Markt&Technik

Das Commodore 128 Handbuch

Peter Rosenbeck

Ein unentbehrliches Nachschlagewerk für Profis:
BASIC 7.0 • CP/M 3.0 • Peripherie • Einsprungadressen
Schnittstellen • Bank-Switching

Das Commodore 128 Handbuch

Ein Handbuch, das als ständiger Begleiter bei der Arbeit mit dem C 128 PC konzipiert ist. Der fortgeschrittene Leser kann es mit ebensoviel Gewinn für seine Arbeit verwenden wie der Anfänger, dessen erster Computer der C 128 PC ist. Hier ist nur ein Ausschnitt aus den Themenbereichen, die das Buch behandelt:

- Es gibt dem Anfänger eine Einführung in die Möglichkeiten des C 128 PC samt Erklärung der wichtigsten Fachbegriffe und der Entstehungsgeschichte des C 128 PC.
- Sie werden mit dem C 64-Modus und der Benutzung von CP/M 3.0 vertraut gemacht.
- Es erklärt das Arbeiten mit den Grafik- und Soundmöglichkeiten des C 128 PC.
- Es führt den Leser in die Arbeitsweise des Systems ein: Die Techniken der Speicherverwaltung und das Banking werden besprochen.
- Es zeigt Ihnen eine Einführung in die Programmierung mit Assemblersprache.
- Es zeigt, wie man die Programmierung des 80-Zeichen-Bildschirms bewerkstelligen kann.

Den Handbuch-Charakter unterstreicht der umfangreiche Nachschlageteil. Dieser enthält unter anderem:

- Eine komplette Beschreibung des BASIC 7.0 mit vielen nützlichen Hinweisen, Beispielen und Tricks.
- Eine Tabelle der wichtigsten Betriebssystem-Calls mit Erläuterung.
- Ein Verzeichnis der Assembler-Befehle des 8502-Prozessors.

ISB N 3-89090-195-6

Markt&Technik

4 001057 901957

DM 52,-
sFr. 47,80
öS 405,60

Peter Rosenbeck

Das Commodore-128-Handbuch

Ein unentbehrliches Nachschlagewerk für Profis:

BASIC 7.0
CP/M 3.0
Peripherie
Einsprungadressen
Schnittstellen
Bank-Switching

Markt & Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Rosenbeck, Peter:
[Das Commodore-hundertachtundzwanzig-Handbuch]
Das Commodore-128-Handbuch : e. unentbehrl. Nachschlagewerk für Profis: BASIC 7.0, CP/M 3.0, Peripherie, Einsprungadressen, Schnittstellen, Bank-Switching / Peter Rosenbeck. –
Haar bei München : Markt-und-Technik-Verlag, 1985.
ISBN 3-89090-195-6

15 14 13 12 11 10 9 8 7 6
90 89 88 87

ISBN 3-89090-195-6

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Inhaltsverzeichnis

1 Kapitel Eins...

... welches eine kurze aber notwendige Einführung in die Computer-Kultur enthält, den neugierigen Neuling mit manch fremdem Wort bekannt macht und in die Historie und Prähistorie der Computer einführt: also dem geneigten Leser allerlei Kurzweil bietet; die Harten Facts kommen später!

Eigentlich ist Ihr C128 ein Computer, den es gar nicht gibt! Warum? Weil er in der Computerwelt so etwas wie das eierlegende Wollmilchschwein darstellt. Er ist nämlich zugleich Homecomputer und Personal Computer, zugleich Spiel- und Lernzeug, er kann Ihnen aber auch im Geschäft helfen, er ist drei Computer in einem, vereinigt die Vorzüge eines bewährten Konzepts mit einer Vielzahl neuer innovativer Ideen und bringt zwei bisher getrennt existierende Computerwelten unter einen Hut... und das alles zu einem sensationell günstigen Preis.

Worterklärung: Homecomputer *sind kleine Computer, die für den häuslichen Gebrauch bestimmt sind. Sie haben meist Eigenschaften, die mehr an den Spieltrieb des Käufers appellieren, als es die der Personal Computer tun. Diese sind teurer als Homecomputer und für den Geschäftsbereich gedacht. Da sie mit ernsthaften Aufgaben betraut werden, gehen ihnen die wichtigsten Attraktionen der Homecomputer meist ab: Grafik und Musik. Dafür können sie auf ihren Bildschirmen mehr Informationen darstellen und auch mehr Daten speichern und bewältigen als die Homecomputer.*

Warum das so ist, können Sie sich am besten klarmachen, wenn ich Ihnen einiges über die Geschichte der Commodore-Computer erzähle. Hier ist er also: der Stammbaum Ihres C128.

1.1 Die Ahnentafel

Die Firma Commodore - ursprünglich ein Hersteller von Büromaschinen, der sich in den 70er Jahren einen Halbleiter-Hersteller einverleibte - war bis ca. 1980 hauptsächlich den Anwendern aus der Geschäftswelt ein Begriff. Ihre Personal Computer - die Maschinen aus der sogenannten

CBM-Serie - erfreuten und erfreuen sich in Deutschland bei kleinen und mittleren Betrieben, aber auch in so manchem Großkonzern ungebrochener Beliebtheit.

Wohl hauptsächlich als Reaktion auf den großen Erfolg, den Heim-Videospiele Ende der 70er Jahre hatten, baute Commodore seinen ersten Homecomputer: der VC 20 war mit seinem eingebauten BASIC, der Möglichkeit zu Farbgrafik und zur Erzeugung von Toneffekten, seinem Arbeitsspeicher von 5K, besonders aber wegen des günstigen Preises auf dem damaligen Markt ein Knüller. In kurzer Zeit wurde der VC 20 zum Bestseller, der wohl eine ganze Generation von Benutzern mit den Freuden der Computerei bekanntmachte.

Worterklärung: BASIC *ist eine besonders einfache und leicht zu erlernende Programmiersprache, die sich nicht nur bei Anfängern großer Beliebtheit erfreut. Heutzutage hat beinahe jeder Homecomputer BASIC eingebaut, kann also sofort in dieser Sprache programmiert werden. Dies war früher keineswegs selbstverständlich, und die Benutzer kleinerer Computer mußten sich oft mit der lästigen und archaischen* Maschinensprache *(gleichsam die "Muttersprache" des Computers) herumschlagen.*

Worterklärung: *Ein* "K" *ist eine Maßzahl für die Kapazität des Arbeitsspeichers des Computers. Der Arbeitsspeicher ist eine Art Kurzzeitgedächtnis für den Computer: je größer er ist, desto kompliziertere Aufgaben kann er erledigen. Ein Arbeitsspeicher von 1K kann ungefähr 1000 Zeichen aufnehmen, 5K bedeuten also eine "Merkfähigkeit" von ca. 5000 Zeichen.*

Aber die Konkurrenz schläft nicht: auch andere Hersteller hatten Homecomputer oder brachten neuentwickelte auf den Markt. Außerdem steht besonders in der Mikroelektronik die technische Entwicklung nicht gerade still. Vielmehr werden auf diesem quirligen Markt laufend neue Erfindungen gemacht und ein Computermodell ist nach fünf Jahren oft schon hoffnungslos veraltet. Um auf dem Gebiet der Homecomputer die Nummer Eins zu bleiben, mußte sich Commodore also etwas einfallen lassen.

Dabei kamen der Firma zwei Entwicklungen zugute, die der hauseigene Chip-Hersteller *MOS Technology* ausgebrütet hatte. Ein Grafik-Chip und ein Musikchip. Der Grafikchip trägt den etwas merkwürdigen Namen *6567 Video Interface Chip II* und ist bei seinen Freunden auch kürzer und prägnanter als *VIC-II* bekannt. Auch die Bezeichnung des Musikchips erinnert eher an einen der beiden Roboter aus *Star Wars*: *6581 Sound Interface Device* oder kurz *SID*.

Worterklärung: *Ein* Chip *ist eine auf einem winzigen (wenige Quadratmillimeter großen) Siliziumplättchen oder -schnipsel (daher der Name) unterge-*

brachte hochintegrierte elektronische Schaltung. Die "schlauen" Teile Ihres Computers, die seine Intelligenz ausmachen, sind allesamt Chips. Es lassen sich auch spezialisierte Chips entwerfen, die - wie etwas ein Grafikchip - *sich besonders auf die Erzeugung von effektvollen Grafiken verstehen. Ein* Musikchip *ist eigentlich ein winziger Synthesizer, mit dem durch entsprechende Programmierung (und mit einem Lautsprecher) beliebige Klänge produziert werden können.*

VIC-II und SID waren für den Einsatz in Spielhallen-Automaten gedacht, auf denen die mittlerweile - wir schreiben das Jahr 1982 - aberwitzig erfolgreichen Action-Spiele laufen. Wer schon mal in einer Spielhalle war, der wird - ob er nun was von Videospielen hält oder nicht - zugeben müssen: die Grafiken und Klänge, die diese Geräte erzeugen, sind schon erstaunlich. Auch hier ist die Konkurrenz hart; MOS mußte sich also mit den beiden Chips gehörig anstrengen. Ihre Fähigkeiten sind entsprechend.

Worterklärung: *Bei Computern spricht man dann von* Grafik, *wenn sie auf dem Bildschirm etwas anderes als Zeichen und Ziffern ausgeben. Das können Kurven sein, die man zur Veranschaulichung statistischer Daten benutzt, aber auch Bilder und Figuren (Autos, Raumschiffe, Spieler), die man für Videospiele braucht.*

Die Großtat von Commodore bestand nun darin, VIC-II und SID in einen Homecomputer einzubauen. Damit alleine hätte die Firma bereits der Konkurrenz einen Pluspunkt vorausgehabt. Es kam aber noch besser: da Speicherbausteine mittlerweile nicht mehr mit Gold aufgewogen werden mußten sondern in großen Serien günstig zu haben waren, wurde der neue Computer mit satten 64K Arbeitsspeicher versehen: beinahe 13 mal mehr als sein Vorgänger. Eine Sensation auf dem damaligen Markt!

Worterklärung: *Auch der Arbeitsspeicher des Comuters besteht aus Chips; die einzige Fähigkeit dieser* Speicherbausteine *besteht darin, Informationen aufzubewahren (solange man ihnen den Strom dafür gibt) und auf Verlangen blitzschnell wieder auszuspucken.*

Dies, zusammen mit einer käuferfreundlichen Preispolitik, verhalf dem C64 - das "C" kommt von Commodore, das "64" von den 64K - zu einem beispiellosen Erfolg. Die Abstimmung mit dem Geldbeutel in den Computershops, Elektrogeschäften und Kaufhäusern fiel eindeutig aus. Selbst heute noch wird der C64 als die Nummer eins auf den Verkaufslisten geführt.

Warum Sie das interessieren sollte? Nun, weil Sie sich ja einen C64 gekauft haben!!

1.2 Der C128 ist ein C64...

Genaugenommen: einen C64 haben Sie sich *auch* gekauft. Ihr Computer, der C128, kann nämlich auf Wunsch seine Identität wechseln und zu einem C64 werden. Da der C64 der Vorgänger vom C128 ist, haben wir hier also den seltenen Fall eines Kindes, das sein eigener Vater werden kann.

Wozu ist das gut? Erstens ist der C64 alles andere als ein schlechter Computer. Zweitens ist er ein Bestseller und wurde millionenfach verkauft. Folglich gibt es drittens jede Menge Software dafür (geschätzt: mehr als 60000 Programme): deswegen!

Worterklärung: *Für Ihren Computer ist* Software *das, was die Schallplatten, Kassetten und CDs für Ihre Stereoanlage sind: ohne wär's nur ein stummer und dummer Kasten. Erst das Programm macht ihn schlau. "Programm" und "Software" ist übrigens dasselbe. "Soft" nennt man diese Ware, weil man sie nicht anfassen kann - oder haben Sie schon mal die Töne auf Ihren Schallplatten gesehen?*

Denn auch der beste Computer nutzt Ihnen nichts ohne Software - es sei denn, Sie sind ein Hacker und erfreuen sich der höheren Weihen der Computerei. Wir gewöhnlichen Sterblichen aber sind für die Benutzung des Computers auf die Hilfe anderer angewiesen, wir brauchen Programme. Der C64 kann wirklich erstaunliche Spiele mit Ihnen spielen; einige davon sind so gut gemacht, daß sogar ein ausgebuffter Öko-Freak und Computer-Hasser sich vergessen und zum Joystick greifen könnte. Der C64 kann eine Lernmaschine sein, ein Gerät, mit dessen Hilfe den Kids sogar die Mathematik Spaß macht und das das Herz im Leibe eines Pädagogen vor Freude hüpfen läßt.

Aber all dies kann er nicht von Haus aus, es ist nicht in ihn eingebaut. Dazu braucht es Programme. Die sind, wenn sie gut sein sollen, gar nicht so leicht zu erstellen. Weswegen sich ein Anbieter überlegt, ob er sich die Mühe machen und einige hochbezahlte und meist auch noch leicht verrückte Programmierer mit der Ausarbeitung eines Computerprogrammes betrauen soll. Wenn sich aber ein Computer so oft verkauft wie der C64, dann sieht die Sache schon anders aus. Denn dann hat unser Anbieter einen riesigen Markt und das Wagnis könnte sich lohnen.

Der Verkaufserfolg des C64 - ursprünglich wohl auf die überragenden Möglichkeiten der Hardware zurückzuführen - führte dazu, daß sich bald eine Unzahl von Software-Häusern auf dem Markt drängelte. Konkurrenz ist gut für die Kunden: die Programme wurden immer besser und immer billiger. Hinzu kamen die Benutzer selbst: manch einer entdeckte sein Talent für das Programmieren und es ist eigentlich gar nicht erstaunlich,

daß viele der besten Programme für den C64 nicht von Profis, sondern von begeisterten Hobbyisten stammen. Letztere werden in der Sprache der Insider übrigens auch "Freaks" genannt.

Dieser überwältigende Fundus von guten und billigen Programmen ist mit das stärkste Argument für den Kauf eines C64. Oder war es: denn jetzt ist es ein Argument für den Kauf des C128. Denn dieser kann seine Identität wechseln und auf Befehl zum C64 werden. Alle Programme für den C64 laufen somit auch auf dem C128.

Wenn sowas geht - wenn also ein Computer die Programme oder Bauteile eines anderen Computers benutzen kann - dann spricht man von "Kompatibilität". Die etwas erfahreneren und wettergegerbteren unter den Lesern werden jetzt vielleicht müde lächeln. Schon mancher Hersteller hat von seinem Produkt behauptet, es sei zu den Vorgängermodellen oder zu Geräten anderer Hersteller kompatibel. Bei genauerem Hinsehen war's aber dann mit der versprochenen Verträglichkeit nicht allzu weit her. Die Hersteller von Homecomputern haben sich um Kompatibilität oft am allerwenigsten gekümmert. Auch Commodore bietet dafür ein Beispiel. Denn der C64 ist zu seinem Vorgänger, dem VC20 - sehr zum Leidwesen der vielen VC20-Besitzer - nicht im geringsten kompatibel.

Anders der C128. Eines der wichtigsten Ziele von Commodore bei seiner Entwicklung war es, den vielen Besitzern eines C64 die Benutzung ihrer bereits vorhandenen Software zu ermöglichen. Sollten Sie ein Umsteiger vom C64 sein, dann kann ich Sie hiermit beruhigen: die Kompatibilität ist wirklich vollständig. Selbst Programme, die auf üble Weise im Betriebssystem herumpfuschen, laufen anstandslos auf dem C128. Ihre Softwarebibliothek ist also von bleibendem Wert.

Worterklärung: *Das* Betriebssystem *des Computers ist die Software, die seinen Betrieb erst ermöglicht, indem sie ihm unter anderem zeigt, wie er die Benutzerwünsche entgegennehmen (die Tastatur lesen) und seine Ergebnisse mitteilen (Informationen auf einem Bildschirm ausgeben) kann.*

Wenn der C128 Ihr erster Computer ist, dann bedeutet das für Sie, daß Sie auf ein unerschöpfliches Reservoire von Software zurückgreifen können. Besonders die Spiele und Lernprogramme für den C64 sind hier von großer Attraktivität. Die Programme für "geschäftliche" Anwendungen, die ebenfalls in großer Zahl feilgeboten werden, sollten Sie jedoch nicht so ohne weiteres zu kaufen erwägen. Denn als Besitzer eines C128 gibt es eine bessere Lösung für Sie. Aber dazu kommen wir noch.

1.3 Der Zweite von den Dreien: der 128er-Modus

"Wenn der C64 so gut ist, warum sollte ich mir dann einen C128 kaufen, der ja auch noch etwas teurer ist?" Nach dem Lob, das ich im letzten Kapitel dem C64 gesungen habe, liegt diese Frage nahe. Aber bedenken Sie: der C64 ist - jedenfalls nach den Maßstäben der schnellebigen Computerwelt - mittlerweile schon etwas betagt. Die Entwicklung ist nicht stehengeblieben, und heutzutage stellt der Käufer ganz andere Anforderungen an einen Homecomputer als noch vor drei Jahren. Und einiges gibt es am C64 ja auch auszusetzen.

Da ist zum einen die Tatsache, daß der 64er - trotz aller Beteuerungen von Commodore und ungezählter Software-Lieferanten - für ernsthafte geschäftliche Anwendungen nicht geeignet ist. Es gibt zwar eine ganze Menge Geschäftsprogramme für ihn, aber Textverarbeitung, Datenbanken, Tabellenkalkulation und was der Büromensch sonst noch so braucht: für den C64 sind sie, im Vergleich mit professionellen Programmen, nur Spielzeugversionen.

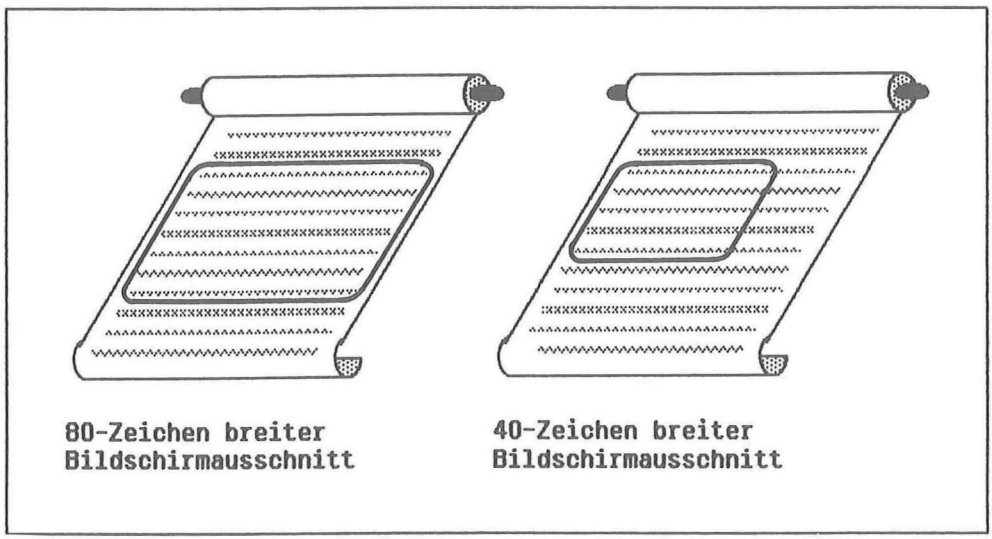


Abbildung 1.1: *Der Vorteil eines größeren Bildschirm-Fensters*

Dies hat hauptsächlich zwei Gründe. Der C64 kann auf seinem Bildschirm nur 40 Zeichen pro Zeile auf insgesamt 25 Zeilen darstellen. Als Standard für Geschäftsprogramme hat sich jedoch ein Bildschirmformat von 80 Zeichen pro Zeile und 24 Zeilen auf dem Schirm etabliert. Schon am Bei-

spiel der Textverarbeitung kann man sich klarmachen, daß die geringe Zeichenzahl des C64 einen gravierenden Nachteil darstellt. Es ist etwa so, als ob Sie einen längeren Text - z.B. einen Geschäftsbericht oder ein Referat - in normal großer Schrift auf lauter kleinen Zettelchen niederschreiben würden. Bei der Textverarbeitung stellt der Bildschirm ja nur einen Ausschnitt des gesamten Textes dar; je kleiner dieser ist, desto unbequemer läßt es sich arbeiten. Der zweite Grund für die Untauglichkeit des C64 im harten Business: seine Floppy.

Worterklärung: *Eine* Floppy *dient zum dauerhaften Speichern von Daten. Auf einer Art biegsamen Schallplatte, der Diskette, die sich im Laufwerk - eben der Floppy - dreht, werden mit einem magnetischen Aufzeichungsverfahren die Informationen aufgezeichnet. Dazu und zum erneuten Abrufen der Informatione dient ein Lese-/Schreibkopf, der sich ähnlich wie bei einem Plattenspieler über der Oberfläche der Diskette hin- und herbewegen und so schnell zu jedem gewünschten Punkt gelangen kann.*

Kommerzielle Programme müssen Daten speichern. Denken Sie nur an einen Kaufmann, der seinen C64 als elektronische Kasse verwenden möchte. Jeder Posten eines Einkaufs soll sofort auf dem Computer erfaßt und daraufhin eine Gesamtrechnung erstellt werden. Er möchte sich die Einkaufsdaten natürlich längerfristig speichern, um sie später analysieren und auswerten zu können. ("Verkaufen sich die blauen Gummibärchen besser als die roten?"...) Da die Informationen im Arbeitsspeicher nach Abschalten der Betriebsspannung verlorengehen, benötigt er ein dauerhaftes Speichermedium. Dafür nimmt man meist eine Floppy, da diese mehr Speicherkapazität hat und eine bequemere Handhabung erlaubt als ein Kassettenrecorder. Der Kaufmann speichert also jeden Posten einer Rechnung zuerst auf Diskette und läßt dann per Programm die Rechung ausdrucken.

Worterklärung: *Für Benutzer mit schmalerem Geldbeutel gibt es als Alternative zur Floppy die Möglichkeit, Daten auf* Kassette *aufzuzeichnen. Bei Commodore kann er dazu leider nicht jeden beliebigen* Kassettenrekorder *benutzen, sondern er muß auf ein spezielles Gerät zurückgreifen. die "Datasette". Die Informationsaufzeichnung funktioniert hier ähnlich wie bei einem Kassettenrecorder, nur werden statt Musik eben Computerdaten auf die Kassette aufgenommen. Die Datasette ist zwar billiger als die Floppy, dafür aber weniger zuverlässig und langsamer.*

Wenn er dieses mit dem C64 und seiner Floppy vorhat, dann sollte er besser vor seiner Kasse eine größere Anzahl bequemer Sessel aufstellen. Wozu? Um den Kunden die Wartezeit zu versüßen! Denn die Floppy des C64 ist so langsam, daß sich bald lange Schlangen von ungeduldigen Kunden bilden werden.

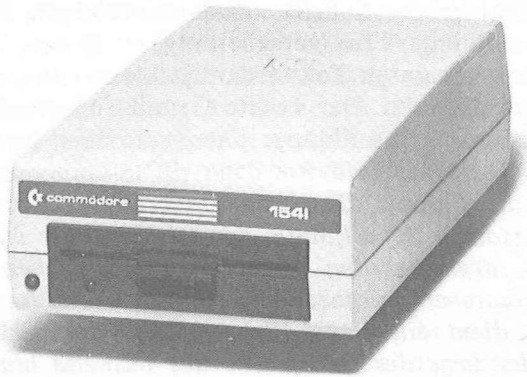


Abbildung 1.2: *Die Floppy 1541*

Die Floppy des C64 (die den Namen "1541" trägt) kann dem Computer die von ihm gewünschten Informationen nämlich nur mit einer Geschwindigkeit von 300 Zeichen pro Sekunde übermitteln. Das kommt einem beim ersten Hören durchaus ausreichend vor. Aber die einzigen, die mit dieser Übertragungsgeschwindigkeit wirklich zufrieden sein können, sind die Zigaretten- und Kaffeehersteller. Denn in der Praxis ergeben sich durch diese Übertragungsrate so lange Wartezeiten, daß man z.B. beim Laden eines größeren Programmes ruhig schon mal eine Tasse Kaffee zu sich nehmen und/oder eine Zigarette rauchen kann, bis der Vorgang beendet ist.

Worterklärung: *Programme für den Computer zeichnet man dauerhaft auf einem Datenträger auf, meist einer Diskette. Ehe der Computer ein solches Programm ausführen kann, muß er es erst einmal von diesem Datenträger "herunterholen" und in seinen Arbeitsspeicher - sein Kurzzeitgedächtnis - übertragen. Diesen Vorgang nennt man* Laden.

Geschäftsprogramme wie die oben erwähnten haben nun die Eigenschaft, notorisch oft auf die Diskette zuzugreifen. Die Zwangspausen, die das jedesmal nach sich zieht, verleiden dem Benutzer meist nach kurzer Zeit die Lust an der Arbeit.

Es gibt also zwei zentrale Schwachpunkte im Design des C64: sein popeliger Bildschirm und die Mickymaus-Floppy. Beide Defekte sind im C128 beho-

ben: Der zweite der drei Computer, die im Gehäuse Ihres C128 verborgen sind, kann einen 80-Zeichen-Bildschirm bedienen, so Sie einen haben. Er kann auch mit einer neuen Floppy arbeiten, einem Gerät namens "1571", das nicht nur schicker aussieht als sein Vorgänger. Vielmehr kann die 1571 Informationen fünfmal schneller an den oder vom Computer weitergeben als ihre Vorgängerin. Eine sehr angenehme Eigenschaft für den Benutzer, wenn auch vielleicht weniger erstrebenswert für die Kaffee- und Zigarettenindustrie!

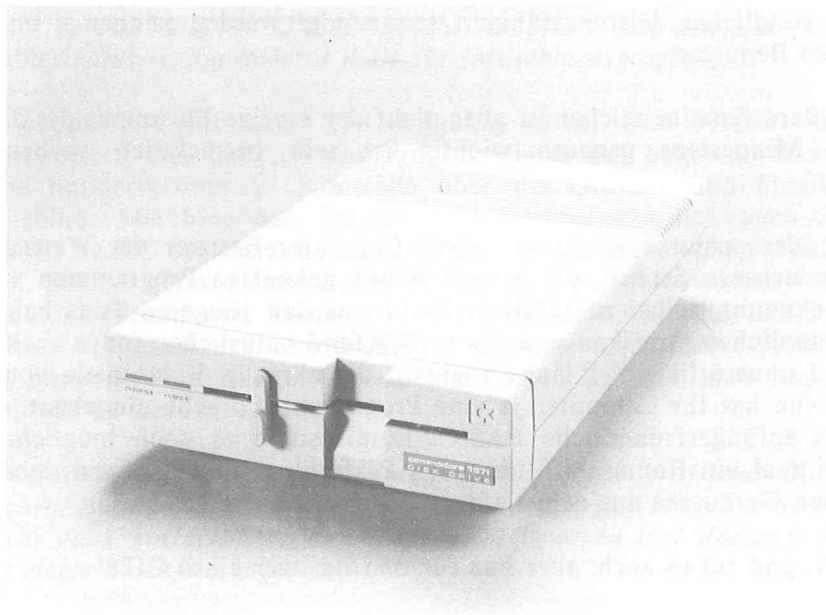


Abbildung 1.3: *Die Floppy 1571*

Da wir jetzt schon über zwei unterschiedliche Computer reden müssen, wird es Zeit, sich mit der dafür üblichen Ausdrucksweise vertraut zu machen. Commodore spricht in diesem Zusammenhang von "Modi" bzw. "Betriebsarten". Die Tatsache, daß der C128 drei Computer in einem ist, hört sich in offiziellem Commodore-Deutsch so an: "Der C128 verfügt über drei Modi". Wie Sie jetzt wissen, steckt in Ihrem Gerät ein vollständiger C64. Auf Commodorianisch: "Der C128 verfügt über einen C64-Modus, in dem er zu seinem Vorgängermodell voll kompatibel ist." Dann ist die Kiste aber auch noch eine erheblich verbesserte Nachfolgeversion des C64. Originalton Commodore: "Der 128er-Modus weist eine Vielzahl von Verbesserungen und Erweiterungen gegenüber dem Vorgängermodell auf". Zu diesen Verbesserungen und Erweiterungen kommen wir noch; ebenso zu dem dritten Computer, den Sie erworben haben und der bei Commodore als "CP/M-Modus" bezeichnet wird.

Der zweite Computer ist also der "eigentliche" C128; warum aber wird er so genannt? Mal sehen: der C64 verdankt ja seinen Namen der Tatsache, daß er über 64K Arbeitsspeicher verfügt. Nun ist verdächtigerweise 128 gleich zweimal 64. Sollte also...?

Genau! In Ihrem Computer ist doppelt soviel Arbeitsspeicher drin wie im C64!! Wenn der Arbeitsspeicher aber so groß ist, dann kann sich der Computer darin mehr Daten merken - und das macht ihn schneller. Außerdem kann er längere und komplexere Programme verarbeiten, die dadurch benutzerfreundlicher, leistungsfähiger, spannender werden können - und das freut den Benutzer.

Der größere Arbeitsspeicher ist aber nicht der einzige Pluspunkt des 128er-Modus. Mindestens genauso wichtig ist sein beträchtlich verbessertes BASIC.

Früher oder später erwacht in jedem Computer-Besitzer der Wunsch, all die wunderbaren Sachen, die er von seinen gekauften Programmen vorgeführt bekommt, selbst zu machen. Nicht nur den jüngeren Fans haben es die erstaunlichen Grafiken und die verblüffend natürlichen (oder auch verblüffend unnatürlichen) Klänge angetan, die aktuelle Videospiele von sich geben. Nun hat Ihr Computer ja eine Programmiersprache eingebaut, nämlich das anfängerfreundliche BASIC. Damit sollte es doch möglich sein, selbst einmal ein Raumschiff über den Bildschirm zu jagen und dabei die passenden Geräusche aus dem Lautsprecher ertönen zu lassen.

Sollte es, und tut es auch: aber nur für den Besitzer eines C128!

Wer nämlich nur einen C64 besitzt, stellt nach einiger Zeit zu seinem großen Frust fest, daß die Ton- und Grafikmöglichkeiten gar nicht in BASIC programmiert werden können. Das eingebaute BASIC des C64 kann die beiden Chips nicht direkt steuern, ohne deren Mithilfe jedem Spiel die Würze abgeht. Wer sich dennoch nicht entmutigen ließ, der mußte sich mit sehr maschinennahen Details abmühen und mittels PEEKs und POKEs im direkten Zugriff auf die Steuerregister der beiden Bausteine arbeiten. Diese Methode ist sehr umständlich und fehleranfällig. Viele Freaks bissen daher in den sauren Apfel und quälten sich mit Maschinensprache ab. Dazu hat aber beileibe nicht jeder Zeit und Lust. Die Konsequenz: für den "normalen" Benutzer waren die fortgeschritteneren Möglichkeiten außerhalb seiner Reichweite, wenn er sich nicht zusätzliche Hardware- oder Software-Erweiterungen für seinen C64 zulegte.

Worterklärung: *Obwohl in Ihren Computer BASIC eingebaut ist und es so aussieht, als ob dies seine "Muttersprache" wäre: in Wirklichkeit ist es gar*

nicht so. Eigentlich versteht der Computer nur die Maschinensprache, *ein archaisches und primitives Kauderwelsch, das genau wiedergibt, worin die eingebauten Fähigkeiten der Maschine bestehen. Da Computer dumm sind - sie können nur zwischen 0 und 1 unterscheiden - ist es damit nicht weit her. Man kann jedoch diese Grundfähigkeiten benutzen, um den Computer zu "erziehen", um ihm Schwierigeres beizubringen. Eines dieser schwierigeren Dinge ist es, BASIC zu verstehen. Denken Sie also daran: der Computer ist so programmiert, daß er BASIC versteht. Jeder BASIC-Befehl wird aber letztendlich in eine ganze Folge von Befehlen aus der Maschinensprache übersetzt: einem knappen BASIC-Befehl entspricht also ein Kurzroman in* Assembler *(dies ist ein anderes Wort für "Maschinensprache").*

Diese Zeiten sind mit dem C128 endgültig vorbei! Sein BASIC wurde so stark erweitert, daß dadurch der Programmierer auf bequeme Weise eine beinahe hundertprozentige Kontrolle über die Grafik- und Musik-Bausteine seiner Maschine hat. Sprites (und selbstverständlich auch Shapes) können in BASIC definiert, bewegt und auf verschiedene Arten manipuliert werden. Klänge können erzeugt, modifiziert und gefiltert werden - und das alles mit einfachen und komfortablen BASIC-Befehlen. Sie können jetzt also Ihrer optischen und akustischen Kreativität ungehindert freien Lauf lassen, ohne sich mit lästigen Details abzuplagen und erst zum Assembler-Spezialisten zu werden.

Worterklärung: Sprites *sind selbstdefinierte farbige grafische Objekte - Männchen, Autos, Flugzeuge, was Sie wollen - die der Programmierer unter anderem ganz einfach über den Bildschirm bewegen und dadurch Effekte wie in einem Trickfilm erzielen kann.*

Die Erweiterungen des BASIC im 128er-Modus gehen aber nicht nur in eine Richtung. Vielmehr wurden in den Sprachumfang Befehle mit aufgenommen, die ein strukturierteres Programmieren erlauben.

Was soll nun das schon wieder: "strukturiertes Programmieren"?

Die Programmiersprache BASIC war in den letzten Jahren ein wenig in das Kreuzfeuer pädagogisch und theoretisch motivierter Kritiker geraten. In der professionellen Softwareindustrie hat sich ein Programmierstil als der nützlichste erwiesen, den man mit dem Schlagwort "strukturiertes Programmieren" kennzeichnet. Die Kritik an BASIC konzentrierte sich folglich im wesentlichen darauf, daß BASIC diesen Programmierstil angeblich nicht zuläßt. Darüber kann man sich nun streiten, und die treuen BASIC-Fans werden dem sicher auch stürmisch widersprechen. Sicher ist jedoch: BASIC erzwingt diesen Stil nicht.

Das ist in der neuesten BASIC-Version von Commodore, der Version 7.0 des C128, geändert. Neue Befehle erlauben es, Programmschleifen mit bisher nicht gekannter Eleganz und Klarheit zu formulieren und so eine wesentliche Forderung der Programmier-Pädagogen zu erfüllen.

Worterklärung: *Ebenso wie die natürlichen - von Menschen gesprochenen - Sprachen sind auch die Programmiersprachen nicht "tot". Vielmehr ändern sie sich im Laufe der Zeit. Dies gilt ganz besonders für BASIC, an dem in seiner mittlerweile über zwanzigjährigen Entwicklung viele Verbesserungen vorgenommen wurden. Man spricht in diesem Zusammenhang von "Versionen". Wenn ein BASIC-Hersteller eine neue Entwicklung hat, dann baut er sie in sein BASIC ein und schafft damit eine neue BASIC-Version. Das BASIC des C64 war die Version 2.0; im C128 findet sich die Version 7.0.*

Weitere in das neue BASIC mit aufgenommene Befehle betreffen die Arbeit mit der Floppy. Um sich z.B. anzusehen, welche Daten auf einer Diskette gespeichert sind, mußte der C64-Benutzer ein richtiges kleines BASIC-Programm schreiben. Mit

```
LOAD "$",8
LIST
```

bekam er beinahe dasselbe Ergebnis wie der Besitzer eines C128 mit

```
DIRECTORY
```

Beinahe - denn das C64-Programm bewirkt nicht nur eine Anzeige des Inhaltsverzeichnisses, sondern löscht nebenbei auch noch ein eventuell im BASIC-Speicher befindliches Programm. DIRECTORY ist da zivilisierter und tut das nicht.

Noch ein letztes Beispiel zur Abschreckung. Um nach einer Diskettenoperation den Status der Floppy abzufragen (zwecks Fehlerüberprüfung) sagt man auf dem C128 kurz und knapp:

```
? DS$
```

Hingegen auf dem C64:

```
100 OPEN1,8,15
110 INPUT#1,ER,ER$,T,S
120 PRINT ER;" ";ER$;" ";T;" ";S
130 CLOSE1
RUN
```

Kommentar überflüssig!

Als ob das alles nicht schon genug wäre, hat man im 128er-Modus auch noch die Möglichkeit, mit einem einfachen Befehl den Computer doppelt so schnell zu machen! Sie sagen einfach

```
FAST
```

und er gehorcht Ihren Befehlen mit doppelter Geschwindigkeit.

1.4 Wie vertragen sich die beiden?

Vermutlich werden Sie sich vor lauter "C64" und "128" und hin und her schon gefragt haben, wie das Verhältnis der beiden Modi zueinander ist. Wie wird Ihr Computer ein C64 bzw ein C128?

Beim Einschalten des Geräts befinden Sie sich automatisch im 128er-Modus (wenn Sie nicht eine spezielle Diskette in Ihre Floppy legen; darüber später mehr). Sie haben dann das BASIC 7.0 zu Ihren Diensten und können nach Herzenslust über die 128K Ihres Arbeitsspeichers verfügen. Sollten Sie nun in den C64-Modus wechseln wollen, - vielleicht um ein Spielchen zu wagen - dann brauchen Sie nur das Kommando

```
GO64
```

zu geben. Der Weg vom 128er- zum 64er-Modus ist allerdings eine Einbahnstraße: ist man erst mal in diesem Modus, so führt kein Weg zurück zum 128er, außer dem Abschalten der Maschine oder dem Betätigen des RESET-Knopfs. Deshalb werden Sie vor dem Wechsel vorsichtshalber gefragt, ob Sie das auch wirklich wollen. Auf die Frage:

```
ARE YOU SURE (Y/N)
```

können Sie jetzt mit "Y" antworten (für engl. "Yes") oder es sich noch anders überlegen und "N" eingeben.

Noch einmal zur Warnung: aus dem 64er-Modus gibt es keinen Weg zurück zum 128er-Modus. Den kann es auch gar nicht geben, da es ja dazu eines Kommandos bedürfte, das der C64 gar nicht hat. Enthielte der 64er-Modus so ein Kommando, dann wäre er aber nicht mehr voll kompatibel zum C64! Wenn Sie also den Schritt vollzogen haben, dann ist z.B. ein BASIC-Programm, das Sie zuvor im 128er-Modus bearbeitet haben, ein für allemal dahin (wenn Sie es nicht zuvor gespeichert haben)!

1.5 Der Dritte im Bunde

Während der 64er-Modus und der 128er-Modus enge Verwandte sind, kommt der dritte Ihrer Computer aus einer ganz anderen Ecke: aus der rationellen und nüchternen Welt der geschäftlichen Anwendungen. Neben den Modi "64" und "128" gibt es einen dritten Modus, den "CP/M-Modus". Allerdings steht dieser Modus nur für Anwender zur Verfügung, die über eine Floppy verfügen.

Die beiden Welten "Homecomputer" und "Personal Computer" waren bisher säuberlich getrennt. Die Homecomputer-Industrie konzentrierte sich darauf, die Grafik- und Musikfähigkeit ihrer Geräte zu verbessern und durch immer raffiniertere BASIC-Erweiterung den Komfort für die Benutzer zu steigern. Grafik und Musik waren in der Geschäftswelt jedoch lange ohne Bedeutung. Die Hauptanforderungen an einen Personal Computer lauteten: 80-Zeichen-Bildschirm und schnelle Diskettenoperationen. Dies wurde wiederum - hauptsächlich aus Kostengründen - bei den Homecomputern vernachlässigt; man sieht es am C64.

Die Entwicklung der Personal Computer lief also unabhängig zu der bei Homecomputern. Sie ist hauptsächlich mit einem Namen verknüpft: CP/M. Das ist die Abkürzung für "Control Program for Microcomputers" und bezeichnet ein Betriebssystem, das auf die Verwaltung von Disketten und Dateien spezialisiert ist.

Der normale Homecomputer-Benutzer ist sich meist gar nicht darüber im Klaren, daß es ein Betriebssystem gibt. Wenn er den Computer einschaltet, dann befindet er sich sofort im BASIC. Alles, was er mit dem Computer tut, erscheint ihm als eine Fähigkeit von BASIC: das Lesen der Zeichen, die er auf der Tastatur tippt, das Ausgeben von Informationen auf dem Bildschirm oder Drucker, die Einrichtung von Dateien auf der Diskette, deren Beschreiben und Lesen usw. - all dies scheint zu den "angeborenen" Fähigkeiten von BASIC zu gehören.

Tut es aber nicht! BASIC versteht davon eigentlich gar nichts. Es hat jedoch einen Arbeitsknecht zur Verfügung, der sich auf diese Dinge versteht und dem BASIC seine Befehle erteilt, wenn es dergleichen Dinge ausgeführt haben will. Schreibt man also ein kleines BASIC-Programm, um jemanden freundlich zu begrüßen, wie etwa:

```
PRINT "GUTEN TAG"
```

Dann sagt das BASIC zum Knecht: "Gib mir doch mal GUTEN TAG auf dem Bildschirm aus!" und kümmert sich nicht mehr um die Angelegenheit. der Knecht aber rackert sich ab, er weiß, was er mit dem Bildschirm zu

tun hat, damit darauf nacheinander an der richtigen Stelle zuerst ein G, dann ein U, dann ein T usw. erscheint. Der Knecht ist das Betriebssystem; bei Commodore (im 64er- und 128er-Modus) heißt er "Kernal".

Wir merken uns: Beim 64er- und 128er-Modus ist das Betriebssystem hinter dem eingebauten BASIC verborgen. Der Kernal tritt bescheiden zurück in die zweite Reihe.

Anders im CP/M-Modus. Hier "spricht" der Benutzer mit dem Betriebssystem CP/M. Dieses versteht nur eine bescheidene Anzahl an Befehlen, die sich hauptsächlich auf Disketten und Dateien beziehen. Aber es kann einen 80-Zeichen-Bildschirm bedienen; und seine Diskettenoperationen sind wirklich schnell. Während die Datenübertragung zwischen 1571 und Computer im 128er-Modus mit 1500 Zeichen pro Sekunde läuft, kommen unter CP/M in der Sekunde 3500 Zeichen rüber - mehr als zehnmal soviel wie im betulichen 64er-Modus!

Schnelle Diskettenoperationen und ein 80-Zeichen-Bildschirm nutzen aber nichts, wenn man keine Programme hat, die diese Möglichkeiten ausnutzen. Hier aber liegt CP/Ms große Stärke.

Für CP/M gibt es nämlich annähernd so viele Programme wie für den C64. Während aber die überwiegende Mehrzahl der 64er-Programme der Unterhaltung dient, ist das Verhältnis bei CP/M gerade umgekehrt. Es existieren weltweit ca. 10000 CP/M-Programme und davon sind mehr als 90% geschäftlicher Natur! Während aber die für den C64 erhältlichen Business-Programme mehr oder weniger Notlösungen sind, handelt es sich bei den einschlägigen CP/M-Produkten um echte Profi-Werkzeuge.

Hier nur ein kurzer Vorgeschmack der Möglichkeiten: der Klassiker unter den Textsystemen - WordStar - ist ein CP/M-Programm. Ebenso *das* Datenbanksystem - dBASE II - und *die* Tabellenkalkulation - VisiCalc. Diese drei reichen schon aus, um einen mittleren Betrieb zu managen. Und sie laufen auf Ihrem C 128! Daneben gibt es unter CP/M auch eine Vielzahl von Branchenlösungen, also Programme, die auf die speziellen Bedürfnisse eines Betriebs zugeschnitten sind und Aufgaben wie Lagerverwaltung, Fakturierung, Lohn- und Gehaltsabrechnung übernehmen.

Wenn Sie ein Programmierer sind, dann wird es Sie freuen, daß unter CP/M auch so gut wie jede Programmiersprache in einer ernstzunehmenden Version erhältlich ist. BASIC sowieso, aber auch Pascal, C, COBOL, FORTRAN, PL/1, Ada, Modula II, ja sogar solche Exoten wie LISP oder PROLOG, Sprachen, die in der Künstlichen Intelligenz eingesetzt werden.

Der CP/M-Modus unterscheidet sich in einem ganz wesentlichen Punkt von den Betriebsarten 64 und 128: er setzt einen eigenen Prozessor voraus.

Worterklärung: *Der* Prozessor *ist das eigentliche Herzstück eines Computers. Eigentlich nichts anderes als ein Chip, übernimmt er doch die Regie im ganzen System und hat gegenüber den anderen Bausteinen das Sagen. Für kleine Computer werden hauptsächlich zwei "Familien" von Prozessoren verwendet. Die eine Familie stammt von Commodores Elektronik-Tochter MOS Technology. Zu ihren Mitgliedern zählen die Typen 6502, 6510 (im C64 vertreten) und 8502 (im C128 vertreten). Die andere Familie geht auf die Firma Intel zurück; ihr prominentestes Mitglied ist die 8080. Eng damit verwandt ist die Z80, die sich ebenfalls im C128 findet. Wie es in der Computerbranche leider der Brauch ist, ist die Z80 nicht im geringsten mit der 8502 kompatibel. Die beiden haben nichts miteinander gemeinsam.*

CP/M und die dafür erhältliche Software läuft nämlich nur auf Computern, die als Prozessor einen 8080 oder einen dazu kompatiblen Baustein aufweisen. Während der 64er- und 128er-Modus den selben Prozessor benutzen können, ist für CP/M also ein eigener Baustein erforderlich. Die Ingenieure von Commodore haben sich da für den Z80 entschieden, den wohl leistungsfähigsten 8080-kompatiblen Prozessor. Diese Entscheidung stellt eine sensationelle Hochzeit zweier Welten dar. Denn bisher war den Homecomputern die Welt der wirklich leistungsfähigen Geschäfts-Software verschlossen. Die CP/M-Benutzer hingegen mußten für ihre Freizeitgestaltung zu anderen Mitteln als dem Computer greifen. Was es nämlich unter CP/M an Spielen gibt, ist dem verwöhnten C64-Benutzer nicht mal ein müdes Lächeln wert.

Die Gründe für das Defizit an Geschäftsprogrammen für den C64 kennen Sie ja bereits. Daß es unter CP/M so wenig Spiele gibt, hat seinen Grund nicht in der Leistungsfähigkeit von CP/M-Computern; die wäre mehr als ausreichend. Die Ursache ist vielmehr die Tatsache, daß es mehr als einen CP/M-Computer gibt und auch geben soll. CP/M ist so gebaut, daß es ohne große Probleme auf verschiedene Rechner übertragen werden kann. Jeder Computer benutzt aber einen anderen Bildschirm, eine andere Tastatur, andere Disketten-Laufwerke usw. Deshalb ist CP/M in zwei Teile gegliedert: einen hardwareabhängigen Teil - den Insidern auch unter der Bezeichnung "BIOS" bekannt - und einen universellen Teil, in dem die ganze Logik von CP/M untergebracht ist, das "BDOS".

Die Commodore-Spiele leben von der Grafik; diese ist jedoch stark von der Hardware abhängig (dem VIC-II). Ein CP/M-Programm, das ja auf vielen verschiedenen Computern laufen soll, darf aber an die Hardware seiner Maschine keine anderen Anforderungen stellen, als daß Buchstaben und Ziffern auf dem Bildschirm darstellbar sind. Farben? Sprites? Vorder- und

Hintergrund? All das ist hardwarespezifisch und darf unter CP/M nicht benutzt werden, wenn das Programm auf allen CP/M-Rechnern laufen soll. Wer Programme schreibt, um damit seinen Lebensunterhalt zu verdienen, hat daran aber ein lebhaftes Interesse, da so der Interessentenkreis vergrößert wird. Darum gibt es so gut wie keine anspruchsvollen professionellen Videospiele unter CP/M.

Dies dürfte sich aber ziemlich bald ändern. Denn im C128 kann der CP/M-Programmierer auf alle Bausteine des Computers zugreifen, also auch auf SID und VIC-II. Da der Markt entsprechend groß ist, kann man wohl mit dem baldigen Erscheinen interessanter CP/M-Programme für den C128 rechnen.

CP/M hat übrigens auch eine lange Geschichte hinter sich. Es war das erste Betriebssystem für Microcomputer und ist wohl auch deswegen - und natürlich wegen seiner Vorzüge - zum Industriestandard geworden. Die Microcomputer der ersten Generation konnten - ebenso wie der C64 - nur 64K Arbeitsspeicher benutzen. Als CP/M entwickelt wurde dachte niemand daran, daß dies einmal zu wenig sein könnte. Speicherbausteine waren nämlich teuer und schienen es noch eine Zeitlang zu bleiben.

Die Preisentwicklung auf dem Chip-Markt hat aber, wie wir mittlerweile wissen, selbst die kühnsten Träume übertroffen. Aus Kostengründen ist heute also kein Hersteller gehindert, mehr als 64K in einen Computer zu bauen. Aber CP/M kann nicht mehr verwalten!

Genauer: konnte es nicht. Die Hersteller von CP/M - die Firma Digital Research - haben nämlich reagiert und ihr System völlig neu überarbeitet. Es entstand die CP/M-Version 3.0, welche mehr als 64K Arbeitsspeicher bedienen kann und auch sonst noch viele Verbesserungen aufweist. Und Commodore hat natürlich diese neueste Version in Ihren Computer eingebaut! Mehr noch: auch ein notorisches CP/M-Problem wurde auf elegante Weise von Commodore gelöst. Dies betrifft die Diskettenformate.

Worterklärung: *Eine fabrikneue Diskette kann vom Computer nicht ohne Vorbereitung benutzt werden. Ehe man Informationen darauf aufzeichnen kann, muß sich das System an bestimmten Stellen der Diskette Markierungen anbringen, die ihm erlauben, die Daten korrekt einzuordnen. Das ist ein bißchen so, wie wenn Sie ein leeres Zimmer (die fabrikneue Diskette) als Bibiliothek verwenden wollen. Ehe Sie die Bücher (die Daten) "speichern" können, müssen Sie erst Regale (entspricht den Markierungen) aufstellen. Dieses Aufbringen von Markierungen und Steuerinformationen nennt man* Formatieren.

CP/M half zwar mit, die Welt der Personal Computer zu standardisieren. Was aber die verschiedenen Computerhersteller nicht daran hinderte, sich für ihre Diskettenlaufwerke jedesmal ein neues Format einfallen zu lassen. Eigentlich kann jeder CP/M-Computer Daten und Programme jedes anderen CP/M-Computers lesen - wenn die Disketten-Formate übereinstimmen. Eine ziemlich vollständige Formate-Übersicht neueren Datums brachte es aber auf 84 (!) verschiedene CP/M-Diskettenformate! Darunter befinden sich eine Menge Formate von teilweise recht exotischen Fabrikaten, denen man selten begegnet.

Die Entwickler des C128 sind nun den Weg gegangen, 9 Diskettenformate in das System einzubauen und per Knopfdruck wählbar zu machen. Neben den beiden hauseigenen Commodore-Formaten können Sie also durch einfache Kommandos von der Tastatur bestimmen, ob Sie eine Diskette für einen Osborne-, Kaypro-, Epson- oder IBM-Computer lesen bzw. beschreiben wollen. Für IBM sind außerdem insgesamt 4 Formate vorhanden. Wem das noch nicht reicht, der kann mithilfe eines Spezialprogramms auch noch auf andere, seltenere Formate umschalten. Dem Datenaustausch steht somit nichts mehr im Wege!

Sie sehen also: mit Ihrem C128 sind Sie auf der Höhe der technischen Entwicklung und haben von allem das Beste. Die folgenden Kapitel geben Ihnen nun detailliertere Informationen zu den Modi Ihres C128, so daß Sie die überragenden Möglichkeiten Ihres Computers auch wirklich ausschöpfen können.

2 Ihr Computer und wie Sie damit arbeiten

Früher füllten Computer ganze Turnsäle, brauchten eigene Klimaanlagen und machten einen Höllenlärm. Heute haben Sie ein schick gestiltes flaches Gehäuse vor sich - entfernt an eine Reiseschreibmaschine erinnernd - das geräuschlos und dienstbereit auf Ihrem Schreibtisch steht und auch noch gut aussieht. Trotz der äußerlichen Unterschiede: was die Rechenleistung betrifft, so ist Ihr C128 von dem lärmenden und sündteuren Ungetüm von vor 15 Jahren gar nicht mal so weit entfernt.

Sie werden das bei der praktischen Arbeit mit dem System bald selbst entdecken. Kapitel Zwei wird Sie mit den wichtigsten Kenntnissen versehen, die Sie bei dieser Entdeckungsreise brauchen. Es widmet sich den äußeren Bestandteilen des C128 - den Bedienelementen und Anschlüssen - und zeigt Ihnen, wie Sie damit umgehen.

2.1 Die Tastatur

Die Tastatur Ihres Computers weist 92 Tasten auf. Wenn Sie bisher nur mit der Tastatur einer Schreibmaschine vertraut waren, dann ist das eine verwirrende Fülle. Wie Sie jedoch bald sehen werden, ist der Aufbau der Tastatur klar und logisch. Sie ist nämlich in mehrere Bereiche untergliedert, die unterschiedliche Funktionen zu erfüllen haben.

Bei genauerer Betrachtung erkennt man, daß die von der Schreibmaschine gewohnten Tasten alle auf der Tastatur zu finden sind, wenngleich auch teilweise mit weiteren merkwürdigen Zeichen beschriftet. Diesen Teil der Tastatur nennen wir die Schreibmaschinen-Tastatur; er ist in der Abbildung 2.2 hervorgehoben dargestellt.

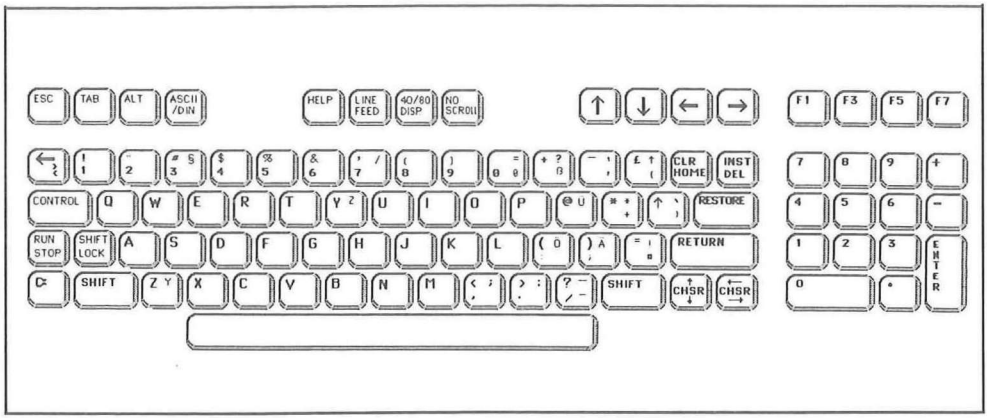


Abbildung 2.1: *Die Tastatur des C128*

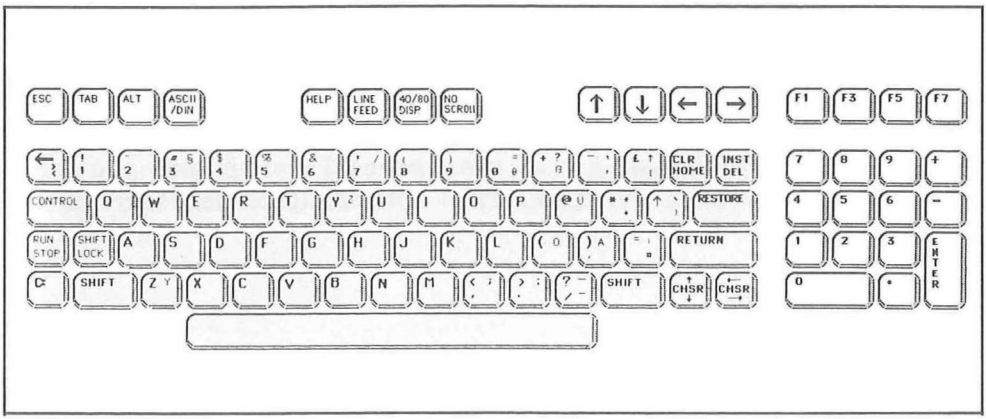


Abbildung 2.2: *Die Schreibmaschinentastatur*

Die Schreibmaschinen-Tastatur brauchen Sie, um Ihre Befehle an den Computer zu erteilen oder ihn mit Daten zu versorgen (im Volksmund: "füttern"), die er für seine Arbeit braucht. Wenn Sie z.B. mit einem Textprogramm arbeiten, können Sie damit ganz normale Texte wie etwa Briefe erfassen, die dann auf dem Bildschirm erscheinen, aber auch gespeichert werden können.

Jede Schreibmaschine weist mehrere Tasten auf, die nicht der Erzeugung von Zeichen dienen, sondern mit denen bestimmte Funktionen der Maschine gesteuert werden können. Da ist zum einen die Tabulator-Taste, die ja bei Betätigung kein Zeichen auf das Papier bringt, sondern dafür sorgt,

daß der Wagen bzw. Kugelkopf oder das Typenrad um einige Schreibstellen vorwärtsbewegt wird. Eine weitere Taste dient zum Umschalten der Tastatur; mit ihr erzeugt man die Großbuchstaben. Man kann diesen Großbuchstaben-Modus bei der Schreibmaschine auch feststellen; dazu gibt es eine eigene Feststelltaste. Solche Tasten, die die Funktionsweise des Geräts beeinflussen, nennt man Funktionstasten.

Nicht anders ist es bei Ihrem C128. Er verfügt über die bereits von der Schreibmaschine her bekannten Funktionstasten und außerdem noch über einige zusätzliche Tasten, deren Bedeutung Sie noch erfahren werden. Die Abbildung 2.3 gibt Ihnen einen Überblick über die Funktionstasten des Computers.

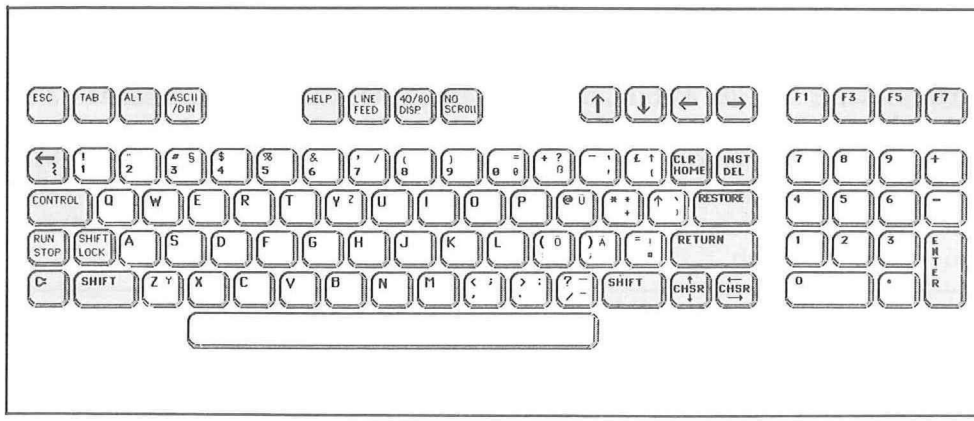


Abbildung 2.3: *Die Funktionstasten*

Es verbleibt noch ein dritter Tastenbereich, der sich bei einer Schreibmaschine nicht findet. Am rechten Rand der Tastatur ist ein sogenannter Zehnerblock abgesetzt, wie er auch auf Rechenmaschinen zu finden ist. Er dient der bequemen Eingabe von Zahlenwerten in den Computer und ist in Abbildung 2.4 dargestellt.

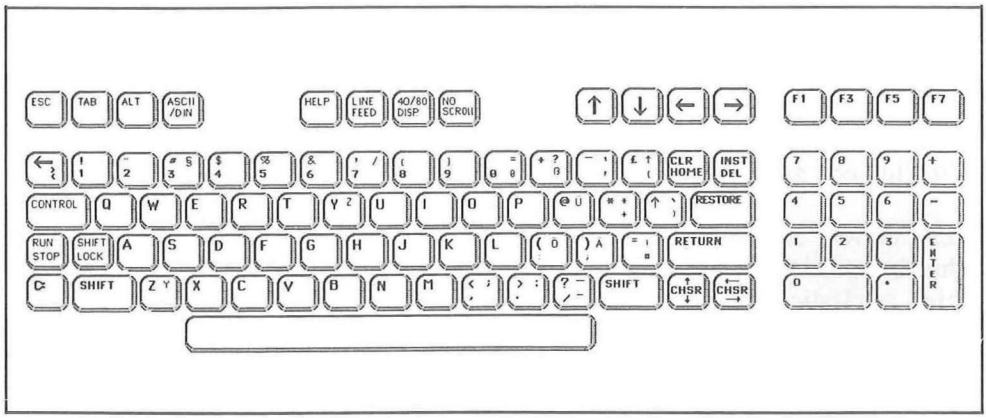


Abb. 2.4: *Der Zehnerblock*

Die Umsteiger vom C64 werden mit einem großen Teil der Tastatur bereits vertraut sein. Der in Abbildung 2.5 hervorgehobene Bereich stimmt nämlich mit der Tastatur des C64 überein. Im 64er-Modus haben auch nur diese Tasten eine Wirkung. Die übrigen - einschließlich des Zehnerblocks - sind nur für den 128er- und CP/M-Modus von Bedeutung.

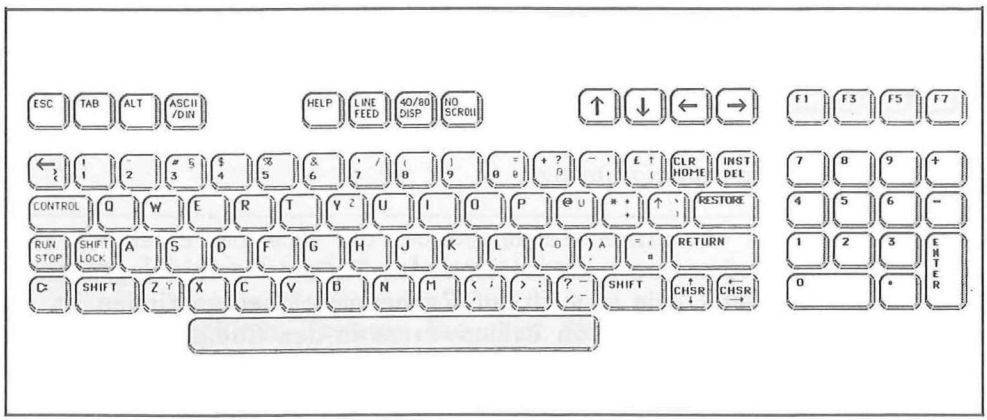


Abbildung 2.5: *Die Tastatur für den 64-er Modus*

Eigentlich befinden sich auf dem C128 also mehrere Tastaturen; weswegen jeder davon ein eigenes Unterkapitel gewidmet wird.

2.1.1 Die Schreibmaschinentastatur

Schon der Umstand, daß entgegen der Gewohnheit die Tasten der Schreibmaschinen-Tastatur mit bis zu vier unterschiedlichen Beschriftungen versehen sind - betrachten Sie nur einmal die der Zifferntasten! - läßt vermuten, daß diese Tastatur einige Besonderheiten aufweist. Wir wollen erst mal herausfinden, was es mit den Mehrfach-Beschriftungen auf der Oberseite auf sich hat.

Solche Mehrfach-Beschriftungen gibt's ja auch bei der Schreibmaschine. Sie wissen, daß die oberste Tastenreihe doppelt belegt ist. Einfaches Betätigen einer Taste erzeugt das untere der beiden aufgemalten Zeichen - eine Ziffer. Um das obere Zeichen zu erhalten müssen Sie auf der Schreibmaschine die Feststelltaste betätigen (eine Funktionstaste).

Nicht anders ist es beim C128. Die Feststelltaste ist hier, wie auf amerikanischen Schreibmaschinen üblich, mit SHIFT beschriftet. Drücken Sie SHIFT zusammen mit einer Zifferntaste - oberste Reihe der Schreibmaschinen-Tastatur - so erhalten Sie das betreffende Sonderzeichen.

Aber beim C128 sind nicht nur die Zifferntatsen mehrfach beschriftet; auch bei einigen Buchstabentasten geht es auf der Oberseite ganz schön eng zu! Nehmen Sie nur die Taste rechts neben dem "L": da finden Sie eine öffnende eckige Klammer, aber auch den Doppelpunkt und ein - etwas schattiert gemaltes - großes "Ö".

Dies hängt damit zusammen, daß der C128 über mehrere eingebaute Zeichensätze verfügt. Beim Einschalten des Gerätes - durch Betätigen des Netzschalters an der rechten Seite des Gehäuses - befinden Sie sich im Commodore-spezifischen amerikanischen Zeichensatz (versichern Sie sich aber, daß die Funktionstaste ASCII/DIN in der obersten linken Tastenreihe nicht gedrückt ist!). Bei Betätigen einer normalen Buchstaben-Taste erscheint auf dem Bildschirm des Computers der entsprechende Buchstabe, allerdings in Großschrift. Eine "normale" Buchstabentaste ist eine, deren Oberseite nur einfach beschriftet ist, also z.B. die Taste "T" oder "P".

Bei der "Klammer-Doppelpunkt-Ö-Taste" haben wir es hingegen mit einer nicht-normalen (verrückten?) Taste zu tun. Betätigen Sie diese Taste, so erscheint auf dem Bildschirm der Doppelpunkt. So ist es auch mit den anderen Tasten: Auf Druck erscheint der Buchstaben bzw. das Zeichen, das auf der Tastenoberseite fett und schwarz aufgebracht ist und das unten steht. Machen Sie noch einen Test: in der Mitte der oberen Buchstaben-Reihe findet sich noch so eine "verrückte" Taste. Hier thront ein fettes "Y" neben einem kleineren bläßlichen "Z". Auf Tastendruck erscheint das "Y": die Fetten kommen vor den Blassen!

Des Rätsels Lösung: Beim Einschalten des Computers herrschen auf der Tastatur ziemlich genau die Verhältnisse, die man auf einer amerikanischen Schreibmaschine vorfindet. Ich bezeichne diesen Zustand mal als den "amerikanischen Großbuchstaben-Modus": Die Buchstaben "Y" und "Z" sind vertauscht und die deutschen Umlaute "Ä", "Ö" und "Ü" gibt es nicht; außerdem erscheinen nur Großbuchstaben. Durch "Umschalten" ändert sich da auch nichts. Wenn Sie eine der beiden SHIFT-Tasten zusammen mit einer Buchstaben-Taste - z.B. dem Buchstaben "S" - betätigen, dann erhalten Sie keineswegs einen Kleinbuchstaben. Vielmehr erscheint auf dem Bildschirm ein seltsames Gebilde, das Sie mit keiner Schreibmaschine der Welt hervorrufen können: ein kleines Herzchen.

Es handelt sich dabei um ein Grafikzeichen. Das sind die Bestandteile, aus denen sich bei vielen Videospielen oder anderen Programmen, die mit Bildern arbeiten, die Illustrationen zusammensetzen. Eine typische Eigenheit des C128 und seines Vorgängers, des C64 ist es, auf Knopfdruck eben solche Grafikzeichen erzeugen zu können. Sowas ist oft sehr nützlich; Sie werden diese Möglichkeit übrigens bei Personal Computern - gleich welcher Preis- und Leistungsklasse - vergeblich suchen.

Betrachten Sie die "S"-Taste noch einmal genauer. Sie werden feststellen, daß sie auch auf ihrer Vorderseite beschriftet ist, und zwar zweimal, wobei die rechte Beschriftung eben jenes gerade erzeugte Herzchen ist. Links daneben findet sich aber noch ein weiteres Grafikzeichen. Um dieses zu erhalten, müssen Sie die entsprechende Taste zusammen mit der "Commodore"-Taste C= betätigen. Im Falle der "S"-Taste erhalten Sie also ein kleines rechtwinkliges "Knie".

Mit der amerikanischen Großbuchstaben-Tastatur können Sie also die Zeichen der Schreibmaschinentastatur erreichen, die entweder fett auf der Oberseite aufgedruckt sind, oder die Sie an der Stirnseite der Taste sehen. Für den Umgang mit dieser Teil-Tastatur gelten folgende Regeln:

1. Bei Betätigung einer Buchstabentaste erscheint der Großbuchstabe auf der Tasten-Oberseite am Bildschirm.

2. Bei Betätiung einer Buchstabentaste zusammen mit SHIFT erscheint das rechte Grafikzeichen auf der Tasten-Vorderseite.

3. Bei Betätiung einer Buchstabentaste zusammen mit C= erscheint das linke Grafikzeichen auf der Tasten-Vorderseite.

Etwas anders ist es bei den Zifferntasten (der obersten Tastenleiste der Schreibmaschinen-Tastatur). Normaler Tastendruck erzeugt die Ziffer (bzw.

eines der Sonderzeichen "Plus", "Minus" oder "Pfund"). SHIFT zusamen mit einer Zifferntaste erzeugt das Sonderzeichen in der oberen Reihe - wie dies auch bei einer Schreibmaschine üblich ist. Was aber ist mit den Aufschriften auf der Tasten-Vorderseite? Man sieht hier keine Grafikzeichen, sondern abgekürzte Farbnamen.

Drücken Sie doch einmal die C=-Taste zusammen mit der "3": es erscheint kein Zeichen, stattdessen ändert der Cursor (das blinkende Rechteck, das Ihnen anzeigt, wo das nächste Zeichen auf dem Bildschrim erscheint) seine Farbe! Wenn Sie jetzt irgendeinen Text schreiben, dann erscheint dieser nicht im gewohnten Weiß auf dem Bildschrim. Vielmehr sind die Buchstaben allesamt rosa gefärbt (das sehen Sie natürlich nur, wenn Sie einen Farbbildschirm an den Computer angeschlossen haben).

Mit den Zifferntasten kann man also die Cursor- und damit die Zeichenfarbe ändern. Es sind aber zwei Farbkürzel auf jeder Zifferntaste; wie erreichen Sie die zweite Farbe?

Dazu braucht's noch eine weitere Funktionstaste. Diese ist mit CONTROL beschriftet. CONTROL zusammen mit einer Zifferntaste aus der Zifernleiste wählt die zweite Farbe aus. Es kommen also noch ein paar Regeln hinzu:

4. Bei Betätigen einer Zifferntaste erscheint die Ziffer (untere Reihe der Tasten-Beschriftung) auf dem Bildschirm.

5. Bei Betätigung einer Zifferntaste zusammen mit SHIFT erscheint das Sonderzeichen (obere Reihe der Tasten-Beschriftung) auf dem Bildschirm.

6. Bei Betätigung einer Zifferntaste zusamen mit C= ändert der Cursor die Farbe. Die Farbe ergibt sich aus der unteren der beiden Aufschriften auf der Tastenvorderseite.

7. Bei Betätigung einer Zifferntaste zusammen mit CONTROL ändert der Cursor die Farbe. Die Farbe ergibt sich aus der oberen der beiden Aufschriften auf der Tastenvorderseite.

Keine Regel ohne Ausnahme! Hier sind es die Tasten "9" und "0", die querschießen. Was die Oberseite betrifft, so halten sich die beiden ja noch an unsere Regeln. Aber die Vorderseite! Hier hat C= keine Wirkung, wohl aber die CONTROL-Taste. Betätigt man CONTROL zusammen mit "9", dann ändert sich erst mal gar nichts - bloß der Cursor hält im Blinken inne, als ob er einmal durchschnaufen müßte. Wenn man dann aber irgend etwas schreibt, dann erscheinen die Buchstaben revers auf dem Bildschirm:

Vordergrund- und Hintergrundfarbe sind vertauscht! Sie sehen also z.B. nicht ein weißes Zeichen auf dunklem Grund, sondern ein dunkles Zeichen vor weißem Grund (Die genauen Farbverhältnisse hängen natürlich von der eingestellten Zeichenfarbe ab).

Wie aber kommt man zurück aus der inversen Darstellung? Ganz einfach: Mit der "0"-Taste zusammen mit CONTROL.

Mit der amerikanischen Großbuchstaben-Tastatur wären wir durch. Kompliziert, meinen Sie? Sie haben recht! Aber ein Computer verhält sich zu einer Schreibmaschine nun mal wie ein Düsenjäger zu einem Fahrrad: man kann wesentlich mehr damit machen, dafür sind aber auch wesentlich mehr Knöpfe dran!

Wenn es eine amerikanische Großbuchstaben-Tastatur gibt, dann sollte es wohl auch eine amerikanische Kleinbuchstaben-Tastatur geben. Und siehe da: es gibt sie. Schreiben Sie mal irgendeinen Text auf den Bildschirm, zum Beispiel Ihren Namen. Und jetzt drücken Sie die beiden Tasten SHIFT und C= (zusammen betätigen!): alle Großbuchstaben werden klein.

Dieser Kleinbuchstaben-Modus erinnert viel stärker an eine Schreibmaschine als der andere. Auf Tastendruck erscheinen Kleinbuchstaben. Mit SHIFT kann man auf Großbuchstaben umschalten. Die Ziffernleiste funktioniert wie erwartet. Und die Grafik-Zeichen?

Versuchen Sie es mal mit der C=-Taste: die funktioniert. Sie können also an die linken Grafikzeichen ran; hier verhält sich der Kleinbuchstaben-Modus wie der Großbuchstaben-Modus.

Aber die rechten Grafikzeichen sind futsch! CONTROL-A produziert kein Herzchen mehr, sondern gar nichts. Daran müssen Sie sich im Kleinbuchstaben-Modus eben gewöhnen: die rechten Grafikzeichen sind nicht mehr erreichbar. Sie verschwinden auch vom Bildschirm, wenn Sie sie erst im Großbuchstaben-Modus erzeugen und dann mit SHIFT-C= umschalten.

Das war's zur amerikanischen Tastatur. Schwant Ihnen schon was? Ja, hier ist Sie: die deutsche Tastatur!

2.1.2 Die deutsche Tastatur.

Eine Eigenschaft, die den C128 vor allen anderen Konkurrenten auszeichnet, werden besonders die unter Ihnen zu schätzen wissen, die einen geschäftlichen Einsatz des Gerätes planen. Der Computer verfügt über den

deutschen Zeichensatz und über eine (deutsche) DIN-Tastatur. Das war ja eines der Probleme mit dem C64: nur mit umständlichen Verrenkungen konnte man ihn dazu gebrauchen, deutsche Textverarbeitung zu machen. Die Tastatur mußte neu belegt, der Zeichensatz umdefiniert werden - alles ziemlich unbequem. Diese Zeiten sind vorbei. Versteht sich, daß die folgenden Ausführungen nur für den 128er-Modus gelten. Da der 64er-Modus voll kompatibel zum C64 sein muß, steht diese Möglichkeit hier nicht zur Verfügung.

Eine Funktionstaste außerhalb der Schreibmaschinen-Tastatur trägt die Aufschrift ASCII/DIN. Die Taste rastet ein; wird sie gedrückt, dann ändert sich das Aussehen der Bildschirm-Darstellung. Die Zeichen nehmen eine andere Form an (wer bereits einen Computer der CBM-Reihe benutzte, dem wird sie vertraut sein). Aber das ist noch nicht alles: drücken Sie mal auf die Taste mit dem (grauen) "Ü". Sie sehen: nach dem Umschalten bekommen Sie das Zeichen mit der schattierten Darstellung. Ist eine Tastenkappe sowohl mit einem fetten als auch mit einem schattierten Aufdruck versehen, dann können Sie in diesem Modus nur die schattierten Zeichen auf den Bildschirm bekommen. Die Fetten treten jetzt in den Hintergrund.

Bei mehrfacher schattierter Beschriftung funktioniert die Umschaltung mit SHIFT; die Taste mit dem (schattierten) "+"-Zeichen und ihre rechte Nachbarin sind hierfür Beispiele.

Sonst ist aber alles gleichgeblieben: Grafikzeichen sind nach wie vor über SHIFT und C= erreichbar. Die Ziffernleiste wird mit SHIFT umgeschaltet, die Farben über C= und CONTROL aus der Ziffernleiste ausgewählt und die reverse Darstellung wie gewohnt ein- und wieder ausgeschaltet. Auch einen Kleinbuchstaben-Modus gibt es, der auch hier mittels SHIFT-C= eingeschaltet werden kann. Dieser Modus ist für den geschäftlichen Anwender der wichtigste. Er erlaubt die Darstellung des gesamten deutschen Zeichensatzes einschließlich der Umlaute in Groß-und Kleinschrift. Daß man da wieder an einen Teil der Grafikzeichen nicht herankommt, läßt sich wohl verschmerzen.

2.1.3 Die 64er-Tastatur

Jetzt, wo Sie mit den wichtigsten Betriebsarten Ihrer Tastatur vertraut sind läßt sich auch sagen, welche Tasten im 64er-Modus wirksam sind. Diese 64er-Tastatur ist in Abb. 2.5 dargestellt; sie ist identisch mit der amerikanischen Tastatur. Großbuchstaben, Grafikzeichen und Farbumschaltung sind also damit möglich. Auch können Sie mit SHIFT-C= in den

Kleinschrift-Modus umschalten. Eine Umschaltung in den deutschen Zeichensatz ist jedoch nicht möglich, da die dazu nötige ASCII/DIN-Taste nicht mehr auf der 64er-Tastatur befindlich ist. Mit anderen Worten: nur das, was von der 64er-Tastatur aus erreichbar ist, ist im 64er-Modus erlaubt. Da man zur Umschaltung in den deutschen Zeichensatz aber die 64er-Tastatur verlassen muß, geht dieses Umschalten auch nicht im 64er-Modus, so wie es auch auf dem C64 nicht möglich ist (Sie wissen schon: die Kompatibilität!). Die schattierten Aufschriften auf den Tasten-Oberseiten haben im 64er-Modus also keine Bedeutung.

Da sind jedoch noch einige Tasten auf der 64er-Tastatur, die sich nicht auf einer Schreibmaschine befinden und die bisher noch nicht erklärt wurden. Es sind dies Funktionstasten. Da solchen Tasten bei Computern große Bedeutung zukommt, wird ihnen auch ein eigenes Kapitel gewidmet.

2.1.4 Die Funktionstasten

Bei den Funktionstasten ist der Punkt erreicht, wo sich der Computer gravierend von einer Schreibmaschine zu unterscheiden beginnt. Auf der Tastatur getippte Zeichen erscheinen auf dem Bildschirm; wo sie erscheinen, bestimmt ein leuchtendes blinkendes Rechteck, die Schreibmarke oder - im Computerchinesisch - der Cursor. Erste Computer-Besonderheit: Diesen Cursor kann man auf dem Bildschirm frei bewegen. Dazu gibt es eine Anzahl Tasten, die sinnigerweise Cursor-Steuertasten genannt werden. Sie finden diese Tasten auf der 64er-Tastatur in der rechten unteren Ecke (Aufschrift: CRSR und jeweils zwei Pfeile); außerdem gibt es noch einen eigenen Block mit vier Tasten in der obersten Funktionstasten-Leiste.

Die horizontale Bewegung des Cursors erfolgt auf der 64er-Tastatur mit der äußersten rechten Taste. Ein Tastendruck, und der Cursor wandert nach rechts, ohne eine Spur zu hinterlassen. Die Taste ist mit einer Wiederholfunktion ausgestattet: lassen Sie den Finger drauf, und der Cursor flitzt über den Bildschirm. Stößt er an den rechten Bildschirm-Rand, so hüpft er an den Anfang der rechten Zeile. Die gleiche Funktion bewirkt die Taste mit dem Rechtspfeil in der obersten Tastenleiste.

Die umgekehrte Richtung bekommen Sie mit SHIFT. Der Cursor geht nach links; falls er an den linken Rand stößt, hüpft er eine Zeile nach oben, aber an das Ende dieser Zeile.. Auch für diese Cursor-Bewegung gibt es eine eigene Taste in der obersten Tastenleiste, die natürlich mit einem Linkspfeil beschriftet ist.

Die linke Nachbarin auf der 64er-Tastatur ist für vertikale Bewegungen. Sie bewegt den Cursor nach oben oder unten. Auch dazu gibt es zwei Geschwister in der obersten Leiste mit der gleichen Funktion.

Strenggenommen hätte Commodore auf die doppelte Ausführung dieser Tasten verzichten können. Es hat sich jedoch - besonders bei Home Computern - eingebürgert, für jede Cursor-Bewegung eine eigene Steuertaste zur Verfügung zu stellen. Auch kann man sich vorstellen, daß dies für manche Action-Spiele vorteilhaft ist. Allerdings sind die Einzeltasten, da sie nicht in der 64er-Tastatur liegen, im 64er-Modus ohne Bedeutung. Man wird sehen, wie die Software-Hersteller diese Tasten für Programme im 128er-Modus nutzen werden.

Mit den Cursor-Steuertasten können Sie also die Schreibmarke frei über den Bildschirm bewegen und so z.B. einen vorhandenen Text überschreiben und korrigieren. Die senkrechte Bewegung hat übrigens noch eine Besonderheit. An den unteren Bildschirm-Rand angelangt, kann man den Cursor weiter 'nach unten' bewegen, mit dem Effekt, daß dadurch der gesamte Bildschirm-Inhalt nach oben verschoben wird. Der Text wandert über den Bildschirmrand hinaus (und ist verloren; Sie können ihn nicht mehr herholen). Dieses Phänomen nennt man "rollen" oder fachmännisch "scrollen" (vom Englischen "to scroll"; dt. "rollen"). Das Scrollen stellt sich auch ein, wenn Sie in der rechten unteren Bildschirmecke versuchen, den Cursor weiter nach rechts zu bewegen. Wir werden darüber später noch etwas zu sagen haben.

So recht zur Geltung kommt die Tastatur erst, wenn man zusätzlich zur Cursor-Bewegung noch zwei weitere Funktionstasten beherrscht: die CLR/HOME und die INST/DEL-Taste rechts oben auf der 64er-Tastatur.

Mit INST/DEL können Sie ein Zeichen auf dem Bildschirm löschen oder eines Einfügen. Zum Löschen gehen Sie mit dem Cursor hinter das zu löschende Zeichen und drücken auf INST/DEL. Voilà: das Zeichen verschwindet! Zum Einfügen gehen Sie vor das Zeichen, vor dem Sie einfügen wollen und drücken SHIFT zusammen mit INST/DEL. Es entsteht eine Lücke, wobei wunderbarerweise alle Zeichen hinter der Lücke um eine Position nach rechts gewandert sind. Nichts ist verlorengegangen und Sie können das gewünschte Zeichen einfügen.

Sie sehen: was jetzt besprochen wurde, geht weit über die Möglichkeiten einer Schreibmaschine hinaus. Wenn man einen Text in dieser Art - mit Cursor-Steuerung, Einfügen und Löschen - bearbeitet, dann spricht man übrigens von "edieren". Edieren ist nützlich bei der Textverarbeitung, aber auch beim Programmieren. Selbsterstellte BASIC-Programme, aber auch

Programme in Maschinensprache können nämlich so bequem bearbeitet, erweitert und korrigiert werden.

Die Taste CLR/HOME schickt den Cursor in die linke obere Ecke. Zusammen mit SHIFT bewirkt sie ein Großreinemachen: der gesamte Bildschirm wird gelöscht. Das ist nützlich, wenn es mal gar zu wild aussieht auf dem Bildschirm und man die Übersicht verloren hat.

Drei Funktionstasten auf der 64er-Tastatur sind noch erklärungsbedürftig. Am einfachsten ist's mit der SHIFT-LOCK-Taste. Sie entspricht der Feststelltaste für Großschreibung bei Schreibmaschinen. Sie rastet ein und sorgt dafür, daß die SHIFT-Taste als dauernd gedrückt gilt. Welche Auswirkung das in den verschiedenen Modi hat (amerikanisch oder deutsch, groß oder klein), wissen Sie ja jetzt.

Die RUN/STOP-Taste ist nur im Zusammenhang mit BASIC-Programmen bedeutsam. Mit ihr kann man ein laufendes BASIC-Programm anhalten. Ein solcherart unterbrochenes Programm kann jedoch mit CONT (s. Anhang A) wieder fortgesetzt werden. Die Wirkung zusammen mit SHIFT ist im 128er-Modus anders als im 64er-Modus. Im 128er-Modus wird nach der Tastenfolge SHIFT-RUN/STOP versucht, ein Programm von Diskette zu laden und zu starten. Im 64er-Modus möchte der Computer hingegen von der Datasette laden. Die Taste RUN/STOP ist mit einer ganzen Zeichenfolge belegt. Diese Eigenschaft teilt sie mit einigen anderen Tasten, zu denen wir gleich kommen.

Bleibt noch die RESTORE-Taste. Wird Sie zusammen mit RUN/STOP gedrückt, dann kehrt der Computer in den Grundzustand zurück: ein eventuell laufendes Programm wird abgebrochen. Der Bildschirm wird gelöscht und die freundliche Meldung "READY" erscheint. Anders als bei der RUN/STOP-Unterbrechung können Sie hier Ihr Programm aber nicht mehr mit CONT fortsetzen - wohl aber mit dem BASIC-Befehl RUN erneut starten.

Belegbare Funktionstasten

Eine belegbare Funktionstaste gibt bei Betätigung nicht ein einzelnes Zeichen aus, sondern eine ganze Zeichenfolge: ein Wort, einen Satz oder was Sie wollen. Ein entfernter Verwandter dieser Tasten-Familie ist uns schon begegnet: RUN/STOP erzeugt mit SHIFT eine Ladebefehl mit anschließendem RUN-Befehl. An dem Text, den diese Taste ausgibt, kann der Benutzer allerdings nichts ändern.

Die belegbaren Funktionstasten (Beschriftung: F1/F2 bis F7/F8) sind da anders. Sie können selbst bestimmen, was diese Tasten auf Druck von sich geben sollen - allerdings gilt dies nur im 128er-Modus! Bei Einschalten des Geräts sind die Tasten bereits mit einigen nützlichen Kommandos vorbelegt. Über die genaue Belegung können Sie sich auch mit dem BASIC-Kommando KEY informieren. Dieses dient außerdem auch dazu, eigene Belegungen vorzunehmen. Die Tasten selbst sind doppelt belegt: die Funktion 1 erhält man über einfachen Tastendruck, die Funktion 2 nur zusammen mit SHIFT. Hier die Funktionen im Einzelnen:

1 GRAPHIC-Befehl (s. Anhang A)
2 Ein Programm wird von Diskette geladen; Sie müssen noch den Programmnamen nachliefern (s. DLOAD-Befehl im Anhang)
3 Zeigt das Inhaltsverzeichnis der Diskette an
4 Löscht den Bildschirm - auch den Grafikbildschirm
5 Ein Programm wird auf Diskette gespeichert; Sie müssen noch den Programmnamen nachliefern (s. DSAVE-Befehl im Anhang)
6 Läßt das BASIC-Programm im Speicher laufen
7 Listet das BASIC-Programm im Speicher auf
8 Schaltet den Monitor ein.

Falls Sie Anfänger sind, dann werden Sie mit vielen der Funktionen jetzt noch nichts anfangen können. Die dazu nötigen Informationen kommen jedoch in späteren Kapiteln.

Obwohl diese Funktionstasten noch in der 64er-Tastatur liegen, verhalten Sie sich doch im 64er-Modus anders. Hier sind sie nicht mit Befehlen belegt, diese Möglichkeit existiert dort garnicht! Man kann die Tasten lediglich aus einem selbstgeschriebenem Programm heraus abfragen. Glückliche C128-Besitzer!

Bleiben noch zwei Tastenblocks zu erklären. Am linken Rand der Funktionstasten-Leiste findet sich neben der Umschalttaste für den deutschen Zeichensatz, die Sie schon kennen, eine Tabulatortaste (TAB). Bei ihrer Betätigung hüpft der Cursor um 8 Bildschirm-Positionen weiter. Zusammen mit SHIFT kann man sich aber auch eigene Tabulator-Positionen definieren. Diese Tastenkombination vereinbart an der Bildschirmspalte eine Tabulatorposition, an der der Cursor gerade steht.

Die ALT-Taste rechts davon hat keine Funktion. Sie wird aber sicher von Programmen genutzt werden, da sie es dem Programmierer erlaubt, die Tastatur mehrfach mit Funktionen zu belegen - noch mehr, als es sowieso schon der Fall ist!

Die HELP-Taste löst den BASIC-Befehl HELP aus; mehr darüber finden Sie im Anhang A. Da diese Taste von Programmen abgefragt werden kann, darf man davon ausgehen, daß eine Menge benutzerfreundlicher Programme für den C128 auf den Markt kommen werden.

Die Taste LINE FEED erzeugt einen Zeilenvorschub. Programmierer werden das zu schätzen wissen. Für den Benutzer hat das momentan noch weniger Bedeutung.

Die Taste rechts neben LINE FEED schaltet auf den 80-Zeichen-Bildschirm um. Diesem ist ein eigenes Unterkapitel gewidmet.

NO SCROLL hält die Bildschirmausgabe an. Beim Betrachten längerer Programmlistings, die nicht auf eine Bildschirmseite passen, ist das eine äußerst schätzenswerte Eigenschaft. Durch erneutes Drücken dieser oder einer beliebigen anderen Taste wird die Anzeige fortgesetzt.

Haben Sie's gemerkt? Eine Taste - ESC; ganz oben links - habe ich mir bis zuletzt aufgehoben. Die erweitert nämlich die Edier-Möglichkeiten Ihres C128 ganz beträchtlich. Und wird deswegen in einem eigenen Kapitel besprochen.

2.1.5 Der Bildschirm-Editor

Schon die bisher besprochenen Funktionstasten erlauben es, einen auf dem Bildschirm befindlichen Text bequem zu bearbeiten. Das werden Sie beim Programmieren bald zu schätzen wissen; aber auch andere Arten der Textverarbeitung profitieren davon.

Aber Sie werden bald feststellen, daß es noch bequemer geht. Zum Beispiel muß man zum Einfügen mehrere Zeichen mit den bisherigen Möglichkeiten erst eine genügend große Lücke mit INST/DEL schaffen, mit den Cursor-Tasten an deren Anfang zurückkehren und dann den Einfügetext schreiben.

Es gibt jedoch eine bessere Möglichkeit: drücken Sie einmal auf die ominöse ESC-Taste und dann auf das "A". Wenn Sie jetzt an irgendeiner Stelle des Bildschirms etwas schreiben, dann wird dieser Text vor einem bereits bestehenden eingefügt. Oder stört Sie das hysterische Geblinke des Cursors? Kein Problem: drücken Sie ESC und anschließend E. Der Zappelphilipp rührt sich nicht mehr.

Was wir da gerade ausprobiert haben ist der Bildschirm-Editor. Ihn bedienen Sie, indem Sie die ESC-Taste drücken und danach eine Zeichentaste betätigen. Man spricht in diesem Zusammenhang auch von "Escape-Sequenzen" (ESC ist die Abkürzung für "Escape", welches wörtlich "entkommen, flüchten" bedeutet). Es folgt eine Übersicht über alle Escape-Sequenzen und ihre Bedeutung für den Bildschirm-Editor. Wenn Sie jetzt noch nicht alles verstehen sollten, so merken Sie sich doch, daß Sie später, wenn Sie Experte geworden sind, an dieser Stelle nachschlagen können.

ESC-A: Schaltet den Einfügemodus ein.

ESC-B: Setzt die rechte untere Ecke des Bildschirmfensters auf die aktuelle Cursor-Position. Die Definition von Bildschirmfenstern erfolgt analog zum WINDOW-Befehl von BASIC (s. Anhang A) und hat auch die gleiche Wirkung. Sie müssen also unter Umständen noch die linke obere Ecke setzen (s. ESC-T). Ein einmal gesetztes Bildschirmfenster kann jedoch im Bildschirm-Editor nicht mehr gelöscht werden, da Sie die Fenstergrenzen mit dem Cursor nicht mehr verlassen können. Da hilft dann nur mehr ein Rücksetzen mit RUN/STOP+RESTORE.

ESC-C: Schaltet den Einfügemodus wieder aus (s. ESC-A).

ESC-D: Löscht die Zeile, in der sich der Cursor befindet.

ESC-E: Friert den Cursor ein, so daß dieser nicht mehr blinkt.

ESC-F: Schaltet das Cursor-Blinken wieder ein (s. ESC-E).

ESC-G: Der Signalton wird wieder zugelassen (s. ESC-H).

ESC-H: Unter bestimmten Umständen (z.B. bei der Arbeit mit dem Monitor) wird ein Signalton erzeugt. Mit dieser Funktion können Sie das unterdrücken. Der Signalton kann dann auch nicht mehr mit CONTROL-G hervorgerufen werden.

ESC-I: Fügt eine Leerzeile an der Cursorposition ein.

ESC-J: Setzt den Cursor an den Anfang der aktuellen Zeile.

ESC-K: Setzt den Cursor an das Ende der aktuellen Zeile.

ESC-L: Das Bildschirm-Scrollen wird wieder eingeschaltet (s. ESC-M).

ESC-M: Das Bildschirm-Scrollen wird ausgeschaltet. Versuchen Sie manuell (mittels LIST) oder per Programm (mittels PRINT) mehr Zeilen auszugeben, als auf den Bildschirm (oder in das Fenster) passen, dann wird nicht etwa der Bildschirm nach oben gerollt und immer in der untersten Bildschirmzeile ausgegeben. Vielmehr beginnt die Ausgabe nach der letzten Bildschirm- (oder Fenster-)zeile mit der ersten Zeile.

ESC-N: Schaltet die normale Bildschirmdarstellung ein. Dieser Befehl hat nur auf dem 80-Zeichen-Bildschirm Wirkung (s. ESC-R).

ESC-O: Schaltet den Einfügemodus, den Quotemodus und die inverse Zeichendarstellung aus.

ESC-P: Löscht in der aktuellen Zeile vom Cursor bis zum Zeilenanfang.

ESC-Q: Löscht in der aktuellen Zeile vom Cursor bis zum Zeilenende.

ESC-R: Schaltet auf dem 80-Zeichen-Bildschirm die inverse Darstellung ein: der Hintergrund wird hell, die Zeichen dunkel (je nach gewählter Farbe).

ESC-S: Schaltet im 80-Zeichen-Bildschirm den normalen Block-Cursor wieder ein (s. ESC-U).

ESC-T: Setzt die linke obere Ecke des Bildschirmfensters. Wenn Sie dies unterlassen und nur mit ESC-B den unteren Fensterrand definieren, dann wird die Positon 0,0 (linke obere Bildschirmecke) als Fenster-Anfang benutzt.

ESC-U: Damit können Sie auf dem 80-Zeichen-Modus statt des blinkenden Quadrats (Block-Cursor) einen Unterstrich zum Cursor machen.

ESC-V: Rollt den Bildschirm um eine Zeile nach oben.

ESC-W: Rollt den Bildschirm um eine Zeile nach unten.

ESC-X: Schaltet zwischen dem 40- und 80-Zeichen-Bildschirm um. Um diese Möglichkeit zu nutzen, müssen Sie mit zwei Monitoren zugleich arbeiten!

ESC-Y: Setzt die normalen Tabulatoren (falls Sie diese überdefiniert haben; alle 8 Spalten).

ESC-Z: Löscht alle Tabulatoren.

ESC-@: Löscht vom Cursor bis zum Bildschirm- (oder Fenster-)ende.

2.1.6 Der Quote-Modus

Beim Durchlesen der Möglichkeiten des Bildschirms ist Ihnen vielleicht unter "ESC-O" ein neuer Ausdruck aufgestoßen: der "Quote-Modus". Dies ist wieder eine besondere Commodore-Spezialität.

Gehen Sie doch mal in eine freie Bildschirmzeile und tippen Sie ein Anführungszeichen, etwa so: " - und jetzt versuchen Sie mal, mit dem Cursor nach links zu fahren. Ja, von wegen! Nichts geht mehr. Es erscheint ein inverses Grafikzeichen. Auch die anderen Steuertasten für den Cursor haben nicht die gewünschte Wirkung sondern produzieren plötzlich Zeichen. Jetzt tippen Sie noch ein Anführungszeichen und probieren erneut die Cursorsteuerung. Plötzlich funktioniert sie wieder.

Die Programmierer unter uns wissen, daß in BASIC mit dem Anführungszeichen ein String eingeleitet wird. Der C128 (und ebenso sein Vorgänger, der C64) verfügt über die einmalige Möglichkeit, in diese Strings direkt Steuerzeichen aufzunehmen. Dies kann man in Programmen oft gut gebrauchen, ist aber bei anderen Homecomputern nur mit Mühe zu erreichen. Der Commodore-Programmierer hat's da einfach: wenn er einen String 'öffnet' (indem er ein Anführungszeichen tippt), dann geht das System automatisch in den sog. "Quote-Modus". Alle - oder fast alle - Funktionstasten werden

wirkungslos. Stattdessen wird die betreffende Funktion zu einem Teil des Strings. Dies geht solange weiter, bis Sie den String durch ein weiteres Anführungszeichen abschließen, oder den String durch RETURN beenden. Über einen abgeschlossenen String können Sie sich übrigens ohne Einschränkungen mit den Cursor-Tasten bewegen; Sie brauchen nicht zu befürchten, daß Ihr Cursor mitten im String 'eingefangen' wird.

Eine Funktionstaste behält aber ihre Wirkung bei: mit INST/DEL können Sie das letzte getippte Zeichen löschen. Sie können auch den ganzen String bis einschließlich zum öffnenden Anführungszeichen 'von hinten auffressen'. Jedoch: den Quote-Modus beenden Sie dadurch nicht. Die einzige Möglichkeit besteht darin, ein zweites Anführungszeichen zu tippen.

Es sei denn, Sie benutzen den Editor. ESC-O ist eine bequeme Möglichkeit zum Beenden des Quote-Modus. Sie ist jedoch nur im 128er-Modus verfügbar. Im 64er-Modus gibt es den Bildschirm-Editor nicht. Das dazu nötige ESC befindet sich ja auch gar nicht auf der 64er-Tastatur.

2.2 Die Peripherie

Jetzt haben wir der Oberseite Ihres Computers aber genug Aufmerksamkeit gewidmet! Es wird Zeit, sich der vielfältigen Fauna von Löchern in der Gehäuseseite zuzuwenden, die der Eingeweihte lässig als "Peripherie-Anschlüsse" bezeichnet.

Peripherie läßt sich in drei Klassen einteilen: die absolut notwendige, der angenehme Luxus und das, worüber sich streiten läßt. Absolut notwendig ist das, was der Computer braucht, um mit Ihnen in Verbindung zu treten, also ein Bildschirm. Zum angenehmen Luxus zählen Drucker und Floppys, aber auch Joysticks. In die dritte Kategorie fällt die computergesteuerte Temperaturüberwachung für das Wasser im Goldfischglas. Allen ist gemeinsam: sie müssen über spezielle Anschlüsse mit dem Computer verbunden werden. Diese Anschlüsse wollen wir jetzt mal genauer untersuchen.

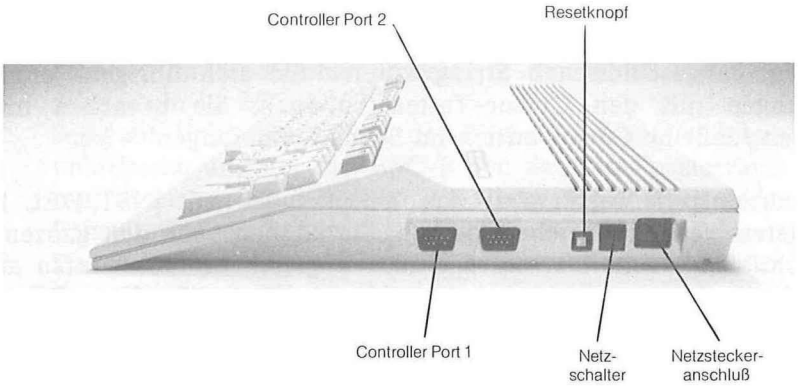


Abbildung 2.6: *Seitenansicht des C128*

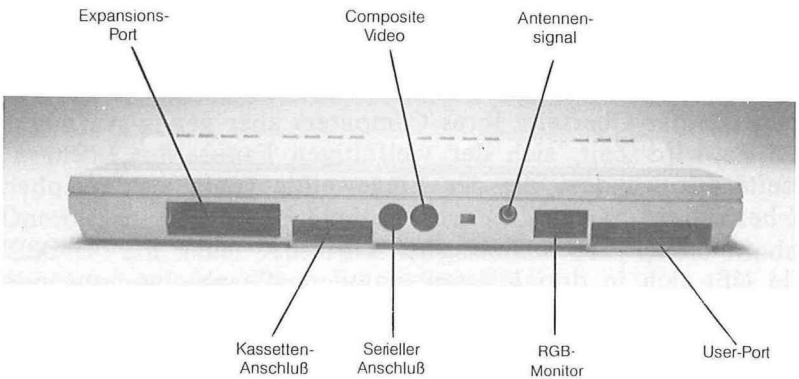


Abbildung 2.7: *Rückansicht des C128*

2.2.1 Bildschirme und Monitore

Die Anschlüsse für die Bildschirme befinden sich an der Rückseite des C128. Anschlüsse? Bildschirme? Warum in der Mehrzahl? Betrachten Sie Abbildung 2.7.

Da sehen Sie drei Anschlüsse (oder "Buchsen"), die mit "Antennensignal", "Composite Video" und "RGB Monitor" beschriftet sind. An jede Buchse können Sie einen Monitor anschließen. Das bedeutet nicht, daß Sie mit

Ihrem Gerät drei Programme auf einmal empfangen können (obwohl Sie, wie sich noch zeigen wird, 'in Stereo' fernsehen können). Es bedeutet nur, daß Commodore versucht hat, für jeden Geldbeutel eine Lösung zu finden.

Ein Bildschirm gehört nämlich nicht zum Lieferumfang des C128; dies ist bei Homecomputern aus Preisgründen generell der Fall. Diese Geräte bieten daher auch allesamt die Möglichkeit, ein handelsübliches Fernsehgerät als Bildschirm zu betreiben. Dieses wird an der Buchse angeschlossen, die mit "Antennensignal" beschriftet ist.

Na, dann ist ja alles in Ordnung. Fernsehen hat ja jeder; warum dann noch zusätzlich zwei weitere Anschlüsse? Die Antwort: weil die Fernseh-Lösung zwar die kostengünstigste, aber nicht unbedingt die beste ist. Dazu eine kleine Einführung in die Technik, mit der Bilder dargestellt und übertragen werden:

Ihr Computer setzt die Bilder auf dem Schirm aus einer Unzahl winziger Punkte zusammen. Für jeden dieser Punkte merkt sich der Computer, in welcher Farbe und mit welcher Helligkeit er zu leuchten hat. Die Farb-Information stellt er zusammengesetzt dar, nämlich als Anteile an den Grundfarben Rot, Grün und Blau. Es sind also zur vollständigen Beschreibung eines dieser vielen Bildpunkte vier Informationen nötig: Rot-, Grün-, Blauanteil und Intensität. Diese Informationen muß der Computer dem Bildschirm übermitteln. Da der mit seinem eingebauten Lautsprecher auch noch die Tonerzeugung übernimmt (für Sound-Effekte), muß auch noch ein Audio-Signal rübergeschickt werden.

Setzen Sie ein Fernsehgerät ein, dann passiert - bildhaft - mit diesen fünf Informationen (vier für das Bild, eine für den Ton) ungefähr folgendes: sie werden allesamt in einen großen Topf gesteckt und dann kräftig geschüttelt. Aus diesem Farb-, Intensitäts- und Tonmischmasch muß sich Ihr Fernsehgerät dann wieder die richtigen Informationen zusammenklauben. Das klappt nicht immer einwandfrei - die Bildqualität ist entsprechend.

Mit anderen Worten: ein Fernsehgerät ist es gewohnt, als Antennensignal eine modulierte Trägerfrequenz zu empfangen, in der alle gewünschten Informationen enthalten sind. Bild und Ton werden zu einem Signal zusammengefaßt (in einen gemeinsamen Topf geschüttet), welches dann der Trägerfrequenz aufmoduliert wird (entspricht kräftigem Schütteln dieses Topfes). Das Fernsehgerät demoduliert das Antennensignal und entschlüsselt dann die übermittelte Information, aber eben leider nicht perfekt.

Einige moderne Fernsehgeräte, aber auch spezielle Video-Monitore können zur Erzeugung eines besseren Bildes dienen. Es gibt nämlich noch einen weiteren Videoausgang, den "Composite Video"-Anschluß. An diesem Aus-

gang steht ein zusammengesetztes Bildsignal zur Verfügung, allerdings ohne Modulation und ohne Toninformation. Diese wird auf einer speziellen Leitung übertragen. Sie können daran einen passenden Monitor anschließen und erhalten ein besseres Bild als mit dem Fernsehgerät.

Unter allen Homecomputer-Herstellern ist Commodore der einzige, der für diesen Ausgang eine weitere Lösung bereithält. Die Firma stellt einen spezielen Monitor her, der die Typenbezeichnung "1702" trägt und als "Direct Monitor" bekannt ist. Er wird ebenfalls in den Composite-Video-Eingang eingestöpselt, spaltet das Signal jedoch in drei Bestandteile auf: eine Leitung für Farbe (von Commodore "Chroma" genannt), eine für Intensität ("Luma") und eine für den Ton ("Audio"). Der Direct Monitor ist von den preisgünstigen Lösungen wohl die beste. Wer mag, kann den 1702 übrigens auch über den Composite-Video-Anschluß betreiben; an seiner Vorderseite ist dafür eine Buchse enthalten.

Jetzt aber zur optimalen Lösung, die jedoch auch ein größeres Loch in den Geldbeutel reißt: der RGB-Monitor. Am besten ist es natürlich, alle fünf Informationen auf je einer eigenen Leitung zu übertragen; unmoduliert, versteht sich. Genau das geschieht beim RGB-Monitor. An den dafür vorgesehenen Anschluß können Sie ein solches Gerät anschließen. Es bietet ein Optimum an Bildqualität. Gerade diejenigen unter Ihnen, die viel mit dem 80-Zeichen-Bildschirm arbeiten, sollten sich überlegen, ob sich die - zugegebenermaßen höhere - Investition für sie nicht auszahlt. Auch Commodore bietet ein passendes Gerät mit der Typbezeichnung "1902" an. Ein RGB-Monitor kann sowohl im 128er-als auch im CP/M-Modus betrieben werden.

Es geht bei allen vier Möglichkeiten auch etwas billiger. Anstatt einen Farbfernseher oder -monitor zu kaufen, kann man auch ein Schwarzweiß-Gerät benutzen (bei den Monitoren sagt man da vornehm "monochromer Monitor"). Hier ist meistens die Bildschärfe höher als bei der Farbversion. Auf die Farben muß man natürlich verzichten; man sieht stattdessen Grauabstufungen (oder Grünabstufungen, wenn man einen Phosphor-Monitor benutzt). Aber die weniger verspielten Naturen werden das in Kauf nehmen.

Jetzt die gute Nachricht für die Stereo-Fans: der RGB-Anschluß und der Antennen- bzw. Composite-Anschluß können gemeinsam betrieben werden! Sie können also zwei Bildschirme an Ihren C128 hängen. Das geht allerdings nur im 128er-Modus, hat da aber einige schöne Vorteile. Es ist so z.B. möglich, auf einem Composite Monitor farbige Grafiken darzustellen, und auf dem 80-Zeichen-Bildschirm erläuternden Text dazu auszugeben (auch die umgekehrte Möglichkeit läßt sich realisieren; mehr darüber in

einem späteren Kapitel). Man stelle sich bloß mal vor, welche neue Dimensionen sich dadurch für Adventure-Spiele eröffnen!

Ehe Sie sich jetzt zum nächsten Fernseh-Händler begeben: betrachten Sie noch mal in aller Ruhe die folgende Abbildung, in der die vier technischen Möglichkeiten nochmal illustriert werden.

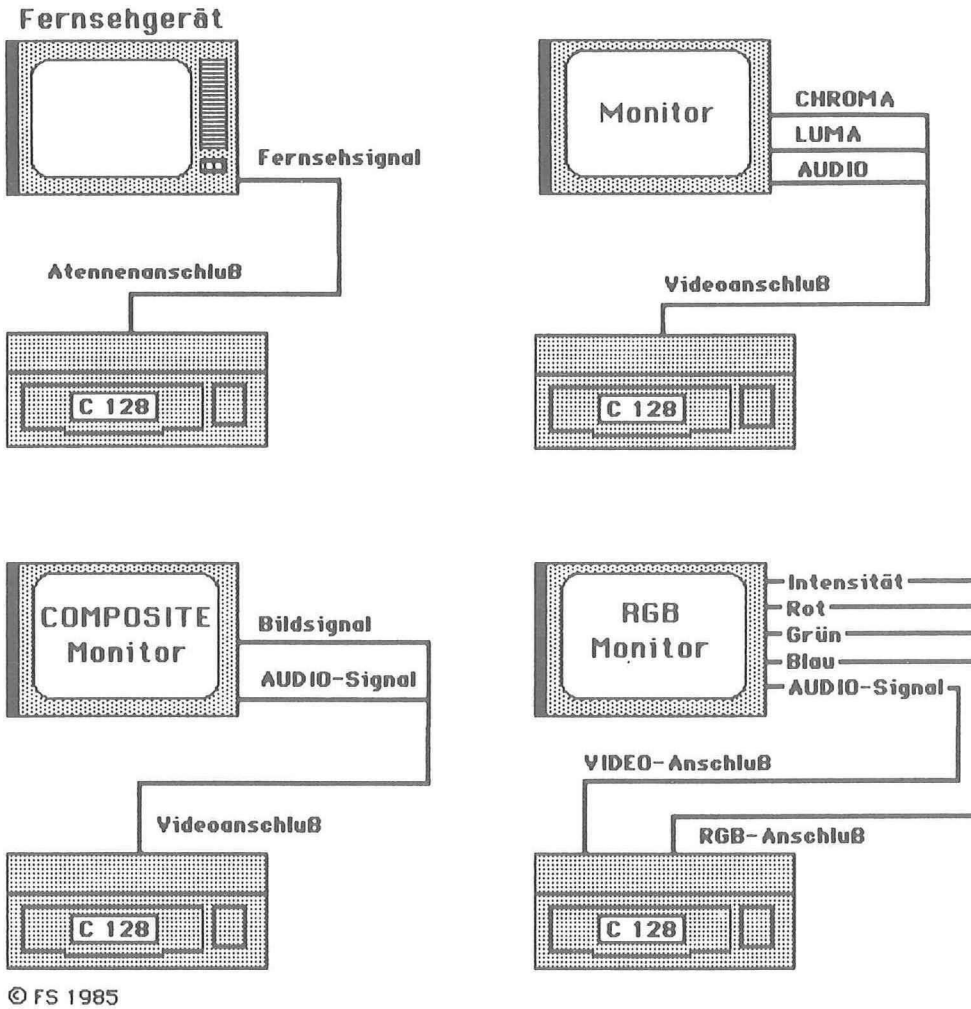


Abb 2.8: *Die verschiedenen Bildschirm-Anschlüsse*

2.2.2 Massenspeicher

Nach dem Anschluß eines Bildschirms oder Monitors ist Ihr Computer-System komplett in dem Sinne, daß Sie damit arbeiten können. Sie können z.B. eigene Programme in BASIC entwickeln oder mit dem Monitor das System erforschen. Die meisten Beispiele in diesem Buch können Sie also jetzt schon ausprobieren. Aber die jetzige Geräte-Zusammenstellung hat doch einen gravierenden Nachteil. Programme, ohne die Ihr Rechner ja nicht viel wert ist, gelangen nur durch Eintippen in die Maschinen. Ihre selbstentwickelten Programme müssen Sie auch jedesmal neu eintippen; denn nach Abschalten der Betriebsspannung sind sie verschwunden.

Dies kommt daher, weil der Computer in der vorliegenden Konfiguration - der "Minimalkonfiguration" - über keinen dauerhaften Speicher verfügt. Bloß der Arbeitsspeicher ist vorhanden; davon haben Sie zwar reichlich (128K), aber es handelt sich eben um einen flüchtigen Speicher.

Nichtflüchtige Speicher, die ihre Informationen auch nach dem Abschalten behalten, nennt man auch "Massenspeicher". Die bei Benutzern von Homecomputern verbreitetsten Massenspeicher sind Kassetten-Recorder und Disketten.

Der Anschluß für einen Kassettenrecorder befindet sich auf der Geräte-Rückseite. Hier müssen Sie mit einer Commodore-Besonderheit leben. Die meisten Hersteller von Homecomputern erlauben den Betrieb eines handelsüblichen Kassetten-Geräts mit ihren Rechnern. Anders Commodore: hier tut's nur ein eigens entwickeltes Gerät, die sog. "Datasette", welche an den dafür vorgesehenen Port angeschlossen wird.

Die Datasette ist billig, was sie attraktiv macht. Sie hat jedoch einen großen Nachteil: die Arbeit mit ihr ist umständlich und sie ist langsam. Während man das umständliche Hantieren noch in Kauf nehmen könnte, so wird es bei der Geschwindigkeit schon kritischer. Gute Programme sind meist umfangreich. Entsprechend lang dauert es, sie von Datasette zu laden oder sie dort abzuspeichern. Außerdem müssen Sie über Ihre Kassetten selbst buchführen: wollen Sie mehrere Programme oder Dateien auf Kassette speichern, so müssen Sie sich selbst merken, an welcher Stelle des Bandes diese jeweils beginnen. Man kann auch den Computer selbst suchen lassen (s. LOAD-Kommando in Anhang A), aber das dauert!

Eine bessere Lösung, und für den, der im CP/M-Modus arbeiten will ein absolutes Muß, ist die Floppy. Sie wird an den seriellen Port an der Rückseite des Gehäuses angeschlossen. Die überwiegende Mehrzahl der kommerziell vertriebenen Programme sind auf Diskette zu erhalten. Der Umgang mit Disketten ist wesentlich einfacher und die Speichergeschwindig-

keit wesentlich größer als bei Kassetten. Da lohnt die etwas höhere Investition schon. Allerdings sollten Sie für Ihren 128er gleich die neue Floppy 1571 erwerben, und nicht ihre für den C64 gedachte Vorgängerin, die 1541. Im 64er-Modus tut's die 1541 zwar auch, aber für den 128er-Modus ist sie einfach zu langsam und hat zu wenig Speicherkapazität. CP/M ist mit diesem Gerät fast unzumutbar. Die eingeschworenen CP/M-Benutzer sollten sich überlegen, ob sie nicht gleich ein Doppellaufwerk (Typbezeichnung 1572) erwerben, da CP/M erst mit zwei Laufwerken seine volle Leistung entfaltet.

2.2.3 Die Daisy Chain

Wer die Anschlüsse am C128 bereits genauer untersucht hat, der wird vielleicht einen Drucker-Anschluß vermißt haben. Drucker gehören - wie die Massenspeicher - zwar nicht zum absolut nötigen Minimum, machen jedoch das Leben leichter. Für den geschäftlichen Einsatz sind sie unerläßlich; da möchte man ja unter anderem seine Geschäftskorrespondenz per Textsystem vom C128 erledigen lassen, oder Berichte und Statistiken ausdrucken. Also: ein Drucker sollte schon her!

Aber wohin damit? Auch das ist eine Commodore-Spezialität: der Drucker hängt an derselben Leitung wie das Floppy-Laufwerk. Das bedeutet aber nun nicht: entweder Floppy oder Drucker. Vielmehr sind Floppy und Drucker auf dieser Leitung hintereinandergeschaltet. Dieses Hintanstellen ist nicht auf zwei Geräte beschränkt: Sie können auch noch ein weiteres Floppy-Laufwerk und/oder einen weiteren Drucker (z.B. einen Schönschreibdrucker) in diese Kette einreihen.

Die Kettenbildung ermöglicht eine spezielle Technik, die man auch als "daisy chaining" bezeichnet, weil hier die einzelnen Geräte wie Blumen in einer Blumengirlande ("Daisy" heißt "Gänseblümchen") hintereinandergereiht sind. Die Gerätekette heißt entsprechend "Daisy Chain"; daher also die Überschrift über dieses Kapitel.

Daisy chaining setzt bei den Geräten eine gewisse Intelligenz voraus. Der Computer schickt nämlich beim Verkehr mit diesen Peripheriegeräten zusätzlich zu den Daten noch eine Gerätenummer mit auf die Daisy Chain. Beide - Nachricht und Nummer - passieren nacheinander die Geräte in der Kette und die müssen schlau genug sein, um an der Gerätenummer zu erkennen, daß sie gemeint sind. In einem BASIC-Programm sorgt übrigens die OPEN-Anweisung dafür, daß der Computer beim Datenverkehr die richtigen Gerätenummern benutzt (s. Anhang A).

2.2.4 Die übrigen Anschlüsse

Was jetzt noch an Anschlüssen übrig ist, ist zumeist für den Luxus und das Kind im Manne gedacht. Da sind zuerst mal die beiden Steuereingänge (Aufschrift: "Controller Port 1" und "Controller Port 2", an der rechten Gehäuseseite. Hier stecken Sie Ihren Joystick, ein Game Paddle oder einen Lichtstift ein. Das sind zugegebenermaßen keine lebensnotwendigen Peripheriegeräte; aber sie versüßen das Leben. Durch die C64-Kompatibilität können Sie auf Ihrem C128 eine Menge Spielsoftware fahren, bei der Joysticks wichtig sind. Achten Sie übrigens darauf, den richtigen der beiden Anschlüsse zu erwischen. Programme, die von Commodore selbst stammen benutzen meist den Port 1, während viele Drittanbieter Port 2 bevorzugen. Bei Spielen mit zwei Spielern können natürlich beide Ports benötigt werden.

Neben diesen beiden Ports finden Sie den Reset-Knopf. Manchmal kann es sein, daß sich in einem selbstgeschriebenen Programm ein Fehler befindet, der den Computer 'lahmlegt': er reagiert auf keine Befehle von der Tastatur mehr. Eine bereits erwähnte Wiederbelebungsmethode besteht darin, mit der Tastenfolge RUN/STOP-RESTORE einen Warmstart zu versuchen. Dies funktioniert jedoch nicht in allen Fällen. Was immer funktionieren würde, wäre, das Gerät ein- und wieder anzuschalten. Ein-und Ausschaltvorgänge sind jedoch ausgesprochene Streß-Situationen für den Computer. Um ihm diese zu ersparen, gibt es den Reset-Knopf. Er versetzt Ihren Computer in den Grundzustand, so daß er wieder gesprächsbereit ist. Allerdings: war vor dem Reset ein Programm im Speicher, dann ist es jetzt weg!

Der Reset-Knopf ist außerdem auch die einzige Möglichkeit, um nach einem Wechsel in den 64er-Modus wieder in den 128er-Modus zu gelangen.

Der rechte Nachbar des Reset-Knopfes ist der Netzschalter. Wozu der gut ist, ist klar. Aber eines noch: gewöhnen Sie sich an, vor dem Einstecken eines Peripheriegeräts den Strom abzuschalten. Dies gilt nicht für Joysticks und anderes Spiel-Zeug am Steuereingang. Andere Geräte aber beziehen ihre Netzspannung vom C128 über die Peripherieanschlüsse. Da ist es sicherer, diese Spannung vor dem Einstecken abzuschalten. Dies gilt ganz besonders für Programm-Cartridges!

Programm- oder Erweiterungs-Cartridges werden an der Rückseite des Gehäuses eingesteckt, im sogenannten "Expansion Port" oder Modul-Steckplatz. Programm-Moduln haben hierzulande keine große Bedeutung als Datenträger; Disketten sind beliebter. Den Software-Herstellern wären sie allerdings lieber, da sich mit diesen Datenträger ein perfekter Kopierschutz erzielen läßt. Der Port hat aber für die Freaks und Hacker große Bedeu-

tung, da der gesamte Systembus des Computers und auch einige wichtige Steuerleitungen an dieser Stelle zur Verfügung stehen. Sie sollten sich jedoch nur dann an die Programmierung dieser Schnittstelle wagen, wenn Sie genau wissen was Sie tun. Es ist nicht ausgeschloßen, daß durch Ihre Fehler der Computer selbst Schaden nimmt!

Noch was: Stecken Sie ein Programm-Modul für den C64 in diesen Modul-Schacht, dann geht der C128 beim Einschalten automatisch in den 64er-Modus.

Bleibt noch der User-Port. Hier handelt es sich um eine wenig verwendete Schnittstelle, die im Gegensatz zum seriellen Anschluß (für die Floppy und die Daisy Chain) einen bitparallelen Verkehr mit der Außenwelt erlaubt. Bei serieller Kommunikation werden die einzelnen Bits, aus denen sich die Information zusamensetzt, nacheinander über eine Leitung gejagt. Die parallele Kommunikation überträgt hingegen 8 Bits zugleich auf 8 getrennten Datenleitungen. An sich gäbe es eine Norm für parallele Schnittstellen - bekannt unter der Bezeichnung RS232 - aber Commodore hält sich bei seinem User Port bedauerlicherweise nicht daran. Es gibt jedoch einen Zusatzbaustein für den C64 zu kaufen, der in den User Port eingesteckt wird und dann daraus eine genormte RS232-Schnittstelle macht. Wenn Sie sowas haben, dann können Sie daran unter anderem auch ein Modem anschließen, um mit Ihrem Computer übers Telefon Daten auszutauschen. Welche Programme und Bausteine im 128er-Modus mit diesem Anschluß arbeiten bleibt abzuwarten.

3 Modi

Es ist Ihnen ja bekannt: der C128 ist drei Computer in einem. Das darf man sich jedoch nicht so vorstellen, daß Commodore drei vollständige Computer in ein Gehäuse gepackt hätte. Dies verbietet sich sowohl aus Platz- wie auch aus Kostengründen. Die Lösung, welche die Firma Commodore eingeschlagen hat, ist auch viel eleganter. Sie hat in ein Gehäuse alle Komponenten gepackt, die die drei 'Computer' benötigen und hat dann das Gesamtsystem so entwickelt, daß die unterschiedlichen 'Computer' möglichst viele dieser Komponenten untereinander teilen können. Ein Beispiel: Computer brauchen Arbeitsspeicher. Wenn nun tatsächlich drei Computer in Ihrem C128 steckten, dann müßte auch dreimal Arbeitsspeicher für jeden der Computer vorhanden sein. Da der C64 mit 64K arbeitet, der C128 mit 128K und CP/M ebenfalls mit 128K, wären wir somit bereits bei 320K angelangt.

So ist es aber nicht. Tastächlich haben die Commodore-Ingenieure 'nur' 128K in Ihr System eingebaut - für einen Homecomputer ohnehin schon eine beachtliche Menge. Diese 128K reichen für jeden der drei eingebauten 'Computer' aus, ja, der C64 benötigt nur die Hälfte davon. Fazit: der Arbeitsspeicher wird geteilt. Folgerung: weil nicht jeder 'Computer' seinen eigenen Arbeitsspeicher hat, können Sie auch nicht gleichzeitig betrieben werden. Es kann immer nur einer drankommen, die anderen beiden haben zu schweigen. Um nicht die irrige Vorstellung aufkommen zu lassen, daß wirklich drei Computer vorhanden sind, die dann aber auch parallel betrieben werden könnten, spricht man bei Commodore eben von den "Modi".

Man befindet sich bei der Arbeit mit dem C128 stets in einem von drei möglichen Modi. Ist man in einem Modus, so treten die anderen beiden Möglichkeiten völlig in den Hintergrund. Man kann jedoch umschalten, einen Modus verlassen und in den nächsten gehen. Die Umschalterei ist aber nicht grenzenlos: einige Wege sind Einbahnstraßen.

Das Konzept des C128 besteht also darin, in einem Gehäuse so viele Komponenten unterzubringen, daß drei verschiedene Modi realisiert werden können und dafür zu sorgen, daß die drei Modi möglichst viel gemeinsam benutzen können - schon um die Herstellungskosten zu drücken.

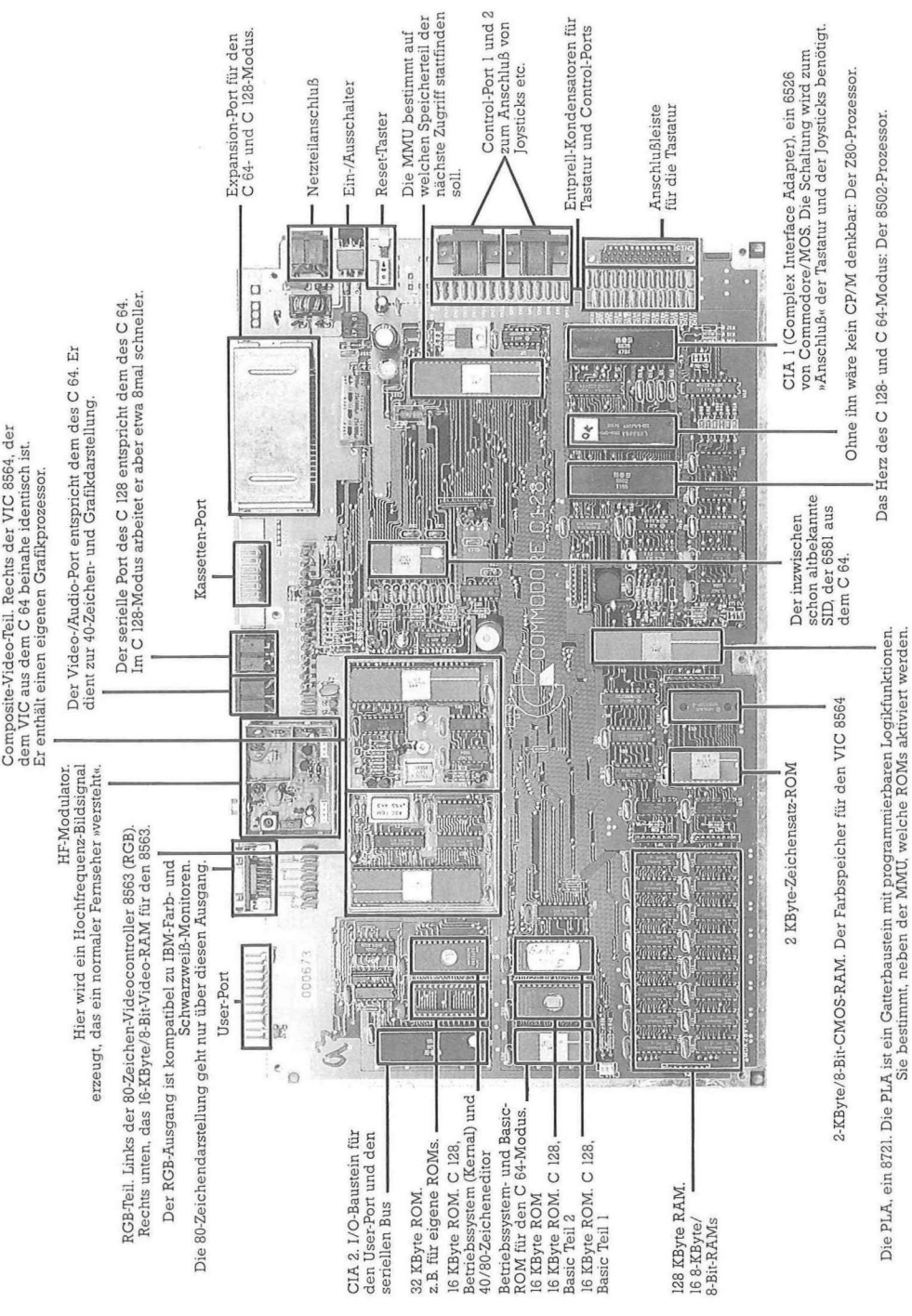


Abbildung 3.1: *Das Innenleben des C128*

Für die weitere Erörterung ist es angebracht, mal einen kurzen Blick in das Innere Ihres Computers zu werfen, um zu sehen, was alles in der Kiste steckt.

Jeder der drei Modi greift sich eine andere Zusammenstellung aus diesem reichlichen Angebot an elektronischen Bauteilen heraus.

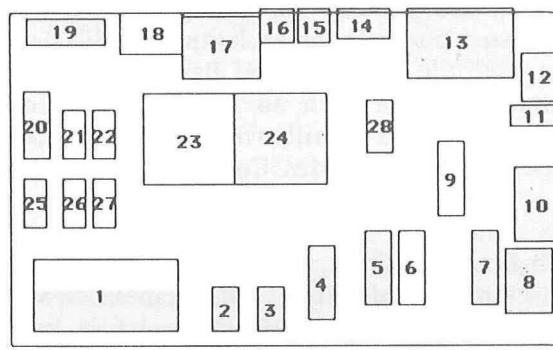
Ein Baustein ist darunter, der diese Zusammenstellung regelt. Es ist die MMU, die sich auf der rechten Hälfte der Platine befindet und über die Sie in späteren Kapiteln noch Ausführliches erfahren werden.

Angesichts des dichten Gedränges auf der Platine ist es nicht sinnvoll, die Baustein-Zusammenstellung für jeden der drei Modi bis in alle Einzelheiten zu schildern. Stattdessen wird umrißhaft angedeutet, was wo zur Verfügung gestellt ist. Die Umrisse beziehen sich aber auf die Abbildung 3.1.

3.1 Der 64er-Modus

3.1.1 Was ist da?

Die folgende Abbildung gibt Ihnen einen Überblick über die Komponenten, die im 64er-Modus aktiv sind; sie sind hier schraffiert dargestellt.



Nr.	Beschriftung	Nr.	Beschriftung	Nr.	Beschriftung
1	RAM	11	Reset-Taste	21	ROM
2	Zeichensatz	12	Netzspannung	22	C 128-Kernal, Editor
3	Farbspeicher	13	Expansion-Port	23	8563 Video-Controller
4	PLA	14	Kassetten-Port	24	VIC
5	8502-Prozessor	15	Serieller Port	25	Basic 2.0 und 64er-Kernal
6	Z80-Prozessor	16	Video/Audio	26	Basic 7.0
7	CIA	17	Antennensignal	27	Basic 7.0
8	Tastatur	18	RGB-Ausgang	28	SID
9	MMU	19	User Port		
10	Control Ports	20	CIA		

Abbildung 3.2: *Die Komponenten im 64er-Modus*

3.1.2 Wie komme ich hin?

Zum Betreten des 64er-Modus gibt es zwei Möglichkeiten. Sie stecken bei ausgeschaltetem Gerät ein Steckmodul für den C64 in den dafür vorgesehenen Expansion Port. Wenn Sie jetzt den Computer einschalten, dann fühlt dieser die Gegenwart des Steckmoduls und geht automatisch in den 64er-Modus.

Die zweite Möglichkeit setzt voraus, daß Sie im 128er-Modus sind. In diesem Fall geben Sie das Kommando GO64 (s. Anhang A), das Sie nach einer kurzen Rückfrage (mit "Y" beantworten) in den 64er-Modus bringt.

Es gibt keine direkte Möglichkeit, um aus dem CP/M-Modus in den 64er-Modus zu gelangen. Sie müssen dazu erst in den 128er-Modus gehen.

3.1.3 Was kann ich da machen?

Hauptsächlich bestehende Programme benutzen. Das Angebot an C64-Software ist schier unerschöpflich, und alles läuft auf dem C128 im 64er-Modus.

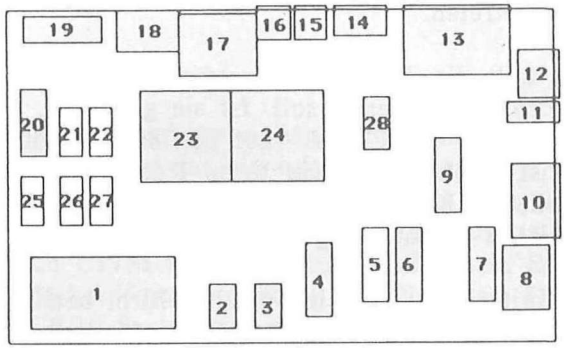
Sie können natürlich auch im 64er-Modus programmieren. Da steht Ihnen aber nur das ältere BASIC 2.0 zur Verfügung, das es an Komfort mit der Version 7.0 bei weitem nicht aufnehmen kann. Wie Sie dem Blockdiagramm entnehmen können, sind die dafür nötigen Bausteine im 64er-Modus ausgeblendet. Um Besonderes vollbringen zu können, z.B. Grafik oder Musik, brauchen Sie in diesem Modus detaillierte Kenntnisse über die Funktionsweise der wesentlichen Bausteine des Computers, besonders von SID und VIC.

Sollten Sie dennoch Lust verspüren, sich im 64er-Modus programmiererisch zu betätigen, dann empfehle ich Ihnen, dafür eines der vielen guten Handbücher zum C64 zu erwerben. Die Maschine ist ja mittlerweile bestens dokumentiert und es ist kein Problem, an passende Literatur heranzukommen. Alle nötigen Details auch hier aufzuführen würde den Rahmen dieses Buches bei weitem übersteigen.

3.2 Der CP/M-Modus

3.2.1 Was ist da?

Diese Frage soll wieder ein Blockdiagramm beantworten. Die aktiven Komponenten sind wie gewohnt schraffiert.



Nr.	Beschriftung	Nr.	Beschriftung	Nr.	Beschriftung
1	RAM	11	Reset-Taste	21	ROM
2	Zeichensatz	12	Netzspannung	22	C 128-Kernal, Editor
3	Farbspeicher	13	Expansion-Port	23	8563 Video-Controller
4	PLA	14	Kassetten-Port	24	VIC
5	8502-Prozessor	15	Serieller Port	25	Basic 2.0 und 64er-Kernal
6	Z80-Prozessor	16	Video/Audio	26	Basic 7.0
7	CIA	17	Antennensignal	27	Basic 7.0
8	Tastatur	18	RGB-Ausgang	28	SID
9	MMU	19	User Port		
10	Control Ports	20	CIA		

Abbildung 3.3: *Die Komponenten im CP/M-Modus*

An dieser Konfiguration fällt auf, daß alle Bausteine, die irgendwie mit dem Betriebssystem und/oder BASIC zu tun haben, ausgeblendet sind. Nun haben Sie aber bereits in der Einleitung erfahren, daß ein Computer ohne Betriebssystem blind und taub ist. Stellt sich also die Frage: Wie kommt's, daß man trotzdem unter CP/M arbeiten kann? Diese Frage beantwortet paradoxerweise das nächste Unterkapitel, in dem Sie erfahren, wie Sie in den CP/M-Modus kommen.

3.2.2 Wie komme ich hin?

...indem Sie in Ihr Laufwerk eine CP/M-Systemdiskette einlegen und den Rechner einschalten. Oder Sie geben, falls Sie sich bereits im 128er-Modus

befinden, bei eingelegter CP/M-Diskette den BOOT-Befehl (s. Anhang A). Außerdem müssen Sie in beiden Fällen noch mitteilen, ob Sie mit einem 40- oder 80-Zeichen-Bildschirm arbeiten wollen.

Erinnern Sie sich noch an das zweite Kapitel, in dem die Tastatur besprochen wurde? Eine Taste ist da ganz stiefmütterlich behandelt worden, nämlich die mit der Aufschrift "40/80 Display" in der obersten Funktionstasten-Leiste. Diese Taste rastet bei Betätigung ein. Sie hat im laufenden Betrieb keine Funktion, ist jedoch wesentlich, wenn Sie den 128er- oder den CP/M-Modus betreten.

Im CP/M-Modus teilt die Taste dem System mit, ob es einen Bildschirm mit 40 oder 80 Zeichen bedienen soll. Ist sie gedrückt, so erfolgen Ausgaben des Systems - also von CP/M - auf den 80-Zeichen-Bildschirm, und Sie sollten tunlichst einen solchen an Ihren Rechner angeschlossen haben. Mehr darüber steht im Kapitel 2.2.1. Wegen der besseren Bildschärfe ist hier übrigens ein RGB-Monitor angeraten.

CP/M kann auch mit einem 40-Zeichen-Bildschirm betrieben werden. Dies ist jedoch sehr unbequem, da CP/M auf die Zeilenbeschränkung keine große Rücksicht nimmt. Das System ist nun mal für das Arbeiten mit 80-Zeichen-Bildschirmen gemacht und geht da keine Kompromisse ein.

Aus alledem folgt, daß die Minimalkonfiguration für den CP/M-Betrieb einen 80-Zeichen-Monitor (vorzugsweise) RGB und ein Diskettenlaufwerk voraussetzt. Warum ein Diskettenlaufwerk? Weil Sie nur in den CP/M-Modus kommen, wenn Sie eine CP/M-Systemdiskette eingelegt haben (siehe oben.) Warum aber ist dies notwendig?

Weil von dieser Diskette das Betriebssystem kommt! Das Blockdiagramm in Abbildung 3.3 macht ja deutlich, daß die Bauteile, die 'normalerweise' das Betriebssystem enthalten, jetzt ausgeblendet sind. In seinem Inneren findet der Computer also nichts mehr vor, was ihm den Verkehr mit der Außenwelt erlaubt; muß er es sich also von außen besorgen.

Dies ist in Übereinstimmung mit der grundlegenden Philosophie von CP/M. Dabei handelt es sich ja um ein Disketten-Betriebssystem; was liegt näher, als das Betriebssystem selbst auf einer Diskette unterzubringen!

Für den 128er-Modus schalten Sie den Rechner einfach ein, und er "ist da". Die MMU blendet dazu nur das Betriebssytem ein, das sie in den eingebauten ROMs vorfindet, und alles nimmt seinen geregelten Gang. Unter CP/M funktioniert es nicht ganz so simpel. Da muß erst einmal das Betriebssystem von der Diskette in den Arbeitsspeicher gebracht werden, ehe Sie irgendetwas tun können. Diesen Vorgang (Laden des Betriebssystems)

nennt man auch "Booten". Das ist der Grund, warum Sie beim Einstieg in den CP/M-Modus - egal, ob durch Einschalten oder aus dem 128er-Modus - die Meldung "BOOTING..." zu sehen bekommen. Es ist auch der Grund, warum das BOOT-Kommando so getauft wurde.

Der Weg zu CP/M ist übrigens auch eine Einbahnstraße. Zurück in den 128er-Modus geht's nur, indem man die Systemdiskette aus dem Laufwerk nimmt und den Reset-Knopf betätigt.

3.2.3 Was kann ich da machen?

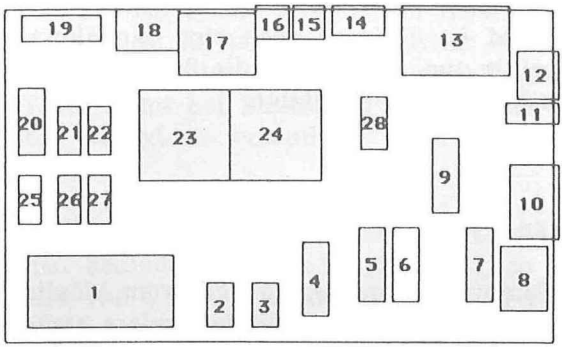
CP/M ist das Computer-Land der unbegrenzten Möglichkeiten. Darüber hat Ihnen ja schon Kapitel Eins das ein oder andere erzählt. Es ist unmöglich, in einem Buch alle Programme zu beschreiben, die unter CP/M laufen und auf dem Markt erhältlich sind. Aber nicht nur die Programme sind beinahe unerschöpflich; auch CP/M kennt viele Möglichkeiten. Es beherrscht viele Kommandos für die komfortable Arbeit mit Disketten. Das in Ihren Computer eingebaute CP/M-System ist außerdem eine beträchtlich verbesserte und erweiterte Version des 'alten' CP/M, die nicht nur mit 128K arbeiten kann, sondern viele zusätzliche Kommandos kennt. Zum Lieferumfang Ihres Systems gehören 28 zum Teil sehr umfangreiche und mächtige Kommandos. Diese hier darzustellen würde aus dem vorliegenden Buch eine mehrbändige Enzyklopädie machen. Deshalb verweise ich Sie auch im Falle CP/M auf die vorhandene Literatur. Da CP/M in hohem Maße standardisiert ist, können Sie jedes Buch, das die Version 3.0 behandelt, als Arbeitsgrundlage verwenden.

Auch besteht kein Zweifel, daß in Bälde eine Vielzahl von Büchern erscheinen werden, die den CP/M-Modus auf dem C128 genau unter die Lupe nehmen. Dies ist deswegen besonders interessant, weil auf Ihrem Rechner unter CP/M Zugang zu den Grafik- und Sound-Bausteinen besteht - ein Unikum für CP/M. Endlich können auch die eingeschworenen Z80-Assemblerfreaks vernünftige Videospiele produzieren...

3.3 Der 128er-Modus

Er ist der Star dieses Buches. Zum einen deswegen, weil er beim Einschalten des Gerätes automatisch da ist. Sie brauchen kein Floppy-Laufwerk, um ihn zu initialisieren. Sie müssen auch nicht erst durch irgendwelche Kommandos in diesen Modus einsteigen. Stattdessen schalten Sie den C128 ein, und schon ist er da!

Der 128er-Modus ist derjenige, der den Chip-Vorrat Ihres Computers am besten ausschöpft. Betrachten Sie nur einmal das Blockdiagramm in Abbildung 3.4.



Nr.	Beschriftung	Nr.	Beschriftung	Nr.	Beschriftung
1	RAM	11	Reset-Taste	21	ROM
2	Zeichensatz	12	Netzspannung	22	C 128-Kernal, Editor
3	Farbspeicher	13	Expansion-Port	23	8563 Video-Controller
4	PLA	14	Kassetten-Port	24	VIC
5	8502-Prozessor	15	Serieller Port	25	Basic 2.0 und 64er-Kernal
6	Z80-Prozessor	16	Video/Audio	26	Basic 7.0
7	CIA	17	Antennensignal	27	Basic 7.0
8	Tastatur	18	RGB-Ausgang	28	SID
9	MMU	19	User Port		
10	Control Ports	20	CIA		

Abbildung 3.4: *Die Komponenten im 128er-Modus*

Der 128er-Modus ist im Wesentlichen eine Verbesserung des C64. Es ist neben dem doppelt so großen Arbeitsspeicher eigentlich nur ein neuer Baustein hinzugekommen: der 8563-Videocontroller, der für die Bedienung des 80-Zeichen-Bildschirms und des RGB-Monitors zuständig ist. Die anderen Verbesserungen - und davon gibt es eine ganze Menge - liegen alle in der Software. Die Software, genauer: die Betriebssoftware, steckt in den ROM-Bausteinen, die sich im linken Teil der Platine befinden. Beim 64er-Modus ist hier nur ein Baustein eingeblendet. Der 128er-Modus braucht hingegen deren drei: einen für das Betriebssystem (C128-Kernal und Editor) und zwei für das eingebaute BASIC. Im C64 sind BASIC und Betriebssystem in einem einzigen Speicherbaustein untergebracht!

Allein dies vermittelt wohl schon einen Eindruck von den gesteigerten Möglichkeiten des C128. Aber gehen wir diese am besten einzeln durch!

3.3.1 Der Speicher

Ein Arbeitsspeicher von 128K bedeutet, daß komplexere und damit leistungsfähigere Software auf dem Computer betrieben werden kann. Auch steht für die Entwicklung eigener Programme, sei es in BASIC oder in Maschinensprache, mehr Platz zur Verfügung. Die Art, wie der C128 seinen Speicher benutzt und verwaltet ist übrigens anders als das bei anderen Systemen, die auch mit 128K ausgestattet sind, der Fall ist. Dies gilt auch für den CP/M-Modus.

Personal Computer und auch CP/M halten das Betriebssystem oder große Teile davon nämlich während des Betriebs andauernd im Arbeitsspeicher. Moderne Betriebssysteme sind jedoch sehr umfangreich. Man kann davon ausgehen, daß Systeme wie CP/M oder das für Personal Computer weit verbreitete MS-DOS ca. 40K an Speicherplatz beanspruchen. Diese 40K werden vom verfügbaren Arbeitsspeicher abgezwackt, so daß für Benutzerprogramme effektiv nur mehr ca. 180K verfügbar sind.

Commodore verfolgt hier eine andere Philosophie. Zu deren Erklärung ist es allerdings nötig, ein wenig technischer zu werden, als dies bisher der Fall war. Sollten Ihnen also die folgenden Ausführungen beim ersten Lesen etwas kryptisch erscheinen, so trösten Sie sich: die Sache *ist* kompliziert und erschließt sich erst bei genauerer Beschäftigung.

Das Betriebssystem wird im C128 in einem eigenen Festwertspeicher (den ROMs) gehalten und nur bei Bedarf ganz oder teilweise in den verfügbaren Arbeitsspeicher eingeblendet. Dieses ein- und ausblenden besorgt im C128 die MMU, der "Speicher-Manager", über die Sie Genaueres in den Kapiteln 5 und 6 erfahren. Für den ambitionierten Programmierer ist es jedoch wichtig, zu wissen, welche ROM-Teile an welchen Stellen des Arbeitsspeichers eingeblendet werden. Dies verdeutlicht man üblicherweise mit sog. Memory Maps, also Speicher-Orientierungsplänen. Die Memory Map für den C128 ist im Folgenden abgebildet. Der Vollständigkeit halber können Sie dieser Übersicht auch noch entnehmen, wie die Speicherbelegung im 64er-Modus aussieht.

Es fällt auf, daß in dieser Zeichnung teilweise mit drei Ebenen gearbeitet wird. Dies kommt daher, daß der C128 in der Lage ist, an der gleichen Stelle im Arbeitsspeicher gleichzeitig (oder beinahe gleichzeitig) unterschiedliche Dinge unterzubringen. Betrachten wir zuerst einen einfachen Fall: Sie bearbeiten ein BASIC-Programm mit dem Bildschirm-Editor. Der Memory-Map können Sie entnehmen, daß der Editor im Bereich zwischen C000 und D000 'lebt'. Das BASIC-Programm aber befindet sich im Textspeicher; dieser geht von den Adressen 4000 bis C000 (Adressangaben macht man üblicherweise im Hexadezimalsystem; dies ist aber für die vor-

liegende Darstellung nicht so wichtig). Man sieht jedenfalls: es gibt keine Überschneidungen. BASIC-Programm und Editor können friedlich koexistieren, also gleichzeitig im Speicher vorhanden sein.

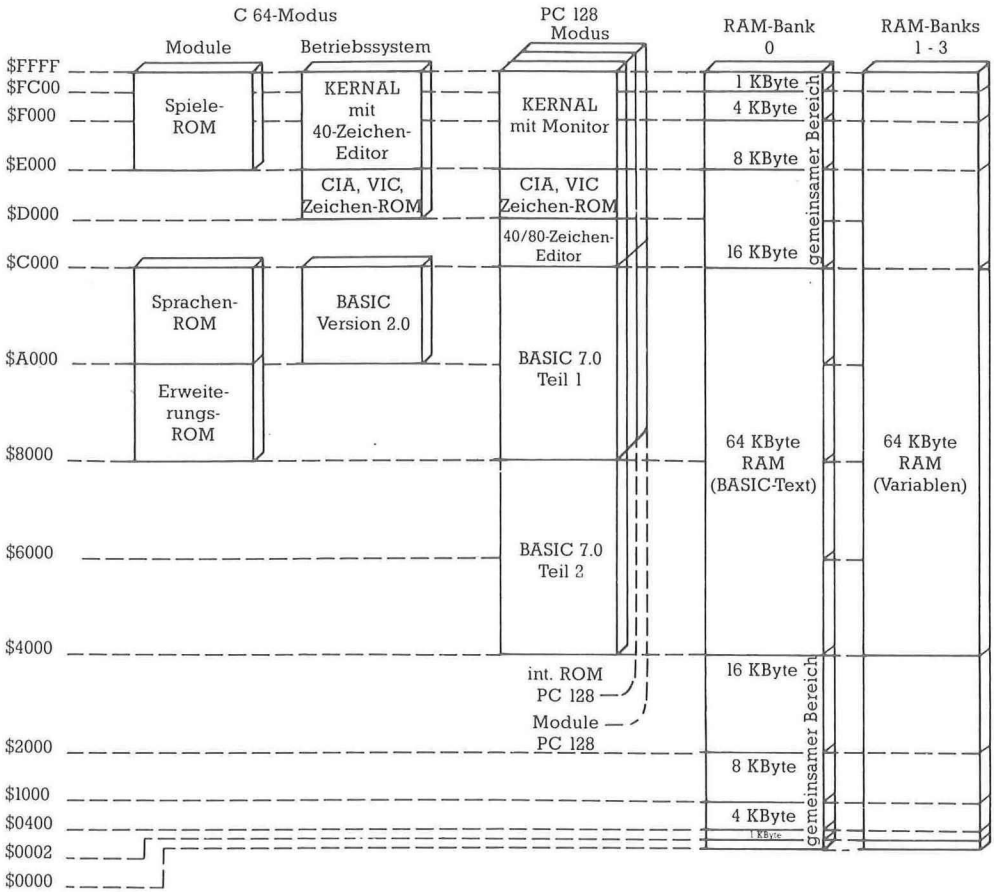


Abbildung 3.5: *Die Memory Map*

Anders ist es, wenn Sie dieses BASIC-Programm jetzt ausführen wollen. Dazu brauchen Sie den BASIC-Interpreter, und der liegt laut Memory Map im Bereich zwischen 4000 und C000 (daß er zweigeteilt ist, wollen wir

momentan außer Acht lassen). In diesem Bereich liegt aber auch Ihr BASIC-Programm! Wie soll das möglich sein?

Hier kommen die mehrfachen Ebenen ins Spiel. Der BASIC-Interpreter lebt zwar auf den selben Adressen wie das Programm, das er ausführt. Er führt dieses Leben aber in einer anderen Ebene als das BASIC-Programm. Man kann sich das so vorstellen, daß bei der Programmausführung ständig zwischen den Ebenen hin- und hergeschalten wird. Zur Ausführung eines Programms sieht der Prozessor erstmal in Ebene A nach, wie der Programmtext lautet. Dann holt er sich Ebene B mit dem BASIC-Interpreter her und übergibt diesem den eben aufgesuchten Programmteil, auf daß dieser ihn ausführen möge. Ist der Interpreter fertig, wird Ebene B verlassen und es geht zurück in Ebene A, also weiter im Programmtext. Natürlich ist der ganze Mechanismus so ausgelegt, daß die Inhalte der einzelnen Ebenen durch die Umschalterei nicht verlorengehen.

Sie meinen, das wäre kompliziert? Ich auch! Dabei ist es sogar stark vereinfacht. In Wahrheit ist es nämlich nicht der Prozessor, der mit den Ebenen rumjongliert, sondern sein Über-Ich, die MMU, die dem Prozessor jeden Wunsch von den Lippen ablesen kann. Sie weiß, ob er jetzt BASIC-Text oder den BASIC-Interpreter sehen muß und schiebt ihm die jeweils passende Ebene unter. Der Fachmann spricht hier übrigens nicht von Ebenen, sondern von "Speicherkonfigurationen".

Die Umschalterei ist damit noch nicht zu Ende. Der Arbeitsspeicher wird nämlich in zwei Bestandteile aufgeteilt, sog. "Arbeitsspeicher-Bänke" (mehr darüber im Kapitel 5.3). Für den BASIC-Programmierer haben diese Bänke folgende Bedeutung. Eine Bank ("Bank 0"; in der EDV wird immer mit Null beginnend gezählt) nimmt das BASIC-Programm selbst auf, und zwar im bereits erwähnten Textspeicher. Die Daten, mit denen Sie im BASIC-Programm arbeiten - also Variablen, Arrays und Strings - packt der C128 jedoch in eine eigene Bank, die Bank 1. Nur so ist es möglich, für BASIC-Programme den sensationellen Speicherplatz von 122365 Bytes verfügbar zu machen.

Das Herstellen einer bestimmten Speicherkonfiguration umfaßt für die MMU also zwei Schritte: Auswählen der richtigen Speicherbank und Einblenden der gewünschten "Ebene", also eines ROMs oder eines Ausschnitts daraus. Mit dem eingebauten Maschinensprachen-Monitor können Sie diesen Prozess auch teilweise selbst steuern; schlagen Sie dazu in den Kapiteln 5.4 und 6 nach. Wie die beiden Speicherbänke im 'normalen' BASIC-Betrieb belegt sind, soll Ihnen noch die folgende Abbildung zeigen.

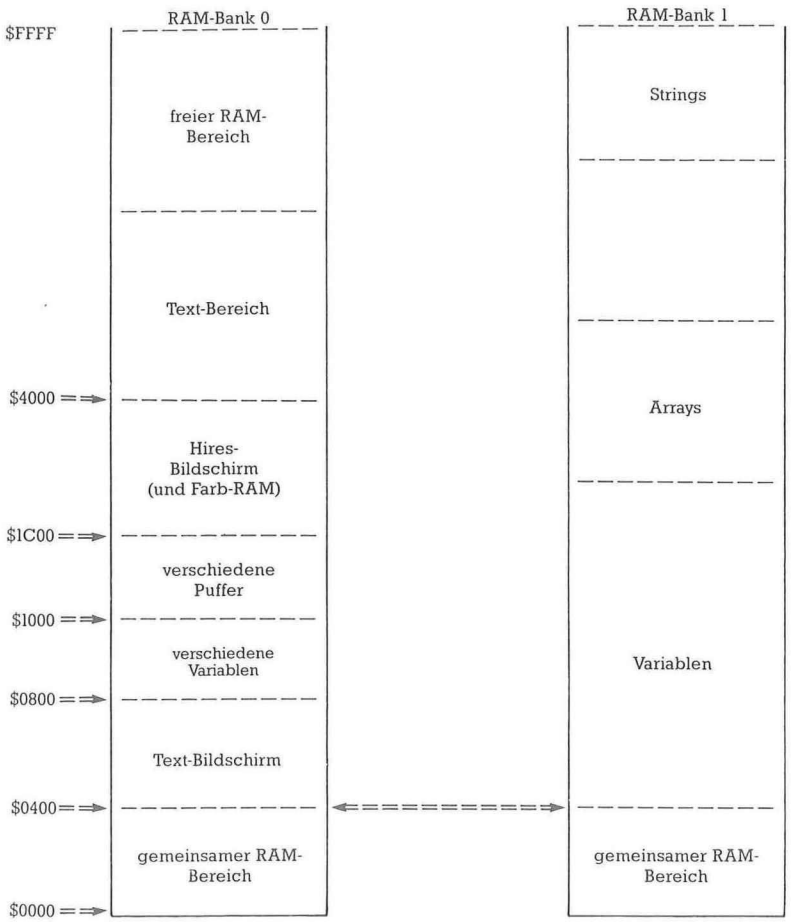


Abbildung 3.6: *Die Aufteilung des BASIC-Speichers*

Der gemeinsame RAM-Bereich im unteren Teil beider Bänke wird gebraucht, um ein reibungsloses Umschalten zwischen den Bänken zu ermöglichen. Ohne ihn würde sich der Computer nämlich selbst den Ast absägen, auf dem er sitzt. Einmal in die Bank 1 gelangt, hätte er keine Möglichkeit, wieder in Bank 0 zurückzufinden.

Der Bereich zwischen 1C00 und 4000 ist übrigens nicht automatisch für das BASIC-Programm tabu, wie man bei Betrachten der Abbildung meinen könnte. Normalerweise beginnen die BASIC-Programme sogar bei 1C00. Wenn Sie jedoch den hochauflösenden Grafikbildschirm benutzen wollen (s.

GRAPHIC-Befehl im Anhang A), dann wird dieser Bereich für die Grafik reserviert und ein eventuell darin befindliches Programm verdrängt, d.h. nach oben an die Anfangsadresse 4000 verschoben. Ein Freigeben des Grafikbereichs läßt das Programm wieder 'nach unten sacken', also wieder ab 1C00 beginnen.

Ein Trost: wenn Sie vorhaben, sich nur mit BASIC zu beschäftigen, dann brauchen Sie von den eben geschilderten Vorgängen, von den verschlungenen Pfaden der Speicherverwaltung und den Speicherkonfigurationen nichts zu wissen. All dies läuft automatisch und ohne Ihr Zutun ab. Solche Kenntnisse werden erst bedeutsam, wenn man selbst in den Eingeweiden der Maschine herumdoktern will. Aber in den Meisten von uns erwacht die Lust dazu früher oder später. Darum habe ich Ihnen diese Ausführungen nicht erspart.

3.3.2 40 vs. 80 Zeichen.

Zurück zu Einfacherem! Das Platinen-Foto in Abbildung 3.1 zeigt bei genauerem Studium, daß zwei Baugruppen zur Bilderzeugung vorhanden sind. Einmal der bereits im ersten Kapitel sattsam besprochene VIC. Diesem benachbart ist ein Baustein mit der Bezeichnung 8563, der sich um die Erzeugung der nötigen Signale für den RGB-Monitor kümmert. Die Dinge liegen so: VIC ist für die Erzeugung eines 40-Zeichen-Textes (und für die hochauflösende Grafik) verantwortlich. Der 8563 kümmert sich um den 80-Zeichen-Bildschirm. Nebenbei: auch mit diesem Baustein ist Grafik-Programierung möglich, allerdings nicht so bequem wie mit dem VIC. Dies gehört zu den mehr verborgenen Möglichkeiten des C128, die jedoch hier ebenfalls schonungslos offengelegt werden; schlagen Sie nach in Kapitel 7.1!

Beide Baugruppen sind im 128er-Modus verfügbar. Das bedeutet, daß Sie hier die Wahl zwischen 40- oder 80-Zeichen-Textdarstellung haben. Wie aber üben Sie Ihr Wahlrecht aus?

Ganz einfach: durch Knöpfedrücken. Wenn Sie Ihren Computer einschalten, und die Taste "40/80 DISPLAY" ist gedrückt, dann legt der C128 seine Bildschirmausgaben auf den 80-Zeichen-Bildschirm. Dieser muß dazu allerdings auch an die RGB-Buchse auf der Gehäuse-Rückseite angeschlossen sein. Andernfalls sehen Sie gar nichts.

Ist die Taste nicht gedrückt, dan wird - vom VIC - der 40-Zeichen-Bildschirm bedient. Das geht entweder über Antennensignal oder über den Composite-Video-Ausgang. Es ist allerdings möglich, im laufenden Betrieb

zwischen den beiden Bildschirmen hin- und herzuschalten. Dazu gibt es die Editor-Funktion ESC-X. Der Editor ist in Kapitel 2.1.5 genauer besprochen. In diesem Zusammenhang soll noch ein Schmankerl für die Programmierer - ob BASIC oder Maschinensprache - erwähnt werden. Nichts hindert Sie daran, in Ihrem Programm die Zeichenfolge ESC-X auszugeben. In BASIC sieht das beispielsweise so aus:

```
PRINT CHR$(27);"X"
```

Das eröffnet ungeahnte Möglichkeiten. Denken Sie mal an ein Adventure-Spiel. Sie schreiben auf den 80-Zeichen-Bildschirm die Spielanleitung. Endlich haben Sie mal genügend Platz, um das vernünftig machen zu können. Dann kommt ESC-X (wie oben dargestellt), und auf dem Grafikbildschirm erscheinen die zauberhaftesten Illustrationen. Wer zum Masochismus neigt, der macht es dann eben umgekehrt: Grafik auf dem 80-Zeichen-Bildschirm, Text vom VIC erzeugt. Beides setzt allerdings voraus, daß zwei Bildschirme gleichzeitig angeschlossen sind. Aber dem wahren Fan ist es das schon wert...

Die Stellung der Taste "40/80 DISPLAY" beim Einschalten bestimmt also, welcher Bildschirm bedient wird. Ein 'weiches' Umschalten geht mit ESC-X.

3.3.3 BASIC 7.0

Der Star des 128er-Modus ist unbestritten das neue BASIC. Schon die Tatsache, daß der Computer dieses in zwei Chips aufbewahrt, während beim C64 ein einziger nicht nur das BASIC, sondern auch das Betriebssystem aufnimmt, zeugt von seiner Mächtigkeit. Ihm wollen wir uns hier genauer widmen.

Zuvor ein Wort zur Warnung: dieses Kapitel ist keine Einführung in BASIC und kann es gar nicht sein. Dafür gibt es eigene Bücher. Es zeigt Ihnen jedoch, was in dem neuen BASIC alles möglich ist. Und für die praktische Arbeit ist der Anhang A da. Alle BASIC-Befehle (insgesamt 140!) finden Sie da alphabetisch aufgelistet, erklärt und mit Beispielen versehen, von denen ich hoffe, daß Sie Ihnen manche Anregung geben können.

Ehe wir uns den Befehlen zuwenden, soll jedoch noch eine Möglichkeit erwähnt werden, die - unabhängig vom Leistungsumfang des BASIC - durch die schiere Speicherfülle besteht. Text- und Variablenspeicher sind nämlich so umfangreich, daß Sie ohne weiteres mehr als ein Programm

darin unterbringen können. Dies nur zur Anregung der kleinen grauen Zellen...

BASIC 7.0 bietet Ihnen verbesserte Mögichkeiten zur Strukturierung Ihrer Programme. Schleifen können übersichtlicher formuliert werden, als mit der FOR-Schleife oder GOTO mit explizitem IF-Test; dies waren die einzigen Möglichkeiten in der Version 2.0. Auch die Tests mit IF ... THEN ... ELSE sind jetzt komfortabler. Sie sind nicht länger auf eine Programmzeile beschränkt, sondern können sich über mehrere Zeilen erstrecken und damit beliebig komplex sein. Für die Testbedingungen innerhalb von IF-Abfragen haben Sie jetzt auch mehr logische Ausdrücke zur Verfügung als zuvor. Die Daten-Ein- und Ausgabe wurde erheblich verbessert und erweitert und ist jetzt viel bequemer zu benutzen. Dasselbe gilt für das Einrichten von Dateien und die Arbeit damit. Schließlich hat Commodore in das neue BASIC Befehle zur komfortablen Programmentwicklung mit aufgenommen, wobei auch vernünftige Testhilfen nicht vergessen wurden.

Das neue BASIC kann selbstverständlich auch den vorhandenen Arbeitsspeicher voll nutzen; die Einschaltmeldung "122365 BASIC BYTES FREE" gibt davon beredte Kunde. Im Umgang mit dem Speicher ist es auch gewitzter geworden; die gefürchtete "Garbage Collection", die so manchen C64-Programmierer wegen ihrer endlosen Wartezeiten zur Verzweiflung trieb, gehört der Vergangenheit an.

Das neue BASIC macht seinem Benutzer das Leben wirklich angenehm; dazu gehören auch die programmierbaren Funktionstasten (die Sie in Kapitel 2.1.4 bereits kennenlernten) und der verbesserte Bildschirm-Editor (bekannt aus Kapitel 2.1.5). Auch ist es viel offenherziger geworden: wer sich in seinem Inneren oder in dem des Betriebssystems umsehen will, der kann mit dem in BASIC verfügbaren MONITOR-Befehl nach Herzenslust alles inspizieren und sogar verändern. BASIC und der Maschinensprachen-Monitor leben in der normalen Speicherkonfiguration friedlich nebeneinander und kommen sich nicht ins Gehege. Wegen seiner Bedeutung für die fortgeschrittene Arbeit ist dem Monitor in diesem Buch ein eigenes Kapitel gewidmet.

Schließlich sei noch einmal erwähnt, daß BASIC im 128er-Modus die neue und schnellere Floppy 1571 unterstützt. Das bedeutet nicht nur, daß es die von ihr gebotenen Übertragungsgeschwindigkeiten ausnutzt. Es sind auch etliche Leistungsfähige Befehle in BASIC mit aufgenommen, die das Arbeiten mit Disketten erleichtern.

BASIC 7.0 besitzt außerdem eine eingebaute Uhr, die alle 1/60 Sekunden hochgezählt wird, und die ganz einfach über die Systemvariable TI$ zu erreichen ist. Diese Uhr ist jedoch nicht als eigener Baustein in der Maschine

vorhanden, sondern sie wird per Software realisiert. C64-Benutzer kennen sie bereits als die gute alte "Jiffy Clock"; im neuen BASIC ist sie aber leichter handzuhaben.

Bei soviel Lob wollen wir einen kleinen Schönheitsfehler nicht verschweigen; er betrifft die Variablen in BASIC. In der Version 7.0 können Sie beliebig lange Variablennamen haben. Sie können also schreiben:

```
10 PETER = 5
```

Aber: zur Unterscheidung nimmt BASIC nur die ersten beiden Zeichen; was in den ersten beiden Zeichen übereinstimmt, ist für BASIC dieselbe Variable! Machen Sie mal weiter mit dem Programm:

```
20 PETRA = 10
30 PRINT PETER,PETRA
```

Wenn Sie dies jetzt mit RUN laufenlassen, dann bekommen Sie nicht 5 und 10 zu sehen, sondern zweimal die 10.

Eine weitere Einschränkung für Variablen-Namen: sie dürfen keines der BASIC-Befehlsworte als Teil enthalten. Geben Sie im Direktmodus ein:

```
ANTON = 99
```

und BASIC meckert mit einem SYNTAX ERROR. Warum? Weil "ANTON" mit "ON" endet - und das ist ein BASIC-Befehl (s. Anhang A)!

Dafür brauchen Sie zwischen den einzelnen Worten in einem Programm keine Leerzeichen zu schreiben. So ist

```
FORI=1TO10:PRINTI:NEXTI
```

völlig in Ordnung, wenn auch nicht besonders angenehm fürs Auge.

Einige Erweiterungen des neuen BASIC sind so innovativ, daß sie verdienen, in eigenen Unterkapiteln abgehandelt zu werden.

3.3.3.1 Programmsteuerung

Was einen Computer meilenweit über einen Taschenrechner erhebt, ist seine Fähigkeit, Arbeitsschritte nicht immer stur hintereinander auszuführen, sondern eigenständige Entscheidungen über das zu treffen, was

er als nächstes vorhat. Der Taschenrechner kriegt von Ihnen eine Rechenaufgabe, löst sie, wartet ergeben auf die nächste, löst sie, und so gehts immer weiter im stumpfsinnigen Trott.

Nicht so ein Computer, der in BASIC programmiert wird. Der kann mal das eine, mal das andere machen, Dinge wiederholt ausführen und all dieses selbst kontrollieren. Die Zauberworte in diesem Zusammenhang sind "Verzweigung", "Schleifen" und "Entscheidungen".

Verzweigungen wurden im BASIC 2.0 mit GOTO, ON GOTO, GOSUB und ON GOSUB realisiert. Da hat sich im neuen BASIC nicht viel verändert. Anders sieht es mit den Schleifen aus.

In BASIC 2.0 hatten Sie zwei Möglichkeiten der Schleifenbildung: die FOR- oder Zählschleife als die komfortablere Möglichkeit und die selbstgebastelte Schleife mit GOTO. Nun gibt es aber die vielfältigsten Schleifentypen: abweisende Schleifen, nicht abweisende Schleifen, Zählschleifen, WHILE-Schleifen, UNTIL-Schleifen und was der Varianten mehr sind. All diese Schleifentypen mußten Sie bisher im BASIC 2.0 mit FOR und/oder GOTO nachbilden.

Die Situation ist ein bißchen so, als hätten Sie zum Reden über die vielfältigen motorisierten Fahrzeug-Typen nur das Wort "Fahrzeug" zur Verfügung. Statt "Motorrad" müßten Sie dann sagen: "zweirädriges Fahrzeug ohne Blech drumrum". Ein "Kabrio" wäre für Sie ein "vierrädriges Fahrzeug ohne Blech obenrum". Und ein Lastwagen? Wie wär's mit "großes vier- oder mehrrädriges Fahrzeug mit überall Blech außenrum aber hinten nix zum Niedersitzen"?

Sie sehen: das geht zwar, ist aber arg umständlich. Ähnlich war es im alten BASIC. Für die vielen Schleifentypen gab es nur zwei Worte. Jetzt haben Sie hingegen den folgenden Wortschatz: FOR...NEXT, DO...LOOP, WHILE, UNTIL und EXIT. Damit lassen sich Programmierprobleme wesentlich knapper und übersichtlicher darstellen. Mehr darüber steht im Anhang A unter den jeweiligen Befehlsworten.

3.3.3.2 Programmentwicklung

Die Programmentwicklung umfaßt zwei Schritte. Erstens: das Schreiben des Programms. Zweitens: das Entfernen der Fehler aus demselben. Nirgendwo wird dem Menschen so deutlich gemacht, daß er nicht unfehlbar ist, wie beim Programmieren. Seien Sie also nicht über sich verzweifelt, wenn es in

Ihren Programmen von Fehlern wimmelt: je ausgebuffter der Profi, desto vertrackter sind die Fehler, die er macht.

Zum Glück unterstützt Sie BASIC 7.0 bei beiden Schritten. Schon beim Eingeben eines Programms brauchen Sie sich um die Zeilennummern nicht mehr zu kümmern; Sie verwenden einfach den AUTO-Befehl. Änderungen, Erweiterungen und Löschungen werden einfach mit den Kommandos RENUMBER und DELETE. Das erste erlaubt es, ein Programm oder Teile davon neu zu numerieren. Mit dem zweiten können Sie einzelne Zeilen oder Zeilenbereiche löschen. Selbstverständlch könen Sie sich auch mit LIST das ganze Programm oder Teile davon auf dem Bildschirm ansehen. Nachträgliche Veränderungen am Programmtext sind mit dem komfortablen Bildschirmeditor leicht möglich.

Möglichkeiten zur Fehlersuche sind überlebensnotwendig. BASIC 2.0 war da äußerst karg; oft konnte man nur durch von Hand eingestreute PRINT-Befehle einem Fehler auf die Schliche kommen. Wieviel anders geht das im neuen BASIC! Hier ist zu Ihrer Bequemlichkeit ein neues Konzept eingeführt worden, nämlich die "Fehlerfallen".

Sie können BASIC anweisen, im Falle eines Fehlers nicht einfach zu streiken und mit einer Fehlermeldung abzubrechen. Stattdessen ist es möglich, bei Auftreten eines Fehlers - ganz gleich, wann und wo im Programm das geschieht - einen speziellen Programmteil anzuspringen, den Sie für die Fehlerbehandlung vorgesehen haben. Dazu gibt es die TRAP-Anweisung.

Im angesprungenen Programmteil (der sog. "Fehler-Routine") können Sie nun den Fehler genau analysieren. Dazu stehen Ihnen spezielle Systemvariablen zur Verfügung, aus denen Sie nicht nur den Fehlertyp, sondern auch die Zeilennummer erfahren können, in der der Fehler auftrat (s. EL, ER, ERR$, DS und ST im Anhang A.2). Wenn Sie sich mit dem Fehler genügend beschäftigt und ihn z.B. durch die Routine beseitigt haben, dann können Sie mit RESUME dafür sorgen, daß das Programm an der Stelle seine Arbeit wieder aufnimmt, an der der Fehler auftrat.

Einigen logischen Programmfehlern ist auf diese Art nicht beizukommen. Man sieht in diesen Fällen zwar, daß das Programm nicht das Gewünschte tut, kann aber nicht sagen, warum das so ist, weil kein BASIC-Fehler dabei auftritt. In diesem Fall ist es nützlich, jeden einzelnen Schritt verfolgen zu können, den BASIC unternimmt. Dies ermöglicht die TRON-Anweisung. Ist Sie gegeben, so informiert Sie BASIC über jede Programmzeile, die es im Zuge seiner Arbeit anläuft. Sie können so leicht erkennen, ob das Program auch wirklich da hingeht, wo es nach Ihrer Vorstellung hingehen soll. Die TRON-Anweisung kann sowohl im Direktmodus, als auch in einem Programm gegeben werden. Sie können sogar ein laufendes Pro-

gramm mit RUN/STOP unterbrechen, im Direktmodus den TRON-Befehl erteilen und dann mit CONT das Programm fortsetzen. Ausgeschaltet wird die Wirkung des TRON-Kommandos mittels TROFF.

3.3.3.3 Ein-/Ausgabe

Die Möglichkeiten, Daten von der Außenwelt zu erhalten und sie zu speichern haben zwar im neuen BASIC nicht zugenommen; da war der alte C64 schon hinreichend ausgestattet. Aber all dies geht im neuen BASIC viel einfacher. Der C64 war ursprünglich für den Betrieb mit einem Kassetten-Gerät ausgelegt. Zum Zeitpunkt seines Erscheinens auf dem Markt waren die komfortableren Floppys für Besitzer von Homecomputern noch viel zu teuer. Entsprechend wurde der Befehlsvorrat des C64 ganz auf die Arbeit mit der Datasette abgestimmt. Floppys können zwar bedient werden, aber nicht so bequem wie die Datasette.

Dies ist jetzt anders. Es gibt für jede wichtige Operation im Zusammenhang mit Disketten einen eigenen Befehl. Das beginnt schon bei der Anzeige des Inhaltsverzeichnisses, die man jetzt über den DIRECTORY-Befehl erhält (mit dem auch die Funktionstaste F3 belegt ist). Dabei wird selbstverständlich ein BASIC-Programm im Speicher nicht überschrieben; auch dies ein großer Fortschritt gegenüber dem C64.

Fabrikneue Disketten bereiten Sie mit dem HEADER-Befehl ganz einfach für die Arbeit vor (man nennt das auch "Formatieren"). Zum Öffnen von Disketten-Dateien gibt es jetzt den DOPEN-Befehl; analog schließen Sie Ihre Dateien mit DCLOSE. Ist beim Verkehr mit der Diskette im Programm mal was schiefgegangen und Sie konnten nicht mehr alle geöffneten Kanäle schließen, ist auch das kein Problem. Der Befehl DCLEAR bereinigt die Situation.

Laden und Speichern von Diskettendateien geht auch besonders einfach. Sie benutzen einfach DLOAD und DSAVE. Wenn Sie auf Nummer sicher gehen wollen, so können Sie mit DVERIFY auch noch überprüfen, ob Ihr Programm auch korrekt gespeichert wurde.

Schließlich existiert mit BLOAD noch ein blitzschneller Ladebefehl, der zusammen mit seinem Kollegen BSAVE beliebige Binärdaten, also z.B. auch komplette Grafiken zwischen Diskette und Computer hin- und herschaufeln kann.

Das Arbeiten mit Datendateien wird auch bequemer. APPEND fügt Daten an das Ende einer sequentiellen Datei an. Bei Relativdateien erlaubt Ihnen

die RECORD-Anweisungen den freien Zugriff auf jede Stelle innerhalb der Datei.

3.3.3.4 BASIC und Maschinensprache

Den Freak wird's freuen: auch die Kommunikation zwischen BASIC und selbstgeschnitzten Maschinenprogrammen ist komfortabler geworden. Die bereits aus dem BASIC 2.0 bekannten Befehle PEEK und POKE, USR und SYS sind auch hier vorhanden. Neu ist der BANK-Befehl, der das Umschalten von Speicherbänken für nachfolgende Befehle erlaubt. Neu ist auch, daß man dem SYS-Befehl Parameter übergeben kann, die die Maschinen-Register mit Werten laden.

Es gibt aber noch eine weitere bequeme Möglichkeit zur Daten-Kommunikation zwischen BASIC und Maschinen-Routinen. Mit POINTER können Sie sich nämlich die Adresse von BASIC-Variablen mitteilen lassen, so daß diese direkt im Maschinenprogramm benutzt werden können.

Zu allen hier erwähnten Befehlen finden Sie im Anhang A ausführliche Informationen. Die Darstellung an dieser Stelle verfolgte nur den Zweck, Sie mit den erweiterten Möglichkeiten bekannt zu machen und Ihre Phantasie anzuregen. Zwei wichtige Bereiche der Programmierung auf dem C128 sind dabei völlig unter den Tisch gefallen: die Grafik-Programmierung und die Sound-Programmierung. Weil das neue BASIC hier so viel zu bieten hat, ist diesen Themen auch ein eigenes Kapitel gewidmet.

4 Grafik und Sound

4.1 Wie die Bilder auf den Schirm kommen

Die Grafik war schon immer eine der Stärken der Commodore-Homecomputer. Davon zeugen unter anderem auch die vielen wundervollen Videospiele, die es für den C64 gibt. Was auf dieser Maschine aber schwach ist: die Grafikprogrammierung.

Dies scheint ein Widerspruch zu sein. Die Situation auf dem C64 ist jedoch die: man hat alle Möglichkeiten - spricht: den Grafik-Chip - direkt unter seinen Fingerspitzen, aber man kommt in BASIC nicht an ihn heran! Stattdessen mußte man ein Mischmasch aus Maschinen-Programmierung und BASIC benutzen. Der VIC wird über sogenannte Register programmiert; das sind lediglich Speicherstellen im Arbeitsspeicher, die für diesen Zweck reserviert sind und in die man bestimmte Werte (Byte-Muster) zu schreiben hat, um den VIC zu irgendeiner Aktion zu bewegen. Grafikprogrammierung verlangte auf dem C64 also die Kenntnis dieser Register und der verlangten Werte. Hatte man sich die Informationen erst mal beschafft, dann ging es erst los. Das Programm zur Grafikprogrammierung artete meist in eine wahre PEEK-und POKE-Orgie aus, die schnell vergessen ließ, warum BASIC eine "höhere Programmiersprache" genannt wird.

Damit ist es jetzt zu Ende. Allerdings muß derjenige, der seinen C128 ausschließlich in Assembler programmieren will, noch immer auf dieses archaische Verfahren zurückgreifen.

Für die anderen aber brechen herrliche Zeiten an. Zur vollständigen Nutzung des Befehlssatzes im neuen BASIC 7.0 ist es aber von Vorteil, sich mit der Frage zu beschäftigen, wie der C128 die Bilddarstellung bewerkstelligt.

4.1.1 Der Textbildschirm

Das Naheliegendste ist es, sich zuerst einmal die Frage zu stellen, wie der Computer die Zeichen auf dem Bildschirm darstellt. Der 40-Zeichen-Bildschirm mag Ihnen vielleicht als nicht besonders lohnend für eine genauere Untersuchung erscheinen. Tatsächlich stecken in ihm aber vielfältige Möglichkeiten. Es wird Sie vielleicht erstaunen zu erfahren, daß die weitaus meisten Videospiele ausschließlich durch geschickte Nutzung seiner Möglichkeiten ihre erstaunlichen Effekte erzielen.

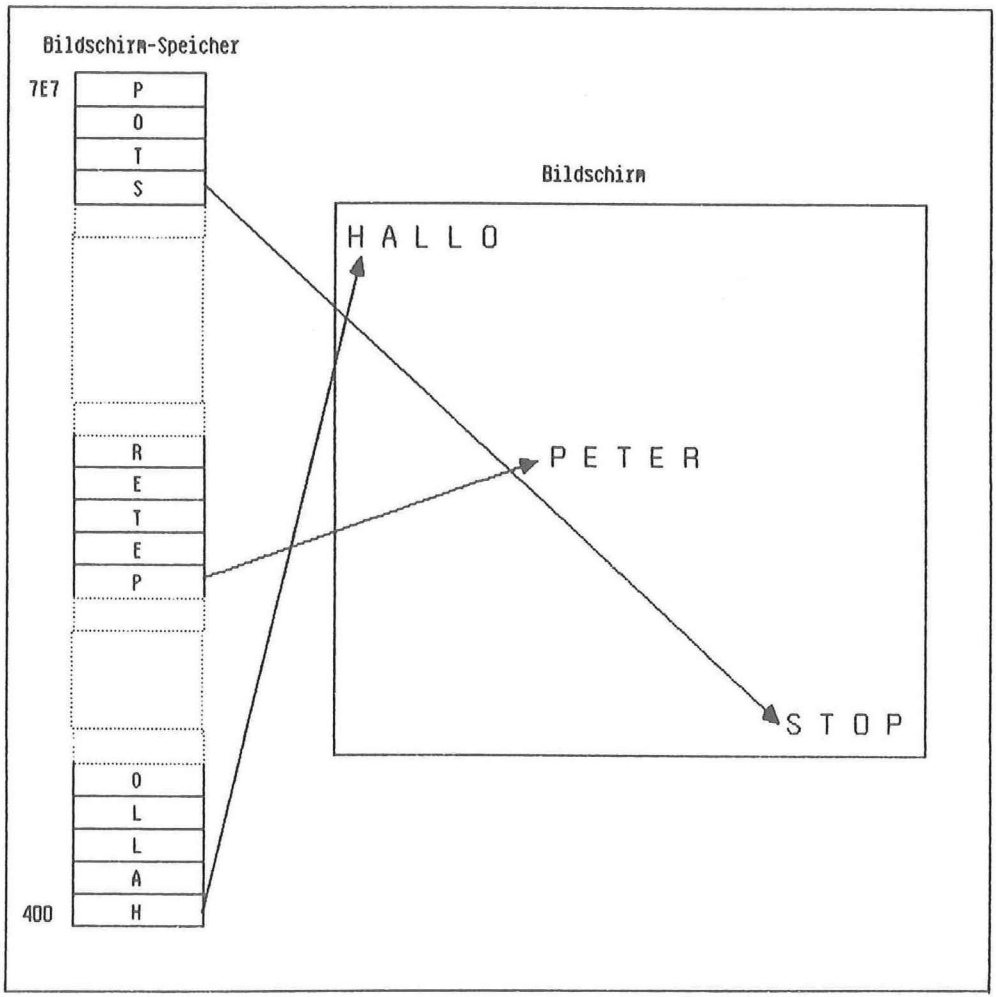


Abbildung 4.1: *Zusammenhang zwischen Bildschirmzeichen-Speicher und der Anzeige auf dem Bildschirm*

Zur Darstellung eines Bildschirminhalts mit Text oder Grafikzeichen benötigt der C128 genauso wie sein Vorgänger, der C64, mehrere Bereiche im Arbeitsspeicher. Ein Speicherbereich nimmt die Zeichen auf, die auf dem Bildschirm erscheinen sollen. Da hier 25 Zeilen mit jeweils 40 Zeichen dargestellt werden können, muß dieser Speicherbereich Platz für insgesamt 1000 Zeichen haben. Er wird auch Bildschirmzeichen-Speicher genannt und befindet sich an den Speicherstellen 400 bis einschließlich 7E7 (hexadezimal; entspricht dezimal 1024 bis einschließlich 2023). Was hier drinsteht bestimmt, was auf dem Bildschirm angezeigt wird. Dies soll die Abbildung 4.1 verdeutlichen.

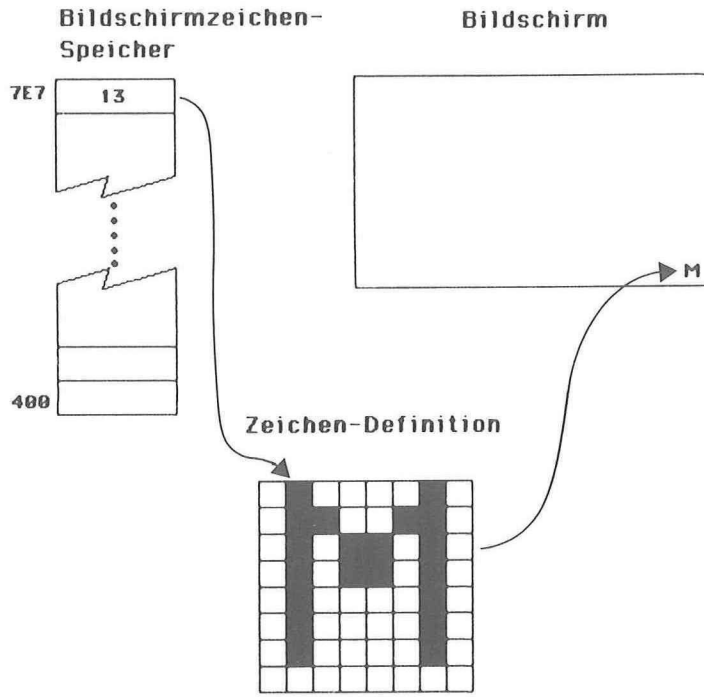


Abbildung 4.2: *So kommt das "M" auf die Mattscheibe*

Allerdings stehen in den Speicherstellen des Bildschirmspeichers nicht die Zeichen selbst, sondern lediglich Zeiger auf ihre Definition (sog. Bildschirm-Codes). Wenn Sie Ihren Bildschirm genau betrachten, dann werden Sie sehen, daß jedes Zeichen aus einer kleinen Matrix aus 8 mal 8 Punkten zusammengesetzt ist. Im Bildschirmspeicher wird nun nicht diese Matrix festgeschrieben. Die hätte auch gar keinen Platz, da man pro Zeichen ja 8 Bytes bräuchte, was bei 1000 Bildschirmstellen schon 8000 Bytes ausmacht.

Stattdessen wird nur auf die Zeichendefinition verwiesen. Dazu gibt es den sog. Zeichensatz-Generator, welcher nichts anderes als ein weiterer Speicherbereich ist, der sich aus insgesamt 256 solcher Zeichen-Matrizen zusammensetzt. Die Zahl 256 kommt dadurch zustande, daß insgesamt 128 Zeichen definiert werden müssen (Text- und Grafik-Zeichen); dazu kommt noch einmal für jedes Zeichen seine inverse Darstellung. VIC sucht sich also für jede Zeichenposition auf dem Bildschirm erst den sog. Bildschirmcode im Bildschirm-Textspeicher, geht damit in den Zeichensatz-Generator, um die Zeichendefinition zu ermitteln und bringt dann das entsprechende Zeichen auf den Bildschirm. Dazu die Abbildung 4.2.

Jetzt kommt Farbe ins Spiel! Sie wissen ja aus Kapitel 2, daß Sie die Zeichenfarbe ändern können. Wenn Sie wollen, können Sie jedes einzelne Zeichen auf dem Bildschirm mit einer anderen von insgesamt 16 möglichen Farben darstellen. Diese Farbwahl muß dem VIC auch mitgeteilt werden. Dazu ist ein weiterer Bereich im Arbeitsspeicher reserviert, der sog. Farbspeicher. Da dort die Farbe für insgesamt 1000 Zeichen vermerkt werden muß (24 Zeilen mit 40 Zeichen), werden wieder 1000 Speicherstellen benötigt. Sie finden diesen Farbspeicher an den Stellen D800 bis DBE7 (55296 bis 56295 dezimal). Jedes Byte dieses Speichers entspricht einem Byte im Bildschirmzeichen-Speicher und sagt aus, welche Farbe das zugehörige Zeichen haben soll. Im Farbspeicher stehen Farbcodes. Wenn also an der Adresse D800 eine 5 steht (Farbcode für Grün) und an der Adresse 400 eine 7 (Bildschirmcode für "G"), dann bedeutet das, daß in der linken oberen Bildschirm-Ecke ein grünes G erscheinen wird.

Jetzt sind beinahe alle Informationen vorhanden. Da gibt es aber noch eine Hintergrundfarbe und eine Randfarbe. Auch die sind nicht festgeschrieben, sondern hängen vom Inhalt von Speicherstellen ab: Adresse D020 (Dezimal: 53280) enthält den Farbcode für den Bildschirmrand, ihre Nachbarin, die Adresse D021 (Dezimal: 53281) bestimmt die Farbe des Bildschirmhintergrunds. Die Abbildung 4.3 veranschaulicht noch einmal das Zusammenspiel all dieser Informationsquellen.

Der große Vorteil dieses Verfahrens: alles ist variabel. Mit dem üblichen Zeichensatz kann man nicht unbedingt aufregende Grafiken machen. Aber man kann sich diesen Zeichensatz ganz oder teilweise neu definieren. Die Bitmuster, die das Aussehen der 8x8-Matrix bestimmen, beginnen ab der Adresse D000. Dort überlagern sie sich allerdings teilweise mit dem Farbspeicher. Dies geht, wie Sie sich erinnern werden, weil der Speicher durch Konfiguration in mehreren Ebenen belegbar ist. Aber VIC ist sehr tolerant: er verlangt nicht kategorisch, daß die Zeichendefinitionen an dieser Stelle zu finden sind. Stattdessen ist ihm jede andere Adresse genauso recht. Das bedeutet: Sie können sich irgendwo im Arbeitsspeicher einen eigenen

Zeichensatz definieren, der alle für Ihr Programm nötigen Grafikzeichen enthält, und dann VIC sagen, er möge ab jetzt diesen Satz benutzen.

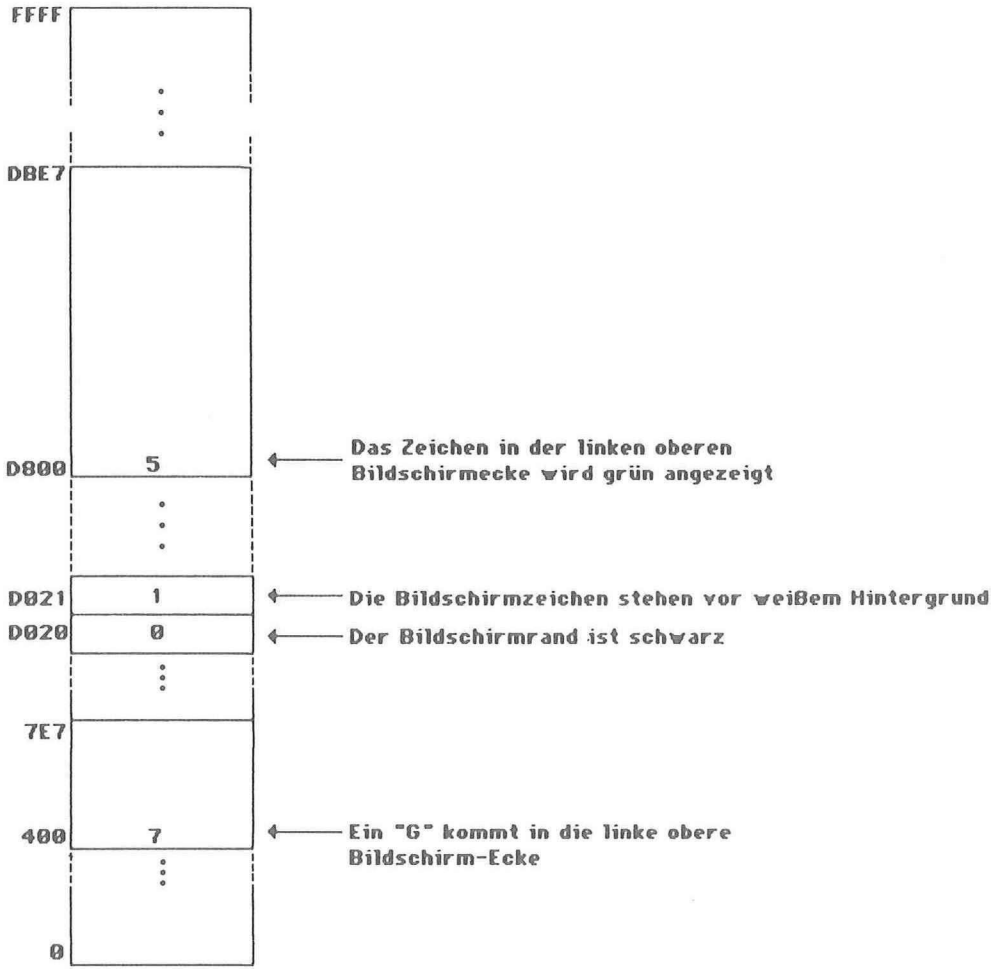


Abbildung 4.3: *Zeichen-, Farb-, Hintergrund- und Randinformation*

Ebenso ist der Bildschirmzeichen-Speicher nicht an der in der Abbildung aufgeführten Stelle festgenagelt. Es ist sogar möglich, mehrere Bildschirmzeichen-Speicher gleichzeitig verfügbar zu halten und VIC von einem zum anderen zu geleiten. Wenn Sie sich schon mal gefragt haben, wieso viele Videospiele so schnell zwischen unterschiedlichen Spiel-Szenen hin- und herschalten können: hier ist die Antwort zu finden!

Die fortgeschritteneren Anwendungen wie mehrfache Zeichensätze oder parallele Bildschirm-Zeichenspeicher bleiben auch im BASIC 7.0 eine Domäne der PEEK- und POKE-Programmierung bzw. der Maschinensprache. Was aber wesentlich einfacher geworden ist, ist das Aussuchen von Farben für Zeichen-, Bildschirm-Hintergrund- und Randfarbe. Dafür gibt es jetzt die COLOR-Anweisung, die sich aber nicht nur um den Textbildschirm kümmert, sondern auch um die hochauflösende Grafik.

4.1.2 Die hochauflösende Grafik.

Womit wir beim zweiten wichtigen Punkt angelangt wären. Neben dem 40-Zeichen-Textbildschirm gibt es im 128er-Modus noch einen hochauflösenden Grafik-Modus. Im Textmodus wird der Bildschirm als eine zweidimensionale Tabelle mit 40 mal 25 Stellen aufgefaßt. Es ergeben sich insgesamt 1000 Stellen, eine für jede mögliche Zeichenposition. Im Grafik-Modus hingegen sehen Sie den Bildschirm als Tabelle mit 320 mal 200 Punkten. Das macht insgesamt 64000 Bildpunkte, die Sie in diesem Modus einzeln ansprechen können. Sie können damit Zeichnungen mit einer wesentlich größeren Auflösung (oder Bildschärfe) erstellen. Ähnlich wie im Text-Modus können diese Zeichnungen auch coloriert werden; Sie werden darüber gleich mehr erfahren.

Den hochauflösenden Grafik-Modus gab es auch schon auf dem C64. Seine Programmierung war aber ausgesprochen haarig und nur etwas für intime Kenner der Maschine. Jetzt gibt es aber eine riesige Auswahl an BASIC-Befehlen für die Arbeit mit diesem Modus, die keinen Wunsch mehr offen lassen.

Viele dieser Befehle beziehen sich auf ein Koordinatensystem, das es erlaubt, jeden einzelnen Bildpunkt in der hochauflösenden Grafik anzusprechen. Dieses Koordinatensystem adressiert die linke obere Ecke des Bildschirms mit den Koordinaten "0,0", die rechte obere Ecke mit "319,0" und die rechte untere Ecke mit "319,199". Der Bildpunkt, der sich genau in der Mitte des Bildschirms befindet, hat also die Koordinaten "160,100". Die Abbildung 4.4 soll dies verdeutlichen.

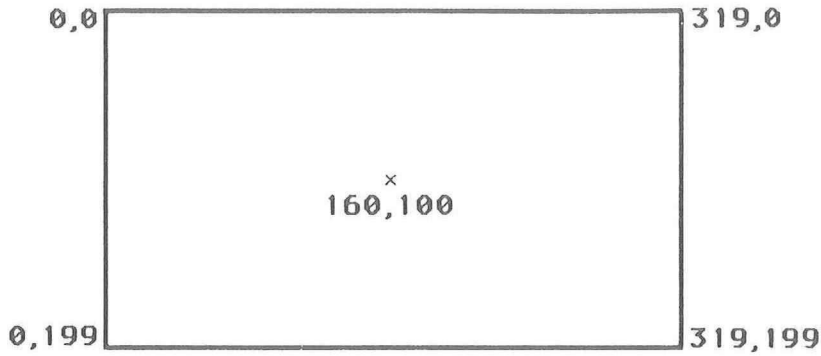


Abbildung 4.4: *Das Koordinatensystem für hochauflösende Grafik*

Das Koordinatensystem ist bei der Arbeit mit dem Pixel-Cursor von Bedeutung. Mit dem Wort "Pixel" bezeichnet man üblicherweise einen einzelnen Bildpunkt auf dem Grafik-Bildschirm. Ebenso, wie auf dem Textbildschirm der Cursor angibt, wo das nächste Zeichen erscheinen wird, bestimmt am Grafikschirm der Pixel-Cursor, wo der nächste Bildpunkt bei einer Zeichnung gesetzt wird. Den Pixel-Cursor selbst kann man zwar nicht sehen (er wäre auch gar zu winzig, weswegen der Computer gar nicht erst versucht, ihn anzuzeigen); aber man kann ihn bewegen. Dazu gibt es die LOCATE-Anweisung.

Drei BASIC-Anweisungen sind es im wesentlichen, die Ihnen gestatten, Zeichnungen auf den Grafik-Bildschirm zu bannen. Da ist einmal die BOX-Anweisung, die das Zeichnen beliebiger Rechtecke erlaubt. Diese Rechtecke können auch noch um den Mittelpunkt gedreht und farbig ausgefüllt werden. Die CIRCLE-Anweisung ermöglicht Ihnen, Kreise und Ovale, aber auch beliebige Polygonzüge auf den Bildschirm zu bannen. Auch Ausschnitte aus diesen Figuren können produziert und das ganze wiederum um den Mittelpunkt gedreht werden. CIRCLE kann zwar - anders als BOX - die erzeugten Figuren nicht ausmalen, aber dafür gibt es dann die praktische PAINT-Anweisung. Diese vermag beliebige geschlossene Umrisse auszumalen, welche nicht unbedingt mit CIRCLE oder BOX erzeugt werden müssen. Es gibt nämlich noch einen weiteren sehr generellen Zeichenbefehl, der DRAW heißt und zwei beliebige Punkte auf dem Bildschirm durch eine Gerade verbinden kann. Auch Einzelpunkte können mit DRAW gesetzt werden, weswegen sich der Befehl vorzüglich dazu

eignet, die von der Grafik-Sprache LOGO her bekannte Schildkröten-Grafik in BASIC nachzubilden.

4.1.3 Shapes

Auf dem hochauflösenden Grafikbildschirm können Sie Zeichnungen nicht nur erzeugen. Sie können Sie sich auch speichern und später (oder woanders auf dem Bildschirm) wieder anzeigen. Dazu hat Ihr BASIC 7.0 ein neues Merkmal bekommen, die sogenannten "Shapes". Shapes sind BASIC-Stringvariablen, in denen jedoch keine Texte gespeichert sind, sondern Ausschnitte aus dem Grafikbildschirm. Wenn Ihnen eine besonders hübsche Zeichnung gelungen ist und sie diese der Nachwelt aufbewahren wollen, dann teilen Sie BASIC mit dem SSHAPE-Befehl einfach mit, wo sich die Zeichnung auf dem Bildschirm befindet und in welchen String sie gespeichert werden soll. Diese Stringvariable können Sie jetzt nicht nur in Ihrem Programm weiterbearbeiten - was aber zu durchaus merkwürdigen Ergebnissen führen kann - sie läßt sich auch auf Diskette oder Kassette speichern.

Den umgekehrten Weg muß man natürlich auch gehen können. Der besteht darin, ein in einer Stringvariablen gespeichertes Shape wieder auf den Bildschirm zu zaubern; dafür ist GSHAPE da. Die Anweisung kann allerdings mehr als nur ein gespeichertes Bild wieder zurückzuholen. Sie können nämlich die Art der Anzeige beeinflussen und so interessante Effekte erzielen. Im Anhang A finden Sie ein Beispiel, das zeigt, wie man ein Shape vor einem Hintergrund bewegen kann, ohne diesen zu löschen.

Zeichnungen erzeugen und speichern ist in vielen Anwendungen eine äußerst nützliche Eigenschaft. Denken Sie nur an die Möglichkeit, Statistiken durch (farbige!) Grafiken zu illustrieren. Mit Ihrem neuen BASIC 7.0 ist dies kein Problem mehr. Aber was noch mehr ist: sie können solche Grafiken auch beschriften! Dies ist keine Selbstverständlichkeit, da der Grafik- und Textmodus üblicherweise streng getrennt sind und es unter den normalen Umständen nicht möglich ist, Buchstaben in den Grafikbildschirm zu schreiben. Der CHAR-Befehl macht aber genau dies möglich. Sie geben nur die Anfangskoordinaten an, und schon erscheint der Text an der gewünschten Stelle.

4.1.4 Farbe auf dem Grafikbildschirm

Die Bilder werden durch Farbe erst schön! Die bereits erwähnte COLOR-Anweisung erlaubt es Ihnen, auch für den Grafikbildschirm eine Zeichen-Farbe auszuwählen - so geschrieben, weil Sie damit ja wirklich zeichnen. Wem das aber noch nicht genug ist, wem es nicht buntschillernd genug sein kann, der kann sogar mit zwei Zeichenfarben zugleich arbeiten. Dies nennt man den Multicolor-Modus, der es erlaubt, jedem Bildpunkt auf dem Grafik-Bildschirm eine von zwei möglichen Farben zuzuweisen.

Allerdings geht hier die Farbenpracht auf Kosten der Auflösung. BASIC kann dies nämlich nur bewerkstelligen, indem es zur Darstellung der Farbinformation immer zwei horizontale Bildpunkte zusamenfaßt. Die horizontale Auflösung beträgt jetzt nicht mehr 320 Punkte, sondern 160 Punkte. Sie bewegen sich in diesem Modus also in einem Koordinatensystem von 160 mal 200 Punkten. Dafür kann jeder dieser 32000 Bildpunkte jetzt in einer von zwei möglichen Farben leuchten.

Zum Einstellen des gewünschten Modus - Text, hochauflösende Grafik, Multicolor - gibt es die praktische GRAPHIC-Anweisung. Die Funktionstaste F1 ist mit diesem Kommando belegt. GRAPHIC hat eine Besonderheit zu bieten: den geteilten Bildschirm. Sie können sich nämlich einen hochauflösenden Grafikbildschirm einblenden, aber die letzten vier Bildschirmzeilen für Textanzeige reservieren. Das hat den Vorteil, daß Sie im unteren Textfenster Grafikanweisungen schreiben und zugleich oben deren Resultat begutachten können. Ein geteilter Bildschirm ist übrigens auch im Multicolor-Modus möglich.

Das Arbeiten mit dem geteilten Bildschirm hat nur einen kleinen Schönheitsfehler: haben Sie nämlich in der Anweisung im Textfenster einen (Syntax- oder ähnlichen) Fehler gemacht, dann schaltet BASIC automatisch auf den Textbildschirm um und blendet die Grafik aus. Das bedeutet aber nicht, daß Ihre Zeichen-Bemühungen verloren wären. Bei Rückkehr in den jeweiligen Grafik-Modus werden Sie den Bildschirm genauso vorfinden, wie Sie ihn verlassen haben.

4.1.5 Die Creme de la Creme: Sprites.

Die Krönung all der schönen Grafik - sind die Sprites! Sprites sind vom Benutzer definierbare Grafikobjekte, die bewegt werden können. Die Figuren können innerhalb eines Feldes von 24 mal 21 Punkten beliebige Umrisse annehmen. So ist es möglich, eine Spielfigur als Sprite zu definieren,

ein Rennauto, ein kleines Flugzeug, ein Segelboot, eine Rakete... Die Möglichkeiten sind grenzenlos.

Daß man Figuren definieren kann, ist an sich noch nichts Sensationelles. Auch die Shapes eignen sich dafür. Aber Sprites kann man beleben, genauer: man kann sie bewegen lassen. Dazu müssen Sie dem VIC nur den Befehl erteilen, ein Sprite mit einer bestimmten Geschwindigkeit und in einem bestimmten Winkel über den Bildschirm zu schieben. Er tut dies, ohne daß er noch weitere Instruktionen von Ihnen benötigt. Die Bewegung läuft völlig selbsttätig unter der Kontrolle des VIC ab; vor allem ist sie völlig kontinuierlich. Das Sprite gleitet ohne Rucken glatt über den Bildschirm; für den Betrachter sieht es aus, als ob ein Zeichentrickfilm abliefe. Als Bewegungsrichtungen sind alle Richtungen der Windrose möglich. Während das Sprite über den Bildschirm wandert, können Sie per Programm andere Dinge darauf ausgeben, ja sogar ein Programm schreiben. Sprites können übrigens über jeden Bildschirm, also auch über den Grafik- und Multicolor-Schirm bewegt werden.

Aber damit nicht genug. VIC kann nämlich bis zu 8 Sprites gleichzeitig verwalten. Diese Sprites werden aber unterschiedlich behandelt. Jedes Sprite hat eine eigene Nummer und lebt in einer eigenen Ebene. Die Ebene bestimmt, welche Priorität das Sprite gegenüber seinen Kollegen hat. Die Priorität ist dann wichtig, wenn zwei Sprites auf dem Bildschirm zusammenstoßen oder kollidieren. Dann schiebt sich nämlich das Sprite mit der höheren Priorität über das Sprite mit der niedrigeren Priorität. Dadurch entstehen dreidimensionale Effekte: die Illusion des Trickfilms ist perfekt.

Der räumliche Eindruck kann durch eine weitere Möglichkeit noch gesteigert werden. Aus den obigen Ausführungen geht hervor, daß Sprites über den Bildschirm wandern können, während andere Daten - Grafiken oder Text - dort angezeigt werden. Man kann nun für jedes Sprite vereinbaren, ob es Priorität über die Anzeigedaten hat. Das bedeutet, daß sich ein Sprite bei Überdeckung mit den Anzeigedaten je nach Priorität davor oder dahinter vorbeibewegt. Dadurch wird die Illusion der Räumlichkeit noch perfekter. Anzeigedaten werden übrigens durch vorbeiziehende Sprites nicht geändert; sie bleiben unverändert erhalten.

Der Benutzer hat über die Sprites in seinem Programm - oder auch im Direktmodus - volle Kontrolle. Er kann sie ein- und ausschalten (wodurch sie auf dem Bildschirm sichtbar oder unsichtbar werden), er kann jederzeit ihre Richtung und Geschwindigkeit verändern, er kann sie verschieden einfärben, er kann sie sogar wachsen und schrumpfen lassen. Dazu dienen die Befehle SPRITE und MOVESPR.

Sprites werden häufig als bewegte Spielfiguren eingesetzt. In diesen Fällen ist es wichtig, Kollisionen zwischen Sprites in einem Programm erkennen und entsprechend behandeln zu können. Wenn Sie z.B. ein Autorennen programmieren, in dem die Rennwagen Sprites sind, dann würde eine Kollision zweier Sprites einen Zusammenstoß auf der Rennbahn signalisieren. Für diesen Fall sind Vorkehrungen im Programm zu treffen.

Damit Sie nun nicht pausenlos die Position der einzelnen Sprites im Programm abfragen müssen, hat sich Commodore etwas besonderes einfallen lassen. Sie können nämlich für den Fall einer Sprite-Kollision eine "Sprite-Falle" aufbauen, ähnlich wie es Fehlerfallen für die Fehlerbehandlung gibt. Mit der COLLISION-Anweisung können Sie veranlassen, daß bei einer Kollision zwischen Sprites ein spezieller Programmabschnitt automatisch angelaufen wird. In diesem Unterprogramm können Sie dann z.B. abfragen, welche Sprites und wieviele Sprites miteinander kollidierten (mit BUMP und/oder RSPOS) und entsprechende Maßnahmen ergreifen. Natürlich gibt es auch Befehle, mit denen Sie sich umfassend über den momentanen Zustand eines Sprites informieren können (Farbe, Geschwindigkeit, Winkel, Vergrößerungsfaktor etc.) damit lassen sich alle denkbaren Spielsituationen bequem bewältigen!

Auch das Erzeugen von Sprites - auf dem C64 ein schon beinahe qualvoll zu nennendes Unterfangen - ist im BASIC 7.0 ein Kinderspiel. Dafür gibt es jetzt einen eigenen Sprite-Editor, der mit dem Befehl SPRDEF aufgerufen wird. Weil Sie ihn ja sicher oft brauchen und er doch einiges kann, wollen wir ihm ein eigenes Unterkapitel widmen.

4.1.6 SPRDEF: der Sprite-Editor

Nach Aufruf des Editors ist der Bildschirm in zwei Bereiche unterteilt. Links sehen Sie in vergrößerter Darstellung die "Arbeitsfläche", einen Bereich mit 24 Zeilen und 21 Reihen. Darauf "malen" Sie die Bildpunkte, aus denen sich das Sprite zusammensetzen soll. Rechts erscheint das Sprite dann in Originalgröße, damit Sie gleich den richtigen Eindruck vom Ergebnis Ihrer Arbeit bekommen.

Ehe Sie mit dem Entwerfen anfangen können, müssen Sie dem Editor aber noch die Nummer des Sprites mitteilen, das bearbeitet werden soll. Geben Sie hier eine Zahl zwischen 1 und 8 ein. Dabei kann natürlich auch die Nummer eines bereits definierten Sprites angegeben werden, das Sie eventuell verschönern oder anderswie modifizieren wollen.

Wird ein neues Sprite gewählt, so empfiehlt es sich, zuerst einmal die Arbeitsfläche durch Löschen zu bereinigen. Dazu dient die CLR-Taste. Jetzt können Sie beginnen, Ihr Design zu entwerfen. Sie sehen auf der Arbeitsfläche einen Cursor in Form eines Pluszeichens. Mit den Pfeiltasten zur Cursor-Steuerung können Sie diesen - wie Sie es auch vom Textcursor gewohnt sind - frei über die Arbeitsfläche bewegen. Der Cursor ist mit einer automatischen Wiederholfunktion ausgestattet, die jedoch durch Betätigen der "A"-Taste ausgeschaltet werden kann. Diese ist ganz besonders vorteilhaft beim Zeichnen von diagonalen Linien. Um an den Anfang der nächsten Zeile des Arbeitsbereichs zu kommen benutzt man wie gewohnt RETURN; in die linke obere Ecke geht's mit HOME.

Zum Setzen eines Bildpunktes müssen Sie eine der Zifferntasten 1 bis 4 betätigen. Die Zahl gibt an, aus welcher Farbquelle die Farbe für den Bildpunkt bezogen werden soll. Ein Punkt kann die Hintergrundfarbe oder die Textfarbe (Vordergrund des 40-Zeichen-Bildschirms) annehmen. Außerdem sind Sprites im Multicolor-Modus möglich, wodurch sich zwei weitere Angaben ergeben. Ist der Bildpunkt gesetzt, so sehen Sie auch gleich im rechten Teil des Bildschirms, wie das in Originalgröße aussieht.

Sie haben für die Vordergrundpunkte des Sprites die Wahl zwischen allen 16 Farben Ihres C128. Mit der gewohnten Tastenkombination CTRL-1 bis CTRL-8 bzw. C=-1 bis C=-8 können Sie im Editor die Farbe umschalten.

Auch das Aufblähen von Sprites ist im Editor möglich; die Taste "X" dehnt das Sprite entlang der waagrechten Achse; mit der "Y"-Achse wird es in der Senkrechten gedehnt.

Sind Sie mit dem Ergebnis Ihrer Arbeit zufrieden, so können Sie es mit SHIFT-RETURN abspeichern. Ein weiteres RETURN bringt Sie zurück ins BASIC. Sollten Sie das Sprite nicht speichern wolllen, so verlassen Sie den Editor mit der STOP-Taste.

Ähnlich den Shapes können auch Sprites in einem String abgespeichert werden. Dies geht mit dem SPRSAV-Befehl, der auch in der umgekehrten Richtung betrieben werden kann. Somit kann der Inhalt eines Strings zur Definition eines Sprites dienen. Dieser String kann auch ein Shape sein. Sollen besonders komplizierte Gebilde entworfen werden, so empfiehlt es sich vielleicht, diese erst auf dem Grafikbildschirm zu zeichnen, dann als Shape abzuspeichern und schließlich mit SPRSAV zu einer Sprite-Definition zu machen.

Jetzt aber viel Spaß bei der Arbeit!!

4.2 (Fast) alle Klänge dieser Welt

... können mit dem 128er erzeugt werden. Musikstücke mit bis zu 3 Stimmen (er besitzt drei getrennt steuerbare Tongeneratoren) und Geräusche vom Gewehrschuß bis zum futuristisch anmutenden Elektronik-Gebrabbel sind möglich. Mit dem 64er wurde die Leistungsfähigkeit des Sound-Chips 6581 - der auch im PC128 eingebaut ist - anschaulich demonstriert; nicht ohne Grund gibt es für den C64 schon seit langem Klavier-ähnliche Tastaturen zu kaufen. Daß es mit den Sound-Befehlen des leistungsstarken C128-BASIC nun noch einfacher sein wird, Geräusche verschiedenster Art zu erzeugen, dürfte einleuchten. Doch bedarf es, will man die Sound-Features des SID voll ausnutzen, eines gewissen Maßes an Grundwissen über das Thema Schwingungserzeugung.

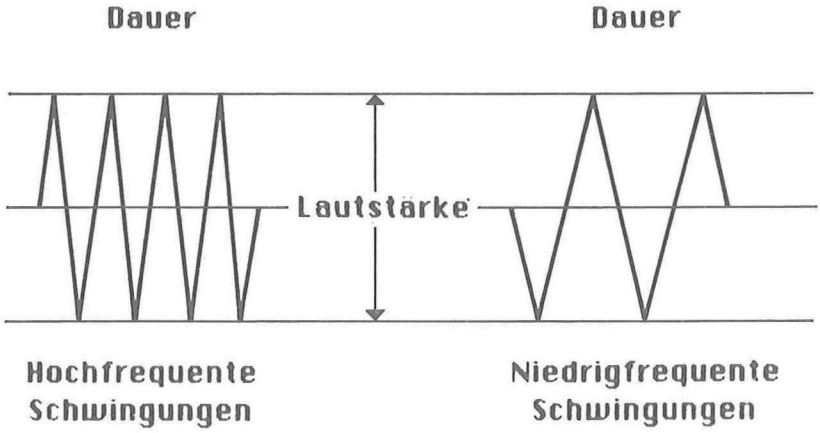


Abbildung 4.5: *Schwingungen*

4.2.1 Was ist eine Schwingung

Physikalisch angehauchte Leser werden jetzt sagen: Ein periodisch auftretendes Ereignis. Und damit haben sie auch vollkommen recht! Irgendein Vorgang, der sich in gewissen zeitlichen Abständen wiederholt, kann als Schwingung bezeichnet werden; denken Sie nur einmal an die Wellen an der Wasseroberfläche einer Pfütze, wenn Sie einen Stein hineinwerfen. Die

Auf- und Abbewegung des Wassers wäre in diesem Fall das sich wiederholende Ereignis. Oder ein Lautsprecher, der mit einem Ton angesteuert wird: Die Membran hebt und senkt sich so oft in einer Sekunde, wie es durch die Frequenz (Tonhöhe) des Steuersignals vorgeschrieben wird und so hoch oder tief, wie es die Amplitude (so eine Art Lautstärke) verlangt. Womit wir schon beim Thema wären: Die Bedeutung von Frequenz und Amplitude, den so häufig mißbrauchten Begriffen sollen Sie sich anhand der Abbildung 4.5 verdeutlichen.

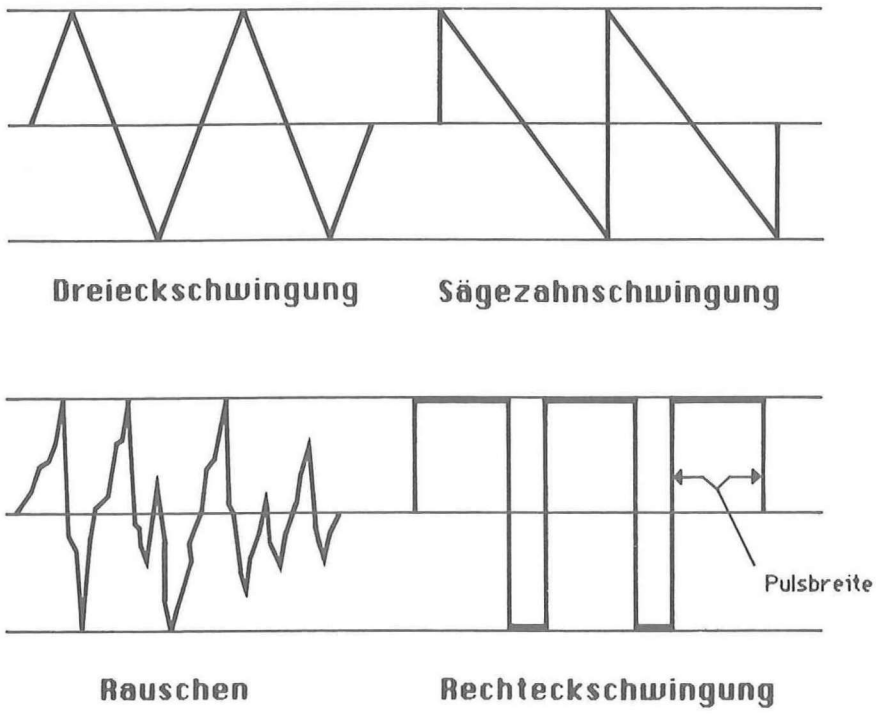


Abbildung 4.6: Schwingungsformen

Wie Sie deutlich erkennen können, dauert es bei Schwingungen mit niedriger Frequenz länger, bis sie wieder am gleichen Punkt angelangt sind als bei hochfrequenten Schwingungen. Und der Ausschlag nach oben bzw. unten, der als Amplitude bezeichnet wird, kennzeichnet die Intensität (die Stärke) des Signals.

Doch sind dies nicht die einzigen Parameter (Kenngrößen), die Aussagen darüber machen, wie eine Schwingung aussehen oder klingen soll. Denn

4.2.2 die Schwingungsform

der Schwingung ist nämlich ebenfalls ausschlaggebend für deren Klang. Die in der Mathematik und Physik so oft und gern verwendete SINUS-Kurve kennen Sie ja sicher schon; andere werden mit RECHTECK-, DREIECK- oder SÄGEZAHN-Schwingung bezeichnet (keine Verletzungsgefahr!).

Sehen Sie sich in Abbildung 4.6 einmal an, welche Kurven der PC128 für Sie bereithält.

Bis auf das "RAUSCHEN" sind die Bezeichnungen für die Wellenformen ja noch einleuchtend; aber wieso denn RAUSCHEN?

Nun, das RAUSCHEN ist das Ergebnis einer rein zufälligen Wellenform und klingt - ja, eben wie Rauschen. Daher auch der Name. Probieren Sie doch einmal folgenden Befehl aus:

```
SOUND 1,5000,50,,,,3
```

Wenn der Lautstärkeregler Ihres Fernsehers oder Monitors richtig eingestellt war, so konnten Sie ein Rauschen hören; um die anderen Wellenformen auszuprobieren brauchen sie nur den letzten Parameter (hier die "3") ändern. Die Null (0) erzeugt eine DREIECK-Schwingung, deren Klang dem der SINUS-Schwingung sehr ähnlich ist. Mit der 1 und 2 werden SÄGEZAHN- und RECHTECK-Schwingung angesteuert.

Die verschiedenen Wellenformen erzeugen also stark unterschiedliche Klänge, vom weichen Sound der DREIECKS- bis hin zur obertonreichen SÄGEZAHN-Schwingung. Sicher gibt es noch viele andere Formen, die zudem noch viel komplexer sind (denken Sie nur an die vielen verschiedenen Klänge, die mit Musikinstrumenten erzeugt werden können). Jedoch ist die Grundausstattung an Klangformen, die der PC128 bietet, durchaus akzeptabel.

4.2.3 Die Hüllkurve: Der Ton kommt und geht

Einen Klang nachzuahmen heißt nicht nur, dessen Wellenform, sondern auch das Ein- und Ausschwingverhalten nachzubilden; denn kein Ton oder

Geräusch setzt im einen Moment mit voller Intensität ein und verschwindet im nächsten Augenblick wieder vollkommen (natürlich gibt es auch hier Ausnahmen).

Dazu ist es erforderlich, die Amplitude der Schwingung im Griff zu haben. Und diese Möglichkeit ist im Sound-Chip voll realisiert worden; insgesamt vier Werte sind es, die den "Klangzauberer SID" veranlassen, die Hüllkurve einer Schwingung zu steuern:

ATTACK:
: Anstiegszeit, bis der Spitzenwert erreicht ist

DECAY:
: Abfallzeit vom Spitzenwert auf den durch SUSTAIN angegebenen Lautstärke-Wert

SUSTAIN:
: Lautstärke des Tons, nachdem ATTACK und DECAY vorbei sind

RELEASE:
: Abfallzeit von der SUSTAIN-Lautstärke auf Null

Es macht gar nichts, wenn Sie da noch nicht ganz "durchblicken"; die nächste Grafik wird um so besser verdeutlichen, was die vier Parameter bewirken:

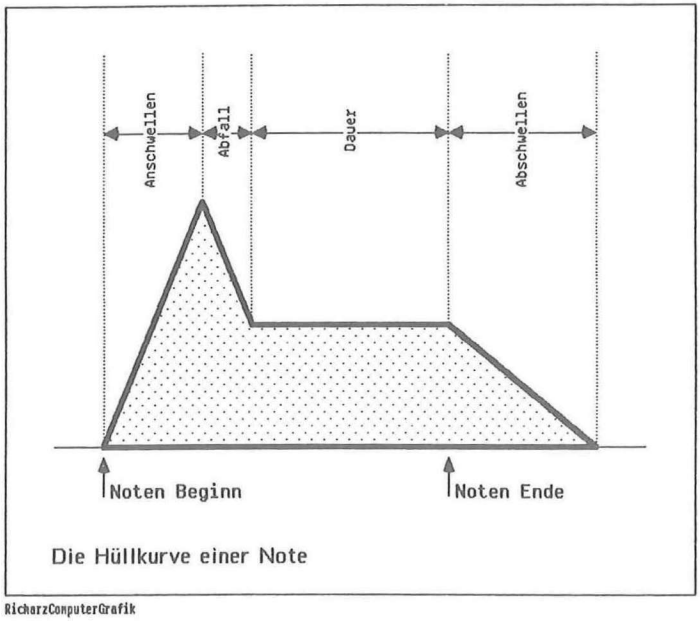


Abbildung 4.7: *Hüllkurven-Parameter*

Wenn Sie mit den ADSR-Parametern (ADSR ist nur eine Abkürzung für Attack, Decay, Sustain und Release) experimentieren wollen, sollten sie sich den BASIC-Befehl ENVELOPE im Anhang etwas näher ansehen.

Um mit diesen Parametern besser arbeiten zu können, ist es natürlich notwendig, zu wissen, welcher Wert zu welchem Ergebnis führt; in der folgenden Tabelle werden Sie diese Informationen finden.

Wert	ATTACK	DECAY/RELEASE
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

4.2.4 Filter: Hier kommt nur durch, wer darf

Filter werden überall dort eingesetzt, wo es gilt, Teile eines Gesamten auszusondern; denken Sie nur an ihre Kaffeemaschine...

Aber wozu denn einen Ton "wegfiltern", wo man ihn doch erst so mühsam erzeugt hat?

Ganz im Gegenteil! Die erzeugten Töne sind nicht unbedingt das, was durch einen Filter beseitigt werden soll. Vielmehr gilt es, die bereits erwähnten Obertöne zu unterdrücken oder durch ein Unterdrücken der Tonfrequenz deren Obertöne hervorzuheben. Das aus den Oszillatoren kommende Frequenzgemisch aus Grundton und Harmonischen (das sind die Obertöne) bietet eine Unmenge von Möglichkeiten des Einsatzes von Filtern.

Drei verschiedene Filtermöglichkeiten sind im SID eingebaut:

a) Herausfiltern von hohen Frequenzen (Tiefpaß).

b) Herausfiltern von hohen und tiefen Frequenzen, so daß nur noch ein schmales Frequenzband "durchgelassen" wird (Bandpaß).

c) Herausfiltern von niedrigen Frequenzen (Hochpaß).

Warum der Tiefpaß ausgerechnet hohe Frequenzen wegfiltern soll ist schnell erklärt: Nur die tiefen Frequenzen dürfen passieren. Beim Hoch- und Bandpaß ist diese Eselsbrücke natürlich genauso anwendbar.

Zusätzlich zu diesen drei "regulären" Filtern kann durch Kombination von Hoch- und Tiefpaß ein Filter simuliert werden, das nur einen bestimmten Frequenzbereich ausfiltert (also das Gegenteil des Bandpasses bewirkt).

Das Setzen von Filtern besorgen Sie im BASIC 7.0 mit dem FILTER-Befehl (mit welchem auch sonst?). Hüllkurven definieren Sie mit ENVELOPE, beliebige Töne können Sie mit SOUND erzeugen. Den Gipfel des Komforts bietet aber die PLAY-Anweisung. Zu allen Befehlen finden Sie im Anhang A ausführliche Erläuterungen und Beispiele.

5 Mit dem MONITOR durch die MEMORY MAP

Die aufregendste Art und Weise, einem Computer mitzuteilen, was er gefälligst zu tun (oder zu lassen) hat, ist leider auch die schwierigste: Die Assemblerprogrammierung. Doch die Ergebnisse und vor allen Dingen die Geschwindigkeiten, die sich damit erzielen lassen, sind überzeugend.

Der sogenannte MONITOR unterstützt den Programmierer bei der Erstellung eben solcher Programme. Er übersetzt den für den Menschen (relativ) leicht lesbaren Assemblercode direkt in Maschinencode, den wiederum der Prozessor versteht. Man bedient ihn (mit Einschränkungen) ähnlich wie den BASIC-Interpreter: Programme können eingegeben und gleich gestartet werden. Allerdings ist es im MONITOR nicht möglich, wie in BASIC, symbolische Adreßangaben (Variablen) zu verwenden. Dies schränkt die Möglichkeiten des Assemblerprogrammierers, vor allem in Bezug auf Größe und Komplexität des zu erstellenden Programmes gewaltig ein. Hierfür verwendet man dann spezielle Assemblierungsprogramme, die durch das Zulassen von symbolischen Adressen für Datenbereiche und Sprungziele (Labels) eine einfachere und bequemere Programmierung erlauben. Sicher wird es nicht lange dauern, bis auch für den 128er so hilfreiche und leistungsfähige Assemblerprogramme erhältlich sind, wie es sie für den C64 schon seit geraumer Zeit gibt.

5.1 Wie sag' ich's meinem MONITOR?

Der BASIC-Interpreter bietet zum Erstellen von BASIC-Programmen einige Kommandos, die Ihnen das Erstellen und Verändern von Programmen erleichtern sollen. Ähnlich ist es auch im Monitor. Mit dem "D"-Befehl wird beispielsweise (ähnlich dem BASIC-Befehl "LIST") Ihr in Maschinenprache im Arbeitsspeicher stehendes Programm in Assemblercode rückübersetzt und auf dem Bildschirm ausgegeben. "G" ist das Äquivalent zu "RUN", "L" für "LOAD" usw.

Beachten Sie bitte, daß Befehle wie "LIST" und "RUN" auch in BASIC-Programmen (also nicht nur im Direktmodus) verwendet werden können,

während dies für die MONITOR-Kommandos in Assemblerprogrammen nicht gilt. Hier erwartet der MONITOR Assemblerbefehle aus dem Befehlssatz des 8502 und deren Operanden. Der MONITOR ist ein Werkzeug zur Bearbeitung der Assemblersprache; seine Befehle gehören aber nicht zur Assemblersprache. Dies ist bei BASIC anders: die Befehle zum Bearbeiten von BASIC-Programmen sind auch BASIC-Befehle.

Folgende Tabelle gibt die Kommandos an, mit denen der MONITOR bedient werden kann:

Anweisung: **A (Assemble)**
Syntax: A <adresse> <assembleranweisung>
oder: . <adresse> <assembleranweisung>
Zweck: Übersetzung der <assembleranweisung> in Maschinencode und deren Abspeicherung an <adresse> und (falls 2- oder 3-Byte-Befehl) den darauffolgenden Speicheradressn.
Besonderheiten: Zur einfacheren Fortsetzung der Eingabe wird eine mit "A" beginnende Zeile erzeugt, deren Adreßangabe an die zuvor übersetzte <assembleranweisung> anschließt (ähnlich BASIC: AUTO).

Anweisung: **C (Compare)**
Syntax: C <start-adresse> <end-adresse> <vergleichs-adresse>
Zweck: Vergleicht den durch <start-adresse> und <end-adresse> gekennzeichneten Speicherbereich mit den ab <vergleichs-adresse> stehenden Bytes. Die nicht übereinstimmenden Speicherstellen werden ausgegeben.

Anweisung: **D (Disassemble)**
Syntax: D [<start-adresse> [<end-adresse>]]
Zweck: Rückübersetzung eines im Speicher befindlichen Maschinenprogramms in Assembleranweisungen.
Besonderheiten: Maximal 20, mindestens 7 Befehlszeilen (eine Seite) werden erzeugt (abhängig von den übersetzten Befehlen), falls <end-adresse> nicht angegeben wird. Ohne Bereichsangabe (also nur "D") wird mit der Disassemblierung an der zuvor nicht mehr übersetzten Adresse fortgefahren. Da die durch den "D"-Befehl disassemblierten Zeilen mit "." beginnen, können diese überschrieben und mit anschließendem <RETURN> assembliert werden werden.

Anweisung: **F (Fill)**
Syntax: F <start-adresse> <end-adresse> <füll-element>
Zweck: Speicherbereiche mit <füll-element> vorbesetzen.
Besonderheiten: Es kann (logischerweise) nur ein Byte als <füll-element> übergeben werden. Auch die durch <end-adresse> gekennzeichnete Speicherstelle wird mit dem <füll-element> beschrieben.

Anweisung: **G (Go)**
Syntax: G [<start-adresse>]
Zweck: Starten eines Maschinenprogramms an <start-adresse>.
Besonderheiten: Wird keine <start-adresse> angegeben, so wird der aktuelle Program Counter (PC) als Startadresse angenommen.

Anweisung: **H (Hunt)**
Syntax: H <start-adresse> <end-adresse> <such-element>
Zweck: Durchsuchen des angegebenen Speicherbereiches nach <such-element>.
Besonderheiten: Werden mehrere Bytes in <such-element> angegeben, so müssen diese durch ein Leerzeichen getrennt werden. Zu suchende Strings werden mit dem einfachen Hochkomma (') eingeleitet. Übereinstimmende Speicherstellen werden ausgegeben.

Anweisung: **L (Load)**
Syntax: L <filename> [,<gerätenummer> [,<ladeadresse>]]
Zweck: Laden eines externen Files mit dem Namen <filename> vom Peripheriegerät mit <gerätenummer> in den Arbeitsspeicher.
Besonderheiten: Wird <ladeadresse> angegeben, so wird das File auf jeden Fall an <ladeadresse> abgelegt. Die <gerätenummer> wird im hexadezimalen Format erwartet.

Anweisung: **M (Memory)**
Syntax: M [<start-adresse> [<end-adresse>]]
Zweck: Ausgabe von Speicherinhalten (Bytes) am Bildschirm und (falls darstellbar) deren ASCII-Zeichen.
Besonderheiten: Wird keine <end-adresse> angegeben, so werden im 40-Zeichen-Modus 12 Zeilen mit je 8 Bytes und im 80-Zeichen-Modus 12 Zeilen mit je 16 Bytes ausgegeben. Ohne <start-adresse> beginnt die Ausgabe ab Adresse $0000 oder der zuvor nicht mehr ausgegebenen Adresse. Da die mit dem

"M"-Kommando erzeugten Datenzeilen mit ">" (siehe ">"-Kommando) beginnen, können diese durch Überschreiben und anschließendes <RETURN> geändert werden.

Anweisung: **R (show Registers)**
Syntax: R
Zweck: Anzeigen der Prozessor-Register.
Besonderheiten: Wird nach Beendigung eines durch den MONITOR gestarteten Maschinenprogrammes durch BRK automatisch durchgeführt. Die ausgegebene Registerzeile beginnt (siehe ";" Kommando) mit ";" und kann durch Übertippen und anschließendes <RETURN> geändert werden.

Anweisung: **S (Save)**
Syntax: S <filename>,<gerätenummer>,<start-adresse>,<end-adresse>
Zweck: Abspeichern von Programmen auf Kassette oder Diskette.
Besonderheiten: Die durch <end-adresse> gekennzeichnete Speicherstelle wird nicht mehr abgespeichert.

Anweisung: **T (Transfer)**
Syntax: T <start-adresse> <end-adresse> <ziel-adresse>
Zweck: Der Inhalt des durch <start-adresse> und <end-adresse> gekennzeichneten Bereichs wird an den durch die Anfangsadresse <ziel-adresse> bestimmten Bereich kopiert.
Besonderheiten: Nach der Übertragung eines jeden Bytes wird ein Test auf Übereinstimmung durchgeführt; fehlerhafte Speicherstellen werden ausgegeben.

Anweisung: **V (Verify)**
Syntax: V <filename> [,<gerätenummer> [<vergleichs-adresse>]]
Zweck: Verifizierung eines abgespeicherten, aber noch im Arbeitsspeicher befindlichen Programms.
Besonderheiten: Im Fehlerfall wird die Meldung "ERROR" ausgegeben.

Anweisung: **X (eXit)**
Syntax: X
Zweck: Rückkehr aus dem MONITOR in den BASIC-Interpreter.

Anweisung: @
Syntax: @
Zweck: Lesen des DOS-Fehlerkanals; Ausgabe der Meldung.

Zusätzliche Schlüsselwörter:

. Hat die gleiche Bedeutung wie "A".

> Eingeben von Bytes, die im Speicher abgelegt werden sollen. Durch den Befehl "M" erzeugte Zeilen beginnen automatisch mit diesem Zeichen.

; Neubelegung der Prozessorregister; die durch den "R"-Befehl erzeugte Zeile beginnt automatisch mit ";".
Bei beiden, der Eingabe von Bytes mit ">" und der Änderung der Prozessorregister mit ";" führt erst ein abschliessendes "RETURN" zur Übernahme der Daten!

5.2 10 ist nicht mehr zehn !

Wie Sie bereits in der Einleitung zu diesem Kapitel lesen konnten, haben Sie innerhalb des MONITORS nicht die Möglichkeit, sich mit Hilfe von Variablen Speicherplatz zu reservieren und mit deren Namen einfach und bequem auf den darin gespeicherten Wert zuzugreifen. Vielmehr müssen Sie sich hier selbst merken, an welcher Speicheradresse Sie Ihre Daten untergebracht haben. Um diese Speicheradressen und auch deren Inhalte, die Bytes, darzustellen, hat man mehrere Zahlensysteme zur Verfügung: Dual, Oktal, Dezimal und Hexadezimal. Damit der MONITOR noch zwischen Zahlen unterscheiden kann, die sich auf unterschiedliche Zahlensysteme beziehen, müssen diese mit einem der folgenden Prefixes gekennzeichnet sein:

Zahlensystem	:	Prefix
Dual	:	%
Oktal	:	&
Dezimal	:	+
Hexadezimal	:	$ (oder auch: ohne Prefix)

Demnach wären die Zahlen 10, $10, &20 und %10000 gleichwertig mit der (Dezimal-) Zahl +16.

Wie Sie sehen, ist (natürlich nur in diesem Zusammenhang) 10 wirklich nicht mehr zehn, sondern sechzehn.

Alles schön und gut, werden Sie jetzt sagen, aber wie setzt man denn dieses wunderbare Programm namens MONITOR in Gang? Nichts einfacher als das. Geben Sie ein:

```
MONITOR <RETURN>
```

und schon meldet sich der MONITOR, indem er sich bei Ihnen mit seinem Namen vorstellt (manchmal ist er wirklich ein höfliches Kerlchen) und die Inhalte der Prozessorregister ausgibt (dies entspricht dem oben vorgestellten "R"-Befehl). Tut er das alles nicht, dann sollten Sie vielleicht doch lieber erst Ihren Computer einschalten, bevor Sie ihn füttern.

Der blinkende Cursor signalisiert, wie auch im BASIC-Interpreter, die momentane Eingabeposition. Sie können also bereits jetzt versuchen, die oben vorgstellten Befehle auszuprobieren (tun Sie's ruhig! Im schlimmsten Fall läuft der Computer in den Wald und findet nicht mehr heraus; dann müssen Sie ihm eben wieder heraushelfen, indem Sie die RESET-Taste drücken).

Hier nun einige Anregungen für die ersten Kontakte mit dem MONITOR:

```
M 1000 <RETURN>
```

Am rechten Bildrand sehen Sie Worte, die durchaus einen Sinn ergeben; denn der Bereich ab $1000 ist für die Funktionstastenbelegung reserviert und enthält die Strings, die durch die einzelnen Funktionstasten erzeugt werden. Wenn sie wollen, könnten Sie durch Änderung der (zweistelligen) Hexadezimalzahlen, der Bytes, auch die Funktionstasten-Strings ändern (allerdings geht das von BASIC aus einfacher; vgl. die KEY-Anweisung in Anhang A).

```
M A0 A2 <RETURN>
```

Fahren Sie mit dem Cursor ruhig öfter in die Befehlszeile und tippen Sie dann <RETURN>. Sie werden sehen, daß sich an den Adressen $A1 bis $A2 etwas verändert (und wenn Sie lange genug warten, auch bei $A0). Die Inhalte dieser drei Speicherstellen sind nichts anderes als die interne Darstellung der Systemuhr, die man von BASIC aus mit TI$ beziehungsweise TI erreichen kann (vgl. Anhang A).

```
F 400 7E7 +160 <RETURN>
```

Nur nicht wundern! Sie haben soeben den Bildschirmspeicher Ihres 40-Zeichen-Bildschirms (der liegt im Bereich von $0400 bis einschließlich $07E7) mit reversen Blanks gefüllt. Sollten sie jedoch einen 80-Zeichen-

Bildschirm angeschlossen haben, so passiert beim Verändern dieses Bereiches gar nichts, denn der für die 80-Zeichen-Darstellung verantwortliche Videocontroller (8563) hat einen eigenen Speicher hierfür und wird nur über die Adressen $D600 und $D601 im Ein/Ausgabe-Bereich angesprochen.

```
F 4D800 4DBE7 +14 <RETURN>
```

Wie Sie sicher schon erraten haben, wurde durch diesen Befehl der Inhalt des Farbspeichers für den Textbildschirm (40-Zeichen-Modus) verändert. Die erste Ziffer (die "4") gibt übrigens an, welche Bank vom MONITOR aus angesprochen werden soll. Und was, bitte, ist eine Bank?

5.3 Bänke, auf denen niemand sitzt.

Der Arbeitsspeicher Ihres C128 ist, wie der Name schon ahnen läßt, 128 Kilobyte groß (in der erweiterten Version 256 kByte).

Der Mikroprozessor 8502 kann - wie alle marktüblichen 8-Bit-Prozessoren - allerdings nur mit 64K Arbeitsspeicher auf einmal arbeiten (vornehmer ausgedrückt: er kann nur 64K adressieren). Der Arbeitsspeicher ist das "Kurzzeitgedächtnis" des Rechners: größerer Arbeitsspeicher, bessere Leistung. Da der Prozessor allein mit einem größeren Speicher nichts anfangen kann, stellt man ihm noch einen schlauen Gehilfen zur Seite, der ihn mal die einen 64K, mal die anderen 64K des Gesamtgedächtnisses unterschiebt. Der 8502 merkt davon gar nichts; er ist im Glauben, nur mit 64K zu arbeiten. Da ihm aber zweimal 64K untergeschoben werden, arbeitet er effektiv mit 128K.

So funktionierts; und genannt wird das ganze "Banking".

Bis zu vier derartiger Bänke zu 64K sind konfigurierbar (können Sie in Ihren Computer stecken und werden vom Bertriebssystem im 128er-Modus unterstützt). Jedoch läßt einiges darauf schließen, daß der 128er irgendwann einmal (mit der entsprechenden Erweiterung und vielleicht einem veränderten Betriebssystem) 16 mal 64K Arbeitsspeicher verarbeiten können wird. Dies entspräche dann einem gesamten Arbeitsspeicher von höre und staune einem ganzen Megabyte. Wann und ob überhaupt eine Erweiterung dieser Größe von COMMODORE angeboten wird, läßt sich jetzt noch nicht sagen; warten wir's ab!

5.4 Der Big Boss: Die MMU

Um jede der 64K großen Speicherbänke anwählen zu können, "sitzt" im C128 ein Baustein, der, einfach ausgedrückt, zwischen den 64 kByte großen Bänken hin- und herschalten kann und den passenden Namen "MEMORY MANAGEMENT UNIT" (MMU) trägt. Dieser Speichermanager ist also der große Bruder, der dem Mikroprozessor das "Gedächtnis" unterschiebt, das er sehen soll. Im übrigen steuert die MMU auch, in welchem Modus (C64, C128 oder Z80 CPU aktiv) sich der Computer befindet. Dies erklärt auch, warum es aus dem 64er-Modus kein Zurück mehr gibt: Der C64 kannte in der Originalversion keine MMU und darf sie deshalb auch hier nicht beachten.

Allerdings besitzt der C128 nicht nur ein "Gedächtnis", in dem zwar die tollsten Programme und Daten abgelegt werden können (das RAM, Random Access Memory), das aber durch Ausschalten des Computers wieder gelöscht wird, sondern auch ein "festes Wissen", welches Ihnen jedesmal beim Einschalten des Computers wieder zur Verfügung steht; zum Beispiel der BASIC-Interpreter und das Betriebssystem (zu finden im ROM, Read Only MEMORY). Damit die CPU dieses ROM auch zu Gesicht bekommt, übernimmt die MMU neben der Wahl der Arbeitsspeicher-(RAM-) Bank auch des Einblenden gewisser ROM-Bereiche in den für den Prozessor sichtbaren Adreßraum. So kann es beispielsweise sein, daß der Prozessor im Adreßbereich von $0000-$BFFF den Arbeitsspeicher der RAM-Bank 0 sieht, bei $C000-$FFFF hingegen das Betriebssystem.

Welche weiteren ROM-Speicher die MMU noch in den Adreßraum einblenden kann, entnehmen Sie bitte der folgenden Tabelle.

```
16 KB  BASIC-ROM (low, $4000 bis $7FFF)
16 KB  BASIC-ROM (high, $8000 bis $BFFF)
16 KB  Betriebssystem und Bildschirmeditor ($C000 bis $FFFF)
 4 KB  Zeichensatz oder Ein-/Ausgabebereich (Character ROM oder I/O
       bei $D000 bis $DFFF)
32 KB  internes oder externes "FUNCTION ROM" ($8000 bis $FFFF)
```

Mit dem Einblendungsmechanismus könnten in der Grundversion des C128 insgesamt 128 mehr oder weniger sinnvolle Speicherkonfigurationen realisiert werden. Das Betriebssystem selbst verwaltet nur sechzehn davon; diese Teilmenge ist jedoch vollkommen ausreichend:

Speicherkonfigurations-Index	gewählte RAM-Bank	Speicherkonfiguration $C000 bis $FFFF	$8000 bis $BFFF	$4000 bis $7FFF	$D000 bis $DFFF
0	0	RAM	RAM	RAM	RAM/ROM
1	1	RAM	RAM	RAM	RAM/ROM
2	2	RAM	RAM	RAM	RAM/ROM
3	3	RAM	RAM	RAM	RAM/ROM
4	0	FUNCT.ROM INTERN	FUNCT.ROM INTERN	RAM	I/O
5	1	FUNCT.ROM INTERN	FUNCT.ROM INTERN	RAM	I/O
6	2	FUNCT.ROM INTERN	FUNCT.ROM INTERN	RAM	I/O
7	3	FUNCT.ROM INTERN	FUNCT.ROM INTERN	RAM	I/O
8	0	FUNCT.ROM EXTERN	FUNCT.ROM EXTERN	RAM	I/O
9	1	FUNCT.ROM EXTERN	FUNCT.ROM EXTERN	RAM	I/O
A	2	FUNCT.ROM EXTERN	FUNCT.ROM EXTERN	RAM	I/O
B	3	FUNCT.ROM EXTERN	FUNCT.ROM EXTERN	RAM	I/O
C	0	SYST. ROM	FUNCT.ROM INTERN	RAM	I/O
D	0	SYST. ROM	FUNCT.ROM EXTERN	RAM	I/O
E	0	SYST. ROM	SYST. ROM BASIC HI	SYST. ROM BASIC LOW	RAM/ROM
F	0	SYST. ROM	SYST. ROM BASIC HI	SYST. ROM BASIC LOW	I/O

Natürlich könnten Sie auch Speicherkonfigurationen kreieren, die Ihren eigenen Vorstellungen eher entsprechen. Jedoch verlassen Sie dann den sicheren Boden des Betriebssystems und sind auf die eigenen Fähigkeiten als Assemblerprofi angewiesen.

Nun werden Sie auch verstehen, weshalb Sie beim Beschreiben des Farbspeichers des 40-Zeichen-Bildschirms eine "4" vor dessen Adreßangaben $D800 und $DBE7 setzen mußten. Weil dieser Speicher im I/O-Bereich ($D000-$DFFF) liegt, hätten Sie allerdings auch jede andere Speicherzusammenstellung wählen können, in der dieser Bereich eingeblendet ist. Wenn Sie dem MONITOR als erstes Zeichen der Adreßangabe (wohlgemerkt: einer fünfstelligen Angabe) einen dieser Speicherkonfigurationsindices liefern, so wird er Ihre Konfigurationswahl berücksichtigen.

Hierzu noch einige Beispiele:

```
M F4417 <RETURN>
```

In der Speicherkonfiguration "F" ist der BASIC-Interpreter zugeschaltet. Was Sie am rechten Rand des Bildschirms sehen, ist der Anfang der Tabelle der BASIC-Schlüsselwörter. Schalten Sie mit <C=><SHIFT> auf Groß-/Kleinschreibung um. Damit das Ende eines jeden Wortes eindeutig feststeht, ist jeweils der letzte Buchstabe großgeschrieben; dies wird erreicht durch Setzen des höchstwertigen Bits dieses Bytes.

```
M F46A9 <RETURN>
```

Wir bewegen uns immer noch innerhalb der gleichen Tabelle; aufmerksamen Lesern wird vielleicht der geheimnisvolle "QUIT"-Befehl (oben in der rechten Hälfte der MONITOR-Anzeige) auffallen. Dem BASIC-Interpreter zum Fraß vorgeworfen, reagiert dieser mit einem "? UNIMPLEMENTED COMMAND ERROR" und schweigt sich über eine zukünftige Verwendung von "QUIT" aus. Ärgern Sie Ihren COMMODORE-Händler doch ein bißchen damit!

Fühlen Sie sich jetzt wissend genug, Ihr erstes Assemblerprogramm zu schreiben? Dann auf zum nächsten Kapitel.

5.5 Bildschirmausgabe ohne PRINT

Bevor Sie auf Ihr erstes Assemblerprogramm losgelassen werden (wehe, wenn..), sollen Sie erst dessen Äquivalent in BASIC zu sehen bekommen. Verlassen Sie deshalb den MONITOR (mit: X <RETURN>), löschen Sie den Bildschirm (durch <SHIFT> und <CLR/HOME>) und achten Sie vor der Eingabe der folgenden BASIC-Befehlszeile darauf, daß sich der Cursor in der linken oberen Ecke des Bildschirms befindet:

```
A = 2 : POKE 1024 , A <RETURN>
```

Wie Sie unschwer erkennen können, hat sich das "A" in der Bildschirmecke in ein "B" verwandelt. Und warum? Bei den ersten Versuchen, mit dem MONITOR zu hantieren, haben Sie erfahren, wo der Bildschirmzeichenspeicher des C128 zu finden ist: Bei $400 bis $7E7 oder dezimal 1024 bis 2023. Es ist also nichts anderes passiert als ein Beschreiben des Bildschirmspeichers an dessen erster Adresse mit dem Bildschirmcode des Zeichens "B". Beachten sie hierbei bitte, daß die Werte, die Sie durch die BASIC-Funktion ASC(<zeichen>) erhalten, nicht zwangsläufig auch dem Bildschirmcode von <zeichen> entsprechen (s. Anhang).

Nun das Ganze noch einmal von Vorne, jetzt allerdings in Assemblersprache; starten Sie hierzu den MONITOR (durch die Eingabe von: MONITOR <RETURN>) und geben Sie folgende Zeile ein:

```
A 1400 LDA #$02 <RETURN>
```

Durch diese Zeile haben Sie dem MONITOR mitgeteilt,

A	daß eine Assemblerzeile folgt, die Sie gerne übersetzt haben möchten.
1400	daß der übersetzte Maschinencode ab der Speicheradresse $1400 abgespeichert werden soll.
LDA #$02	daß der Prozessor, sollte er den (in Maschinencode übersetzten) Befehl irgendwann einmal ausführen, den Wert $02 in sein Register A (Akkumulator) lädt.

Das Doppelkreuz (#) vor dem Operanden $02 bewirkt, daß der nachfolgende Wert direkt verarbeitet werden soll. Hätten Sie dieses Zeichen weggelassen, so hätte der MONITOR den Assemblerbefehl "LDA" in einen ganz anderen Maschinencode übersetzt (nämlich in den Code für Laden des Akkumulators mit dem in der Speicherschublade $02 liegenden Wert).

Wenn Sie diese Zeile richtig eingegeben haben, reagiert der MONITOR darauf mit einer Veränderung der Zeile (er zeigt Ihnen, welchen Maschinencode er aus dem Assemblerbefehl und dessen Operanden übersetzt hat) und erzeugt (ähnlich dem AUTO-Feature in BASIC) darunter eine Zeile mit der Speicheradresse, an die der nächste Maschinenbefehl abgespeichert werden kann:

```
A 01400 A9 02     LDA #$02
A 01402
```

Die beiden Bytes $A9 und $02 sind der aus der Assembleranweisung übersetzte (erzeugte) Maschinencode. $A9 steht für "Lade den Akkumulator direkt (#) mit dem dahinter angegebenen Operanden" und $02 ist, wie Sie schon ahnen, der Operand selbst.

Als nächste Assemblerprogrammzeile geben Sie bitte ab der aktuellen Cursorposition (hinter der vom MONITOR erzeugten Speicheradresse 1402) ein:

```
STA $400 <RETURN>
```

Richtig geraten. Dieser Assemblerbefehl bewirkt das Abspeichern des Akkumulatorinhalts - nämlich $02 - an der Speicherstelle $400. Als abschließenden Befehl geben Sie ein:

```
BRK <RETURN>
```

Diese Break-Instruktion (Instruktion ist nur ein anderes Wort für Befehl oder Anweisung) beendet die Abarbeitung Ihres Programms, sobald es hier ankommt; dann übernimmt auch der MONITOR wieder das Kommando und meldet sich durch die Ausgabe der Prozessorregister des 8502.

Bewegen Sie nun den Cursor - wie vorhin - in die linke obere Ecke des Bildschirms; keine Angst, wenn dabei Ihr Programm verschwinden sollte - durch "D 1400" können Sie es wieder sehen. Starten Sie das Programm durch die Eingabe von

```
G 1400 <RETURN>
```

Sie sehen, es funktioniert wirklich. Wie auch seine BASIC-Version ersetzt das Maschinenprogramm den an Speicherstelle $400 stehenden Wert durch $02.

5.6 Rechnen kann er auch !

... allerdings beherrscht der 8502 nur die Rechenoperationen Addition und Subtraktion. Wie ist es dann möglich, daß man in BASIC auch dividieren, multiplizieren und sogar potenzieren kann ? Nun, dies wird durch ein Zurückführen dieser Rechenoperationen auf die vorhandenen Additions- und Subtraktionsbefehle ermöglicht. Oder einfacher: Jede nicht im Prozessorbefehlssatz vorgesehene (kompliziertere) Rechenoperation muß durch ein Programm, das auf Addition und Subtraktion zurückgreift, simuliert werden. Der BASIC-Interpreter hat derartige Routinen bereits eingebaut.

Im folgenden Programmbeispiel werden wir uns mit der einfachsten Form der Addition beschäftigen: Zwei im Arbeitsspeicher stehende Byte-Werte (denn größere Zahlen kann der 8502 nicht auf einmal addieren) sollen addiert und das Ergebnis im Speicher an Adresse $1412 abgelegt werden. Bitte tippen Sie die Kommentare (die werden üblicherweise durch ein Semikolon ";" gekennzeichnet) nicht ab ! Zum einen würde sich da der MONITOR beschweren, zum anderen würden diese Kommentare nicht abgespeichert werden; die ganze Arbeit wäre also für die Katz'.

```
A 1400 LDA $1410      ;an Adresse $1410 steht die erste Zahl
A 1402 CLC            ;eventuell noch vorhandenen Übertrag löschen
A 1403 ADC $1411      ;zum Akkumulator den Wert addieren, der an $1411
                      ;steht
A 1406 STA $1412      ;das Ergebnis wird bei $1412 abgespeichert
A 1409 BRK            ;BREAK - Abbruch (der MONITOR tritt wieder in
                      ;Aktion)
```

Bevor Sie das Programm mit "G 1400" starten, müssen Sie aber erst noch die beiden Zahlen im Speicher ablegen, damit sie der Prozessor beim Programmablauf dann auch findet. Mit "M 1400" können Sie sich einen Speicherauszug dieses Bereiches am Bildschirm ausgeben lassen. Ändern Sie dann die ersten beiden Bytes in der Zeile, die mit ">01410" beginnt, in "03" und "05" um (<RETURN> nicht vergessen). Diese Zeile muß dann so beginnen:

```
>01410 03 05 ...
```

Wenn Sie nun das Programm wie oben beschrieben starten und sich erneut einen Speicherauszug ausgeben lassen, wird das Ergebnis im dritten Byte-Wert zu finden sein:

```
>01410 03 05 08 ...
```

Der Befehl "CLC" (CLear Carry) löscht einen eventuell vorhandenen Übertrag (von einer vorhergehenden Rechenoperation). Dieses Carry-Flag ist in einem speziellen Register des Prozessors, dem Statusregister, zu finden. Es wird beispielsweise dann auf wahr (logisch 1) gesetzt, wenn bei einer Addition ein Wert entsteht, der nicht mehr in den Akkumulator paßt. Der Akkumulator kann den entstehenden Wert nicht mehr ganz aufnehmen und das Carry-Flag wird gesetzt; es kann dann für weitere Rechenoperationen verwendet werden. Deshalb heißt der Additionsbefehl auch "ADd with Carry", "Addiere mit Übertrag". Entsprechend dazu existiert ein Subtraktionsbefehl "SBC", "SuBtract with Carry". Achten Sie bitte darauf, daß dieser Befehl, falls Sie nicht das Ergebnis einer vorhergehenden Subtraktion mitverarbeiten wollen, ein gesetztes Carry-Flag verlangt. Sie erreichen dies durch den Befehl "SEC", "SEt Carry".

5.7 Der interne Aufbau des 8502

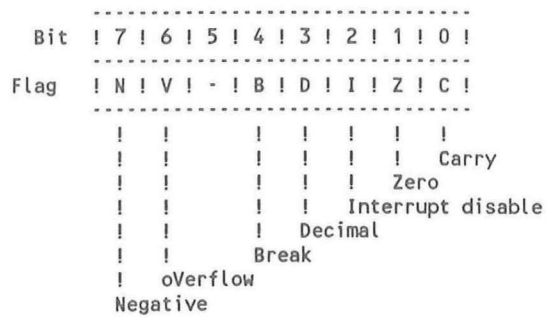
Um die bereits verwendeten Begriffe wie Akkumulator, Register oder Carry besser verstehen zu können, ist es nötig, etwas mehr in das Innenleben des 8502 vorzudringen.

Bereits in dem ersten Beispielprogramm in Assembler wurde ein Wert in das Akkumulator-Register des Prozessors geladen. Es sieht also fast so aus, als ob nicht nur die 64 kByte großen "Gedächtnisbänke" existieren, sondern auch noch ein prozessorinterner Merkzettel. Und dies ist noch lange nicht alles, denn im Prozessor wird die eigentliche "Denk"-Arbeit des Computers verrichtet. Sehen Sie sich einmal an, was so alles an Registern in diesem Prozessor drinsteckt:

Der *Akkumulator* ist das einzige Register, mit dem der 8502 Rechenoperationen durchführen kann. Wie bereits zuvor erwähnt, können damit Additionen und Subtraktionen erledigt werden; doch auch nach links und rechts schieben und sogar im Kreis rotieren lassen kann man die 8 Bits im Akkumulator. Eine besondere Stellung nehmen die logischen Befehle ein, mit denen einzelne Bits des Akkumulators gezielt gesetzt und gelöscht werden können.

X und *Y* können - wie auch der Akkumulator - ein Byte aufnehmen. Sie werden benutzt als Schleifenzähler, zur Zwischenspeicherung von Akkumulatorinhalten und - besonders wichtig - zur einfacheren Bearbeitung hintereinanderliegender Speicherstellen (indizierte Adressierung).

Das *Statusregister* gibt darüber Auskunft, was dem Prozessor gerade erlaubt oder verboten ist und auch über Besonderheiten, die durch eine Befehlsausführung eingetreten sein können (z. B. Setzen des Carry-Bits). Es kann abgefragt werden und dann Grundlage für Entscheidungen sein, die zu einer Programmverzweigung (wie in BASIC: IF...THEN...GOTO) führen. Dem Statusregister oder auch Prozessorregister liegt folgender Aufbau zugrunde:



Mit dem *Stapelzeiger* (Stackpointer) wird der für den Prozessor reservierte Datenhaufen - der Stack oder Stapel - verwaltet. Einen Datentyp nennt man dann Stapel, wenn man immer nur Zugriff auf das oberste Element des Stapels hat: immer nur obendrauf legen und wieder von oben 'runternehmen ("last in - first out" oder "LIFO"). Dieser Datenbereich liegt im Speicher zwischen $0100 und $01FF. Ist der Stapel leer, so enthält der Stapelzeiger den Wert $FF und würde das erste auf den Stapel gebrachte Byte an der Adresse $1FF ablegen. Mit zunehmender Stapelfüllung schrumpft der Stapelzeiger nach unten. Übrigens gibt es noch einen Bereich, ganz "in der Nähe" des Stack, der vom Prozessor besonders behandelt wird: die Zeropage (Seite 0 im Adreßraum, von $0000 bis $00FF). Mit den speziellen Zeropage-Befehlen - das erste Adreßbyte hat den Wert $00 und wird bei der Adreßangabe des Maschinenbefehls weggelassen - ist eine schnellere Befehlsausführung als im restlichen Speicher möglich.

Im *Programmzähler* (Program Counter) schließlich "merkt" sich der Prozessor, an welcher Speicherstelle er gerade mit der Abarbeitung eines Maschinenbefehls beschäftigt ist. Damit Maschinenprogramme durchgeführt werden können, die im gesamten Speicherbereich von 64K einer Bank liegen, muß der Program Counter 2 Byte (16 Bit) fassen können.

Zur Verdeutlichung der Verwendung dieser Register sollten Sie sich folgende Programmbeispiele zu Gemüte führen.

```
1400 LDY #$00       ;Indexregister Y mit dem Wert $00 laden
1402 LDA $0400,Y    ;lade den Akkumulator mit dem Wert der
                    ;Speicherstelle $400 plus dem Inhalt des Registers Y
1405 ORA #$80       ;setze höchstwertiges Bit im Akkumulator
1408 STA $0400,Y    ;speichere den Akkumulator wieder dort ab
140B INY            ;erhöhe den Wert des Registers Y um 1
140C CPY #$50       ;hat Y schon den Wert $50 erreicht ?
140E BNE $1402      ;wenn nicht, dann springe wieder zu $1402
1410 BRK            ;Programmende
```

Daß dieses Programm wieder irgendwelche Manipulationen am Bildschirmzeichenspeicher durchführt, haben Sie sicher schon erkannt. Und allzuviel kann mit den Zeichen, die da geladen und wieder abgespeichert werden, eigentlich nicht geschehen. Lassen Sie sich also überraschen!

Einige "neue" Befehlstypen wurden hier verwendet. ORA beispielsweise ist ein Befehl aus der Gruppe der logischen Befehle, der auch AND und EOR, das exklusive Oder zugehören (dieses gibt es auch im BASIC 7.0 und nennt sich da XOR; mehr dazu im Anhang A). INY, INX und INC erhöhen den Wert von X-, Y-Register oder Speicherbytes um 1; DEY, DEX und DEC bewirken genau das Gegenteil. CPY, CPX und CMP vergleichen die Regi-

ster mit einem Wert und setzen, entsprechend dem Ergebnis des Vergleiches, einige Bits im Statusregister. Das Entscheidende an diesem Programm ist jedoch, daß eine Schleife eingebaut ist. Erreicht wird dies mit einem Sprung an die Adresse $1402; verantwortlich hierfür zeichnet die Instruktion BNE ("Branch if Not Equal" oder "Branch if Not Equal to zero"). Alle Befehle aus der "Branch..."-Gruppe BCC, BCS, BEQ, BNE, BPL, BMI, BVC und BVS ziehen das zugehörige Bit aus dem Statusregister zur Entscheidung heran, ob der Sprung gewagt werden soll oder nicht.

Weil der Bildschirm ein dankbares Demonstrationsobjekt für erste Schritte in Assemblersprache ist, soll auch das nächste Programm dort etwas verändern. Allerdings wird diesmal der Farbspeicher des 40-Zeichen-Bildschirms in Angriff genommen. Sie wissen ja, daß man diesen nur in bestimmten Speicherkonfigurationen erreichen kann. Folglich muß das Programm entweder mit dem MONITOR und einem angepaßten Adreßteil des "G"-Befehls (beispielsweise mit "G 41400") aufgerufen werden oder sich selbst um die korrekte Speicherzusammenstellung kümmern. Die erste Möglichkeit ist die einfachere Variante:

```
1400  LDA $A2       ;Byte-Wert aus Systemuhr in Akkumulator laden
1401  LDY #$00      ;Indexregister vorbesetzen
1403  STA $D800,Y   ;Akkumulator indiziert abspeichern
1406  INY           ;Indexregister Y um 1 erhöhen
1407  CPY #$50      ;ist in Y schon der Wert $50 erreicht ?
1409  BNE $1403     ;Nein? dann springe zu Adresse $1403
140B  BRK           ;Programmende
```

Auch diesmal wird nicht verraten, was beim Programmaufruf passiert; aber sicher haben Sie das schon längst erraten. Starten Sie das Programm ruhig mehrmals, denn es wird nicht immer das gleiche tun.

5.8 Warum wechseln Sie denn nicht einfach die Bank ?

Im vorigen Beispiel hat der MONITOR für die passende Speicherkonfiguration des Programmablaufes gesorgt. Jetzt soll das unter der Regie unseres Programmes durchgeführt werden. Dies ist dann zwar nicht der übliche Weg, den man beschreiten würde, wenn man unter der Obhut des Betriebssystems in die gewünschte Konfiguration gelangen möchte; aber es ist der schnellste Weg - und ganz nebenbei wird dann auch deutlich, wie die MMU angesteuert und für eigene Zwecke mißbraucht werden kann. Dazu ist es wichtig, zu wissen, wo im Speicher der gerade aktiven RAM-Bank (denn nur dort kann der Prozessor momentan arbeiten) eine Steuerungsmöglichkeit der MMU besteht. In jeder Speicherkonfiguration (außer im

64er-Modus, und der zählt nicht) ist die MMU an den Speicheradressen $FF00 bis $FF04 ansprechbar. Besonderes Interesse verdient die Adresse $FF00, das "Configuration Register" der MMU:

Configuration Register	
Bits	Bedeutung
7,6	RAM-Bank-Auswahl 00 = Bank 0 01 = Bank 1 10 = Bank 2 11 = Bank 3
5,4	Speicherbereich $C000-$FFF 00 = SYSTEM ROM (KERNAL, Char. ROM - I/O) 01 = INTERNAL FUNCTION ROM (hi) 10 = EXTERNAL FUNCTION ROM (hi) 11 = RAM
3,2	Speicherbereich $8000-$BFFF 00 = BASIC ROM (hi) 01 = INTERNAL FUNCTION ROM (lo) 10 = EXTERNAL FUNCTION ROM (lo) 11 = RAM
1	Speicherbereich $4000-$7FFF 0 = BASIC ROM (lo) 1 = RAM
0	Speicherbereich $D000-$DFFF 0 = I/O 1 = RAM/ROM (abhängig von Bits 4 und 5)

Die Abkürzungen "hi" und "lo" sind keine exotischen Begrüßungsformeln. Sie beziehen sich auf Adreßangaben. Adressen sind 16-Bit große Werte, die man also in zwei Bytes zu 8 Bit abspeichern muß. Man spricht in diesem Zusammenhang von einem höherwertigen Teil der Adressangabe oder "High-Byte" und dem niederwertigen oder "Low-Byte". Weil das aber beinahe jeder verstehen würde, benutzen die eingefleischten Assembler-freaks die Abkürzungen "hi" und "lo".

Die Register an den Adressen $FF01 bis $FF04 sind sogenannte "Load Configuration Register", deren Bedeutung später beschrieben werden soll.

Und nun ans Werk: Programmgesteuerter Speicherkonfigurationswechsel ohne auf Betriebssystemroutinen oder ausgelagerte Teile davon zurückzugreifen. Wichtig ist, bevor ein neuer Speicher konfiguriert wird, sich den alten Wert zu merken. Der neue Wert, der an der Adresse $FF00 abgelegt

werden soll, muß im folgenden Beispiel den I/O-Bereich (in dem der Farbspeicher ansprechbar ist) einblenden und die gleiche RAM-Bank wählen, in der das Programm abgespeichert ist (andernfalls könnte sich der Computer "aufhängen").

Folglich könnte eine Bitkombination wie z.B. %00000000 (= $00) für diese Zwecke geeignet sein; kontrollieren Sie das ruhig anhand der letzten Tabelle.

```
1400  LDA $FF00      ;gerade gültige Konfiguration laden
1403  PHA            ;und auf den Stack damit
1404  LDA #$00       ;neue Konfiguration laden
1406  STA $FF00      ;und aktivieren
                     ;
1409  LDA $A2        ;Byte-Wert aus Systemuhr in Akkumulator laden
140B  LDY #$00       ;Indexregister vorbesetzen
140D  STA $D800,Y    ;Akkumulator indiziert abspeichern
1410  INY            ;Indexregister Y um 1 erhöhen
1411  CPY #$F0       ;ist in Y schon der Wert $F0 erreicht?
1413  BNE $140C      ;Nein? dann springe zu Adresse $140D
                     ;
1415  PLA            ;alte Konfiguration vom Stapel nehmen
1416  STA $FF00      ;und aktivieren
1419  BRK            ;Programmende
```

Falls Sie das Programm mit "G 01400" gestartet hatten, werden Sie nach Ablauf des Programmes im Akkumulator den Wert $3F finden. Dies entspricht genau der Speicherkonfiguration mit dem Index 0. Für Interessierte: Die 16 Konfigurationsbytes des Betriebssystems befinden sich an den Adressen $F7F0 bis $F7FE (zum Ansehen nicht vergessen, das Betriebssystem auch einzublenden!). Natürlich ist diese Konfiguration nicht mehr aktiv, sobald der MONITOR das Kommando wieder übernimmt.

PHA und PLA dienen zur Zwischenspeicherung des Akkumulatorinhalts im Stack. Außer PHP und PLP, mit denen der Prozessorstatus (das Statusregister) auf den Stack gelegt und vom Stack genommen werden kann, sind dies die einzigen Befehle, mit denen Zwischenspeicherung direkt realisiert werden kann; jedoch ist es möglich, den Stackpointer selbst zu modifizieren: der Stapelzeiger kann mit TSX ("Transfer S into X") in das X-Register übertragen, verändert und durch TXS ("Transfer X into S") neu gesetzt werden. Und auch die Instruktionen JSR ("Jump to SubRoutine", wie GOSUB), RTS ("ReTurn from Subroutine", entspricht RETURN), und RTI ("ReTurn from Interrupt") arbeiten mit dem Stack.

Programmgesteuerte Änderung der Speicherung können Sie sich sparen, wenn Sie Ihre Maschinencode-Routinen von BASIC aus aufrufen. Mit Hilfe des BANK-Befehls ist eine Voreinstellung der Konfiguration leicht

möglich; das Maschinenprogramm kann dann mit dem BASIC-Befehl SYS aufgerufen werden. Wichtig ist, daß an die Stelle des BRK-(Break-)Befehls als definiertes Ende der Maschinenroutine jetzt RTS tritt.

Noch ein Tip: Setzen Sie doch - versuchsweise - an die Stelle der BRK-Instruktion den Befehl JMP $1400. Dann werden Sie diese Endlosschleife zwar nur noch mit <RUN/STOP>/<RESTORE> oder drücken der RESET-Taste verlassen können, sich aber umso besser die Geschwindigkeit verdeutlichen können, mit der Maschinenprogramme ablaufen.

5.9 Unterprogramme in Assemblersprache

Viel gibt es dazu nicht zu sagen; dem BASIC-Programmierer wird diese Technik schon so geläufig sein, daß er damit umzugehen weiß und die vielen Vorzüge bereits in seinen eigenen Programmen zu schätzen gelernt hat. In der Assemblersprache des 8502 kann sie genauso einfach eingesetzt werden:

```
1400  LDY #$00       ;Indexregister Y vorbesetzen
1402  LDA $0400,Y    ;Akkumulator indiziert laden
1405  JSR $141A      ;Unterprogrammaufruf
1408  STA $0400,Y    ;Akkumulator indiziert abspeichern
140B  LDA $04F0,Y    ;Akkumulator indiziert laden
140E  JSR $141A      ;Unterprogrammaufruf
1411  STA $04F0,Y    ;Akkumulator indiziert abspeichern
1414  INY            ;Indexregister um 1 hochzählen
1415  CPY #$F1       ;hat Y schon den Endwert überschritten ?
1417  BNE $1402      ;wenn nicht, dann springe zu $1402
1419  BRK            ;Programmende
                     ;
                     ;Unterprogramm
141A  BMI $1421      ;ist das höchstwertige Bit im Akkumulator gesetzt ?
                     ;dann springe nach $1421
141C  ORA #$80       ;sonst setze das höchstwertige Bit
141E  JMP $1423      ;und springe auf jeden Fall nach $1423
1421  AND #$7F       ;erstes Bit im Akkumulator löschen
1423  RTS            ;Rückkehr ins aufrufende Programm
```

Das Angenehme an Unterprogrammen ist einerseits die Möglichkeit, diese getrennt vom aufrufenden Programmteil auzutesten und andererseits, daß man sich häufig wiederholende Abläufe, anstatt sie jedesmal explizit auszuformulieren, bequem in einer Routine unterbringen kann.

Durch JSR wird der momentane Inhalt des Program Counters (die Adresse des dem JSR folgenden Befehls minus 1) auf den Stapel gebracht: Zuerst das höherwertige Byte des PC, dann das niederwertige. Die Instruktion RTS

lädt den Program Counter mit den beiden auf dem Stack zuoberst liegenden Bytes (und addiert 1 dazu). Somit ist es auch möglich, zwei Bytes aus dem Akkumulator auf den Stack zu "pushen", die eine Speicheradresse darstellen und mit anschließendem RTS mit der Progammausführung an dieser Adresse fortzufahren. Zusätzlich zum (vor dem) Program Counter holt sich der Befehl RTI auch noch den Prozessorstatus vom Stack; mit den Return-Instruktionen sind so leistungsfähige, elegante, aber auch verwirrende Programmiertechniken möglich.

5.10 Interruptverarbeitung oder: Allzeit bereit

Das Stichwort "Interrupt" (Unterbrechung) haben Sie ja bereits im Zusammenhang mit dem Stack und dem Befehl RTI gelesen. Unterbrechungsanforderungen - im allgemeinen von externen Einheiten bzw. Ein-/Ausgabegeräten - sind es, die den Prozessor, der mit seiner "normalen Arbeit" beschäftigt ist, dazu zwingen (falls es ihm gestattet ist), auf die Anforderung zu reagieren. Der Befehl, der gerade in Bearbeitung ist, wird noch vollständig ausgeführt und eine Interruptbehandlung wird eingeleitet.

Das Arbeiten mit Interrups weist gewisse Ähnlichkeiten mit dem Aufstellen von Fehlerfallen im BASIC 7.0 auf (s. TRAP-Befehl im Anhang A). Auch hier wird eine Routine zur Behandlung einer Programmunterbrechung angesprungen und nach Beendigung mit RESUME wieder verlassen. Im Programm geht's dann hinter dem Fehler weiter.

Ähnlich in Maschinensprache: Nach der Behandlung des Interrupts wird - zumindest in den meisten Fällen - mit der Ausführung bei dem Befehl fortgefahren, der im Normalfall, also ohne Unterbrechung abgearbeitet worden wäre. Bis auf den NMI, den "Non-Maskable Interrupt", können Interrupts durch Setzen des Interrupt-Bits im Statusregister verboten werden (Befehle SEI/CLI; "SEt/CLear Interrupt mask").

Interrupts beanspruchen übrigens Platz auf dem Stack; denn sowohl der Program Counter als auch Statusregister müssen ja, damit eine korrekte Weiterbearbeitung des unterbrochenen Programms möglich wird, zwischengespeichert werden (dies geschieht bei jedem Interrupt automatisch). Sinnvoll ist es jedoch, auch die übrigen Register (A, X, Y) auf dem Stack zu retten, damit der ursprüngliche Zustand vor der Unterbrechung wieder vollkommen hergestellt werden kann.

Am besten kann man sich die Interruptverarbeitung am Beispiel des durch das Betriebssystem unterstützten und alle 1/60 sec. autretenden Interrupts verdeutlichen, der durch den Timer eines im C128 eingebauten Chips er-

zeugt wird. An den Adressen $0314 und $0315 befindet sich eine Adreßangabe, ein Vektor oder Zeiger, der angibt, wo der Prozessor beim Eintreten eines derartigen Interrupts mit der Programmausführung weitermachen soll. Dieser Zeiger wird im nächsten Beispiel "verbogen". Er zeigt anschließend auf die Adresse, an der unsere Routine, die dann auch 60mal in der Sekunde durchlaufen wird, beginnt. Die Verbiegung muß nur einmal stattfinden; es ist also eine Initialisierung des Vektors und vor allen Dingen ein "Retten" des ursprünglichen Zeigers nötig, da die Arbeit des Betriebssystems auch noch ausgeführt werden muß:

```
                    ;Initialisierungsroutine
1400  SEI           ;Interrupts, die diese Routine stören könnten, verbieten
1401  LDX $0314     ;low-Byte des Vektors ins Register X
1404  LDY $0315     ;hi-Byte des Vektors ins Register Y
1407  STX $141A     ;low-Byte des Vektors retten
140A  STY $141B     ;hi-Byte des Vektors retten
140D  LDX #$20      ;low-Byte des neuen Vektors
140F  LDY #$14      ;hi-Byte des neuen Vektors
1411  STX $0314     ;neuen Interrupt-Vektor
1414  STY $0315     ;abspeichern
1417  CLI           ;Interrupts wieder zulassen
1418  BRK           ;Ende der Initialisierung

                    ;eigentliche Routine
1420  LDA $A1       ;niederwertigstes Byte der Systemuhr laden
1422  LSR           ;LSR schiebt den Akkumulator um 1 Bit nach rechts
1423  LSR           ;dies entspricht dem "Teilen durch 2"
1424  LSR           ;
1425  LSR           ;insgesamt also durch 16 geteilt
1426  STA $D020     ;im Speicher für die Bildschirmrandfarbe ablegen
1429  JMP ($141A)   ;mit der normalen Bearbeitung fortfahren
```

Mit dem ersten Durchlaufen der Initialisierung bei $1400 wird die Routine ab $1420 aktiviert. Versuchen Sie nicht, die Initialisierung bei $1400 ein zweites mal aufzurufen (es sei denn, Sie wollen Ihren Computer "in den Wald schicken" oder hatten ihn inzwischen mit <RUN/STOP>/<RESTORE> in den Grundzustand versetzt); andernfalls würden sich die Vektoren bei $0314/$0315 und $141A/$141B gleichen, dadurch eine Schleife entstehen und alle 1/60 Sekunde 3 neue Stack-Plätze beansprucht werden. Und ruckzuck ist der Stack dann voll.

5.11 Call me! - das Betriebssystem

Zum Abschluß dieser Einführung in die Assemblerprogrammierung mit dem C128 soll das Betriebssystem dazu benutzt werden, wozu es eigentlich gedacht ist: Zur Vereinfachung von Assemblerprogrammen durch den Aufruf seiner Ein/Ausgaberoutinen. Eine ganz einfache Aufgabenstellung ist

die Ausgabe eines im Arbeitsspeicher befindlichen Textes mit Hilfe einer Betriebssystemroutine. Vergessen Sie nicht, auch diesmal für die korrekte Speicherkonfiguration zu sorgen, bevor das Programm beginnt (z. B. "G E1400" im MONITOR). Speichern Sie auch (mit dem "M"-Befehl) ab Adresse $1420 den Text ab, der ausgegeben werden soll. Schließen Sie den Text mit einem Nullbyte ($00) ab, damit in der Routine erkannt werden kann, wo der Text zu Ende ist.

```
1400  LDY #$00       ;Indexregister vorbesetzen
1402  LDA $1420,Y    ;ein auszugebendes Zeichen in den Akkumulator
1405  CMP #$00       ;ist es ein Nullbyte ?
1407  BEQ $140F      ;dann gehe zu $140F
1409  JSR $FFD2      ;Aufruf der Bildschirmausgaberoutine,
                     ;auszugebendes Zeichen im Akkumulator
140C  INY            ;Indexregister um 1 hochzählen
140D  BNE $1402      ;weiter, wenn Maximallänge noch nicht
                     ;überschritten
140F  BRK            ;Ende der Routine
```

6 Wissenswertes über die MMU

Über einen Bereich, in dem Register der MMU zu finden sind, haben Sie bereits etwas erfahren: Die Speicherstellen $FF00 bis $FF04. Es existiert jedoch noch ein weiterer "Zugang" zur MMU, nämlich die Adressen $D500 bis $D50B; allerdings nur, wenn der I/O-Bereich ($D500-$DFFF) in der aktuellen Speicherkonfiguration eingeblendet ist. Im anderen Fall, also abgeschalteter I/O-Bereich (oder C64 aktiv), kann die MMU nicht über diese Adressen erreicht werden.

$FF04	LCRD	Load Configuration Register D
$FF03	LCRC	Load Configuration Register C
$FF02	LCRB	Load Configuration Register B
$FF01	LCRA	Load Configuration Register A
$FF00	CR	Configuration Register

$D50B	VR	Version Register
$D50A	P1H	Page 1 Pointer High
$D509	P1L	Page 1 Pointer Low
$D508	P0H	Page 0 Pointer High
$D507	P0L	Page 0 Pointer Low
$D506	RCR	RAM Configuration Register
$D505	MCR	Mode Configuration Register
$D504	PCRD	Preconfiguration Register D
$D503	PCRC	Preconfiguration Register C
$D502	PCRB	Preconfiguration Register B
$D501	PCRA	Preconfiguration Register A
$D500	CR	Configuration Register

6.1 Configuration Register

Das Configuration Register an Adresse $D500 entspricht, was die Bedeutung der einzelnen Bits betrifft, genau dem bereits im letzten Kapitel ausführlich beschriebenen Configuration Register bei $FF00. Der besseren Übersicht hier noch einmal dessen Belegung:

Configuration Register	
Bits	Bedeutung
7,6	RAM-Bank-Auswahl 00 = Bank 0 01 = Bank 1 10 = Bank 2 11 = Bank 3
5,4	Speicherbereich $C000-$FFF 00 = SYSTEM ROM (KERNAL, Char. ROM - I/O) 01 = INTERNAL FUNCTION ROM (hi) 10 = EXTERNAL FUNCTION ROM (hi) 11 = RAM
3,2	Speicherbereich $8000-$BFFF 00 = BASIC ROM (hi) 01 = INTERNAL FUNCTION ROM (lo) 10 = EXTERNAL FUNCTION ROM (lo) 11 = RAM
1	Speicherbereich $4000-$7FFF 0 = BASIC ROM (lo) 1 = RAM
0	Speicherbereich $D000-$DFFF 0 = I/O 1 = RAM/ROM (abhängig von Bits 4 und 5)

6.2 Preconficuration Register

Die Preconfiguration Register dienen zum Voreinstellen im Programmablauf benötigter Konfigurationen. In ihnen können die Konfigurationsbytes abgelegt werden, die durch (irgendeinen) Schreibzugriff auf das entsprechende Load Configuration Register dann aktiviert werden. Ein Auslesen der Load Configuration Register liefert übrigens den Wert des im zugehörigen Preconfiguration Register abgespeicherten Konfigurationsbytes.

6.3 Mode Configuration Register

Welche CPU und welche Betriebsart (C64, C128 oder Z80 aktiv) eingeschaltet sind, kann mit dem Mode Configuration Register eingestellt werden. Ihm liegt folgender Aufbau zugrunde:

Mode Configuration Register	
Bits	Bedeutung
7	Stellung des 40/80-Zeichen-Schalters 0 = Schalter geschlossen 1 = Schalter offen
6	Betriebsart 0 = C128 1 = C64
5	Bidirektionaler Port Eingang : /EXROM Ausgang : Farb-RAM-Bank während VIC aktiv
4	Bidirektionaler Port Eingang : /GAME Ausgang : Farb-RAM-Bank während CPU aktiv
3	FSDIR (Fast ser. disk) Kontrollbit Eingang : Fast serial disk enable Ausgang : für FSD-Hardware
2,1	Nicht benutzt
0	CPU-Einstellung 0 = Z80 1 = 8502

6.4 RAM Configuration Register

Unter Verwendung des RAM Configuration Registers ist es möglich, Speicherbereiche zu definieren, die unabhängig von der aktuellen RAM-Bank auf den Arbeitsspeicher der RAM-Bank 0 zugreifen. Interessant: Die Möglichkeit, dem VIC verschiedene RAM-Bänke unterzuschieben (bei einer Erweiterung auf 1 MByte voraussichtlich nicht möglich).

RAM Configuration Register	
Bits	Bedeutung
7,6	RAM-Bank des VIC (Bit 7 für Erweiterung) x0 = RAM-Bank 0 x1 = RAM-Bank 1
5,4	Nicht benutzt (für Erweiterung auf 1 MByte)
3,2	Lage der gemeinsamen Speicherbereiche 00 = keine gemeinsamen Speicherbereiche 01 = Speicheranfang 10 = Speicherende 11 = Speicheranfang und -Ende
1,0	Größe des/der gemeinsamen Speicherbereiche 00 = 1 KByte 01 = 4 KByte 10 = 8 KByte 11 = 16 KByte

6.5 Page Pointer

Mit den Page Pointer Registern wurde im C128 ein Verlagern von Zeropage und Stack (normalerweise bei $0000..$00FF bzw. $0100..$01FF) in beliebige andere Speicherseiten ermöglicht. Dadurch wächst der Speicherbereich mit der schnellen Zeropage-Adressierung und auch der Datenbereich des Stapels (bei entsprechender Verwaltung) beträchtlich an. Eine Neubesetzung der Page Pointer muß immer zuerst am höherwertigen Byte erfolgen.

Page Pointer, höherwertiges Byte	
Bits	Bedeutung
7,6,5,4	ohne Bedeutung
3,2,1,0	gewählte Bank (auch ROM möglich !)

Page Pointer, niederwertiges Byte	
Bits	Bedeutung
7 bis 0	Speicherseite innerhalb der 64K-Bank

6.6 Version Register

Zuguterletzt sei noch das Version Register beschrieben, das einzige Register, aus dem für Betriebssoftware die Größe des verfügbaren Speichers ersichtlich ist. Außerdem ist hier auch noch die Versionsnummer der MMU untergebracht.

System Version Register	
Bits	Bedeutung
7,6,5,4	verfügbare 64K-Bänke
3,2,1,0	Versionsnummer der MMU

Nicht ganz einfach scheint es, die durch den "Wunderbaustein" MMU entstehenden Möglichkeiten zu übersehen oder sie sogar sinnvoll einzusetzen. Doch jede Frage verlangt auch nach einer Antwort; und die wird man wohl erst dann finden, wenn man sich lange und ausgiebig mit dem C128 beschäftigt hat.

7 Hicks und Hacks

7.1 Grafik auf dem 80-Zeichen-Bildschirm

Sind Sie glücklicher Besitzer eines Monitors, mit dem Sie das 80-Zeichen-Feature des C128 nutzen können? Ärgern Sie sich auch darüber, daß es nicht möglich ist, die leistungsfähigen Grafik-Befehle des C128 auch für den viel besseren Monitor zu nutzen?

Vielleicht wissen Sie noch gar nicht, daß der Videocontroller, der für die 80-Zeichen-Darstellung verantwortlich ist, voll grafikfähig ist? Ja, wirklich! Mit einer Auflösung von 640x200 Pixels! Ganz nett, gell?

Leider, leider sind - wie oben bereits erwähnt - die Grafikbefehle hierfür nicht zu gebrauchen. Deshalb sollen hier Sie wenigstens das wichtigste Handwerkszeug für die Grafikprogrammierung des 8563 (so heißt der schlaue Bursche mit den 80 Zeichen in der Tasche) präsentiert bekommen.

Sie werden sich wahrscheinlich sagen: "Ja, wenn das so ist, brauche ich ja nur in den Zeichenspeicher dieses 8563 zu POKE'n...". Weit gefehlt!

Die einzigen Verbindungspunkte zwischen dem 8563 und den restlichen Computerteilen sind die Adressen $D600 und $D601 (Vorsicht: I/O-Bereich!). Durch diese beiden "Register" müssen Sie Ihre Daten für und Kommandos an den VDC schleusen. Dieser verfügt über einen eigenen, nur ihm zugänglichen Speicher von 16 KByte. Dort speichert er im Textmodus (Textmodus? - Ja, es gibt auch noch einen Grafikmodus) einen ganzen Haufen ab:

25*80 Bytes Zeichen (Zeiger auf den Zeichendefinitionsbereich)

25*80 Bytes Zeichenattribute (Farbe,Unterstreichung,...)

8 KByte Definitionsbereich für Zeichen.

Die ersten beiden Angaben sind noch einleuchtend, denn so ähnlich ist es beim VIC ja auch. Doch wozu extra noch einmal die Zeichen definieren, da das CHARACTER ROM doch schon einmal bei $D000-$DFFF existiert?

Nun - der 8563 wird eben nur über die bereits zuvor erwähnten Adressen versorgt und hat, im Gegensatz zum VIC, der eigentlich als Coprozessor fungiert, nicht die Möglichkeit des direkten Zugriffs auf das CHARACTER ROM. Deshalb bekommt er beim Anschalten die Zeichendefinitionen durch den Engpaß bei $D600 und $D601 Byte für Byte eingetrichtert.

Versuchen Sie doch einmal einen Zeichensatzwechsel (mit der Taste <ASCII/DIN>) und beobachten Sie die sich ändernden Zeichen genau:

Sie werden feststellen, daß sich einige Zeichen früher "im neuen Kleid" zeigen als andere. Das liegt ganz einfach daran, daß durch den Engpaß eine Verzögerung entsteht und deshalb manche Zeichen eher auf die neue Definition zugreifen können als andere.

Den 16K-Speicher interpretiert der 8563, schaltet man den Grafikmodus ein, als "Bit-Mapped"-Grafik. Das heißt, jedes gesetzte Bit leuchtet und jedes nicht gesetzte bleibt dunkel. 16K mal 8 Bit ergibt über den Daumen gepeilt 128000 Bildpunkte. Und die gilt es erst einmal zu löschen (oder wollen Sie gerne auf bereits vollgeschmiertem Papier malen?). In BASIC würde das Löschen dieses Speichers viel zu viel Zeit in Anspruch nehmen (Achtung: Beim Löschen werden auch die Zeichendefinitionen überschrieben; Notlösung zur Wiederherstellung der Buchstaben: <ASCII/DIN> drücken). Deshalb sollten Sie folgendes "Werkzeug" in Ihre Grafik-Utilities-Sammlung für den 8563 aufnehmen, mit dem Sie - neben dem Löschen des gesamten Bildschirms auch noch einzelne Punkte setzen oder löschen können:

```
1400 JMP $141E      ;Grafikmodus anschalten
1403 JMP $1426      ;Grafikmodus ausschalten
1406 JMP $1487      ;80-Zeichen-Bildschirm löschen
1409 JSR $FF62      ;Zeichensatz neu laden
140C JMP $FF81      ;Editor initialisieren
140F JMP $14A3      ;Punkt setzen
1412 JMP $149D      ;Punkt löschen
1415 JMP $????      ;frei für Erweiterungen
1418 JMP $????      ;frei für Erweiterungen
141B JMP $????      ;frei für Erweiterungen
                    ;
                    ;Grafikmodus anschalten
141E LDA #$80       ;Bit 7 setzen
1420 LDX #$19       ;Register-Nr. 25
1422 JSR $CDCC      ;VDC-Register 25 mit $80 besetzen
1425 RTS
```

```
                        ;
                        ;Grafikmodus ausschalten
1426 LDA #$40           ;Bit 6 setzen
1428 LDX #$19           ;Register-Nr. 25
142A JSR $CDCC          ;VDC-Register 25 mit $40 besetzen
142D RTS
                        ;
                        ;Update-Adresse setzen (in X und Y)
142E LDA #$12           ;Register 18
1430 STA $D600
1433 STX $D601
1436 JSR $1445          ;Warten auf Statusbit
1439 LDA #$13           ;Register 19
143B STA $D600
143E STY $D601
1441 JSR $1445          ;Warten auf Statusbit
1444 RTS
                        ;
                        ;Warten bis Statusbit gesetzt
1445 BIT $D600
1448 BPL $1445
144A RTS
                        ;
                        ;Warten bis Statusbit gelöscht
144B BIT $D600
144E BMI $144B
1450 RTS
                        ;
                        ;Wortzähler Null setzen
1451 LDA #$1E
1453 STA $D600
1456 JSR $1445
1459 LDA #$00
145B STA $D601
145E RTS
                        ;
                        ;Datenbyte Null setzen
145F LDA #$1F
1461 STA $D600
1464 JSR $1445
1467 LDA #$00
1469 STA $D601
146C RTS
                        ;
                        ;Datenbyte vom VDC holen
146D LDA #$1F
146F STA $D600
1472 JSR $1445
1475 LDA $D601
1478 RTS
                        ;
                        ;Datenbyte im VDC ablegen
1479 PHA
147A LDA #$1F
147C STA $D600
147F JSR $1445
1482 PLA
1483 STA $D601
1486 RTS
```

```
                    ;
                    ;Bildschirm löschen
1487 LDX #$00
1489 LDY #$00
148B JSR $142E      ;Update-Adresse setzen
148E JSR $1451      ;Wortzähler Null setzen
1491 JSR $145F      ;Datenbyte Null setzen
1494 INY
1495 BNE $148B
1497 INX
1498 CPX #$40
149A BNE $148B
149C RTS
                    ;
                    ;Punkt löschen
149D PHA
149E LDA #$00       ;Flag für Löschen
14A0 JMP $14A6
                    ;
                    ;Punkt setzen
14A3 PHA
14A4 LDA #$FF       ;Flag für Setzen
14A6 STA $C3        ;Flag zwischenspeichern
14A8 PLA
14A9 JSR $14DB      ;Adreßberechnung A, X, Y --> $C1, $C2, A
14AC BCS $149C      ;Angabe außerhalb des Bereiches
14AE STA $C4        ;Bitmaske zwischenspeichern
14B0 LDY $C1
14B2 LDX $C2
14B4 JSR $142E      ;Update-Adresse setzen
14B7 JSR $146D      ;Datenbyte vom VDC holen -> A
14BA PHA
14BB LDA $C3
14BD BEQ $14C5      ;löschen: zu $14C3
14BF PLA
14C0 ORA $C4        ;Datenbyte mit Bitmaske ODERn
14C2 JMP $14CE      ;zum Abspeichern
14C5 PLA
14C6 STA $C3        ;Datenbyte zwischenspeichern
14C8 LDA $C4        ;Bitmaske
14CA EOR #$FF       ;alle Bits umdrehen
14CC AND $C3        ;mit Datenbyte UNDen
14CE PHA
14CF LDY $C1
14D1 LDX $C2
14D3 JSR $142E      ;Update-Adresse setzen
14D6 PLA
14D7 JSR $1479      ;Datenbyte im VDC ablegen
14DA RTS
                    ;
                    ;Adreß-Berechnung A, X, Y --> $C1, $C2, A
14DB CMP #$03       ;zurück, wenn A >= 3
14DD BCS $1534
14DF CMP #$02       ;weiter, wenn A < 2
14E1 BNE $14E7
14E3 CPX #$80       ;X - Koordinate zu groß?
14E5 BPL $1534
14E7 CPY #$C8
14E9 BCS $1534
```

```
14EB PHA
14EC TXA
14ED PHA
14EE TYA
14EF AND #$0F       ;Index für Tabelle 1 und 2
14F1 TAX
14F2 LDA $1540,X
14F5 STA $C1
14F7 LDA $1550,X
14FA STA $C2
14FC TYA
14FD AND #$F0
14FF LSR
1500 LSR
1501 LSR
1502 LSR            ;Index für Tabelle 3
1503 TAX
1504 LDA $1560,X
1507 CLC
1508 ADC $C2
150A STA $C2
150C PLA
150D TAX
150E PLA
150F TAY
1510 LDA $1570,Y    ;Wert aus Tabelle $
1513 CLC
1514 ADC $C1
1516 STA $C1
1518 BCC $151C
151A INC $C2
151C TXA
151D AND #$F8
151F LSR
1520 LSR
1521 LSR
1522 CLC
1523 ADC $C1
1525 STA $C1
1527 BCC $152B
1529 INC $C2
152B TXA
152C AND #$07
152E TAX
152F LDA $1573,X
1532 CLC
1533 RTS
1534 SEC
1535 RTS

>01540 00 50 A0 F0 40 90 E0 30
>01548 80 D0 20 70 C0 10 60 B0
>01550 00 00 00 00 01 01 01 02
>01558 02 02 03 03 03 04 04 04
>01560 00 05 0A 0F 14 19 1E 23
>01568 28 2D 32 37 3C 41 46 00
>01570 00 20 40 80 40 20 10 08
>01578 04 02 01 00 00 00 00 00
```

Beachten Sie bitte, daß diese von BASIC aus aufzurufenden Routinen mit der richtigen Speicherkonfiguration versorgt werden (also beispielsweise "BANK 15" vor den "SYS.."-Befehlen. Die X- und Y-Koordinaten der zu setzenden oder löschenden Punkte werden ganz einfach mit dem SYS-Befehl übergeben:

```
SYS <adresse>,<X-High-Byte>,<X-Low-Byte>,<Y-Byte>
```

Mit dem folgenden BASIC-Programm, das auf die eben beschriebenen Routinen zurückgreift, können Sie sich eine Sinuskurve am 80-Zeichen-Monitor ausgeben lassen:

```
100 BANK 15
110 SYS DEC("1400") : REM *** GRAFIK EINSCHALTEN
120 SYS DEC("1406") : REM *** BILDSCHIRM LÖSCHEN
130 FOR X = 0 TO 639
140 Y=SIN(X/100)    : REM *** SINUSWERT BERECHNEN
150 Y=99*Y+100      : REM *** WERT AN BILDSCHIRMAUSGABE ANPASSEN
160 SYS DEC("140F"), INT(X/256), X AND 255, Y
170 NEXT X
180 SYS DEC("1403") : REM *** TEXT EINSCHALTEN
190 SYS DEC("1409") : REM *** ZEICHENSATZ NEU LADEN, EDITOR INITIALISIEREN
```

Sie sehen, die Anwendung dieser Routinen ist kinderleicht. Und falls Sie nach einem Programmfehler einmal die Grafik verlassen wollen, so hilft meistens <RUN/STOP>/<RESTORE> und anschließendes Betätigen der Taste <ASCII/DIN>.

7.2 Auch der 64er kann 80 werden

... wenn man ihn dementsprechend programmiert. Die Verbindungssadressen sind - wie auch beim C128 - die Adressen $D600 und $D601. Ob er das allerdings auch in der endgültigen Version können wird, bleibt zu hoffen. Und hierzu gleich noch ein Gag: In der getesteten Ausführung reagierte der "beinahe-64'er" beim Betätigen der Taste <ASCII/DIN> prompt mit einem Zeichensatzwechsel - ganz atypisch für einen 64'er.

7.3 Tuning für den 64er

Der im PC128 eingebaute (leicht überarbeitete) VIC besitzt zwei neue Register, von denen besonders das zweite, Register #48 (dezimal) interessant sein dürfte. Mit diesem Register läßt sich nämlich die Geschwindigkeit des Prozessors verdoppeln (auch im 64er Modus). Probieren Sie's aus! Hierzu müssen Sie nur das Bit 1 in diesem Register setzen (oder löschen).

Anhang A: BASIC-Befehlsübersicht

A.1 Alphabetisches Verzeichnis der Anweisungen

Die ausführliche Übersicht über das BASIC des C128 hat folgenden Aufbau:

1. Die Befehlsworte von BASIC werden einzeln beschrieben. Die Befehlsbeschreibungen sind alphabetisch geordnet; jede Beschreibung beginnt auf einer eigenen Seite.

2. In der *ersten Zeile* auf der Seite befindet sich links das Befehlswort, in der Mitte der Typ des Befehls und am rechten Rand eine Versionsangabe.

3. Es werden folgende *Befehlstypen* unterschieden: Anweisungen, Kommandos und Funktionen. *Anweisungen* werden üblicherweise in BASIC-Programmen benutzt; sie sind Bestandteil des BASIC-Sprache. *Kommandos* richten sich an das Betriebssystem; sie werden üblicherweise im Direktmodus benutzt, wenngleich die überwiegende Mehrzahl der Kommandos auch in Programmen verwendet werden darf. *Funktionen* unterscheiden sich von den übrigen Befehlstypen dadurch, daß sie einen Wert zurückgeben. Bei Verwendung im Direktmodus muß dieser Wert beispielsweise mit PRINT ausgegeben werden, um keinen SYNTAX ERROR zu erzeugen.

4. Die *Versionsangabe* informiert darüber, in welcher BASIC-Version der Befehle verfügbar ist. Eine "7" am rechten Rand zeigt an, daß es sich um einen Befehl aus der Version 7.0 handelt, eine "2" entsprechend, daß der Befehl der Version 2.0 angehört. Zwei Zahlen zeigen an, daß der Befehl in beiden Versionen bekannt ist. Ein Ausrufezeichen deutet Änderungen an. Der Eintrag 2(!),7 bedeutet also: "der Befehl ist in beiden BASIC-Versionen vorhanden, hat sich jedoch in der Version 7.0 in seiner Funktionsweise geändert.

5. Die *Beschreibung* selbst gliedert sich in vier Teile: Schreibweise, Zweck, Kommentar und Beispiele.

6. Die *Schreibweise* teilt Ihnen die genaue Form mit, in der der Befehl gegeben werden muß, um von BASIC richtig verstanden zu werden. Dies schließt ein, welche und wieviele Parameter der Befehl benötigt. Eckige Klammern bei der Parameterangabe zeigen an, daß der jeweilige Parameter optional ist, das heißt, wahlweise angegeben werden kann.

7. Unter *Zweck* erfahren Sie, wozu der jeweilige Befehl gut ist.

8. Der *Kommentar* weist Sie auf Besonderheiten des Befehls hin, wie z.B. Einschränkungen oder Tricks, die nicht unmittelbar offensichtlich sind.

9. Die *Beispiele* wurden so gewählt, daß Sie auch aus ihnen nützliche Informationen entnehmen können, die manchmal über die bloß Illustrierung des jeweiligen Befehls hinausgehen. Sie sollten sie auf jeden Fall ausprobieren!

ABS — Funktion — 2,7

Schreibweise:

`ABS(ausdr)`

Zweck:

Liefert den Absolutwert von "ausdr"

Kommentar:

Der Absolutwert einer positiven Zahl ist diese Zahl selbst; bei einer negativen Zahl erhält man den Absolutwert durch Multiplikation mit -1 (entspricht Weglassen des Minuszeichens).

Die Funktion ist z.B. dann nützlich, wenn man den Größenunterschied zwischen zwei Zahlenwerte herausfinden will, ohne sich um deren Vorzeichen zu kümmern.

Beispiele:

```
PRINT ABS(34) - ABS(13 - 74/3)
100 IF ABS(A1%) > 5 THEN 400
```

AND — logische Funktion — 2,7

Schreibweise:

```
ausdr1 AND ausdr2
```

Zweck:

Hat den logischen Wahrheitswert "wahr", wenn sowohl "ausdr1" als auch "ausdr2" wahr sind. Ist nur einer der beiden mit AND verknüpften Ausdrücke falsch, so hat auch der Gesamtausdruck den Wert "falsch".

Wenn sowohl "ausdr1" als auch "ausdr2" reine Zahlenausdrücke sind (also keinen Vergleichsoperator enthalten), dann liefert AND als Wert das Ergebnis eines bitweisen AND mit den beiden Ausdrücken ausdr1 und ausdr2.

Kommentar:

Die Funktion hat zwei unterschiedliche Verwendungsweisen: einmal wird sie in Programmverzweigungen eingesetzt, um zusammengesetzte Bedingungen zu überprüfen (vgl. Beispiel 1, wo der Sprung nur ausgeführt wird, wenn die überprüfte Zahl zwischen 5 und 10 liegt).

Die andere Verwendungsweise (als "bitweise" logische Operation) ist besonders dann interessant und nützlich, wenn direkte Bytemanipulationen (z.B. bei der Arbeit mit Adressen) gewünscht werden. Möchte man etwa das höherwertige Halbbyte eines Bytes "ausblenden", so wird diese mit dem Zahlenwert 15 über AND verknüpft (da 15 dem Bitmuster 00001111 entspricht: vgl. Beispiel 2). Das direkte Arbeiten mit Adressen ist so allerdings nicht möglich, da AND nur Zahlen von -32768 bis 32767 akzeptiert, Adressen aber 16-Bit-Werte sind (0 bis 65535).

BASIC verwendet für die Wahrheitswerte "wahr" und "falsch" ebenfalls Zahlenwerte: "wahr" ist -1, falsch ist 0. Genaugenommen ist also das bitweise AND für beide Verwendungsweisen zuständig.

Beispiele:

```
IF A => 5 AND A <= 10 THEN 120
LO = BYTE AND 15
```

APPEND

Anweisung 7

Schreibweise:

```
APPEND # logische_dateinummer,
datei_name, [Dlaufwerk]
[ON Ugerätenummer]
```

Zweck:

Für sequentielles Beschreiben wird die Datei namens "datei_name" unter der Dateinummer "logische_dateinummer" geöffnet. Der Dateizeiger wird an das Ende der Datei gesetzt, so daß nachfolgende Ausgaben mittels PRINT# Informationen an das Dateiende anfügen.

Bei Mehrfachlaufwerken kann zusätzlich noch eine Geräteadresse über "gerätenummer" angegeben und ein Laufwerk über die Angabe "laufwerk" ausgewählt werden.

Kommentar:

Falls eine Datei mit dem angegebenen Namen nicht existiert, so führt dies auf den Fehler 62.

Für "laufwerk" ist der Wert 0, für "gerätenummer" der Wert 8 voreingestellt.

Werden für einen der Parameter Variablen verwendet, so sind diese in runde Klammern zu setzen.

Bei logischen Dateinummern zwischen 1 und 127 werden Datenausgaben mittels PRINT# mit dem Zeichencode für "Wagenrücklauf" abgeschlossen. Bei Werten zwischen 128 und 255 wird zusätzlich noch ein Zeilenvorschub ausgegeben.

Beispiele:

```
10 DOPEN#1,"TEST1",W
20 PRINT#1,"HALLO"
30 DCLOSE#1
40 APPEND#1,"TEST1"
50 PRINT#1,"WIE GEHT ES DIR"
60 DCLOSE#1
70 D$ = "TEST1"
80 N = 1
90 APPEND#(N),(D$)
100 PRINT#1,"BIST DU MIT DEM C128 ZUFRIEDEN?"
110 DCLOSE#(N)
120 DOPEN#1,(D$)
130 FOR I = 1 TO 3
140 INPUT#1,A$ : PRINT A$
150 NEXT I
160 DCLOSE#1
```

ASC

(!),7

Stringfunktion

Schreibweise:

```
ASC(str$)
```

Zweck:

Ergibt den (dezimalen) ASCII-Wert des ersten Zeichens von "str$".

Kommentar:

Falls "str$" ein Einzelzeichen ist, so wird dessen Positionsnummer in der Zeichentabelle (ASCII-Wert) als Dezimalzahl ausgegeben. Ist "str$" ein String mit mehr als einem Zeichen, so untersucht die Funktion nur das erste Zeichen im String.

Die Funktion ist z.B. dann nützlich, wenn man mit Einzelzeichen rechnen will.

Hinsichtlich der Behandlung des leeren Strings unterscheiden sich der 64er- und 128er-Modus:

64er-Modus: Falls "str$" der leere String ist (also ""), wird die Fehlermeldung ILLEGAL QUANTITY ERROR ausgegeben.

128er-Modus: Falls "str$" der leere String ist, so liefert die Funktion den Wert 0 (ohne Fehlermeldung).

Beispiele:

```
10 A$ = "STRING"
20 LEER$ = ""
30 PRINT ASC("A"),ASC(A$),ASC(""),ASC(LEER$)
```

ATN — Funktion — 2,7

Schreibweise:

`ATN(ausdr)`

Zweck:

Liefert den Arcustangens des arithmetischen Ausdrucks "ausdr".

Kommentar:

Das Ergebnis der Funktion liegt im Bereich zwischen pi/2 und -pi/2.

Beispiele:

```
10 FOR I = 0 TO 360 STEP 10
20 PRINT ATN(I)
30 NEXT I
```

AUTO **Befehl** 7

Schreibweise:

```
AUTO [schrittweite]
```

Zweck:

Schaltet die automatische Zeilennumerierung ein bzw. aus, die die Programmeingabe erleichtert, indem sie dem Benutzer die Zeilennummern vorgibt.

Kommentar:

Nach der Eingabe einer Programmzeile mittels RETURN wird automatisch die nächste Programmzeilen-Nummer ausgegeben. Die Zeilennumerierung richtet sich nach der Angabe in "schrittweite". Dies muß eine Zahlenkonstante sein. Der Cursor steht in der neuen Zeile bereits an der richtigen Position für die Programmeingabe.

Die erste Zeilennummer des Programmes muß allerdings vom Benutzer vorgegeben werden; von da an zählt das System automatisch hoch, bis Sie eine leere Programmzeile eingeben und die automatische Numerierung abschalten.

Um die automatische Zeilennumerierung auszuschalten, muß AUTO ohne Parameter angegeben werden.

AUTO warnt den Benutzer nicht, wenn bereits exisitierende Zeilennummern eines Programmes erzeugt werden; es ist also möglich, unbeabsichtigt Programmteile zu überschreiben.

Beispiele:

```
AUTO 10          (erzeugt Zeilennummern in Zehnerabständen)
AUTO 21          (die Zeilennummern werden mit einer Schrittweite
                  von 21 generiert)
AUTO             (schaltet die Zeilennumerierung aus)
```

BACKUP Anweisung 7

Schreibweise:

```
BACKUP Dlaufwerk TO Dlaufwerk
[,ON Ugerätenummer]
```

Zweck:

Erstellt eine Eins-zu-Eins-Kopie einer Diskette.

Kommentar:

Die Anweisung funktioniert nur bei Doppellaufwerken. Bei Eingabe im Direktmodus wird erst mit "ARE YOU SURE" um Bestätigung gefragt (bestätigen mit "Y").

Die Diskette, auf die kopiert werden soll, muß nicht erst mit HEADER formatiert werden, da BACKUP alle Informationen einschließlich des Formats mitkopiert. Sie können also eine fabrikneue Diskette verwenden. Wenn Sie für die Datensicherung eine bereits in Gebrauch befindliche Diskette benutzen, dann sind nach dem Kopiervorgang die darauf befindlichen Informationen (unrettbar!) verloren. Daher auch die Rückfrage im Direktmodus.

Als "gerätenummer" ist der Wert 8 voreingestellt.

Wollen Sie für einen Parameter Variablen verwenden, so müssen diese in runden Klammern stehen.

Beispiele:

```
BACKUP D0 TO D1
(Die Diskette im Laufwerk 0 wird nach Laufwerk 1 kopiert)

BACKUP D0 TO D1, ON U9
(Wie oben, aber auf Einheit 9)
```

BANK — Anweisung — 7

Schreibweise:

```
BANK (ausdr)
```

Zweck:

Wählt die Speicherbank aus, auf die sich nachfolgende PEEK-, POKE-, SYS- oder WAIT-Befehle beziehen sollen.

Kommentar:

"ausdr" muß einen Wert zwischen 0 und 15 haben; ansonsten wird ein ILLEGAL QUANTITY ERROR ausgegeben. Physikalisch sind im C128 nur zwei Speicherbänke vorhanden. Was für den Fall geschieht, daß "ausdr" größer 2 ist, können Sie dem Beispiel entnehmen.

Beispiele:

```
10 BANK 0 : POKE 8000,255
20 BANK 1 : POKE 8000,100
30 FOR I = 0 TO 15
40 BANK I : PRINT I,PEEK(8000)
50 NEXT I
```

BEGIN...BEND Anweisung 7

Schreibweise:

```
IF bedingung THEN BEGIN
.
.     (Anweisungen)
.
BEND : ELSE BEGIN
.
.     (Anweisungen)
.
BEND
```

Zweck:

Das Anweisungspaar BEGIN...BEND tritt nur in Verbindung mit der IF...THEN...ELSE Anweisung (siehe diese) auf und dient dazu, innerhalb des THEN- oder ELSE-Zweiges einen beliebig langen Block von BASIC-Anweisungen zu umschließen.

Kommentar:

Mit diesem Anweisungspaar wird ein Defekt der BASIC-Version 2.0 behoben, das verlangt, die gesamte IF ... THEN ... ELSE-Anweisung in eine Zeile zu schreiben. Die hinter THEN und ELSE möglichen Anweisungen waren dadurch in ihrer Länge sehr beschränkt.

Mit BEGIN und BEND ist es jetzt möglich, sowohl im THEN- als auch im ELSE-Zweig beliebig viele Anweisungen abarbeiten zu lassen.

Bei der Abgrenzung von Anweisungsblocks ist Vorsicht geboten, da die üblichen Regeln für IF...THEN...ELSE weiterbestehen. Insbesondere führt BASIC alle Anweisungen im THEN- oder ELSE-Zweig bis zum Zeilenende aus. So wird im zweiten Beispiel die Meldung "WEITER" nur ausgegeben, wenn X den Wert 1 hat.

Beispiele:

```
10 X = INT(100*RND(1))+1
20 PRINT "SIE MUESSEN EINE ZAHL ZWISCHEN 1"
30 PRINT "UND 100 ERRATEN"
40 INPUT "WAS RATEN SIE";R
50 IF R<>X THEN BEGIN
60 :  IF R<X THEN PRINT "ZU KLEIN!"
70 :  IF R>X THEN PRINT "ZU GROSS!"
80 BEND : GOTO 40 : ELSE PRINT "RICHTIG!" : END
```

```
10 IF X=1 THEN BEGIN : A=5
20 B=6 : C=7
30 PRINT A*B*C : PRINT "WEITER"
```

BLOAD Anweisung 7

Schreibweise:

```
BLOAD datei_name [, Dlaufwerk]
[,Ugerätenummer] [ON Bbank] [,Pstart]
```

Zweck:

Lädt eine Binärdatei namens "datei_name" in die mit "bank" ausgewählte Speicherbank ab der Startadresse "start". Die Datei befindet sich auf "laufwerk" der Floppy Disk mit der Nummer "gerätenummer".

Kommentar:

Eine (z.B. mittels BSAVE erzeugte) Binärdatei, die z.B. ein Maschinenprogramm oder andere Binärdaten enthält, wird von Diskette geladen. Mit "bank" kann eine von 16 möglichen Speicherbänken durch Werte von 0 bis 15 für das Laden ausgewählt werden. Die Startadresse, ab der die Binärdaten geladen werden sollen, bestimmen Sie mit der Angabe "start".

Sollen für einen der Parameter Variablen verwendet werden, so müssen diese in runde Klammern eingeschlossen werden.

Mit BLOAD können Sie auch BASIC-Programme laden, und zwar schneller, als dies mit LOAD und DLOAD (s. dort) möglich ist. Dazu müssen Sie aber den Parameter "start" mit der Ladeadresse von BASIC-Programmen versehen. Normalerweise ist dies 7169 (Hexadezimal: 1C01). Sollten Sie aber mit GRAPHIC (s. dort) Speicherplatz für den Graphik-Bildschirm reserviert haben, so liegt diese Adresse bei 16385 (Hexadezimal: 4000). Die Beispiele 3 und 4 machen dies deutlich. Aus Gründen, die hier nicht näher erläutert werden können (d.h. ich weiß noch nicht genau, warum), ist das Programm zwar lauffähig (mit RUN oder GOTO), darf aber nicht mehr verändert (editiert) werden (auch keine Zeilennummern löschen oder einfügen!).

Beispiele:

```
BLOAD "BINDAT" ON B1,P4000

10 D$ = "MASCHPROG"
20 BK = 0
30 BLOAD (D$) ON B(BK)

BLOAD "BASIC.PRG",P7169
```
(BASIC-Programm ohne Grafikbildschirm laden)

```
BLOAD "BASIC.PRG",P16385
```
(BASIC-Programm laden, wenn zuvor GRAPHIC gegeben wurde)

BOOT Anweisung 7

Schreibweise:

```
BOOT datei_name [,Dlaufwerk]
[,Ugerätenummer]
```

Zweck:

Lädt die Binärdatei "datei_name" vom angegebenen Laufwerk und Gerät und veranlaßt deren Ausführung ab der darin spezifizierten Startadresse.

Kommentar:

Bei dem Inhalt der Datei muß es sich selbstverständlich um ein ausführbares Maschinenprogramm handeln.

Als "gerätenummer" ist 8, als "laufwerk" 0 voreingestellt.

Beispiele:

```
BOOT "LADER"

10 D$ = "PROG"
20 BOOT (D$)
```

BOX — Grafik-Anweisung — 7

Schreibweise:

```
BOX [farbquelle] ,x1,y1 [, x2, y2 ,
winkel, füllung]
```

Zweck:

Ein Rechteck wird an einer vom Benutzer bestimmten Bildschirmposition und in einem von ihm gewählten Drehwinkel zur waagerechten Bildschirm-Achse gezeichnet und wahlweise farbig ausgefüllt. Die Parameter und ihre zulässigen Werte sind:

farbquelle :
0 Hintergrundfarbe 40-Zeichen-Bildschirm
1 Vordergrund-Farbe Grafikbildschirm
2 Multicolor1
3 Multicolor2

x1,y1: Eckkoordinaten (0 <= x1 <= 319; 0 <= y1 <= 199)

x2,y2: Koordinaten des Ecks gegenüber von (x1,y1)

winkel: Drehwinkel um den geometrischen Mittelpunkt

füllung:
0 Rechteck nicht ausfüllen
1 Rechteck ausfüllen

Kommentar:

Der Befehl erlaubt es, an einer beliebigen Stelle des Grafikbildschirms (zuvor mittels GRAPHIC-Kommando auswählen!) ein Rechteck beliebiger Größe zu zeichnen. (Es muß natürlich auf den Bildschirm passen!). Das Rechteck kann außerdem mit der momentan gewählten Farbquelle ausgemalt und um den Mittelpunkt gedreht werden.

Es müssen nicht alle Parameter angegeben werden. Die "farbquelle" kann weggelassen werden; voreingestellt ist der Grafikbildschirm. Ebenso kann eines der beiden Koordinatenpaare fehlen. Es wird dann entweder die linke obere Bildschirmecke oder die aktuelle Position des Pixel-Cursors (siehe LOCATE) als gegenüberliegender Eckpunkt genommen. Als "winkel" ist 0 voreingestellt; ebenso hat "füllung" den Standardwert 0.

BOX verändert die aktuelle Position des Pixel-Cursors; nachdem die Zeichnung beendet ist, befindet er sich bei x2,y2.

Wollen Sie das Rechteck füllen lassen, ("füllung" = 1), so wird dazu die Vordergrundfarbe der aktuellen Farbquelle benutzt. Um also z.B. ein farbiges Quadrat mit einer andersfarbigen Umrandung zu zeichnen, müssen Sie zuerst das gefüllte Quadrat in der gewünschten Farbe zeichnen, dann die Farbquelle für die Umrandung ändern und anschließend ein (nicht gefülltes!) Quadrat um das bereits vorhandene herum zeichnen lassen.

Beispiele:

```
BOX 1,10,10,60,60
```
(Ein Quadrat in der Vordergrund- (Zeichen-)Farbe mit der linken oberen Ecke bei 10,10 und der rechten unteren Ecke bei 60,60 wird gezeichnet)

```
BOX ,10,10,60,60
```
(zeichnet dasselbe Quadrat wie im letzten Beispiel)

```
BOX ,10,10,60,60,45,1
```
(zeichnet das Quadrat aus dem ersten Beispiel, aber um 45 Grad im Uhrzeigersinn gedreht und ausgefüllt)

```
BOX ,60,60
```
(zeichnet ein Quadrat, das an der linken oberen Bildschirmecke beginnt und den Punkt 60,60 als rechten unteren Eckpunkt hat)

```
10 GRAPHIC 1
20 SCNCLR
30 X = 5
40 FOR I = 0 TO 36 STEP 5
50 BOX 1,160+X,160-X,100-X,I,I
60 NEXT
70 END
```

BSAVE

Anweisung 7

Schreibweise:

```
BSAVE datei_name [, Dlaufwerk]
[,Ugerätenummer] [ON Bbank]
[,Pstart TO Pend]
```

Zweck:

Schreibt den mittels "start" und "end" bestimmten Bereich des Arbeitsspeichers aus der mit "bank" gewählten Speicherbank auf eine Datei namens "datei_name", die sich auf "laufwerk" der Floppy Disk mit der Nummer "gerätenummer" befindet.

Kommentar:

Werden für einen der Parameter Variablen verwendet, so müssen diese in runde Klammern eingeschlossen werden.

Sie können mit dieser Möglichkeit nicht nur Maschinenprogramme, sondern auch Grafiken aus dem Grafikspeicher auf Datei sichern und später (mit BLOAD; s. dort) wieder laden.

Beispiele:

```
BSAVE "BINDAT1", D0, ON B1, P1024 TO P3071
```
(der Bereich ab Adresse 1024 bis einschließlich Adresse 3071 - entspricht 2K - auf Bank 1 wird in der Datei "BINDAT1" auf Laufwerk 0 als Binärdaten abgelegt.)

BUMP — Funktion — 7

Schreibweise:

BUMP (n)

Zweck:

Stellt fest, welches Sprite (0 bis 7) seit der letzten BUMP-Abfrage entweder mit einem anderen Sprite ("n" = 1) oder mit Anzeigedaten ("n" = 2) kollidiert.

Der Wert von BUMP() ist eine Zahl zwischen 0 und 255, die als Byte interpretiert werden muß, bei dem ein Bit gesetzt ist, wenn das entsprechende Sprite kollidiert ist.

Kommentar:

BUMP() kann verwendet werden, ohne daß zuvor mittels COLLISION eine Programm-Unterbrechung für Sprite-Kollisionen aktiviert wurde.

BUMP() liefert Werte, bei denen die einzelnen Bitpositionen ausgewertet werden müssen. Ein Wert von 1 bedeutet, daß Sprite 1 kollidierte, 2 weist auf Sprite 2 hin, 4 auf Sprite 3, 8 auf Sprite 4 usw. Ein Wert von 147 bedeutet somit, daß die Sprites 1,2,5 und 8 kollidiert sind. Im Beispiel 1 ist ein kleines Programm aufgeführt, mit dem Sie bei Mehrfachkollisionen feststellen können, welche Sprites miteinander (bzw. mit dem Hintergrund) zusamengestoßen sind.

Bei Kollisionen von mehr als 2 Sprites reicht die von BUMP() gelieferte Information nicht mehr aus, um festzustellen, wer mit wem kollidiert ist; in diesem Fall müssen Sie die Funktion RSPPOS zu Hilfe ziehen.

Das zweite Programmfragment zeigt Ihnen, wie Sie in einem Unterprogramm zur Behandlung von Programmunterbrechungen (siehe COLLISION-Anweisung) Operationen für ein bestimmtes Sprite (hier Nummer 2) auslösen können.

Beispiele:

```
10 K = BUMP(0)
20 FOR I = 0 TO 7
30 TST = 2^I
40 IF K AND TST THEN PRINT "SPRITE 2;I+1;" KOLLIDIERTE"
50 NEXT
.
.
.
110 COLLISION 1,1000 : REM Unterprogramm ab Zeile 1000 bei
120 :                  REM Spritekollision
.
.
.
1000 REM Unterbrechungs-Routine mit Abfrage auf Sprite 2
1010 COLLISION 1 : REM Unterbrechungen abschalten
1020     IF BUMP(0) AND 2 THEN GOSUB 2000
1030 COLLISION 1,1000 : REM Unterbrechung wieder zulassen
1040 RETURN
.
.
2000 REM Kollisionsbehandlung für Sprite 2
.
.
2999 COLLISION 1.1000 : RETURN
```

CATALOG **Befehl** 7

Schreibweise:

```
CATALOG [Dlaufwerk] [Ugerätenummer]
[namensmuster]
```

Zweck:

Der CATALOG-Befehl ist identisch zum DIRECTORY-Befehl (siehe dort).

CHAR Anweisung 7

Schreibweise:

```
CHAR [farbquelle] ,x,y [,str$] [,revers]
```

Zweck:

Gibt eine Zeichenkette auf dem Bildschirm aus. Im Unterschied zu PRINT kann dabei die Zeichenausgabe auch auf dem Grafikbildschirm erfolgen. Die Position und gegebenenfalls reverse Darstellung können gewählt werden.

Die Parameter und ihre zulässigen Werte sind:

farbquelle:
0 Hintergrundfarbe
1 Vordergrund- (Zeichen-)Farbe
2 Multicolor 1
3 Multicolor 2

x,y: Die Spalten- und Zeilenkoordinaten für den Text
(0 <= x <= 79; 0 <= y <= 25)

str$: Die auszugebende Zeichenfolge

revers:
0 normale Darstellung
1 reverse Darstellung

Kommentar:

Die Anweisung ist nützlich, um z.B. Grafiken zu beschriften, die auf dem Grafikbildschirm erstellt wurden. Hier kann PRINT nicht verwendet werden, da dessen Ausgaben ausschließlich auf den Textbildschirm gehen.

Die Zeichendefinitionen für CHAR stammen direkt aus dem Zeichen-ROM des C128.

Sollte der auszugebende String nicht auf eine Zeile passen, so wird er auf der nächsten fortgesetzt.

Auf dem Textbildschirm arbeitet CHAR genauso wie die PRINT-Anweisung. Hier können also im Ausgabestring auch Steuercodes enthalten sein. Im Grafikmodus werden diese Steuerzeichen jedoch nicht ausgewertet.

Beachten Sie, daß mit CHAR ein String nur auf Zeichengrenzen beginnen kann; die "feinere" Positionierung, wie sie der Pixel-Cursor erlaubt (siehe LOCATE), ist hier nicht möglich.

Beispiele:

```
10 CIRCLE ,160,100,65,50 : REM Kreis um 160,100 zeichnen
20 CHAR 1,17,12,"Kreis" :  REM und innen beschriften
```

CHR$ String-Funktion 2,7

Schreibweise:

```
CHR$(ausdr)
```

Zweck:

Liefert das Zeichen, dessen Position in der Codetabelle durch "ausdr" bestimmt ist.

Kommentar:

Die Funktion ist nützlich, um Zeichen in einem Programm auszugeben, die über die Tastatur nicht erreicht werden können (also vorzugsweise Steuerzeichen).

Beispiele:

```
PRINT CHR$(11)
```
(Blockiert die Umschaltung zwischen Groß-und Kleinschreibung mittels SHIFT und C=)

```
PRINT CHR$(12)
```
(Entriegelt diese Blockade wieder)

```
PRINT CHR$(14)
```
(Schaltet um in den Kleinschrift-Modus - was sonst SHIFT und C= zusammen machen)

```
PRINT CHR$(142)
```
(Schaltet um in Großschrift-Modus)

```
PRINT CHR$(19)
```
(Cursor geht in linke obere Bildschirm-Ecke)

```
PRINT CHR$(27);"A"
```
(Schaltet den Einfüge-Modus am Bildschirm ein)

```
PRINT CHR$(27);"C"
```
(Schaltet den Einfüge-Modus am Bildschirm aus)

CIRCLE — Grafik-Anweisung — 7

Schreibweise:

```
CIRCLE [farbquelle] ,[x, y], xr [, yr,
start, ende, winkel, segmentwinkel]
```

Zweck:

Zeichnet einen Kreis, eine Ellipse, einen Kreisbogen oder einen beliebigen Polygonzug mit wählbarem Mittelpunkt auf den Grafikbildschirm.

Die Parameter und ihre Bedeutung:

farbquelle:
0 Hintergrundfarbe
1 Vordergrund- (Zeichen-)Farbe
2 Multicolor 1
3 Multicolor 2

x,y: der Mittelpunkt des Polygonzugs oder Kreises

xr: Radius der Haupt- (X-)Achse

yr: Radius der Neben- (Y-)Achse

start: Startwinkel für Kreisausschnitte

ende: Endwinkel für Kreisausschnitte

winkel: Drehwinkel der Gesamtfigur

segmentwinkel: Winkel zwischen einzelnen Segmenten eines Polygonzugs.

Kommentar:

Neben Kreisen kann man mit CIRCLE auch Ellipsen und regelmäßige Vielecke sowie Ausschnitte aus diesen Figuren zeichnen. Wem das seltsam vorkommt, der mache sich klar, daß man einen Kreis auch als ein regelmäßiges Vieleck mit sehr vielen Ecken auffassen kann.

Die Anzahl der Ecken des Polygons wird mit dem letzten Parameter "segmentwinkel" bestimmt. Je größer der Winkel, desto geringer die Eckenzahl. Zugelassen sind allerdings nur Werte zwischen 0 und 255. Ein Wert von 2 ist hier voreingestellt.

Mit den Parametern "x" und "y" wird der Mittelpunkt des Vielecks festgelegt. Es ist auch möglich, die Figur im Uhrzeigersinn um diesen Mittelpunkt drehen zu lassen, und zwar über den Parameter "winkel".

Für das Zeichnen von Kurvenzügen müssen zwei(!) Radien angegeben werden. Somit ist es möglich, auch Ellipsen mittels CIRCLE zu erstellen. Diese sind also dadurch ausgezeichnet, daß sie auf der waagrechten (X-Achse; Parameter "xr") einen anderen Radius als auf der Senkrechten (Y-Achse; Parameter "yr") aufweisen.

Falls der Parameter "yr" nicht angegeben wird, so wird er mit "xr" gleichgesetzt; dies bewirkt, daß ein Kreis bzw. Kreisausschnitt gezeichnet wird.

Es ist auch möglich, nur einen Ausschnitt aus der Gesamtfigur auf dem Bildschirm erscheinen zu lassen. Dazu müssen die Parameter "start" und "ende" angegeben sein. Am besten macht man sich das am Beispiel eines Kreises klar. Denken Sie an das Zifferblatt einer Uhr: Besitzt "start" den Wert 0 (voreingestellt), so wird mit dem Zeichnen in der "12-Uhr-Position" begonnen. Besitzt Ende den Wert 45, so endet der Ausschnitt in der "Drei-Uhr-Position". Hat "start" den Wert 60 und "ende" den Wert 270, so erhält man einen Kreisausschnitt, der an einen lächelnden Mund erinnert. Voreingestellt ist für "ende" der Wert 360 (also einmal rundrum).

Beispiele:

```
CIRCLE ,160,100,65,10
```
(Zeichnet eine Ellipse in der Mitte des Bildschirms)

```
CIRCLE ,160,100,100
```
(Zeichnet einen Kreis in der Mitte des Bildschirms)

```
CIRCLE ,260,40,20,,,,,90
```
(Ergibt ein um 45 Grad gedrehtes Quadrat)

```
CIRCLE ,60,140,20,,,,20,120
```
(Zeichnet ein um 20 Grad gedrehtes gleichseitiges Dreieck)

CLOSE — Anweisung — 2,7

Schreibweise:

```
CLOSE logische_dateinummer
```

Zweck:

Schließt die Datei mit der angegebenen logischen Dateinummer.

Kommentar:

Die Dateinummer wurde der Datei beim Öffnen mit OPEN zugeordnet. Für das Arbeiten mit Disketten-Dateien gibt es das praktischere DCLOSE.

Beispiel:

```
CLOSE 2
```

CLR — Anweisung — 2,7

Schreibweise:

CLR

Zweck:

Löscht alle Variablen, Felder und vom Benutzer definierten Funktionen.

Kommentar:

Im Unterschied zu NEW bleibt ein im Speicher befindliches BASIC-Programm nach CLR erhalten. Der Befehl wird automatisch bei jedem RUN und NEW durchgeführt und auch dann, wenn ein Programm ediert (am Bildschirm bearbeitet) wurde.

Beispiel:

```
CLR
```

CMD — Befehl — 2,7

Schreibweise:

```
CMD logische_dateinummer
```

Zweck:

Lenkt die Ausgaben, die normalerweise auf den Bildschirm gehen, an ein anderes Gerät, z.B. den Drucker oder eine Diskettendatei.

Kommentar:

Bildschirmausgaben werden mit PRINT oder LIST erzeugt, aber nicht mit POKE in den Bildschirmspeicher.

Das Gerät, auf welches die Ausgabe umgelenkt wird, muß zuvor (mittels OPEN oder DOPEN) geöffnet werden.

Um die Ausgabe wieder auf den Bildschirm zu legen, benutzen Sie PRINT#1.

Beispiel:

```
OPEN 1,4        (Drucker öffnen mit logischer Nummer 1)
CMD 1           (Ausgabe auf Drucker umlenken)
LIST
PRINT#1         (Ausgabe wieder auf Bildschirm)
CLOSE 1         (Drucker schließen)
```

COLLECT **Befehl** 7

Schreibweise:

```
COLLECT [Dlaufwerk ON Ugerätenummer]
```

Zweck:

Löscht nicht ordnungsgemäß geschlossene Dateien von einer Diskette.

Kommentar:

Der den Dateien zugewiesene Platz wird auf der Diskette wieder freigemacht; sämtliche Hinweise auf die Dateien werden aus dem Inhaltsverzeichnis entfernt.

Beispiele:

```
COLLECT

COLLECT DO ON U10
```

COLLISION — Grafik-Anweisung — 7

Schreibweise:

```
COLLISION typ [, zeilennummer]
```

Zweck:

Programmunterbrechungen für den Fall einer Sprite-Kollision werden aktiviert bzw. desaktiviert. Im Falle der Aktivierung wird außerdem noch angegeben, bei welcher Zeilennummer das Unterprogramm beginnt, das die Unterbrechung behandelt.

Die Bedeutung des Parameters "typ":

0	Unterbrechung bei Kollision zwischen Sprites
1	Unterbrechung bei Kolision Sprite/Anzeigedaten
3	Aktivierung des Lichtstifts

Kommentar:

Mit der COLLISION-Anweisung wird eine wichtige Eigenschaft der BASIC-Version 7.0 realisiert. Dadurch ist es nämlich einem Programm möglich, auf bestimmte Ereignisse (Sprite-Kollisionen) zu reagieren, ohne das Vorliegen dieser Ereignisse selbst abfragen zu müssen.

Wenn der über den Parameter "typ" spezifizierte Ereignistyp eintritt, dann wird das laufende Programm automatisch angehalten (die gerade in Bearbeitung befindliche Anweisung wird aber noch zu Ende geführt). Anschließend wird mit GOSUB automatisch zu dem Unterprogramm verzweigt, das im Parameter "zeilennummer" angegeben ist. Dieses Unterprogramm muß mit RETURN abgeschlossen sein, um eine korrekte Rückkehr in das eigentliche Programm zu ermöglichen.

Es können mehrere Unterbrechungstypen gleichzeitig mittels COLLISION aktiviert sein. Allerdings kann immer nur ein Unterbrechungstyp auf einmal in einem Unterprogramm abgehandelt werden. Unterbrechungen innerhalb von Unterbrechungen (Schachtelung) sind somit nicht zugelassen. Deshalb sollten von der Unterbrechungsroutine zuerst alle aktiven Unterbrechungen ausgeschaltet werden.

Bedenken Sie, daß die Situation, die die Unterbrechung verursacht hat, auch noch nach Durchlaufen der Unterbrechungsroutine vorliegen kann!

Eine Kollision zwischen Sprites liegt nur dann vor, wenn sich Teile der Sprites überlappen, die im Vordergrund liegen. Ausgeschaltete Sprites (s. SPRITE-Anweisung), die sich überlappen, erzeugen keine Kollision.

Sprites können Unterbrechungen nur dann auslösen, wenn sie sich im sichtbaren Bereich des Bildschirms befinden.

Um eine Unterbrechung eines bestimmten Typs auszuschalten wird die COLLISION-Anweisung ohne den zweiten Parameter ("zeilennummer") benutzt.

Um herauszufinden, welche Sprites kollidierten, benutzen Sie die BUMP()-Funktion.

Beispiel:

Ein Programmfragment für die Kollisionsbehandlung mit zwei Unterbrechungstypen:

```
.
.
100 COLLISION 0,1000  :  REM Unterbrechung für Sprite/Sprite
110 COLLISION 1,2000  :  REM Unterbrechung für Sprite/Anzeige
.
.
1000 COLLISION 0 : COLLISION 1 : REM Unterbrechung aus
.                (Anweisungen für Unterbrechungsbehandlung bei
.                Kollision von Sprites)
1900 RETURN
.
.
2000 COLLISION 0 : COLLISION 1 : REM Unterbrechung aus
.                (Anweisungen für Unterbrechungsbehandlung bei
.                Kollision eines Sprites mit Anzeigedaten)
2900 RETURN
```

COLOR — Grafik-Anweisung — 7

Schreibweise:

```
COLOR farbquelle, farbcode
```

Zweck:

Ordnet der in "farbquelle" ausgewählten Quelle eine von 16 möglichen Farben zu.

Die Parameter und ihre Werte:

farbquelle:

0	Hintergrund des Text- und Grafikbildschirms
1	Vordergrund Grafikbildschirm
2	Multicolor 1
3	Multicolor 2
4	Randfarbe
5	Textfarbe 40-Zeichen-Bildschirm
6	Hintergrundfarbe 80-Zeichen-Bildschirm

farbcode:

1	schwarz	9	orange
2	weiß	10	braun
3	rot	11	gelb-grün
4	türkis	12	rosa
5	purpur	13	blau-grün
6	grün	14	hellblau
7	blau	15	dunkelblau
8	gelb	16	hellgrün

Beispiele:

```
10 COLOR 0,15 : REM Hintergrund wird dunkelblau
20 COLOR 1,4  : REM Vordergrundfarbe für Grafikbildschirm
30 :            REM wird türkis
40 COLOR 4,11 : REM Randfarbe wird gelb-grün
```

CONCAT **Befehl** 7

Schreibweise:

```
CONCAT [Dquellaufwerk,] quelldatei TO
[Dziellaufwerk,] zieldatei
[ON Ggerätenummer]
```

Zweck:

Die Daten zweier sequentieller Dateien werden aneinandergefügt (verkettet), indem der Inhalt von "quelldatei" an die "zieldatei" angehängt wird.

Kommentar:

Die Angaben zu "quelldatei" und "zieldatei" können sowohl Stringkonstanten als auch Variablen sein. Im letzteren Fall müssen sie allerdings in runde Klammern eingeschlossen werden. Dies gilt auch für die anderen Parameter.

Die Verkettung erfolgt, indem die Daten in der "quelldatei" an das Ende der "zieldatei" angehängt werden.

Beispiele:

```
CONCAT "DATEI1" TO "DATEI2"

10 QL = 0 : ZL = 1
20 Q$ = "QUELLDAT" : Z$ = "ZIELDAT"
30 CONCAT D(QL), (Q$) TO D(ZL), (Z$)
```

CONT — Befehl — 2,7

Schreibweise:

CONT

Zweck:

Fortsetzen eines unterbrochenen Programmes an der Stelle der Unterbrechung.

Kommentar:

Ein Programm, das entweder durch Drücken der STOP-Taste, durch eine STOP-Anweisung oder eine END-Anweisung im Programm angehalten wurde, wird wieder fortgesetzt.

CONT funktioniert nicht, wenn das angehaltene Programm zwischenzeitlich geändert wurde (selbst wenn Sie versehentlich den Cursor auf eine Programmzeile bewegt und die RETURN-Taste gedrückt haben, ohne wirklich etwas zu verändern). Auch bei Programmbeendigung durch einen Syntax-Fehler ist CONT unwirksam, oder wenn zwischen dem Anhalten und dem versuchten Fortsetzen durch den Benutzer ein Fehler herbeigeführt wurde. In diesen Fällen können Sie das Programm immer noch mit GOTO fortsetzten - vorausgesetzt, Sie kennen die richtige Zeilennummer.

COPY **Befehl** 7

Schreibweise:

```
COPY [Dquellaufwerk,] quelldatei TO
[Dziellaufwerk,] zieldatei
[ON Ggerätenummer]
```

Zweck:

Kopieren einer Datei von einem Laufwerk auf ein anderes (nur bei Doppelfloppy) oder Kopieren auf derselben Diskette unter einem neuen Namen.

Kommentar:

Es ist nicht möglich, mit COPY Dateien von einem Gerät zum anderen zu kopieren (die Angabe zweier unterschiedlicher Gerätenummern ist nicht erlaubt).

Falls bei Verwendung einer Doppelfloppy die Angabe von "quelldatei" und "zieldatei" unterbleibt, werden alle Dateien im ersten angegebenen Laufwerk kopiert.

Werden "quellaufwerk" und "ziellaufwerk" nicht angegeben, so wird die "quelldatei" auf der gleichen Diskette dupliziert, erhält aber einen neuen Namen, nämlich den in "zieldatei" angegebenen.

Bei der Angabe der "quelldatei" können Jokerzeichen ("?" und "*") verwendet werden, um so mit einem Kommando mehrere Dateien zu erfassen. (Vorsicht, wenn nur ein Laufwerk vorhanden!)

Beispiele:

```
COPY D0, "TEST" TO D1, "TEST.BAK"
```
("TEST" wird von Laufwerk 1 kopiert und erhält auf Laufwerk 2 den neuen Namen "TEST.BAK")

```
COPY D0 TO D1
```

```
COPY "PROGRAMM" TO "SICHERUNG"
```
(Es wird auf der gleichen Diskette eine Kopie der Datei "PROGRAMM" erstellt, die den Namen "SICHERUNG" erhält)

COS

Funktion 2,7

Schreibweise:

```
COS(ausdr)
```

Zweck:

Liefert den Cosinus des durch "ausdr" bestimmten Winkels.

Kommentar:

"ausdr" muß in Bogengraden angegeben sein.

Beispiel:

```
PRINT COS(2/3)
```

DATA — Anweisung — 7

Schreibweise:

```
DATA datenliste
```

Zweck:

Hält Daten bereit, die mittels READ an Variable zugewiesen werden sollen. Die einzelnen Elemente der "datenliste" müssen durch Komma getrennt werden.

Kommentar:

Die Elemente der "datenliste" können Zahlen oder Stringkonstanten sein. Die Stringkonstanten müssen nicht in Anführungszeichen eingeschlossen sein, solange sie nicht führende Leerzeichen oder ein Komma enthalten sollen.

Zwei unmittelbar aufeinanderfolgende Kommas in der "datenliste" (ohne zwischenliegenden Wert) bewirken, daß der mit READ zugewiesenen Wert gleich 0 ist, falls an eine numerische Variable zugewiesen wird, oder der leere String "" bei Stringvariablen.

Im Zusammenhang mit READ und DATA ist auch die RESTORE-Anweisung von Bedeutung, mit der die Daten aus der "datenliste" mehrfach mittels READ zugewiesen werden können.

Beispiel:

```
100 DATA 10, 200, HALLO PETER
110 DATA DIESER STRING ENDET MIT LEERZEICHEN       ,
120 DATA "    DIESER STRING BEGINNT MIT LEERZEICHEN"
130 DATA "DIESER STRING ENTHAELT 1 ,"
```

DCLEAR **Befehl** 7

Schreibweise:

```
DCLEAR [Dlaufwerk] [ON Ugerätenummer]
```

Zweck:

Schließt alle noch offenen Dateien (logischen Dateinummern) auf der angegebenen Floppy. "laufwerk" kann entweder 0 oder 1 sein. Als "gerätenummer" sind Werte zwischen 4 und 15 zulässig. Der Wert 8 ist voreingestellt.

Kommentar:

Wird zur Umlenkung der Bildschirm-Ausgabe mittels CMD (s. dort) ein Kanal eröffnet, so bleibt dieser aktiv, bis er entweder mit DCLEAR oder mit PRINT# geschlossen wird.

DCLOSE **Anweisung** 7

Schreibweise:

```
DCLOSE [logische_dateinummer]
[,ON Ugerätenummer]
```

Zweck:

Schließt eine einzelne Datei oder alle Dateien auf dem angesprochenen Gerät.

Kommentar:

Wird kein Parameter angegeben, so werden alle Dateien auf der Floppy geschlossen; die Voreinstellung für "gerätenummer" ist also 8.

Um nur eine einzelne Datei zu schließen, müssen Sie die "logische_dateinummer" bei DCLOSE angeben, die Sie dieser Datei zuvor mit OPEN oder DOPEN zugewiesen haben.

Beispiele:

```
DCLOSE              (Alle Dateien werden geschlossen)
DCLOSE #2           (Die Datei mit der logischen Nummer 2
                     wird geschlossen)
DCLOSE ON U9        (Alle Dateien der Floppy mit Gerätenummer
                     9 werden geschlossen.
```

DEC — Funktion — 7

Schreibweise:

DEC(str$)

Zweck:

Umwandlung eines Hexadezimalstrings in eine Dezimalzahl.

Kommentar:

Der Parameter "str$" muß ein String sein, der sich als gültige Hexadezimalzahl interpretieren läßt. Er darf also nur die Ziffernzeichen (0 bis 9), die Buchstaben A bis F und beliebig viele Leerzeichen enthalten.

Die größte zulässige Hexadezimalzahl, die in "str$" zugelassen ist, ist "FFFF". Größere Zahlen ergeben einen ILLEGAL QUANTITY ERROR.

Beispiele:

```
PRINT DEC("F0A6")
PRINT DEC("FFFF")
PRINT DEC("  F A C   1  ")              (Ergibt 64193)
PRINT DEC("")                          (Ergibt 0!)
```

DEF FN — Anweisung — 2,7

Schreibweise:

```
DEF FN name(var)=ausdr
```

Zweck:

Definiert eine Funktion namens "name" mit dem Argument "var", deren Wert durch "ausdr" bestimmt ist.

Kommentar:

Mit dieser Anweisung hat der Benutzer die Möglichkeit, komplexe Berechnungen als eigene Funktion zu definieren und unter einem selbstgewählten Namen anzusprechen. Wird die betreffende Berechnung mehrfach in einem Programm benötigt, so kann dadurch sehr viel Platz gespart werden.

Als "name" ist jeder in BASIC zugelassene Variablenname erlaubt. Das in Klammern stehende Argument "var" muß eine numerische Variable sein. Nach dem Gleichheitszeichen kommt ein numerischer Ausdruck, der den Wert der Funktion bestimmt. Da in BASIC die Zahlenwerte 0 und -1 für die Wahrheitswerte falsch und wahr herhalten müssen, sind hier auch logische Ausdrücke erlaubt (s. Beispiel 2). In dem Ausdruck kann die Variable "var" enthalten sein (muß aber nicht!). Wird die selbstdefinierte Funktion benutzt, so hat "var" den Wert, den Sie beim Funktionsaufruf in die runden Klammern geschrieben haben.

Der Aufruf der Funktion im Programm muß in der Form FNname geschehen, wobei "name" der bei der Definition vergebene Name ist. Dies ist nötig, um Verwechslungen mit gleichnamigen Variablen zu verhindern (s. FN-Anweisung).

Die Länge der selbstdefinierten Funktion ist auf eine Zeile beschränkt.

Beispiel:

```
10 DEF FNFLAECHE(R)=R*R*3.14
20 PRINT FNFLAECHE(7), FNFLAECHE(1), FNFLAECHE(13)
```

(Beim ersten Aufruf der Funktion in Zeile 20 erhält R den Wert 7, beim zweiten den Wert 1 und beim letzten den Wert 13)

DELETE — Befehl — 7

Schreibweise:

```
DELETE anfzeile [- endzeile ]
```

Zweck:

Löscht Zeilen aus einem BASIC-Programm.

Kommentar:

Der Befehl kann nur im direkten Modus gegeben werden. Er löscht Einzelzeilen oder ganze Zeilenbereiche aus dem im Arbeitsspeicher befindlichen BASIC-Programm.

Wird nur "anfzeile" angegeben, so wird die Programmzeile mit dieser Nummer gelöscht. Bei Angabe beider Werte werden alle Zeilen gelöscht, die in dem Bereich von "anfzeile" bis "endzeile" liegen.

Fehlt die Angabe "anfzeile", so wird ab Programmbeginn gelöscht; fehlt hinter dem Bindestrich "endzeile", so wird bis zum Programmende gelöscht.

Beispiele:

```
DELETE 30
```
(Löscht Zeile 30)

```
DELETE 13 - 98
```
(Löscht alle Zeilen, die in diesem Bereich liegen - eine Zeile 13 muß es dazu im Programm nicht unbedingt geben!)

```
DELETE  - 150
```
(Löscht vom Programmanfang bis Zeile 150)

```
DELETE 8000 -
```
(Löscht ab Zeile 8000 bis zum Programmende)

DIM — Anweisung — 2,7

Schreibweise:

```
DIM variable(num1,...,numn)
[variable(num1,...,numn) ...]
```

Zweck:

Reserviert Speicherplatz für ein eindimensionales (Array) oder ein mehrdimensionales (Matrix) Feld.

Kommentar:

Ein- oder mehrdimensionale Arrays werden manchmal auch "indizierte Variable" genannt, weil man zum Herausgreifen einer Komponente aus einem Array oder einer Matrix - sei's zum Abspeichern, sei's zum Lesen - einen oder mehrere in runden Klammern stehende Indices angeben muß.

Sollen Zahlen-Arrays mit mehr als 11 Komponenten, mehrdimensionale Felder oder Stringarrays in einem Programm verwendet werden, so müssen diese mittels DIM erst dimensioniert werden, d.h. ihre Größe und die Anzahl der Dimensionen muß vereinbart werden.

Dazu schreibt man in der DIM-Anweisung hinter den Namen der zu dimensionierenden indizierten Variable (Parameter "variable") in runden Klammern für jede Dimension die maximale Anzahl an Elementen (Parameter "num1" ... "numn"). Dies müssen keineswegs Konstanten sein; auch Variable sind erlaubt.

In einer DIM-Anweisung können nach dem gleichen Muster mehrere Dimensionierungen vorgenommen werden.

Es können auch mehrdimensionale Stringfelder vereinbart werden. Dazu muß die "variable" mit dem sattsam bekannten Dollarzeichen für Stringvariable enden.

Die Anzahl der Dimensionen für eine Matrix ist prinzipiell nicht beschränkt. Aber: jede DIM-Anweisung frißt Speicherplatz, so daß man beim Vereinbaren sehr vieler sehr großer Felder bald an die Decke stößt. Am meisten Platz benötigen Stringfelder (pro Komponente maximal 255

Bytes). Integer-Felder brauchen hingegen lediglich zwei Fünftel des Speicherplatzes für Gleitkommazahlen.

Man kann in einem Programm dasselbe Feld nicht mehrfach dimensionieren. Man kann jedoch den vom Feld belegten Speicherplatz mit CLR freigeben (s. dort) und dann ein neues Feld mit dem gleichen Namen dimensionieren. Dann sind aber die alten Werte weg.

Beispiele:

```
DIM A(12,13,14,5,5), A$(3,3)
```
(Dimensioniert eine fünfdimensionale Matrix von Gleitkommazahlen mit dem Namen A, die auf der ersten Dimension 12 ,auf der zweiten 13, auf der dritten 14... Elemente hat sowie eine zweidimensionale Stringmatrix mit Namen A$)

```
DIM A%(50,50)
```
(Dimensioniert eine zweidimensionale Integermatrix)

DIRECTORY **Befehl** 7

Schreibweise:

```
DIRECTORY [Dlaufwerk] [,Ugerätenummer]
[,"namensmuster"]
```

Zweck:

Zeigt das Inhaltsverzeichnis einer Diskette an.

Kommentar:

Das Inhaltsverzeichnis der Diskette in der über die Parameter "laufwerk" (0 oder 1; Voreinstellung 0) und "gerätenummer" (Voreinstellung: 8) wird angezeigt. Man kann auch durch Angabe eines "namensmusters" nur bestimmte Dateien anzeigen lassen. Im "namensmuster" sind die Jokerzeichen "?" und "*" erlaubt.

Ein im Arbeitsspeicher befindliches BASIC-Programm wird nicht gelöscht, wenn man den DIRECTORY-Befehl benutzt.

Ist das Inhaltsverzeichnis länger als 25 Zeilen, so kann mit der Taste No Scroll das Rollen des Bildschirms verhindert werden; die Taste C= verlangsamt die Bildschirmanzeige.

Die Ausgabe des Inhaltsverzeichnisses (Standardlaufwerk 0 und Standardgerät 8) auf dem Drucker ist nicht mit DIRECTORY möglich, sondern erfordert ein eigenes Programm:

```
LOAD "$0",8 : OPEN4,4 : CMD4 : LIST : PRINT#4 : CLOSE4
```

Beispiele:

```
DIRECTORY
```
(Inhaltsverzeichnis der Diskette in LAufwer 0 wird angezeigt)

```
DIRECTORY D1, U8, "DAT*"
```
(Alle Dateien auf Laufwerk eins in Gerät 8, deren Name mit "DAT" beginnt, werden angezeigt)

```
DIRECTORY "PRO????E"
```
(Dieses Namensmuster passt u.a. auf die Dateien PROGRAME und PROMILLE; sollten diese also auf der Diskette sein, so werden sie angezeigt. Es passt jedoch nicht auf PROGRAMME!)

DLOAD **Befehl** 7

Schreibweise:

```
DLOAD "namensmuster"  [,Dlaufwerk]
[,Ugerätenummer]
```

Zweck:

Lädt eine Programmdatei von Diskette in den Arbeitsspeicher.

Kommentar:

Der Name der zu ladenden Datei (Parameter "namensmuster") muß angegeben werden. Dabei sind jedoch die Jokerzeichen "?" und "*" zulässig.

Der DLOAD-Befehl kann auch in einem BASIC-Programm verwendet werden, um aus dem Programm heraus ein anderes Programm aufzurufen und mit RUN zu starten. Alle Variablen des ersten (aufrufenden) Programmes bleiben erhalten.

Beispiele:

```
DLOAD "DIS*"
```
(Lädt das erste Programm auf der Diskette, das auf das Namensmuster "DIS*" paßt, also z.B. ein Programm namens DISKDOC)

```
10 A$ = "CHAIN"
20 DLOAD (A$)
30 RUN
```

DO...LOOP Anweisung 7

Schreibweise:

```
DO [UNTIL bedingung]
.
.     (anweisungen)
[EXIT]
.
LOOP [UNTIL bedingung]
```

oder:

```
DO [WHILE bedingung]
.
.     (anweisungen)
[EXIT]
.
LOOP [WHILE bedingung]
```

Zweck:

Definiert eine Programmschleife und steuert deren Abarbeitung.

Kommentar:

Mit den Befehlswörtern DO, LOOP, UNTIL, WHILE und EXIT lassen sich auf bequeme Weise eine große Vielzahl unterschiedlicher Programmschleifen formulieren. Diese Sprachelemente ermöglichen es, in BASIC strukturiert zu programmieren.

Die Wirkung der Anweisungen: alles, was in einem Programm zwischen DO und LOOP liegt, wird wiederholt ausgeführt. Die zwischen diesen beiden Zauberworten stehenden Anweisungen nennt man auch den "Schleifenrumpf". Dieser kann eine BASIC-Zeile lang sein, mehrere Zeilen oder auch gar keine umfassen. Die "strukturierte" Art, eine Endlosschleife zu schreiben, lautet also "10 DO : LOOP", und nicht mehr - wie im BASIC 2.0 - "10 GOTO 10". Das erste Beispiel zeigt eine Endlosschleife mit zwei Anweisungen im Schleifenrumpf.

Natürlich muß man eine Schleife auch wieder mal verlassen können. Dazu dient der Befehl EXIT. Er kann irgendwo im Schleifenrumpf stehen und bewirkt das sofortige Verlassen des Schleifenkörpers (genauer: es wird mit der Anweisung hinter LOOP fortgefahren). Meist wird man die EXIT-Anweisung von einer IF-Entscheidung kontrollieren lassen. Dies demonstriert Beispiel 2; beachten Sie, daß hier auch noch die Meldung "ENDE" gedruckt wird, ehe die Schleife nach zwanzigmaligem Durchlauf verlassen wird.

Nur mit DO..LOOP und EXIT gebildete Schleifen sind aber immer noch nicht strukturiert genug im Sinne der reinen Lehre. Es gibt nämlich die Möglichkeit, eine Schleife solange laufen zu lassen, bis eine bestimmte Bedingung wahr ist (nichts anderes macht das Beispiel 2!), aber diese Bedingung nicht irgendwo im Schleifenrumpf zu verstecken, sondern klar und übersichtlich gleich am Anfang der Schleife zu notieren. Dazu benutzt man das Schlüsselwort UNTIL. Beispiel 3 arbeitet damit und hat genau die gleiche Wirkung wie Beispiel 2, liest sich jedoch viel schöner.

Beispiel 3 ist ein Beispiel für eine sogenannte "abweisende" Schleife; ist nämlich die hinter UNTIL stehende "bedingung" bereits bei Eintritt in die Schleife erfüllt, so wird der Schleifenrtumpf gar nicht erst durchlaufen. Dies zeigt Beispiel 4 noch einmal deutlicher. Hier wird der Stern nicht gedruckt; auch wird A nicht hochgezählt.

Selbstverständlich gibt es auch nichtabweisende Schleifen; diese werden auf jeden Fall mindestens einmal durchlaufen. Dazu muß lediglich das Schlüsselwort UNTIL nicht an den Anfang der Schleife (hinter DO), sondern eben an ihr Ende (hinter LOOP) geschrieben werden. Beispiel 5 demonstriert dies. Beachten Sie, daß sowohl der Stern, als auch ENDE und der (um 1 erhöhte!) Wert von A ausgegeben wird.

Anstatt Schleifen solange durchlaufen zu lassen, bis (englisch "until") eine Bedingung eintritt, kann man den Rumpf auch wiederholen lassen, solange (englisch "while") eine Bedingung wahr ist. Diese WHILE-Schleifen gibt es auch wieder in einer abweisenden und einer nichtabweisenden Variante, je nachdem, ob Sie das WHILE hinter DO oder hinter LOOP schreiben. Hierzu die Beispiele 6 und 7, welche beide 10 Sternchen drucken.

Die "bedingung", mit der eine Schleife kontrolliert wird, kann auch zusammengesetzt sein (z.B. mit AND, OR und deren Verwandten).

Selbstverständlich kann in jeder Schleife - ob UNTIL, ob WHILE, ob abweisend oder nicht - eine oder mehrere EXIT-Anweisungen auftauchen. Aber seien Sie sparsam mit deren Gebrauch; Sie machen sich sonst die ganze schöne Übersichtlichkeit kaputt!

Neben der Übersichtlichkeit haben DO...LOOP-Schleifen einen Vorteil vor FOR-Schleifen: in ihnen ist es erlaubt, Integer-Variablen als Zähler zu verwenden. Das kann die Schleifen schneller machen.

DO...LOOP-Schleifen können nach denselben Regeln geschachtelt werden wie FOR...NEXT-Schleifen.

Die UNTIL- und WHILE-Anweisungen können sowohl am Anfang als auch am Ende der Schleife stehen, ja sie können sogar zweimal (einmal am Anfang, einmal am Ende) erscheinen und gemischt werden (vgl. das letzte Beispiel; ob das aber noch strukturiert ist...).

Beispiele:

```
10 DO                    : REM BEISPIEL 1
20 :  PRINT "*";
30 :  PRINT "-";
40 LOOP

10 DO                    : REM BEISPIEL 2
20 :  A = A + 1
30 :  IF A = 20 THEN EXIT
40 LOOP : PRINT "ENDE"
50 PRINT A

10 DO UNTIL A = 20       : REM BEISPIEL 3
20 :  A = A + 1
30 LOOP : PRINT "ENDE"
40 PRINT A

10 A = 30                : REM BEISPIEL 4
20 DO UNTIL A > 20
30 :  A = A + 1
40 :  PRINT "*"
50 LOOP : PRINT "ENDE"
60 PRINT A

10 A = 30                : REM BEISPIEL 5
20 DO
30 :  A = A + 1
40 :  PRINT "*"
50 LOOP UNTIL A > 20 : PRINT "ENDE"
60 PRINT A

10 DO WHILE A < 10       : REM BEISPIEL 6
20 :  PRINT "*";
30 :  A = A + 1
40 LOOP

10 DO                    : REM BEISPIEL 7
20 :  PRINT "*";
30 :  A = A + 1
40 LOOP WHILE A < 10

10 DO UNTIL X >= 5       : REM EINFACH GEMOPPELT
20 :  X = X + 1
30 LOOP WHILE X <= 5     : REM DOPPELT GEMOPPELT
```

DOPEN — Anweisung — 7

Schreibweise:

```
DOPEN #logische_gerätenummer, "dateiname"
[,Lrecordlänge] [,Dlaufwerk]
[,ON Ugerätenummer] [W]
```

Zweck:

Öffnet eine Datendatei zum Lesen und/oder Schreiben.

Kommentar:

Soll eine Relativdatei geöffnet werden, so muß der Parameter "recordlänge" angegeben werden; er bestimmt die Länge der Records in der Datei.

Soll eine sequentielle Datei zum Schreiben eröffnet werden, so muß der Parameter "W" angegeben werden. Fehlt er, so wird eine sequentielle Datei mit Lesezugriff eröffnet.

Die geöffnete Datei hat den Namen "dateiname" (muß in Anführungszeichen stehen!) und kann im Programm unter der Nummer angesprochen werden, die Sie als "logische_gerätenummer" vergeben haben.

Beispiele:

```
DOPEN #2,"ADRESSEN",W
```
(Öffnet die sequentielle Datei "ADRESSEN" zum Schreiben und weist ihr die logische Nummer 2 zu)

```
DOPEN #4,"INDEX",L25,D0,U9
```
(Öffnet die relative Datei (Direktzugriffsdatei) "INDEX" auf LAufwerk 0 der Floppy mit Gerätenummer 9, wobei die logischen Sätze der Datei eine Länge von 25 Bytes haben. Die Datei wird zum Schreiben geöffnet und hat die Nummer 4)

```
DOPEN#1,"KUNDEN"
```
(Die sequentielle Datei KUNDEN wird zum Lesen geöffnet)

DRAW — Grafik-Befehl — 7

Schreibweise:

```
DRAW farbquelle [, x1, y1 [TO x2, y2
...]]
```

Zweck:

Zeichnet auf dem Grafikbildschirm Einzelpunkte, Linien oder Polygonzüge. Mögliche Werte für den Parameter "farbquelle:"

0	Hintergrundfarbe
1	Vordergund-(Zeichen-)Farbe
2	Multicolor 1
3	Multicolor 2

Kommentar:

Nachdem die Farbquelle gewählt ist, wird auf dem Grafikbildschirm entweder ein Punkt mit den Koordinaten "x1" und "y1" gezeichnet, oder, falls "TO x2,y2" angegeben ist, eine Linie mit Startpunkt bei "x1,y1" und Endpunkt bei "x2,y2". Danach können beliebig viele weitere TO-Angaben folgen, wodurch komplexe geometrische Figuren gezeichnet werden.

Es ist auch möglich, nur mehrere Koordinatenpaare anzugeben, ohne diese mit TO zu verbinden. Dann werden nur die Einzelpunkte gezeichnet, aber nicht über eine Linie verbunden.

Fehlt bei einer TO-Angabe der Startpunkt, so wird die aktuelle Position des Pixel-Cursors als Startpunkt benutzt.

Das erste Beispielprogramm zeichnet ein Dreieck mit einem Rahmen drumrum. Das zweite läßt den Sternenhimmel aufleuchten (mit STOP-Taste abbrechen!).

Beispiele:

```
10 GRAPHIC 1 : SCNCLR
20 DRAW 1, 100,100 TO 160,40 TO 220,100 TO 100,100
30 DRAW 1, 9,9 TO 9,190 TO 310,190 TO 310,9 TO 9,9
```

```
10 GRAPHIC 1 : SCNCLR : REM STERNENHIMMEL
20 Y = RND(-TI)
30 DO
40 :  X=RND(Y)*320 : Y=RND(X)*200
50 :  DRAW 1,X,Y
60 LOOP
```

DSAVE **Befehl** 7

Schreibweise:

```
DSAVE "dateiname" [,Dlaufwerk]
[,Ugerätenummer]
```

Zweck:

Speichert ein Programm auf Diskette. Der Dateiname muß angegeben werden.

Kommentar:

Zum Abspeichern auf Kassette müssen Sie mit SAVE arbeiten.

Der Dateiname kann auch als Variable angegeben werden, die dann allerdings in runden Klammern stehen muß.

DSAVE kann auch in einem Programm verwendet werden; nach seiner Ausführung kehrt BASIC aber auf jeden Fall in den Direktmodus zurück.

Beispiel:

```
DSAVE "UMSATZ"
```

DVERIFY **Befehl** 7

Schreibweise:

```
DVERIFY "dateiname" [,Dlaufwerk]
[,Ugerätenummer] [verschieb]
```

Zweck:

Vergleicht ein in einer Datei auf Diskette gespeichertes Programm mit dem Programm im Hauptspeicher. Der Parameter "dateiname" muß angegeben werden.

Kommentar:

Stimmen die verglichenen Programme genau überein, so erfolgt die Meldung OK. Bei Unterschieden meldet DVERIFY sich mit "VERIFY ERROR".

Hat der Parameter "verschieb" den Wert 0, so werden BASIC-Programme verglichen (Voreinstellung). Ein Wert von 1 dient zum Vergleichen von Maschinenprogrammen, die an einer anderen Startadresse beginnen (s. LOAD-Anweisung).

Wenn Sie ein Programm mit SAVE oder DSAVE abspeichern, dann einen Grafikbereich reservieren oder freigeben (s. GRAPHIC-Anweisung) und anschließend den Vergleich mit DVERIFY anstellen, dann wird auch bei völlig identischen Programmen die Meldung "VERIFY ERROR" ausgegeben. Dies kommt daher, daß durch das Freigeben oder Zuweisen von Platz für den Grafikspeicher das BASIC-Programm im Hauptspeicher verschoben wird. Somit stimmen seine Adreßbezüge nicht mehr mit der auf der Diskette gespeicherten Version überein.

Beispiel:

```
DVERIFY "TESTPROG"
```

END — Anweisung — 2,7

Schreibweise:

END

Zweck:

Beendet die Programmausführung.

Kommentar:

Das Programm wird an der Zeile angehalten, in der die END-Anweisung steht. Folgen dahinter noch weitere Programmzeilen, so kann mit CONT das Programm fortgesetzt werden.

ENVELOPE — Sound-Anweisung — 7

Schreibweise:

```
ENVELOPE n, [, a, d, s, r, w, p]
```

Zweck:

Definiert eine von 10 möglichen Hüllkurven. Die Parameter und ihre möglichen Werte:

n:	Nummer der Hüllkurve (0 bis 9)
a:	Anschlagzeit ("attack"; von 0 bis 15)
d:	Abschwellzeit ("decay"; von 0 bis 15)
s:	Haltezeit ("sustain"; von 0 bis 15)
r:	Ausklingzeit ("release"; von 0 bis 15)
w:	Wellenform: 0 = Dreieck 1 = Sägezahn 2 = Rechteck 3 = Rauschen 4 = Ringmodulation
p:	Impulsbreite ("pulse width"; 0 bis 4095)

Kommentar:

Die mit ENVELOPE definierten Hüllkurven können in der PLAY-Anweisung (s. dort) verwendet werden.

Die Parameterwerte für "a", "d" und "r" legen Zeitmaße fest, während sich der Parameter "s" auf die Amplitude (Lautstärke) bezieht. Die tatsächliche Dauer der Haltezeit wird erst in der PLAY-Anweisung festgelegt. Die tatsächliche Zeitdauer von Anschlagzeit, Abschwellzeit und Ausklingzeit für unterschiedliche Parameterwerte zeigt folgende Tabelle:

Parameterwert	Anschlag	Abschwellen/Ausklingen
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1,5 s
11	800 ms	2,4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Beim Einschalten des Gerätes sind bereits 10 Hüllkurven vordefiniert. Deren Definition sieht folgendermaßen aus:

n	a	d	s	r	w	p	Instrument
0	0	9	0	0	2	1536	Klavier
1	12	0	12	0	1		Akkordeon
2	0	0	25	0	0		Zirkusorgel
3	0	5	5	0	3		Trommel
4	9	4	4	0	0		Flöte
5	0	9	2	1	1		Gitarre
6	0	9	0	0	2	512	Cembalo
7	0	9	9	0	2	2048	Orgel
8	8	9	4	1	2	512	Trompete
9	0	9	0	0	0		Xylophon

ERR$ — Funktion — 7

Schreibweise:

ERR$(n)

Zweck:

Liefert den Text für den Fehler mit der Nummer "n".

Kommentar:

Die Funktion hat als Wert einen String mit einem Text, der den Fehler mit der angegebene Nummer (Parameter "n") näher erläutert. Damit können Sie unter Zuhilfenahme der Anweisungen TRAP und RESUME sowie der Fehlervariablen ER und EL eine komfortable Fehlerbehandlung in Ihr Programm einbauen.

Das Beispielprogramm listet alle Fehlermeldungen des BASIC 7.0

Kommentar:

Falls "str$" ein Einzelzeichen ist, so wird dessen Positionsnummer in der Zeichentabelle (ASCII-Wert) als Dezimalzahl ausgegeben. Ist "str$" ein String, so untersucht die Funktion nur das erste Zeichen im String.

Die Funktion ist z.B. dann nützlich, wenn man mit Einzelzeichen rechnen will.

Hinsichtlich der Behandlung des leeren Strings unterscheiden sich der 64er und 128er-Modus:

64er-Modus: Falls "str$" der leere String ist (also ""), wird die Fehlermeldung ILLEGAL QUANTITY ERROR ausgegeben.

128er-Modus: Falls "str$" der leere String ist, so liefert die Funktion den Wert 0 (ohne Fehlermeldung).

Beispiel:

```
10 DO UNTIL N > 40
20 :  N = N + 1
30 :  PRINT ERR$(N)
40 LOOP
```

EXP — Funktion — 2,7

Schreibweise:

EXP(x)

Zweck:

Ergibt die "n"-te Potenz von e.

Kommentar:

BASIC arbeitet mit einem Näherungswert für e von 2.71828183. Der höchste zulässige Wert für "n" ist 88.0296919.

Beispiel:

```
PRINT EXP(3.14)
```

FAST — Anweisung — 7

Schreibweise:

FAST

Zweck:

Schaltet den Systemtakt um. Der Prozessor wird nun nicht mehr mit einem Megahertz, sondern mit zwei Megahertz Taktfrequenz betrieben und ist damit doppelt so schnell.

Kommentar:

IM FAST-Modus hat der VIC-Chip keine Zeit mehr für den Bildschirmaufbau; man sieht also nichts - weder auf dem Text- noch auf dem Grafikbildschirm. Anders ist es mit dem 80-Zeichen-Bildschirm, da dieser ja von einem eigenen Chip verwaltet wird.

Man sollte den FAST-Modus also dann anwenden, wenn rechenintensive Programmteile anstehen. Nach Umschalten mittels SLOW ist der alte Bildschirm wieder da.

Beispiel:

```
10 FAST : REM 3 SEKUNDEN FUNKSTILLE
20 SLEEP 3
30 SLOW
```

FETCH **Anweisung** 7

Schreibweise:

```
FETCH anzahl, ziel, bank, quelle
```

Zweck:

Überträgt eine wählbare Anzahl Bytes (Parameter "anzahl") aus einer Speichererweiterung-Bank (Parameter "bank") in den BASIC-Arbeitsspeicher (Bank 1).

Kommentar:

Der Parameter "ziel" bestimmt die Adresse in Bank 1, ab der die Bytes "eingespielt" werden sollen; der Parameter "quelle" gibt an, von wo ab die Bytes übertragen werden sollen.

Beispiel:

```
FETCH 1024,52000,6,1215
```

(Überträgt 1024 Bytes aus Bank 7 in Bank 1. Das erste übertragene Byte aus Bank 7 stammt von Adresse 1215; es landet in Adresse 52000 des BASIC-Speichers. Das zweite Byte stammt von Adresse 1216 und landet auf Adresse 52001 usw.)

FILTER — Sound-Anweisung — 7

Schreibweise:

```
FILTER [frequenz] [, tiefpaß] [, bandpaß]
[,hochpaß] [, resonanz]
```

Zweck:

Definiert die Parameter für den Klangfilter. Die Bedeutung der einzelnen Parameter:

frequenz:	die Grenzfrequenz für das Filter (0 bis 2047)
tiefpaß:	Tiefpaßfilter ein (0) oder aus (1)
bandpaß:	Bandpaßfilter ein (0) oder aus (1)
hochpaß:	Hochpaßfilter ein (0) oder aus (1)
resonanz:	Resonanz (0 bis 15)

Kommentar:

Die Parameter "tiefpaß", "hochpaß" und "bandpaß" können alle zugleich gesetzt sein; dadurch entstehen Bandsperr-Effekte.

Der Parameter "resonanz" bestimmt, inwieweit ein Ton hart oder weich klingt.

Ist ein Parameter nicht angegeben, so bleibt der letzte eingestellte Wert gültig.

Die tatsächliche Grenzfrequenz entspricht nicht direkt dem Parameter "frequenz", sondern errechnet sich nach der Formel "frequenz" * 5.8 + 30.

Das Filter kommt erst zur Wirkung, wenn es (in der PLAY-Anweisung; s. dort) eingeschaltet wird.

Beispiel:

```
FILTER 1000, 0, 0, 1, 10
```
(setzt die Grenzfrequenz auf 1000, schaltet den Tief- und Hochpaßfilter aus sowie den Bandpaßfilter ein und setzt die Resonanz auf 10.)

FNname — Funktion — 2,7

Schreibweise:

```
FNname(x)
```

Zweck:

Wendet eine vom Benutzer definierte Funktion "name" auf ihr Argument "x" an.

Kommentar:

Eine zuvor mit DEF FN (s. dort) definierte Funktion wird aufgerufen. "x" ist das Argument, mit dem die in der Definition bestimmten Berechnungen ausgeführt werden.

Beispiel:

```
10 DEF FNFLAECHE(R)=R*R*3.14
.
.
110 PRINT FNFLAECHE(13), FNFLAECHE(14), FNFLAECHE(12)
```

FOR...NEXT Anweisung 2,7

Schreibweise:

```
FOR variable = anfwert TO  endwert
[STEP schrittweite]
.
. (anweisungen)
.
NEXT
```

Zweck:

Leitet eine Zählschleife (FOR-Schleife) ein. Die Anweisungen zwischen FOR und NEXT (der Schleifenrumpf) werden sooft durchlaufen, bis die Zählvariable "variable" den angegebenen Endwert (Parameter "endwert") erreicht hat. Nach jedem Durchlauf wird die Schleifenvariable "variable" um den Wert "schrittweite" erhöht, falls die STEP-Anweisung gegeben wurde; fehlt die Angabe, wird der Standardwert 1 verwendet.

Kommentar:

Es ist nicht möglich, eine Integervariable als Zählvariable zu verwenden. Der Versuch führt auf einen SYNTAX ERROR.

Ist bei Eintritt in die Schleife der Anfangswert der Schleifenvariable größer als der Endwert, dann wird die Schleife dennoch einmal durchlaufen. (Beispiel 1)

Durch Angeben einer negativen Schrittweite ist es auch möglich, in der Schleife rückwärts zu zählen. (Beispiel 2)

Ist bei negativer Schrittweite der Anfangswert kleiner als der Endwert, so wird die Schleife dennoch einmal durchlaufen (Beispiel 3)

Beim Austritt aus der Schleife hat die Schleifenvariable nicht den Endwert, der getestet wird, sondern sie wurde noch einmal um "schrittweite" erhöht (Beispiel 4).

Achtung! Bei sehr kleinen Schrittweiten können Rundungsfehler auftreten, so daß die Schleifenvariable nach Verlassen der Schleife nicht unbedingt den erwarteten Endwert hat (Beispiel 5).

Beispiele:

```
FOR I = 3 TO 1 : PRINT "IN DER SCHLEIFE" : NEXT

FOR I = 10 TO 5 STEP -1 : PRINT I : NEXT

FOR I = 3 TO 5 STEP -1 : PRINT I : NEXT

FOR I = 3 TO 6 STEP 11 : PRINT I : NEXT : PRINT 1

10 FOR I = 1 TO 1.1 STEP 0.001
20 NEXT I
30 PRINT I
```

(Ergebnis: 1.10000002 statt 1.1001)

FRE — Funktion — 2(!),7

Schreibweise:

```
FRE(n)
```

Zweck:

Gibt die Anzahl der freien Bytes im Hauptspeicher an.

64-er Modus: Der Parameter "n" hat keine Bedeutung; es muß aber ein Wert angegeben werden.

128-er Modus: Falls "n" gleich 0 ist, wird der freie Platz im Programmspeicher (Bank 0) zurückgegeben; hat "n" den Wert 1, dann erfahren Sie den Platz im Variablen-Bereich (Bank 1).

Beispiel:

```
10 PRINT "PLATZ IM TEXTSPEICHER: "; FRE(0)
20 PRINT "PLATZ IM VARIABLENSPEICHER: "; FRE(1)
```

GET — Anweisung — 2,7

Schreibweise:

```
GET var
```

Zweck:

Liest ein Zeichen von der Tastatur.

Kommentar:

Wurde bei Aufruf der Anweisung GET eine Taste gedrückt, so wird das eingegebene Zeichen der Variablen "var" zugewiesen. Wurde keine Taste gedrückt, so wird der Variablen der leere String "" zugewiesen.

Ist "var" keine Stringvariable, so ergibt dies einen Fehler, falls eine andere als eine Zifferntaste betätigt wurde.

Anstelle einer einzelnen Variable kann für "var" auch eine Liste von Variablen angegeben werden, die durch Komma getrennt sind. Dann müssen so viele Tasten gedrückt werden, wie Variablen in der Liste enthalten sind.

Die GET-Anweisung darf nur in einem Programm benutzt werden; im Direkt-Modus ist sie nicht erlaubt.

Zeichen werden durch GET gelesen, ohne daß die RETURN-Taste (zum "wegschicken") gedrückt werden muß; vielmehr ist es mit GET sogar möglich, die RETURN-Taste (CHR$(13)) zu lesen.

Im BASIC 7.0 gibt es mit der Anweisung GETKEY (s. dort) eine bequemere Möglichkeit, die Tastatur abzufragen.

Das Beispiel zeigt die Standardmethode zur Abfrage der Tastatur im 64er-Modus.

Beispiele:

```
10 GET A$ : IF A$ = "" THEN GOTO 10
20 PRINT "GELESENES ZEICHEN: ";A$;" DEZIMAL: ";ASC(A$)
```

GET#

Anweisung 2,7

Schreibweise:

```
GET# logische_dateinummer, var
```

Zweck:

Liest Zeichen von einem beliebigen Eingabegerät oder einer Datei, denen zuvor mit OPEN oder DOPEN (s. dort) eine "logische_dateinummer" zugewiesen wurde.

Kommentar:

Die Funktionsweise dieser Anweisung entspricht der von GET (s. dort).

GETKEY — Anweisung — 7

Schreibweise:

```
GETKEY var
```

Zweck:

Liest Zeichen von der Tastatur.

Kommentar:

GETKEY entspricht in seiner Funktionsweise weitgehend der GET-Anweisung (s. dort), wartet jedoch selbst auf einen Tastendruck. Sie müssen mit dieser Anweisung also nicht länger selbst überprüfen, ob eine Taste gedrückt wurde (s. Beispiel zu GET).

Anstelle einer einzelnen Variable kann für "var" auch eine Liste von Variablen angegeben werden, die durch Komma getrennt sind. Dann müssen so viele Tasten gedrückt werden, wie Variablen in der Liste enthalten sind.

GETKEY darf nur in Programmen (also nicht im Direktmodus) verwendet werden.

Das Beispiel zeigt einen Programmausschnitt, der dafür sorgt, daß nur Buchstabentasten ausgewertet werden.

Beispiele:

```
10 DO
20 :   GETKEY A$
30 LOOP WHILE A$<"A" OR A$>"Z"
```

GO64 Befehl 7

Schreibweise:

GO64

Zweck:

Schaltet vom 128er-Modus in den 64er-Modus um.

Kommentar:

Ehe Sie in den 64er-Modus gelangen, werden Sie vorsichtshalber gefragt (ARE YOU SURE?) und müssen mit "Y", "YEAH", "YAWN" oder irgend einem anderen Wort, das mit "Y" beginnt, bestätigen. Dies kommt daher, weil es aus dem 64er-Modus kein Zurück mehr in den 128er-Modus gibt. Ein im Arbeitsspeicher befindliches 128er-Programm ist nach dem Moduswechsel verloren.

Der Befehl kann im Direktmodus, aber auch in einem Programm gegeben werden.

GOSUB — Anweisung — 2,7

Schreibweise:

```
GOSUB zeilennummer
```

Zweck:

Die Programmausführung wird mit dem Unterprogramm fortgesetzt, das in der Zeile "zeilennummer" beginnt.

Kommentar:

Die GOSUB-Anweisung ähnelt der Sprunganweisung GOTO (s. dort). BASIC merkt sich jedoch im Unterschied zu GOTO, in welcher Zeile die GOSUB-Anweisung steht. Trifft es im Unterprogramm auf eine RETURN-Anweisung, so wird diese verlassen und die Programmausführung wird mit der nächsten Anweisung hinter dem GOSUB fortgesetzt.

Unterprogramme können auch geschachtelt werden, d.h. es ist möglich, in einem Unterprogramm ein anderes aufzurufen. Es ist sogar zulässig, daß ein Unterprogramm sich selbst aufruft. Dadurch sind in BASIC auch rekursive Berechnungen möglich. Das Beispielprogramm zeigt, wie dies zur Berechnung der Fakultäts-Funktion eingesetzt werden kann.

Beispiel:

```
10 DIM R(500)
20 DO
30 :   INPUT N
40 :   GOSUB 1000
50 :   PRINT "FAK(";N;") = ";R(N)
60 LOOP WHILE N > 0
70 :
1000  REM REKURSIVES UNTERPROGRAMM
1010  IF N = 0 THEN R(N) = 1 : RETURN
1020  N = N - 1
1030  GOSUB 1000 :  REM REKURSIVES AUFFUELLEN DES FELDES
1040  N = N + 1
1050  R(N) = N * R(N-1)  :  REM REKURSIVES BERECHNEN
1060  RETURN
```

GOTO — Anweisung — 2,7

Schreibweise:

```
GOTO zeilennummer
oder:
GO TO zeilennummer
```

Zweck:

Die Programmausführung wird mit der Anweisung fortgesetzt, die in der Zeile "zeilennummer" steht.

Kommentar:

Im Direktmodus hat man mit GOTO die Möglichkeit, ein Programm bei einer beliebigen Programmzeile zu starten, ohne daß zuvor die Variablen gelöscht werden.

GRAPHIC — Grafik-Anweisung — 7

Schreibweise:

```
GRAPHIC modus [,löschen] [,text]
oder
GRAPHIC CLR
```

Zweck:

Aktiviert den gewünschten Grafikmodus und reserviert Speicherplatz für den Grafikbildschirm bzw. gibt den reservierten Speicherplatz wieder frei. Die Parameter und ihre zulässigen Werte sind:

modus:

0	Textmodus mit 40 Zeichen/Zeile
1	hochauflösende Grafik (320 mal 200 Pixel)
2	geteilter Bildschirm: hochauflösende Grafik und Text gemischt
3	Multicolor-Bildschirm (160 mal 200 Pixel)
4	geteilter Bildschirm: Multicolor und Text gemischt
5	Textbildschirm mit 80 Zeichen/Zeile

löschen:
der gewählte Grafikbildschirm wird gelöscht (1) oder bleibt unverändert (0).

text:
ein ganzzahliger Wert zwischen 0 und 24, der bei den Modi 2 und 4 festlegt, in welcher Zeile der Textbildschirm beginnen soll. Standardwert: 19

Kommentar:

Beim Einschalten befindet sich der C128 im 40-Zeichen-Textmodus. Um einen anderen Grafikmodus anzuwählen, müssen Sie die GRAPHIC-Anweisung benutzen.

Aktivieren Sie einen der Grafikbildschirme, dann wird im Programmbereich (Bank 0) ein Speicherbereich von 10K am Anfang des Textbereichs zugewiesen, wobei der Textbereich und damit auch ein BASIC-Programm im Speicher nach oben verschoben wird. Der für BASIC-Programme verfügbare Platz wird dadurch kleiner (können Sie mit FRE(0) nachprüfen; s. dort).

Der Speicherbereich bleibt auch zugewiesen, wenn Sie mittels GRAPHIC 0 wieder in den 40-Zeichen-Bildschirm zurückkehren. Erst durch die Anweisung GRAPHIC CLR wird er wieder freigegeben (und das BASIC-Programm

wieder nach unten verschoben, was Auswirkungen auf die DVERIFY-Anweisung haben kann; s. dort).

Sollte Ihr BASIC-Programm sehr umfangreich sein, so ist es dem GRAPHIC-Kommando eventuell nicht möglich, Speicherplatz zu reservieren; Sie erhalten dann eine Fehlermeldung.

Beispiele:

```
GRAPHIC 0,1
```
(Schaltet in den Textbildschirm -- 40 Zeichen -- um und löscht diesen)

```
GRAPHIC 2,0,15
```
(Grafikbildschirm geteilt; Text beginnt bei Zeile 15)

```
GRAPHIC 2,1
```
(Grafikbildschirm geteilt und gelöscht; Text beginnt bei Zeile 19)

```
GRAPHIC CLR
```
(Für Grafik-Bildschirme zugewiesener Speicherplatz wird wieder freigegeben)

GSHAPE — Grafik-Anweisung — 7

Schreibweise:

```
GSHAPE str$, x, y, modus
```

Zweck:

Zeichnet ein im String "str$" abgespeichertes Shape ab Position "x,y" im Modus "modus" auf den Grafik-Bildschirm. Die Parameter und ihre Werte:

x:		x-Koordinate der linken oberen Ecke des Shape (0 bis 319 oder 0 bis 159 im Multicolor-Modus bzw. 0 bis 1024, falls die SCALE-Anweisung gegeben wurde)
y:		y-Koordinate des linken oberen Eckpunkts (0 bis 199).
modus	0	unveränderte Übertragung
	1	invertierter Modus
	2	OR-Modus
	3	UND-Modus
	4	XOR-Modus

Kommentar:

Der Inhalt einer Stringvariable wird als Grafikinformation auf den Grafikbildschirm übertragen. Üblicherweise hat die Variable ihren Inhalt zuvor mit der Anweisung SSHAPE (s. dort) erhalten.

Durch Angabe der Koordinaten "x" und "y" bestimmen Sie, von wo ab mit der Übertragung begonnen werden soll. So ist es möglich, ein Shape an einer beliebigen Stelle des Bildschirms erscheinen zu lassen und sogar über den Bildschirm zu bewegen. (Zeichentrick-Effekte lassen sich jedoch einfacher und schneller mit Sprites erreichen; vgl. z.B. MOVSPR).

Der Parameter "modus" bestimmt, wie die Übertragung vorgenommen wird. Hat er den Wert 0 (Voreinstellung), so erscheint das Shape genauso, wie es zuvor abgespeichert wurde.

Ein "modus"-Wert von 1 sorgt dafür, daß das Shape invertiert dargestellt wird: Bildpunkte in Vordergrundfarbe werden in Hintergrundfarbe angezeigt und umgekehrt. (Zum Setzen der Farben vgl. die COLOR-Anweisung).

Hat "modus" den Wert 2, so werden die Bildpunkte des Shapes zu den Bildpunkten eines bereits auf dem Bildschirm vorhandenen Untergrundes "addiert": Ein Bildpunkt in der Ergebniszeichnung wird eingeschaltet, wenn er entweder im Shape oder im Untergrund eingeschaltet ist. Mehrere Shapes können so übereinandergelegt werden.

Für "modus" gleich 3 wird ein Bildpunkt im Ergebnis nur dann eingeschaltet, wenn er sowohl im Shape als auch im bereits vorhandenen Untergrund gesetzt ist. Zeichnen Sie also ein Shape im Modus 3 auf einen leeren Bildschirm (einen leeren Untergrund), so sehen Sie gar nichts.

Der Modus 4 setzt Bildpunkte nur dann, wenn sie sich im Shape und im Untergrund unterscheiden. Trifft ein eingeschalteter Shape-Bildpunkt auf einen ausgeschalteten Untergrund-Punkt, so wird das Pixel eingeschaltet. Ebenso sehen Sie im Ergebnis etwas, wenn die fragliche Koordinate im Shape aus- und im Untergrund eingeschaltet ist. Sind die Pixel im Shape und im Untergrund gleich (beide eingeschaltet oder beide ausgeschaltet), so bleibt der Bildpunkt im Ergebnis ausgeschaltet.

Der Modus 4 ist nützlich, um ein Shape über den Bildschirm zu bewegen, ohne daß die darauf enthaltene Information gelöscht wird. Das Programmbeispiel demonstriert dies.

Beispiel:

```
10 GRAPHIC 2,1
20 CIRCLE 1,10,8,10,8  :  REM KREIS ZEICHNEN
30 SSHAPE A$,0,0,21,18 :  REM UND IN A$ SPEICHERN
40 SCNCLR
50 CHAR 1,0,0,"DIESER STRING WIRD UEBERSCHRIEBEN!"
60 FOR X = 0 TO 295 STEP 2 : REM ENTLANG DER X-ACHSE
70 GSHAPE A$,X,0,4      :  REM DEN KREIS VERSCHIEBEN
80 FOR D = 1 TO 50 : NEXT  D : REM KURZE PAUSE
90 GSHAPE A$,X,0,4      : KREIS AM ALTEN ORT LOESCHEN MIT XOR
100 NEXT X
```

HEADER **Befehl** 7

Schreibweise:

```
HEADER diskname [Ikennung], [Dlaufwerk]
[,ONgerätenummer]
```

Zweck:

Formatiert eine Diskette in einem wählbaren Laufwerk und gibt ihr den Namen "diskname".

Kommentar:

"diskname" kann maximal 16 Zeichen lang sein. Er kann im HEADER-Befehl als Konstante angegeben sein (und muß dann in Anführungszeichen stehen), oder als Variable in runden Klammern.

Die "kennung" ist zwei Zeichen lang. Beim Formatieren einer fabrikfrischen Diskette muß sie unbedingt angegeben werden. Sie wird in den Header jeden Diskettensektors eingetragen und findet sich außerdem am Anfang des Directory.

Wird eine bereits formatierte Diskette mit HEADER neu formatiert, dann kann die Angabe der "kennnung" unterbleiben; es wird dann für die Neuformatierung die bereits vorhandene Kennung verwendet. In diesem Fall geht der Formatierungsvorgang auch schneller vor sich, da nur das Inhaltsverzeichnis gelöscht und nicht auch der gesamte Datenbereich neu formatiert wird.

Wollen Sie eine gebrauchte Diskette also völlig neu formatieren, dann müssen Sie die Kennung angeben.

Das Formatieren einer Diskette löscht alle auf dieser Diskette vorhandenen Informationen. Deswegen werden Sie vor dem Formatieren mit der Meldung "ARE YOU SURE" um Bestätigung gebeten. Erst wenn Sie "Y" sagen, wird die Formatierung vorgenommen.

Beispiel:

```
HEADER "SPIELE", IA5

HEADER (B$)
(geht nur bei bereits formatierter Diskette!)

HEADER "SCHROTT", IFF, D1, U9
```

HELP — Befehl — 7

Schreibweise:

HELP

Zweck:

Zeigt nach einem Fehler im BASIC-Programm die fehlerhafte Zeile auf dem Bildschirm an.

Kommentar:

Auf dem 40-Zeichen-Bildschirm wird die fehlerhafte Programmzeile invers dargestellt; auf dem 80-Zeichen-Bildschirm wird sie unterstrichen.

HEX$ Funktion 7

Schreibweise:

HEX$(n)

Zweck:

Wandelt eine Dezimalzahl ("n") in einen Hexadezimalstring um.

Kommentar:

Die Funktion liefert einen vier Zeichen langen String als Wert, der die betreffende Dezimalzahl darstellt.

Die Dezimalzahl "n" darf im Bereich zwischen 0 und 65535 liegen.

Beispiel:

```
PRINT HEX$(7411)
(Ergebnis: 1cf3)
```

IF...THEN...ELSE

Anweisung 2(!),7

Schreibweise:

```
IF bedingung THEN then-zweig
[:ELSE else-zweig]
```

Zweck:

Bewirkt eine bedingte Programmverzweigung.

Kommentar:

Wenn die "bedingung" wahr ist, dann wird/werden die Anweisung(en) im "then-zweig" durchgeführt. Ist die Bedingung nicht erfüllt und eine ELSE-Angabe vorhanden, dann wird/werden die Anweisung(en) im "then-zweig" durchgeführt. Anschließend wird mit der nächsten Anweisung hinter der IF-Anweisung fortgefahren.

Die "bedingung" kann eine einfache Variable oder ein zusammengesetzter logischer Ausdruck sein. Sie wird wahr, wenn sie einen Wert ungleich 0 hat. Die "bedingung" ist falsch, wenn ihr Wert gleich 0 ist.

Sowohl im "then-zweig" als auch im "else-zweig" können weitere IF-Anweisungen stehen. Ist die Anzahl von THEN- und ELSE-Angaben unterschiedlich, so bezieht sich ELSE immer auf das letzte davorstehende THEN.

64er-Modus: Die gesamte IF-Anweisung (einschließlich "then-zweig" und "else-zweig" muß auf eine einzige BASIC-Zeile passen.

128er-Modus: Sowohl im "if-zweig" als auch im "else-zweig" kann mithilfe der Anweisungen BEGIN...BEND (s. dort) ein Anweisungsblock gebildet werden, wodurch sich die IF-Anweisung auch über mehrere Zeilen erstrecken kann.

Beispiele:

```
10 A = 3
20 IF A THEN DIRECTORY
```

INPUT — Anweisung — 2,7

Schreibweise:

```
INPUT ["meldung";] varlist
```

Zweck:

Liest Werte von der Tastatur in Variable und gibt ggf. eine Meldung aus.

Kommentar:

In "varlist" können beliebig viele durch Komma getrennte Variablen stehen. Trifft der Interpeter auf die INPUT-Anweisung, so wird das Programm angehalten und der Benutzer durch Anzeigen eines Fragezeichens zur Eingabe von Werten aufgefordert. Die eingegebenen Werte müssen mit RETURN abgeschlossen werden. Ist in "varlist" mehr als eine Variable angegeben, so müssen ebensoviele Werte, durch Komma getrennt und abgeschlossen von RETURN, vom Benutzer eingegeben werden.

Der erste eingegebene Wert wird der ersten Variablen in "varlist" zugewiesen, der zweite Wert kommt in die zweite Variable usw. Falls der Typ des eingegebenen Werts nicht mit dem Variablentyp übereinstimmt (wenn etwa ein Zeichen als Wert für eine numerische Variable eingegeben wird), erfolgt die Meldung REDO FROM START und Sie müssen den Wert erneut eingeben.

Ist in "varlist" mehr als eine Variable angegeben und der Benutzer gibt nicht genügend Werte für alle diese Variablen an, so wird er durch "??" zur Eingabe weiterer Daten aufgefordert. Wird jetzt nur die RETURN-Taste gedrückt - ohne Eingabe eines Wertes - so wird der letzte eingegebene Wert übernommen.

Ist der Parameter "meldung" spezifiziert - als Zeichenkonstante, umschlossen von Anführungszeichen - dann wird erst diese Meldung auf dem Bildschirm ausgegeben, ehe der Benutzer seine Daten liefern kann. Sie können so einen erklärenden Text ausgeben, der dem Benutzer besser als das nichtssagende Fragezeichen klarmacht, was von ihm erwartet wird. Ist die Meldung vorhanden, so verzichtet BASIC auf

die Ausgabe des Fragezeichens. Hinter dem Parameter "meldung" muß ein Strichpunkt stehen.

Die INPUT-Anweisung ist im Direktmodus nicht erlaubt.

Beispiele:

```
10 INPUT "GEBEN SIE MIR ZWEI ZAHLEN";A,B
20 PRINT "DATENFUETTERUNG ERST AB 23 UHR!"

INPUT A$
```

INPUT# Anweisung 2,7

Schreibweise:

```
INPUT # logische_dateinummer, varlist
```

Zweck:

Dateneingabe von einem beliebigen Gerät oder einer Datei.

Kommentar:

Die Anweisung funktioniert wie die INPUT-Anweisung (s. dort); allerdings ist keine Meldung zugelassen. Das Gerät oder die Datei muß zuvor mit OPEN oder DOPEN geöffnet worden sein und eine "logische_dateinummer" erhalten haben.

INSTR — Funktion — 2,7

Schreibweise:

```
INSTR (str$1, str$2 [,start])
```

Zweck:

Liefert die Position, ab der der String "str$1" im String "str$2" enthalten ist.

Kommentar:

Läßt sich "str$1" (der Suchstring) in "str$2" (der Durchsuchstring) nicht finden, so liefert die Funktion den Wert 0. Es kann auch über den Parameter "start" eine Startposition angegeben werden, ab der im Durchsuchstring mit der Suche begonnen wird.

Das letzte Beispiel ist ein kleines Programm, das ausgibt, wie oft der Buchstabe "E" in einem String S$ vorkommt.

Beispiel:

```
IF INSTR(A$,B$) > 0 THEN PRINT "ICH HAB'S!"

PRINT INSTR("INSTR","EINSTREUEN")

10 POS = 1
20 DO WHILE POS <> 0
30 :  CNT = CNT + 1
40 :  POS = INSTR("E",S$,POS)
40 LOOP
50 PRINT POS
```

INT — Funktion — 2,7

Schreibweise:

```
INT(ausdr)
```

Zweck:

Ergibt die größte ganze Zahl, die kleiner oder gleich "ausdr" ist.

Kommentar:

"ausdr" kann eine Zahl oder ein beliebiger arithmetischer Ausdruck sein. Ist "ausdr" positiv, so ergibt sich der Wert von INT einfach durch Abschneiden der Nachkommastellen. Anders bei negativen Zahlen: hier erhält man den INT-Wert, wenn man zuerst die Nachkommastellen abschneidet und dann 1 subtrahiert.

Das letzte Beispiel zeigt, wie man die INT-Funktion einsetzt, um in BASIC auf die nächste ganze Zahl auf- oder abzurunden.

Beispiele:

```
PRINT INT(3.1415)

PRINT INT(-4.222)
(Hat den Wert -5)

PRINT INT(-13.75)
(Hat den Wert -14)

X = INT(X + 0.5)
```

JOY — Funktion — 7

Schreibweise:

```
JOY(n)
```

Zweck:

Gibt die Position eines Joystick und den Zustand des Feuerknopfes wieder.

Kommentar:

Falls "n" den Wert 1 hat, wird der Joystick-Port 1, bei "n"=2 der Port 2 abgefragt.

Die Funktion gibt Werte zwischen 0 und 8 oder zwischen 128 und 136 zurück. Die Werte zwischen 0 und 8 entsprechen den 7 möglichen 8 möglichen Positionen eines Joysticks:

0	Mitte
1	Norden
2	Nordwest
3	Westen
4	Südwest
5	Süden
6	Südost
7	Osten
8	Nordost

Falls zusätzlich noch der Feuerknopf gedrückt wurde, erhöht sich das Ergebnis von JOY um den Wert 128.

Beispiel:

```
PRINT JOY(2)
```
(Ergebnis 135 bedeutet: Joystick 2 zeigt nach links; Feuerknopf ist gedrückt; denn 135 = 128 + 7)

KEY Anweisung 7

Schreibweise:

```
KEY [taste, str$]
```

Zweck:

Zeigt die Belegung der Funktionstasten an oder ändert sie.

Kommentar:

Wird KEY ohne Parameter angegeben, so bekommen Sie die momentane Belegung der Funktionstasten auf dem Bildschirm angezeigt.

Hat der Parameter "taste" einen Wert zwischen 1 und 8, so wird die Funktionstaste mit der entsprechenden Nummer mit dem Ausdruck "str$" belegt. Wenn Sie danach die betreffende Funktionstaste drücken, so wird "str$" ausgegeben, genauso, als hätten Sie dies von der Tastatur eingegeben. Wie das erste Beispiel zeigt, ist es auch möglich, das abschließende RETURN (CHR$(13)) für Befehlseingabe mit aufzunehmen.

Das zweite Beispiel belegt die Funktionstasten genauso wie im 64er-Modus.

Insgesamt stehen für die 8 Funktionstasten nur 255 Zeichen zur Verfügung.

Beispiele:

```
KEY 7, "GRAPHICO"+CHR$(13)+"LIST"+CHR$(13)

FOR I = 1 TO 8 : KEY I, CHR$(I + 132) : NEXT
```

LEFT$ — Funktion — 2,7

Schreibweise:

```
LEFT$ (str$, ausdr)
```

Zweck:

Liefert die ersten Zeichen eines Strings.

Kommentar:

Wieviele Zeichen zurückgegeben werden hängt von "ausdr" ab. Dieser kann eine Zahl oder ein beliebiger arithmetischer Ausdruck sein. Sein Wert muß jedoch zwischen 0 und 255 liegen. Hat "ausdr" den Wert 0, so wird der leere String "" zurückgegeben.

Beispiel:

```
PRINT LEFT$("GEHEIM",2) + "MEIN"
```

LEN — Funktion — 2,7

Schreibweise:

```
LEN (str$)
```

Zweck:

Liefert die Länge des Strings "str$".

Kommentar:

Die Länge eines Strings liegt zwischen 0 und 255 und ist die Anzahl an Zeichen, die im String enthalten sind. Dazu zählen auch nichtabdruckbare Zeichen (Steuerzeichen) und Grafikzeichen.

LET — Anweisung — 2,7

Schreibweise:

```
LET var = wert
```

Zweck:

Weist der Variablen "var" den Wert "wert" zu.

Kommentar:

Selten gebrauchte Schreibweise für die Zuweisung, die in BASIC kürzer als "var = wert" geschrieben werden kann.

Beispiel:

```
LET A$ = "ANFANG"
```

LIST **Befehl** 2,7

Schreibweise:

```
LIST [anfzeile] [ - [endzeile]]
```

Zweck:

Listet ein im Arbeitsspeicher befindliches BASIC-Programm auf dem Bildschirm auf.

Kommentar:

LIST ohne Parameter gibt das gesamte Programm aus. Ist nur der Parameter "anfzeile" gegeben, so wird lediglich diese eine Zeile gelistet. Kommt hinter "anfzeile" ein Bindestrich, so wird von der Zeile bis zum Programmende gelistet. Folgt hinter dem Bindestrich der Parameter "endzeile", so wird der Bereich zwischen "anfzeile" und "endzeile" gelistet. Fehlt "anfzeile", so wird vom Programmanfang bis "endzeile" ausgegeben.

Das Rollen des Bildschirms kann im 128er-Modus mit der Taste NO SCROLL verhindert werden. Drückt man diese Taste, so wird die Bildschirmanzeige angehalten und erst bei erneutem Drücken wieder fortgesetzt.

Mit der C=-Taste kann die Bildschirmanzeige verlangsamt werden.

Beispiele:

```
LIST

LIST 15

LIST 34 -

LIST 220 - 810

LIST - 915
```

LOAD

Befehl 2,7

Schreibweise:

```
LOAD str$ [,geräteadresse] [,verschieb]
```

Zweck:

Lädt eine Programmdatei von Band oder einem anderen Gerät.

Kommentar:

Wenn Sie keine Parameter angeben, so versucht LOAD von der Datasette zu laden, nachdem zuvor die Meldung "PRESS PLAY ON TAPE" ausgegeben wurde. Dann sucht LOAD nach dem ersten Programm auf der Kassette. Sobald es eines gefunden hat, meldet es sich mit "FOUND name", wobei "name" die gefundene Programmdatei bezeichnet. Sie können jetzt die C=-Taste betätigen, wodurch das Programm geladen wird, oder auf die Leertaste drücken, was LOAD weitersuchen läßt.

Meist ist jedoch der Parameter "str$" angegeben - als Stringkonstante oder Variable. Dann sucht LOAD solange auf der Datasette, bis eine Datei gleichen Namens gefunden wird. Der Parameter "str$" darf die Jokerzeichen "*" und "?" enthalten.

Sie können auch eine "geräteadresse" angeben, um von einem anderen Gerät zu laden. Der Wert 8 bewirkt, daß von Diskette geladen wird. Im 128er-Modus verwenden Sie dazu aber besser DLOAD (s. dort).

Hat der Parameter "verschieb" den Wert 1, so bedeutet dies, daß ein Maschinenprogramm geladen werden soll. Dies wird dann nicht notwendigerweise an die übliche Startadresse für BASIC-Programme geladen; vielmehr wird die Ladeadresse aus der Programmdatei übernommen.

Mit LOAD können auch BASIC-Progamme nachgeladen werden. Dazu benutzt man im 128er-Modus aber besser die APPEND-Anweisung (s. dort). Wird LOAD für ein BASIC-Programm benutzt und "verschieb" hat den Wert 1, dann wird das BASIC-Programm nicht an den Anfang des Textspeichers geladen (und überschreibt somit ein bereits vorhandenes Programm), sondern ab der Adresse, von der aus es mit SAVE abgespeichert wurde.

Beispiele:

```
LOAD "MEINPROG"

LOAD "M*"
```

LOCATE **Anweisung** 7

Schreibweise:

LOCATE x, y

Zweck:

Bewegt den Pixel-Cursor an die angegebene Stelle des Grafikbildschirms.

Kommentar:

Der Pixel-Cursor ist - anders als der normale Textcursor - unsichtbar, er kann aber bewegt werden. Die aktuelle Position des Pixel-Cursor ist Ausgangspunkt für alle Zeichenbefehle, die mit dem Grafikbildschirm zu tun haben (DRAW, BOX, CIRCLE).

Für den Parameter "x" sind Werte zwischen 0 und 319, für "y" zwischen 0 und 199 erlaubt. Im Multicolor-Modus darf "x" nur bis 159 gehen. Ist die Skalierung eingeschaltet (s. SCALE-Anweisung), dann dürfen beide Parameter Werte zwichen 0 und 1023 annehmen.

Die aktuelle Position des Pixel-Cursors ändert sich nicht nur durch LOCATE, sondern durch jede Zeichenanweisung (DRAW, BOX, CIRCLE). Nach dem Zeichnen einer Figur befindet sich der Pixel-Cursor auf dem letzten Bildpunkt, der zur Vollendung der Zeichnung benötigt wurde - und bleibt auch dort, wenn Sie ihn nicht mit LOCATE verschieben.

Wenn Sie die aktuelle Position des Pixel-Cursors wissen wollen, so müssen Sie die RDOT-Funktion benutzen (s. dort).

Beispiel:

```
10 GRAPHIC 1
20 LOCATE 150, 100 : REM PIXEL-CURSOR IN DIE MITTE
```

LOG — Funktion — 2,7

Schreibweise:

LOG(ausdr)

Zweck:

Liefert den natürlichen Logarithmus.

Kommentar:

Der natürliche Logarithmus ist der Logarithmus zur Basis e (s. EXP-Funktion). Wenn Sie den Logarithmus zu einer anderen Basis benötigen, so müssen Sie das Ergebnis der LOG-Funktion mit dem natürlichen Logarithmus der Basis dividieren. Das Beispiel zeigt Ihnen, wie Sie den Logarithmus dualis (zur Basis 2) einer beliebigen Zahl erhalten können.

Beispiel:

```
10 INPUT "ZAHL: "; N
20 PRINT "ZWEIER-LOGARITHMUS :";
30 PRINT LOG(N)/LOG(2)
```

MID$ — Funktion — 2,7

Schreibweise:

```
MID$ (str$, ausdr1 [, ausdr2])
```

Zweck:

Liefert einen Ausschnitt aus dem String "str$" mit einer Länge von "ausdr1", beginnend ab der mit "ausdr2" bezifferten Zeichenposition von str$.

Kommentar:

Sowohl "ausdr1" als auch "ausdr2" können Werte zwischen 0 und 255 annehmen. Falls einer der Werte oder beide größer als 255 sind, oder falls "ausdr2" größer als die Länge des Strings "str$" ist, so wird der leere String zurückgegeben. Falls "ausdr2" fehlt, so wird der Teilstring bis zum Ende von "str$" zurückgegeben.

Beispiel:

```
150 S$ = MID$("ABCDEFGHIJKLMNOP", X+3, 5)
```

MONITOR **Befehl** 7

Schreibweise:

MONITOR

Zweck:

Schaltet in den Maschinensprache-Monitor um.

Kommentar:

Die Funktionsweise des Monitors ist im Text ausführlich beschrieben. Hier noch einmal eine Kurzübersicht über die Kommandos und ihre Wirkung.

A	Assemblieren einer Zeile mit 8500-Code
C	Speicherbereiche vergleichen (Compare)
D	Disassemblieren
F	Füllen des Arbeitsspeichers mit einem Bytewert
G	Progamm ab angegebener Adresse ausführen (Go)
H	Sucht im Arbeitsspeicher nach Bytes (Hunt)
L	Laden einer Datei von Band oder Diskette
M	Speicherinhalt hexadezimal anzeigen (Memory)
R	Registerinhalte anzeigen
S	Sichern auf Band oder Platte
T	Bytes im Speicher verschieben (Transfer)
V	Speicherbereich mit Band oder Diskette vergleichen (Verify)
X	Monitor verlassen (Exit)
.	(Punkt) gleiche Wirkung wie A-Befehl
>	(Größer-Zeichen) Speicher verändern
;	(Strichpunkt) Prozessor-Register verändern
@	(Klammeraffe) Diskettenstatus anzeigen

MOVSPR — Anweisung — 7

Schreibweise:

```
MOVSPR spritenr, x,y
oder
MOVSPR spritenr, +/-x, +/-y
oder
MOVSPR spritenr, x#y
```

Zweck:

Bewegt ein Sprite zu einer wählbaren Position oder setzt es mit wählbarem Winkel und Geschwindigkeit in Bewegung.

Kommentar:

In der ersten Form wird das Sprite mit der Nummer "spritenr" (0 bis 7) auf die angegebene absolute Position bewegt.

Die zweite Form dient zur relativen Positionierung. Das Sprite wird bezüglich der letzten eingenommenen Position weiterbewegt.

Die dritte Form setzt ein Sprite in Bewegung oder stoppt es. Sind die Werte für "x" und "y" durch ein #-Zeichen getrennt, so gibt "x" einen Bewegungswinkel und "y" die Geschwindigkeit (0 bis 15) für das ausgewählte Sprite an. Die Winkelangaben orientieren sich an der Windrose: ein Wert von 0 bewegt das Sprite nach oben (Norden), 90 führt es nach Osten (rechts) und 210 nach Westen (links). Je größer der Wert für "y" in dieser Form ist, desto schneller flitzt das Sprite über den Bildschirm.

Ein einmal mit der dritten Form in Bewegung gesetztes Sprite bleibt auch in Bewegung, solange es eingeschaltet ist. Wenn es an den Rand stößt, so erscheint es am gegenüberliegenden Bildschirmrand wieder, ganz so, als wäre der Bildschirm wie ein Zylinder in sich gekrümmt. Wird es (mit dem SPRITE-Befehl) abgeschaltet, so merkt es sich seine letzte Position, Winkel und Geschwindigkeit. Wenn Sie es später wieder einschalten, dann läuft es also unbeirrt ab der Stelle weiter, an der es abgeschaltet wurde.

Beispiele:

```
MOVSPR 1, 285, 178
```
(Sprite 2 in die rechte untere Ecke)

```
MOVSPR 2, +50, -60
```
(Sprite 3 wird weiterbewegt: 50 Pixels von dér letzten Position aus nach rechts und 60 Pixels von der letzten Position aus nach oben)

```
MOVSPR 3, 135#8
```
(Sprite 4 startet mit mittlerer Geschwindigkeit in Richtung Südosten)

NEW — Befehl — 2,7

Schreibweise:

```
NEW
```

Zweck:

Löscht das im Arbeitsspeicher befindliche BASIC-Programm.

Kommentar:

Das Programm wird nicht wirklich gelöscht in dem Sinne, daß es mit anderen Informationen überschrieben wird. Stattdessen werden nur einige interen Zeiger zurückgesetzt: der Computer setzt den Zeiger auf das Programmende auf den Anfang des Textbereichs. Für ihn ist das Programm somit weg. Außerdem setzt er noch die im Programm verwendeten Variablen zurück. Aber solange kein neues Programm eingegeben wird, ist der vollständige Text des alten Programms noch da.

Man kann den NEW-Befehl auch in ein BASIC-Programm einbauen - aber warum sollte man das tun wollen? (Andererseits ist's auch eine Art Kopierschutz!)

Beispiel:

```
NEW
```

NEXT — Anweisung — 2,7

Schreibweise:

```
NEXT [varlist]
```

Zweck:

Schließt eine FOR...NEXT-Schleife ab (s. FOR-Anweisung).

Kommentar:

NEXT kann ohne Angaben erfolgen; es schließt dann die letzte FOR-Schleife ab, d.h. die Schleifenvariable wird um den angegebenen Wert erhöht und die Schleife erneut durchlaufen oder ggf. verlassen. Wer will, kann hinter NEXT auch (im Parameter "varlist") die Schleifenvariable hinschreiben; diese ist auf jeden Fall übersichtlich, wenn auch nicht notwendig.

Es ist auch möglich, mit einer NEXT-Anweisung mehrere Schleifen zugleich abzuschließen; dann müssen im Parameter "varlist" alle Schleifenvariablen der abzuschließenden Schleifen, durch Komma getrennt, angegeben werden (siehe Beispiel).

Beispiel:

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO I
30 FOR K = 1 TO J
40 REM GROSSER LEERLAUF
50 NEXT I,J,K
```

NOT — Funktion — 2,7

Schreibweise:

NOT (bed)

Zweck:

Kehrt den Wahrheitswert eines logischen Ausdrucks um.

Kommentar:

Falls der Wert von "bed" wahr ist (entspricht -1), so liefert NOT den Wert "falsch" (entspricht 0) und umgekehrt.

Beispiel:

```
10 INPUT A
20 IF NOT(A<0) THEN PRINT "ZAHL IST POSITIV"
```

ON — Anweisung — 2,7

Schreibweise:

```
ON ausdr GOTO zeile1 [, zeile2, ...]
oder
ON ausdr GOSUB zeile1 [, zeile2, ...]
```

Zweck:

In Abhängigkeit vom Wert von "ausdr" wird mit GOTO oder GOSUB in verschiedene Programmteile verzweigt.

Kommentar:

"ausdr" muß Werte annehmen, die zwischen 1 und der Gesamtzahl der "zeile"-Parameter liegen. Bei einem Wert von 1 wird in "zeile1" verzweigt, der Wert 2 verzweigt in "zeile2" usw.

Hat "ausdr" den Wert 0 oder ist der Wert größer als die Anzahl der Zeilennummern, so wird mit der Anweisung hinter ON fortgefahren.

Bei einem negativen Wert von "ausdr" erhalten Sie einen ILLEGAL QUANTITY ERROR.

Beispiele:

```
10 INPUT "WOHIN SOLL ICH GEHEN (1 BIS 5)
20 ON I GOSUB 1000,2000,3000,4000,5000
30 PRINT "DAS KANN ICH NICHT!" : GOTO 10
.
.
.
```

OPEN — Anweisung — 2,7

Schreibweise:

```
OPEN logische_dateinr, gerätenummer,
[sekundäradresse] [,"dateiname, typ,
modus"] / [, kommando]
```

Zweck:

Öffnet einen Datenkanal zwischen dem Rechner und einem beliebigen Peripheriegerät.

Kommentar:

Mit der OPEN-Anweisung kann der Computer mit beliebigen Peripheriegeräten Daten austauschen (lesen und/oder schreiben - je nach Art des Geräts), also mit der Datasette oder Floppy-Stationen, einem Drucker, aber auch mit der Tastatur und dem Bildschirm!

Der erste Parameter ("logische_dateinr") ist eine Zahl, die der Programmierer wählt und die dem geöffneten Kanal zugeordnet wird; im weiteren Verlauf des Programmes wird der Kanal dann unter dieser Nummer angesprochen (z.B. in PRINT#-Anweisungen). Erlaubt sind Werte zwischen 0 und 255; da aber Zahlen über 128 spezielle Funktionen bei den Peripheriegeräten auslösen können, sollten Sie mit solchen Werten vorsichtig sein.

Der zweite Parameter muß angegeben werden und wählt das Peripheriegerät aus, zu dem der Kanal geöffnet werden soll. Mögliche Werte für "gerätenummer" sind:

0	die Tastatur
1	die Datasette
3	der Bildschirm
4-7	Drucker
8-11	Diskettenlaufwerk(e)

Die Angaben zur "gerätenummer" dürfen auf dem C128 mit einer führenden Null (08 statt 8) versehen sein.

Hinter der "gerätenummer" kann eine weitere Zahl folgen, die "sekundäradresse"; bei der Datasette bedeutet hier ein Wert von 0 "lesender Zugriff", 1 bedeutet "Schreibzugriff" und 2 bewirkt, daß eine Endemarkierung auf das Band ge-

schrieben wird. Bei der Floppy ist dies die Kanalnummer; Kanalnummer 15 ist der Befehlskanal. Bei Druckern kann man mit der Sekundäradresse verschiedene Druckmodi einstellen. Einzelheiten dazu erfahren Sie in den verschiedenen Geräte-Handbüchern.

Schließlich können Sie in das Kommando noch einen String aufnehmen. Bei Kassetten- oder Diskettendateien ist dies der Dateiname. Mögliche Werte für den Parameter "typ" sind (erlaubte Abkürzungen in Klammern):

PRG für Programmdateien (Abk. P)
SEQ für sequentielle Dateien (Abk. S)
REL für Relativdateien (Direktzugriff möglich) (Abk. L)
USR für Binärdaten (z.B. Maschinenprogramme) (Abk. U)

Mögliche Werte für "modus" folgen; der APPEND-Modus ist allerdings nur bei Programmdateien, Relativdateien und USR-Dateien erlaubt:

READ lesender Zugriff (Abk. R)
WRITE Lese-/Schreibzugriff (Abk. W)
APPEND ans Ende der Datei schreiben (Abk. A)

Es ist auch möglich, anstelle des Dateinamens einen Kommandostring an die Datasette oder Floppy zu schicken.

Beispiele:

```
OPEN 3,3
```
(Öffnet den Bildschirm und weist dem Kanal die Nummer 3 zu)

```
OPEN 1,0
```
(Öffnet die Tastatur als Kanal mit der Nummer 1)

```
OPEN 1,1,0,"BANDDAT"
```
(Öffnet die Datasette, sucht darauf nach der Datei "BANDDAT" und weist sie dem Kanal 1 mit lesendem Zugriff zu)

```
OPEN 1,4
```
(Drucker wird als Kanal 1 eröffnet)

```
OPEN 15,8,15
```
(Öffnet den Kommandokanal der Floppy)

```
OPEN 8,8,12,"KUNDENDATEN,SEQ,WRITE"
```
(Öffnet eine sequentielle Diskettendatei namens "KUNDENDATEN" zum Schreiben)

OR — Funktion — 2.7

Schreibweise:

```
ausdr1 OR ausdr2
```

Zweck:

Verknüpft wahrheitswertige Ausdrücke.

Kommentar:

Falls nur einer (oder auch beide) der mit OR verknüpften Ausdrücke wahr ist, dann ist der gesamte OR-Ausdruck wahr (gleich -1), ansonsten falsch (gleich 0).

Falls "ausdr1" und "ausdr2" nicht wahrheitswertig, sondern numerische Ausdrücke sind, so liefert OR stattdessen das Ergebnis einer bitweisen ODER-Verknüpfung von "ausdr1" und "ausdr2" (vgl. Beispiel 2).

"ausdr1" und "ausdr2" können selbstverständlich auch mit OR verknüpfte zusammengesetzte Ausdrücke sein.

Beispiele:

```
10 INPUT "BITTE EINE BUCHSTABENTASTE DRUECKEN: ";A$
20 IF A$ > "J" OR A$ < "E" THEN GOTO 10
30 PRINT "ZWISCHEN E UND J LIEGT: ";A$
```

```
PRINT 1 OR 2
```

(liefert als Ergebnis die Zahl 3, da bei deren Binärdarstellung sowohl das erste als auch das zweite Bit gesetzt ist)

PAINT — Grafik-Anweisung — 7

Schreibweise:

```
PAINT farbquelle [, x, y, modus]
```

Zweck:

Malt eine Fläche aus.

Kommentar:

Mit der Anweisung ist es möglich, jeden beliebigen geschlossenen Polygonzug (natürlich auch Kreise und Ellipsen) mit einer wählbaren Farbe auszufüllen.

Die Farbe, mit der ausgemalt wird, bestimmt der Parameter "farbquelle", bei dem Werte zwischen 0 und 3 zugelassen sind. Allerdings muß die Füllfarbe mit der Farbe des Polygonzugs (der Umrandung) übereinstimmen; im Multicolor-Modus darf es auch eine andere Vordergrundfarbe sein.

Die Parameter "x" und "y" bestimmen die Startkoordinaten für das Ausmalen; beachten Sie, daß dieser Punkt nicht innerhalb des Polygonzugs zu liegen braucht. Es wird dann einfach die umgebende Fläche ausgemalt. Voreingestellt ist für "x" und "y" die aktuelle Position des Pixel-Cursors (s. LOCATE).

Den Ausmalvorgang können Sie mit dem Parameter "modus" beeinflussen. Ist "modus" gleich 0, so wird die auszumalende Fläche von der (im Parameter "farbquelle") gewählten Farbe begrenzt. Ein Wert von 1 besagt, daß jede andere als die Hintergrundfarbe als Begrenzungsfarbe angesehen werden soll.

Nach Beendigung des Ausmalvorgangs befindet sich der Pixel-Cursor an der Startposition "x,y".

Falls der Startpunkt "x,y" bereits ausgemalt ist, so passiert nichts; Sie müssen sich dann einen anderen Startpunkt innerhalb der auszumalenden Fläche wählen.

Beispiel:

```
10 GRAPHIC 1
20 CIRCLE ,160,100,50  : REM KREIS IN VORDERGRUNDFARBE
30 PAINT ,160,100      : REM AUSFÜLLEN
```

PEEK — Funktion — 2,7

Schreibweise:

```
PEEK(adr)
```

Zweck:

Liefert den Bytewert, der an einer bestimmten Adresse des Arbeitsspeichers gespeichert ist.

Kommentar:

"adr" ist ein numerischer Ausdruck, der die Adresse der Speicherstelle angibt, die untersucht werden soll. Sein Wert muß zwischen 0 und 65535 liegen. Der von PEEK zurückgegebene Wert liegt im Bereich zwischen 0 und 255.

Im 128er-Modus ist es möglich, mit dem BANK-Befehl (s. dort) die Speicherbank auszuwählen, auf die sich die Adressangabe bezieht.

Beispiel:

```
PRINT PEEK(800)
```

PEN — Funktion — 7

Schreibweise:

```
PEN (n)
```

Zweck:

Liefert die x- und y-Koordinate und/oder den Zustand des Lichtstifts. Mögliche Werte für den Parameter "n" und deren Bedeutung:

0	x-Koordinate auf Grafikbildschirm (60 bis 320)
1	y-Koordinate auf Grafikbildschirm (50 bis 200)
2	Spaltennummer auf dem 80-Zeichen-Bildschirm
3	Zeilennummer auf dem 80-Zeichenbildschirm
4	liefert 1, falls der Lichtstift seit der letzten Abfrage aktiviert wurde und 0 sonst

Kommentar:

Die von PEN zurückgegebenen Grafikkoordinate beziehen sich auf den gesamten Bildschirm (einschließlich Hintergrund); deswegen beginnen die Rückgabe-Werte auch erst bei 60 bzw. 50, also dem Bereich des beschreibbaren Bildschirms.

Liegt die Position des Lichtstifts außerhalb des gültigen Bildschirmbereichs, so wird als Wert eine Null zurückgegeben.

Um bei der Arbeit mit dem Lichtstift sicherzugehen, sollten Sie einen weißen Bildschirmhintergrund wählen.

Beispiele:

```
10 DO UNTIL PEN(4) : LOOP
20 PRINT PEN(2), PEN(3)
```

(Druckt die Lichtstift-Koordinaten, falls dieser aktiviert wurde)

PLAY — Sound-Anweisung — 7

Schreibweise:

```
PLAY str$
```

Zweck:

Spielt eine Melodie.

Kommentar:

Die Melodie wird von der Zeichenkette "str$" bestimmt. In dieser Zeichenkette sind Steuerzeichen anzugeben, die in vielfältiger Weise erlauben, Melodie, Tonhöhe, Klangfarbe, Tempo und Anzahl der Stimmen zu variieren. Die Steuerzeichen und ihre Bedeutung im Einzelnen:

On: Wählt die Oktave aus, in der ein oder mehrere nachfolgende Töne erklingen sollen. Zulässig sind Werte von 0 bis 6.

Tn: Wählt die Tonhüllkurve für die nachfolgenden Töne aus. Zulässige Werte für "n" gehen von 0 bis 9. Beim Einschalten des Gerätes sind bereits 10 Hüllkurven vordefiniert (s. ENVELOPE-Anweisung).

Un: Setzt die Lautstärke ("n" von 0 bis 9).

Vn: Bestimmt die Stimme, in der der Ton erklingt (1 bis 3)

Xn: Filter ein ("n" = 1) oder aus ("n" = 0); s. FILTER-Befehl.

A,B,C,D,E,F,G: zu spielende Noten .

#: Nachfolgende Note ist um einen Halbton erhöht.

$: Nachfolgende Note ist um einen Halbton erniedrigt.

W: Ganze Note folgt

H: Halbe Note folgt

Q: Viertel folgt

I: Achtel folgt

S: Sechzehntel folgt

.: Punktierte Note (steht vor (!) der Note)

R: Pause folgt

M: Wartet, bis alle momentan spielenden Stimmen fertig sind; Angabe über Dauer folgt.

Beispiele:

```
PLAY "O4CDEFGABO5C"
```
(Spielt eine Tonleiter)

```
10 FOR I = 0 TO 9
20 PRINT I, : PLAY "T"+CHR$(48+I)+"O4CDEFGABO5C"
30 SLEEP 1
40 NEXT
```
(Spielt die Tonleiter mit allen zehn Instrumenten)

```
10 I$ = "T7U9O5Q#.FIEH.DO4A#FWBHEO5Q.EIDH.#CO4BAO5H#F"
20 PLAY I$
```
(Eine alte Melodie)

POINTER — Funktion — 7

Schreibweise:

```
POINTER (var)
```

Zweck:

Liefert die Adresse des ersten Bytes in Bank 1 für die Variable "var".

Kommentar:

Mit der Funktion können Sie feststellen, an welcher Stelle des Variablenspeichers (Bank 1) der Interpreter eine bestimmte Programmvariable abgelegt hat. Die Variable muß jedoch bereits mit einem Wert versehen und - bei Feldvariablen - dimensioniert sein.

Die Funktion ist nützlich, um die Startadresse von BASIC-Variablen an Maschinenprogramme übergeben zu könen.

Beispiel:

```
10 A% = 4
20 BANK 1 : PRINT PEEK(1 + POINTER(A%))
```

POKE — Anweisung — 2,7

Schreibweise:

```
POKE adr, wert
```

Zweck:

Ändert den Inhalt von Speicherstellen im RAM.

Kommentar:

Der über "adr" (0 bis 65535) bestimmten Adresse in der ausgewählten Bank (s. BANK-Anweisung) wird der neue Wert "wert" (0 bis 255) zugewiesen. (vgl. auch PEEK)

Beispiel:

```
POKE 28000,8
```

POS — Funktion — 7

Schreibweise:

```
POS(n)
```

Zweck:

Liefert die Spaltenposition des Textcursors.

Kommentar:

"n" ist ein Dummy-Paramter, d.h. er muß zwar angegeben werden, aber sein Wert hat keinerlei Auswirkungen auf die Funktionsweise von POS.

Auf dem 80-Zeichen-Bildschirm liefert POS eine Zahl zwischen 0 und 79.Auf dem 40-Zeichen-Bildschirm beziehen sich Werte über 39 auf die zweite Zeile.

POT

Funktion 7

Schreibweise:

```
POT(n)
```

Zweck:

Liefert den Wert eines von 4 angeschlossenen Game Paddles (Drehregler).

Kommentar:

Für "n" sind Werte zwischen 1 und 4 erlaubt, die sich auf das ausgewählte Paddle beziehen. POT liefert je nach Stellung des Drehknopfs Werte zwischen 0 und 255. Werte über 255 bedeuten, daß zudem noch der Feuerknopf gedrückt wurde.

PRINT — Anweisung — 2,7

Schreibweise:

```
PRINT printlist
```

Zweck:

Ausgabe von Daten auf den Bildschirm.

Kommentar:

Der Parameter "printlist" kann eine Variable, ein numerischer Ausdruck oder eine in Anführungszeichen eingeschlossene Stringkonstante sein. Es können in "printlist" aber auch mehrere dieser Elemente angegeben sein, die dann durch Komma oder Strichpunkt voneinander getrennt sein müssen.

Ein Komma als Trenner bedeutet, daß vor der Ausgabe des nachfolgenden Elements erst auf die nächste Tabulatorposition vorgeschoben wird.

Verwenden Sie einen Strichpunkt als Trenner, dann unterbleibt diese Tabulation. Zwei durch Strichpunkt getrennte Datenelemente in der "printlist" werden also unmittelbar hintereinander auf dem Bildschirm ausgegeben.

Beispiele:

```
PRINT A
PRINT "HALLO"
PRINT 2;3;4+7,"STOP"
```

PRINT#

Anweisung 2,7

Schreibweise:

```
PRINT# logische_dateinummer, printlist
```

Zweck:

Datenausgabe auf ein beliebiges Gerät.

Kommentar:

Die Anweisung funktioniert ähnlich wie PRINT (s.dort), allerdings kann über den Parameter "logische_dateinummer" ein beliebiges Ausgabegerät angewählt werden. Dies muß zuvor jedoch mit OPEN oder DOPEN (s. dort) geöffnet worden sein.

Die Trennzeichen Komma und Strichpunkt in der "printlist" wirken sich genauso aus wie in der PRINT-Anweisung.

PRINT USING Anweisung 7

Schreibweise:

```
PRINT [#logische_dateinummer] USING
formatlist; printlist
```

Zweck:

Formatierte Ausgabe auf ein beliebiges Gerät.

Kommentar:

Die Anweisung ähnelt der PRINT#-Anweisung (s. dort); allerdings kann der Programmierer im Parameter "formatlist" detailliert angeben, in welchem Format die Informationen in der "printlist" ausgegeben werden sollen.

Die Formatangaben in "formatlist" und die Ausgabewerte in "printlist" müssen durch einen Strichpunkt getrennt werden.

Der Parameter "formatlist" muß ein String sein, der - neben Textzeichen - auch Format-Steuerzeichen enthält; die Format-Steuerzeichen legen das genaue Ausgabeformat für die Datenelemente der "printlist" fest. Es sind folgende Format-Steuerzeichen erlaubt:

`#` Reserviert Platz für ein einzelnes Zeichen in der Ausgabe. Wenn Sie drei Zeichen ausgeben wollen, dann müssen Sie "###" schreiben und stellen damit Platz für drei Zeichen in der Ausgabe bereit (das sog. "Ausgabefeld"). Hat eine auszugebende Zahl in dem Ausgabefeld nicht Platz, dann wird das Ausgabefeld mit Sternchen gefüllt und das Datenelement nicht ausgegeben. Beachten Sie, daß ein Vorzeichen mit zur Gesamtlänge der Zahl beiträgt. Die Zahl -1000 benötigt also ein 5 Stellen langes Ausgabefeld. Anders ist es bei Strings: hier werden nur soviel Zeichen des Strings ausgegeben, wie in das Datenfeld passen.

`+, -` Kann entweder am Anfang oder am Ende der "formatlist" stehen. Ist eine auszugebende Zahl positiv, dann wird das +-Zeichen gedruckt; ist die Zahl negativ, dann wird das Minuszeichen ausgegeben. Ist das Minuszeichen in der "formatlist" angegeben und die Ausgabezahl ist positiv, dann wird anstelle des Minuszeichens ein Leerzeichen ausgegeben.

`=` (nur für Strings) Der String wird zentriert im Datenfeld ausgegeben.

`>` (nur für Strings) Der String wird rechtsbündig im Datenfeld ausgegeben.

`.` Zeigt an, an welcher Stelle des Ausgabefelds ein Dezimalpunkt oder -komma (s. PUDEF-Anweisung) stehen soll. Der Punkt darf im der

"formatlist" nur einmal vorkommen. Die Anzahl der Stellen vor dem Punkt muß ausreichen, um die Vorkommastellen der auszugebenden Zahl aufzunehmen; ansonsten wird das Ausgabefeld mit Sternchen gefüllt.

, Zeigt an, wo Zahlenbeträge durch ein Komma getrennt werden sollen. Dies entspricht der amerikanischen Schreibweise, mit der bei großen Zahlen der besseren Lesbarkeit wegen Zifferngruppen getrennt weden. In Deutschland ist dazu ein Leerzeichen üblich (also "1 000 000,99" anstatt "1,000,000.99"); das tatsächlich bei der Ausgabe verwendete Trennzeichen kann jedoch mit der PUDEF-Anweisung neu festgelegt werden (s. dort). Im Ausgabefeld muß vor einer Komma-Position mindestens ein #-Zeichen stehen.

$ Gibt ein Währungszeichen vor der ersten gültigen Ziffer der Ausgabezahl mit aus. Voreingestellt ist das Dollarzeichen; Umdefinition mit PUDEF ist möglich.

^^^^ Vier Pfeile am Ende oder Anfang der "formatlist" bewirken, daß eine Zahl in wissenschaftlicher (Exponential-) Schreibweise ausgegeben wird. Dazu muß jedoch vor oder hinter den vier Pfeilen ein Plus- oder Minuszeichen stehen.

Beispiele:

```
10 A$ = "BETRAG: ####.##"
20 PRINT USING A$; 55.45
30 PRINT USING A$; 575.03
40 PRINT USING A$;  3.14
```

PUDEF Anweisung 7

Schreibweise:

```
PUDEF "defliste"
```

Zweck:

Definiert Ausgabezeichen für PRINT USING neu.

Kommentar:

Mit dieser Anweisung können bis zu vier Ausgabezeichen neudefiniert werden. Der Parameter "defliste" muß ein String sein, der bis zu vier Zeichen lang ist. Die Position im String bestimmt, welches Zeichen neu definiert werden soll.

Erste Position: Definiert das Füllzeichen neu. Normalerweise wird ein Ausgabefeld mit Leerzeichen aufgefüllt.

Zweite Position: Definiert das Trennzeichen für Zifferngruppen neu. Voreingestellt ist hier das Komma; in Deutschland ist allerdings das Leerzeichen oder der Punkt üblich.

Dritte Position: Definiert den Dezimalpunkt. In Deutschland ist statt des Punktes ein Komma üblich.

Vierte Position: Definiert das Währungszeichen neu; voreingestellt ist das Dollarzeichen. Da das Währungszeichen von PRINT USING vor die Zahl geschrieben wird, in Deutschland die "DM"-Angabe jedoch an den Geldbetrag angehängt wird, hat eine Umdefinition hier nicht viel Sinn. Außerdem ist nur ein Zeichen zugelassen.

Beispiele:

```
PUDEF "#.,"
```
(Ändert Dezimalpunkt und Trenner auf die in Deutschland gebräuchlichen Werte).

RCLR — Funktion — 7

Schreibweise:

RCLR(farbquelle)

Zweck:

Gibt den Farbwert zurück, der für "farbquelle" gesetzt ist.

Kommentar:

Der Parameter "farbwert" darf folgende Werte haben:

0	Hintergrund
1	Vordergrund
2	Multicolor 1
3	Multicolor 2
4	Rand (VIC-erzeugt)
5	Zeichenfarbe 40-Zeichen-Bildschirm
6	Hintergrund 80-Zeichen-Bildschirm

Die Funktion liefert Werte zwischen 1 und 16, die den Farbnummern entsprechen (s. COLOR-Anweisung).

Beispiele:

```
PRINT RCLR(4)
```

RDOT **Funktion** 7

Schreibweise:

```
RDOT (n)
```

Zweck:

Liefert Informationen über den Pixel-Cursor.

Kommentar:

Falls "n" den Wert 0 hat, liefert RDOT die horizontale Position (x-Koordinate) des Pixel-Cursors, bei "n" = 1 sehen Sie die y-Koordinate und bei "n" = 2 die aktuelle Farbquelle.

Beispiele:

```
PRINT RDOT(1)
```

READ

Anweisung 2,7

Schreibweise:

```
READ varlist
```

Zweck:

Liest Daten aus DATA-Anweisungen (s. dort) in die Variablen, die im Parameter "varlist" angegeben sind.

Kommentar:

Die einzelnen Variablen in der "varlist" müssen durch Komma getrennt werden.

Achten Sie darauf, daß numerische Variablen in der "varlist" auch nur Zahlen durch READ zugewiesen erhalten; andernfalls wird der READ-Vorgang mit einer Fehlermeldung abgebrochen.

Beispiele:

```
READ A$,B,G$
```

RECORD Anweisung 7

Schreibweise:

```
RECORD # logische_dateinr, satznr
[,bytenr]
```

Zweck:

Positioniert den Dateizeiger für eine Datei mit Direktzugriff.

Kommentar:

Der Dateizeiger bestimmt, an welcher Stelle in einer Relativdatei (Datei mit Direktzugriff) der nächste Lese- oder Schreibvorgang erfolgt. Mit dieser Anweisung ist es möglich, den Zeiger auf ein beliebiges Byte innerhalb einer solchen Datei zu positionieren. Anschließend kann mit INPUT# oder PRINT# gelesen oder geschrieben werden.

Der Parameter "logische_dateinr" ist die Nummer einer Relativdatei, die zuvor mit OPEN (s. dort) geöffnet wurde.

Als "satznr" müssen Sie einen ganzzahligen Wert zwischen 1 und 65535 angeben. Sie wählen dadurch den logischen Satz innerhalb der Datei aus, auf den zugegriffen wird.

Mit "bytenr" bestimmen Sie, ab welchem Byte (Zeichen) im ausgewählten Satz zugegriffen werden soll. Erlaubt sind Werte zwischen 1 und 255; voreingestellt ist der Wert 1.

Wenn Sie auf einen nicht vorhandenen Satz positionieren, so erhalten Sie die Fehlermeldung RECORD NOT PRESENT.

Beispiele:

```
RECORD #3,150,17
```
(Wählt den 150. Satz aus und setzt den Dateizeiger auf das 17te Zeichen innerhalb dieses Satzes)

REM — Anweisung — 2,7

Schreibweise:

REM

Zweck:

Kennzeichnet einen Kommentar.

Kommentar:

Kommentare werden vom BASIC-Interpreter überlesen; sie haben keinen Einfluß auf die Ausführung des Programms.

Kommentare nehmen im Textspeicher (BASIC-Speicher) Platz weg; auch verlangsamen sie die Ausführung des Programms. Sie tragen aber wesentlich zur Lesbarkeit des Programms bei. Nach Monaten, in denen Sie sich mit einem Programm nicht beschäftigt haben, sind Kommentare vielleicht Ihr einziger Anhaltspunkt, um Ihnen dunkle Stellen im Programm zu erläutern. Geizen Sie also nicht damit!

Auch ist es weitverbreitete Praxis, einzelne Zeilen aus einem BASIC-Programm (z.B. zu Testzwecken) "auszublenden", indem man REM an deren Anfang setzt. Dies ist besser, als die Zeilen zu löschen, da man sie so im Bedarfsfall noch zur Verfügung hat.

Beispiele:

```
10 REM FOR I = 1 TO 10
20 PRINT "KOMMENTAR" :REM DRUCKT EINMAL "KOMMENTAR"
30 REM NEXT
```

RENAME Befehl 7

Schreibweise:

```
RENAME alter_name TO neuer_name
[,Dlaufwerk] [,Ugerätenummer]
```

Zweck:

Benennt Diskettendateien um.

Kommentar:

Die Datei namens "alter_name" auf der Diskette in der ausgewählten Floppy erhält den Namen "neuer_name". Die Parameter können Stringkonstanten sein (in Anführungszeichen), oder Variable (in Klammern), dürfen aber höchstens 16 Zeichen lang sein.

Beispiele:

```
RENAME "SCHROTT" TO "QUATSCH" ,D1
```
(Umbenennen auf Laufwerk 1)

RENUMBER **Befehl** 7

Schreibweise:

```
RENUMBER [neu_start, schrittw, alt_start]
```

Zweck:

Numeriert ein BASIC-Programm oder Ausschnitte daraus neu.

Kommentar:

"neu_start" ist die Zeilennummer, mit der der neu zu numerierende Programmausschnitt beginnen soll. Der Wert 10 ist voreingestellt.

"schrittw" gibt die Schrittweite an, mit der numeriert wird. Auch hier ist 10 voreingestellt, d.h. es wird in Zehner-Schritten neu numeriert.

"alt_start" gibt die Zeilennummer des BASIC-Programms im Arbeitsspeicher an, ab der mit der Numerierung begonnen werden soll. Auch hier ist 10 voreingestellt.

Alle Bezüge auf Zeilennummern im Programm (z.B. hinter GOTO, GOSUB, RESTORE etc.) werden durch RENUMBER an die neuen Verhältnisse angeglichen.

Auch wenn es so scheinen mag: Sie können mit RENUMBER keine Programmzeilen verschieben. Der Versuch, einen Zeilenbereich durch entsprechendes Neunumerieren nach vorne oder hinter über bereits existierende Zeilen zu verschieben führt auf einen SYNTAX ERROR (!).

Der Befehl darf nur im Direktmodus gegeben werden.

Beispiele:

```
RENUMBER
```
(Das gesamte Programm wird in Zehnerschritten neunumeriert)

```
RENUMBER 1000,20,180
```
(Ab Zeile 180 wird neu numeriert; die erste Zeile erhält die Nummer 1000; es wird in 20er-Schritten hochgezählt)

RESTORE Anweisung 2.7

Schreibweise:

```
RESTORE [zeilennr]
```

Zweck:

Setzt den DATA-Zeiger zurück.

Kommentar:

Eine DATA-Liste kann dadurch mehrmals in einem Programm für READ verwendet werden. Mit "zeilennr" können Sie wählen, für welche DATA-Zeile der Zeiger zurückgesetzt werden soll. Fehlt die Angabe, wird der Zeiger auf den Anfang des BASIC-Programms gesetzt.

Beispiele:

```
10 DATA 1,2,3,4,5,6,7
20 READ A,B,C,D
30 RESTORE
40 READ E,F,G
50 PRINT A,B,C,D,E,F,G
60 RESTORE
70 READ A,B,C,D,E,F,G
80 PRINT A,B,C,D,E,F,G
```

RESUME — Anweisung — 7

Schreibweise:

```
RESUME
oder
RESUME NEXT
oder
RESUME zeilennr
```

Zweck:

Setzt das Programm nach Ausführung einer Fehlerbehandlung fort.

Kommentar:

Falls im Programm ein Fehler aufgetreten ist und mittels TRAP (s. dort) eine "Fehlerfalle" aufgebaut wird, dann läuft BASIC die Zeilennummer der Falle an. Steht hier nun die RESUME-Anweisung, so wird das Programm wieder fortgesetzt.

Ohne Parameterangabe erfolgt die Fortsetzung mit der Anweisung, die den Fehler verursacht hat.

Geben Sie RESUME NEXT an, so wird mit der Anweisung hinter der fehlerhaften weitergemacht.

Geben Sie eine "zeilennr" an, so wird mit der entsprechenden Programmzeile fortgefahren.

Beispiele:

```
100 TRAP 500
110 FR I = 1 TO 10
120 TRAP 1000
130 NEXT
140 PRINT "JETZT BIN ICH FERTIG"
500 RESUME NEXT
1000 PRINT "NEXT OHNE FOR GEHT NICHT!"
1010 RESUME NEXT
```

RETURN — Anweisung — 2,7

Schreibweise:

RETURN

Zweck:

Rückkehr aus einem Unterprogramm.

Kommentar:

Ein mittels GOSUB (s. dort) angesprungenes Unterprogramm wird beendet. Die Programmausführung wird mit der Anweisung fortgesetzt, die hinter dem Unterprogramm-Aufruf folgt.

Unterprogramme können auch über GOTO verlassen werden; allerdings wird dann der interne Kellerspeicher aufgebläht.

RGR — Funktion — 7

Schreibweise:

RGR(x)

Zweck:

Gibt den aktuellen Grafik-Modus an.

Kommentar:

Der Parameter "x" ist ein Dummy-Wert. Er muß zwar angegeben werden, hat aber keinen Einfluß auf die Funktion. RGR liefert Werte zwischen 0 und 5 (s. GRAPHIC-Befehl)

RIGHT$ String-Funktion 2,7

Schreibweise:

```
RIGHT$ (str$,ausdr)
```

Zweck:

Gibt einen Teilstring mit den letzten Zeichen aus dem Gesamtstring "str$" zurück.

Kommentar:

Der Parameter "ausdr" bestimmt, wieviele Zeichen vom Ende des Strings "str$" in das Ergebnis übernommen werden. Ein Wert von 5 liefert also z.B. die fünf letzten Zeichen in "str$". "ausdr" darf nur Werte zwischen 0 und 255 annehmen. Hat es den Wert 0, dann wird der leere String "" als Ergebnis zurückgeliefert.

Ist der Wert von "ausdr" größer als die Gesamtlänge von "str$", dann wird der gesamte String "str$" als Ergebnis geliefert.

Beispiel:

```
PRINT RIGHT$("GEMEINSCHAFT",6)
```

RND — Funktion — 2,7

Schreibweise:

RND(n)

Zweck:

Liefert eine Zufallszahl.

Kommentar:

RND hat als Wert eine Zufallszahl zwischen 0.0 und 1.0. Zur Erzeugung einer Zufallszahl führt der Computer eine Reihe von Berechnungen mit einer Zahl, der sog. "Startzahl" aus. Um zu verhindern, daß ein Programm bei mehrmaliger Anwendung immer dieselbe Folge von Zufallszahlen liefert (was nicht mehr sonderlich zufällig ist; deshalb spricht man auch von "Pseudo-Zufallszahlen"), muß die Anfangszahl neu gesetzt werden.

Der Zahlenwert des Parameters "n" hat folgende Bedeutung: ist "n" negativ, dann wird die Startzahl neu gesetzt und RND liefert eine neue Zufallszahlenfolge. Am besten beginnt man eine Folge solcher Zahlen mit RND(-TI) und bezieht sich damit auf die Systemuhr.

Bei einem positiven Wert von "n" wird einfach, basierend auf der letzen Startzahl, eine neue Zahl geliefert.

Beispiele:

```
PRINT INT(RND(-TI) *6 + 1)
```
(Liefert Zufallszahlen von 1 bis 6 und simuliert somit einen Würfel)

RREG Anweisung 7

Schreibweise:

```
RREG [a] [, x] [, y] [, s]
```

Zweck:

Weist Variablen den Inhalt der Prozessorregister zu.

Kommentar:

Die Variaben "a", "x", "y" und "s" erhalten den Inhalt des A-, X-, Y-, und S-Register des Prozessors. Dies ist nützlich, um Werte aus einem über SYS aufgerufenen Maschinenprogramm in ein BASIC-Programm zu übernehmen.

RSPCOLOR **Funktion** 7

Schreibweise:

```
RSPCOLOR (n)
```

Zweck:

Liefert den aktuellen Multicolor-Wert für Sprites.

Kommentar:

Hat "n" den Wert 1, so liefert die Funktion den Farbcode für Multicolor 2 als Zahl zwischen 1 und 16 (s. COLOR). Bei "n" = 2 erhalten Sie die Daten für Multicolor 2.

RSPPOS — Funktion — 7

Schreibweise:

```
RSPPOS (sprite,n)
```

Zweck:

Liefert die aktuelle Position oder Geschwindigkeit eines Sprites auf dem Grafik-Bildschirm.

Kommentar:

Mit dem Parameter "sprite" (0 bis 7) wählen Sie aus, über welches Sprite Sie informiert werden wollen.

Hat der Parameter "n" den Wert 0, so erhalten Sie die x-Position auf dem Grafikbildschirm, bei "n" = 1 gibt's die y-Position und bei "n" = 2 die aktuelle Geschwindigkeit.

RSPRITE — Funktion — 7

Schreibweise:

```
RSPRITE (n,m)
```

Zweck:

Liefert die Attribute eines Sprites.

Kommentar:

Mit dem Parameter "n" wählen Sie das Sprite aus, das Sie untersucht haben wollen. Der Parameter "m" bestimmt, welche Informationen Sie bekommen:

0	Die Funktion liefert 1, falls das Sprite eingeschaltet ist und 0 sonst.
1	Liefert die Spritefarbe.
2	Liefert 1, falls das Sprite Priorität über den Hintergrund hat und 0 sonst.
3	Ist das Sprite in x-Richtung ausgedehnt? (1=Ja / 0=Nein)
4	Ist das Sprite in y-Richtung ausgedehnt? (1=Ja / 0=Nein)
5	Multicolor-Modus für das Sprite aktiv? (1=Ja / 0=Nein)

RUN — Befehl — 2,7

Schreibweise:

RUN [zeilennr]

Zweck:

Führt ein BASIC-Programm aus.

Kommentar:

Ein im Arbeitsspeicher befindliches BASIC-Programm (eingetippt oder mit LOAD oder DLOAD geladen) wird gestartet. Wenn Sie keinen Parameter angeben, dann wird am Anfang des Programmes mit der Ausführung begonnen. Ansonsten startete das Programm mit der angegebenen "zeilennr".

RUN löscht den Variablenbereich. Ein angehaltenes Programm sollten Sie also besser mit GOTO fortsetzen, wenn es mit den alten Daten weiterrechnen soll (s. Beispiel).

Beispiele:

```
10 A = 5
20 PRINT A
30 RUN 40
40 PRINT A
```

RWINDOW **Funktion** 7

Schreibweise:

```
RWINDOW (n)
```

Zweck:

Liefert Informationen über das aktuelle Bildschirm-Fenster (s. WINDOW-Befehl)

Kommentar:

Falls der Parameter "n" den Wert 0 hat, erhalten Sie die Anzahl der Zeilen im aktuellen Fenster, bei "n"=1 die Anzahl der Spalten. Für "n"=3 liefert RWINDOW entweder 40 oder 80, je nachdem, ob das Fenster auf dem 40- oder dem 80-Zeichen-Bildschirm liegt.

SAVE — Befehl — 2,7

Schreibweise:

```
SAVE [dateiname ,gerätenummer, endemarke]
```

Zweck:

Speichert ein BASIC-Programm ab.

Kommentar:

Das Programm kann auf ein wählbares Gerät gespeichert werden. Zum Speichern auf Diskette gibt es jedoch den einfacheren DSAVE-Befehl (s. dort).

Falls SAVE ohne weitere Parameter eingegeben wird, so wird das BASIC-Programm auf die Datasette gespeichert. Dabei wird ein eventuell bereits auf dem Band befindliches Programm durch das neue überschrieben - also Vorsicht! Es erscheint vor dem Abspeichern die Meldung PRESS RECORD & PLAY ON TAPE, die Sie auffordert, den Aufnahmeknopf auf der Datasette zu drücken und diese einzuschalten.

Beim Speichern auf Diskette muß der Parameter "dateiname" angegeben sein; auch beim Speichern auf Datasette schadet er nichts, da das Programm so identifiziert werden kann. Der Parameter kann entweder als Stringkonstante (in Anführungszeichen) oder als Variable (in runden Klammern) angegeben werden.

Über den Parameter "gerätenummer" können Sie auswählen. ob das Programm auf Band (1; ist voreingestellt) oder auf Diskette (8; besser mit DSAVE) gespeichert werden soll.

Wählen Sie die Datasette ("gerätenummer" ist 1), dann können Sie auch den Parameter "endemarke" angeben. Hat er den Wert 1, dann wird hinter das Programm eine Bandende-Marke geschrieben. Diese bewirkt, daß beim Laden von diesem Band (s. LOAD) nicht hinter der Marke weitergesucht wird.

Beispiele:

```
SAVE
```
(Das Programm wird ohne Namen auf Datasette gespeichert)

```
SAVE "PROGRAMM",1,1
```
(PROGRAMM kommt auf Datasette mit Endemarke)

SCALE — Grafik-Anweisung — 7

Schreibweise:

```
SCALE [n] [,xmax, ymax]
```

Zweck:

Schaltet die Bildschirm-Skalierung im Grafikmodus ein oder aus. Falls "n" den Wert 1 hat, wird sie eingeschaltet, für "n"=0 wieder ausgeschaltet. "xmax" und "ymax" geben die maximalen Koordinatenwerte auf der waagerechten bzw. senkrechten Achse an. Zulässige Werte für "xmax" : 320 bis 1023 (im Multicolor-Modus: 160 bis 1023). Zulässige Werte für "ymax": 200 bis 1023.

Kommentar:

Mit der SCALE-Anweisung ist es möglich, ein anderes Koordinatennetz über den Grafik-Bildschirm zu legen. Im normalen Grafikmodus (nicht im Multicolor-Modus!) arbeiten Sie so mit einer Auflösung von 360 mal 200 Bildpunkten. Diese können auch durch ein Koordinatensystem angesprochen werden, das auf der waagerechten (x-)Achse 360 Punkte und auf der senkrechten (y-)Achse 200 Punkte hat. Der Mittelpunkt des Bildschirms liegt also in diesem System bei 160,100.

Wenn Sie die SCALE-Anweisung benutzen, dann legen Sie über das Gitternetz von Bildpunkten einen neuen Maßstab. Die Anzahl der auf dem Bildschirm vorhandenen Bildpunkte bleibt natürlich gleich, aber sie können (und müssen) nach Einschalten der geänderten Skalierung anders angesprochen werden. Wenn Sie z.B. den Befehl SCALE 1,400,400 geben, dann kennt Ihr Koordinatensystem auf beiden Achsen jeweils 400 Punkte; der Mittelpunkt des Grafikbildschirms liegt jetzt bei 200,200. Dies ist derselbe Bildpunkt, der bei normalem Koordinatensystem mit 160,100 angesprochen wird!

Wenn Sie also die Anzahl der Koordinatenpunkte erhöhen, so erhöht sich deswegen nicht dieAuflösung des Bildschirms (die Anzahl der vorhandenen Bildpunkte); nur werden diese Bildpunkte jetzt anders adressiert. Dies ist sinvoll, wenn man z.B. Grafikprogramme, die für einen anderen Bildschirm geschrieben wurden, auf den C128 übernehmen will. Viele

hochauflösenden Grafikbildschirme arbeiten mit 1024 mal 1024 Bildpunkten. Programme für diese Bildschirme können durch Verwendung von SCALE 1,1023,1023 problemlos auf den C128 übernommen werden.

Beachten Sie, daß Programme, die mit der normalen Skalierung arbeiten (360 mal 200), nach Veränderung durch SCALE ihre Zeichnungen verzerrt (verkleinert) darstellen!

Die SCALE-Anweisung hat keine Auswirkung auf bereits auf dem Grafik-Bildschirm vorhandene Zeichnungen; diese bleiben unverändert erhalten.

Beispiele:

```
SCALE 1,500,500
```
(Grafikbildschirm wird so adressiert, daß der Mittelpunkt bei 250,250 liegt)

```
SCALE 0
```
(Standard-Skalierung (360 mal 200) wieder einschalten)

SCNCLR — Grafik-Anweisung — 7

Schreibweise:

```
SCNCLR [n]
```

Zweck:

Löscht den aktuellen oder den angegebenen Bildschirm.

Kommentar:

SCNCLR ohne Parameter löscht den aktuellen Bildschirm (Text mit 40 oder 80 Zeichen, Grafik oder Multicolor). Ist der Parameter "n" angegeben, so kann auch ein anderer als der gerade angezeigte Bildschirm gelöscht werden. Dazu muß "n" folgende Werte haben:

0	40-Zeichen-Text
1	Grafikbildschirm
2	Grafikbildschirm geteilt
3	Multicolor
4	Multicolor geteilt
5	80-Zeichen-Text

Haben Sie mit WINDOW (s. dort) ein Textfenster definiert, so bezieht sich der SCNCLR-Befehl nur auf dieses Fenster.

Beispiele:

```
SCNCLR
```
(löscht den aktuellen Bildschirm)

```
SCNCLR 4
```
(löscht den geteilten Multicolor-Bildschirm)

SCRATCH **Befehl** 7

Schreibweise:

```
SCRATCH dateiname [,Dlaufwerk]
[,Ugerätenummer]
```

Zweck:

Löscht eine oder mehrere Dateien auf dem angegebenen Gerät.

Kommentar:

Der Befehl löscht den Eintrag im Inhaltsverzeichnis für die angegebene(n) Datei(en). Die Daten in den Dateien bleiben zwar (bis zum nächsten Beschreiben der Diskette) erhalten, sind aber für den Computer nicht mehr erreichbar.

Um ein versehentliches Löschen zu verhindern, werden Sie mit der Meldung ARE YOU SURE erst um Bestätigung gebeten; Sie müssen mit Y quittieren, falls gelöscht werden soll.

Der Parameter "dateiname" muß ein String sein, der entweder als Konstante (in Anführungszeichen) oder als Variable (in runden Klammern) angegeben werden kann. Er bestimmt, welche Datei gelöscht wird.

In "dateiname" können auch Jokerzeichen ("*" und "?") vorkommen, wodurch mit einem Befehl mehrere Dateien gelöscht werden können. Für alle Einträge im Inhaltsverzeichnis, die auf das angegebene Namensmuster passen, wird die Löschung durchgeführt.

Geben Sie das SCRATCH-Kommando im Direktmodus, so wird die erfolgreiche Löschung mit der Meldung "01, FILES SCRATCHED, nn, 00" bestätigt; "nn" ist hierbei die Anzahl der gelöschten Dateien.

Beispiele:

```
SCRATCH "SICHERUNG"
```
(löscht die Datei Sicherung von Floppy 0)

```
SCRATCH "*",U9
```
(alle Dateien auf Floppy 0 im Gerät 9 werden gelöscht)

```
10 A$="????.PRG"
20 SCRATCH (A$)
```
(löscht alle Dateien auf der Diskette, deren Name 4 Zeichen lang ist und in ".PRG" endet)

SGN

Funktion 2,7

Schreibweise:

SGN(n)

Zweck:

Gibt an, ob eine Zahl positiv, negativ oder gleich 0 ist.

Kommentar:

Falls "n" größer 0 ist, so liefert SGN den Wert 1; ist "n" kleiner 0, so erhalten Sie -1 und für "n"=0 gibt's den Wert 0.

Beispiele:

```
PRINT SGN(77), SGN(0), SGN(-44)
```

SIN — Funktion — 2,7

Schreibweise:

`SIN(x)`

Zweck:

Liefert den Sinus eines Winkels.

Kommentar:

Der Parameter "x" ist der fragliche Winkel im Bogenmaß.

Beispiel:

```
10 GRAPHIC 1,1
20 FOR I = 1 TO 900
30 X = X + .3
40 DRAW ,X,100+(15*SIN(X))
50 NEXT
```

SLEEP **Anweisung** 7

Schreibweise:

```
SLEEP n
```

Zweck:

Hält die Programmausführung für eine bestimmte Zeit an.

Kommentar:

Mit dem Parameter "n" können Sie angeben, wieviele Sekunden lang der Rechner sich nicht mehr rühren soll. ("n" darf Werte zwischen 1 und 65535 annehmen, die Sekunden darstellen).

Dies ist z.B. nützlich, um ein Programm eine definierte Zeitspanne lang zu bremsen, etwa wenn man mehrere Bildschirmgrafiken zeichnet und etwas Zeit vergehen lassen will, ehe man die nächste Grafik auf den Bildschirm zaubert.

Sollten Sie sich mit dem Sekundenwert mal vertan haben: der Computer kann mit der RUN/STOP-Taste vorzeitig aufgeweckt werden. So man die Zeilennummer kennt, kann man dann mit GOTO weitermachen.

Die von SLEEP gezählte Zeit ist im FAST-Modus (s. dort) die gleiche wie im SLOW-Modus (s. dort).

Beispiele:

```
SLEEP (65535)
```
(Der Rechner stellt sich ca. 18 Stunden lang tot.)

SLOW **Befehl** 7

Schreibweise:

SLOW

Zweck:

Stellt den 1-Megahertz-Systemtakt wieder ein.

Kommentar:

Bei rechenintensiven Programmteilen empfiehlt es sich, dem Computer mit der FAST-Anweisung (s. dort) auf die Sprünge zu helfen. Allerdings ist es dann mit der Anzeige des 40-Zeichen-Bildschirms vorbei. Ist der Rechenvorgang abgeschlossen, so kann man mit SLOW im Programm die Anzeige wieder herholen.

Das Beispielprogramm zeigt Ihnen, um wieviel schneller dieselbe Berechung im FAST-Modus durchgeführt wird.

Beispiele:

```
10 A = TI
20 FOR I = 1 TO 1000
30 K = I * 34/I
40 NEXT
50 PRINT "IM SLOW-MODUS: "; TI - A
60 FAST
70 A = TI
80 FOR I = 1 TO 1000
90 K = I * 34/I
100 NEXT I
110 T = TI -A
120 SLOW : PRINT "IM FAST-MODUS: ";T
130 END
```

SOUND — Anweisung — 7

Schreibweise:

```
SOUND stimme, frequenz, dauer, richtung,
grenz_freq, schrittweite, wellenform,
pulsbreite
```

Zweck:

Erzeugt Klänge mit wählbarer Frequenz, Dauer, Glissando, und Wellenform und gibt sie auf dem Lautsprecher aus.

Kommentar:

Mit der SOUND-Anweisung haben Sie in hohem Maße die Kontrolle über die erzeugten Klänge. Allerdings wird man zum Spielen von Melodien wohl eher die PLAY-Anweisung (s. diese) verwenden, da sie einfacher zu handhaben ist.

Der erste Parameter, "stimme", gibt an, in welcher Stimme der erzeugte Ton erklingen soll. Erlaubt sind hier Werte zwischen 1 und 3.

Der Parameter "frequenz" bestimmt die Tonhöhe. Er steht jedoch nicht für die tatsächliche Tonfrequenz, sondern ergibt sich erst durch Umrechung aus dieser. Erlaubt sind Werte zwischen 0 und 65535. Die Frequenz des gesuchten Tones muß durch den Wert 0.058721 dividiert werden, um den Parameter für die SOUND-Anweisung zu erhalten. Um den Kammerton "a" zu erhalten (Frequenz 440 Hz) müssen Sie also für den Parameter "frequenz" den Wert 440 / 0.058721 = 7493 eingeben. Die folgende Tabelle zeigt die Werte für die Tonleiter (Oktave 4 in der Commodore-Sprechweise):

Ton	Physikalische Frequenz	"frequenz"-Parameter
C	261.6	4455
Cis/Des	277.2	4720
D	293.7	5001
Dis/Es	311.1	5298
E	329.6	5613
F	349.2	5947
Fis/Ges	370.0	6301
G	392.0	6676
Gis/As	415.3	7072
A	440.0	7493
Ais/Be	466.2	7939
H	493.9	8411
c	523.5	8911

Beachten Sie, daß der Faktor 0.058721 von der Netzfrequenz abhängig ist und sich also von Land zu Land ändert!

Der Parameter "dauer" gibt an, wie lange der Ton andauern soll. Die Angabe erfolgt in sechzigstel von Sekunden; zulässig sind Werte zwischen 0 und 32767.

Mit dem Parameter "richtung" können Sie den Ton "verschieben", also einen Glissando-Effekt erzeugen. Wenn Sie diesen Parameter mit einem Wert besetzen, dann sollten Sie auch Angaben zu den Parametern "grenz_freq" und "schrittweite" machen. Mögliche Werte sind hier:

0	erzeugt ein Glissando nach oben
1	erzeugt ein Glissando nach unten
2	Ton oszilliert (auf- und abschwellen wie bei einer Sirene)

Soll der Ton verschoben werden (Glissando-Effekt; Parameter "richtung" ist besetzt), dann sollten Sie auch angeben, wo mit dem Verschieben Schluß ist. Dies sagt der Parameter "grenz_freq". Ein zu verschiebender Ton wird zwischen den Werten "frequenz" und "grenz_freq" verschoben. Geht das Glissando nach oben, so nimmt BASIC den kleineren der beiden Werte und verschiebt dann solange nach oben, bis der größere erreicht ist. Bei einem Verschieben nach unten wird genau umgekehrt vorgegangen. Soll der Ton oszillieren, so wird seine Tonhöhe zwischen den beiden Werten "frequenz" und "grenz_freq" hin- und herwandern. Voreingestellt ist hier der Wert 0, d.h. wenn Sie nichts angeben, dann kommt ein Glissando nach oben immer von ganz unten (aus den Bässen), egal mit welchem Wert für "frequenz" Sie einsteigen. Ob allerdings bei Glissando-Effekten eine Grenzfre-

quenz überhaupt erreicht wird, hängt von der Tondauer und dem nächsten Parameter ab!

Der Parameter "schrittweite" gibt bei Glissando-Effekten (Parameter "richtung" ist gesetzt) an, wie schnell das Glissando sein soll. Je kleiner der Wert von "schrittweite" ist, desto langsamer wird verschoben. Werte zwischen 0 (Ton bleibt konstant) und 32767 sind erlaubt. 0 ist voreingestellt, d.h. wenn Sie nichts angeben, rührt sich nichts! Bei sehr schnellem Verschieben kann es passieren, daß die Grenzfrequenz für das Verschieben erreicht wird, ehe die Tondauer verstrichen ist. Dann fängt die Verschieberei einfach wieder von Vorne an - damit lassen sich leicht Sirenen-Effekte erzielen. Auch ist es natürlich möglich, daß bei kleinem "schrittweite"-Wert die Grenzfrequenz in der angegebenen Tondauer nicht erreicht wird.

Der Parameter "wellenform" bestimmt, ob eine Dreiecksschwingung (0), eine Sägezahnschwingung (1), eine Rechteckschwingung mit variabler Pulsbreite (2) oder Rauschen (3) erzeugt wird. Wählen Sie für "wellenform" den Wert 2 (variable Pulsbreite), dann können Sie auch noch den nächsten Parameter angeben und den Klang noch weiter beeinflussen.

Der Parameter "pulsbreite" beeinflußt den Klang, falls "wellenform" den Wert 2 hat. Zugelassen sind Werte zwischen 0 und 4095. Bei einem Wert von 2048 erhalten Sie eine Klangkurve mit quadratischer Gestalt. Dies ist der "satteste" Klang. Andere Werte für "pulsbreite" machen den Klang "dünner".

Sollen mehrstimmige Klänge ertönen, so müssen Sie mehrere SOUND-Befehle mit unterschiedlichen Stimmen erteilen. Kommen zwei SOUND-Befehle mit der gleichen Stimme hintereinander, so wartet BASIC, bis der erste Ton fertig ist und spielt dann erst den zweiten.

Beispiele:

Anmerkung: die Beipiele sollten hintereinander ausprobiert werden.

```
SOUND 1,7493,60
```
(der Kammerton für eine Sekunde)

```
SOUND 1,7493,125,0,6493,8
```
(wir nähern uns dem "a" von unten - trotz "richtung" = 0!)

```
SOUND 1,7493,125,1,6493,8
```
(und jetzt von oben -- obwohl "richtung" den Wert 1 hat)

```
SOUND 1,7493,125,2,6493,16
```
(ein Jaulen)

```
SOUND 1,7493,125,1,6493,16
```
(nur der Parameter "richtung" hat sich geändert!)

```
SOUND 1,7493,125,1,6493,,1
```
(Sägezahnschwingung; die Tonhöhe ändert sich nicht, weil keine "schrittweite" angegeben ist)

```
SOUND 1,7493,125,1,6493,,2
```
(die perfekte Rechteck-Schwingung)

```
SOUND 1,7493,125,1,6493,,2,40
```
(jetzt klingt's schon dünner)

```
10 SOUND 1,4455,60
20 SOUND 2,5613,60
30 SOUND 3,6676,60
```
(ein Dreiklang)

SPC

Funktion 2,7

Schreibweise:

```
SPC(n)
```

Zweck:

Gibt Leerzeichen auf dem Bildschirm oder einem anderen Ausgabegerät aus.

Kommentar:

Der Parameter "n" gibt an, wieviele Leerzeichen ausgegeben werden sollen. Erlaubt sind Werte zwischen 0 und 255.

Die Funktion ist nur in Ausgabe-Anweisungen sinnvoll (z.B. PRINT) und hat dort die Wirkung eines variablen Tabulators. Durch das Ausgeben von Leerzeichen können Druckausgaben eingerückt werden.

Beispiele:

```
PRINT SPC(15);"*"
(druckt den Stern auf Spalte 16)
```

SPRCOLOR **Anweisung** 7

Schreibweise:

```
SPRCOLOR [multi_1] [,multi_2]
```

Zweck:

Setzt die Vordergrund-Farbe für Multicolor-Sprites.

Kommentar:

Mit "multi_1" wird die Farbe 1, mit "multi_2" wird die Farbe 2 für alle Multicolor-Sprites gesetzt. Die Parameter können Werte zwischen 1 und 16 annehmen (s. COLOR-Anweisung).

SPRDEF **Befehl** 7

Schreibweise:

SPRDEF

Zweck:

Ruft den eingebauten Sprite-Editor.

Kommentar:

Dieser Befehl ermöglicht es Ihnen, auf bequeme Weise Sprites zu entwerfen und abzuspeichern. Sie bewegen dabei einen kreuzförmigen Sprite-Cursor über eine Arbeitsfläche von 24 mal 21 Punkten, auf der das Sprite vergrößert dargestellt ist. Außerdem sehen Sie das Sprite auch noch in Originalgröße am rechten oberen Bildschirmausschnitt. Der Sprite-Editor kennt eine Anzahl Befehle, die wie folgt lauten:

1...8
Zu Beginn der Arbeit werden Sie nach der Nummer des Sprites gefragt, das Sie bearbeiten wollen. Sie können hier Werte zwischen 1 und 8 eingeben.

A
Schaltet die automatische Wiederholfunktion für den Sprite-Cursor aus und bei nochmaliger Betätigung wieder an. Der Sprite-Cursor bleibt dadurch nach Setzen eines Punktes an der gleichen Stelle und bewegt sich nicht hinter den gesetzten Punkt.

M
Wählt den Multicolor-Modus eins oder zwei für das Sprite.

CRSR-Tasten
Steuern den Sprite-Cursor in die angegebene Richtung

SHIFT-RETURN
Speichert das entworfenen Sprite und verlässt den Sprite-Editor.

HOME-Taste
Der Sprite-Cursor geht in die linke obere Ecke der Arbeitsfläche.
CLR-Taste Löscht die Arbeitsfläche.

1 bis 4
Wählt eine der vier möglichen Farbquellen und setzt so einen Bildpunkt.

CTRL-1 ... CTRL-8
Wählt die Vordergrundfarbe für das Sprite (Farbnummern 1 bis 8).

C=-1 ... C=-8
Wählt die Vordergrundfarbe für das Sprite (Farbnummer 9 bis 16)

STOP-Taste
Verläßt den Sprite-Editor ohne Sichern des Sprites

X
Dehnt das Sprite in x-Richtung

Y
Dehnt das Sprite in y-Richtung

SPRITE Anweisung 7

Schreibweise:

```
SPRITE nummer, ein_aus, vordergrund,
priorität, x-exp, y-exp, modus
```

Zweck:

Schaltet Sprites ein und definiert ihre Eigenschaften.

Kommentar:

Ein mit dem Sprite-Editor (s. SPREDEF) entworfenes und gespeichertes Sprite wird erst sichtbar, wenn es mit dem SPRITE-Befehl eingeschaltet wird. Außerdem können Sie mit diesem Befehl noch verschiedene Eigenschaften des Sprites beeinflussen. Die Parameter haben folgende Wirkung:

"nummer" ist eine Zahl zwischen 1 und 8, die das Sprite auswählt, um das es geht.

"ein_aus" ist entweder 1, um das Sprite einzuschalten und somit sichtbar zu machen, oder 0, um es auszuschalten. Wenn Sie ein in Bewegung befindliches Sprite (s. MOVSPR) ausschalten, dann verharrt es an seiner augenblicklichen Position. Wenn Sie es wieder einschalten, dann bewegt es sich von da weiter.

"vordergrund" wählt die Farbe für den Vordergrund. Die Farbnummern sind bei COLOR beschrieben; erlaubt sind Werte zwischen 1 und 16.

"priorität" ist 1, wenn sich das Sprite vor Anzeigedaten bewegen soll und 0, wenn es dahinter "lebt".

"x-exp" ist 1, wenn das Sprite in der Waagerechten ausgedehnt werden soll.

"y-exp" ist 1, wenn das Sprite in der Senkrechten ausgedehnt werden soll.

"modus" ist 1, wenn Sie ein normales Sprite haben wollen (24 mal 21 Bildpunkte) und 0 für ein Multicolor-Sprite (12 mal 21 Punkte).

Beispiel:

```
SPRITE 1,1,6,0,1
```
(Sprite 1 wird eingeschaltet und in Farbe 6 - grün - vor den Bildschirmdaten und auf der x-Achse gedehnt angezeigt)

SPRSAV Anweisung 7

Schreibweise:

```
SPRSAV n,str$
oder
SPRSAV str$,n
```

Zweck:

Speichert ein Sprite als Bitmuster in einem String ab und umgekehrt.

Kommentar:

In der ersten Form der Anweisung wird das Sprite mit der Nummer "n" im String "str$" abgelegt. In der zweiten Form wird der Inhalt des Strings "str$" als neues Sprite mit der Nummer "n" definiert.

Das Beispielprogramm zeigt, wie man ein einmal erzeugtes Sprite (hier die Nummer 1) "klonen" kann, indem man sein Muster allen anderen Sprites zuweist.

Der Befehl ist auch nützlich, um eine zuvor mit Grafik-Befehlen erstellte Zeichnung als Sprite zu definieren. Dazu wird erst mit SSHAPE (s. dort) die Grafik in einen String gebracht und dann mit SPRSAV einem Sprite als Definition zugewiesen.

Beispiele:

```
10 SPRSAV 1,A$
20 FOR I = 2 TO 8
30 SPRSAV A$,I
40 NEXT
```

SQR — Funktion — 2,7

Schreibweise:

```
SQR(ausdr)
```

Zweck:

Berechnet die Quadratwurzel einer Zahl.

Kommentar:

Der Wert von "ausdr" muß größer oder gleich 0 sein.

Beispiele:

```
PRINT SQR(13*13)
```

SSHAPE Anweisung 7

Schreibweise:

```
SSHAPE str$, x1, y1, x2, y2
```

Zweck:

Speichert einen rechteckigen Ausschnitt aus dem Graphik-Bildschirm in einer Stringvariable ab.

Kommentar:

Sie können sich mit dieser Methode Grafiken "aufheben" und sie auch wieder auf dem Bildschirm erscheinen lassen, ja sogar bewegen (s. GSHAPE-Anweisung). Allerdings ist die Stringlänge in BASIC auf 255 Zeichen begrenzt, so daß nur Grafiken gespeichert werden können, die in 255 Grafik-Zeichen untergebracht werden können.

"str$" ist die Stringvariable, in der die Grafik gespeichert werden soll.

"x1,y1" bezeichnet die linke obere Ecke des Bildschirmbereichs, den Sie speichern wollen.

"x2,y2" bezeichnet die rechte untere Ecke des Bildschirmbereichs, den Sie speichern wollen.

Sie können nach folgender Formel berechnen, ob Ihre Grafik in einem String Platz hat:

```
Länge=INT((ABS(x1-x2)+1)/8+.99)*(ABS(y1-y2)+1)+4
```

Im Multicolor-Modus gilt folgende Formel:

```
Länge=INT((ABS(x1-x2)+1)/4+.99)*(ABS(y1-y2)+1)+4
```

Die letzten vier Bytes des erzeugten Strings enthalten die Anzahl Zeilen und Spalten, die von SSHAPE übertragen wurden. Diese Werte werden von GSHAPE benötigt.

Beispiel:

```
SSHAPE F$, 10,10,33,31
```

STASH **Befehl** 7

Schreibweise:

```
STASH anzahl, woher, bank, wohin
```

Zweck:

Speichert Bytes aus dem BASIC-Variablenspeicher (Bank 1) in eine wählbare andere Bank.

Kommentar:

Variablen können damit zum Beipiel in eine Speicher-Erweiterung kopiert werden.

Der Parameter "anzahl" bestimmt, wieviele Bytes übertragen werden sollen. Erlaubt sind Werte zwischen 0 und 65535.

Der Parameter "woher" gibt an, von welcher Adresse der Bank 1 mit der Übertragung begonnen werden soll. Erlaubt sind Werte zwischen 0 und 65535.

Mit "bank" wählen Sie die Bank aus, in die übertragen werden soll. Erlaubt sind Werte zwischen 0 und 7.

Der Parameter "wohin" gibt die Zieladresse des ersten übertragenen Bytes in der gewählten Bank an.

Beispiele:

```
STASH 1000,52000,7,2000
```
(Speichert 1000 Bytes ab Adresse 52000 der Bank 1 in der Bank 7 ab Adresse 2000)

STOP — Anweisung — 2,7

Schreibweise:

STOP

Zweck:

Hält das Programm unter Angabe der Zeilennummer an.

Kommentar:

Im Unterschied zur END-Anweisung (s. diese) wird nach Anhalten des Programms die Meldung BREAK IN LINE xxx ausgegeben, wobei "xxx" die Zeile ist, in der der STOP-Befehl steht. Dies kann bei der Fehlersuche nützlich sein.

Ein mit STOP angehaltenes Programm kann mit CONT (s. dort) wieder fortgesetzt werden.

STR$ — Funktion — 2.7

Schreibweise:

```
STR$(ausdr)
```

Zweck:

Liefert die String-Darstellung einer Zahl.

Kommentar:

Damit kann man Zahlen in Zeichenketten umwandeln, z.B. um sie anschließend "nachzubehandeln". Man könnte z.B. zwischen den Hunderter- und Tausenderstellen der Zahl ein Leerzeichen einschieben (zwecks besserer Lesbarkeit; allerdings geht dies einfacher mit PRINT USING; s. dort).

Falls der Wert von "ausdr" positiv ist, dann beginnt der von STR$ gelieferte String mit einem Leerzeichen. Ist die Zahl negativ, steht an dieser Stelle das Minuszeichen.

Beispiele:

```
PRINT STR$(123), STR$(-123)
```

SWAP Anweisung 7

Schreibweise:

```
SWAP anzahl, woher, bank, wohin
```

Zweck:

Tauscht Bytes mit dem BASIC-Variablenspeicher (Bank 1) und einer wählbaren anderen Bank.

Kommentar:

Variablen können damit zwischen dem Variablenspeicher (Bank 1) und einer Speicher-Erweiterung ausgetauscht werden.

Der Parameter "anzahl" bestimmt, wieviele Bytes ausgetauscht werden sollen. Erlaubt sind Werte zwischen 0 und 65535.

Der Parameter "woher" gibt an, ab welcher Adresse der Bank 1 mit dem Austausch begonnen werden soll. Erlaubt sind Werte zwischen 0 und 65535.

Mit "bank" wählen Sie die Bank aus, mit der der Austausch vorgenommen werden soll. Erlaubt sind Werte zwischen 0 und 7.

Der Parameter "wohin" gibt die Zieladresse des ersten ausgetauschten Bytes in der gewählten Bank an.

Beispiel:

```
SWAP 1000,52000,7,2000
```
(Tauscht 1000 Bytes ab Adresse 52000 im BASIC-Variablenspeicher - Bank 1 - mit einem gleichgroßen Bereich in der Bank 7 ab Adresse 2000).

SYS

Anweisung 2,7

Schreibweise:

```
SYS adresse [,a, x, y, s]
```

Zweck:

Ruft ein Programm in Maschinensprache.

Kommentar:

64er-Modus: Der Parameter "adresse" gibt die Startadresse an. Die übrigen Parameter sind in diesem Modus nicht erlaubt.

128er-Modus: Zusätzlich zur Startadresse können auch noch Werte für die Prozessor-Register übergeben werden. Der Akkumulator erhält den Wet von "a", das X-Register den von "x", das Y-Register den von "y" und das Statusregister den von "s". Mit dem Bank-Befehl (s. dort) kann auf eine andere Speicherbank umgeschaltet werden.

TAB — Funktion — 2,7

Schreibweise:

```
TAB(ausdr)
```

Zweck:

Bewegt den Cursor bei der Ausgabe an eine bestimmte Spalte.

Kommentar:

Der Wert von "ausdr" (zwischen 0 und 255) bestimmt, an welche Spalte der Cursor bewegt werden soll.

Die Funktion sollte nur zusammen mit PRINT eingesetzt werden, da sie auf anderen Ausgabegeräten als dem Bildschirm keinen Auswirkung hat. Für den Einsatz mit PRINT# verwenden Sie also besser die SPC-Funktion (s. dort)

Beispiele:

```
PRINT TAB(12);"*"
(Sternchen wird auf Spalte 13 gedruckt)
```

TAN — Funktion — 2,7

Schreibweise:

```
TAN(ausdr)
```

Zweck:

Berechnet den Tangens eines Winkels.

Kommentar:

Der Wert von "ausdr" gibt den Winkel im Bogenmaß an.

TEMPO — Anweisung — 7

Schreibweise:

TEMPO n

Zweck:

Setzt das Metrum für Melodien.

Kommentar:

Die TEMPO-Anweisung erfüllt einen ähnlichen Zweck für Melodien, die mit PLAY gespielt werden (s. dort) wie ein Metronom für einen Musiker. Sie legt nämlich fest, wie lange eine Zähleinheit in einer Melodie dauern soll und bestimmt somit deren TEMPO.

Je größer der Wert des Parameters "n" ist, desto schneller wird die Melodie und umgekehrt. Ein Wert von 255 ist für Klangeffekte bei Spielen interessant.

Um die tatsächliche Zeitdauer einer Zähleinheit zu bestimmen, können Sie die Formel Zeitdauer = 19.22 / n verwenden, wobei "n" der in der Tempo-Anweisung gesetzte Wert ist.

Wenn Sie also TEMPO 10 geben, dann dauert eine Zähleinheit ungefähr 2 Sekunden.

Beispiel:

```
TEMPO 14
```

TRAP Anweisung 7

Schreibweise:

```
TRAP [zeilennr]
```

Zweck:

Aktiviert Fehlerfallen oder schaltet sie aus. "zeilennr" ist die Nummer einer Zeile im BASIC-Programm.

Kommentar:

Die Anweisung TRAP hat zusammen mit einer Zeilenummer folgende Wirkung: Wenn ein Fehler im Programm auftritt (Ausnahme: UNDEFINED STATEMENT), dann bricht das Programm nicht ab. Vielmehr wird zu der in "zeilennr" angegebenen Programmzeile verzweigt. Hier sollte sich ein Unterprogramm (die "Fehlerroutine") befinden, das den Fehler abfängt.

Die Fehlerroutine kann die Zeilennummer, in der der Fehler aufgetreten ist, über die Systemvariable EL abfragen. Die Systemvariable ER enthält den Fehlercode. Mittels ERR$(ER) ist es also möglich, in der Fehlerroutine den Klartext des Fehlers zu bekommen (s. ERR$-Funkton).

Die Fehlerroutine sollte mit einer RESUME-Anweisung (s. dort) verlassen werden.

Es ist nicht möglich, einen Fehler, der innerhalb einer Fehlerroutine auftritt, mit TRAP abzufangen.

Geben Sie die Anweisung TRAP ohne Parameter, dann wird die Fehlerfalle inaktiv.

Beispiele:

(siehe unter RESUME)

TROFF **Befehl** 7

Schreibweise:

TROFF

Zweck:

Schaltet die Ablaufüberwachung aus.

Kommentar:

Näheres finden Sie bei der TRON-Anweisung.

TRON

Anweisung 7

Schreibweise:

TRON

Zweck:

Schaltet die Ablaufüberwachung ein.

Kommentar:

Wenn die Ablaufüberwachung (Trace) eingeschaltet ist, dann wird bei Ausführung eines Programms die Zeilennummer jeder durchlaufenen BASIC-Anweisung (in eckigen Klammern) auf dem Bildschirm ausgegeben. Dies kann bei der Fehlersuche nützlich sein.

Die Ablaufüberwachung wird mit TROFF wieder ausgeschaltet.

Beide Befehle können auch in einem Programm stehen.

USR — Funktion — 2,7

Schreibweise:

```
USR(ausdr)
```

Zweck:

Aufruf eines Maschinensprachen-Programms mit Parameter-Übergabe.

Kommentar:

Beim Aufruf von USR wird das Maschinenprogramm angesprungen, dessen Startadresse sich in den Speicherstellen 1281 und 1282 befindet. Der Wert des Parameters "ausdr" wird an das Maschinenprogramm übergeben. Das Maschinenprogramm kann ebenfalls einen Wert übergeben, der zum Wert der USR-Version wird. Diese Möglichkeit der Parameterübergabe existiert im 64er-Modus für die SYS-Funktion nicht. Im 128er-Modus ist es jedoch möglich, sich mit der RREG-Anweisung (s. dort) die Registerinhalte zu besorgen.

VAL — Funktion — 2,7

Schreibweise:

VAL(str$)

Zweck:

Liefert den Zahlenwert des Strings "str$".

Kommentar:

Der Parameter "str$" sollte nur Ziffernzeichen und den Dezimalpunkt enthalten. Allerdings ist auch die Exponentialschreibweise erlaubt (s. Beispiel).

Die Funktion arbeitet den String von links nach rechts ab und hört mit der Umwandlung bei dem ersten Zeichen im String auf, das nicht mehr in einer Zahl erlaubt ist (s. Beispiel 2).

Beispiel:

```
10 SCI$ = ".13" + "E5"
20 PRINT 3 + VAL("17.4") * VAL(SCI$)

PRINT VAL("123OTTO4")
```

VERIFY **Befehl** **7**

Schreibweise:

```
VERIFY "dateiname" [Ugerätenummer]
[,verschieb]
```

Zweck:

Vergleicht ein in einer Datei auf Band oder Diskette gespeichertes Programm mit dem Programm im Hauptspeicher. Der Parameter "dateiname" muß angegeben werden.

Kommentar:

Stimmen die verglichenen Programme genau überein, so erfolgt die Meldung OK. Bei Unterschieden meldet VERIFY sich mit "VERIFY ERROR".

Hat der Parameter "verschieb" den Wert 0, so werden BASIC-Programme verglichen (Voreinstellung). Ein Wert von 1 dient zum Vergleichen von Maschinenprogrammen, die an einer anderen Startadresse beginnen (s. LOAD-Anweisung).

Wenn Sie ein Programm mit SAVE oder DSAVE abspeichern, dann einen Grafikbereich reservieren oder freigeben (s. GRAPHIC-Anweisung) und anschließend den Vergleich mit VERIFY anstellen, dann wird auch bei völlig identischen Programmen die Meldung "VERIFY ERROR" ausgegeben. Dies kommt daher, daß durch das Freigeben oder Zuweisen von Platz für den Grafikspeicher das BASIC-Programm im Hauptspeicher verschoben wird. Somit stimmen seine Adreßbezüge nicht mehr mit der auf der Diskette gespeicherten Version überein.

Wenn Sie keine "gerätenummer" angeben, dann wird automatisch eine Banddatei verifiziert. Mit dem Befehl VERIFY "*",8 wird automatisch das zuletzt auf einer Diskette abgespeicherte Programm mit dem Arbeitsspeicher verglichen (siehe auch DVERIFY).

Auf der Datasette ist VERIFY auch nützlich, um das Band zu positionieren, ohne daß ein bereits vorhandenes Programm überschrieben wird. Ehe Sie ein neues Programm (mit SAVE) sichern, geben Sie VERIFY. Dann vergleicht Ihr

Computer das Programm im Speicher mit dem auf Band, und zwar bis zum Ende der Banddatei. Dann erst meldet VERIFY, daß die Programme nicht übereinstimmen. Sie können jetzt aber sicher sein, daß das Band hinter das alte Programm positioniert ist und Sie dieses mit SAVE nicht irrtümlich löschen können.

VOL — Anweisung — 7

Schreibweise:

```
VOL n
```

Zweck:

Setzt die Lautstärke für die Tonwiedergabe.

Kommentar:

"n" darf ein ganzzahliger Wert zwischen 0 und 15 sein. Ein Wert von 0 läßt den Lautsprecher verstummen (Ausschalten der Tonausgabe). Ansonsten gilt: je größer der Wert, desto lauter.

Beispiele:

```
VOL 0
```
(Schaltet die Tonausgabe aus)

```
VOL 15
```
(Maximale Lautstärke)

WAIT — Anweisung — 2,7

Schreibweise:

```
WAIT adresse, maske_1 [,maske_2]
```

Zweck:

Anhalten der Programmausführung, bis eine Speicherstelle einen bestimmten Wert annimmt.

Kommentar:

Mit dieser Anweisung wird das Programm solange angehalten, bis sich in der angegebenen Speicherstelle (Parameter "adresse") ein bestimmtes Bitmuster einstellt. Damit kann man also die Programmausführung unterbrechen und in Abhängigkeit von einem äußeren Ereignis wieder fortsetzen. Dies kann bei der Programmierung peripherer Geräte von Nutzen sein.

Ob sich das gewünschte Bitmuster eingestellt hat können Sie mit den beiden Parametern "maske_1" und "maske_2" überprüfen. Beide dürfen Werte zwischen 0 und 255 annehmen; diese werden von WAIT als Bitmuster interpretiert. WAIT verknüpft den Wert von "maske_1" mit dem Wert in der angegebenen Adresse. Ist außerdem noch "maske_2" spezifiziert, so wird das Verknüpfungsergebnis auch noch über XOR mit "maske_2" verknüpft.

Man kann mit WAIT sehr leicht eine Endlosschleife erzeugen. Diese kann dann nur mehr mit einer "Notbremse" verlassen werden: drücken Sie die RUN/STOP-Taste zusammen mit RESTORE. Das Programm im Speicher bleibt dadurch erhalten.

WIDTH — Anweisung — 7

Schreibweise:

```
WIDTH n
```

Zweck:

Setzt die Strichstärke für Zeichnungen auf dem Grafikbildschirm.

Kommentar:

Bei "n" = 1 wird mit einfacher, bei "n" = 2 mit doppelter Strichstärke gezeichnet.

Beispiel:

```
10 GRAPHIC 1,1
20 WIDTH 1 : CIRCLE ,160,100,80
30 WIDTH 2 : CIRCLE ,160,100,50
```

WINDOW — Anweisung — 7

Schreibweise:

```
WINDOW x1, y1, x2, y2 [, lösch]
```

Zweck:

Definiert ein Bildschirm-Fenster für die Textausgabe.

Kommentar:

Die beiden Parameter "x1" und "y1" definieren die linke obere, die Parameter "x2" und "y2" die rechte untere Ecke. Die x-Werte können von 0 bis 39 oder 79 liegen, je nachdem, ob das Fenster für den 40- oder 80-Zeichen-Bildschirm vereinbart wird.

Ist der Parameter "lösch" auf 1 gesetzt, so wird das Fenster nach dem Einrichten auch noch gelöscht.

Die ESCAPE-Befehle des Bildschirmeditors beziehen sich auf das aktuelle Fenster; ebenso die Tasten zur Cursor-Steuerung (Pfeiltasten), alle Ausgabebefehle und die anderen bildschirmorientierten Anweisungen (z.B. SCNCLR).

Beispiele:

```
100 WINDOW 10, 10, 15, 20, 1
```
(Definiert ein 5 Spalten breites Fenster mit 10 Zeilen und löscht es).

XOR — Funktion — 2,7

Schreibweise:

```
XOR (ausdr1, ausdr2)
```

Zweck:

Führt die Exklusive-Oder-Verknüpfung mit "ausdr1" und "ausdr2" durch.

Kommentar:

Wenn "ausdr1" und "ausdr2" wahrheitswertig sind (also z.B. Vergleichsbedingungen), dann wird der mit XOR verknüpfte Ausdruck nur wahr, wenn ihre Wahrheitswerte verschieden sind. Handelt es sich bei "ausdr1" und "ausdr2" um Zahlenausdrücke, dann erfolgt eine bitweise XOR-Verknüpfung.

Das Beispiel zeigt, wie mit XOR der Inhalt zweier numerischer Variable ohne Hilfe einer dritten ausgetauscht werden kann(!).

Beispiel:

```
10 A = 15 : B = 17
15 PRINT "A= ";A;"B= ";B
20 A = XOR(A, B)
30 B = XOR(A, B)
40 A = XOR(A, B)
50 PRINT "A= ";A;"B= ";B
```

A.2 Systemvariablen

Die folgenden Variablen haben in BASIC 7.0 feste Bedeutung; ihre Namen sind reserviert und können nicht vom Benutzer verwendet werden.

DS Enthält den aktuellen Status der Floppy; dies geschieht durch Lesen des Befehlskanals. Eine Erläuterung der Fehler finden Sie im Anhang A.3.

DS$ Eine Klartextmeldung über den Status der Floppy (s. DS). Sie können mit diesen beiden Variablen herausfinden, ob Diskettenoperationen erfolgreich waren und im Fehlerfall die Ursache erfahren. Eine Erläuterung der Fehler finden Sie im Anhang A.3.

ER Nützlich bei Fehlerfallen (s. TRAP-Anweisung); enthält die Nummer des letzten Fehlers, der in einem BASIC-Programm auftrat. Mit der Funktion ERR$ (s. dort) können Sie sich die Klartextmeldung des Fehlers besorgen. Eine Erläuterung der Fehler finden Sie im Anhang A.3.

EL Nützlich bei Fehlerfallen (s. TRAP-Anweisung); enthält die Zeile des BASIC-Programms, in der der letzte Fehler auftrat.

ST Statusvariable für Ein- und Ausgabeoperationen (aber nicht für normale Bildschirmausgabe und Tastatureingabe). Wenn ST den Wert 0 hat, dann war die letzte Ein-/Ausgabeoperation erfolgreich.

TI Die interne Systemuhr; dies ist eine Softwareuhr, die alle sechzigstel Sekunden hochgezählt wird. Beim Einschalten des C128 wird die Uhr auf den Wert 0 gesetzt und von da an kontinuierlich hochgezählt. Der Benutzer kann die Uhr stellen, indem er den Wert der Systemvariablen TI$ verändert.

TI$ Stringdarstellung der Systemuhr. Die Zählvariable TI wird als 24-Stunden-Uhr dargestellt. Die ersten beiden Ziffern des Strings sind die Stunden, die nächsten beiden die Minuten und die letzten beiden die Sekunden. Wenn Sie die Systemuhr stellen wollen, dann müssen Sie dieser Variablen einen neuen Wert zuweisen, der als gültige Zeitangabe interpretiert werden kann.

In der Systemuhr ist übrigens - trotz gegenteiliger Behauptung in der Dokumentation - ein Fehler: eine Sekunde vor Mitternacht (also "235959") springt sie um auf 24 Uhr ("240000") und danach auf Null Uhr und 1 Sekunde ("000001"). Ihr 128er kennt also eine 25. Stunde am Tage, die allerdings nur eine Sekunde lang dauert!

A.3 Fehlermeldungen

Die folgenden Fehlermeldung werden von BASIC ausgegeben. Hinter der Fehlermeldung finden Sie in Klammern die Fehlernummer (für die ERR$-Funktion; s. dort) und eine Erläuterung des Fehlers.

TOO MANY FILES (1):
Die Anzahl der Dateien, die gleichzeitig geöffnet sein können, ist auf 10 beschränkt. Sie haben also versucht, mit OPEN oder DOPEN eine weitere Datei zu öffnen,

FILE OPEN (2):
Sie haben versucht, eine Datei zu öffnen und wollten dieser eine bereits vergebene logische Dateinummer zuteilen.

FILE NOT OPEN (3):
Sie haben in einer Ein-/Ausgabeanweisung eine logische Dateinummer verwendet, zu der keine geöffnete Datei existiert.

FILE NOT FOUND (4):
Sie haben versucht, eine Datei zu laden, die auf der Diskette oder der Kassette nicht vorhanden ist. Bei Kassettenoperationen kann es auch sein, daß Sie das Band zu weit vorgespult haben.

DEVICE NOT PRESENT (5):
Sie haben versucht, ein nicht vorhandenes Gerät anzusprechen (z.B. den DIRECTORY-Befehl gegeben, ohne daß die Floppy eingeschaltet ist).

NOT INPUT FILE (6):
Sie haben versucht, Daten mittels GET# oder INPUT# von einer Datei zu lesen, die Sie nur zum Beschreiben eröffnet haben.

NOT OUTPUT FILE (7):
Sie haben versucht, Daten mittels PRINT# in eine Datei zu schreiben, die Sie nur zum Lesen eröffnet haben.

MISSING FILE NAME (8):
Im Kommando fehlt der Dateiname.

ILLEGAL DEVICE NUMBER (9):
Sie wollten mit einem Gerät etwas Verbotenes anstellen (z.B. von einem Drucker lesen, oder SAVE auf den Bildschirm).

NEXT WITHOUT FOR (10):
Entweder haben Sie FOR-Schleifen nicht richtig verschachtelt, oder der Variablenname hinter NEXT stimmt nicht mit dem hinter FOR überein (s. auch das Beispiel unter RESUME für eine weitere Möglichkeit).

SYNTAX (11):
Ihnen ist beim Schreiben einer BASIC-Anweisung ein formaler Fehler unterlaufen. Sie haben vielleicht zu viele oder zu wenige Klammern gesetzt, ein Befehlswort falsch geschrieben...

RETURN WITHOUT GOSUB (12):
BASIC ist auf eine RETURN-Anweisung gestoßen, ohne daß zuvor mit GOSUB ein Unterprogramm-Aufruf erfolgt. Sowas passiert z.B. dann, wenn man mit GOTO "seitlich" in ein Unterprogramm einsteigt.

OUT OF DATA (13):
Eine READ-Anweisung verlangt mehr Daten, als ihr die zugehörige DATA-Anweisung bieten kann.

ILLEGAL QUANTITY (14):
Ein Zahlenwert als Parameter für eine Funktion oder Anweisung befindet sich außerhalb des erlaubten Wertebereichs. Den Fehler bekommen Sie z.B. dann, wenn Sie mit COLOR eine siebzehnte Farbe auswählen wollen (gibt nur 16!).

OVERFLOW (15):
Das Ergebnis einer Berechnung übersteigt die größte darstellbare Zahl (immerhin 1.701411833E+38)!

OUT OF MEMORY (16):
Entweder braucht Ihr Programm oder die Variablen Ihres Programms zu viel Platz (würde mich interessieren, wie Sie das bei 128K machen!), oder im Programm sind zu viele noch nicht abgeschlossene (aktive) DO-, FOR- und GOSUB-Anweisungen.

UNDEF'D STATEMENT (17):
Sie beziehen sich im Programm (z.B: in GOTO, GOSUB oder ON GOTO) auf eine Zeilennummer, die es in Ihrem Programm gar nicht gibt.

BAD SUBSCRIPT (18):
Das Programm versuchte, ein Feldelement zu erreichen, das die Dimensionen (s. DIM-Anweisung) sprengte. Passiert, wenn Sie z.B. auf das hundertste Element eines 99-elementigen Feldes zugreifen wollen.

REDIM'D ARRAY (19):
Sie haben versucht, denselben Array (Feld) im Programm ein zweites Mal zu dimensionieren (mittels DIM); das geht nur einmal.

DIVISION BY ZERO (20):
Schulmathematik: durch Null darf man nicht dividieren!

ILLEGAL DIRECT (21):
Der Befehl geht nicht im Direktmodus (z.B. INPUT und GET).

TYPE MISMATCH (22):
Sie haben da eine Zahl verwendet, wo der Rechner einen String erwartete und umgekehrt.

STRING TOO LONG (23):
Ihr String ist zu lang; mehr als 255 Zeichen sind nicht drin.

FILE DATA (24):
Die vom Band gelesenen Daten sind nicht in Ordnung (Aufzeichnungsfehler?).

FORMULA TOO COMPLEX (25):
Ein arithmetischer Ausdruck ist zu kompliziert. Vereinfachen Sie ihn (z.B. über mehrere Zeilen aufteilen oder weniger Klammern verwenden).

CAN'T CONTINUE (26):
CONT (s. dort) geht nur, wenn zuvor RUN erfolgte, oder es trat ein Fehler auf, oder Sie haben das Programm ediert.

UNDEFINED FUNCTION (27):
Sie möchten eine Funktion anwenden, die Sie nicht definiert haben (s. DEF und FN).

VERIFY (28):
Fehler beim Überprüfen mit VERIFY oder DVERIFY (s. dort)

LOAD (29):
Ladeproblem; probieren Sie's noch mal.

BREAK (30):
Sie haben die STOP-Taste betätigt.

CANT'T RESUME (31):
RESUME (s. dort) geht nur, wenn die Fehlerfalle aktiviert ist (s. TRAP-Anweisung).

LOOP NOT FOUND (32):
BASIC kann das zum DO (s. dort) gehörige LOOP nicht finden.

LOOP WITHOUT DO (33):
BASIC ist auf ein LOOP gestoßen, ohne daß mit DO (s. dort) eine Schleife aktiviert wurde.

DIRECT MODE ONLY (34):
Der Befehl ist nur im Direktmodus und nicht innerhalb eines Programms erlaubt.

NO GRAPHICS AREA (35):
Zeichenkomandos (BOX, DRAW, CIRCLE...) setzen voraus, daß zuvor mit GRAPHICS (s. dort) Speicherbereich reserviert wurde.

BAD DISK (36):
Diskette kann nicht formatiert werden (s. HEADER), weil Sie keine Identifikationsnummer (bei fabrikneuen Disketten) vergeben haben oder weil die Diskette beschädigt ist.

Die folgenden Fehlernummern und -meldungen werden über die Variablen DS und DS$ zurückgegeben. Es sind Fehlermeldungen des Betriebssystems.

READ ERROR (block header not found) (20):
Der Floppy-Controller kann den Header des fraglichen Datenblocks nicht finden. Entweder hat er eine falsche Sektornummer erhalten, oder der Header ist zerstört.

READ ERROR (no sync character) (21):
Der Controller kann auf der angewählten Spur kein Synchronisationszeichen finden. Entweder ist der Lese-/Schreibkopf falsch justiert, keine Diskette eingelegt, die Diskette nicht formatiert oder nicht richtig eingelegt. Kann aber auch auf einen Hardware-Fehler zurückgehen.

READ ERROR (data block not present) (22):
Ein nicht richtig aufgezeichneter Datenblock sollte gelesen oder verifiziert werden. Tritt auf, wenn nicht zulässige Spuren- oder Sektorenangaben an den Controller erfolgten.

READ ERROR (checksum error in data block) (23):
Prüfsummenfehler in einem oder mehreren Datenbytes auf der Diskette. Die Daten wurde zwar in den DOS-Puffer übertragen, aber die Prüfsumme über den Daten ist verkehrt. Kann auf einen Aufzeichungsfehler, aber auch auf Erdungsprobleme hinweisen.

READ ERROR (byte decoding error) (24):
Die Daten wurde zwar in den DOS-Puffer übertragen, aber eines der Datenbytes weist ein unzulässiges Bitmuster auf. Kann auch auf Erdungsprobleme hinweisen.

WRITE ERROR (write-verify error) (25):
Der Controller hat Unterschiede zwischen den auf Diskette geschriebenen und den im DOS-Puffer enthaltenen Daten entdeckt.

WRITE PROTECT ON (26):
Schreibgeschütztes Medium kann nicht beschrieben werden.

READ ERROR (27):
Prüfsummenfehler im Header.

WRITE ERROR (28):
Datenblock ist zu lang

DISK ID MISMATCH (29):
Der Controller soll auf eine nicht initialisierte Diskette zugreifen oder der Header ist nicht in Ordnung.

SYNTAX ERROR (allgemein) (30):
Ein über den Kommandokanal erteilter Befehl kann vom DOS nicht verstanden werden. Bei der Befehlsbildung ist ein formaler Fehler unterlaufen.

SYNTAX ERROR (unzulässiges Kommando) (31):
DOS erkennt das Kommando nicht. Das Kommando muß in der ersten Spalte beginnen.

SYNTAX ERROR (unzulässiges Kommando) (32):
Kommandos dürfen nicht länger als 58 Zeichen sein.

SYNTAX ERROR (ungültiger Dateiname) (33):
Unzulässige Verwendung von Jokerzeichen ("*" und "?") bei OPEN oder SAVE.

SYNTAX ERROR (Dateiname fehlt) (34):
Dateiname entweder nicht angegeben oder von DOS nicht erkennbar.

SYNTAX ERROR (ungültiges Kommando) (39):
Das über den Befehlskanal gesandte Kommando (Sekundäradresse 15) ist unzulässig.

RECORD NOT PRESENT (50):
Versuch, hinter dem letzten Satz einer Datei zu lesen. Falls eine Datei (mittels PRINT#) erweitert werden soll, dann kann dieser Fehler ignoriert werden. INPUT# oder GET sollten jedoch im Anschluß an diesen Fehler erst nach erneutem Positionieren aufgerufen werden.

OVERFLOW IN RECORD (51):
PRINT# versucht, mehr auszugeben, als in einen Record paßt; Informationen werden abgeschnitten. Beachten Sie, daß CHR$(13) zur Information mitzählt.

FILE TOO LARGE (52):
Die Satzposition in einer relativen Datei zeigt an, daß Überlauf auf der Diskette auftritt.

WRITE FILE OPEN (60):
Der Fehler tritt auf, wenn eine Datei mit Schreibzugriff nicht geschlossen wurde und nun zum Lesen geöffnet werden soll.

FILE NOT OPEN (61):
Sie versuchen, auf eine Datei zuzugreifen, die nicht mit OPEN geöffnet wurde.

FILE NOT FOUND (62):
Die gesuchte Datei ist nicht auf der Diskette.

FILE EXISTS (63):
Eine Datei gleichen Namens existiert bereits auf der Diskette.

FILE TYPE MISMATCH (64):
Der Dateityp stimmt nicht.

NO BLOCK (65):
Kein Datenblock.

ILLEGAL TRACK AND SECTOR (66):
DOS hat versucht, einen Sektor oder Block zu erreichen, der im aktuellen Format nicht existiert.

ILLEGAL SYSTEM T OR S (67):
Systemspur oder -sektor nicht zulässig.

NO CHANNEL (70):
Der verlangte Kanal exisitiert nicht, oder alle Kanäle sind bereits belegt. Sie können höchstens 5 sequentielle Dateien oder 6 Relativdateien zugleich öffnen (aber nicht mehr als 10 insgesamt!).

DIRECTORY ERROR (71):
Fehler im Inhaltsverzeichnis.

DISK FULL (72):
Entweder sind alle Datenblöcke belegt, oder es ist kein Platz mehr im Inhaltsverzeichnis.

DOS MISMATCH (73):
Die Betriebssystem- (DOS-)Versionen 1 und 2 sind lesekompatibel, aber nicht schreibkompatibel. Daten können zum Lesen problemlos zwischen beiden Versionen ausgetauscht werden. Eine in der Version 2 formatierte Diskette kann aber nicht von der Version 1 beschrieben werden, da sich die Formate unterscheiden.

DRIVE NOT READY (74):
Sie haben keine Diskette ins Laufwerk eingelegt!

Anhang B: Bildschirm und Zeichencodes

Der folgende Anhang beschreibt den amerikanischen Zeichencode; dieser ist im 64er- und 128er-Modus gleich. Die Beschreibung hat folgenden Aufbau:

1. In der linken Ecke der Beschreibung steht die laufende Nummer, welche zugleich den Bildschirmcode darstellt, also diejenige (dezimale) Zahl, die Sie in den Textspeicher schreiben müssen, um das jeweilige Zeichen auf den Bildschirm zu bekommen.

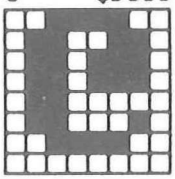
2. Rechts daneben finden Sie die Startadresse innerhalb des Zeichengenerators. Die Angaben sind in Hexadezimalschreibweise. Im 128er-Modus können Sie mit der Konfiguration E im Monitor diese Definition betrachten. Mit dem Befehl "M ED480" sehen Sie also z.B. die Definition des inversen "P".

3. Darunter befindet sich die vergrößerte 8x8-Punkte-Matrix der Zeichendefinition.

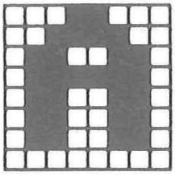
4. Der Anhang umfaßt folgende Zeichensätze:

Zeichensatz		laufende Nummer
a	Großbuchstaben und Grafikzeichen:	000 bis 127
b	wie (a), aber invers:	128 bis 255
c	Groß- und Kleinbuchstaben:	256 bis 383
d	wie (c), aber invers:	384 bis 511

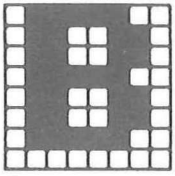
0 $D000



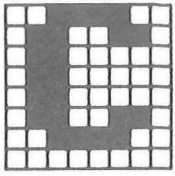
1 $D008



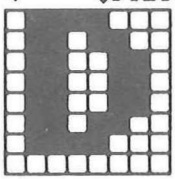
2 $D010



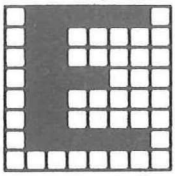
3 $D018



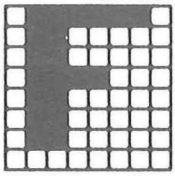
4 $D020



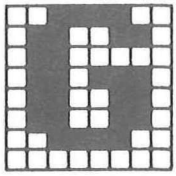
5 $D028



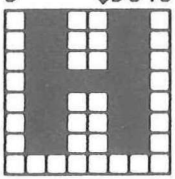
6 $D030



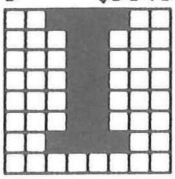
7 $D038



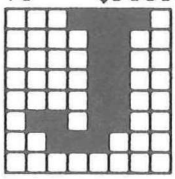
8 $D040



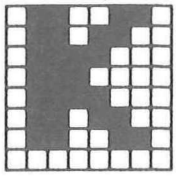
9 $D048



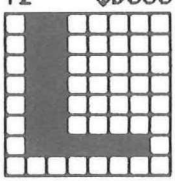
10 $D050



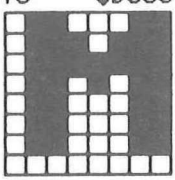
11 $D058



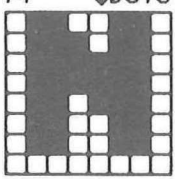
12 $D060



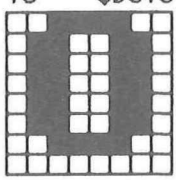
13 $D068



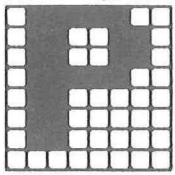
14 $D070



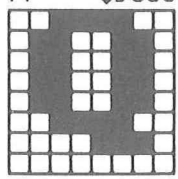
15 $D078



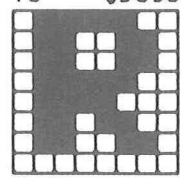
16 $D080



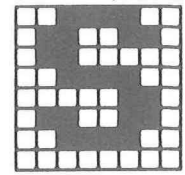
17 $D088



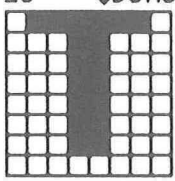
18 $D090



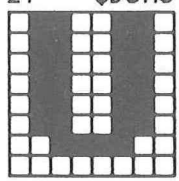
19 $D098



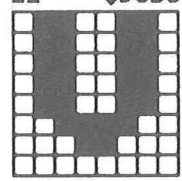
20 $D0A0



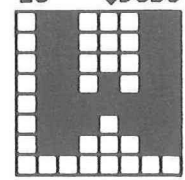
21 $D0A8



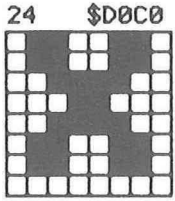
22 $D0B0



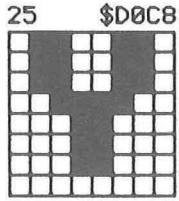
23 $D0B8



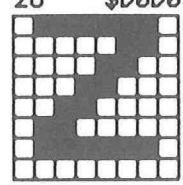
24 $D0C0



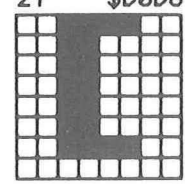
25 $D0C8



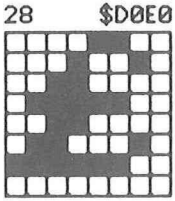
26 $D0D0



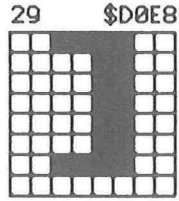
27 $D0D8



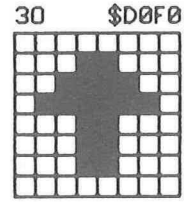
28 $D0E0



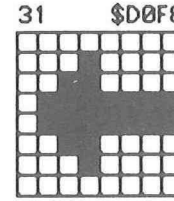
29 $D0E8



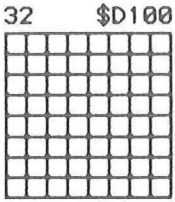
30 $D0F0



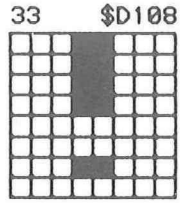
31 $D0F8



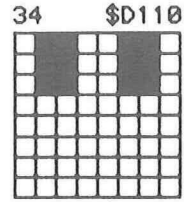
32 $D100



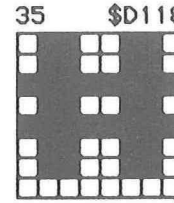
33 $D108



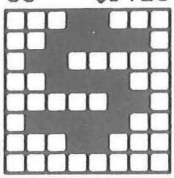
34 $D110



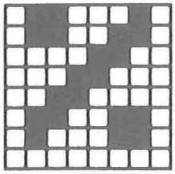
35 $D118



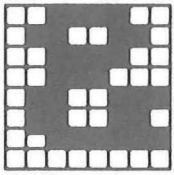
36 $D120



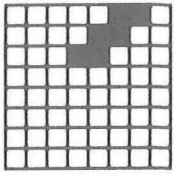
37 $D128



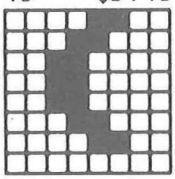
38 $D130



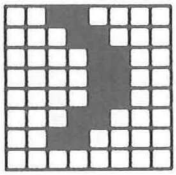
39 $D138



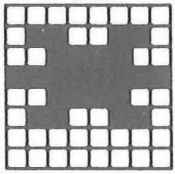
40 $D140



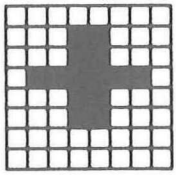
41 $D148



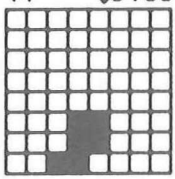
42 $D150



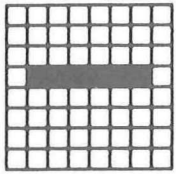
43 $D158



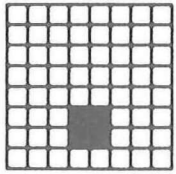
44 $D160



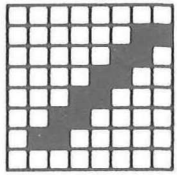
45 $D168



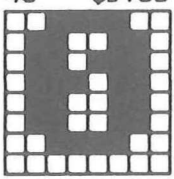
46 $D170



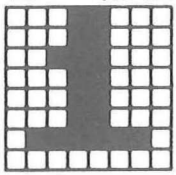
47 $D178



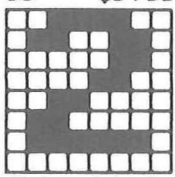
48 $D180



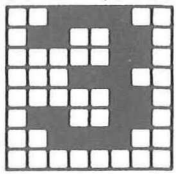
49 $D188



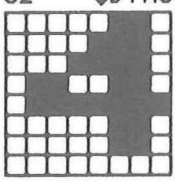
50 $D190



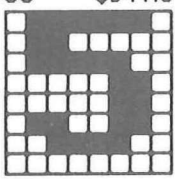
51 $D198



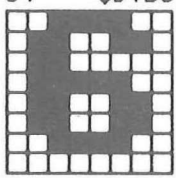
52 $D1A0



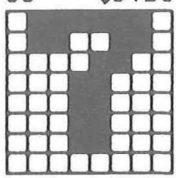
53 $D1A8



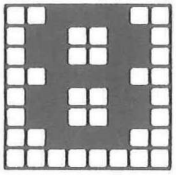
54 $D1B0



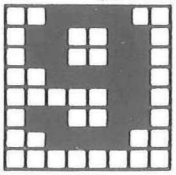
55 $D1B8



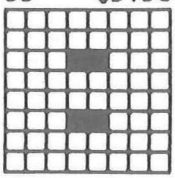
56 $D1C0



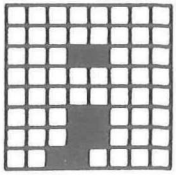
57 $D1C8



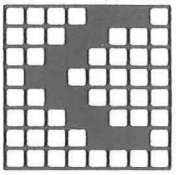
58 $D1D0



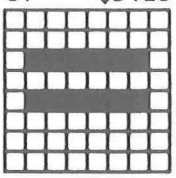
59 $D1D8



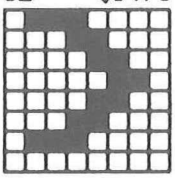
60 $D1E0



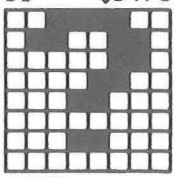
61 $D1E8



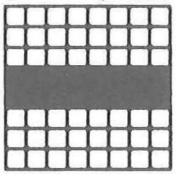
62 $D1F0



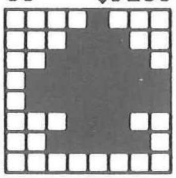
63 $D1F8



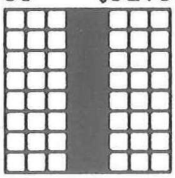
64 $D200



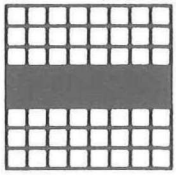
65 $D208



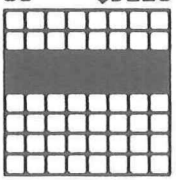
66 $D210



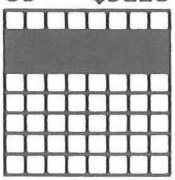
67 $D218



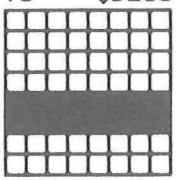
68 $D220



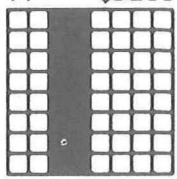
69 $D228



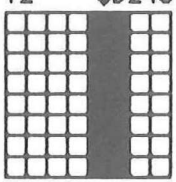
70 $D230



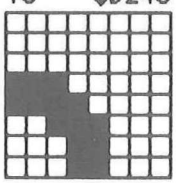
71 $D238



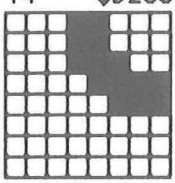
72 $D240



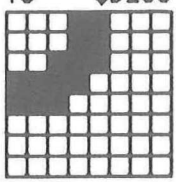
73 $D248



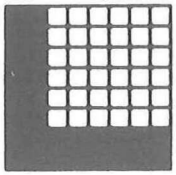
74 $D250



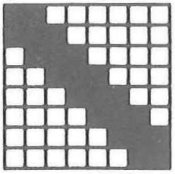
75 $D258



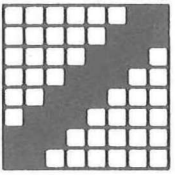
76 $D260



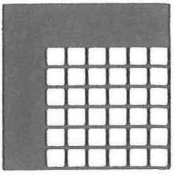
77 $D268



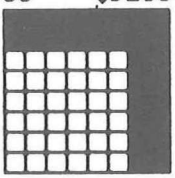
78 $D270



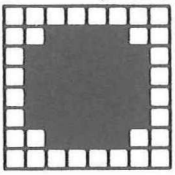
79 $D278



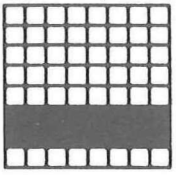
80 $D280



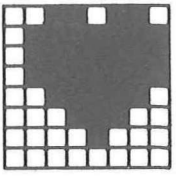
81 $D288



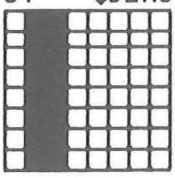
82 $D290



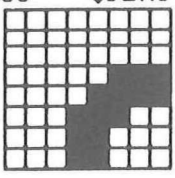
83 $D298



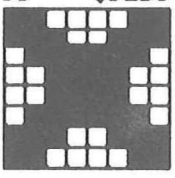
84 $D2A0



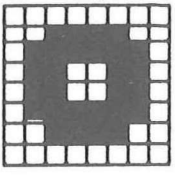
85 $D2A8



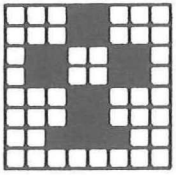
86 $D2B0



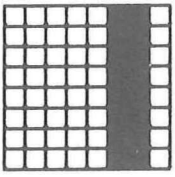
87 $D2B8



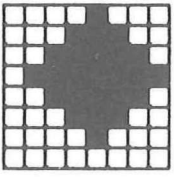
88 $D2C0



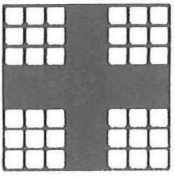
89 $D2C8



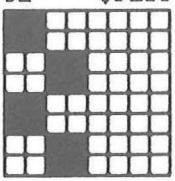
90 $D2D0



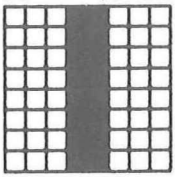
91 $D2D8



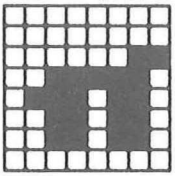
92 $D2E0



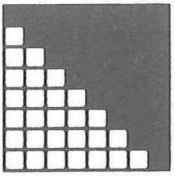
93 $D2E8



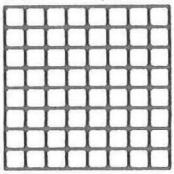
94 $D2F0



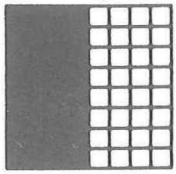
95 $D2F8



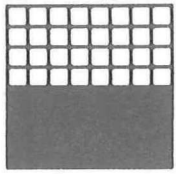
96 $D300



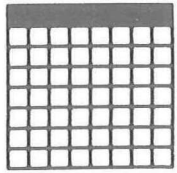
97 $D308



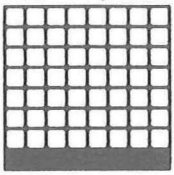
98 $D310



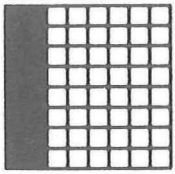
99 $D318



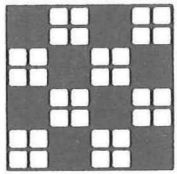
100 $D320



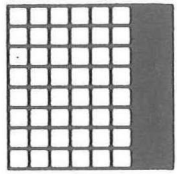
101 $D328



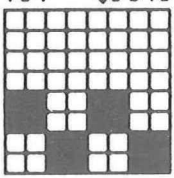
102 $D330



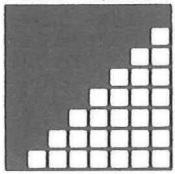
103 $D338



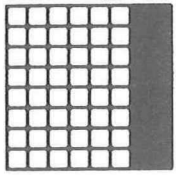
104 $D340



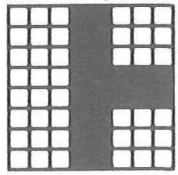
105 $D348



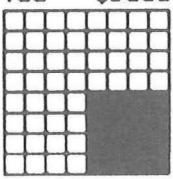
106 $D350



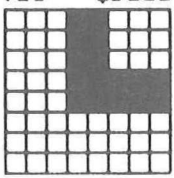
107 $D358



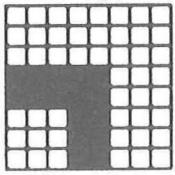
108 $D360



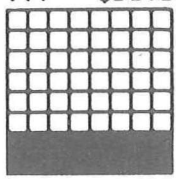
109 $D368



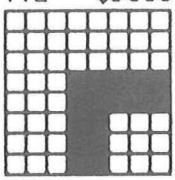
110 $D370



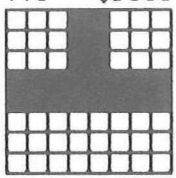
111 $D378



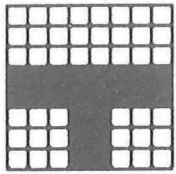
112 $D380



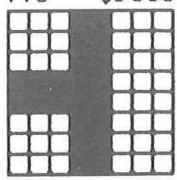
113 $D388



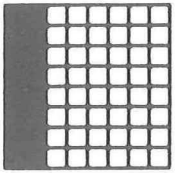
114 $D390



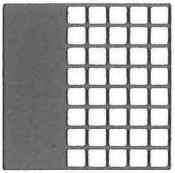
115 $D398



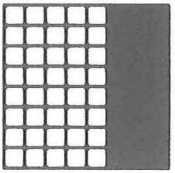
116 $D3A0



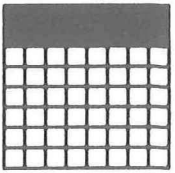
117 $D3A8



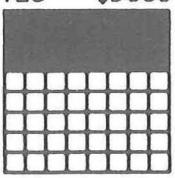
118 $D3B0



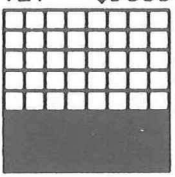
119 $D3B8



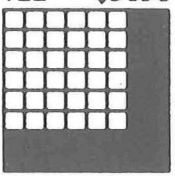
120 $D3C0



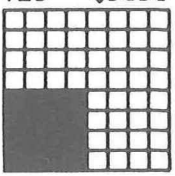
121 $D3C8



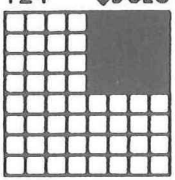
122 $D3D0



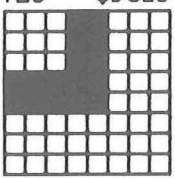
123 $D3D8



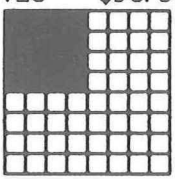
124 $D3E0



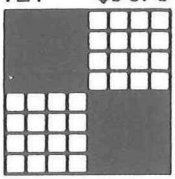
125 $D3E8



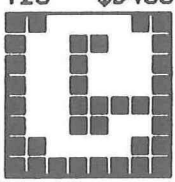
126 $D3F0



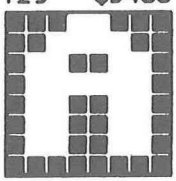
127 $D3F8



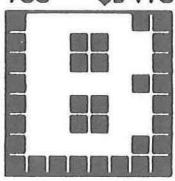
128 $D400



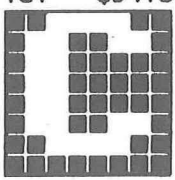
129 $D408



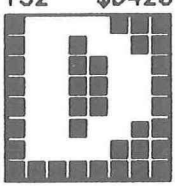
130 $D410



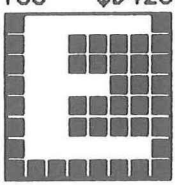
131 $D418



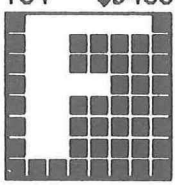
132 $D420



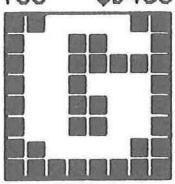
133 $D428



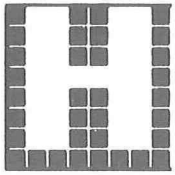
134 $D430



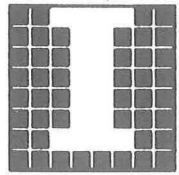
135 $D438



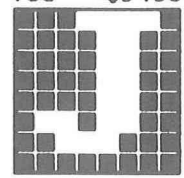
136 $D440



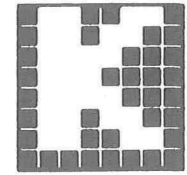
137 $D448



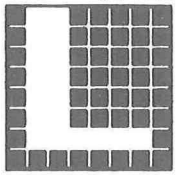
138 $D450



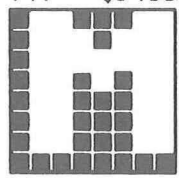
139 $D458



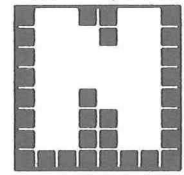
140 $D460



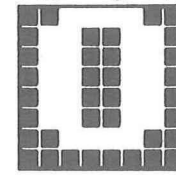
141 $D468



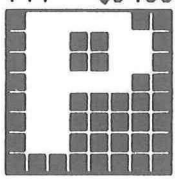
142 $D470



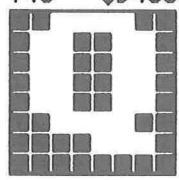
143 $D478



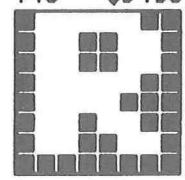
144 $D480



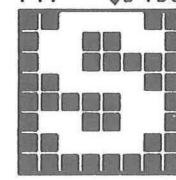
145 $D488



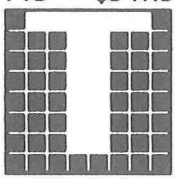
146 $D490



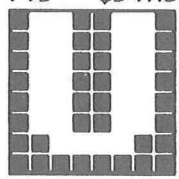
147 $D498



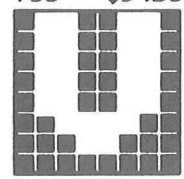
148 $D4A0



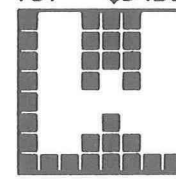
149 $D4A8



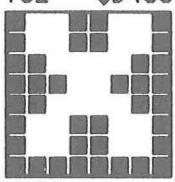
150 $D4B0



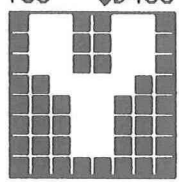
151 $D4B8



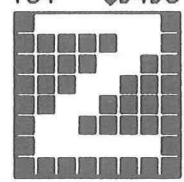
152 $D4C0



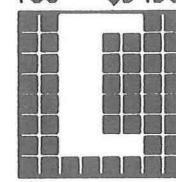
153 $D4C8



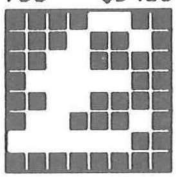
154 $D4D0



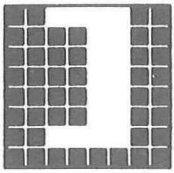
155 $D4D8



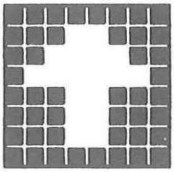
156 $D4E0



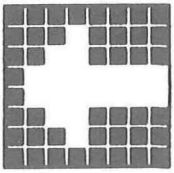
157 $D4E8



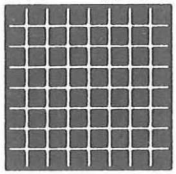
158 $D4F0



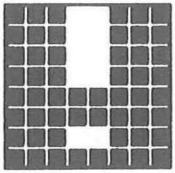
159 $D4F8



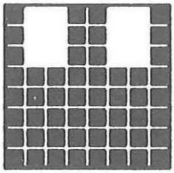
160 $D500



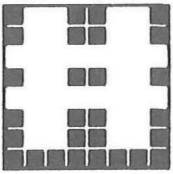
161 $D508



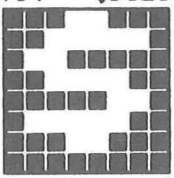
162 $D510



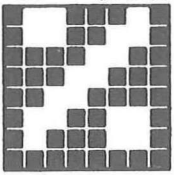
163 $D518



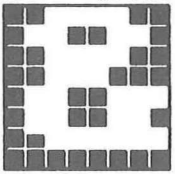
164 $D520



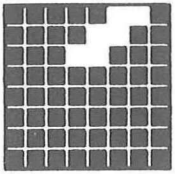
165 $D528



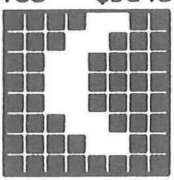
166 $D530



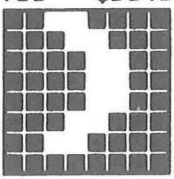
167 $D538



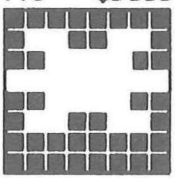
168 $D540



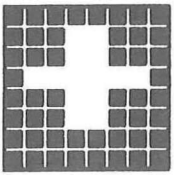
169 $D548



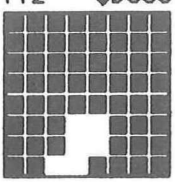
170 $D550



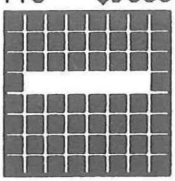
171 $D558



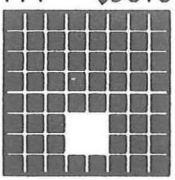
172 $D560



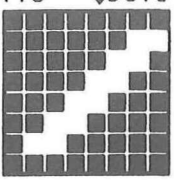
173 $D568



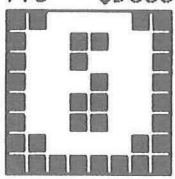
174 $D570



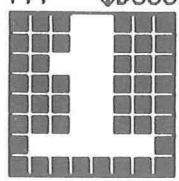
175 $D578



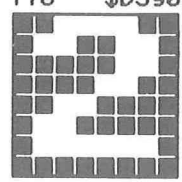
176 $D580



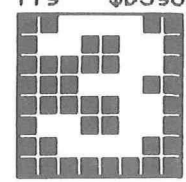
177 $D588



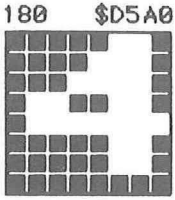
178 $D590



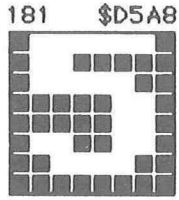
179 $D598



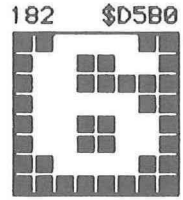
180 $D5A0



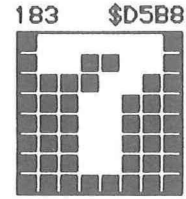
181 $D5A8



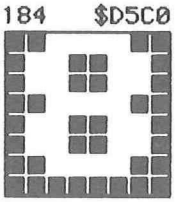
182 $D5B0



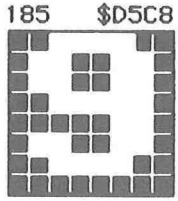
183 $D5B8



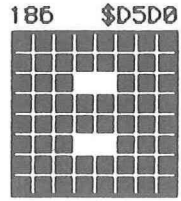
184 $D5C0



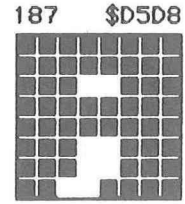
185 $D5C8



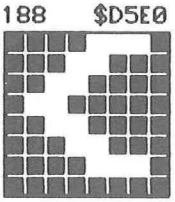
186 $D5D0



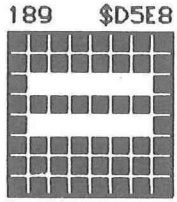
187 $D5D8



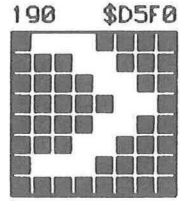
188 $D5E0



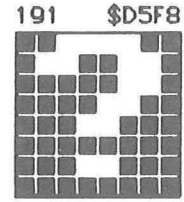
189 $D5E8



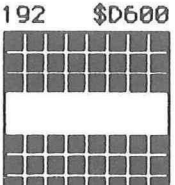
190 $D5F0



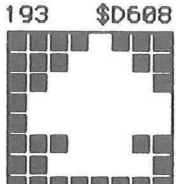
191 $D5F8



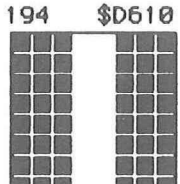
192 $D600



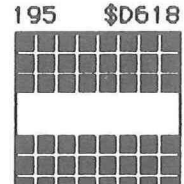
193 $D608



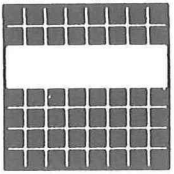
194 $D610



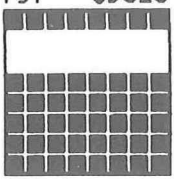
195 $D618



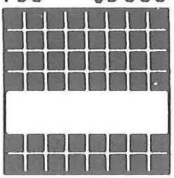
196 $D620



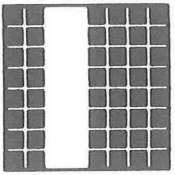
197 $D628



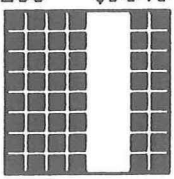
198 $D630



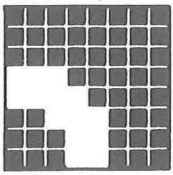
199 $D638



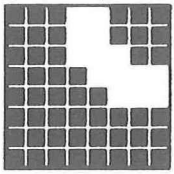
200 $D640



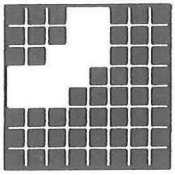
201 $D648



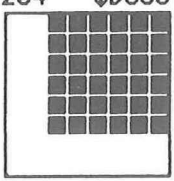
202 $D650



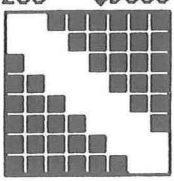
203 $D658



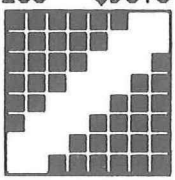
204 $D660



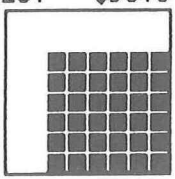
205 $D668



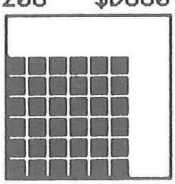
206 $D670



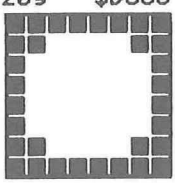
207 $D678



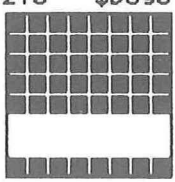
208 $D680



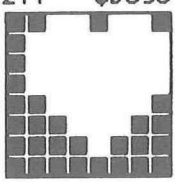
209 $D688



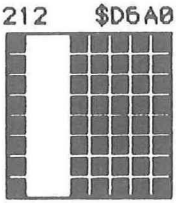
210 $D690



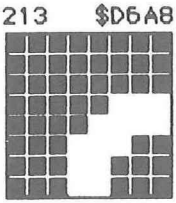
211 $D698



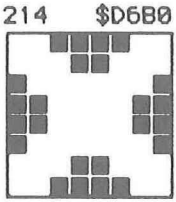
212 $D6A0



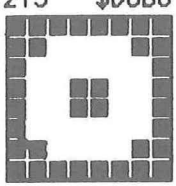
213 $D6A8



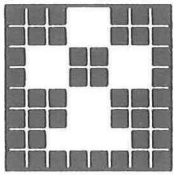
214 $D6B0



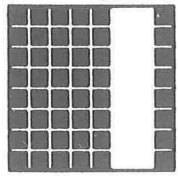
215 $D6B8



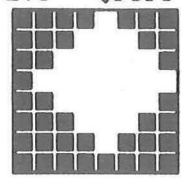
216 $D6C0



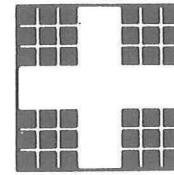
217 $D6C8



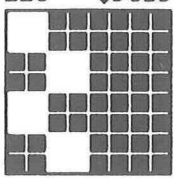
218 $D6D0



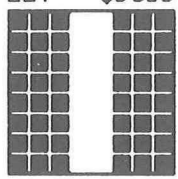
219 $D6D8



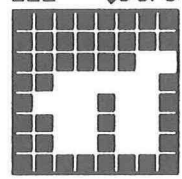
220 $D6E0



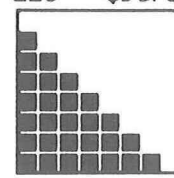
221 $D6E8



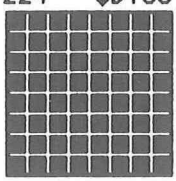
222 $D6F0



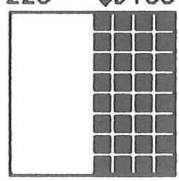
223 $D6F8



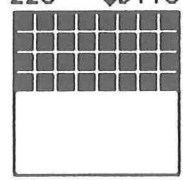
224 $D700



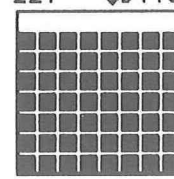
225 $D708



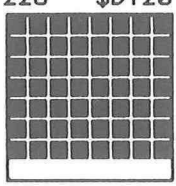
226 $D710



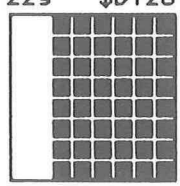
227 $D718



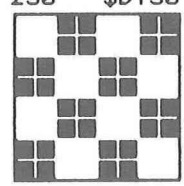
228 $D720



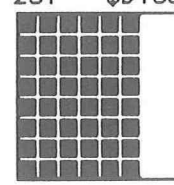
229 $D728



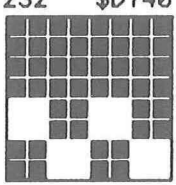
230 $D730



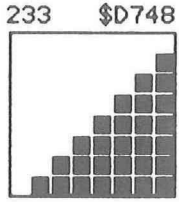
231 $D738



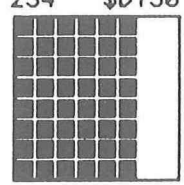
232 $D740



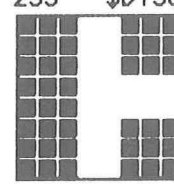
233 $D748



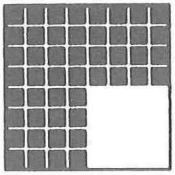
234 $D750



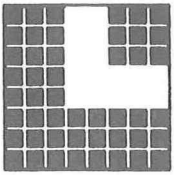
235 $D758



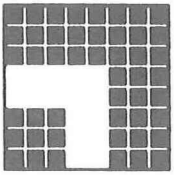
236 $D760



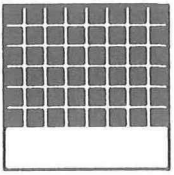
237 $D768



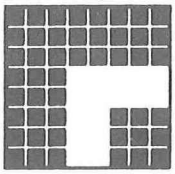
238 $D770



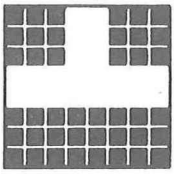
239 $D778



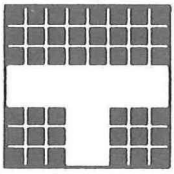
240 $D780



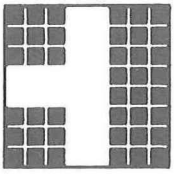
241 $D788



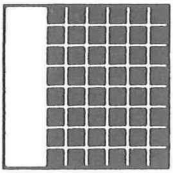
242 $D790



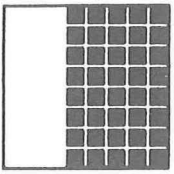
243 $D798



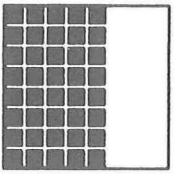
244 $D7A0



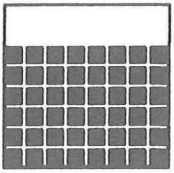
245 $D7A8



246 $D7B0

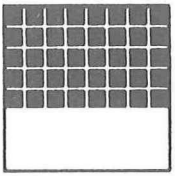


247 $D7B8

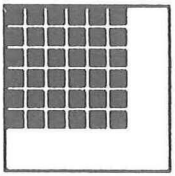


248 $D7C0

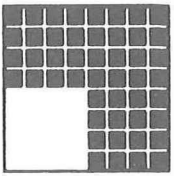
249 $D7C8



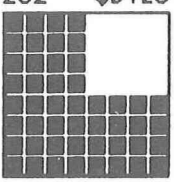
250 $D7D0



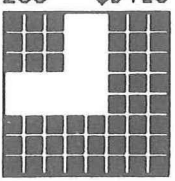
251 $D7D8



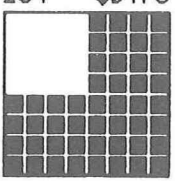
252 $D7E0



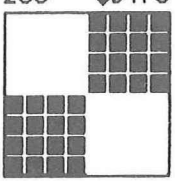
253 $D7E8



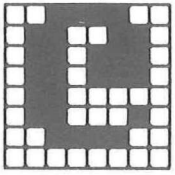
254 $D7F0



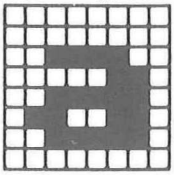
255 $D7F8



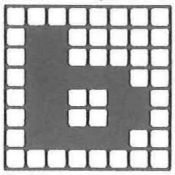
256 $D800



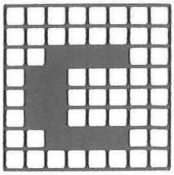
257 $D808



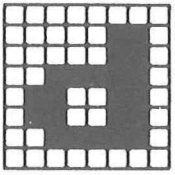
258 $D810



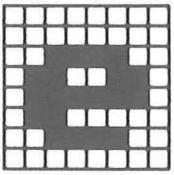
259 $D818



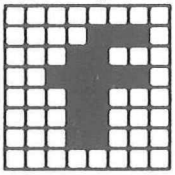
260 $D820



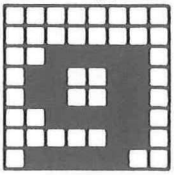
261 $D828



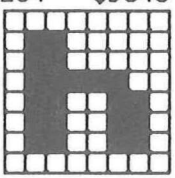
262 $D830



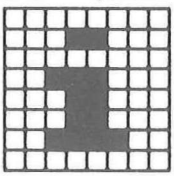
263 $D838



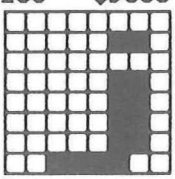
264 $D840



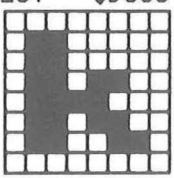
265 $D848



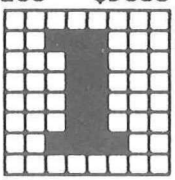
266 $D850



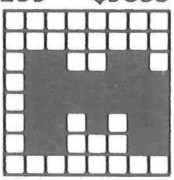
267 $D858



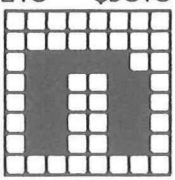
268 $D860



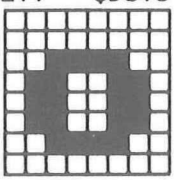
269 $D868



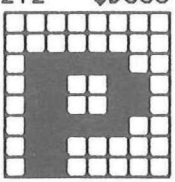
270 $D870



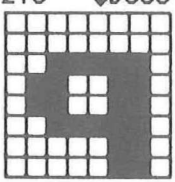
271 $D878



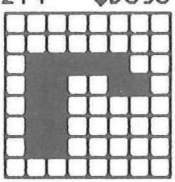
272 $D880



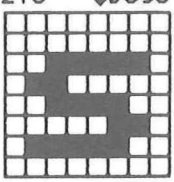
273 $D888



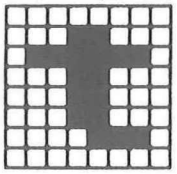
274 $D890



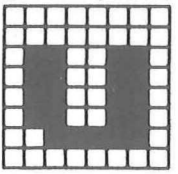
275 $D898



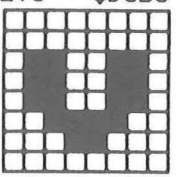
276 $D8A0



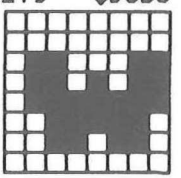
277 $D8A8



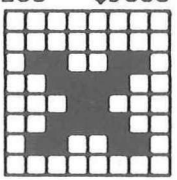
278 $D8B0



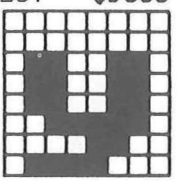
279 $D8B8



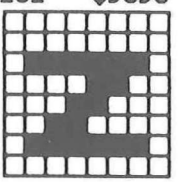
280 $D8C0



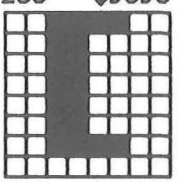
281 $D8C8



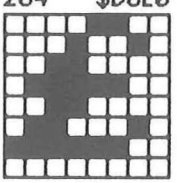
282 $D8D0



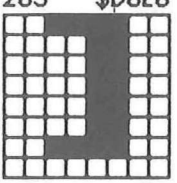
283 $D8D8



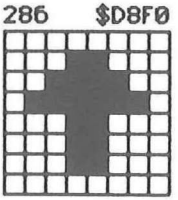
284 $D8E0



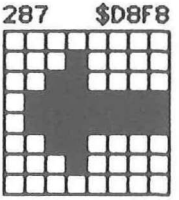
285 $D8E8



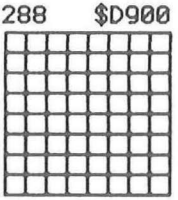
286 $D8F0



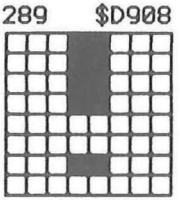
287 $D8F8



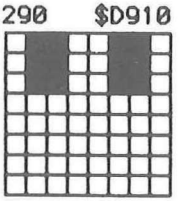
288 $D900



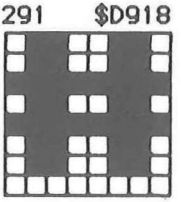
289 $D908



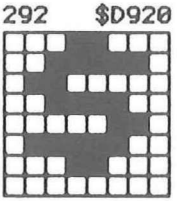
290 $D910



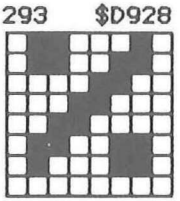
291 $D918



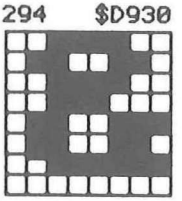
292 $D920



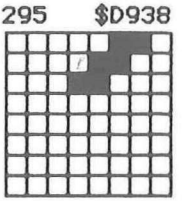
293 $D928



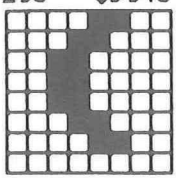
294 $D930



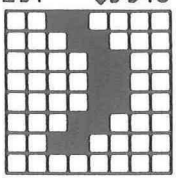
295 $D938



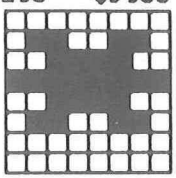
296 $D940



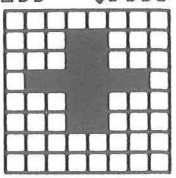
297 $D948



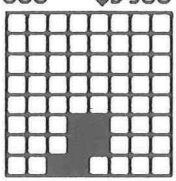
298 $D950



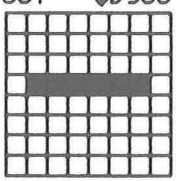
299 $D958



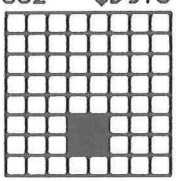
300 $D960



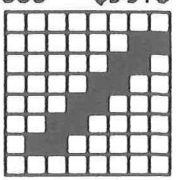
301 $D968



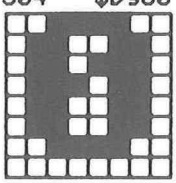
302 $D970



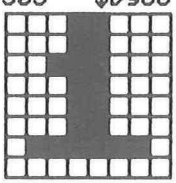
303 $D978



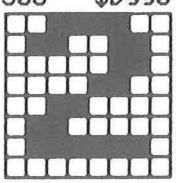
304 $D980



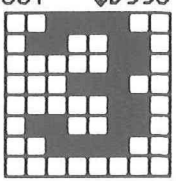
305 $D988



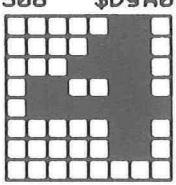
306 $D990



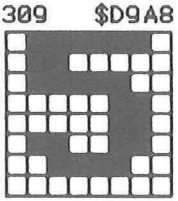
307 $D998



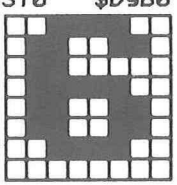
308 $D9A0



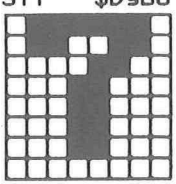
309 $D9A8



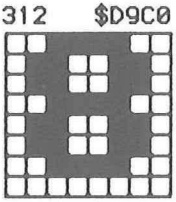
310 $D9B0



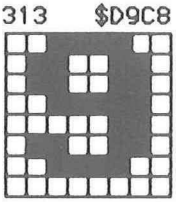
311 $D9B8



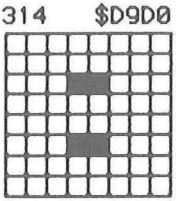
312 $D9C0



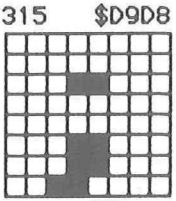
313 $D9C8



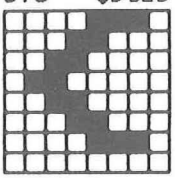
314 $D9D0



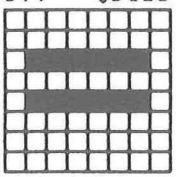
315 $D9D8



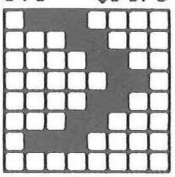
316 $D9E0



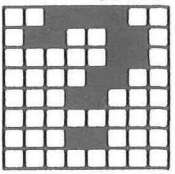
317 $D9E8



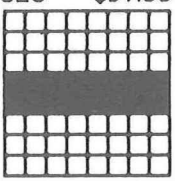
318 $D9F0



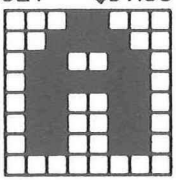
319 $D9F8



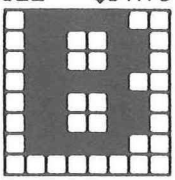
320 $DA00



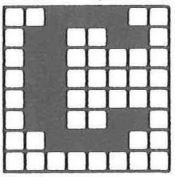
321 $DA08



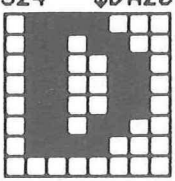
322 $DA10



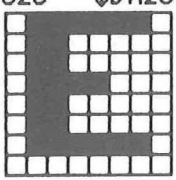
323 $DA18



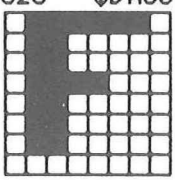
324 $DA20



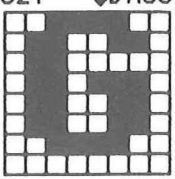
325 $DA28



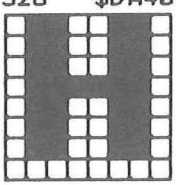
326 $DA30



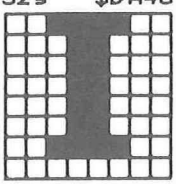
327 $DA38



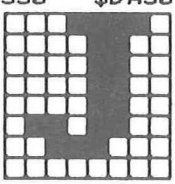
328 $DA40



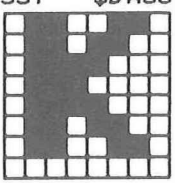
329 $DA48



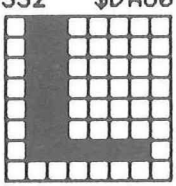
330 $DA50



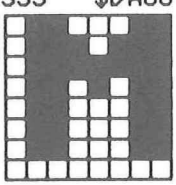
331 $DA58



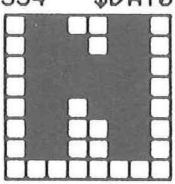
332 $DA60



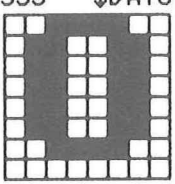
333 $DA68



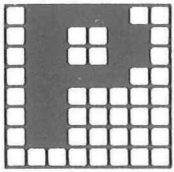
334 $DA70



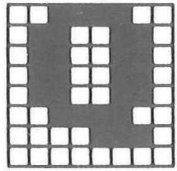
335 $DA78



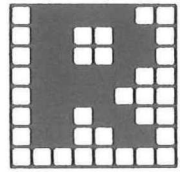
336 $DA80



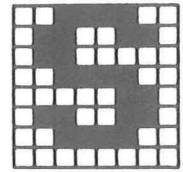
337 $DA88



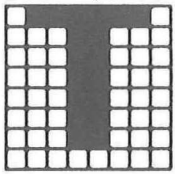
338 $DA90



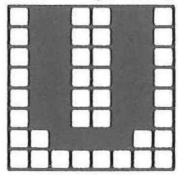
339 $DA98



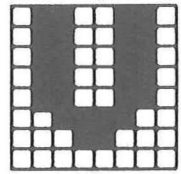
340 $DAA0



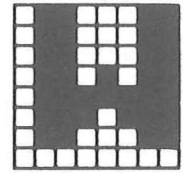
341 $DAA8



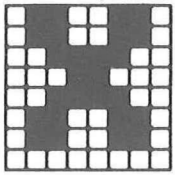
342 $DAB0



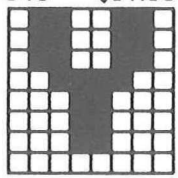
343 $DAB8



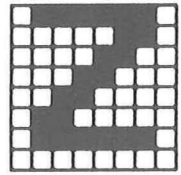
344 $DAC0



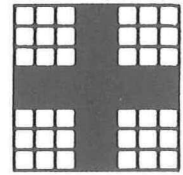
345 $DAC8



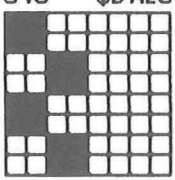
346 $DAD0



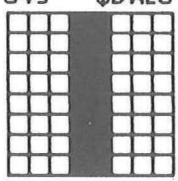
347 $DAD8



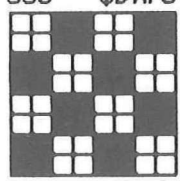
348 $DAE0



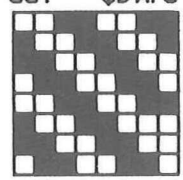
349 $DAE8



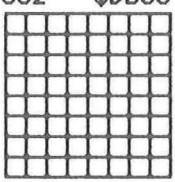
350 $DAF0



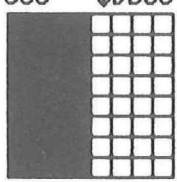
351 $DAF8



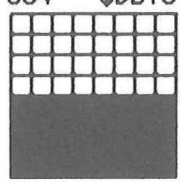
352 $DB00



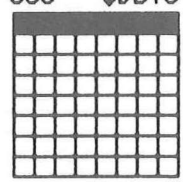
353 $DB08



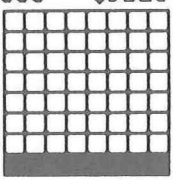
354 $DB10



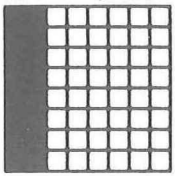
355 $DB18



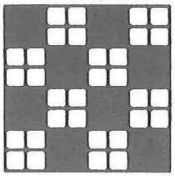
356 $DB20



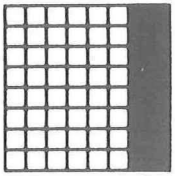
357 $DB28



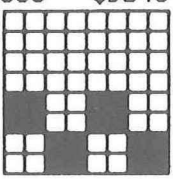
358 $DB30



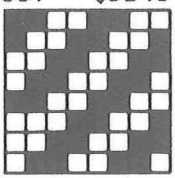
359 $DB38



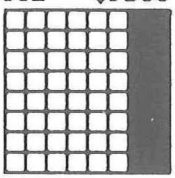
360 $DB40



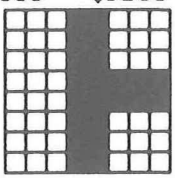
361 $DB48



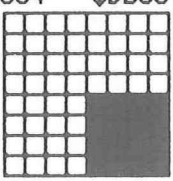
362 $DB50



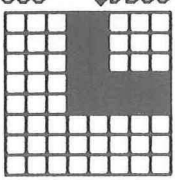
363 $DB58



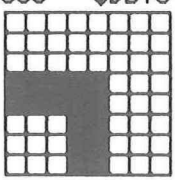
364 $DB60



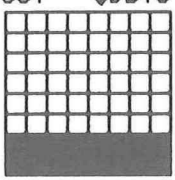
365 $DB68



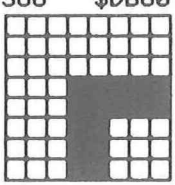
366 $DB70



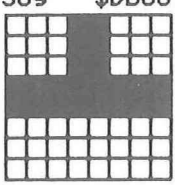
367 $DB78



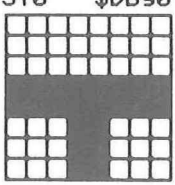
368 $DB80



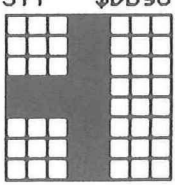
369 $DB88



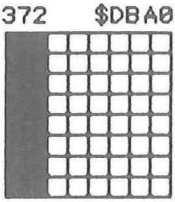
370 $DB90



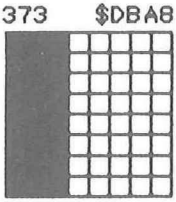
371 $DB98



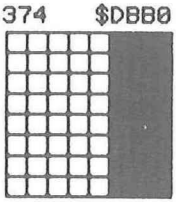
372 $DBA0



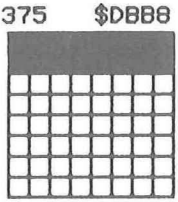
373 $DBA8



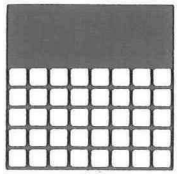
374 $DBB0



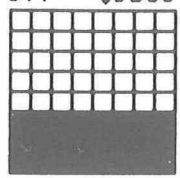
375 $DBB8



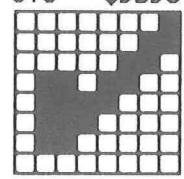
376 $DBC0



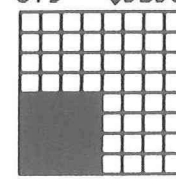
377 $DBC8



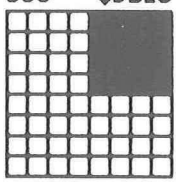
378 $DBD0



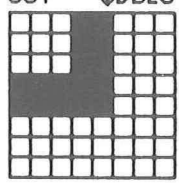
379 $DBD8



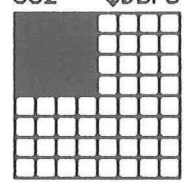
380 $DBE0



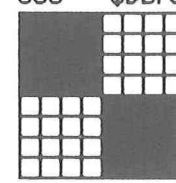
381 $DBE8



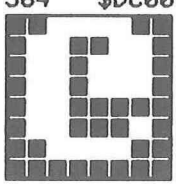
382 $DBF0



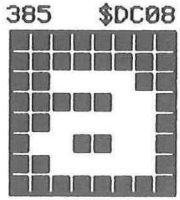
383 $DBF8



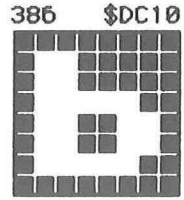
384 $DC00



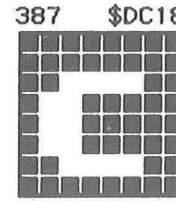
385 $DC08



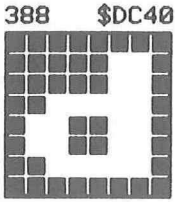
386 $DC10



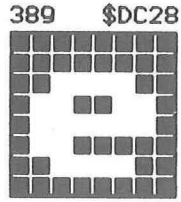
387 $DC18



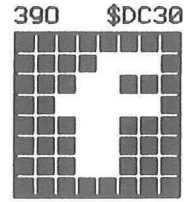
388 $DC40



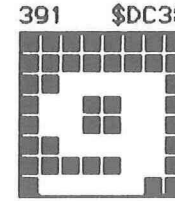
389 $DC28



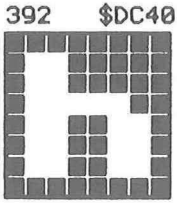
390 $DC30



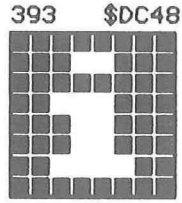
391 $DC38



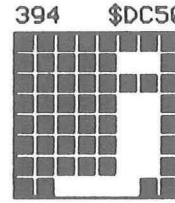
392 $DC40



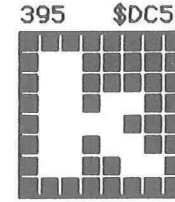
393 $DC48



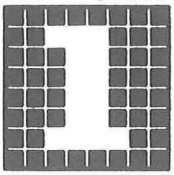
394 $DC50



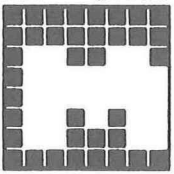
395 $DC58



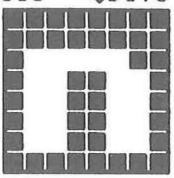
396 $DC60



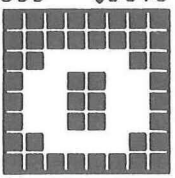
397 $DC68



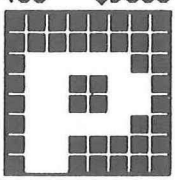
398 $DC70



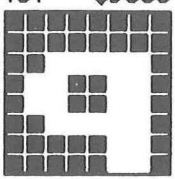
399 $DC78



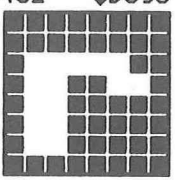
400 $DC80



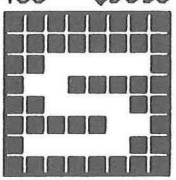
401 $DC88



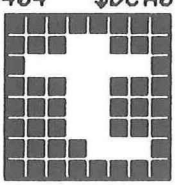
402 $DC90



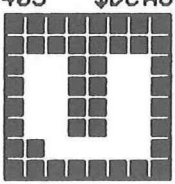
403 $DC98



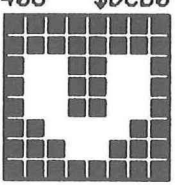
404 $DCA0



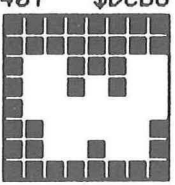
405 $DCA8



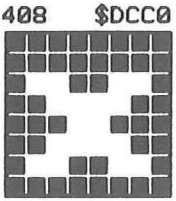
406 $DCB0



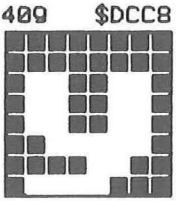
407 $DCB8



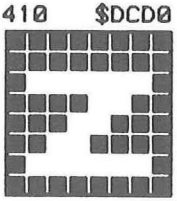
408 $DCC0



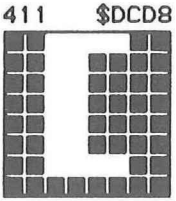
409 $DCC8



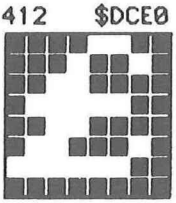
410 $DCD0



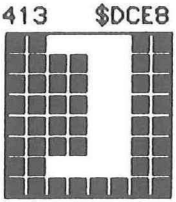
411 $DCD8



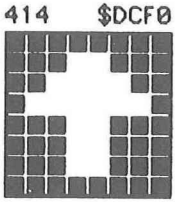
412 $DCE0



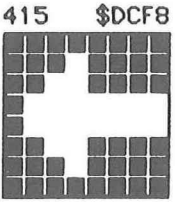
413 $DCE8



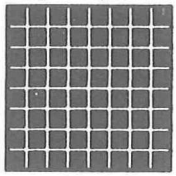
414 $DCF0



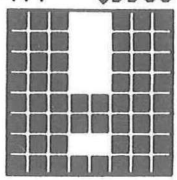
415 $DCF8



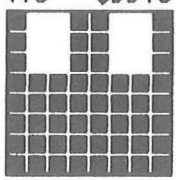
416 $DD00



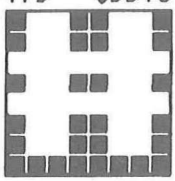
417 $DD08



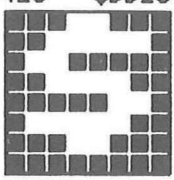
418 $DD10



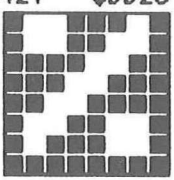
419 $DD18



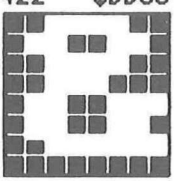
420 $DD20



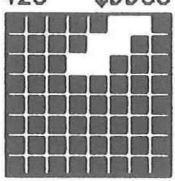
421 $DD28



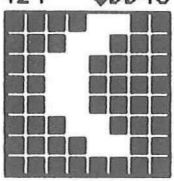
422 $DD30



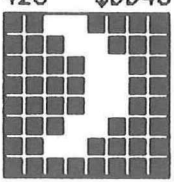
423 $DD38



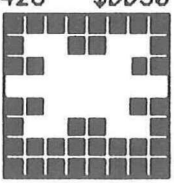
424 $DD40



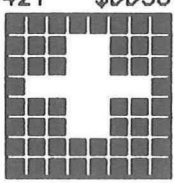
425 $DD48



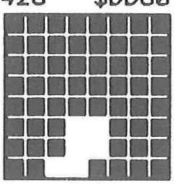
426 $DD50



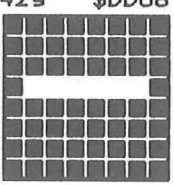
427 $DD58



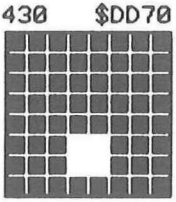
428 $DD60



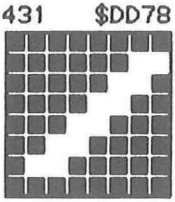
429 $DD68



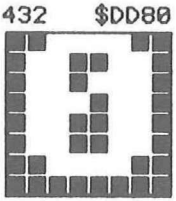
430 $DD70



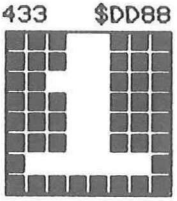
431 $DD78



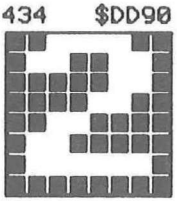
432 $DD80



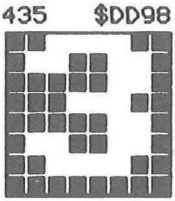
433 $DD88



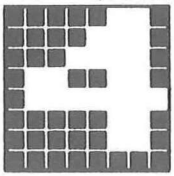
434 $DD90



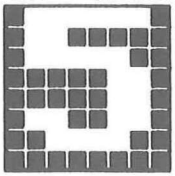
435 $DD98



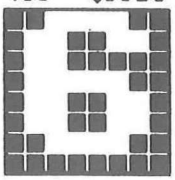
436 $DDA0



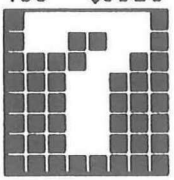
437 $DDA8



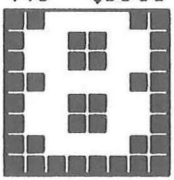
438 $DDB0



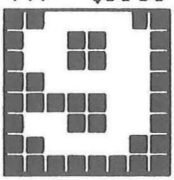
439 $DDB8



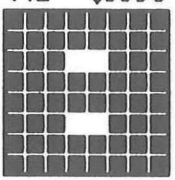
440 $DDC0



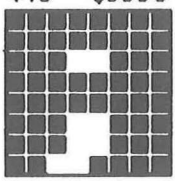
441 $DDC8



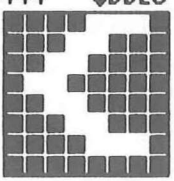
442 $DDD0



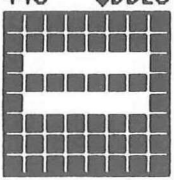
443 $DDD8



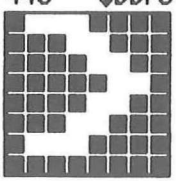
444 $DDE0



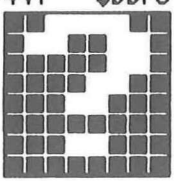
445 $DDE8



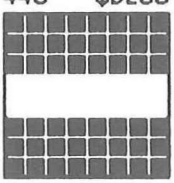
446 $DDF0



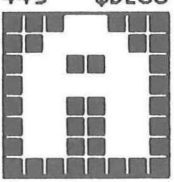
447 $DDF8



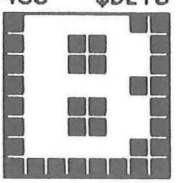
448 $DE00



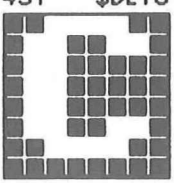
449 $DE08



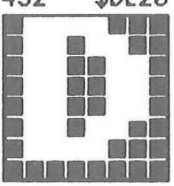
450 $DE10



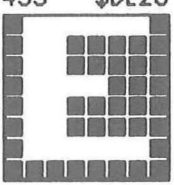
451 $DE18



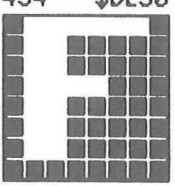
452 $DE20



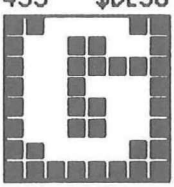
453 $DE28



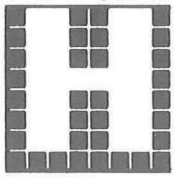
454 $DE30



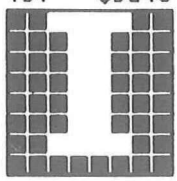
455 $DE38



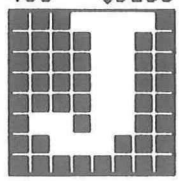
456 $DE40



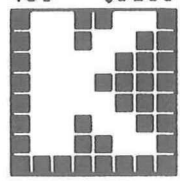
457 $DE48



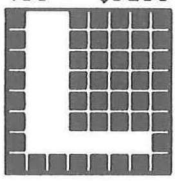
458 $DE50



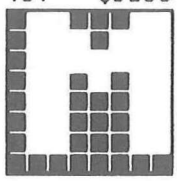
459 $DE58



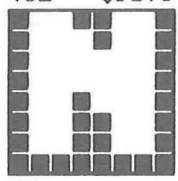
460 $DE60



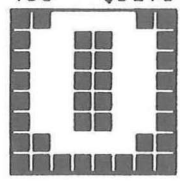
461 $DE68



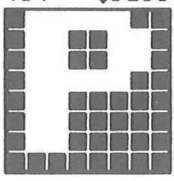
462 $DE70



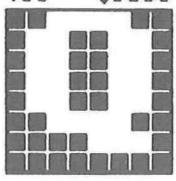
463 $DE78



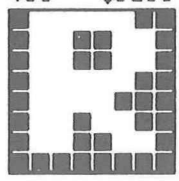
464 $DE80



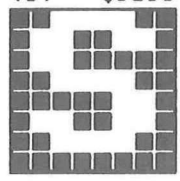
465 $DE88



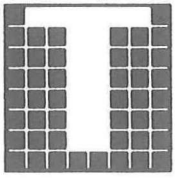
466 $DE90



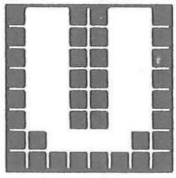
467 $DE98



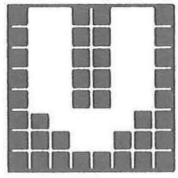
468 $DEA0



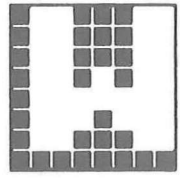
469 $DEA8



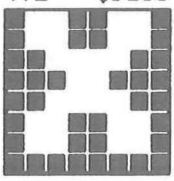
470 $DEB0



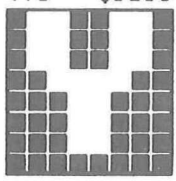
471 $DEB8



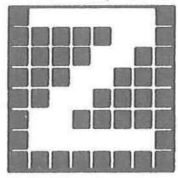
472 $DEC0



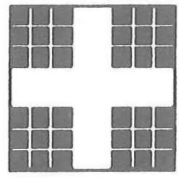
473 $DEC8



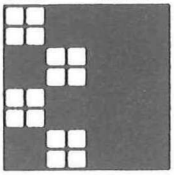
474 $DED0



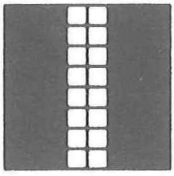
475 $DED8



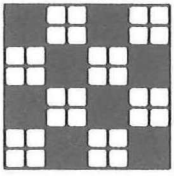
476 $DEE0



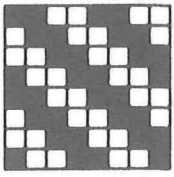
477 $DEE8



478 $DEF0

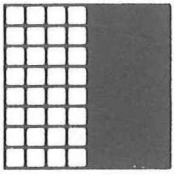


479 $DEF8

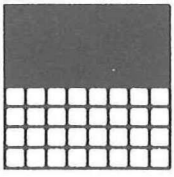


480 $DF00

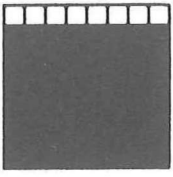
481 $DF08



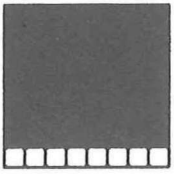
482 $DF10



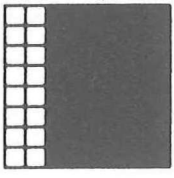
483 $DF18



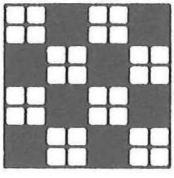
484 $DF20



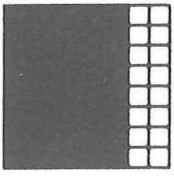
485 $DF28



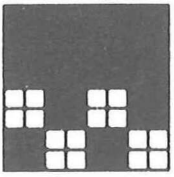
486 $DF30



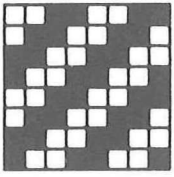
487 $DF38



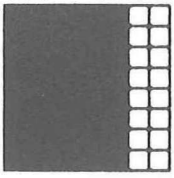
488 $DF40



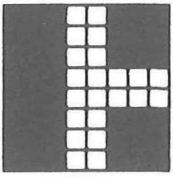
489 $DF48



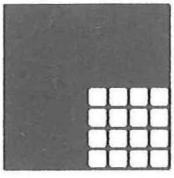
490 $DF50



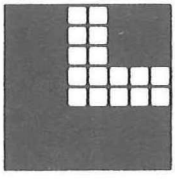
491 $DF58



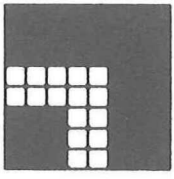
492 $DF60



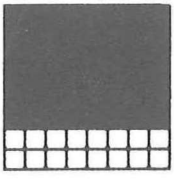
493 $DF68



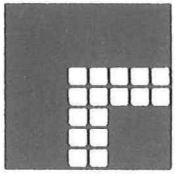
494 $DF70



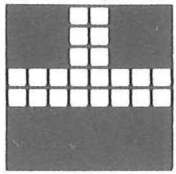
495 $DF78



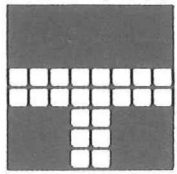
496 $DF80



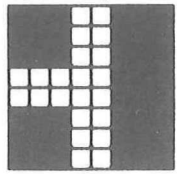
497 $DF88



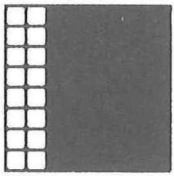
498 $DF90



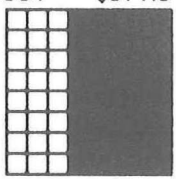
499 $DF98



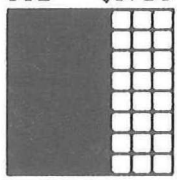
500 $DFA0



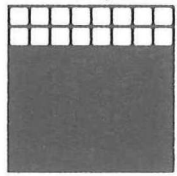
501 $DFA8



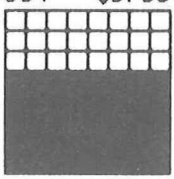
502 $DFB0



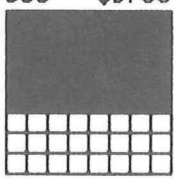
503 $DFB8



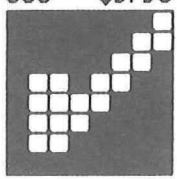
504 $DFC0



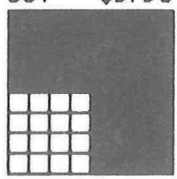
505 $DFC8



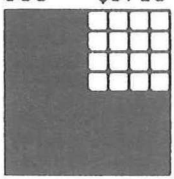
506 $DFD0



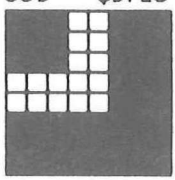
507 $DFD8



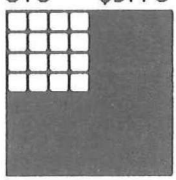
508 $DFE0



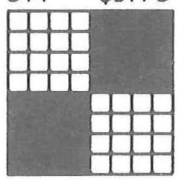
509 $DFE8



510 $DFF0



511 $DFF8



Anhang C: Assemblermnemonics und Maschinencodes

ADC Add memory to Accumulator with Carry
Addiere Speicher zu Akkumulator mit Übertrag

AND 'AND' Memory with Accumulator
log. Und Speicher/Akkumulator

ASL Shift Left one bit in memory or Accumulator
Um 1 Bit nach links verschieben in Speicher oder Akkumulator

BCC Branch on Carry Clear
Verzweige, wenn Übertragsbit nicht gesetzt

BCS Branch on Carry set
Verzweige, wenn Übertragsbit gesetzt

BEQ Branch on Result Zero
Verzweige, wenn Ergebnis = 0

BIT Test Bits in Memory with Accumulator
Vergleiche Bits im Speicher mit Akkumulator

BMI Branch on Result Minus
Verzweige, wenn Ergebnis negativ

BNE Branch on Result not Equal to Zero
Verzweige, wenn Ergebnis <> 0

BPL Branch on Result Plus
Verzweige, wenn Ergebnis positiv

BRK Force Break
Unterbrechung

BVC Branch on Overflow Clear
Verzweige, wenn Überlaufbit nicht gesetzt

BVS Branch on Overflow Set
Verzweige, wenn Überlaufbit gesetzt

CLC Clear Carry Flag
Löschen Übertragsbit

CLD Clear Decimal Mode
Löschen Dezimalmodus

CLI Clear Interrupt Disable Bit
Löschen Unterbrechungsbit

CLV Clear Overflow Flag
Löschen Überlaufbit

CMP Compare Memory and Accumulator
Vergleiche Speicher und Akkumulator

CPX Compare Memory and Index X
Vergleiche Speicher und Index X

CPY Compare Memory and Index Y
Vergleiche Speicher und Index Y

DEC Decrement Memory by One
Speicherinhalt um 1 verringern

DEX Decrement Index X by One
Index X um 1 verringern

DEY Decrement Index Y by One
Index Y um 1 verringern

EOR 'Exclusiv-Or' Memory with Accumulator
Exklusives Oder Speicher/Akkumulator

INC Increment Memory by One
Speicher um 1 erhöhen

INX Increment Index X by One
Index X um 1 erhöhen

INY Increment Index Y by One
Index Y um 1 erhöhen

JMP Jump to New Location
Verzweigung zu neuer Speicherstelle

JSR Jump to New Location but save return address
Verzweigung - Rücksprungadresse sichern

LDA Load Accumulator with Memory
Lade Speicherinhalt in Akkumulator

LDX Load Index X with Memory
Lade Speicherinhalt in Index X

LDY Load Index Y with Memory
Lade Speicherinhalt in Index Y

LSR Shift Right One Bit in Memory or Accumulator
Verschiebung um 1 Bit nach rechts in Speicher oder Akkumulator

NOP No Operation
Keine Operation

ORA 'OR' Memory with Accumulator
'Oder' Speicher/Akkumulator

PHA Push Accumulator on Stack
Setze Akkumulatorinhalt in den Stapel

PHP Push Processor Status on Stack
Setze Prozessor-Status in den Stapelspeicher

PLA Pull Accumulator from Stack
Übertrage von Stapelspeicher nach Akkumulator

PLP Pull Processor Status from Stack
Hole Prozessorstatus von Stapelspeicher

ROL Rotate One Bit Left in memory or Accumulator
Rotiere um 1 Bit nach links in Speicher oder Akkumulator

ROR Rotate One Bit Right in Memory or Accumulator
Rotiere um 1 Bit nach rechts in Speicher oder Akkumulator

RTI Return from Interrupt
Rücksprung von Unterbrechung

RTS Return from Subroutine
Rücksprung vom Unterprogramm

SBC Subtract Memory from Accumulator with Borrow
Speicherinhalt von Akkumulator abziehen mit Übertrag

SEC Set Carry Flag
Setzen Übertragsbit

SED Set Decimal Mode
Setzen Dezimalmodus

SEI Set Interrupt Disable Status
Setzen Unterbrechungsstatus

STA Store Accumulator in Memory
Speichern Akkumulator in Speicher

STX Store Index X in Memory
Speichern Index X in Speicher

STY Store Index Y in Memory
Speichern Index Y in Speicher

TAX Transfer Accumulator to Index X
Übertrage Akkumulator auf Index X

TAY Transfer Accumulator to Index Y
Übertrage Akkumulator auf Index Y

TSX Transfer Stack Pointer to Index X
Übertrage Stapelzeiger auf Index X

TXA Transfer Index X to Accumulator
Übertrage Index X in den Akkumulator

TXS Transfer Index S to Stack Pointer
Übertrage Index S in den Stapelzeiger

TYA Transfer Index Y to Accumulator
Übertrage Index Y in den Akkumulator

Anweisung	Format der Anweisung in Assembler	Hexadezimaler Code	Dezimaler Code
ADC	ADC #nn	69	105
	ADC aa	65	101
	ADC aa,X	75	117
	ADC aaaa	6D	109
	ADC aaaa,X	7D	125
	ADC aaaa,Y	79	121
	ADC (aa,X)	61	97
	ADC (aa),Y	71	113
AND	AND #nn	29	41
	AND aa	25	37
	AND aa,x	35	53
	AND aaaa	2D	45
	AND aaaa,X	3D	61
	AND aaaa,Y	39	57
	AND (aa,X)	21	33
	AND (aa),Y	31	49
ASL	ASL A	0A	10
	ASL aa	06	6
	ASL aa,Y	16	22
	ASL aaaa	0E	14
	ASL aaaa,X	1E	30
BCC	BCC aa	90	144
BCS	BCS aa	B0	176
BEQ	BEQ aa	F0	240
BIT	BIT aa	24	36
	BIT aaaa	2C	44
BMI	BMI aa	30	48
BNE	BNE aa	D0	208
BLP	BLP aa	10	16
BRK	BRK	00	0
BVC	BVC aa	50	80
BVS	BVS aa	70	112
CLC	CLC	18	24
CLD	CLD	D8	216
CLI	CLI	58	88
CLV	CLV	B8	184

Anweisung	Format der Anweisung in Assembler	Hexadezimaler Code	Dezimaler Code
CMP	CMP #nn	C9	21
	CMP aa	C5	197
	CMP aa,X	D5	213
	CMP aaaa	CD	205
	CMP aaaa,X	DD	221
	CMP aaaa,Y	D9	217
	CMP (aa,X)	C1	193
	CMP (aa),Y	D1	209
CPX	CPX #nn	E0	224
	CPX aa	E4	228
	CPX aaaa	EC	236
CPY	CPY #nn	C0	192
	CPY aa	C4	196
	CPY aaaa	CC	204
DEC	DEC aa	C6	198
	DEC aa,X	D6	214
	DEC aaaa	CE	206
	DEC aaaa,X	DE	222
DEX	DEX	CA	202
DEY	DEY	88	136
EOR	EOR #nn	49	73
	EOR aa	45	69
	EOR aa,X	55	85
	EOR aaaa	4D	77
	EOR aaaa,X	5D	93
	EOR aaaa,Y	59	89
	EOR (aa,X)	41	65
	EOR (aa),Y	51	81
INC	INC aa	E6	230
	INC aa,X	F6	246
	INC aaaa	EE	238
	INC aaaa,X	FE	254
INX	INX	E8	232
INY	INY	C8	200
JMP	JMP aaaa	4C	76
	JMP (aaaa)	6C	108
JSR	JSR aaaa	20	32
LDA	LDA #nn	A9	169
	LDA aa	A5	165
	LDA aa,X	B5	181
	LDA aaaa	AD	173

Anweisung	Format der Anweisung in Assembler	Hexadezimaler Code	Dezimaler Code
LDA	LDA aaaa,X	BD	189
	LDA aaa,Y	B9	185
	LDA (aa,X)	A1	161
	LDA (aa),Y	B1	177
LDX	LDX #nn	A2	162
	LDX aa	A6	166
	LDX aa,Y	B6	182
	LDX aaaa	AE	174
	LDX aaaa,Y	BE	190
LDY	LDY #nn	A0	160
	LDY aa	A4	164
	LDY aa,X	B4	180
	LDY aaaa	AC	172
	LDY aaaa,X	BC	188
LSR	LSR A	4A	74
	LSR aa	46	70
	LSR aa,X	56	86
	LSR aaaa	4E	78
	LSR aaaa,X5E94		
NOP	NOP #	EA	234
ORA	ORA #nn	09	9
	ORA aa	05	5
	ORA aa,X	15	21
	ORA aaaa	0D	13
	ORA aaaa,X	1D	29
	ORA aaaa,Y	19	25
	ORA (aa,X)	01	1
	ORA (aa),Y	11	17
PHA	PHA	48	72
PHP	PHP	08	8
PLA	PLA	68	104
PLP	PLP	28	40
ROL	ROL A	2A	42
	ROL aa	2638	
	ROL aa,X	36	54
	ROL aaaa	2E	46
	ROL aaaa,X	3E	62

Anweisung	Format der Anweisung in Assembler	Hexadezimaler Code	Dezimaler Code
ROR	ROR A	6A	106
	ROR aa	66	102
	ROR aa,X	76	118
	ROR aaaa	6E	110
	ROR aaaa,X	7E	126
RTI	RTI	40	64
RTS	RTS	60	96
SBC	SBC #nn	E9	233
	SBC aa	E5	229
	SBC aa,X	F5	245
	SBC aaaa	ED	237
	SBC aaaa,X	FD	253
	SBC aaaa,Y	F9	249
	SBC (aa,X)	E1	225
	SBC (aa),Y	F1	241
SEC	SEC	38	56
SED	SED	F8	248
SEI	SEI	78	120
STA	STA aa	85	133
	STA aa,X	95	149
	STA aaaa	8D	141
	STA aaaa,X	9D	157
	STA aaaa,Y	99	153
	STA (aa,X)	81	129
	STA (aa),Y	91	145
STX	STX aa	86	134
	STX aa,Y	96	150
	STX aaaa	8E	142
STY	STY aa	84	132
	STY aa,X	94	148
	STY aaaa	8C	140
TAX	TAX	AA	170
TAY	TAY	AB	168
TSX	TSX	BA	186
TXA	TXA	8A	138
TXS	TXS	9A	154
TYA	TYA	98	152

Anhang D: Zero-Page und erweiterte Zero-Page

00	d6510	Prozessor-Datenrichtungsregister
01	r6510	Prozessor-Datenregister
02-09		Register f. Monitor und "weite" Sprünge
02-82		BASIC-Zeropage-Speicherbereich
09	charac	Such-Zeichen
0A	endchr	Flag zum Suchen von Anführungszeichen am Stringende
0B	trmpos	Bildschirmspalte der letzten Tabulatorposition
0C	verck	Flag: 0=LOAD, 1=VERIFY
0D	count	Eingabepuffer-Zeiger/Dimensionen
0E	dimflg	Flag ob Defaultwerte bei Dimensionierung
0F	valtyp	Variablentyp: $FF=String, $00=Zahl
10	intflg	Variablentyp: $00=Real, $80=Integer
13	inpflg	Flag: INPUT/GET/READ
14	tansgn	Flag: Vorzeichen des Tangens/Vergleichsergebnis
16,17	linnum	Temporärer Integer-Wert
18	temppt	Zeiger: Temporärer String-Stack
19,1A	lastpt	Adresse des letzten temporären Strings
1B	tempst	Stack für temporäre Strings
24-27	index	Bereich fuer Hilfs-Zeiger
28-2C	resho	Gleitpunkt-Ergebnis der Multiplikation
2D,2E	txttab	Zeiger: Anfang des BASIC-Programmtextes
2F,30	vartab	Zeiger: Anfang der BASIC-Variablen
31,32	arytab	Zeiger: Anfang der BASIC-Arrays
33,34	strend	Zeiger: Ende der BASIC-Arrays (plus 1)
35,36	fretop	Zeiger: Untergrenze des String-Speichers
37,38	frespc	Hilfszeiger f. Strings
39,3A	max_mem_1	Obergrenze der String-/Variablenbank (= RAM-Bank 1)
3B,3C	curlin	Aktuelle BASIC-Zeilennummer
3D,3E	txtptr	Zeiger: BASIC-Text (für CHRGET u.s.w.)
3F	form	für PRINT USING
3F,40	fndpnt	Zeiger auf durch SEARCH gefundenes Zeichen
41,42	datlin	aktuelle DATA-Zeilennummer
43,44	datptr	Zeiger: Adresse des aktuellen DATA-Ausdruckes
45,46	inpptr	Zeiger: INPUT-Routine
47,48	varnem	aktueller BASIC-Variablenname
49,4A	varpnt	Zeiger: Wert der aktuellen BASIC-Variable
4B,4C	forpnt	Zeiger: Laufvariable für FOR..NEXT
55	helper	Flag: HELP oder LIST
58	oldov	
59-	tempf1	Bereich f. INSTR

63	facexp	FAC#1 Exponent
64	facho	FAC#1 Mantisse
68	facsgn	FAC#1 Vorzeichen
6A	argexp	FAC#2 Exponent
6B	argho	FAC#2 Mantisse
6F	argsgn	FAC#2 Vorzeichen
70	arisgn	Vorzeichen des Vergleichs FAC#1 <--> FAC#2
72,73	fbufpt	Zeiger: Kassettenpuffer
74,75	autinc	Wert Zeilenabstand für AUTO (0=aus)
76	mvdflg	Flag: 10K Hires-Speicher reserviert
77	noze	Zähler für führenden Nullen bei USING
77	sprnum	Zwischenspeicher f. MOVESPR und SPRITE
78	hulp	Zähler
79	syntmp	Zwischenspeicher für indirektes Laden
7A	dsdesc	Deskriptor für Disk-Statusvariable DS$
7D	tos	Obergrenze des "Runtime-" Stack
7F	runmod	Flag: RUN/Dierktmodus
80	parsts	Statuswort des DOS-Parsers
83-8F		Zeropage f. Grafik
83	colsel	aktueller Farbcode
84	multicolor_1	
85	multicolor_2	
86	foreground	
87	scale_x	Skalenfaktor X-Richtung
89	scale_y	Skalenfaktor Y-Richtung
8B	stopnb	PAINT-Abbruch, wenn nicht Hintergrund- / gleiche Farbe
90-CB		KERNAL/Editor
90	status	I/O-Statusbyte
91	stkey	Flag: Stop-Taste
92	svtx	Band-Zwischenspeicher
93	verck	Flag für LOAD/VERIFY
97	xsav	Zwischenspeicher f. BASIN
98	ldtnd	Anzahl offener log. Files
99	dfltn	Voreinstellung Eingabegerät
9A	dflto	Voreinstellung Ausgabegerät
A0-A2	time	Systemuhr (1/60-Sec. Zähler)
A6	bufpt	Zeiger: Kassettenpuffer
A8	rer	Lesefehler (Kassette)
AC,AB	sal	Zeiger: Kassetenpuffer / Bildschirmscroll
AE,AF	eal	Programmende (tape end adr.)
B0	cmp0	Konstanten f. Zeiteinteilung bei Kassettenbetrieb
B2,B3	tape1	Adresse des Band-Puffers
B7	fnlen	Länge des akt. Filenamens
B8	la	akt. log. Filenr.
B9	sa	akt. Sekundäradresse
BA	fa	akt. Gerätenummer
BB,BC	fnadr	Adresse des akt. Filenamens

C0	cas1	Schalter am Kassettenlaufwerk
C1	track	
C2	sector	
C2,C3	stal	I/O Startadresse
C5	data	Daten f. Kassettenoperation
C6	ba	Bank für akt. LOAD/SAVE/VERIFY
C7	fnbank	Bank, in der der akt. Filename zu finden ist
C8,C9	ribuf	Zeiger: RS232 Eingabepuffer
CA,CB	robuf	Zeiger: RS232 Ausgabepuffer
CC-FF		40/80-Zeichen Editor
CC,CD	keytab	Zeiger auf Tastaturdecodierungstabelle
CE,CF	imparm	für BS-Routine PRIMM
D0	ndx	Zeichen in Tastaturpuffer
D1	kyndx	Flag für Funktionstasten
D2	keyidx	Funktionstastenindex
D7	mode	Flag: 40/80-Zeichen Bildschirm ($00/$80)
D8	graphm	Flag: Text/Grafik-Modus
DA	keysiz	Funktionstasten-Bereich
DB	keylen	
DC	keynum	
DD	keynxt	
DE	keybnk	
DF	keytmp	
DA	bitmsk	für TAB() und Zeilenumbruch
E0,E1	pnt	Zeiger: akt. Text-Zeile
E2,E3	user	Zeiger: akt. Zeile (Attribut)
E4	scbot	Untergrenze im Fenster
E5	sctop	Obergrenze im Fenster
E6	sclf	Linker Rand im Fenster
E7	scrt	Rechter Rand im Fenster
E8	lsxp	INPUT: Anfangsspalte
E9	lstp	INPUT: Anfangszeile
EA	indx	INPUT: Endzeile
EB	tblx	akt. Cursor-Zeile
EC	pntr	akt. Cursor-Spalte
ED	lines	Max. Zeilen im Bildschirm
EE	columns	Max. Spalten im Bildschirm
EF	datax	akt. auszugebendes Zeichen
F0	lstchr	zuletzt ausgegebenes Zeichen (f. <esc> - Test)
F1	color	Vordergrundfarbe
F3	rvs	Flag: Revers-Modus
F4	qtsw	Flag: Quote-Modus
F5	insrt	Flag: Insert-Modus
F6	insflg	Flag: Auto-Insert-Modus
F7	locks	Flag: Umschaltung Groß-/Kleinschrift sperren ($80/$00)
F8	scroll	Flag: Scrollen verhindermn ($80/$00)
F9	beeper	Flag: Beeper (<ctrl>-G) ausschalten ($80/$00)

```
0100-0124              Verbindungsstelle zwischen BASIC/DOS
0125-139               für PRINT USING
0200        buf        Eingabepuffer (BASIC & MONITOR)
02A2-0313              ausgelagerte Systemroutinen und Vektoren
0314-0333              Sprungvektoren

0334-0390              Editor-Vektoren
0334,0335   ctlvec
0336,0337   shfvec
0338,0339   escvec
033A,033B   keyvec
033C,033D   keychk
033E,033F   decode
034A        keyd       IRQ-Tastaturpuffer
0354        tabmap     Bitmap der Tabulatorpositionen
035E        bitabl     Bitmap Zeilenumbrüche

400-07E7    vicscn     Bildschirmzeichenspeicher
0800                   BASIC Runtime-Stack (512 Bytes)

0A00-0A1F              Arbeitsbereich des Betriebss Betriebssystems
0A00,0A01              BASIC-Warmstart-Vektor
0A05,0A06   memstr     Untergrenze des verfügbaren Arbeitsspeichers
0A07,0A08   memsize    Obergrenze des verfügbaren Arbeitsspeichers
0A0E        timout     Timeout-Flag für Fast-Serial
0A0F-0A1B              RS232-Bereich
0A1C        serial     Fast-Serial Int./Ext.-Flag

0A20-0A39              globale Deklarationen Bildschirmeditor
0A80-0AB4              für MONITOR
0B00                   Kassettenpuffer
0C00                   RS232 Eingabepuffer
0D00                   RS232 Ausgabepuffer
0E00                   Bereich für Spritedefinitionen
1000                   Funktionstastendefinitionen
1100                   DOS Pufferbereich für Ausgabestrings
                       CP/M-Resetcode
1200                   BASIC-Arbeitsbereich
1C00        rambot     BASIC-Text beginnt hier, falls keine Grafik
                       alloziert ist (sonst $4000).
```

Anhang E: Sprungtabellen: BASIC, Monitor und Editor

A000	basic	BASIC Kaltstart
A003	basicw	BASIC Warmstart
A006	basirq	BASIC-Einstieg für IRQ
B000	monitr	Einstieg für Monitor-Aufruf
B003	monbrk	Monitor-Einstieg für Break-Interrupt
B006	moncmd	Einstieg für Parser des Monitor
C000	cint	Initialisierung
C003	display	Ausgabe: Zeichen in A, Farbe in X
C006	lp2	Ein Zeichen aus Tastaturpuffer holen
C009	loop5	Ein Zeichen vom Bildschirm holen
C00C	print	Ein Zeichen am Bildschirm ausgeben
C00F	scrorg	Größe des aktuellen Bildschirms holen
C012	key	Tastatur nach gedrückter Taste absuchen
C015	repeat	Tastaturlogik erneut aufrufen, um decodierte Taste abzuspeichern
C018	plot	Cursorposition holen/setzen
C01B	cursor	Cursor des 80-Zeichen-Chips bewegen
C01E	escape	Bearbeitung von ESCAPE-Sequenzen, Zeichen in A
C021	pfkey	Funktionstaste neu belegen
C024	edirq	IRQ-Einstieg für Editor
C027	dlchr	Zeichensatz für 80-Zeichen-Chip neu laden
C02A	swapper	Umschaltung zwischen 40-/80-Zeichen-Modus

Anhang F: Die wichtigsten Ein-/Ausgabe-Bausteine

6581 SID Register

REG #									
0_	F7	F6	F5	F4	F3	F2	F1	F0	-- Frequenz Low-Byte Oszillator 1
1_	F15	F14	F13	F12	F11	F10	F9	F8	-- Frequenz Hi-Byte
2_	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	-- Pulsbreite Low-Byte
3_	--	--	--	--	PW11	PW10	PW9	PW8	-- Pulsbreite Hi-Byte
4_	NOISE	PULSE	SAW	TRI	TEST	RING	SYNC	GATE	-- Steuer-Register
5_	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	-- Anstiegzeit/Abschwellzeit
6_	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	-- Haltezeit/Ausklingzeit
7_	F7	F6	F5	F4	F3	F2	F1	F0	-- Frequenz Low-Byte Oszillator 2
8_	F15	F14	F13	F12	F11	F10	F9	F8	-- Frequenz Hi-Byte
9_	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	-- Pulsbreite Low-Byte
10_	--	--	--	--	PW11	PW10	PW9	PW8	-- Pulsbreite Hi-Byte
11_	NOISE	PULSE	SAW	TRI	TEST	RING	SYNC	GATE	-- Steuer-Register
12_	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	-- Anstiegzeit/Abschwellzeit
13_	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	-- Haltezeit/Ausklingzeit
14_	F7	F6	F5	F4	F3	F2	F1	F0	-- Frequenz Low-Byte Oszillator 3
15_	F15	F14	F13	F12	F11	F10	F9	F8	-- Frequenz Hi-Byte
16_	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	-- Pulsbreite Low-Byte
17_	--	--	--	--	PW11	PW10	PW9	PW8	-- Pulsbreite Hi-Byte
18_	NOISE	PULSE	SAW	TRI	TEST	RING	SYNC	GATE	-- Steuer-Register
19_	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	-- Anstiegzeit/Abschwellzeit
20_	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	-- Haltezeit/Ausklingzeit
21_	--	--	--	--	--	FC2	FC1	FC0	-- Frequenz Low-Byte Filter
22_	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	-- Frequenz Hi-Byte
23_	RES3	RES2	RES1	RES0	FILTEX	FILT3	FILT2	FILT0	-- Resonanz/Filterung
24_	3 OFF	HP	BP	LP	VOL3	VOL2	VOL1	VOL0	-- Filterart/Lautstärke
25_	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	-- Pot X
26_	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	-- Pot Y
27_	O7	O6	O5	O4	O3	O2	O1	O0	-- Rauschen Osz.3
28_	E7	E6	E5	E4	E3	E2	E1	E0	-- Hüllkurve Osz.3

8563 Videocontroller

REG #									
0_	HT7	HT6	HT5	HT4	HT3	HT2	HT1	HT0	------ Horizontal Total
1_	HD7	HD6	HD5	HD4	HD3	HD2	HD1	HD0	------ Horizontal Displayed
2_	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0	------ Horizontal Sync Position
3_	VW3	VW2	VW1	VW0	HW3	HW2	HW1	HW0	------ Vert/Horz Sync Width
4_	VT7	VT6	VT5	VT4	VT3	VT2	VT1	VT0	------ Vertical Total
5_	--	--	--	VA4	VA3	VA2	VA1	VA0	------ Vertical Total Adjust
6_	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	------ Vertical Displayed
7_	VP7	VP6	VP5	VP4	VP3	VP2	VP1	VP0	------ Vertical Sync Position
8_	--	--	--	--	--	--	IM1	IM0	------ Interlace Mode
9_	--	--	--	CTV4	CTV3	CTV2	CTV1	CTV0	------ Character Total Vertical
10_	--	CM1	CM0	CS4	CS3	CS2	CS1	CS0	------ Cursor Mode, Start Scan
11_	--	--	--	CE4	CE3	CE2	CE1	CE0	------ Cursor End Scan Line
12_	DS15	DS14	DS13	DS12	DS11	DS10	DS9	DS8	------ Display Start Address hi
13_	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0	------ Display Start Address lo
14_	CP15	CP14	CP13	CP12	CP11	CP10	CP9	CP8	------ Cursor Position hi
15_	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0	------ Cursor Position lo
16_	LPV7	LPV6	LPV5	LPV4	LPV3	LPV2	LPV1	LPV0	------ Light Pen Vertical
17_	LPH7	LPH6	LPH5	LPH4	LPH3	LPH2	LPH1	LPH0	------ Light Pen Horizontal
18_	UA15	UA14	UA13	UA12	UA11	UA10	UA9	UA8	------ Update Address hi
19_	UA7	UA6	UA5	UA4	UA3	UA2	UA1	UA0	------ Update Address lo
20_	AA15	AA14	AA13	AA12	AA11	AA10	AA9	AA8	------ Attribute Start Adr hi
21_	AA7	AA6	AA5	AA4	AA3	AA2	AA1	AA0	------ Attribute Start Adr lo
22_	CTH3	CTH2	CTH1	CTH0	CDH3	CDH2	CDH1	CDH0	------ Character Tot(h), Dsp(v)
23_	--	--	--	CDV4	CDV3	CDV2	CDV1	CDV0	------ Character Dsp(v)
24_	COPY	RVS	CBRATE	VSS4	VSS3	VSS2	VSS1	VSS0	------ Vertical smooth scroll
25_	TEXT	ATR	SEMI	DBL	HSS3	HSS2	HSS1	HSS0	------ Horizontal smooth scroll
26_	FG3	FG2	FG1	FG0	BG3	BG2	BG1	BG0	------ Foregnd/Bgnd Color
27_	AI7	AI6	AI5	AI4	AI3	AI2	AI1	AI0	------ Address Increment / Row
28_	CB15	CB14	CB13	RAM	--	--	--	--	------ Character Base Address
29_	--	--	--	UL4	UL3	UL2	UL1	UL0	------ Underline scan line
30_	WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0	------ Word Count
31_	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	------ Data
32_	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	------ Block Start Address hi
33_	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	------ Block Start Address lo
34_	DEB7	DEB6	DEB5	DEB4	DEB3	DEB2	DEB1	DEB0	------ Display Enable Begin
35_	DEE7	DEE6	DEE5	DEE4	DEE3	DEE2	DEE1	DEE0	------ Display Enable End
36_	--	--	--	--	DRR3	DRR2	DRR1	DRR0	------ DRAM Refresh rate

Beschreibung der MAPPED-Register:

$D600: Lesezugriff ergibt STATUS
Schreibzugriff setzt Registerinhalt

$D601: Daten: Ein-/Ausgabe

Zeichen-Speicher: $0000...$07CF
Attribut-Speicher: $0800...$0FCF
Zeichengenerator: $2000...$3FFF

--	--	R5	R4	R3	R2	R1	R0
STATUS	LP	VBLANK	--	--	--	--	--
D7	D6	D5	D4	D3	D3	D2	D0

ALT	RVS	UL	FLASH	R	G	B	I

8564 VIC Register

REG#									
0	S0x7	S0x6	S0x5	S0x4	S0x3	S0x2	S0x1	S0x0	Sprite 0 X Koordinate
1									Sprite 0 Y Koordinate
2									Sprite 1 X Koordinate
3									Sprite 1 Y Koordinate
4									Sprite 2 X Koordinate
5									Sprite 2 Y Koordinate
6									Sprite 3 X Koordinate
7									Sprite 3 Y Koordinate
8									Sprite 4 X Koordinate
9									Sprite 4 Y Koordinate
10									Sprite 5 X Koordinate
11									Sprite 5 Y Koordinate
12									Sprite 6 X Koordinate
13									Sprite 6 Y Koordinate
14									Sprite 7 X Koordinate
15									Sprite 7 Y Koordinate
16	S7x8	S6x8	S5x8	S4x8	S3x8	S2x8	S1x8	S0x8	Sprite X plus 256
17	RCB	ECM	BMM	Blnk	ROWS	Y2	Y1	Y0	Modus, Y-scroll
18	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raster-Register
19	LPx8	LPx7	LPx6	LPx5	LPx4	LPx3	LPx2	LPx1	Lightpen X-Koordinate
20	LPy7	LPy6	LPy5	LPy4	LPy3	LPy2	LPy1	LPy0	Lightpen Y-Koordinate
21	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0	Sprite X: AN/AUS
22	--	--	Rset	MCM	COLS	X2	X1	X0	Modus, X-Scroll
23	S7Ey	S6Ey	S5Ey	S4Ey	S3Ey	S2Ey	S1Ey	S0Ey	Sprite in Y-Richtung vergrößern
24	VM13	VM12	VM11	VM10	CB13	CB12	CB11	--	Basisadresse BSS und ZG *
25	IRQ	--	--	--	LP	S/S	S/B	RIRQ	Interrupt-Flags
26	--	--	--	--	_LP	_S/S	_S/B	_IRQ	Interrupt-Maske
27	BSP7	BSP6	BSP5	BSP4	BSP3	BSP2	BSP1	BSP0	Sprite-Hintergr. Priorität
28	MCS7	MCS6	MCS5	MCS4	MCS3	MCS2	MCS1	MCS0	Sprite X=Multicolor
29	S7Ex	S6Ex	S5Ex	S4Ex	S3Ex	S2Ex	S1Ex	S0Ex	Sprite in X-Richtung vergrößern
30	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0	Kollision Sprite/Sprite
31	SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0	Kollision Sprite/Hintergr. (Zeichen)
32	--	--	--	--					Randfarbe
33	--	--	--	--					Hintergrundfarbe 1
34	--	--	--	--					Hintergrundfarbe 2
35	--	--	--	--					Hintergrundfarbe 3
36	--	--	--	--					Hintergrundfarbe 4
37	--	--	--	--					Sprite Mehrfarbregister 1
38	--	--	--	--					Sprite Mehrfarbregister 2
39	--	--	--	--					Sprite 0 Farbe
40	--	--	--	--					Sprite 1 Farbe
41	--	--	--	--					Sprite 2 Farbe
42	--	--	--	--					Sprite 3 Farbe
43	--	--	--	--					Sprite 4 Farbe
44	--	--	--	--					Sprite 5 Farbe
45	--	--	--	--					Sprite 6 Farbe
46	--	--	--	--					Sprite 7 Farbe
47	--	--	--	--	--	K2	K1	K0	Tastatur-Kennung K2..K0
48	--	--	--	--	--	--	TEST	2MHz	Takt-Geschwindigkeit

* BGS = Bildschirmspeicher
ZG = Zeichengenerator

Anhang G: Die Schnittstelle zum Betriebssystem

C64MODE

Zweck:	Übergang in den C64-Modus
Adresse:	$FF4D
Anmerkungen:	Nur aus den Speicherkonfigurationen C, D und F erreichbar.

DMA-CALL

Zweck:	Initialisierung des (externen) direkten Speicherzugriffs.
Adresse:	$FF50

BOOTCALL

Zweck:	Von Diskette booten.
Adresse:	$FF53
Parameterübergabe:	A, X
Anmerkungen:	Drivenummer in A, Geräteadresse in X.

PHOENIX

Zweck:	Test, ob in Expansion-Port Speicher-Karten eingesteckt sind; falls ja, Übergabe der Kontrolle an diese.
Adresse:	$FF56

LKUPLA

Zweck:	Suchen einer logischen Adresse.
Adresse:	$FF59
Parameterübergabe:	A, X, Y, Carry und Zeropage-Adressen $B8 bis $BA
Anmerkungen:	Y enthält die zu suchende logische Adresse. Falls gefunden, werden die Fileparameter nach $B8 bis $BA gebracht und in A, X und Y zurückgeliefert. Andernfalls wird das Carry-Flag gesetzt.

LKUPSA

Zweck:	Suchen einer Sekundäradresse.
Adresse:	$FF5C
Parameterübergabe:	A, X, Y, Carry und Zeropage-Adressen $B8 bis $BA
Anmerkungen:	Y enthält die zu suchende Sekundäradresse. Falls gefunden, werden die Fileparameter nach $B8 bis $BA gebracht und in A, X und Y zurückgeliefert. Andernfalls wird das Carry-Flag gesetzt. Sekundäradressen besitzen als HI-Nybble den Wert $6.

SWAPPER

Zweck:	Umschalten zwischen 40- und 80-Zeichen-Bildschirm.
Adresse:	$FF5F

LOAD80CH

Zweck:	Neuladen des Zeichensatzes am 80-Zeichen Bildschirm
Adresse:	$FF62

PFKEY

Zweck:	Umdefinieren einer Funktionstastenbelegung.
Adresse:	$FF65
Parameterübergabe:	A, X, Y und einen Zeiger in der Zeropage.
Anmerkungen:	Der Zeiger gibt die Adresse des ersten Bytes des neuen Funktionstastenstrings an, A enthält die Zeropage-Adresse des Zeigers. In X wird die Nummer der Funktionstaste und in Y die Länge des Strings übergeben.

SETBNK

Zweck:	Arbeitsspeicherbanken für LOAD/VERIFY/SAVE und Filenamen setzen.
Adresse:	$FF68
Parameterübergabe:	A, Y
Anmerkungen:	A muß den Konfigurationsindex des zu bearbeitenden Speicherbereichs, Y den des Filenamens enthalten.

GETCONF

Zweck:	Holen eines bestimmten Konfigurationsbytes
Adresse:	$FF6B
Parameterübergabe:	A, X
Anmerkungen:	X enthält den Konfigurationsindex (0..15); das entsprechende Konfigurationsbyte wird in A zurückgeliefert.

JSRFAR

Zweck:	Sprung in ein Unterprogramm, das in einer anderen Bank als der aktuellen steht.
Adresse:	$FF6E
Parameterübergabe:	Adressen $02 bis $09 in der Zeropage.
Anmerkungen:	Index der Zielkonfiguration in $02; $03 und $04 sind der neue Program Counter (Sprungadresse), $05 der neue Prozessorstatus. Die Register A, X, und Y werden vor dem Sprung mit den ab $06 bis $08 stehenden Werten vorbesetzt; dort finden sich auch die Registerinhalte nach Rückkehr aus dem Unterpro-

gramm wieder. Zusätzlich werden in $05 der Prozessorstatus und in $09 der Stackpointer übergeben.

JMPFAR

Zweck:	Sprung in ein Programm, das in einer anderen Bank als der aktuellen steht.
Adresse:	$FF71
Parameterübergabe:	Adressen $02 bis $08 in der Zeropage.
Anmerkungen:	Index der Zielkonfiguration in $02; $03 und $04 sind der neue Program Counter (Sprungadresse), $05 der neue Prozessorstatus. Die Register A, X und Y werden mit den ab $06 bis $08 stehenden Werten vorbesetzt. Aufruf immer mit JMP!

INDFET

Zweck:	Laden des Akkumulators mit dem Wert einer Speicherstelle, die in einer anderen Bank als der aktuellen liegt.
Adresse:	$FF74
Parameterübergabe:	A, X, Y und ein Zeiger in der Zeropage.
Anmerkungen:	X enthält Konfigurationsindex. Der Zeiger auf die Ziel-Speicherstelle muß in der Zeropage abgelegt sein (Indizierung durch Y möglich). Die Zeropage-Adresse des Zeigers wird im Akkumulator übergeben.

INDSTA

Zweck:	Abspeichern des Akkumulatorinhalts an einer Speicherstelle in einer anderen als der aktuellen Bank.
Adresse:	$FF77
Parameterübergabe:	A, X, Y und ein Zeiger in der Zeropage.
Anmerkungen:	Abzuspeicherndes Byte in A, Konfigurationsindex in X. Der Zeiger (2 Bytes) auf die Speicherstelle, an der der Akkumulatorinhalt abgespeichert werden soll, muß in der Zeropage abgelegt werden (Indizierung durch Y möglich). Zusätzlich ist die Zeropage-Adresse des Zeigers an Adresse $02B9 abzuspeichern.

INDCMP

Zweck:	Vergleich des Akkumulatorinhalts mit einer Speicherstelle, die in einer anderen als der aktuellen Bank steht.
Adresse:	$FF7A
Parameterübergabe:	A, X, Y und ein Zeiger in der Zeropage.
Anmerkungen:	Vergleichsbyte in A, Konfigurationsindex in X. Der Zeiger (2 Bytes) auf das zu vergleichende Byte muß in der Zeropage abgelegt werden (Indizierung durch Y möglich). Zusätzlich ist die Zeropage-Adresse des Zeigers an Adresse $02C8 abzuspeichern.

PRIMM

Zweck:	Ausgabe des nach dem Aufruf dieser Routine stehenden Textes (PRINT IMMEDIATE).
Adresse:	$FF7D
Anmerkungen:	Ende des Textes wird durch ein Nullbyte gekennzeichnet. Diese Routine muß immer mit JSR aufgerufen werden. Nach Ausgabe des Textes wird mit dem nach dem Nullbyte stehenden Befehl fortgefahren.

CINIT

Zweck:	Editor und Videocontroller initialisieren
Adresse:	$FF81
Anmerkungen:	Kann zum Einschalten des 40/80-Zeichen-Modus in Abhängigkeit der 40/80-Zeichen-Taste benutzt werden.

IOINIT

Zweck:	Initialisierung der I/O-Einheiten
Adresse:	$FF84

RAMTAS

Zweck:	Löscht Zeropage, setzt Zeiger auf SYSTOP, SYSBOT, RS232-Ein- und Ausgabepuffer, Kassettenpuffer und BASIC-Warmstart.
Adresse:	$FF87

RESTOR

Zweck:	Systemvektoren initialisieren.
Adresse:	$FF8A

VECTOR

Zweck:	Systemvektoren kopieren oder neu Setzen.
Adresse:	$FF8D
Parameterübergabe:	X,Y, Carry-Bit
Anmerkungen:	Bei gesetztem Carry-Flag werden die Vektoren ab der durch Y und X gekennzeichneten Adresse abgespeichert. Sonst werden die Vektoren aus dem durch X und Y angegebenen Bereich neu geladen.

SETMSG

Zweck:	Auswahl der Meldungen des DOS treffen.
Adresse:	$FF90
Parameterübergabe:	A

SECND

Zweck:	Sekundäradresse nach LISTEN senden.
Adresse:	$FF93
Parameterübergabe:	A

TKSA

Zweck:	Sekundäradresse nach TALK senden.
Adresse:	$FF96
Parameterübergabe:	A

MEMTOP

Zweck:	Speicherobergrenze holen/setzen.
Adresse:	$FF99
Parameterübergabe:	X, Y, Carry-Flag
Anmerkungen:	Bei nicht gesetztem Carry-Flag wird die Speicherobergrenze aus den Registern X (low) und Y (high) neu gesetzt, ansonsten die aktuelle Speicherobergrenze nach X und Y übertragen.

MEMBOT

Zweck:	Speicheruntergrenze holen/setzen.
Adresse:	$FF9C
Parameterübergabe:	X, Y, Carry-Flag
Anmerkungen:	Bei nicht gesetztem Carry-Flag wird die Speicheruntergrenze aus den Registern X (low) und Y (high) neu gesetzt, ansonsten die aktuelle Speicheruntergrenze nach X und Y übertragen.

KEY (SCNKEY)

Zweck:	Tastatur nach gedrückter Taste überprüfen.
Adresse:	$FF9F
Anmerkungen:	Falls eine Taste gedrückt wurde, so wird deren ASCII-Wert dem Tastaturpuffer (ab $34A) zugefügt.

SETTMO

Zweck:	Timeout-Flag für IEEE-Bus setzen.
Adresse:	$FFA2

ACPTR

Zweck:	Ein Byte vom seriellen Bus holen.
Adresse:	$FFA5
Parameterübergabe:	A

CIOUT

Zweck:	Ein Byte am seriellen Bus ausgeben.
Adresse:	$FFA8
Parameterübergabe:	A

UNTLK

Zweck:	UNTALK über seriellen Bus senden.
Adresse:	$FFAB

UNLSN

Zweck:	UNLISTEN über seriellen Bus senden.
Adresse:	$FFAE

LISTN

Zweck:	Kommando LISTEN an ein Gerät senden.
Adresse:	$FFB1
Parameterübergabe:	A
Anmerkungen:	Geräteadresse in A übergeben.

TALK

Zweck:	Kommando TALK an ein Gerät ausgeben.
Adresse:	$FFB4
Parameterübergabe:	A
Anmerkungen:	Geräteadresse in A übergeben.

READSS

Zweck:	I/O-Statusbyte holen
Adresse:	$FFB7
Parameterübergabe:	A

SETLFS

Zweck:	Fileparameter setzen.
Adresse:	$FFBA
Parameterübergabe:	A, X, Y
Anmerkungen:	logische Adresse in A, Geräteadresse in X und Sekundäradresse in Y.

SETNAM

Zweck:	Filenamenparameter setzen.
Adresse:	$FFBD
Parameterübergabe:	A, X, Y
Anmerkungen:	Länge des Filenamens in A, Adresse des Filenamens in X (low) und Y (high).

OPEN

Zweck:	Ein logisches File öffnen.
Adresse:	$FFC0
Anmerkungen:	Vorbereitungsroutinen SETNAM und SETLFS.

CLOSE

Zweck:	Ein logisches File schließen.
Adresse:	$FFC3
Parameterübergabe:	A
Anmerkungen:	logische Filenummer des zu schließenden Files in A.

CHKIN

Zweck:	Kanal als Eingabekanal festlegen.
Adresse:	$FFC6
Parameterübergabe:	X
Anmerkungen:	Logische Filenummer in X übergeben.

CKOUT

Zweck:	Kanal als Ausgabekanal festlegen.
Adresse:	$FFC9
Parameterübergabe:	X
Anmerkungen:	Logische Filenummer in X übergeben.

CLRCH

Zweck:	Ein-/Ausgabekanal schließen.
Adresse:	$FFCC

BASIN

Zweck:	Ein Byte von einem Kanal lesen.
Adresse:	$FFCF
Parameterübergabe:	A

BSOUT

Zweck:	Ausgabe eines Bytes auf einen Kanal (Default = Bildschirm).
Adresse:	$FFD2
Parameterübergabe:	A

LOADSP

Zweck:	Laden von Files.
Adresse:	$FFD5
Parameterübergabe:	X, Y
Anmerkungen:	Adresse, ab der das zu ladende File abgelegt werden soll, in X und Y.

SAVESP

Zweck:	Abspeichern auf Files.
Adresse:	$FFD8
Parameterübergabe:	A, X, Y und ein Zeiger in der Zeropage
Anmerkungen:	X und Y enthalten die End-Adresse, A die Zero-page-Adresse, an der der Zeiger auf die Startadresse des abzuspeichernden Bereiches steht.

SETTIM

Zweck:	Setzt die 3 Bytes der Systemuhr.
Adresse:	$FFDB
Parameterübergabe:	A, X, Y
Anmerkungen:	Höchstwertiges Byte in Y.

RDTIM

Zweck:	Holt die 3 Bytes der Systemuhr.
Adresse:	$FFDE
Parameterübergabe:	A, X, Y
Anmerkungen:	Höchstwertiges Byte in Y.

STOP

Zweck:	Stop-Taste abfragen.
Adresse:	$FFE1
Parameterübergabe:	A, Zero-Flag
Anmerkungen:	Wurde während des letzten UDTIM-Aufrufes die Stop-Taste gedrückt, wird das Zero-Flag gesetzt.

GETIN

Zweck:	Holt ein Zeichen vom Tastaturpuffer, Bildschirm oder RS232.
Adresse:	$FF4E
Parameterübergabe:	A

CLALL

Zweck:	Alle offenen Files schließen, Defaultwerte für Ein-/Ausgabegeräte setzen
Adresse:	$FFE7

UDTIM

Zweck:	Systemuhr updaten.
Adresse:	$FFEA

SCRORG

Zweck:	Größe des aktuellen Bildschirms holen.
Adresse:	$FFED
Parameterübergabe:	X, Y.

PLOT

Zweck:	Cursorposition setzen/holen.
Adresse:	$FFF0
Parameterübergabe:	X, Y, Carry.
Anmerkungen:	Gesetztes Carry-Flag bewirkt ein Holen der Cursorposition.

IOBASE

Zweck:	Holt Basisadresse des I/O-Bereiches.
Adresse:	$FFF3
Parameterübergabe:	X, Y
Anmerkungen:	Basisadresse wird in X (low) und Y (high) übergeben.

Stichwortverzeichnis

Protext

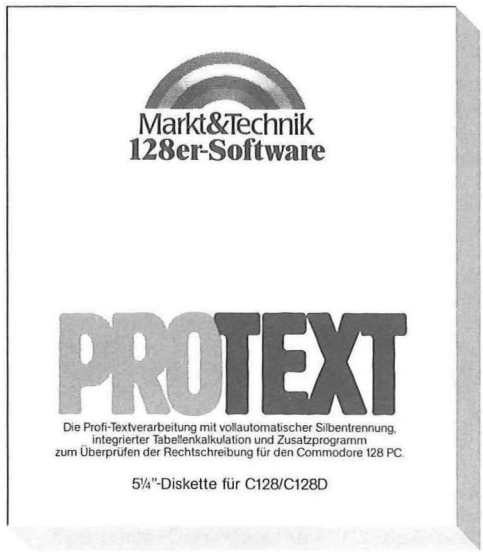
Die Profi-Textverarbeitung mit vollautomatischer Silbentrennung, integrierter Tabellenkalkulation und Zusatzprogramm zum Überprüfen der Rechtschreibung für den Commodore 128 PC.

Protext ist ein leicht bedienbares Textprogramm mit hoher Leistungsfähigkeit. Mit **Protext** sind daher auch Anfänger in der Lage, alle Vorteile eines professionellen Textprogramms zu nutzen. Überzeugen Sie sich selbst!

Was Protext alles kann:

- formatierte Ausgabe auf Bildschirm und Drucker mit programmierbaren Haltepunkten über serielle, V24- oder zwei Software-Centronics-Schnittstellen;
- vielfältige Formatanweisungen
- schnelle selbstlernende Textkorrektur mit deutschem (ca. 25000 Worte) Grundwortschatz sowie neun Kundenbibliotheken, die in Text umgewandelt, bearbeitet, ergänzt, sortiert und ausgedruckt werden können;
- Textübertragung per DFÜ;
- leistungsfähige Rechenmöglichkeiten mit Zeilenmarkierung (Rechentabulator), Kolonnenverarbeitung,

Hardware-Anforderungen: C128 oder C128D, 80-Zeichen-Monitor, Commodore-Drucker oder Drucker mit Centronics-Schnittstelle.



Bestell-Nr. 51254

DM 89,–*

(sFr 81,90/öS 718,–*)

* inkl. MwSt. Unverbindliche Preisempfehlung

Und dazu die weiterführende Literatur:

R. Schineis/H. Thies
Textverarbeitung mit Protext für den C128 PC.
1986, 258 Seiten
Eine systematische Einführung in die Bedienung von Protext. Druckersteuerung, Serienbrief, Tabellenkalkulation. Inklusive Druckertreiberdiskette.

Bestell-Nr. 90375
ISBN 3-89090-375-4
DM 39,–*
(sFr 35,90/öS 304,20*)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

704325

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Version 2.41

dBASE II

für Commodore 128/128 D

dBASE II, das meistverkaufte Programm unter den Datenbanksystemen, gibt es jetzt im CP/M-Modus für den C 128. Es eröffnet Ihnen optimale Möglichkeiten der Daten- und Dateihandhabung. Einfach und schnell können Datenstrukturen definiert, benutzt und geändert werden. Der Datenzugriff erfolgt sequentiell oder nach frei wählbaren Kriterien, die integrierte Kommandosprache ermöglicht den Aufbau kompletter Anwendungen wie Finanzbuchhaltung, Lagerverwaltung, Betriebsabrechnung usw.

Lieferumfang:
- Originalhandbuch von Ashton-Tate
- Beschreibung der Commodore-128-PC-spezifischen Version

Hardware-Anforderungen:
Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle über Userport

Markt&Technik
128er-Software

dBASE™ II
ASHTON-TATE
für den Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Bestell-Nr. 50303

DM 199,–

(sFr 178,-/öS 1890,-*)
inkl. MwSt. Unverbindliche Preisempfehlung

Und dazu die weiterführende Literatur:

Dr. P. Albrecht
dBASE II für den Commodore 128 PC
Dieses klassische Einführungs- und Nachschlagewerk begleitet Sie mit nützlichen Hinweisen bei Ihrer täglichen Arbeit mit dBASE II.
Bestell-Nr. 90189,
ISBN 3-89090-189-1
DM 49,–
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

704322

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Multiplan

Version 1.06

für Commodore 128/128 D

Wenn Sie die zeitraubende manuelle Verwaltung tabellarischer Aufstellungen mit Bleistift, Radiergummi und Rechenmaschine satt haben, dann ist MULTIPLAN, das System zur Bearbeitung »elektronischer Datenblätter«, genau das richtige für Sie! Das benutzerfreundliche und leistungsfähige Tabellenkalkulationsprogramm kann bei allen Analyse- und Planungsberechnungen eingesetzt werden wie zum Beispiel Budgetplanungen, Produktkalkulationen, Personalkosten usw. Spezielle Formatierungs-, Aufbereitungs- und Druckanweisungen ermöglichen außerdem optimal aufbereitete Präsentationsunterlagen!

5¼"-Diskette für den Commodore 128 PC.

Hardware-Anforderungen: Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle

Markt&Technik
128er-Software

MICROSOFT
MULTIPLAN
für den
Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Bestell-Nr. 50203

DM 199,–*

(sFr 178/öS 1890,–*)

* inkl. MwSt. Unverbindliche Preisempfehlung

Und dazu die weiterführende Literatur:

Dr. P. Albrecht
Multiplan für den Commodore 128 PC
1985, 226 Seiten
Mit diesem Buch werden Sie Ihre Tabellenkalkulation ohne Probleme in den Griff bekommen. Als Nachschlagewerk leistet es auch dem Profi nützliche Dienste.
Bestell-Nr. MT 836
ISBN 3-89090-187-5
DM 49,–
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

705323

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bücher zum Commodore 64/128

S. Vilsmeier
3D-Konstruktion mit GIGA-CAD Plus auf dem C64/C128
1986, 370 Seiten, inkl. 2 Disk.
Mit GIGA-CAD können Computergrafiken von besonderer Räumlichkeit und Faszination geschaffen werden. GIGA-CAD Plus ist schneller und einfacher zu bedienen, die Benutzeroberfläche wurde verbessert und der Befehlssatz erweitert. Die Eingabe erfolgt in erster Linie über den Joystick. Hardware-Anforderung: C64 mit Floppy 1541 oder C128 (im 64'er-Modus), Fernseher oder Monitor, Joystick und Commodore- oder Epson-kompatibler Drucker.
● Das verbesserte GIGA-CAD-Programm mit neuen Features wie erweitertem Befehlssatz und bis zu 10mal schneller liegt dem Buch im Floppy-1541-Format bei.
Best.-Nr. 90409
ISBN 3-89090-409-2
DM 49,–
(sFr 45,10/öS 382,20)

H. Haberl
Mini-CAD mit Hi-Eddi plus auf dem C64/C128
1986, 230 Seiten, inkl. Diskette
Auf der beiliegenden Diskette findet der Leser das vollständige Zeichenprogramm »Hi-Eddi«, mit dem das komfortable Erstellen von technischen Zeichnungen, Plänen oder Diagrammen ebenso möglich ist wie das Malen von farbigen Bildern, Entwurf und Ausdruck von Glückwunschkarten, Schildern, ja sogar von bewegten Sequenzen (kleine Trickfilme, Schaufenster-Werbung).
● Wer sagt, daß CAD auf dem C64 nicht möglich ist?!
Best.-Nr. 90136
ISBN 3-89090-136-0
DM 48,–
(sFr 44,20/öS 374,40)

B. Bornemann-Jeske
Vizawrite-Buch für den C64/C128
1987, 228 Seiten
Mit dem »Vizawrite-Buch« liegt erstmals ein vollständiges und detailliertes Arbeitsbuch für den Anfänger und den professionellen Anwender zur Textverarbeitung auf dem C64/C128 vor. Die Grundlagenkapitel führen Sie anhand kurzer Übungsaufgaben in die elementaren Funktionen des Systems ein. Das Kapitel für Fortgeschrittene zeigt Ihnen jede Programmfunktion im Detail. Zahlreiche praktische Tips aus verschiedenen Anwendungsbereichen ermöglichen Ihnen die optimale Nutzung Ihres Textverarbeitungssystems.
Best.-Nr. 90231
ISBN 3-89090-231-6
DM 49,–
(sFr 45,10/öS 382,20)

O. Hartwig
Experimente zur Künstlichen Intelligenz mit C64/C128
1987, 248 Seiten
Sind Maschinen intelligent? Können Computer denken? Erschließen Sie sich eines der interessantesten Gebiete der modernen Computerforschung! Anhand zahlreicher Programme erfahren Sie hier die Möglichkeiten der Künstlichen Intelligenz, speziell auf dem C64 und dem C128. Der Schwerpunkt des Buches liegt auf der Praxis. Alle KI-Techniken werden durch anschauliche Programme vorgestellt, die sofort nachvollziehbar sind. Zusätzlich erhalten Sie jede Menge Anregungen zu eigenen Experimenten. Die KI-Programme können ohne weiteres in eigene Programme integriert werden.
Best.-Nr. 90472
ISBN 3-89090-472-6
DM 49,–
(sFr 45,10/öS 382,20)

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

706358

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0