*(INT(Y/8)):POKEI,PEEK(I)OR2↑(7-(XAND7))
62060 NEXTJ
62070 RETURN
63000 REM CIRCLE X,Y,R
63010 FORJ=X-RTOX+R:X1=J-(X-R)
63020 Y1=Y+SQR(2*R*X1-X1*X1)
63030 I=8192+(Y1AND7)+(8*INT(J/8))+320*(INT(Y1/8)):POKEI,PEEK(I)OR2↑(7-(JAND7))
63040 Y1=Y-SQR(2*R*X1-X1*X1)
63050 I=8192+(Y1AND7)+(8*INT(J/8))+320*(INT(Y1/8)):POKEI,PEEK(I)OR2↑(7-(JAND7))
63060 NEXTJ
63070 RETURN

READY.
```

Vermeiden Sie folgende Variablennamen, wenn Sie eigene Programme schreiben, die auf diese Unterprogramme zurückgreifen:

X, Y, X1, Y1, X2, Y2, R, ZF, HF, I, J

Soundmonitor

Dieses Programm bietet Ihnen die Möglichkeit, Töne auszuprobieren. Beachten Sie beim Eintippen die Steuerzeichen ab Zeile 160. Diese werden von den Funktionstasten erzeugt. Halten Sie sich bei Ihren Eingaben an die Werte in Klammern! Bei der Wellenform Rechteck, die Sie durch ⟨f5⟩ erreichen, müssen Sie zusätzlich noch eine Pulsweite eingeben.

```
SOUND MONITOR


10 PRINT"■■ MUSIC - MONITOR "
20 PRINT"■ATTACK (0-15)";:INPUTA
30 PRINT"■DECAY  (0-15)";:INPUTD
40 PRINT"■SUSTAIN(0-15)";:INPUTS
50 PRINT"■RELEASE(0-15)";:INPUTR
60 PRINT"■FREQUENZ (0-65535)";:INPUTF
70 POKE54296,15
80 POKE54277,A*16+D:POKE54278,S*16+R
90 POKE54273,F/256:POKE54272,F-INT(F/256)*256
100 PRINT"■WELLENFORM     ■F1■ SAEGEZAHN"
110 PRINT"                ■F3■ DREIECK"
120 PRINT"                ■F5■ RECHTECK"
130 PRINT"                ■F7■ RAUSCHEN"
140 GETA$:IFA$=""THEN140
150 POKE54276,0
160 IFA$="■"THENPOKE54276,17
170 IFA$="■"THENPOKE54276,33
180 IFA$="■"THENINPUT"■PULSWEITE(0-4095)";P
190 IFA$="■"THENPOKE54275,P/256:POKE54274,P-INT
    (P/256)*256:POKE54276,65
200 IFA$="■"THENPOKE54276,129
210 PRINT"■":GOTO20

READY.
```

Spriteeditor

Er ist eine Weiterentwicklung des Zeicheneditors. Die Eingabe ist dieselbe. Die Angabe von DATA-Werten ist aber diesmal nur auf direkten Wunsch möglich; danach ist der Sprite in der Matrix gelöscht. Die DATAs müssen Sie wahrscheinlich abschreiben, um sie in Ihren eigenen Programmen verwenden zu können. Im Datenfeld A% (X,Y) werden die Bits abgespeichert, in W% (X,Y) die entsprechenden Dezimalwerte.

```
SPRITES DEF.

10 DIMA%(24,21),W%(3,21)
20 PRINT"▮";
30 FORX=1TO21
40 PRINT"   ........................"
50 NEXT
60 X=1:Y=1
70 GETA$
80 IFA$="▮"THENY=Y-1:IFY=0THENY=21
90 IFA$="▮"THENY=Y+1:IFY=22THENY=1
100 IFA$="▮"THENX=X+1:IFX=25THENX=1
110 IFA$="▮"THENX=X-1:IFX=0THENX=24
120 IFA$="."THENA%(X,Y)=1
130 IFA$=" "THENA%(X,Y)=0
140 IFA$="▮"THENRUN
150 IFA$=CHR$(13)THEN220
160 IFA%(X,Y)=1THENA=160
170 IFA%(X,Y)=0THENA=46
180 POKE985+X+Y*40,102
190 IFA$=""THENFORI=1TO100:NEXT
200 POKE985+X+Y*40,A
210 GOTO70
220 PRINT"▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ADRESSE";
    :INPUTA
230 POKE2040,A:POKE53249,150:POKE53264,1:POKE53269,1
240 FORB=0TO20
250 FORC=0TO2:W%(C,B)=0
260 FORI=1TO8
270 IFA%(C*8+I,B+1)=1THENW%(C,B)=W%(C,B)+2↑(8-I)
280 NEXTI:POKEA*64+B*3+C,W%(C,B):NEXTC,B
290 PRINT"▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮DATA-WERTE (J/N)?";
300 POKE204,0
```

```
310 GETA$:IFA$=""THEN310
320 POKE207,0:POKE204,1:IFA$="N"THEN60
330 IFA$<>"J"THEN300
340 PRINT"■DATA";
350 FORI=0TO62:PRINTPEEK(A*64+I)"■,";
360 NEXTI:PRINT
370 PRINT"■■NAECHSTER SPRITE (J/N)?"
380 GETA$:IFA$="J"THENRUN
390 IFA$="N"THENEND
400 GOTO380

READY.
```

Kleines Fachwortlexikon

Viele der im Text enthaltenen Fachausdrücke sind bei ihrem ersten Auftreten in Kursivschrift gedruckt. Hier werden sie kurz erklärt. Zusätzlich wurden in diese Liste noch andere häufig vorkommende Begriffe aufgenommen. So können Sie dieses kleine Fachwortlexikon auch als Nachschlagewerk benutzen.

A/D-Wandler. Ein Analog-Digital-Wandler ist ein Baustein oder eine Schaltung, die ein analoges Eingangssignal (z. B. Stromstärke) in ein digitales Ausgangssignal (z. B. 1-Byte-Wert) umwandeln kann.

Absolutes Laden. Die ersten zwei Bytes eines Programms auf Diskette sind das Low Byte und das High Byte der Startadresse des Programms. Wenn absolut geladen wird (das heißt LOAD"...",8,1), dann wird das Programm ab dieser Startadresse in den Speicher geladen. Ansonsten (LOAD"...",8) kommt es an den Anfang des BASIC-Speichers (normalerweise 2048).

Adreßbereich. Der Umfang an Speicherplatz, den ein Prozessor adressieren kann. Er hängt ab von der Anzahl der Leitungen im Adreßbus. Beim 6510 sind das 16 Leitungen, also ist der Adreßbereich 2^{16} (65 536 = 64 K).

Adressierung. Um ein Byte in einem Speicherbaustein lesen oder schreiben zu können, muß der Prozessor dieses Byte adressieren; das heißt, er legt die Adresse als 16-Bit-Zahl auf den Adreßbus und kann dann auf den Datenbus das entsprechende Byte schreiben bzw. es von ihm lesen.

ADSR-Hüllkurve, engl. Attack/Decay/Sustain/Release. Nach diesem Konzept arbeitet der SID. Der Verlauf eines Tons wird in vier einzelne charakteristische Merkmale zerlegt. Attack ist die Stärke, mit der der Ton angeschlagen wird. Decay ist das Maß, in dem er auf die Grundlautstärke abfällt. Sustain ist die Länge, in der er auf dieser Lautstärke gehalten wird, Release ist das Maß, in dem er ausklingt.

Alphanumerische Variable. Ein String, z. B. A$, ist eine Variable, die sowohl alphabetische (also Buchstaben) als auch

numerische Informationen (also Zahlen) aufnehmen kann.

ASCII-Code, engl. American Standard Code for Information Interchange. Ein standardisierter Code, in dem den Buchstaben, Zahlen und Steuerzeichen eine Zahl zwischen 0 und 255 (also 1 Byte) zugeordnet wird. Eine Tabelle befindet sich im Commodore-Handbuch auf Seite 135.

Assembler. 1. Die Maschinensprache des 6510. Die Maschinenbefehle lassen sich als symbolische «Opcodes» darstellen. 2. Hilfsprogramm zur Eingabe eines Maschinenspracheprogramms.

BAM, engl. Block Availability Map – ein Verzeichnis der belegten Blocks einer Diskette. Es befindet sich auf jeder Diskette in Spur 18 Block 0.

BASIC-Anfang. Die Adresse im RAM, ab der das BASIC-Programm abgelegt und abgearbeitet wird. Sie kann verschoben werden. Siehe dazu PEEK&POKE-Anhang bei den Adressen 43/44.

BASIC-Erweiterung. Ein (Maschinen)-Programm auf Modul oder Diskette, das weitere, neue BASIC-Befehle bereitstellt, beispielsweise für die Grafik- oder Diskettenprogrammierung.

BASIC-Interpreter. Wie das Betriebssystem (Kernal) ein Maschinenprogramm, das ständig läuft. Es hat die Aufgabe, BASIC-Befehle zu erkennen und auszuführen.

Betriebssystem, auch Kernal. Das Maschinenprogramm, das ständig im Computer läuft und für Funktionen wie Bildschirmausgabe, Tastaturabfrage usw. verantwortlich ist.

Binärzahlen. Zahlensystem, basierend auf der Zahl 2. Jede neue Stelle ist eine Zweierpotenz. Die Zahl wird üblicherweise mit 0 und 1 dargestellt, also

```
0000   (0)
0001   (1)
0010   (2)
0011   (3)
0100   (4)
```

usw.

Bit, engl. binary digit. Die kleinste Einheit der Informationsspeicherung. Es kann zwei Zustände haben: an oder aus, wahr oder falsch, 1 oder 0.

Bitmapping. Um eine hochauflösende Grafik im Speicher verwalten zu können, wird jedem Punkt auf dem Schirm ein Bit im Speicher zugeordnet. Bei einer Auflösung von 320 * 200 Punkten benötigt man 8K-Speicher für die Bitmap.

Block. Ein Sektor auf einer Diskette. Er kann 256 Bytes speichern. Eine VC-1541-formatierte Diskette hat 683 Blocks.

Boolesche Algebra. Binäralgebra. Mit verschiedenen Operationen können einzelne Bits verknüpft werden; benannt nach dem englischen Mathematiker Boole.

Boolesche Operation. Verknüpfung in der Booleschen Algebra – in BASIC vor allem AND, OR und NOT.

Byte. Eine 8-Bit-Zahl. Es kann einen (ganzzahligen) Wert zwischen 0 (binär 00000000) und 255 (binär 11111111) annehmen. Eine Speicherzelle im Commodore 64 faßt genau 1 Byte.

Charakter-ROM. Der Speicherbaustein, in dem das Aussehen der Zeichen beim Commodore 64 abgespeichert ist.

Code ist in der Datenverarbeitung ein genereller Ausdruck dafür, wie Daten übertragen bzw. dargestellt werden. Das Wort hat aber auch einige spezielle Bedeutungen: 1. Von einem Code spricht man auch, wenn ein einzelner Befehl (insbesondere bei Maschinensprache, vgl. Opcode) gemeint ist. 2. Daher wird auch das Ergebnis einer Übersetzung durch einen Compiler gern Code genannt. 3. Code ist auch der Zahlenwert, der einem Buchstaben, einer Taste oder etwas ähnlichem zugeordnet wird, z. B. der Bildschirmcode, Tastaturcode usw.

Compiler. Ein Compiler ist ein Übersetzungsprogramm. Im Gegensatz zu einem Interpreter wird das Programm aber nicht jedesmal Stück für Stück übersetzt und abgearbeitet, sondern einmal vollständig in Maschinensprache übersetzt und dann gestartet. Deshalb sind compilierte Programme wesentlich schneller als BASIC-Programme, aber immer noch deutlich langsamer als reine Assemblerprogramme.

CP/M, engl. Abkürzung für: Control Program for Microcomputers. CP/M ist ein spezielles Betriebssystem, das schon zu einer Art Standard geworden ist. Deshalb gibt es für dieses Betriebssystem eine riesige Menge an Software. Allerdings läuft CP/M nur auf dem Prozessor Z-80. Der kann aber auf Modul nachgerüstet werden, und so ist es auch möglich, CP/M auf dem Commodore 64 laufen zu lassen.

Cursor. Die blinkende Schreibmarke, die anzeigt, wo auf dem Bildschirm gerade geschrieben wird.

Datasette. Der spezielle Kassettenrecorder für den Commodore 64, mit dem Programme auf gewöhnlichen Musikkassetten aufgezeichnet werden können.

Dezimalzahlen. Zahlensystem, basierend auf der Zahl 10, oder einfach: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 usw.

Directory. Das Inhaltsverzeichnis einer Diskette. Es liegt auf der Spur 18. Hier sind alle Programme, deren Länge und Filetyp aufgeführt, die auf einer Diskette gespeichert sind.

Disassembler. Ein Hilfsprogramm, das Maschinenprogramme im Speicher als «Opcodes» der Assemblersprache darstellt.

DOS, engl. Diskette Operations System. Das Betriebssystem der Floppy.

Drucker. Peripheriegerät zum Ausdrukken von Texten, Listings usw. auf Papier.

Editor. Der Teil des Betriebssystems, der für die Bildschirmeingabe zuständig ist (also Cursorbewegung, Farbwahl, Eingabe von Befehlen usw.).

Fehlerkanal. Der Kanal zwischen Floppy und Computer, auf dem die Fehlermeldungen des DOS übertragen werden. Er wird mit OPEN 1,8,15 geöffnet.

File, engl. «Akte». Allgemeine Bezeichnung für eine Aufzeichnung auf Diskette – egal, ob Programm, Datei oder ähnliches.

Floppy. Peripheriegerät zum Abspeichern von Daten und Programmen auf Disketten.

Format. Standard, nach der die Spuren und Sektoren auf einer Diskette aufgeteilt sind. Es ist bei Laufwerken verschiedener Firmen meist unterschiedlich.

Gerätenummer. Da die Peripheriegeräte hintereinandergehängt werden und alle an die gleiche Leitung angeschlossen sind, muß jedes Gerät erkennen können, wann die Daten auf dem Kabel für dieses Gerät bestimmt sind. Dazu hat jedes Gerät eine Gerätenummer. Bei der Floppy ist das die 8, beim Drucker 4 und beim Printer-Plotter 6. Bei vielen Geräten kann diese Gerätenummer geändert werden.

Hardware. Hardware nennt man die Geräte und ihre Bauteile, also den Computer, die Chips usw.

Hardwarehilfe, Hardwareerweiterung. Davon spricht man, wenn besondere Geräte oder Bauteile notwendig sind, um eine bestimmte Aufgabe zu erfüllen, beispielsweise Module zur 80-Zeichen-Darstellung oder für Sprachsynthese oder Scanner, Zeichenbretter usw.

Hexadezimalzahlen. Besonderes Zahlensystem, das auf der Zahl 16 basiert. Ähnlich wie bei den Binärzahlen werden jetzt bei 16er-Potenzen neue Stellen aufgemacht. Die Ziffern sind: 0, 1, 2 ... 8, 9, A, B, C, D, E, F. Zur Kennzeichnung dieser Hexadezimalzahlen wird der Zahl ein $ vorangestellt. Also: $0F = 15, $10 = 16, $FF = 255. Dieses Zahlensystem findet vor allen Dingen bei Monitorprogrammen Anwendung, weil dadurch große Zahlen auf relativ engem Raum dargestellt werden können.

Hires, engl. Abkürzung für High Resolution Graphics, also hochauflösende Grafik. Von ihr spricht man, wenn bei einer grafischen Darstellung direkte Kontrolle über jeden einzelnen Punkt des Bildschirms besteht.

I/O-Register, engl. Abkürzung für Input/Output-Register. Das sind Adressen, die nicht zu einem Speicherbaustein gehören, sondern deren Ansprechen bestimmte Betriebszustände bei einem Peripheriebaustein auslöst. Solche Register sind meist einzelne Bits. Ist zum Beispiel das Bit Nr. 5 im Register 53 265 eingeschaltet, befindet sich VIC im Hires-Modus. Wird es gelöscht, zeigt VIC wieder normalen Text.

IC, engl. Abkürzung für Integrated Circuit = integrierter Baustein. Auf ihnen basiert die gesamte Computertechnik. Ehemals komplizierte Schaltungen, die nur mit Röhren bzw. Transistoren und Kabeln realisiert werden konnten, werden nun durch Mikroelektronik auf engstem Raum zusammengefaßt (integriert).

Initialisierung. Das «Aufwachen» des Computers: Wenn ein Computer oder ein Peripheriegerät eingeschaltet wird, läuft zunächst das Initialisierungsprogramm. Es richtet den Speicher ein, aktiviert die anderen Bauteile usw.

Interface, engl. Schnittstelle. Um Peripheriegeräte mit einer anderen Anschlußbuchse am Commodore 64 betreiben zu können, braucht man ein Interface. Es formt die Signale entsprechend um, so daß z. B. ein Drucker mit Centronics-Interface an den seriellen Bus des Commodore angeschlossen werden kann. Besonders verbreitete Interfaces: Centronics, IEEE-488, V 24.

Interpreter. Das (Maschinen-)Programm, das die BASIC-Befehle interpretiert, also für das Betriebssystem und damit für den Prozessor übersetzt.

Interrupt. Das Betriebssystem unterbricht alle 1/60-Sekunde sein laufendes Programm, prüft, ob eine Taste gedrückt wird, läßt den Cursor blinken, erhöht die

interne Uhr, fragt die Datasette ab usw. Während dieses Interrupts aktualisiert VIC auch die Bildschirmausgabe. Auch andere Bausteine sind vom Interrupt betroffen.

Joystick, engl. Steuerknüppel. Peripheriegerät zur Steuerung von Spielen. Zwei Joysticks können gleichzeitig am Commodore 64 betrieben werden.

Kaltstart. Von einem Kaltstart eines Programms spricht man, wenn quasi bei 0 angefangen wird. Die Variablen haben noch keine Werte (vgl. Warmstart).

Kanalnummer. Auch Filenummer. Die erste Zahl, die beim OPEN-Befehl angegeben wird. Sie kann zwischen 1 und 255 liegen. Kanalnummern über 127 hängen an jede Übertragung einen zusätzlichen CHR$(10) (Line Feed-Code). Zweckmäßigerweise verwendet man für die Kanalnummer dieselbe Zahl wie die Gerätenummer.

Kernal. Anderer Name für Betriebssystem.

Kollisionsdetektor. Spezielle Funktion des VIC. Er stellt fest, ob ein Sprite mit einem anderen Objekt zusammengestoßen ist. Dies ist vor allem bei der Spieleprogrammierung wichtig. Die Register 53 278 und 53 279 von VIC übernehmen diese Aufgabe.

Kommandokanal. Das ist die andere Richtung des Fehlerkanals. Über diesen Kanal werden Befehle (Kommandos) zur Floppy geschickt. Er wird eröffnet mit OPEN 1,8,15.

kompatibel. Zwei Geräte, Programme usw. sind kompatibel, wenn sie miteinander oder einer anstelle des anderen eingesetzt werden können. Kompatibel sind zum Beispiel die Drucker VC-1525 und MPS-801 oder zwei Textprogramme, die die gleichen Textdateien verwenden. Nicht kompatibel sind z. B. 8-Zoll- und 5¼-Zoll-Disketten oder ein Kühlschrank und der Commodore 64.

Lightpen. Ein Lightpen (Lichtgriffel) kann am Joystickport angeschlossen werden. Er besteht im wesentlichen aus einer Fotozelle. Mit der nötigen softwaremäßigen Unterstützung ist es dann z. B. möglich, mit diesem stiftähnlichen Gerät direkt auf den Bildschirm zu zeichnen.

Maschinensprache. Dies ist, im Gegensatz zu BASIC, die Sprache, die der Prozessor direkt versteht. Sie ist komplizierter, weil sie jede Aktion des Prozessors einzeln vorgeben muß, wegen der wegfallenden Übersetzungszeit aber wesentlich schneller. Die Operationen in Maschinensprache werden üblicherweise durch Assembler-Opcodes dargestellt. Beispielsweise steht LDA #$00 für «A-Register des Prozessors mit der Zahl 0 laden». Ein POKE 53280,0 sieht in Maschinensprache so aus:

```
LDA #$00
STA $D020
```

Mehrfarbmodus. Multicolormodus. Spezielle Betriebsart des VIC. Je zwei Punkte werden zu einem zusammengefaßt, der dann eine von vier möglichen Farben haben kann.

Modul. Ein Modul enthält meist zusätzliche Hardware, die über den Erweiterungsanschluß mit dem Commodore 64 verbunden wird. Dabei kann es sich um einen Koprozessor (z. B. den Z-80 für CP/M) oder ROMs handeln, auf denen Software abgespeichert ist.

Monitor. 1. Fernsehähnlicher Bildschirm zur Darstellung des Computerbildes. Er erreicht eine größere Schärfe und Auflösung als ein normaler Fernseher. 2. Spezielles Programm, um Speicherzellen direkt zu ändern und Maschinenprogramme einzugeben.

numerische Variable, z. B. A. Eine Variable, die nur einen Zahlenwert aufnehmen kann.

Opcode. Eine besondere Darstellung für Maschinenbefehle. Ein Maschinenbefehl ist ja eigentlich nur eine Strom-an-/Strom-aus-Kombination. Um diese für den Menschen besser verständlich zu machen, verwendet man Opcodes, z. B. LDA #$00 für «Akkumulator mit 0 laden» oder STA $01 für «Wert im Akkumulator in Adresse 1 ablegen». Weiteres dazu bei Maschinensprache.

Paddle. Spezielles Gerät zu Steuerung von Spielen. Im Gegensatz zu einem Joystick wird nicht nur ein Richtungssignal angegeben, sondern über einen veränderlichen Widerstand ein analoges Signal.

Parallelbus, -interface usw. Im Gegensatz zur seriellen Übertragung werden hier mehrere Bits auf parallelen Leitungen (meist 8) gleichzeitig übertragen. Eine solche Leitung ist wesentlich schneller als eine serielle.

Plotter. Druckerähnliches Gerät, das mit Faserschreibern oder Kugelschreibern schreibt und zeichnet. Besonders ist es zur Ausgabe von Grafiken geeignet.

Pufferspeicher. Ein Speicher, der ankommende Bits aufstaut («puffert») und dann erst weitergibt (insbesondere bei der Floppy an den Schreib-/Lese-Kopf und bei einem Drucker an die Druckmechanik). Dies wird hauptsächlich eingesetzt, um Zeit zu sparen.

RAM, engl. Random Access Memory (Schreib-/Lese-Speicher). Speicherbaustein, der sowohl ausgelesen als auch neu beschrieben werden kann. Er wird nach dem Ausschalten gelöscht.

ROM, engl. Read Only Memory (Nur-Lese-Speicher). Festwertspeicher. Er kann nur gelesen werden, sein Inhalt bleibt dafür aber auch nach dem Ausschalten erhalten.

Routine. Anderer Name für Unterprogramm. Teil eines Programms mit einer festen Aufgabe.

Scanner. Spezielle Videokamera, die mit einem Computer verbunden wird, um ein Bild direkt zu «digitalisieren», also in den Speicher (in die Bitmap) zu übertragen.

Schnittstelle. Anderer Name für Interface. Beispielsweise die serielle Schnittstelle oder der Userport des Commodore 64.

Scrolling, engl. Kunstwort aus screen (Bildschirm) und rolling (Rollen), wörtlich also «Bildschirmrollen». Wenn Sie am unteren Bildschirmrand einen Text eingeben, werden die Zeilen oben aus dem Bildschirm herausgeschoben, um neuen Platz zu schaffen. Gleichzeitig wandert der ganze Text um eine Zeile nach oben. Während des LIST- und des PRINT-Befehls kann dieses Scrolling mit ⟨CTRL⟩ verlangsamt werden.

Seite (page). Der Adreßbereich des 6510 wird in 256 Seiten (pages) zu je 256 Bytes unterteilt. Page 0 (die sogenannte Zeropage) geht demnach von 0 bis 255, Page 1 von 256 bis 511, Page 2 von 512 bis 767 usw.

Sektor. Ein Sektor (Block) ist ein Ausschnitt aus einer Spur einer Diskette. Er kann 256 Bytes speichern.

Sekundäradresse. Sie ist die dritte Zahl, die außer Kanalnummer und Geräteadresse bei einem OPEN-Befehl angegeben werden kann. Die Sekundäradresse wählt verschiedene Modi bei einem Peripheriegerät. Bei OPEN 4,4,X ist X die Sekundäradresse. Ihre Funktion wird im einzelnen in den Handbüchern zu den entsprechenden Peripheriegeräten beschrieben.

Serieller Bus, serielles Interface. Hier werden im Gegensatz zu einer parallelen Übertragung die Bits einzeln hintereinander (seriell) gesendet. Diese Art der Datenübertragung ist langsamer. Der Peripherieanschluß des Commodore 64 ist seriell.

SID. Sound Interface Device. Der Chip 6581 ist der Soundchip im Commodore 64.

Software. Im Gegensatz zur Hardware spricht man von Software, wenn die Programme gemeint sind. Erst durch entsprechende Software kann die Hardware voll ausgenutzt werden.

Sprite. Ein Sprite ist ein bewegliches Objekt, das beim VIC aus 21 * 24 einzelnen Punkten besteht. Beim Commodore 64 können acht Sprites gleichzeitig auf dem Bildschirm sein. VIC unterstützt unter anderem die Bewegung, Vergrößerung und Kollisionsabfrage der Sprites.

Spur. Eine Spur ist ein magnetisch vorgezeichneter Ring auf einer Diskette. Man kann sie vergleichen mit einer Rille auf einer Schallplatte. Eine VC-1541-formatierte Diskette hat 35 Spuren.

Synthesizer. Ein Synthesizer ist ein elektronisches Gerät zur künstlichen Musikerzeugung. Der SID des Commodore 64 hat einen solchen Synthesizer eingebaut.

Token. Wenn ein BASIC-Programm im Speicher steht, werden die Befehle als 1-Byte-Codes dargestellt. So wird Platz gespart. Das Token von PRINT ist zum Beispiel 153.

Toolkit. Ein Toolkit ist eine spezielle BASIC-Erweiterung, die hauptsächlich Befehle oder Routinen zur Unterstützung des Programmierers bietet (also beispielsweise automatische Zeilennumerierung, Suchen und Ersetzen von Befehlen in einem Programm, Auflisten der Variablen usw.)

Unterprogramm. Ein Unterprogramm ist Teil eines Programms, das eine feste Aufgabe hat und vom Hauptprogramm aufgerufen wird.

Userport. Das ist die Schnittstelle ganz rechts am Commodore (von hinten gesehen). Sie ist eine Parallelschnittstelle, die frei programmiert werden kann (also jede Leitung kann einzeln als Ein- oder Ausgabeleitung festgesetzt werden). Mit entsprechender Software können so verschiedene Interfaces zu Druckern oder anderen Peripheriegeräten programmiert werden.

Utility. Ähnlich wie Toolkit. Ein Utility ist ein Programm, das dem Programmierer eine bestimmte Arbeit abnimmt. Wird auch «Dienstprogramm» genannt.

VIC. Video Interface Chip. Der VIC-II-6567-Chip ist der Videochip im Commodore 64.

Warmstart. Im Gegensatz zum Kaltstart spricht man von einem Warmstart, wenn zwar das Programm neu gestartet wird, die Variablen usw. aber ihre alten Werte beibehalten, also nicht gelöscht werden.

Zeropage. Die Seite Nr. 0 im Commodore-Speicher. Da dieser Speicherbereich vom Prozessor besonders günstig adressiert werden kann, speichert er hier wichtige und ständig gebrauchte Werte ab (siehe auch PEEK & POKE-Anhang).

PEEK & POKE-Tabelle

In dieser Tabelle wollen wir die wichtigsten Adressen im Commodore 64 besprechen und zeigen, was man mit ihnen anfangen kann.

Adresse Beschreibung

1: 6510 Ein-/Ausgaberegister. Mit dieser Adresse kann man die Speicheraufteilung des 6510 einstellen. Wenn im BASIC- und Kernal-Bereich auf RAM umgestellt werden soll, müssen diese Programme vorher ins RAM kopiert worden sein.

Bits: 0 BASIC-ROM (1) oder RAM (0)
1 Kernal-ROM (1) oder RAM (0)
2 I/O-Register (1) oder Charakter-ROM (0)
3 Datasette: Datenausgabe
4 Datasette: Taste nicht gedrückt (1)/gedrückt (0)
5 Datasette: Motor aus (1)/an (0)
6, 7 unbenutzt, immer 0

Folgende POKEs sind besonders wichtig:
POKE 1,55 Normalzustand
POKE 1,54 für BASIC im RAM
POKE 1,53 für BASIC und Kernal im RAM (siehe unten)
POKE 1,51 für Charakter-ROM

Obwohl Bit 1 nur für das Kernal zuständig ist, schaltet es, wenn man es von BASIC aus löscht, auch das BASIC ins RAM. POKE 1,52 führt dagegen zum Systemabsturz. Das hat uns auch etwas gewundert, ist aber so ...

43/44: Zeiger auf BASIC-Anfang. 43 ist das Low Byte, 44 das High Byte. Mit POKE 43,1: POKE 44,16: POKE 4096,0: NEW wird der BASIC-Anfang auf 4097 verschoben.

45/46: Zeiger auf BASIC-Ende/Variablen-Anfang. Dieser Zeiger zeigt auf das Ende des Programms. Beim Verschieben des BASIC-Bereichs sollte der Zeiger immer 2 Bytes über den Anfang zeigen. Im Beispiel von oben also POKE 45,3: POKE 46,16.

47/48: Zeiger auf Beginn der Datenfelder. Die Inhalte von Feldern wie A(10,10) werden dort gespeichert. Diese Zeiger werden nach dem Verschieben von BASIC selbständig gesetzt.

49/50: Zeiger auf Ende der Datenfelder. Vgl. 47/48.

51/52: Zeiger auf Strings. Die Inhalte der Strings (A$,B$ usw.) wandern von der oberen Speichergrenze nach unten. Dieser Zeiger zeigt auf die untere Grenze. Sie muß immer größer sein als das Ende der Datenfelder (49/50), sonst gibt es einen OUT OF MEMORY-ERROR. Normalerweise braucht sich der Programmierer darum nicht zu kümmern. Bei platzaufwendigen Programmen läßt sich aber mit diesem Zeiger der übrige Speicherplatz ausrechnen.

55/56: Zeiger auf Grenze des Arbeitsspeichers. Diese Zeiger zeigen dem Betriebssystem das Ende von RAM (normalerweise 40960, also PEEK(55)=0, PEEK(56)=160). Wenn über dem BASIC-Speicher noch Maschinenprogramme oder ähnliches untergebracht werden sollen, kann dieser Zeiger nach unten gesetzt werden.

160–162: TI. Hier ist der momentane Wert der Zeitvariablen TI gespeichert.

198: Anzahl der Zeichen im Tastaturpuffer. Wenn bereits gedrückte Tasten unterdrückt werden sollen, kann dieser Zähler auf 0 gesetzt werden: POKE 198,0. Wenn mit simulierter Tastatur gearbeitet wird, kann dieser Zeiger auf die Anzahl der Tasten gesetzt werden (vergleiche 631 bis 640).

203: gedrückte Taste. Mit PEEK (203) läßt sich ermitteln, welche Taste gedrückt wurde. 64 = keine Taste. Vergleichen Sie Bild 10.1.

204: Cursor an/aus. POKE 204,0 läßt den Cursor während eines Programms an der aktuellen Ausgabeposition blinken. POKE 204,1 schaltet ihn ab. Dabei kann der Cursor jedoch in Blinkphase stehenbleiben. Vgl. 207.

207: Cursor in Blinkphase. Mit PEEK (207) läßt sich auslesen, ob der Cursor beim Blinken gerade an (1) oder aus (0) ist. Wenn der Cursor innerhalb eines Programms verwendet wurde, empfiehlt sich, zum Ausschalten POKE 207,0: POKE 204,1 zu verwenden. Dadurch wird sichergestellt, daß er in Aus-Position abgeschaltet wird und nicht stehenbleibt.

211: Spalte für Cursor. Vgl. 214

214: Zeile für Cursor. Wenn der Cursor (und damit die Startposition des nächsten PRINT-Befehls) an eine bestimmte Bildschirmposition gebracht werden soll, müssen die Koordinaten entsprechend gePOKEd werden: POKE 211,X:POKE 214,Y. Um diese Werte dem Betriebssystem zu übergeben, ist außerdem ein SYS 58732 notwendig.

243/244: Farb-RAM. In Low Byte/High Byte ist hier die Adresse im Farb-RAM gespeichert, die zur aktuellen Cursorposition gehört.

256 bis 511: Prozessor Stack. Diese Page 1 ist eine Art Zwischenspeicher für den Prozessor. Wichtig ist für den BASIC-Programmierer in erster Linie, daß er diesen Bereich schön in Ruhe lassen sollte.

631 bis 640: Tastaturpuffer. In diesem Speicherbereich werden die Codes der Tasten abgelegt, die nicht direkt auf den Bildschirm gebracht wurden. Das passiert praktisch nur, wenn eine Warteschleife oder ein BASIC-Programm läuft. Der Zeiger, wie viele Buchstaben sich in diesem Speicher befinden, liegt in (198). Um die simulierte Tastatur anzuwenden, ist folgendes zu tun: 1. Codes der gewünschten Zeichen in den

Tastaturpuffer POKEn. 2. Anzahl der Zeichen in 198 POKEn. 3. Programmausführung beenden (also Computer in den Direktmodus bringen). Folgendes Programm startet sich z. B. immer wieder selbst: 10 POKE 631, ASC("R"): POKE 632, ASC("U"): POKE 633, ASC("N"): POKE 634,13: POKE 198,4

646: aktueller Farbcode. Ein POKE 646,X ersetzt das entsprechende Farb-Steuerzeichen.

648: Bildschirm-RAM. Diese Adresse beinhaltet die Startadresse des Bildschirm-RAM für das Betriebssystem. Um das Bildschirm-RAM zu verschieben, muß auch das Register 53 272 von VIC entsprechend verändert werden. Um das Bildschirm-RAM auf 2048 zu legen, wären folgende POKEs nötig: POKE648,8: POKE 53272,37. Diese Zeile nimmt die notwendigen Änderungen vor. Vergleichen Sie auch (53 272).

649: maximale Größe des Tastaturpuffers. Sie ist 10. Ein höherer Wert ist nicht zu empfehlen, denn dann werden andere Zeropage-Adressen überschrieben. Sie können den Wert aber verringern. Um die Tastatur kurzfristig zu blockieren, kann er auch auf 0 gesetzt werden.

650: Tastenwiederholung. Folgende Tasten können durch POKEs Wiederholfunktion erhalten:

POKE 650,128	alle Tasten
POKE 650,64	keine Taste
POKE 650,0	normale Aufteilung (⟨CRSR⟩, ⟨SPACE⟩ usw.)

653: Flag für ⟨SHIFT⟩, ⟨C=⟩ und ⟨CTRL⟩. Mit PEEK (653) läßt sich erkennen, ob eine der genannten Tasten gedrückt wird.

Bits: 0 ⟨SHIFT⟩
1 ⟨C=⟩
2 ⟨CTRL⟩

657: ⟨SHIFT⟩ + ⟨C=⟩ verriegeln. Mit POKE 657,128 wird die Umschaltung der Zeichensätze durch ⟨SHIFT⟩ + ⟨C=⟩ gesperrt. Mit POKE 657,0 wird diese Sperre wieder aufgehoben. Entspricht CHR$(8) und CHR$(9).

704 bis 766: Dieser Bereich ist frei. Er kann für ein Sprite verwendet werden. Der Spritezeiger wäre dann 11.

774/775: Zeiger auf BASIC-Text bei LIST. In Low Byte/High Byte wird hier die Adresse der Interpreterroutine abgespeichert, die die BASIC-Tokens in Klartext umwandelt. Durch Verändern dieses Zeigers läßt sich ein sehr wirkungsvoller LIST-Schutz erreichen.

785/786: USR-Adresse. Low Byte und High Byte für die Startadresse des Maschinenprogramms, das mit USR angesprungen wird. USR ist ein ähnlicher Befehl wie SYS, nur daß mit ihm noch eine Variable an das Maschinenprogramm übergeben wird.

828 bis 1023: Kassettenpuffer. Hier werden die Bytes zwischengespeichert, die beim Laden von der Datasette ankommen. Solange keine Kassettenoperationen stattfinden, kann dieser Speicherbereich für Sprites verwendet werden.

Bereich	Spritezeiger
832 bis 894	13
896 bis 958	14
960 bis 1022	15

1024 bis 2023: Bildschirm-RAM. In diesen 1000 Bytes liegen die Bildschirmcodes der Zeichen, die auf dem Bildschirm stehen. Bedenken Sie aber, daß das Bildschirm-RAM verschoben werden kann (vgl. 648).

2040 bis 2047: Spritezeiger 0 bis 7. Hier liegen die Zeiger auf die Speicherbereiche, in denen die Sprites 0 bis 7 abgelegt sind. Für die Startadresse eines Sprites gilt: Startadresse = 64 * Spritezeiger.

2048 bis 40959: 38911-Bytes-Arbeitsspeicher RAM

49152 bis 53247: 4K-RAM, nicht für BASIC verfügbar.

53248 bis 53263: Sprite-Koordinaten:

53248: X-Koordinate Sprite #0
53249: Y-Koordinate Sprite #0
53250: X-Koordinate Sprite #1
53251: Y-Koordinate Sprite #1
...
53262: X-Koordinate Sprite #7
53263: Y-Koordinate Sprite #7

53264: Bits Nr. 8 für Sprites #0 bis #7. Wenn eines dieser Bits gesetzt ist, wird die X-Koordinate des entsprechenden Sprites um 256 erhöht.

Bits:		
	0	9. Bit X-Koord. Sprite #0
	1	9. Bit X-Koord. Sprite #1
	...	
	7	9. Bit X-Koord. Sprite #7

53265: Grafikmodus. Die einzelnen Bits dieses Registers haben folgende Belegung:

Bits:		
	0 bis 2	Feinjustierung des Bildschirmfensters in Y-Richtung
	3	24 (0) oder 25 (1) Zeilen Text
	4	Bildschirm ein (1)/aus (0)
	5	Hires-Modus (1 = an)
	6	Hintergrundfarbmodus (1 = an)

Folgende POKEs sind besonders wichtig:

```
POKE 53265,27 normaler Textmodus
POKE 53265,59 Hires-Grafik ein
POKE 53265,91 Hintergrundfarben ein
```

Probieren Sie Werte von 24 bis 31, um ein Gefühl für die Feinjustierung zu bekommen.

53267: X-Position des Lightpen

53268: Y-Position des Lightpen. Wenn Sie einen Lightpen angeschlossen haben, können Sie mit PEEK (53267) und PEEK (53268) seine Koordinaten abfragen. Diese

werden wahrscheinlich nicht exakt mit den Bildschirmpunkten übereinstimmen, so daß Sie in Ihrem Programm die Werte entsprechend umrechnen müssen, um beispielsweise direkt auf den Bildschirm eine Hires-Grafik zeichnen zu können.

53 269: Sprites an/aus. Wenn ein Bit in diesem Register eingeschaltet ist, erscheint der entsprechende Sprite auf dem Schirm.

Bits:		
	0	Sprite #0 an
	1	Sprite #1 an
	...	
	7	Sprite #7 an

53 270: Multicolormodus und andere Register

Bits:		
	0 bis 2	Feinjustierung des Bildschirmfensters in X-Richtung
	3	38 (0) oder 40 (1) Zeichen pro Zeile
	4	Multicolormodus ein (1)/aus (0)
	5 bis 7	unbenutzt. Nicht verändern!

Um den Multicolormodus einzuschalten, verwenden Sie POKE 53270,216.
Probieren Sie auch die Feinjustierung in X-Richtung mit Werten zwischen 200 und 207.

53 271: Vergrößerung der Sprites in Y-Richtung. Wieder ist jedem Sprite ein Bit zugeordnet.

Bits:		
	0	Y-Vergrößerung von Sprite #0
	1	Y-Vergrößerung von Sprite #1
	...	
	7	Y-Vergrößerung von Sprite #7

53 272: Zeichensatz und Bildschirm-RAM. Mit diesem Register können Sie die Lage des Bildschirm-RAMs und des Zeichensatzes festlegen.

Bits:	7 bis 4	Bildschirm-RAM
Bitkombination	Startadresse des Bildschirm-RAM	
0000	0 (nicht zu empfehlen!)	
0001	1024 (Normalwert)	
0010	2048	
0011	3072	
0100	4096	
0101	5120	
0110	6144	
0111	7168	
1000	8192	
1001	9216	
1010	10240	
1011	11264	
1100	12288	
1101	13312	
1110	14336	
1111	15360	

ca.10 Leerstellen 3 bis 1 Zeichensatz
Bitkombination Startadresse des Zeichensatzes

000	0 (hier liegt immer noch die Zeropage!)
001	2048
010	4096 (siehe unten)
011	6144 (siehe unten)
100	8192
101	10240
110	12288
111	14336

Bit 0 ist unbenutzt, aber immer 1.

Wenn VIC den Zeichensatz ab 4096 sucht, liest er ihn in Wirklichkeit ab der Adresse 53 248 (Charakter-ROM, Grafikmodus). Für 6144 wird ihm 55 296 (Charakter-ROM, Textmodus) untergejubelt. Dafür ist der Interface-Baustein verantwortlich, der uns bei Adresse 56 576 noch genauer interessieren wird.

Wenn Sie nun eine bestimmte Kombination erreichen wollen, müssen Sie einfach das Byte aus den oben gezeigten Werten errechnen. Ein Beispiel: Sie wollen das Bildschirm-RAM ab 4096 und den Zeichensatz ab 2048 anlegen.

Der Inhalt der Adresse 53 272 ist dann 0100 001 1, also: POKE 53272,67. Um diese Konfiguration aber wirklich zum Laufen zu bringen, müssen Sie auch noch die Adresse 648, wie dort beschrieben, ändern. Außerdem sollten Sie BASIC auf 5120 verschieben. (Die nötigen POKEs wären: POKE 648,16: POKE 43,1: POKE 44,20: POKE 5120,0: NEW. Vergleichen Sie gegebenenfalls die verschiedenen Adressen.)

53 275: Sprite-Hintergrund-Priorität. Normalerweise wird ein Sprite vor den Buchstaben bzw. vor der Hires-Grafik dargestellt. Wenn nun ein Bit in dieser Adresse gesetzt ist, wird der entsprechende Sprite hinter diesem Hintergrund dargestellt.

Bits:	0	Sprite #0 hinter dem Hintergrund
	1	Sprite #1 hinter dem Hintergrund
	...	
	7	Sprite #7 hinter dem Hintergrund

53 276: Sprites-Multicolor. Wenn ein Bit in dieser Adresse gesetzt ist, wird der entsprechende Sprite in Multicolor dargestellt. Vergleichen Sie dazu auch die Farbregister 53 285 und 53 286.

Bits:	0	Sprite #0 wird in Multicolor dargestellt.
	...	
	7	Sprite #7 wird in Multicolor dargestellt.

53 277: Sprites-Vergrößerung in X-Richtung. Wenn ein Bit in dieser Adresse gesetzt ist, wird der entsprechende Sprite horizontal vergrößert.

Bits:	0	Sprite #0 wird vergrößert
	...	
	7	Sprite #7 wird vergrößert

53 278: Sprite-Sprite-Kollisionsdetektor. Wenn ein Zusammenstoß zwischen zwei Sprites erfolgt, werden die Bits, die den beteiligten Sprites zugeordnet sind, gesetzt. Sie bleiben gesetzt, bis dieser Wert mit PEEK (53278) ausgelesen wird.

53 279: Sprite-Hintergrund-Kollisionsdetektor. Wenn ein oder mehrere Sprites mit dem Hintergrund (Text, Grafik) zusammenstoßen, werden die Bits, die den beteiligten Sprites entsprechen, gesetzt. Sie bleiben gesetzt, bis dieser Wert mit PEEK (53 279) ausgelesen wird.

53 280: Rahmenfarbe. Sie kann von 0 bis 15 gehen.

53 281: Hintergrundfarbe. Ebenfalls von 0 bis 15. Diese Farbe bekommen im Multicolormodus auch die Punkte mit der Bitkombination 00.

53 282: weiteres Hintergrundfarbregister. Dieses Register ist im Multicolormodus interessant: Bei mehrfarbigen Textdarstellungen nehmen die Punkte mit dem Bitmuster 01 diese Farbe an. Im Hintergrundfarbmodus bekommen die Zeichen mit den BS-Codes 64 bis 127 diese Hintergrundfarbe.

53 283: noch ein Hintergrundfarbregister. Im Multicolormodus (Textdarstellungen) nehmen die Punkte mit dem Bitmuster 10 diese Farbe an. Im Hintergrundfarbmodus bekommen die Zeichen mit den BS-Codes 128 bis 191 diese Hintergrundfarbe.

53 284: das letzte Hintergrundfarbregister. Im Hintergrundfarbmodus bekommen die Zeichen mit den BS-Codes 191 bis 255 diese Hintergrundfarbe.

53 285: Sprite-Multicolor-Register 0. Die Punkte mit der Bitkombination 01 bekommen bei Multicolor-Sprites diese Farbe.

53 286: Sprite-Multicolor-Register 1. Bei Multicolor-Sprites bekommen die Punkte mit der Bitkombination 11 diese Farbe.

53 287: Sprite 0 Farbregister. Dies ist im Normalmodus die Farbe von Sprite #0. Im Multicolormodus bekommen die Punkte des Sprites #0 mit der Bitkombination 10 diese Farbe.

53 288: Sprite 1 Farbregister. Vgl. (53 287)
53 289: Sprite 2 s. o.
53 290: Sprite 3 s. o.
53 291: Sprite 4 s. o.
53 292: Sprite 5 s. o.
53 293: Sprite 6 s. o.
53 294: Sprite 7 s. o.

54 272: SID Stimme 1 Frequenz: Low Byte

54 273: Stimme 1 Frequenz: High Byte. Die Frequenzen für die drei Stimmen können als 16-Bit-Wert (von 0 bis 65 535) angegeben werden. Welcher Frequenz in Hertz dies entspricht, findet sich im Commodore-64-Handbuch auf Seite 158.

54 274: Stimme 1 Pulsbreite: Low Byte

54 275: Stimme 1 Pulsbreite: High Byte. Wenn als Wellenform «Rechteck» gewählt wird, muß in diesen beiden Registern die «Pulsbreite» angegeben werden. Sie kann von 0 bis 4095 reichen. Das bedeutet, daß die Bits 7 bis 4 in diesem Register nicht benutzt werden.

54276: Stimme 1 Wellenform usw. Dieses Register ist das eigentliche Kontrollregister.

Bits:
- 0 Ton-Start (1) Stop (0)
- 1 Synchronisation Stimmen 1 und 3 an (1) aus (0)
- 2 Ringmodulation
- 3 Test-Bit (sollte 0 sein)
- 4 Wellenform Dreieck
- 5 Wellenform Sägezahn
- 6 Wellenform Rechteck
- 7 Wellenform Rauschen

Auf Synchronisation und Ringmodulation können wir an dieser Stelle nicht eingehen. Wir empfehlen, diese Bits beim ersten Experimentieren auszuschalten. Damit ergeben sich folgende POKEs:

POKE 54276,17	Dreieckwelle und Tonstart
POKE 54276,33	Sägezahnwelle und Tonstart
POKE 54276,65	Rechteckwelle und Tonstart
POKE 54276,127	Rauschen und Tonstart

Da mit Bit 0 der Ton gestartet wird, sollte die Einstellung dieses Registers als letzte erfolgen.

54277: Stimme 1: Attack/Decay. Für beide Einstellungen dienen jeweils 4 Bit. Es gibt also 16 Stufen.

Bits:
- 7 bis 4 Attack
- 3 bis 0 Decay

54278: Stimme 1: Sustain/Release. Auch hier gibt es für beide Einstellungen je 4 Bits.

Bits:
- 7 bis 4 Sustain
- 3 bis 0 Release

54279: Stimme 2: Frequenz Low Byte

54280: Stimme 2: Frequenz High Byte

54281: Stimme 2: Pulsbreite Low Byte

54282: Stimme 2: Pulsbreite High Byte

54283: Stimme 2: Wellenform usw.

54284: Stimme 2: Attack/Decay

54285: Stimme 2: Sustain/Release

54286: Stimme 3: Frequenz Low Byte

54287: Stimme 3: Frequenz High Byte

54288: Stimme 3: Pulsbreite Low Byte

54289: Stimme 3: Pulsbreite High Byte

54290: Stimme 3: Wellenform usw.

54291: Stimme 3: Attack/Decay

54 292: Stimme 3: Sustain/Release

54 296: Lautstärke. Dieses Register regelt die Lautstärke für alle drei Stimmen.
Bits: 0 bis 3 Lautstärke (0 bis 15)
4 bis 7 Filtermodi usw.
Auch auf die Filterprogrammierung können wir hier nicht weiter eingehen. Die Bits 4 bis 7 sollten deshalb bei Ihren ersten Experimenten aus sein.

54 297: A/D-Converter Paddle 1. Hier liegen die Adressen der im SID eingebauten A/D-Converter. Aus ihnen kann der Wert von Paddle 1 ausgelesen werden: ? PEEK (54297)

54 298: A/D-Converter Paddle 2 s. o.

55 296 bis 56 295: Hier liegt das Farb-RAM. Es kann nicht verschoben werden.

56 320: I/O-Baustein A. Hier wird normalerweise der Joystick 2 abgefragt. Dieses Register wird aber auch benötigt, um zwischen zwei angeschlossenen Paddlesätzen umzuschalten, da nur ein Paar zum SID weitergeleitet werden kann.
POKE 56320, 64 wählt Paddlesatz am Gameport 1
POKE 56320,128 wählt Paddlesatz am Gameport 2
Für diese Umschaltung müssen allerdings die Interrupts abgeschaltet werden, denn die Tastatur ist ebenfalls an diesem Baustein angeschlossen.
Die Bits 3 und 2 sind außerdem für die Feuerknöpfe der Paddles an Gameport 2 zuständig.

56 321: I/O-Baustein B. Hier kann Joystick 1 gelesen werden.
Beim Paddlebetrieb an Port 1 sind die Bits 3 und 2 für die Feuerknöpfe der Paddles zuständig.

56 334: Interrupt Timer Control Register. Dieses Register hat mehrere Funktionen. Ihre wichtigste ist folgende: Da dieser Interface-Baustein alle 1/60-Sekunde das Signal an den Prozessor liefert, das den Interrupt auslöst, kann man durch Abschalten dieses Signals die Interrupts unterbinden.
POKE56334,0 schaltet die Interrupts ab
POKE56334,1 schaltet sie wieder ein.

56 576: VIC Memory Control. Hier hätten wir jetzt also den oft erwähnten Botschafter, der die Adressen für VIC entsprechend umformt. Die Bits 1 und 0 sind dafür zuständig, welcher 16K-Block von VIC adressiert wird.
Dabei gilt folgende Bitbelegung:

Bits	Startadresse des 16K-Blocks
00	49152
01	32768
10	16384
11	0 (Normalwert)

Normalerweise sind also beide Bits an. VIC adressiert den Bereich von 0 bis 16 383. Hier müssen alle für ihn wichtigen Daten, wie Bildschirm-RAM, Zeichensatz, Spriteadressen, Spritemuster usw., liegen.
Der gesamte 16K-Block läßt sich nun verschieben. Dazu muß man die beiden Bits wie gesagt ändern. Das geht folgendermaßen:

POKE 56578, PEEK(56578) OR 3
POKE 56576, (PEEK(56576)AND 252) OR A

A ist der Dezimalwert der oben gezeigten Bitkombination (also 0 für 00, 1 für 01, 2 für 10 und 3 für 11).

Wenn Sie den 16K-Block verschieben, gibt es aber noch folgendes zu bedenken: Die Adresse des Bildschirm-RAMs ändert sich. Sie müssen also (648) und gegebenenfalls (53 272) entsprechend anpassen. Die Spritezeiger sind jetzt entsprechend hinter dem neuen Bildschirm-RAM zu suchen. Also bei einem Bildschirm-RAM ab 16 384 liegen sie jetzt bei 17 400.

In den Speicherbänken ab 16 384 und 49 152 ist das Charakter-ROM nicht verfügbar. Es muß also vorher auf jeden Fall in den entsprechenden RAM-Bereich kopiert werden. Auch die Bitmap und der dafür zuständige Farbspeicher (das neue Bildschirm-RAM) liegt in dieser neuen Speicherbank. Das Farb-RAM ab 55 296 bleibt in jedem Fall an seinem Platz.

56 578: Datenrichtungsregister für VIC Memory Control. Es legt fest, ob die Bits in der Adresse 56 576 beschrieben oder gelesen werden. Vor einer Umschaltung müssen also die Bits 1 und 0 eingeschaltet werden, wie oben im ersten POKE geschehen.

Literaturverzeichnis und Softwarehinweise

Wir wollen Ihnen abschließend gern einige Tips über Bücher und Programme geben, die uns positiv aufgefallen sind. Leider können diese Empfehlungen nur sehr persönlicher Art sein. Weil sich der den Commodore 64 umgebende Markt ständig ausbreitet, ist eine solche Übersicht sowieso nie repräsentativ, ausgewogen oder gar vollständig. Dennoch wollen wir versuchen, Ihnen damit die Entscheidung ein wenig zu erleichtern.

1. Bücher

64 intern. Düsseldorf: Data-Becker.
Das Interface-Age-Systemhandbuch zum Commodore 64. München: Interface Age.
Das Commodore-64-Adreßbuch – PEEK POKE (Chip Special). Würzburg: Vogel-Verlag.
Commodore 64 mit Simon's Basic. Ludwigshafen: Kiehl.
Das große Floppy-Buch. Düsseldorf: Data-Becker.
COMPOTE's First Book of Commodore 64 (engl.) Greensboro: COMPUTE! Publications, Inc.
Mach mehr aus Deinem Commodore 64. Würzburg: Vogel-Buchverlag.
Grafik mit dem Home-Computer. Würzburg: Vogel-Buchverlag

Alles über den Commodore 64. Frankfurt: Commodore

2. Software

An BASIC-Erweiterungen finden wir besonders nützlich:
Simon's BASIC. Commodore.
EXBASIC Level II. Interface Age.
BASIC 64. Omikron.

Weitere Utilities

Supergraphik 64. Data-Becker
Diskomat. Data-Becker.
SM Kit 64. SM Software.
EXDOS. Interface Age.

Anwenderprogramme wie Text- und Dateiverwaltung und Spiele sind sehr vom persönlichen Geschmack abhängig. Sie sollte man sich im Fachhandel vorführen lassen.

Schlußwort

Ja, liebe Leser. Damit hätten wir es geschafft. Unser Studienaufenthalt ist vorbei. Jetzt geht es nur noch darum, die erworbenen Kenntnisse draußen anzuwenden. Ab jetzt sollten Sie das Buch nur noch als Reiseführer verstehen, den man irgendwo mitgenommen hat. Es wird Ihnen sicherlich immer wieder nützlich sein, darin zu schmökern, Informationen zu suchen und dergleichen. Und ehe Sie sich's versehen hatten, hat Sie die BASIC AIRWAYS schon wieder nach Hause gebracht. Die Rückreise ist immer kürzer als die Hinreise. Und wenn es Ihnen gefallen hat, dann schenken Sie doch auch einmal jemandem, der einen Commodore hat, eine Reise mit uns. Wir glauben, daß das genausogut Anfänger wie Fortgeschrittene sein könnten – oder Eltern, die auch ganz gern wüßten, was ihr Sprößling so Besonderes an diesen Kästen findet.

Zugegeben, wir wollten Sie ein bißchen vom Computern begeistern. Vielleicht ist aus Ihnen mittlerweile ein richtiger Freak geworden. Wenn uns das gelungen ist, fänden wir es gar nicht so schlecht. Natürlich müßte man einige Dinge nicht wissen, die wir nebenbei erklärt haben – genausowenig, wie man überhaupt etwas von Computern wissen muß, wenn man mit ihnen

arbeiten will. Im Grunde reicht es aus, den Knopf zum Ein- und Ausschalten zu finden und eine Diskette einzulegen. Trotzdem haben wir viele allgemeine Dinge erklärt, die erst später interessant werden oder gar nicht sehr Commodore-spezifisch sind, zum Beispiel Adreßbus und Datenbus, das High-Byte-/Low-Byte-Prinzip. Aber das sind Dinge, die Sie irgendwann brauchen könnten, und wir wollten Sie ja möglichst umfassend in die Welt der Computer einführen.

Und jetzt noch ein paar Worte über das Problem Computer. Wenn Sie das nicht interessiert, überlesen Sie es einfach. Es sind ein paar sehr persönliche, subjektive Ansichten. Denn wir dachten, ganz ohne das kann man ein unserer Ansicht nach sehr persönliches Buch über Computer eigentlich nicht schreiben.

Vergessen Sie bei allem Eifer nie, wo er ist – der Knopf zum Ein- und Ausschalten. Computer sind gut, aber viele andere Dinge auch. Nur weil Sie Champion im PacMan-Spiel sind, hat das noch lange nichts mit Ihrer Kondition zu tun. Der Computer kann ein Hobby sein. Mancher will vielleicht auch für den Beruf mehr darüber wissen. Es ist auch nur natürlich, daß man ganz am Anfang mehr Zeit in das neue Hobby investiert als später. Und es gehört dazu, daß man nächtelang an einem Problem, an einem Programm sitzt. Die Freude, wenn es plötzlich doch läuft, wenn man etwas Neues entdeckt hat, ist meist immens. Sie darf nur nicht die einzige sein. Der Computer ist nie ein Kommunikationspartner, genausowenig die Menschen, die nur darüber reden können. Nehmen Sie dies als kleinen Tip mit auf den Weg. Denn wir finden, eine gehörige Portion kritischer Abstand gehört genauso zu einem guten Programmierer oder Computerfreak wie das technische Wissen. Warum wir das sagen? Aus dem Grund, aus dem Leute immer Ratschläge geben: aus eigener Erfahrung.

Auch für uns war es schwer, den richtigen Weg zu finden. Aber glauben Sie uns, wir beide sind sehr froh darüber, daß nach fast vier Monaten ständiger Arbeit damit jetzt die Zeit dafür gekommen ist, wieder einen Commodore 64 anschauen zu können, ohne gleich an ein Buch zu denken, das wir noch schreiben sollten. Und wahrscheinlich werden unsere 64er und auch unser Apple (zur Textverarbeitung) jetzt ein bißchen in Ruhe ihr Leben genießen können. Die Arbeit hat Spaß gemacht, aber glücklicherweise ist sie jetzt vorbei – die nächsten Ferien werden wir in erster Linie zum Faulenzen benutzen. (Na ja, vielleicht noch ein bißchen Software schreiben – aber nur ein bißchen ...)

Das war es von uns zu diesem Thema. Wir wollten Sie nur ein bißchen nachdenklich machen. Der Computer hat nichts Böses an sich, es kommt immer nur auf die Leute an, die davor sitzen. Und das gilt für Ihren Heimcomputer genauso wie für die Computer, die Daten erfassen oder die in West und Ost für militärische Zwecke eingesetzt werden. Es gibt Dinge, die sollte man nie dem Computer überlassen. Wir hoffen, daß die Verantwortlichen sich darüber im klaren sind! Deshalb halten wir auch das Wissen über Computer und ihre Arbeitsweise für sehr wichtig. Denn Wissen kann schützen – nicht nur nützen!

Jetzt nur noch eine kleine Bitte: Wir haben, bevor wir mit unserer Predigt anfingen, sehr oft die Wörter «wollten» und «sollten» verwendet. Der Grund dafür ist, daß wir natürlich nicht wissen, ob uns alles geglückt ist, was wir beabsichtigt hatten. Deshalb würden wir uns freuen, von Ihnen zu hören!

Schreiben Sie uns, was Ihnen gefallen oder nicht gefallen hat. Sollten wir uns wieder mal ans Schreiben eines Buches heranwagen, dann gibt es nur eines, das man nicht ersetzen kann: die Erfahrung mit unserem ersten Buch. In diesem Sinne wünschen wir Ihnen viel Freude mit Ihrem Commodore 64 und natürlich bei allen Tätigkeiten, für die Sie – außer für die Computerei – noch Zeit finden ...

Frankfurt/Main

Hannes Rügheimer
Christian Spanik

VOGEL-BUCHVERLAG WÜRZBURG

Senftleben, D.
Start mit Commodore-Logo
Reihe HC –
Mein Home-Computer
ca. 200 Seiten,
zahlr. Abbildungen,
ca. 30,– DM, 1984
ISBN 3-8023-**0802**-6

Wenn Sie aktiv mit Ihrem Commodore 64 in Logo computern wollen, ist dieses Buch die richtige Starthilfe für Sie. Mit dieser Einführung erlernen Sie in 12 Lektionen das kleine Logo-Einmaleins, bis Sie mit Grafik, Text und Musik spielen, experimentieren und arbeiten können. Über große Bildschirmfotos können Sie Ihre Erfolge kontrollieren und schon bald neue Einsatzbereiche erschließen.

Start mit Commodore-Logo

Der Leser dieser Einführung in die Grafik-Programmierung benötigt lediglich Grundkenntnisse im Programmieren mit BASIC. Der Autor hat ein Höchstmaß an Strukturierung und Kommentierung der Programme angestrebt. Sie wurden auf dem Commodore 64 entwickelt und getestet — sind aber so geschrieben, daß sie sich leicht auf andere grafikfähige Mikrocomputer übertragen lassen.

Baumann, Rüdeger
Grafik mit dem Home-Computer
Reihe HC –
Mein Home-Computer
328 Seiten,
zahlr. Abbildungen,
38,– DM, 1984
ISBN 3-8023-**0769**-0

Baumann, Rüdeger
Computerspiele und Knobeleien programmiert in BASIC
Reihe HC –
Mein Home-Computer
304 Seiten,
zahlr. Abbildungen,
4. Auflage 1984
30,– DM
ISBN 3-8023-**0786**-0

Mit Eigeninitiative weg von der Spielkonserve: Der Leser wird zum aktiven und schöpferischen Umgang mit Computerspielen aufgerufen und angeleitet — aus der Spielidee entwickelt sich die Spielstrategie und hieraus das Programm. Das Programmieren des Computers selbst ist das Spiel; so lernt der Leser spielend das Programmieren. Die Programmbeispiele wurden auf Commodore-Computern erstellt.

Mein Home-Computer

Monat für Monat über 30 Seiten Programme

Und das bringt

Mein Home-Computer

jeden Monat:

- ✻ Programme für alle gängigen Home-Computer
- ✻ Anwendungsbeispiele aus der Praxis
- ✻ Marktübersicht, Tests und Kaufberatung für Zusatzgeräte und Home-Computer
- ✻ Schnellkurse für Einsteiger zum Sammeln
- ✻ Tips und Tricks
- ✻ Interessantes, Aktuelles und Unterhaltsames aus der Home-Computer-Szene
- ✻ News, Clubnachrichten

Holen Sie sich die neueste Ausgabe bei Ihrem Zeitschriftenhändler oder fordern Sie ein Kennenlernheft direkt beim Vogel-Verlag, Leserservice HC, Postfach 67 40, 8700 Würzburg, an.

Ihr erstes Commodore-64-Buch war das Handbuch, das Sie mit dem Gerät erhielten. Doch als Anfänger können Sie damit meist keine allzu großen Sprünge machen. Wenn Sie die Bedienung und Programmierung des Commodore 64 trotzdem ohne viele Vorkenntnisse lernen wollen, sollten Sie dieses Buch lesen. Es will Sie aber auch unterhalten und amüsant sein, denn aller Anfang ist schon schwer genug!

Zuvor sollten Sie auf jeden Fall das Geräte-Handbuch durchgelesen und das eine oder andere Beispiel ausprobiert haben. Dann verhilft Ihnen dieses «Zweite Commodore-64-Buch» zu den Erfahrungen, die Sie zum selbständigen Programmieren brauchen. Schließlich will dieses Buch Ihr «Sprungbrett» sein zu weiterführender Computerliteratur.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0793-3