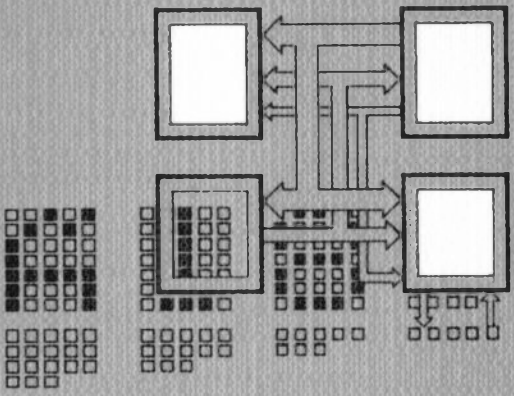
Das kleine Praktikum

Feichtinger
Basic für Mikrocomputer

Geräte – Begriffe – Befehle – Programme



F'

Franzis'

Feichtinger
Basic für Mikrocomputer

Das kleine Praktikum

Herwig Feichtinger

Basic für Mikrocomputer

Geräte – Begriffe – Befehle – Programme

Mit 40 Abbildungen

Franzis'

CIP-Kurztitelaufnahme der Deutschen Bilbliothek

Feichtinger, Herwig:
Basic für Mikrocomputer: Geräte, Begriffe, Befehle, Programme/Herwig Feichtinger. – München: Franzis, 1980.
(Das kleine Praktikum)
ISBN 3-7723-6821-2

Druck: Franzis-Druck GmbH, Karlstraße 35, 8000 München 2
Printed in Germany. Imprimé en Allemagne.

ISBN 3-7723-6821-2

Vorwort

Noch vor wenigen Jahren konnten sich nur mittlere und große Betriebe den Luxus eines Computers leisten. Heute haben Mikrocomputer-Systeme zwischen 1000 und 10 000 DM zwar entgegen kühner Prognosen nicht in jedem Haushalt Einzug gehalten; sie fanden aber wegen ihres relativ geringen Preises und der einfach zu erlernenden Programmiersprache "Basic" in kleineren Betrieben und auch im Privatbereich große Verbreitung. Auch Großfirmen haben die Vorteile solcher "Personal Computer" erkannt und nützen sie für eine dezentralisierte Datenverarbeitung oder auch als Ersatz für umfangreiche "Hardware"-Steuerungen, z.B. in der Meßtechnik.

Während die etablierten Computerhersteller ihre Produkte noch Anfang der siebziger Jahre mit einer Art Mystik umgaben, haben sie sich inzwischen dem Beispiel der zunächst relativ kleinen Mikrocomputer-Hersteller angeschlossen. Und so entstanden auch in den Labors von Siemens, IBM, Hewlett-Packard und Triumph-Adler gegen Ende des letzten Jahrzehnts Geräte der unteren Preisklasse, die sogar für den Hobbyisten erschwinglich sind.

Eine besondere Schwierigkeit für den Anwender ist, daß sich Eigenschaften und Befehlsvorrat der heute angebotenen Mikrocomputer ganz erheblich unterscheiden. Auch ist die dem Gerät beigepackte Dokumentation manchmal etwas mangelhaft. Das vorliegende Buch stellt sich daher die Aufgabe, dem Computer-Interessenten einen objektiven Vergleich heute üblicher Systeme zu ermöglichen, den Anwender etwas tiefer in seinen Computer hineinsehen zu lassen und Sprachschwierigkeiten durch lexikon-artige Kapitel über Fachausdrücke und Basic-Befehle zu vermeiden. Auf eine ausführliche Darstellung der Maschinenprogrammierung wurde hier absichtlich verzichtet, da sie zu systemabhängig ist; stattdessen wird auf Literatur hierüber im gleichen Verlag verwiesen.

Herwig Feichtinger

Inhalt

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden*).
Alle Schaltungen und technischen Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag sieht sich deshalb gezwungen, darauf hinzuweisen, daß er weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen kann. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

*) Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenzinhabers einzuholen.

1 Einführung

1.1 Computer überall . . .

Kaum eine technische Errungenschaft hat unser Dasein wohl so verändert wie der Computer – insbesondere, seit es solche Geräte zu Preisen gibt, die für jedermann erschwinglich sind.

Eine zwingende Notwendigkeit für die Verbreitung von Computern ist das Vorhandensein einer Möglichkeit, diesem "schnellen und fleißigen, aber leider strohdummen Vollidioten" auf eine auch für Nichtspezialisten leicht erlernbare Weise beizubringen, was er gefälligst zu tun hat und was der Benutzer von ihm will.

Diese Voraussetzung schaffen Programmiersprachen, die einen einfachen, überschaubaren und doch für typische Anwender ausreichend leistungsfähigen Wortschatz enthalten, der – das sei dem Computer verziehen – der englischen Sprache entnommen ist (schließlich wollen die Computerhersteller ihre inzwischen zu Massenartikeln gewordenen Produkte nicht nur in Deutschland verkaufen!).

Die nach einhelliger Meinung aller Computerhersteller am leichtesten beherrschbare Programmiersprache ist Basic – eine Abkürzung von "Beginner's All-Purpose Symbolic Instruction Code", also "Universelle Programmiersprache für Anfänger". Diese Bezeichnung, die sich John Kemeny und Thomas Kurtz um 1962 für die von ihnen entwickelte Sprache ausgedacht haben, ist etwas untertrieben, denn mit den komfortablen Befehlen moderner Basic-Versionen können nicht nur Anfänger etwas "anfangen".

Der Vorteil der leichten Erlernbarkeit von Basic wird durch eine Eigenschaft ergänzt, die allen höheren Programmiersprachen wie Fortran, Algol, Pascal usw. anhaftet: Die Sprache ist nicht systemabhängig, sondern ein in Basic geschriebenes Programm läuft auf allen Computern, die Basic verstehen. Damit

ist im Gegensatz zur hexadezimalen Maschinenprogrammierung die Austauschbarkeit von Programmen weitgehend gewährleistet.

Wie einfach der Umfang mit Computern heute ist, wollen wir uns an einem einfachen Beispiel ansehen. Und es wird dann ein wenig verständlicher, warum heute Ärzte, Geschäftsleute, Versicherungsvertreter oder Bankangestellte, aber auch Computer-Hobbyisten ihre Programme selbst schreiben können und so die ständig steigenden Kosten für die Software-Entwicklung umgehen.

Vorher aber noch eine Bemerkung zu den unterschiedlichen Mikrocomputer-Typen, die heute den Markt bestimmen. Sie lassen sich in folgende Gruppen einteilen:

Haushalt-Computer:
Computer mit meist fertig beziehbarer Software für Heimanwendungen mit extrem einfacher Bedienung (z.B. TI 99/4).

Hobby-Computer:
Preiswerte, aber flexible Mikrocomputer mit Ausbaumöglichkeiten und Anschlüssen zur Steuerung externer Geräte, z.B. einer Modelleisenbahn (z.B. AIM-65, PC-100).

Small-Business-Computer:
Mikrocomputer mit hoher Speicherkapazität und professionellem Aufbau für den Einsatz in Büros für Adressen- und Lagerverwaltung usw. (z.B. CBM-3032).

IEC-Bus-Computer:
Mikrocomputer mit IEC-Bus für den Anschluß von fernbedienbaren Meßgeräten (Digitalvoltmeter, Frequenzzähler usw.) zur automatisierten Meßwertverarbeitung (z.B. PET-2001, CBM-3016).

Diese vier Gruppen faßt man unter der englischen Bezeichnung "Personal Computer" zusammen. Außerdem dienen Mikrocomputer auch noch zur Steuerung irgendwelcher Geräte, wie Fernseh- und Rundfunkempfänger, Mikrowellen-

öfen, Zündungen in Kfz-Motoren u.v.a. oder entlasten in "intelligenten" Peripheriegeräten (Drucker, Terminals usw.) eine größere EDV-Anlage von Routineaufgaben.

Inzwischen arbeiten kleine EDV-Anlagen bereits mit Mikroprozessoren als Zentraleinheit, wobei der Begriff "Personal Computer" schon aus Preisgründen mit Vorsicht zu gebrauchen ist.

Doch zurück zur Programmierung – sehen wir uns einmal an, wie man mit der heute überwiegend verwendeten Programmiersprache Basic umgeht.

1.2 So einfach ist Basic

Um eine Vorstellung davon zu bekommen, wie man mit der Programmiersprache Basic arbeitet, schalten wir – falls vorhanden – unseren Basic-Computer ein und warten ab, was passiert. Üblicherweise erscheint auf dem Bildschirm so etwas Ähnliches wie

```
3566 BYTES FREE
```

womit uns der Computer anzeigt, daß er genügend Speicherplatz besitzt, damit wir ihm etwas eingeben können. Es ist naheliegend, auszuprobieren, was der Computer sagt, wenn wir ihm irgendeinen Befehl nach Gutdünken eingeben, z.B. DRUCKE. (Damit er weiß, daß wir außer DRUCKE nichts mehr eingeben wollen, müssen wir noch die mit RETURN bezeichnete Taste betätigen). Der "Erfolg" stellt sich prompt ein; es erscheint so etwas wie

```
SYNTAX ERROR
```

womit uns gesagt wird, daß unser Wunsch nicht verstanden wurde. Hier sehen wir schon eine recht nützliche Eigenschaft von Basic: Wenn wir etwas falsch machen, wird uns das auch gesagt. Wenn wir uns erinnern, daß der Computer nur Englisch versteht, können wir es noch einmal mit PRINT versuchen. Und siehe da: Auf dem Bildschirm erscheint keine Fehlermeldung,

allerdings sonst auch nichts, bestenfalls READY; damit sagt uns der Computer, daß er etwas getan hat, womit er jetzt fertig ist.

Kein Wunder, daß nichts ausgedruckt wurde: Wir müssen ja dem Computer auch sagen, was er drucken soll, zum Beispiel die Summe von 523 und 192. Versuchen wir's nochmal:

```
PRINT 523+192
```

Sobald wir jetzt die Return-Taste drücken, erscheint auf dem Bildschirm die Zahl 715, und dann vielleicht wieder READY (das ist vom Rechnertyp abhängig). Wenn wir dagegen nur 523+192 eintippen, also ohne PRINT, so sagt der Computer gewöhnlich wieder schlicht SYNTAX ERROR – weil er nicht weiß, was er mit dem Ergebnis anfangen soll.

Wir halten fest: Man muß einem Rechner immer haarklein zerpflücken, was man von ihm will. Die geringste Unlogik in der Eingabe führt zum Protest unseres Silizium-Gehirns.

Was wir gerade eingetippt haben, hat mit einem "Programm" noch nichts zu tun, wir haben auch nicht "programmiert". Vielmehr haben wir eine Eigenschaft von Basic ausgenützt, die sonst nur wenige Computersprachen bieten: den Direkt-Modus. Befehle, die so eingegeben werden, wie wir das gerade getan haben, werden sofort ausgeführt und nicht weiter gespeichert.

Anders ist die Situation, wenn wir vor jeden Befehl eine Nummer schreiben, die der Rechner als Programm-Zeilennummer interpretiert. Ganz schlaue Leute nehmen dabei nicht aufeinanderfolgende Zahlen, sondern staffeln die Zeilennummern z.B. in Zehnerschritten, damit man bei eventuell erforderlichen Änderungen noch Platz hat, zwischen zwei Befehle einen neuen einzufügen. Probieren wir mal aus, was jetzt passiert:

```
10 PRINT 523+192
```

Diesmal schweigt der Computer nach dem Drücken der Return-Taste. Wenn wir aber RUN eintippen und nochmals Return drücken, schreibt er brav 715 auf den Schirm. Mit RUN haben wir jetzt nämlich unser aus nur einer Zeile mit der Nummer 10 bestehendes Einfachst-Programm aufgerufen. Wir können es

mit RUN beliebig oft ausführen, und jedesmal wird 715 ausgedruckt (wenn nicht, packen wir den Computer am besten wieder ein, legen die Garantiekarte dazu und tragen ihn zum nächsten Postamt).

Natürlich ist das auf Dauer unbefriedigend. Es wäre schön, wenn unser teurer Rechner zumindest als einfache Addiermaschine zu gebrauchen wäre. Dazu benötigen wir aber schon ein etwas größeres Programm, das auch zeigt, wie man Werte eingeben kann, die für die Rechnung gebraucht werden:

```
10 A = 0
20 INPUT B
30 A=A+B
40 PRINT A
50 GOTO 20
```

Zeile 30 ist für eingefleischte Mathematiker etwas Grauenvolles: Hier wird das Gleichheitszeichen für etwas mißbraucht, das mit Gleichheit absolut nichts zu tun hat. In Basic bedeutet A=A+B nicht etwa, daß die numerischen Werte, repräsentiert durch Variablennamen, auf den beiden Seiten des "="-Zeichens gleich sind. Vielmehr wird damit der Rechner angewiesen, ab sofort nicht mehr den ursprünglichen Wert von A zu verwenden, sondern durch das zu ersetzen, was rechts von "=" steht.

Beim Start des Programmes mit RUN erscheint ein Fragezeichen auf dem Bildschirm, das uns anzeigt, daß wir jetzt etwas eingeben können – nämlich unseren ersten Summanden. Damit der Rechner weiß, wann wir mit der Eingabe fertig sind, müssen wir dann wieder Return drücken. Sofort erscheint die bisherige Summe; sie ist natürlich noch mit dem ersten Summanden identisch. Und wieder erscheint ein Fragezeichen; jetzt geben wir den nächsten Summanden ein – und die neue Summe erscheint.

Menschen sind keine Computer und pflegen sich zu vertippen. Computer wissen das und erlauben dem Menschen daher, seine Irrtümer zu korrigieren, solange er nicht "Return" gedrückt hat: Wenn man die mit einem nach links weisenden Pfeil bezeichnete Taste drückt (manchmal heißt sie auch

"Delete" oder "Backspace"), wandert jenes meist blinkende Etwas, das die aktuelle Schreibposition markiert und von Fachleuten "Cursor" tituliert wird, um ein Zeichen nach links. Jetzt kann man das falsch Eingegebene korrigieren, indem man es einfach überschreibt. Übrigens läßt sich der Cursor meist auch in alle anderen Richtungen bewegen, falls dies einmal erforderlich werden sollte.

Wollen wir das Programm anhalten, um es mit dem Wert Null für A neu starten oder ändern zu können, drücken wir die mit STOP oder BREAK bezeichnete Taste. Sie ist u.U. allerdings wirkungslos, wenn sich das Programm gerade im Eingabe-Modus (Zeile 20) befindet; hier ist ein "Aussteigen" aber auch möglich, indem man einfach die Taste Return gleich nach dem Erscheinen des Fragezeichens drückt.

Wenn wir uns das Programm noch einmal ansehen wollen, so brauchen wir nur LIST (gefolgt von Return) eingeben. Das Löschen einer Zeile ist möglich, indem man ihre Nummer und anschließend sofort Return eingibt. Das gesamte Programm können wir mit NEW löschen – um dieses Primitivprogramm ist es ja auch nicht allzu schade. . .

Bevor wir uns mit der Arbeitsweise unseres kleinen Wunderwerks näher befassen, noch eine Bemerkung über das Format der Zahlendarstellung.

Wie wir das von Taschenrechnern sicher schon gewöhnt sind, arbeiten Computer nicht mit einem Dezimal-Komma, sondern mit einem Dezimal-Punkt. Wenn es sich um eine Zahl kleiner 1 handelt, so muß links neben diesem Punkt nicht unbedingt eine Null stehen. Also:

Deutsche Schreibweise:	0,3582
Computer-Format:	.3582

Auch die Darstellung sehr kleiner oder sehr großer Zahlen löst der Computer ebenso wie gängige Taschenrechner, nämlich mit Hilfe eines dezimalen Exponenten. So wird die Zahl 0,000 000 001 auf dem Bildschirm in der Form 1 E-9 angezeigt. Noch ein paar Beispiele:

Deutsche Schreibweise	*Computer-Format*
12387653486	1.2387653 E 10
$-1{,}8 \cdot 10^{23}$	−1.8 E 23
$4{,}7 \cdot 10^{3}$	4700
$5.34 \cdot 10^{-2}$	0.0534

Das Exponentialformat wählt der Computer automatisch immer dann, wenn sich eine Zahl nicht mehr mit der rechnerintern verfügbaren Stellenzahl darstellen läßt.

Für die Eingabe sehr großer oder sehr kleiner Zahlen verwendet man die gleiche Zeichenfolge, wie sie der Computer ausgeben würde; für 123000000 gibt man also ein: 1.23 E 8. Zulässig wäre auch eine Eingabe in der Form 123 E 6.

Im Vergleich zu den heutigen Taschenrechnern sind Basic-Computer meist nicht gerade Rechenkünstler. Das betrifft nicht nur die zur Verfügung stehenden mathematischen Funktionen, sondern auch Rechengenauigkeit und zulässigen Zahlenbereich: Kaum einem Computer darf man so etwas wie 1 E 99 eingeben. Sehr kleine Zahlen, wie 1E-99, setzt er schlicht und einfach gleich Null.

2 Arbeitsweise von Mikrocomputern

2.1 Hardware-Bausteine eines Mikrocomputers

Programme lassen sich wesentlich effizienter schreiben, wenn man die Hardware seines Computers kennt, jene Bausteine und Bauelemente also, die ihm das "Denken" ermöglichen. Sehen wir uns also einmal ganz grob an, wie ein solches Wunderwerk funktioniert *(Abb. 2.1.1)*.

Zwei wichtige Bestandteile unseres Computers sind bereits zu sehen, ohne daß wir das Gehäuse öffnen müssen, nämlich die Tastatur und das Display zur Daten-Ein- und -Ausgabe. Das Wichtigste ist dagegen im Inneren verborgen und zudem recht klein: Der Mikroprozessor, auch CPU (Central Processing Unit) genannt. Dabei handelt es sich um ein IC (Integrated Circuit, Integrierte Schaltung) mit meist 40 Anschlüssen, das als einziges Bauelement des Computers eine gewisse Intelligenz besitzt. Die CPU kann zum Beispiel zwei zweistellige Dezimalzahlen addieren oder subtrahieren – aber das ist, wenn man

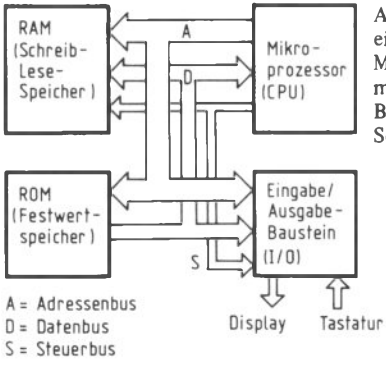


Abb. 2.1.1 Grundsätzlich besteht ein Mikrocomputer aus dem Mikroprozessor, einem ROM mit dem Betriebsprogramm (z.B. Basic-Interpreter) und einem Schreib-Lese-Speicher

von hochwertigen 16-bit-Prozessoren einmal absieht, schon alles, was man ihr an Mathematik zumuten kann.

Ohne drei weitere Bausteine unseres Computersystems ist der Mikroprozessor unfähig, auch nur das Geringste zu leisten. Wir brauchen nämlich noch einen Festwertspeicher (ROM, Read-Only Memory), der ein in der primitiven Maschinensprache geschriebenes Programm enthält, das der CPU sagt, was sie überhaupt zu tun hat; wir brauchen einen Schreib-Lese-Speicher, in dem der Prozessor Zwischenergebnisse ablegen kann (RAM = Random Access Memory, Speicher mit wahlfreiem Zugriff), und schließlich auch einen Baustein, der es dem "Mikroprofessor" ermöglicht, mit der übrigen Welt, sprich z.B. mit der Tastatur, in Verbindung zu treten, weil ein Computer ja meist kein Selbstzweck ist.

Alle diese Bausteine sind mit der CPU über eine Anzahl von Leitungen verbunden, die sich in drei Gruppen einordnen lassen. Der Adressenbus (meist 16 Leitungen) gestattet es dem Prozessor, auf eine ganz bestimmte Speicherzelle im ROM oder RAM oder auf einen Eingabe-Ausgabe-Baustein (I/O = Input/Output) gezielt zuzugreifen. Zu diesem Zweck ist jeder Speicherzelle eine Adresse zugeordnet, durch die sie eindeutig charakterisiert wird.

Wenn die CPU also zum Beispiel die Adresse 9724 auf den Adressenbus gibt, so weiß der Baustein, z.B. ein ROM, zu dem diese Adresse gehört, daß er gemeint ist und gibt die in der entsprechenden Speicherzelle stehenden Daten auf eine zweite Gruppe von Leitungen, den Datenbus. Er umfaßt (bei 8-bit-CPUs) acht Leitungen. Die CPU kann die über den Datenbus empfangenen Daten jetzt z.B. als Programmbefehl interpretieren, in ihr enthaltene Daten im RAM abzuspeichern.

In diesem Fall legt die CPU die Adresse einer geeigneten RAM-Speicherzelle auf den Adressenbus und die abzuspeichernden Daten auf den Datenbus. Damit der RAM-Baustein weiß, daß er nicht Daten abgeben, sondern aufnehmen soll, tritt jetzt eine letzte Gruppe von Leitungen in Aktion (meist nur 2...4 Leitungen, der Steuerbus) und teilt dem RAM-IC mit, daß die CPU eine Information abzuspeichern wünscht.

Wenn das geschehen ist, holt sich der Mikroprozessor den nächsten Befehl aus dem ROM – Schritt für Schritt wird das fest gespeicherte Programm abgearbeitet. Dazu gehört auch die Tastaturabfrage über einen I/O-Baustein und die Anzeige von Ergebnissen auf dem Display des Computers.

Woher weiß der Prozessor aber, was er tun soll, wenn der Computer gerade erst eingeschaltet worden ist? Auch dieses Problem wurde gelöst. Die CPU verfügt zu diesem Zweck über einen Rücksetz-Anschluß (Reset). Eine kleine Logikschaltung sorgt dafür, daß beim Einschalten an diesem Anschluß ein kurzer Impuls erzeugt wird, der den Prozessor veranlaßt, die Programmausführung an einer ganz bestimmten (per Hardware festgelegten) Adresse zu beginnen. Und erst jetzt ist sich der Computer wieder selbst überlassen. Auch während des Betriebes führt ein Reset (manchmal ist eine Taste dafür vorhanden) dazu, daß das Programm an dieser Stelle neu begonnen wird. Basic-Computer pflegen an der entsprechenden Adresse im ROM ein Programm zu besitzen, das den gesamten RAM-Bereich und damit alles, was der Benutzer bisher eingegeben hat, löscht. . .

Abb. 2.1.2 Diesem Herrn haben wir die Erfindung des ersten Mikroprozessors, des 4004 von Intel, zu verdanken: Dr. Ted Hoff

Das zweifellos revolutionäre Konzept eines Mikrocomputers haben wir einem gewissen Marcian E. "Ted" Hoff *(Abb. 2.1.2)* zu verdanken, der schon 1969 auf diese Idee kam und sie 1971 mit dem ersten Mikroprozessor, der 4-bit-CPU 4004 von Intel/ Santa Clara, verwirklichte. 1979 wurde er dafür mit dem Preis des Franklin-Institutes ausgezeichnet – wie vor ihm schon Transistor-Erfinder Bardeen und Informationstheoretiker Shannon.

2.2 So arbeiten Basic-Interpreter

Bisher war immer die Rede davon, daß Programme im ROM stehen – in einem Festwertspeicher also, wo sie vom Benutzer nicht geändert werden können. Aber: Wir können unserem Computer doch unsere eigenen Programme eingeben?! Und außerdem arbeiten wir normalerweise ja nicht mit der relativ primitiven Maschinensprache, mit der man nur zweistellig addieren und subtrahieren kann, sondern mit einer höheren Programmiersprache wie Basic, mit der man sogar Sinus-Werte ausrechnen kann!

Das ist kein Widerspruch. In der Tat kann der Mikroprozessor die Befehle, die wir ihm in Basic eingeben, wie PRINT, INPUT usw., nicht unmittelbar ausführen. Beim PRINT-Befehl muß er sich zum Beispiel aus einigen Speicherzellen erst einmal das herbeiholen, was ausgedruckt werden soll, und zum Display transferieren – und dazu ist eine Vielzahl von Maschinenbefehlen notwendig.

Basic-Befehle werden also anders abgearbeitet. Jeder Befehl einer höheren Programmiersprache wird zunächst einmal nicht in seiner vollen Länge gespeichert – das wäre RAM-Verschwendung. Vielmehr sucht sich der Mikroprozessor dafür einen Code heraus, den er intern (und nur intern) für diesen Basic-Befehl benutzt, z.B. 97 für PRINT. Bei der Ausführung des Befehls – sei es direkt oder innerhalb eines Programmes – "interpretiert" die CPU diese Zahl 97 als eine Folge von Maschinenbefehlen, die genau das tun, was wir von PRINT erwarten. Diese Maschinenbefehle und auch das Umcodier-Programm stehen im ROM des

Mikrocomputers. Andere ROM-Programmteile sorgen dafür, daß eine eingegebene Programmzeile ihrer Zeilennummer entsprechend im RAM eingeordnet und später auch in der richtigen Reihenfolge abgearbeitet wird. Die Gesamtheit der ROM-Programmteile, die unser im RAM gespeichertes selbstgeschriebenes Basic-Programm als Folge passender, für die CPU verständlicher Maschinenbefehle ausführt, nennt man "Interpreter".

Wir merken uns: Ein Interpreter ist ein in Maschinensprache geschriebenes Programm, dessen Befehle für die CPU direkt verständlich sind. Es führt jeden Befehl einer höheren Programmiersprache als Folge einfacher Maschinenbefehle aus.

Woher aber weiß der Basic-Interpreter zum Beispiel den Sinus-Wert von 2,536? Natürlich wäre es Speicherverschwendung, für alle höheren mathematischen Funktionen Tabellen zu verwenden, die im ROM gespeichert sind. Deswegen verwenden Basic-Computer wie Taschenrechner Näherungsformeln, die diese Funktionen auf ganz einfache Operationen (letztlich auf die vier Grundrechenarten) zurückführen. Kennt man diese Formeln, so ist es auch möglich, weniger komfortable Basic-Interpreter durch Verwendung kleiner Unterprogramme mit allen benötigten Funktionen nachzurüsten. Die wichtigsten von ihnen sind deshalb im folgenden wiedergegeben:

$$a^x = e^{x \cdot \ln a}$$

$$\lg x = \frac{\ln x}{\ln 10}$$

$$\ln x = 2 \cdot \left(\frac{x-1}{x+1} + \frac{(x-1)^3}{3\,(x+1)^3} + \frac{(x-1)^5}{5\,(x+1)^5} + \ldots \right)$$

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{1 \cdot 2} + \frac{x^3}{1 \cdot 2 \cdot 3} + \frac{x^4}{1 \cdot 2 \cdot 3 \cdot 4} + \ldots$$

$$\sin x = x - \frac{x^3}{1 \cdot 2 \cdot 3} + \frac{x^5}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} - \ldots$$

$$\cos x = 1 - \frac{x^2}{1 \cdot 2} + \frac{x^4}{1 \cdot 2 \cdot 3 \cdot 4} - \ldots$$

$$\tan x = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \ldots$$

$$\cot x = \frac{1}{\tan x}$$

$$\sqrt{x} = 1 - \frac{1}{2}(1-x) - \frac{1}{8}(1-x)^2 - \frac{1}{16}(1-x)^3 - \ldots$$

Die angegebene Formel für die Wurzel funktioniert nur für x-Werte kleiner 1; will man z.B. die Wurzel aus 14,2 ziehen, so rechnet man:

$$\sqrt{14{,}2} = \sqrt{0{,}142} \cdot \sqrt{100} = 0{,}3768 \cdot 10 = 3{,}768.$$

Ähnlich gilt die Tangens-Näherungsformel nur für Winkel kleiner $\pi/2$, so daß auch hier vorherige Wertanpassungen nötig sind.

Selbstverständlich gibt es auch noch andere Reihen-Formeln zur Ermittlung der Werte höherer mathematischer Funktionen [1], z.B. für den Arcussinus, den Arcuscosinus, den Arcustangens usw., die weniger häufig benötigt werden. Je höher die geforderte Rechengenauigkeit ist, desto mehr Glieder der Reihe müssen berücksichtigt werden – allerdings geht das wieder auf Kosten der Rechenzeit. Die meisten Basic-Computer, die z.B. auf acht Stellen genau addieren, subtrahieren, multiplizieren und dividieren können, arbeiten bei Funktionen wie LOG, SIN, COS usw. je nach x-Wert nur noch mit sechs Stellen Genauigkeit: Ein Kompromiß zwischen Geschwindigkeit und Präzision.

---

[1] Nähere Angaben zu solchen Formeln finden sich z.B. in: Bartsch, *Mathematische Formeln*, Buch- und Zeit-Verlagsgesellschaft, Köln; ferner: Gieck, *Technische Formelsammlung*, Selbstverlag, Heilbronn.

Deutlich wird auch hierbei, daß moderne Taschenrechner den Basic-Computern, was das wirkliche Rechnen angeht, haushoch überlegen sind. Die Vorteile des Computers liegen auf anderem Gebiet: Der Dialog mit dem Benutzer ist komfortabler, die Speicherfähigkeit und Rechengeschwindigkeit ist wesentlich größer, und nicht zuletzt kann man mit dem Computer auch Texte verarbeiten.

2.3 Peripheriegeräte

Als Peripherie bezeichnet man alle Geräte, die nicht im Computer selbst untergebracht, aber mit ihm elektrisch verbunden sind, so daß ein Datenaustausch stattfinden kann. Natürlich sind die unterschiedlichen Computer nicht gleich komfortabel ausgestattet, und manchmal ist schon etwas eingebaut, das man bei anderen als Peripherie bezeichnet. Deswegen besprewir in diesem Abschnitt alles, was nicht in unserem Mikrocomputer-Blockschaltbild enthalten war.

2.3.1 Datensichtgeräte

Die notwendigen Einrichtungen zur Kommunikation mit dem Computer sind nicht notwendigerweise im gleichen Gehäuse wie der Rechner selbst untergebracht. So ist etwa beim TRS-80 der Videomonitor ein abgesetztes Gerät, das über eine Videoleitung mit dem im Tastaturgehäuse untergebrachten Mikrocomputer verbunden ist. Die Erzeugung des Videosignals geschieht dabei also im Computer selbst *(Abb. 2.3.1)*.

Ob das Videosignal direkt (0...5 MHz) oder – mittels eines Hf-Modulators – über die Antennenbuchse eines Fernsehempfängers zugeführt wird, ist keineswegs gleichgültig. Die Darstellung von mehr als 48 Zeichen pro Zeile ist bei Verwendung eines Hf-Modulators wegen des erheblichen Schärfeverlustes auf dem Hochfrequenzweg keinesfalls zu empfehlen. Dies gilt ganz besonders für Computer, die über die Möglichkeit verfügen,

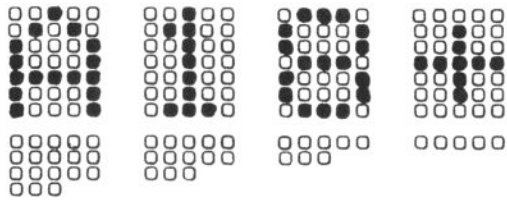


Abb. 2.3.1 Datensichtgeräte und Matrixdrucker setzen jedes Zeichen aus einzelnen Punkten zusammen, z.B. hier als 5x7-Matrix

unterschiedliche Farben auf dem Bildschirm zu erzeugen. Das Vorhandensein eines Video-Anschlusses ist deshalb eine praktisch zwingende Notwendigkeit, wenn man ein Fernsehgerät als Monitor betreiben möchte.

Unter einem Datensichtgerät versteht man aber gewöhnlich nicht nur einen solchen Monitor, sondern eine Anordnung, die mit dem Computer über irgendeine Parallel- oder Seriell-Schnittstelle verkehren kann. Die Erzeugung des Videosignals übernimmt dann das Sichtgerät selbst. Die Daten werden vom Computer codiert in Form einzelner Bits (ASCII) gesendet.

Üblicherweise enthalten Datensichtgeräte auch eine schreibmaschinenähnliche Tastatur, die sich von der deutschen Schreibmaschine vor allem dadurch unterscheidet, daß Y und Z vertauscht sind und alle Sonderzeichen erzeugt werden können, die für ASCII genormt sind. Dies trifft meist nicht für die in den Computern eingebauten Tastaturen zu. Auch entspricht der Zeichencode in Computern mit eingebauter Tastatur und eingebautem Bildschirm oft nicht der ASCII-Norm (PET-2001, CBM-3032, TRS-80 sind Beispiele hierfür). Dieser Nachteil resultiert oft daraus, daß statt der Kleinbuchstaben Grafikzeichen erzeugt werden.

2.3.2 Kassettenrecorder

Wenn man ein Programm mühsam eingetippt hat, möchte man es irgendwie fest abspeichern, ohne daß es beim Ausschalten

des Computers verloren geht. Die preiswerteste Möglichkeit hierzu ist die Aufzeichnung auf gewöhnliche Tonband-Compactcassetten. Die meisten Computer enthalten bereits die notwendige Elektronik, um an einen preiswerten Kassettenrecorder angeschlossen werden zu können. Die Daten werden dabei durch Umtasten zwischen zwei Tonfrequenzen innerhalb des Übertragungs-Frequenzbereiches des Recorders aufgezeichnet, z.B. nach dem Kansas-City-Standard. Der Nachteil dieses Verfahrens ist die relativ geringe Geschwindigkeit. Selbst bei 1200 bit/s – was schon recht viel ist – benötigt man rund 75 Sekunden für 8 KByte, so daß dieses Speichermedium nur für geringere Datenmengen in Frage kommt. Wegen seiner Preiswürdigkeit fand es trotzdem große Verbreitung, besonders bei den "Personal Computern", bei denen allein ein Floppy-Disk-Laufwerk schon halb so viel wie der Computer kosten würde.

Es empfiehlt sich nur die Verwendung mechanisch hochwertiger Kassetten, da schon ein kurzzeitiger Gleichlauffehler zu einer Fehlermeldung führen kann. Wenn die Fehlermeldung schon kurz nach Beginn der Aufzeichnung eintritt, muß man praktisch das gesamte Programm von Hand eingeben. Aber auch, wenn die Fehlermeldung erst am Ende ausgegeben wird, ist dies keine Gewähr dafür, daß der erste Teil des Programms fehlerfrei gelesen wurde, weil ja auch die der Fehlererkennung dienende Prüfsumme erst am Schluß aufgezeichnet wird.

Erfahrungsgemäß bringt die Verwendung von Chromdioxid-Kassetten gegenüber Eisenoxidbändern keinerlei Vorteile, sondern nur den Nachteil eines schnelleren Tonkopf-Abriebs im Recorder. Diesen Kopf sollte man übrigens etwa einmal im Monat mit einem in Spiritus getauchten Wattebausch säubern.

Der Tonkopf ist auch meist dafür verantwortlich, daß eine Kassette, die mit einem anderen Recorder bespielt wurde, nicht fehlerfrei gelesen werden kann. Es empfiehlt sich dann eine Korrektur der Kopf-Taumelschraube; wenn man die Aufzeichnung im Lautsprecher mithören kann, sollte dies einfach auf maximale Höhenwiedergabe geschehen. Ohne Mithörmöglichkeit ist die Einstellung nicht zu empfehlen, da dann die Gefahr besteht, daß man die richtige Position nicht mehr findet.

2.3.3 Floppy-Disk

Für den schnellen Zugriff auf größere Datenmengen und lange Programme stellt die Floppy-Disk die ideale Alternative zum Kassettenrecorder dar. Das Einlesen von einigen KByte dauert nur Sekunden. Mit Doppel-Floppy-Laufwerken ist auch das Kopieren von einer Floppy auf die andere leicht durchführbar. Alle Aktivitäten der Laufwerke können durch den Computer per Programm gesteuert werden *(Abb. 2.3.3.1)*.

Eine Floppy-Disk, auch schlicht Diskette genannt, ist eine dünne, flexible, magnetbeschichtete Kunststoffscheibe, die im Laufwerk mit typ. 360 U/min rotiert. Die Daten sind auf der Diskette in konzentrischen Spuren aufgezeichnet. Während bei den großen Magnetplattenspeichern für jede Spur ein Schreib/Lese-Kopf vorhanden ist, besitzt das Floppy-Laufwerk nur einen Kopf, der mittels eines Schrittmotors auf die gerade interessierende Spur geführt wird. Durch diesen Kopf-Hinlauf bedingt beträgt die mittlere Zugriffszeit etwa 130 ... 140 ms. Der Abstand zwischen zwei Spuren ist dabei etwa ein halber Millimeter.

Abb. 2.3.3.1 Floppy-Disk-Stationen enthalten oft eigene Mikroprozessoren zur Steuerung der Laufwerke und für den Austausch von Daten mit dem angeschlossenen Computersystem – wie hier die Station CBM 3040

Derzeit sind zwei Diskettengrößen üblich, nämlich die "normale" Floppy und die Mini-Floppy; die Speicherkapazität und die maximale Übertragungsgeschwindigkeit beider Versionen unterscheidet sich etwa um den Faktor 3. Die Normalfloppy besitzt einen Durchmesser von 203 mm, ihre kleine Schwester 133 mm.

Bei den preiswerten Kompaktcomputern sind meist Minifloppies als Peripherie erhältlich. Sie arbeiten mit 35 Spuren und besitzen entweder 78 KByte (Single Density, einfache Aufzeichnungsdichte) oder 155 KByte Kapazität (Double Density) je Seite.

Jede Spur ist in der Art von Kuchenstücken in Sektoren unterteilt. Eine Spur dient dabei als Index. "Softsektorierte" Disketten besitzen einige Zentimeter vom Zentralloch entfernt ein Sektorloch, das im Laufwerk optisch abgetastet wird und dem Sektor Null entspricht. Bei der Initialisierung wird die Diskette vom Computer oder von der Laufwerkselektronik in 18 Sektoren formatiert. Der Beginn eines jeden Sektors ist durch eine Aufzeichnungslücke gekennzeichnet, wodurch etwas Platz verschwendet wird.

Etwas seltener geschieht die Sektorierung dadurch, daß die Minidiskette 16 Sektorlöcher enthält, so daß keine Aufzeichnungslücken notwendig sind und der Speicherplatz voll ausgenutzt werden kann. Hier spricht man von "hardsektorierten" Floppies.

Beim Umgang mit Floppies sollte man einige Grundregeln beachten. Zunächst einmal gelten für die Floppy-Oberfläche wegen der hohen Aufzeichnungsdichte noch strengere Sauberkeitsforderungen als bei Tonbandgeräten. Um einen Datenverlust zu vermeiden, sollte man die Disketten nicht neben Netztrafos oder anderen Magnetfeld-Quellen liegen lassen. Wegen der undefinierten Betriebszustände beim Abschalten des Laufwerks oder Computers ist es auch ratsam, Disketten vor dem Ausschalten aus dem Laufwerk herauszunehmen.

Für den Umgang mit dem Floppy-Disk-Laufwerk benötigt ein Computer natürlich besondere Befehle (auch in Basic), es sei denn, das Laufwerk besitzt dank eines eigenen Mikroprozes-

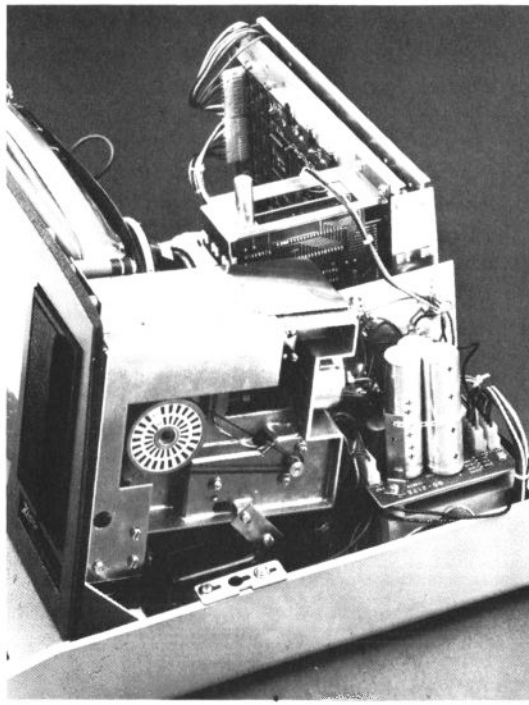


Abb. 2.3.3.2 Floppy-Laufwerk des WH-89. Deutlich ist die Stroboskop-Scheibe erkennbar, die zum Abgleich der Geschwindigkeit dient

sors eine gewisse Eigenintelligenz, so daß es über eine gewöhnliche Schnittstelle (IEC-Bus o.a.) mit dem Rechner verbunden werden kann. Ist das nicht der Fall, so muß ein besonderes "DOS" (Disk Operating System) vorhanden sein, ein Programm, das eine Ergänzung des Basic-Befehlssatzes darstellt und meist auf einer Floppy mit dem Laufwerk mitgeliefert wird.

Aber auch bei "intelligenten" Laufwerken wird meist eine Diskette mitgeliefert, die Programme zum Abrufen eines Inhaltsverzeichnisses, für das Initialisieren anderer Disketten und zum Test des Laufwerks enthält *(Abb. 2.3.3.2)*.

2.3.4 Drucker

Zur Dokumentation von Programmen, aber auch für das Schreiben von Adressenlisten o.a. ist ein Drucker sehr hilfreich. Kaum ein Computer besitzt einen Drucker bereits eingebaut – der AIM-65 alias PC-100 ist hier eine Ausnahme: Er besitzt einen kleinen Thermodrucker mit einer Schriftbreite von 20 Zeichen pro Zeile, was gerade ausreicht, um Programme ausdrucken zu können, wenn auch nicht gerade übersichtlich.

Thermodrucker sind meist die preiswerteste Lösung des Reproduktions-Problems. Dabei wird ein chemisch beschichtetes Papier mit matrixförmig angeordneten Heizelementen lokal erwärmt, wobei es an den erhitzten Punkten dunkel wird. Diese Methode wird auch bei größeren Druckern noch angewendet, z.B. beim Terminal Silent-700 von Texas Instruments, das 80 Zeichen pro Zeile drucken kann.

Ebenfalls recht preisgünstig sind sog. *elektrosensitive Drucker.* Sie arbeiten mit einem aluminiumbeschichteten schwarzen Papier, aus dem durch matrixförmig im Druckkopf angeordnete Kontakte Punkte herausgebrannt werden, die dann schwarz auf hellem Untergrund erscheinen. Sowohl die Thermodrucker als auch diese Alupapierdrucker haben jedoch den Nachteil, daß das dafür notwendige Papier verhältnismäßig teuer ist.

Für ernsthafte Anwendungen sind daher *Normalpapierdrukker* am günstigsten. Es bietet sich an, gebraucht sehr preiswert erhältliche Baudot-Fernschreiber zu verwenden und mit Hilfe geeigneter Codeumwandlungs-Programme als Drucker zu verwenden. Dabei ergeben sich jedoch zwei schwerwiegende Nachteile. Zum einen können solche Fernschreiber entweder nur große oder auch nur kleine Buchstaben produzieren; zum anderen enthalten sie nicht den gesamten Zeichensatz, der für den Ausdruck von Basic-Programmen benötigt wird. Für die Reproduktion von Maschinensprache-Programmen stellt dies jedoch kein Hindernis dar.

Naheliegend ist auch die Verwendung einer elektrischen *Schreibmaschine* als Drucker. Während die meisten elektrischen Typenhebelmaschinen dafür gänzlich ungeeignet sind, weil sie nicht elektrisch angesteuert werden, sondern nur die Hebelkraft

von einem Elektromotor verstärkt wird, ist für bestimmte Kugelkopfmaschinen (z.B. IBM-Selectric) ein Interface lieferbar, das eine Ansteuerung vom Computer über eine Parallelschnittstelle im EBCDIC-Code ermöglicht. Die Umcodierung kann – ebenso wie bei den gerade erwähnten Baudot-Fernschreibern – vom Computer selbst durchgeführt werden. Möglich ist allerdings auch die Verwendung einer gewöhnlichen Schreibmaschine, wenn man jede ihrer Tasten mittels eines Elektromagnets betätigen kann, was zwar einigen mechanischen Aufwand bedeutet, aber zu einem hervorragenden und kaum mit einem Matrix-Drucker zu vergleichenden Schriftbild führt. Diese unkonventionelle Lösung wurde bereits ausprobiert und in der Zeitschrit FUNKSCHAU veröffentlicht (s. Heft 4/1980).

Bei den meisten heute im Zusammenhang mit Mikrocomputern verwendeten Druckern handelt es sich allerdings um Matrixdrucker, die mit Normalpapier arbeiten. Die Zeichen werden wie bei Thermo- und elektrosensitiven Druckern aus einer *Punktmatrix* zusammengesetzt. Die Punkte werden erzeugt, indem man dünne Stifte elektromagnetisch gegen ein Farbband schlägt. Auf diese Weise läßt sich eine sehr hohe Geschwindigkeit erzielen; eine 80-Zeichen-Zeile kann in etwa ein bis zwei Sekunden zu Papier gebracht werden. Üblicherweise setzt man die Zeichen aus einer 5x7-Punktmatrix zusammen; teurere Geräte arbeiten mit einer 7x9-Matrix (dies gilt übrigens auch für Datensichtgeräte).

Für eine 5x7-Matrix werden nun nicht 35 Druckkopfstifte benötigt; vielmehr enthält der Kopf sieben genau übereinanderstehende Stifte. Die Druckerelektronik sorgt selbst für die richtige Ansteuerung der Stiftmagnete, während sich der Kopf horizontal über das Papier hinwegbewegt. Der Ausdruck kann Zeichen für Zeichen, aber auch – wenn der Drucker über einen Pufferspeicher verfügt – zeilenweise geschehen.

Der Anschluß von Druckern geschieht je nach Typ über eine parallele oder serielle Schnittstelle *(Abb. 2.3.4)* oder auch mittels des IEC-Bus. Normalpapierdrucker kosten etwa zwischen 1300 DM (Heathkit) und 4500 DM (Centronics). Bei der Auswahl sollte man beachten, daß die Zeichen und ihre binären

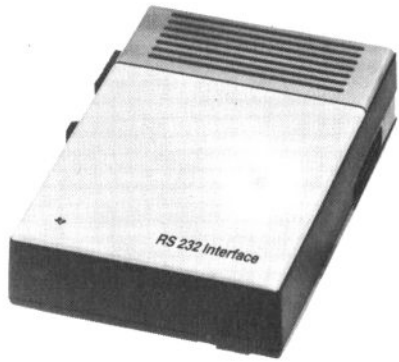


Abb. 2.3.4 Nicht alle Computer gestatten den direkten Anschluß von Druckern oder anderen Peripheriegeräten. Dann ist die Zwischenschaltung eines RS-232-Interface (hier von Texas Instruments für den TI-99/4) erforderlich

Codes möglichst mit denen des Computers übereinstimmen müssen, da nicht alle Computer – wie schon erwähnt – konsequent die ASCII-Norm einhalten.

2.3.5 Was es noch gibt . . .

Natürlich ist das noch nicht alles, was man an einen Computer anschließen kann. Wenn der Rechner wie z.B. der PET-2001 und seine Nachfolger über einen *IEC-Bus* verfügt, lassen sich problemlos alle Geräte an ihn anschließen, die ebenfalls über eine solche Schnittstelle verfügen, z.B. Frequenzzähler, Digitalvoltmeter, Kurvenschreiber (Plotter) und andere Peripherie. Und was nicht mit dem IEC-Bus verbunden werden kann, läßt sich mit einfachen Interface-Schaltungen und Treiber-Sotfware zumindest an die I/O-Ports von PET-2001 und AIM-65/PC-100 anschließen. Beim TRS-80 und auch beim ABC-80 ist das leider

nicht so einfach, weil diese Computer nicht über freie I/O-Ports verfügen. In diesem Fall sind zusätzliche Schaltungen als *Interface* zwischen dem internen Bussystem und der Peripherie erforderlich; beim TRS-80 heißt dieses Zusatzkästchen "Extension Interface". Es ist auch für den Anschluß des Druckers und der Floppy-Disk erforderlich.

Manche Firmen vertreiben sogenannte *"Joysticks"* für Heimcomputer, die etwa die Form eines Steuerknüppels haben und z.B. das Bewegen von Spielfiguren auf dem Bildschirm gestatten. Etwas nützlicher ist schon ein *"Light Pen"*, ein lichtempfindlicher Stift, mit dem man auf dem Schirm zeichnen und korrigieren kann. Auch Geräte für die Sprach-Ein- und -Ausgabe sind bereits zu haben.

Seit 1980 ist auch ein neuartiges Speichermedium im Gespräch: der *optische Plattenspeicher.* Die von Philips 1979 entwickelte, zunächst für die Speicherung von Videosignalen ("Bildplatte") benutzte Platte existiert in zwei Größen, nämlich mit 4 Zoll oder 12 Zoll Durchmesser. Die 4-Zoll-Version kann pro Seite 100 MByte speichern (!). Die 12-Zoll-Version ist für größere EDV-Anlagen vorgesehen und erlaubt die Speicherung von etwa 5×10^8 Byte pro Seite, d.h. insgesamt 10^9 Byte. Die Organisation der 12-Zoll-Platte ähnelt der einer Floppy-Disk: Es werden 128 Sektoren verwendet, die pro Umdrehung d.h. pro Rillen-Spiralwindung, 1 KBit aufnehmen. Insgesamt hat die Platte 45 000 Rillenwindungen, deren Abstand 1,7 μm beträgt. Die mittlere Zugriffszeit, die durch das Hinlaufen des Lesekopfes entsteht, beträgt etwa eine Viertel Sekunde. Dabei wird lediglich eine von rund fünf Millionen Sektoradressen angewählt und der optische Lesekopf zur entsprechenden Rille geschoben. Das Lesen und Schreiben kann mit bis zu 2 MByte/s geschehen; als Schreibkopf dient dabei ein kleiner Diodenlaser, der Löcher in die Platte brennt (Lochdurchmesser ca. 1 μm). Da beim Schreiben sofort ein Lesevorgang zur Kontrolle stattfindet, können Fehler bereits beim Abspeichern der Information erkannt und durch nochmaliges Schreiben (an derselben oder einer anderen Stelle der Platte) ausgebessert werden.

Der Hauptnachteil der optischen Speicherplatte gegenüber einer Floppy ist, daß der Schreibvorgang nur einmal stattfinden kann; gespeicherte Informationen können bestenfalls gelöscht, nicht aber durch neue Daten überschrieben werden. Der Hersteller liefert die Platte bereits "sektoriert", d.h. mit Rillen und Sektoradressen, jedoch sonst unbeschrieben. Hauptanwendungen sind der Vertrieb von Standardprogrammen und die Archivierung von Text und Daten.

Zunehmend gewinnen die sog. *Magnetblasen-Speicher* (Bubble Memories) an Bedeutung. Für manche Basic-Computer, wie etwa für AIM-65 und PC-100, sind bereits anschlußfertige Platinen mit 256 Kbit bis 1 Mbit Kapazität zu haben. Der Preis liegt nur noch unwesentlich über dem guter Floppy-Laufwerke, und die Zugriffszeit ist gegenüber diesen etwa um den Faktor 10 ... 100 geringer. Einziger Nachteil der "Bubbles" ist, daß der Datenträger nicht so leicht austauschbar ist wie eine Floppy. Magnetblasenspeicher behalten ihren Inhalt auch beim Abschalten der Versorgungsspannung; der Zugriff erfolgt ähnlich wie bei Floppy-Laufwerken mit besonderen Steuerprogrammen (Betriebssystem), was den Umgang mit diesen Massenspeichern recht einfach macht.

2.4 Wieviel Speicherplatz brauchen wir?

Mikrocomputer werden mit sehr unterschiedlicher RAM-Kapazität angeboten. Darunter versteht man die dem Anwender für Programme und Daten zur Verfügung stehende Speicherkapazität im Computer. So ist etwa der PET-2001 in seiner einfachsten und preiswertesten Version mit nur 4 KByte RAM ausgestattet und läßt sich bis 32 KByte RAM erweitern. Auf der Platine der Mikrocomputer AIM-65 und PC-100 befinden sich Steckfassungen für maximal 4 KByte RAM, eine Erweiterung ist extern auf maximal 40 KByte möglich. Apple-II und ITT-2020 lassen sich sogar auf bis zu 48 KByte RAM erweitern.

Grundsätzlich können alle 8-bit-Mikrocomputer 2^{16} Byte = 64 KByte mit ihren 16 Adressenleitungen ansprechen. Daß der

adressierbare RAM-Bereich kleiner als 64 KByte ist, ergibt sich daraus, daß alle Computer ja auch Festwertspeicher (ROMs) mit Steuerprogrammen und dem Basic-Interpreter enthalten, die über die gleichen 16 Adressenleitungen angesprochen werden. Je größer die ROM-Kapazität des Rechners ist, desto weniger Adressenraum steht also für den RAM-Arbeitsspeicher zur Verfügung.

Wieviel RAM-Kapazität ist eigentlich sinnvoll? Diese Frage läßt sich sicher nicht allgemein beantworten. Für das Schreiben einfacher Basic-Programme sind bereits 4 KByte absolut ausreichend, während der Speicherbedarf bei Textverarbeitungsaufgaben schnell zunimmt: Jedes Zeichen, jeder Buchstabe belegt 1 Byte. Die Speicherung eines eine Schreibmaschinenseite umfassenden Textes erfordert bereits knapp 2 KByte. Ein 32-KByte-Speicher wird nötig, wenn man z.B. ein Warenverzeichnis (Lagerliste o.ä.) mit 1000 Positionen und etwa 30 Zeichen pro Position im Computer unterbringen möchte.

Wenn man vom reinen Programm-Speicherbedarf ausgeht, so paßt ein Basic-Programm, das von einem üblichen Drucker auf einer Schreibmaschinenseite untergebracht werden kann, in etwa 1 KByte, da die Befehle meist in komprimierter Form abgespeichert werden. Bei vielen nicht-mathematischen Aufgaben läßt sich eine Speichereinsparung von 50 % und mehr bei gleichzeitiger Geschwindigkeitserhöhung um den Faktor 50 ... 500 durch den Übergang auf die Maschinensprache-Programmierung erzielen. Letztere hat allerdings den Nachteil, daß die Programme auf anderen Computertypen nicht mehr laufen und wesentlich schwieriger zu verstehen sind.

3 Personal Computer – EDV für jedermann

Der aus dem Land der ehemals unbegrenzten Möglichkeiten stammende Begriff Personal Computer bedeutet strenggenommen "persönlicher Computer"; die Übersetzungen "Heimcomputer" oder "Hobbycomputer" treffen nicht ganz den Kern der Sache, weil sich solche Geräte ja auch kommerziell, z.B. zur automatischen Meßwertverarbeitung oder für einfache Buchhaltungsaufgaben, einsetzen lassen. Unsere nachfolgende Übersicht kann unmöglich vollständig sein; vielmehr beschränkt sie sich auf typische Vertreter dieser Kategorie und zeigt Unterschiede im Basic-Befehlssatz sowie in der Hardware auf. Die technischen Daten, insbesondere ROM- und RAM-Adressen, beziehen sich auf den Stand von 1980; Änderungen sind z.B. wegen Software-Verbesserungen seitens der Hersteller in manchen Fällen nicht auszuschließen.

3.1 Ein paar Worte zur Marktsituation

1976 kündigte die damals noch nicht allzu bekannte Firma Commodore, die ihren Hauptsitz aus bestimmten Gründen auf den Bahamas hatte, ein Produkt an, das von den etablierten Computerherstellern zunächst als Spielzeug belächelt wurde: den "Personal Electronic Transactor PET-2001". Ohne Mikrocomputer-Herstellern wie Altair und Imsai Pioniergeist absprechen zu wollen, kann man Commodores "PET" als ersten wirklichen Personal Computer bezeichnen. Mittlerweile gab es allerdings rechtliche Probleme: Eine (nicht-elektronische) Firma erwirkte, daß Commodore den Namen PET nicht mehr verwenden darf und auf Personal Electronic Transactor zurückgreifen muß. Neuere Commodore-Computer heißen daher auch CBM (Commodore Business Machines).

Die US-Fachgeschäfte-Kette "Radio Shack" (Tandy) folgte wenig später mit einem gegenüber dem etwa 2900 DM teuren

PET um ein Drittel billigeren Computer, dem TRS-80. Aber während der PET-2001 sich dank seines IEC-Bus-Anschlusses und anderer Ein/Ausgabemöglichkeiten auch für die automatische Meßwertverarbeitung einsetzen läßt, ist dies beim TRS-80 nicht mehr der Fall; Tandy zielte vor allem auf den explosiv wachsenden Markt der Computer-Hobbyisten. Eine Verwendung des TRS-80 wurde erst Jahre später auch im kaufmännischen Bereich möglich, da Peripheriegeräte wie Floppy-Disk und Drucker nicht sofort lieferbar waren.

Ab 1978 folgten weitere Hersteller Schlag auf Schlag mit neuen Computern; Luxor brachte den ersten Heimcomputer mit europäischer Tastatur heraus (ä, ö, ü . . .), den ABC-80, Heathkit stellte den Selbstbau-Kompaktcomputer WH-89 vor. Siemens übernahm die Computerplatine AIM-65 von Rockwell, packte sie in ein schmuckes Gehäuse, das kleiner als manche Reiseschreibmaschine ist, und vertreibt seit 1979 den wohl winzigsten Basic-Computer unter dem Namen PC-100. Er besitzt gegenüber anderen Geräten noch den großen Vorteil, über äußerst komfortable Möglichkeiten zum Programmieren in Maschinensprache zu verfügen. Es folgten Triumph-Adler, Hewlett-Packard, Texas Instruments und viele andere.

Eine Sonderstellung nimmt der Apple-II der amerikanischen Apple Computer Co. ein. Ihn gab es zwar bereits vor dem PET; er konnte aber erstens wegen seines relativ hohen Preises und zweitens wegen des Nichtvorhandenseins von Video-Interface-Schaltungen für die europäische Fernsehnorm in Deutschland zunächst keine Verbreitung finden. Heute stehen Europa-Versionen des Apple-II sowohl von Apple selbst als auch von ITT unter dem Namen ITT-2020 zur Verfügung.

Es ist auffallend, daß die meisten Computerhersteller es nicht gerne hören, wenn ihre Geräte als Hobbycomputer tituliert werden. Sie erwähnen dann zumeist Statistiken, in denen behauptet wird, daß unter den Käufern höchstens 15 Prozent Hobbyisten seien, vermutlich aber noch weniger. Solche Statistiken sind zwar für das Image der Hersteller bei kaufmännischen Anwendern fördernd, aber irreführend. Die Erfahrung zeigt, daß praktisch alle Anwender einen Personal Computer zunächst aus persönlicher Neigung erwerben und alle denkba-

ren und undenkbaren Programme darauf ausprobieren. Erst nach einiger Zeit erkennen sie die berufliche Verwertbarkeit ihres Steckenpferd-Objekts. Wenn man davon ausgeht, daß eine kommerzielle Verwendung von Personal Computern ohne Peripherie (Drucker etc.) nicht sinnvoll möglich ist, so läßt sich obige These leicht dadurch beweisen, daß solche Peripheriegeräte erst nach langer Zeit hinzugekauft werden.

Tatsache ist auch, daß Hobbyisten die Hersteller oft in arge Verlegenheit bringen, weil sie – um den Computer optimal ausnutzen zu können – nach Dingen fragen, deren Dokumentation im Handbuch vom Hersteller leider versäumt wurde, wie etwa die interne Speicherbelegung oder die Adressen bestimmter Programme im Basic-Interpreter-ROM.

Immerhin ist der Heimcomputer-Markt heute so verwirrend, daß es kaum noch möglich ist, einem Neuling ein bestimmtes Produkt zu empfehlen, ohne wochenlang alle Computer auf ihre Eignung für die ganz individuellen Bedürfnisse des Anwenders untersucht zu haben. Nach wie vor gibt es kein "eierlegendes Wollmilchschwein": einen Computer, der die Basic-Befehle eines ABC-80 besitzt, wie der PET-2001 externe Geräte über einen IEC-Bus ansteuern kann, farbige Grafiken wie der TI-99/4 oder der ITT-2020/Apple-II darstellen kann, soviel kostet wie der TRS-80 und wie der AIM-65 über ein komfortables Monitorprogramm verfügt.

Apropos Kosten: Bei der Beurteilung eines Computers sollte man keinesfalls mit dem Preis des Grundsystems rechnen, sondern die notwendigen Eingabe/Ausgabe-Bausteine in Betracht ziehen. So benötigt man beim PET-2001 z.B. kein weiteres Interface für die Ansteuerung externer Geräte, während der zunächst preiswertere TRS-80 wegen des dann plötzlich notwendig werdenden Expansion-Interface höhere Kosten verursacht.

Für alle, die sich nicht sämtliche Programme selbst schreiben wollen, z.B. einen Assembler, einen Disassembler oder einen Texteditor (also Programme, die einen sehr hohen Zeitaufwand und ein großes Maß an Erfahrung verlangen, aber oft benötigt werden), ist natürlich auch die Erhältlichkeit fertiger Software ein ganz wesentliches Entscheidungskriterium beim Kauf eines Computers. Für verbreitete Geräte sind fertige Programme auf

Kassetten und Floppy-Disks erhältlich; außerdem findet man Software für die unterschiedlichsten Anwendungen in den einschlägigen Fachzeitschriften.

Trotzdem kann es manchmal erforderlich sein, sich individuelle Programme von einem Software-Büro erstellen zu lassen (die Adressen solcher Software-Büros sind vom Hersteller des jeweiligen Computers zu erfahren). Dies gilt besonders beim kommerziellen Computereinsatz, wenn es den Mitarbeitern an Erfahrung oder Zeit für das eigene Programmieren fehlt. Bei der Auftragsvergabe an fremde Entwickler sollte man unbedingt beachten:

- Definieren Sie möglichst genau, was das Programm tun soll. Spätere Änderungen verursachen gewöhnlich enorme Kosten und auch einigen Ärger: Das Software-Büro schreibt nicht etwa das Programm, wie Sie es ursprünglich wollten, sondern so, wie der Entwickler sich Ihre Wünsche vorstellt.

- Sprechen Sie vor Beginn der Entwicklung über die zu erwartenden Kosten. Für einen Außenstehenden ist es praktisch unmöglich, den tatsächlichen Zeitbedarf für eine Softwareentwicklung abzuschätzen.

- Verlangen Sie unbedingt eine exakte Dokumentation des Programmes.

Dabei sind auch die Erfahrungen interessant, die professionelle Software-Entwickler gemacht haben. Rund 64 % aller Software-Fehler sind nach einer Diebold-Untersuchung auf Fehler zurückzuführen, die bereits in der Problemanalyse und Planung gemacht wurden. Lediglich 34 % sind auf echte Programmierfehler zurückzuführen.

In der professionellen Programmierung spricht man oft von "Generatoren". Damit sind Computer-Betriebsprogramme gemeint, die eine sehr grob strukturierte Programmierung in Problemgruppen zulassen und das gewünschte Gesamtprogramm selbständig aus Routinen für Standardprobleme zusammensetzen. Die so entstandene Software ist relativ preiswert, da der Zeitaufwand für das Zusammensetzen der Routinen recht gering ist. Allerdings sind diese Programme wegen der großen

geforderten Flexibilität weder speicher- noch geschwindigkeitsoptimiert. Diese Nachteile werden bei Großrechnern heute hingenommen, um die Softwarekosten und die versteckten Programmierfehler durch die Verwendung von Standardroutinen besser in den Griff zu bekommen. Für Heimcomputer ist das Generator-Verfahren dagegen absolut unbrauchbar, weil zu langsame Programme mit zu hohem Speicherbedarf entstehen.

Umgekehrt sind Programmier-Tricks, die Heimcomputer-Besitzer gerne anwenden, auf Großrechnern fehl am Platze. Sie würden umfangreiche Programme noch unüberschaubarer und unverständlicher machen, als sie ohnehin schon sind. Hier lautet die Devise: Lieber ein paar Byte mehr investieren, statt ein Programm zu erstellen, in dem sich nach einem Monat nicht einmal mehr der Programmierer selbst auskennt.

3.2 AIM-65 und PC-100

3.2.1 Hardware

AIM-65 (Advanced Interactive Microcomputer) und PC-100 *(Abb. 3.2)* (Personal Computer) sind zwei intern identische Mikrocomputer, die in unterschiedlichen Ausstattungen zu haben sind. Der PC-100-Kit gleicht dem AIM-65; Siemens kauft ersteren vom AIM-Hersteller Rockwell als Platine ohne Gehäuse und Netzteil, und in dieser Grundausstattung kostet ein solches Gerät knapp 1000 DM. Wenn man noch etwas dazublättert, bekommt man auch die für Basic notwendigen Interpreter-ROMs, für die auf der Platine bereits freie Fassungen vorhanden sind, ebenso wie für eine Erweiterung des RAM auf 4 KByte, wenn man sich ursprünglich für die 1-KByte-Version entschieden hatte. Die Platine ist nach dem Anschluß von +5 V und +24 V betriebsbereit; verzichtet man auf den Drukker, so sind sogar nur +5 V erforderlich. Der PC-100 enthält bereits ein Netzteil und kostet knapp 2000 DM. Zusätzlich zu der per Software abgefragten Tastatur, dem 20stelligen alphanumerischen 16-Segment-LED-Display und dem 20 Zeichen breiten Thermodrucker enthält die Platine auch eine

Abb. 3.2 Der von Siemens gefertigte PC-100 stellt praktisch einen mit Basic-ROMs, Netzteil und Gehäuse ausgerüsteten AIM-65 dar

Schnittstelle für den Anschluß eines externen Terminals oder eines breiteren Druckers. Dabei ist eine automatische Geschwindigkeitsanpassung an das externe Gerät möglich. Zwar verfügt der AIM-65 nicht über einen IEC-Bus-Anschluß wie der PET-2001, aber über 16 Eingabe/Ausgabe-Leitungen und eignet sich daher sehr gut für Steuerungsaufgaben.

Im Gegensatz zu allen anderen in diesem Buch besprochenen Computern verfügt der AIM-65 (damit natürlich auch der PC-100) über ein 8 KByte umfassendes Monitorprogramm, das dem Benutzer ein sehr komfortables Programmieren in der Maschinensprache des verwendeten 6502-Mikroprozessors erlaubt. Dazu gehören ein Assembler (mit absoluter Adressierung), ein Disassembler und zahlreiche Hilfsprogramme z.B. für

das Testen von Maschinenprogrammen im Einzelschritt-Betrieb. Ein Assembler, der zusätzlich noch die Verwendung symbolischer Adressen (Labels) beim Programmieren gestattet, ist als Zubehör in einem 4-KByte-ROM lieferbar.

Doch gehört dies eigentlich schon zur "Firmware", zu den fest im Computer gespeicherten Programmen, die seine Softwareeigenschaften bestimmen. Noch einmal zurück zur Hardware: AIM-65 und PC-100 verfügen über drei unterschiedliche Methoden, Programme oder Daten auf eine Tonbandkassette aufzuzeichnen.

Das "normale" Kassettenformat entspricht der Kansas-City-Norm – aber nur, was die Tonfrequenzen 2400 Hz und 1200 Hz angeht: Jedes Bit beginnt mit einer halben Periode eines 2400-Hz-Tones. Die nachfolgenden drei Halbperioden bestimmen, ob das Bit log. 1 oder log. 0 ist: Drei 2400-Hz-Halbzyklen entsprechen log. 1, drei 1200-Hz-Halbzyklen bedeuten log. 0. Dann beginnt das nächste Bit mit einer halben 2400-Hz-Periode. Beim Abspeichern werden die einzelnen Bytes nicht in zwei ASCII-Hex-Zeichen aufgespalten, sondern direkt auf Band übernommen. Für das Abspeichern von Texten ergibt sich dabei automatisch das ASCII-Format, weil ein Byte genau ein ASCII-Zeichen enthält.

Die Daten werden nicht einfach Bit für Bit, Byte für Byte übertragen, sondern in Blöcken von je 79 Byte. Zusätzlich enthält jeder Block mindestens 32 Synchronisationszeichen am Anfang, gefolgt von dem Zeichen "#" und einem Byte für die hexadezimale zweistellige Blocknummer, sowie – nach den Datenbytes – eine 2-Byte-Blockprüfsumme. Im ersten Block wird zunächst ein Dateiname (File Name) mit aufgezeichnet, der maximal fünf Zeichen umfassen darf und der später erlaubt, beim Wiedereinlesen durch Eingeben des gleichen Dateinamens ein ganz bestimmtes Programm zu laden. Stößt der Computer beim Lesevorgang auf ein Programm mit anderem Dateinamen, so wird nur dessen Name auf dem Display ausgegeben, es wird aber übergangen.

Zwei weitere Aufzeichnungsformate des AIM-65-Kassetteninterface, die allerdings seltener angewandt werden, sind mit

dem Einplatinen-Mikrocomputer KIM-1 kompatibel und unterscheiden sich untereinander nur in der Geschwindigkeit. Jedes Byte wird dabei in zwei hexadezimale Ziffern aufgespalten und in Form zweier ASCII-Zeichen übertragen, wodurch natürlich ein Verlust an Geschwindigkeit unvermeidbar ist. Als Dateiname werden hier – analog zum KIM-1 – gewöhnlich nur zwei hexadezimale Ziffern verwendet.

3.2.2 Speicheraufteilung

Die Mikrocomputer AIM-65 und PC-100 arbeiten mit der CPU 6502, deren 16 Adressenleitungen 2^{16} Byte, also rund 65 KByte ansprechen können. Dieser Gesamtspeicherbereich ist wie folgt belegt (die Adressen werden hier vierstellig hexadezimal genannt):

0000...00DE	Freies RAM (Zero Page)
00DF...016F	Vom Monitorprogramm benutztes RAM
0170...01FF	Stack-RAM
0200...0FFF	RAM (auf der Platine)
1000...9FFF	Frei für externe Erweiterungen
A000...AFFF	I/O-Bausteine und vom Monitorprogramm benutztes RAM
B000...CFFF	Basic-ROMs (8 KByte)
D000...DFFF	Assembler-ROM (4 KByte)
E000...FFFF	ROM mit Monitorprogramm (8 KByte)

Bei der 1-KByte-Ausführung reicht das RAM bis zur Adresse 03FF, bei der 4-KByte-Version bis 0FFF. Der Basic-Interpreter speichert die vom Anwender eingegebenen Basic-Programme im Bereich ab 0200 bis zu einer vorher definierbaren oberen Grenze, die man auch unterhalb 1000 legen kann, falls man am oberen RAM-Ende z.B. ein Maschinenprogramm stehen hat. Die Zero-Page (0000...00FF) wird mit vorübergehend benötigten Daten belegt; die wichtigsten Adressen in diesem Bereich sind nachfolgend im Vergleich mit dem Mikrocomputer PET-2001 wiedergegeben:

Wichtige Zero-Page-Adressen bei PET-2001 und AIM-65 (Version 1.1)

PET	*AIM*	*Inhalt*
00-02	03-05	Sprung zum USR-Programm
05	11	Cursorposition
0A-5A	16-5C	Input Buffer
5C	61	Derzeitige Input-Länge
7A-7B	73-74	Beginn des Basic-Programms
7C-7D	75-76	Beginn der Variablentabelle
7E-7F	77-78	Beginn der Matrix-Variablen
80-81	79-7A	Beginn des freien RAM
82-83	7B-7C	Unteres Ende der Strings
84-85	7D-7E	Oberes Ende der Strings
86-87	7F-80	Oberes Ende des Basic-RAM
88-89	81-82	Derzeitige Zeilennummer
8A-88	83-84	Von END und BREAK abgelegte Zeilennummer
8C-8D	85-86	Von END und BREAK abgelegte Programmadresse
94-95	8D-8E	Aktuelles Variablen-Symbol
96-97	8F-90	Startadresse der aktuellen Variable
B0-B5	A9-AE	Fließkomma-Akku (binär)
–	BB-BD	Sprung zum ATN-Maschinenprogramm

3.2.3 Monitorprogramm

Wie schon erwähnt, ist eine der Stärken von AIM-65 bzw. PC-100 die komfortable Programmierbarkeit in der 6502-Maschinensprache, die durch das 8 KByte umfassende Monitorprogramm zustandekommt. Es gestattet folgende, durch Drücken einer einzigen Taste zugängliche Systembefehle:

E Texeditor initialisieren (beliebiger Textspeicherbereich festlegbar)

T Falls nicht im Texteditor gearbeitet wird, zu diesem springen und die erste Textzeile anzeigen

N Zur Adresse D000 springen (Assembler-ROM-Startadresse)

5 Zur Adresse B000 springen (Basic-Kaltstartadresse, hierbei wird ein im Speicher stehendes Basic-Programm gelöscht)

6 Zur Adresse B003 springen (Basic-Warmstart, kein Programm- oder Datenverlust)

I Maschinenbefehle als mnemonische Abkürzungen eingeben

K Gespeichertes Maschinenprogramm disassembliert ausgeben

★ Zu einer neuen Adresse gehen bzw. Programmzähler ändern

A,X,Y,P,S CPU-Registerinhalt ändern (Akku, X-Register, Y-Register, Statusregister, Stackpointer)

R Inhalt aller CPU-Register anzeigen

M Vier aufeinanderfolgende Bytes an der spezifizierten Adresse anzeigen

SP (Leerraum, Space) Die nächsten vier Bytes anzeigen

/ Die angezeigten Bytes ändern

L Maschinenprogramm z.B. vom Band laden

D Maschinenprogramm z.B. auf Band abspeichern

Breakpoints löschen

4 Breakpoints aktivieren oder ausschalten

B Breakpoint-Adresse eingeben

? Alle Breakpoint-Adressen anzeigen

G Programm starten

Z Trace-Modus ein- und ausschalten (jeder ausgeführte Programmbefehl wird auf Display und evtl. Drucker ausgegeben)

V Register-Trace-Modus ein- und ausschalten (nach jedem ausgeführten Programmbefehl werden die CPU-Registerinhalte ausgegeben)

H Die vier zuletzt vom Programm durchlaufenen Adressen ausgeben

1 Motor von Kassettenrecorder 1 ein- und ausschalten

2 Motor von Kassettenrecorder 2 ein- und ausschalten

3 Tonbandaufzeichnung prüfen ("Verify"), ohne sie zu laden

F1 Indirekter Sprung über 010C (Basic: Break-Taste)

F2 Indirekter Sprung über 010F
F3 Indirekter Sprung über 0112

Die vom Benutzer selbst definierbaren Tasten F1...F3 sind in Basic nicht als Sonderfunktionstasten wirksam, sondern liefern die ASCII-Codes (hex) 5B, 5D und 5E. Es muß vermieden werden, sie außerhalb von Basic zu drücken, solange die zu ihnen gehörenden Vektoren an den Adressen 010C...0114 nicht auf ein gültiges Programm zeigen, da sich sonst der Computer "aufhängen" kann, was u.U. zu einer Zerstörung gespeicherter Programme und Daten führt.

3.2.4 Basic-Interpreter

Wenden wir uns nun dem Basic-Interpreter zu. Er wurde von der amerikanischen Firma Microsoft entwickelt und entspricht in seiner Syntax den meisten anderen Personal-Computer-Interpretern. Die eingegebenen Basic-Befehle werden nicht in voller Länge, sondern in Form eines einzigen Byte codiert abgespeichert. Hier die Befehle und ihre hexadezimalen Code-Äquivalente:

Basic-Befehlscodes beim AIM-65 bzw. PC-100

80 END	90 ON	A0 SPC(	B0 ABS	C0 ASC
81 FOR	91 NULL	A1 THEN	B1 USR	C1 CHR$
82 NEXT	92 WAIT	A2 NOT	B2 FRE	C2 LEFT$
83 DATA	93 LOAD	A3 STEP	B3 POS	C3 RIGHT$
84 INPUT	94 SAVE	A4 +	B4 SQR	C4 MID$
85 DIM	95 DEF	A5 –	B5 RND	C5 GO
86 READ	96 POKE	A6 *	B6 LOG	
87 LET	97 PRINT	A7 /	B7 EXP	
88 GOTO	98 CONT	A8 ↑	B8 COS	
89 RUN	99 LIST	A9 AND	B9 SIN	
8A IF	9A CLEAR	AA OR	BA TAN	
8B RESTORE	9B GET	AB >	BB ATN	
8C GOSUB	9C NEW	AC =	BC PEEK	
8D RETURN	9D TAB(	AD <	BD LEN	
8E REM	9E TO	AE SGN	BE STR$	
8F STOP	9F FN	AF INT	BF VAL	

Der allein nicht zulässige Befehl GO (hex C5) darf nur in Verbindung mit TO vorkommen. Er wird vom AIM-Basic nur deshalb getrennt decodiert, weil es zulässig ist, statt GOTO auch GO TO zu schreiben. Bei allen anderen Basic-Befehlen ist es nicht zulässig, innerhalb des Befehlswortes Leerräume einzufügen; außerhalb der Befehlsworte werden Leerräume dagegen ignoriert, wenn auch mitgespeichert.

Das AIM-Basic belegt den RAM-Speicherbereich ab der Adresse 0200. Nehmen wir an, wir hätten das folgende, zugegebenermaßen ziemlich sinnlose Programm eingetippt:

```
10    GOTO 2000
2000  PRINT "READY"
```

Dieses kurze Programm würde wie folgt vom Basic-Interpreter im RAM abgelegt werden:

0214 0A 00	Zeilennummer 10, hexadezimal
0216 88	Befehlscode für GOTO
0217 20	ASCII: Leerraum zwischen GOTO und 2000
0218 32 30 30 30	ASCII: 2000
021C 00 2A 02	Null-Byte und Zeiger zum nächsten Befehlsende
021F D0 07	Zeilennummer 2000, hexadezimal
0221 97	Befehlscode für PRINT
0222 22 52 45 41 0226 44 59 22	} "READY" im ASCII-Format
0229 00 00 00	Ende des Programms

Bei den meisten anderen Personal Computern würde der Speicherinhalt ganz ähnlich aussehen, wenn man einmal davon absieht, daß das Basic-Programm natürlich nicht immer ab 2000 abgelegt wird.

Wenn wir statt GOTO in unserem Demonstrationsprogramm GO TO geschrieben hätten, so würde der Befehlscode 88, der nur ein Byte belegt, durch die drei Bytes C5 20 9E ersetzt; ein einziger Leerraum kostet uns hier also schon zwei Byte zusätzlichen Speicherbedarf!

Wenn das Programm irgendwelche numerischen Variablen, z.B. A, ZI, C1 usw. verwendet, so werden sie in dem dem eigentlichen Programm folgenden Speicherbereich abgelegt. Handelt es sich bei den Variablen nicht um Matrizen wie A(I) usw., so belegen sie je sechs Bytes, nämlich zwei für den Variablennamen und vier für ihren numerischen Wert. Stringvariablen wie A$usw. werden anders abgespeichert, weil der Computer ja nicht voraussehen kann, wie lang die Strings (Zeichenketten, z.B. Texte) sein werden. Schreiben wir einmal folgendes Demonstrationsprogramm in den AIM-65 oder PC-100:

```
10 AB$="TEST"
```

Wenn wir uns dann mit Hilfe des AIM-Monitorprogramms, in das wir aus Basic mit der Escape-Taste gelangen, den resultierenden RAM-Inhalt ansehen, so ergibt sich folgendes:

```
0223  41   ASCII "A", erster Buchstabe des Variablennamens
0224  C2   ASCII "B" + hex 80, zweiter Buchstabe mit MSB=1
           (Stringvariable)
0225  04   Hexadezimale Länge des Strings
0226  1B   Niederwertiges Byte  }  Anfangsadresse des Strings
0227  02   Höherwertiges Byte   }
0228  00   }
0229  00   }  nicht benutzt
```

Das Kennzeichen für Stringvariable ist, daß das höchstwertige Bit im zweiten Buchstaben des Variablennamens stets 1 ist. Der String selbst sieht hier so aus:

```
021B  54 45 53 54
```

Strings, die nicht im Basic-Programm selbst stehen, sondern erst im Lauf der Zeit eingegeben werden, stehen am oberen Ende des reservierten Speicherbereiches (beim 4-K-AIM knapp unter 1000 hex).

Als einzige der in diesem Buch beschriebenen Computer außer dem Apple-II besitzen AIM-65 und PC-100 ein leistungs-

fähiges Monitorprogramm für die Programmierung in Maschinensprache. Aus diesem Grund werden hier noch die wichtigsten Programmteile der Basic-Interpreter-ROMs angegeben. (UP = Unterprogramm)

00BF	In das RAM geladene Routine zum Holen des nächsten Zeichens aus dem Basic-Programm (UP)
B000	Kaltstartadresse des Basic (AIM-Taste 5)
B003	Warmstartadresse des Basic (AIM-Taste 6)
B00A	Adressen der Routinen für die Befehle END bis MID$
B072	Adressen der Routinen für Operatoren (+, –, OR, NOT usw.) und Vergleiche
B090	Liste der Basic-Befehlsworte (ASCII)
B175	Error-Code-Liste
B1AC	UP: Stack nach Rücksprungadressen und FOR-NEXT-Variablen absuchen
B1DA	UP: Platz für einzufügende Befehle schaffen
B21D	UP: Stackzustand überprüfen
B22A	UP: Platz im RAM prüfen, Strings packen
B257	UP: Out-of-Memory-Error
B274	Break-Meldung
B27F	Einsprung nach Programmende (z.B. END) oder mit AIM-Taste 6; Text einlesen
B329	UP: Zeilennummern abspeichern
B356	Delete (ein Zeichen nach links gehen)
B365	Zeichen vom aktiven "Input Device" einlesen, mit Echo auf aktives "Output Device" (UP)
B3AE	UP: Befehlsworte durch ihre Hex-Codes ersetzen
B436	UP: Zeilennummer im Programm suchen
B465	NEW-Befehl ausführen
B4A7	CONT-Befehl sperren
B4BC	List-Befehl ausführen (000E=List/Save–Flag)
B55C	FOR-Befehl ausführen
B640	Taste F1 abfragen; wenn gedrückt, Programmabbruch (Break)
B685	Befehl CONT ausführen (gesperrt bei 0086 = 00)
B69F	SAVE-Befehl ausführen, dabei List-Routine benützen

B6AB Zeichen holen (UP)
B6B9 Zeichen ausgeben (UP)
B6E3 Tastatur für GET abfragen
B6EC RUN-Befehl ausführen
B6F7 GOSUB-Befehl ausführen
B714 GOTO-Befehl ausführen
B741 RETURN-Befehl ausführen
B767 DATA-Anweisung bearbeiten
B797 IF-Befehl ausführen; Ergebnis 0 oder −1 im Fließkomma-Akku ablegen
B7AA REM-Anweisung: Den folgenden Text überlesen
B7BA ON-Befehl ausführen
B7DA Dezimalzahl (0...65535) in Binärzahl umwandeln und in 0014, 0015 speichern
B814 LET-Befehl ignorieren, "=" ausführen
B8A9 PRINT-Befehl ausführen; bei PRINT! Drucker einschalten
B8FA Zeichen-Einlesen beenden, CR/LF ausgeben (UP)
B988 UP für GET, READ und INPUT
BADC Texte "EXTRA IGNORED" und "? REDO FROM START"
BB00 NEXT-Befehl ausführen
BB7F Wert einer Variablen für Vergleichsoperationen ermitteln (UP)
BC9C NOT-Anweisung ausführen, d.h. Fließkomma-Komplement bilden
BD3F OR-Anweisung ausführen
BD42 AND-Anweisung ausführen
BD6F Vergleichsoperationen durchführen
BDD7 DIM-Anweisung ausführen
BE6E UP: Zeichen im Akku auf Buchstabe testen; wenn ja, dann Carry-Flag setzen
BEDC Startadresse des ersten Matrix-Elements ermitteln (UP)
BF10 Matrix suchen und Adresse des gewünschten Elements ermitteln (UP)
C0BD FRE-Anweisung ausführen
C0DE POS-Anweisung ausführen

C0F1 DEF-Anweisung ausführen
C132 FN-Anweisung ausführen
C1A3 STR$-Anweisung ausführen
C264 Strings an das Speicherende packen
C42A,C43E,C46A,C475
CHR$, LEFT$, RIGHT$, MID$ ausführen
C4BA,C4C9,C4EB,C54C
LEN,ASC,VAL,PEEK ausführen
C563,C56C,C592,C5A9 POKE,WAIT,–,+ ausführen
C6F8 Wertetabelle für Logarithmus-Errechnung
C76A Multiplikationsbefehl ausführen
C838 Datentabelle
C83D Akku runden
C851 Divisionsbefehl ausführen
CA0B INT-Anweisung ausführen
CA32 UP: Zeichenfolge als Zahl in den Fließkomma-Akku einlesen
CB1C UP: Zahl im Fließkomma-Akku in einen String umwandeln und im Zahlenbuffer 0200...0210 ablegen
CC4C Datentabelle für Wurzel- und e-Funktionsberechnung
CC75,CC7F SQR und ↑ ausführen
CCC0 Datentabelle
CCF1 EXP berechnen
CD8C Datentabelle
CD96,CDD2,CDD9,CE22 RND,COS,SIN,TAN ausführen
CE4C Datentabelle für trig. Funktionen
CE86 LOAD-Befehl ausführen (UP)
CE9E Datentabelle
CEA3 Initialisierung (nach AIM-Taste 5)
CFAC Texttabellen: "Memory Size", "Width", "Bytes Free", "AIM 65 Basic V1.1", "(C) 1978 Microsoft"
CFFA Korrekturprogramm für INPUT

Es sei noch erwähnt, daß die Aachener Firma GWK eine AIM-65-Basic-Erweiterung entwickelte, die die Tatsache ausnutzt, daß nach "END" ein Sprung über die Adresse 0000 erfolgt, wo normalerweise der Befehl 4C 00 B9 (JMP B900) steht.

Ändert man diesen Sprung, so kann man *eigene Basic-Befehle* hinzufügen, die mit END: beginnen. Die auf Kassette oder EPROM lieferbare Erweiterung gestattet z.B. das Öffnen und Schließen von Kassettenfiles, das Umnumerieren der Basic-Programmzeilen (RENUMBER) und andere nützliche Dinge.

Da uns nun auch bekannt ist, daß der *CONT-Befehl* durch Zelle hex 0086 bzw. dezimal 134 gesperrt werden kann, ist es möglich, den AIM-65 zu überlisten, wenn er einmal CN ERROR ausgibt. Dies ist bei CONT z.B. der Fall, wenn das Basic-Programm mit der Break-Taste (F1) oder einem STOP-Befehl angehalten und dann ein Direktbefehl falsch geschrieben wurde, so daß ein SN ERROR (Syntaxfehler) auftrat. In diesem Falle geben wir einfach POKE 134,2 ein – und schon läßt sich das Programm wieder mit CONT fortsetzen. RUN hätte hier den großen Nachteil, daß alle Variablenwerte gelöscht würden.

Befehlssatz-Besonderheiten

Das AIM-65- bzw. PC-100-Basic weist einige Besonderheiten auf, die beim Umschreiben von Programmen unbedingt berücksichtigt werden müssen.

So ist etwa das für die Berechnung des Arcustangens (ATN) notwendige Maschinenprogramm aus unerfindlichen Gründen nicht im Basic-ROM untergebracht, so daß, wenn man es tatsächlich benötigt, ein entsprechendes, im Handbuch angegebenes kurzes Programm in das RAM geschrieben werden muß. Wenn in einem Basic-Programm die *ATN-Funktion* vorkommt, so erfolgt ein indirekter Sprung über die dezimalen (!) Adressen 188 und 189 zum Maschinenprogramm, das der Benutzer zur Bearbeitung der ATN-Funktion vorgesehen hat. Selbstverständlich muß es sich dabei nicht unbedingt um die Arcustangens-Funktion handeln; vielmehr eignet sich ATN (ebenso wie USR) für den Aufruf beliebiger 6502-Maschinenprogramme. Dabei ist lediglich darauf zu achten, daß dabei die gleiche Befehlsform wie bei der Berechnung des Arcustangens verwendet wird; ein alleinstehendes ATN führt zu einer Fehlermeldung. Vor dem Aufruf von ATN muß man in die dezimalen Adressen 188 und 189 natürlich noch die Anfangsadresse des Maschinenprogram-

mes eingeben, z.B. mit dem POKE-Befehl. Dabei ist das höherwertige Adreßbyte in 189 und das niederwertige in 188 zu schreiben.

Eine weitere Besonderheit des AIM-Basic hängt damit zusammen, daß ein einzeiliges *LED-Display* zur Ausgabe verwendet wird. Die Anweisung PRINT4-5 führt scheinbar zu dem falschen Ergebnis 1, weil das Minuszeichen auf dem Display sofort vom Cursor (einem nach oben weisenden, die aktuelle Schreibposition markierenden Pfeil) überschrieben wird. Beim Drucker tritt das Problem natürlich nicht auf, weil auf ihm kein Cursor geschrieben wird. Um das Display-Problem zu vermeiden, wenn aus irgendwelchen Gründen wenig später der Cursor auftaucht, um eine Eingabe zu ermöglichen, ist es ratsam, bei obigem Beispiel die Befehlsform PRINT" ";4-5 zu verwenden, um das Rechenergebnis um eine Stelle nach rechts zu verschieben.

Der im Interpreter-ROM vorhandene Befehl *NULL* wird beim AIM-65 bzw. PC-100 grundsätzlich nicht ausgewertet und führt zu FN ERROR sowie zum Abbruch des laufenden Basic-Programms.

Ein alleinstehender Befehl PRINT (ohne Argument) dient bei vielen Basic-Computern dazu, in eine neue Zeile zu gehen. Bei AIM-65 und PC-100 ist PRINT allein wirkungslos. Um trotzdem eine neue Zeile zu beginnen, muß ein Leerraum als Argument verwendet werden; der Befehl lautet dann *PRINT" "*.

Bedingt durch die Länge des "Input Buffer" dürfen beim AIM-65 die eingegebenen Programmzeilen maximal 60 Zeichen lang sein. Dabei ist es auch zulässig, mehrere Befehle, durch Doppelpunkte voneinander getrennt, in eine gemeinsame Zeile zu schreiben. Dabei ist – wie bei jedem anderen Computer auch – auf die Problematik von IF...THEN und IF...GOTO bei *Mehrfachzeilen* zu achten (s. Kapitel 5 und 6).

Während das Abspeichern von Programmen auf Kassette mit SAVE und LOAD kein Problem darstellt, besitzt das AIM-Basic keinerlei besondere Befehle für die *File-Verarbeitung*, d.h. das Abspeichern von Variablenwerten und Strings auf Kassette. Unter Verwendung einiger weniger Maschinenprogramme, die

bereits im ROM des Monitorprogramms zur Verfügung stehen, ist dies aber kein Problem. Folgendes Programmsegment speichert die Matrix-Stringvariable A$(I) bis zu jenem I auf Band, bei dem der Variablenwert END ist:

```
700 J=0:POKE188,113:POKE189,232:POKE41993,32:I=ATN(0)
710 PRINT A$(J):IFA$(J)<>"END"THENJ=J+1:GOTO710
720 PRINTCHR$(13):POKE188,10:POKE189,229:I=ATN(0):
    PRINT"SAVED":RETURN
```

Hierbei kann wie beim Abspeichern von Programmen ein bis zu fünf Zeichen langer Dateiname mit aufgezeichnet werden, der auch bei folgendem Ladeprogramm wieder angegeben werden muß:

```
540 POKE188,72:POKE189,232:I=ATN(0)
550 INPUTA$(J):IFA$(J)<>"END"THENJ=J+1:GOTO550
560 POKE188,3:POKE189,255:I=ATN(0):PRINT"LOADED":
    RETURN
```

Und so funktionieren die Routinen: Zunächst wird jeweils der Vektor für die ATN-Funktion auf WHEREI bzw. WHEREO gesetzt, um dann bei der Ausführung der Blindzuweisung I=ATN(0) dem Benutzer die Eingabe des File-Namens zu erlauben. Dabei wird auch INFLG bzw. OUTFLG auf das gewünschte Ausgabegerät gesetzt, hier auf "T". Ab sofort wird bei INPUT- bzw. PRINT-Befehlen der Kassettenrecorder als Ein- und Ausgabegerät betrachtet. Sobald die File geladen oder abgespeichert ist, wird INFLG bzw. OUTFLG zur Bedienung von Tastatur und Display auf hex 0D zurückgesetzt. Bei der SAVE-Routine tut dies der zweite ATN-Aufruf; er sorgt auch dafür, daß der 6522-Timer angehalten wird, da er vorher im Freilauf-Modus arbeitete.

Wenn die Fernsteuerung des Kassettenrecorder-Motors nicht verwendet wird oder wenn man während des Ladens der File nicht alles gleichzeitig ausdrucken möchte, empfiehlt es sich, vor der Laderoutine den Drucker mit POKE 42001,0 aus- und bei Bedarf später mit POKE 42001,128 wieder einzuschalten.

Leider ist dies bei laufendem Basic-Programm nicht anders möglich.

Es ist ein gewisses Handicap des AIM-Basic, daß keinerlei Möglichkeit besteht, einzelne Zeichen in einer bereits gespeicherten Programmzeile zu ändern. Vielmehr muß die gesamte Zeile neu eingegeben werden. Auch diese Eigenschaft hängt wieder mit dem Display-Konzept zusammen, läßt sich aber durch vom Anwender selbst geschriebene Maschinenprogramme für die Ein- und Ausgabe ändern. Zu Problemen führt oft auch, daß das sonst für "Backspace"-Korrekturen oft benutzte CTRL H(hex 08) beim AIM-65 durch Delete/Rubout (hex 7F) ersetzt ist, was oft zu Inkompatibilitäten zwischen Computer und Terminals führt, wenn nicht mit der eingebauten Tastatur gearbeitet wird. Sowohl von der Tastatur her als auch wegen des 16-Segment-Display und der verwendeten Drucker-Software ist auch die Verarbeitung von Kleinbuchstaben (hex 60... 7E) problematisch und praktisch nur mit einem externen Terminal möglich. Für viele Anwendungsfälle spielt dies jedoch eine untergeordnete Rolle.

3.2.5 Text-Editor

Ein bereits in der Grundausstattung von PC-100 und AIM-65 in den Monitor-ROMs vorhandener Programmteil ist der AIM-Texteditor. Er wird unter anderem auch zum Schreiben des Quellenprogrammes für den Assembler benötigt, der den Quellencode dann in den Objektcode, sprich in die 6502-Maschinensprache, übersetzt. Der Editor wird vom Monitorprogramm aus mit der Taste E initialisiert und erlaubt folgende Befehle:

Q	Rückkehr zum Monitorprogramm
R	Zeilen von der Tastatur, von der Kassette oder anderswo her einlesen
I	Eine Zeile vor die angezeigte Zeile einsetzen
K	Angezeigte Zeile löschen
U	Nächsthöhere (=vorhergehende) Zeile anzeigen
D	Eine Zeile nach "unten" gehen

T Erste (oberste) Textzeile anzeigen
B Letzte Textzeile anzeigen
L Beliebige Anzahl von Zeilen auflisten oder extern abspeichern
SP (Leertaste) Aktuelle Zeile anzeigen bzw. ausdrucken
F In allen folgenden Zeilen nach einem bestimmten Stichwort suchen
C Zeilentext ändern. Dabei müssen der bisherige Zeilenteil und der neue, durch den er ersetzt werden soll, eingegeben werden.

Der letztgenannte Editorbefehl stellt nur einen teilweisen Ersatz für die komfortablen Editier- und Korrekturmöglichkeiten von Computern dar, die mit einem Bildschirm mit voller Cursorsteuerung ausgestattet sind. Es ist übrigens auch möglich, ein Basic-Programm mit Hilfe des Editors zu schreiben und zu korrigieren, auf Band zu laden und von dort in den Basic-Interpreter zu holen. Umgekehrt kann man auch ein mit dem Basic-Interpreter geschriebenes Programm per Kassettenrecorder in den Texteditor laden.

Diese Übereinstimmung des Kassettenformats bei Basic und Texteditor hat allerdings auch Nachteile: Im Gegensatz z.B. zum PET-2001 werden Basic-Programme nämlich nicht in komprimierter Form, d.h. mit hexadezimal codierten Basic-Befehlen, sondern in Form ihrer einzelnen ASCII-Zeichen auf Band geladen. Für den Befehl GOTO 200 überträgt der AIM-65 daher sieben Zeichen, der PET-2001 aber nur vier. (Allerdings kann der PET-2001 Programme nicht mischen.) Dieser Nachteil von AIM-65 und PC-100 läßt sich vermeiden, wenn man beim Abspeichern eines Basic-Programms wie folgt vorgeht:

1. Mit der Escape-Taste aus dem Basic-Interpreter "aussteigen".
2. Den Inhalt der hexadezimalen Adressen 75 und 76 ermitteln und notieren.
3. Mit Taste D Abspeicherung auf Kassette initialisieren (Dump). Dabei ist zunächst der Bereich ab 0200 bis zu der in 76/75 stehenden Adresse abzuspeichern.

4. Nach MORE? den Adressenbereich 0000...00D6 abspeichern; nach dem zweiten MORE? nur N drücken.

Das Basic-Programm befindet sich jetzt in komprimierter Form nebst den notwendigen Zero-Page-Zellen auf der Kassette. Es kann jetzt allerdings nicht mehr mit LOAD von Basic aus geladen werden. Stattdessen ist es vom Monitorprogramm aus mit Taste L wie ein Maschinenprogramm zu laden. Dann kann man mit Taste 6 zum Basic-Interpreter springen, ohne das gerade geladene Programm wieder zu löschen. Die Methode bietet auch den Vorteil, daß der Mikrocomputer zwischen den einzelnen Datenblöcken weniger "Totzeit" benötigt, so daß der Gap-Wert (A409) nicht größer als sein ursprünglicher Wert 08 gemacht werden muß.

3.3 ABC-80

Der von der schwedischen Firma Luxor gebaute und von Databoard entwickelte Mikrocomputer ABC-80 besitzt von allen in diesem Buch vorgestellten Personal Computern außer dem PC-1000 den komfortabelsten Basic-Befehlssatz. Der ABC-80 wurde 1979 eingeführt und arbeitet mit der CPU Z-80. In der Grundausstattung stehen 16 KByte RAM zur Verfügung. Eine besonders erwähnenswerte Eigenschaft ist, daß alle in der deutschen Sprache vorkommenden Buchstaben (auch ä, ö, ü) als Tasten vorhanden sind, so daß eine Textverarbeitung ohne Kompromisse möglich ist.

3.3.1 Hardware

Der ABC-80 *(Abb. 3.3.1)* besteht aus einer Tastatur, in deren Gehäuse auch der Z-80-Mikrocomputer untergebracht ist, einem davon getrennten und über eine Videoleitung verbundenen Schwarzweiß-Fernsehmonitor und einem ebenfalls externen Kassettenrecorder als Grundausstattung. Der für die Erzeugung des Videosignals benutzte Zeichengenerator besitzt zwar alle in Europa üblichen Buchstaben, nicht aber das Dollar-Zeichen; es ist hier durch "¤" ersetzt (dies ist bei der Syntax von Stringbe-

Abb. 3.3 Der ABC-80 von Luxor besitzt einen vergleichsweise recht komfortablen Basic-Befehlssatz und arbeitet mit der CPU Z-80

fehlen zu beachten). Auf dem Bildschirm mit 30 cm Diagonale lassen sich 24 Zeilen mit je 40 Zeichen darstellen. Für Kurvendarstellungen oder andere Grafiken läßt sich der Bildschirm auch in 72 Grafik-Zeilen mit je 78 Positionen einteilen, wobei sich einzelne Punkte durch Angabe ihrer Koordinaten setzen lassen. Der ABC-80 gestattet auch das Erzeugen der unterschiedlichsten Toneffekte durch drei softwaregesteuerte Tongeneratoren, die sich beliebig mischen lassen.

Wie beim AIM-65 kann der Motor des Kassettenrecorders per Programm gesteuert werden. Der externe Anschluß von Meßgeräten u.a. mit IEC-Bus ist im Grundzustand nicht möglich und erfordert ein zusätzliches ROM sowie ein Hardware-Interface.

Eine kleine Erweiterung ist auch für den Anschluß seriell arbeitender Peripherie nötig: Die UART-Platine gestattet nach dem Einbau die Kommunikation mit einem externen Terminal, einem Drucker oder auch einem Telefon-Modem, das ebenfalls als Zubehör lieferbar ist.

Eine Eigenschaft, die man irrtümlich bei anderen Computern oft voraussetzt und sich dann über mancherlei Seltsames wundert, ist beim ABC-80 durchaus erwähnenswert: Die intern für die Bildschirmdarstellung verwendeten Zeichencodes entsprechen der weiter hinten in diesem Buch abgedruckten ASCII-Norm. Abweichungen ergeben sich natürlich bei den europäischen Zeichen; sie werden wie folgt codiert:

Zeichen	Dez. Code	ASCII-Äquiv.
£	35	#
¤	36	$
E	64	@
Ä	91	[
Ö	92	'
Å	93	]
Ü	94	↑
e	96	\
ä	123	[
ö	124	|
å	125	]
ü	126	~
■	127	DEL

Mit bestimmten Steuerzeichen ist es möglich, Teile des Bildschirms für grafische Darstellungen zu nutzen. Welche Teile das sind, kann per Software frei bestimmt werden. Insgesamt stehen 56 unterschiedliche Grafik-Symbole zur Verfügung, die Kombinationen aus einer Matrix von 2 x 3 Quadraten innerhalb eines Zeichenrechteckes darstellen. Die Software des ABC-80 sorgt selbst dafür, daß beim Setzen dicht nebeneinander liegender Grafikpunkte mit dem SET-Befehl die entsprechende Zeichenform ausgewählt wird, so daß die schon erwähnte hohe Auflösung bei Grafiken zustandekommt.

Wenn man davon absieht, daß eine farbige Darstellung nicht möglich ist, entspricht der im ABC-80 verwendete Zeichencode dem im Bildschirmtext-System der Deutschen Bundespost verwendeten Code. Zusammen mit dem Telefon-Modem kann der ABC-80 daher als Bildschirmtext-Endgerät betrieben werden.

3.3.2 Speicheraufteilung

Da eine andere CPU verwendet wird, sieht die Speicherbelegung beim ABC-80 völlig anders aus als beim AIM-65 oder PET-2001. Die in unserer Übersicht angegebenen Adressen sind hier sowohl dezimal als auch hexadezimal angegeben.

Dez.	hexadez.	Inhalt
0-	0-	16 KByte ROM mit Basic-Interpreter
16384-	4000-	8 KByte ROM (in der Grundausstattung nicht belegt)
24576-	6000-	4 KByte ROM für Floppy-Disk-Steuerung (Option)
28672-	7000-	1 KByte ROM für IEC-Bus-Steuerung (Option)
29696-	7400-	1 KByte nicht belegter Speicherplatz
30720-	7800-	1 KByte ROM für die Drucker-Ansteuerung (Option)
31744-	7000-	1 KByte RAM als Bildschirmspeicher (Video-RAM)
32768-	8000-	16 KByte externe RAM-Erweiterung (Option)
49152-	0000-	16 KByte internes RAM, nutzbar bis etwa dez. 64000
65408-	FF80-	128 Byte für Maschinenprogramme oder Drukker-Puffer
65535	FFFF	

Abgesehen von diesen Speicheradressen gibt es beim Z-80 noch besondere Adressen für den Eingabe-Ausgabebaustein (Z-80-PIO). Der ABC-80 belegt sie (dezimal) wie folgt:

Adr.	Funktion	
0	Datenausgabe und -eingabe	Anschlüsse des externen ABC-80-Bus
1	Kanalwahl-Ausgabe und Status-Eingabe	
2...5	Kommandos C1...C4	
6	Tonerzeugungs-Baugruppe	
56	Dateneingang von der Tastatur	
57	Tastatur-Kontrolleingang	
58	Dateneingang für Modem und Kassetteninterface, Relaisausgang	
59	Kontrolleingang für Modem und Kassetteninterface	

3.3.3 Monitorprogramm

Obwohl – wie wir noch sehen werden – der Basic-Interpreter des ABC-80 zu den leistungsfähigsten überhaupt gehört, enthält dieser Computer leider kein Monitorprogramm wie der AIM-65, um Programme in seiner Prozessor-Maschinensprache eingeben zu können.

Es ist allerdings ein gewisser Trost, daß zahlreiche sonst selten aufzufindende Basic-Befehle für Spezialaufgaben oft die Notwendigkeit beseitigen, Maschinenprogramme zu schreiben, z.B. für das Umnumerieren von Basic-Programmzeilen u.a. Wenn man sich den Luxus einer Floppy-Disk erlaubt, kann man trotzdem in Maschinensprache programmieren, da auf diesem Speichermedium geeignete Software (Assembler usw.) erhältlich ist. Eine Alternative ist auch das Schreiben von Maschinenprogrammen mit Hilfe eines in Basic geschriebenen Monitorprogrammes, wofür das ABC-80-Handbuch ein einfaches Beispiel enthält.

3.3.4 Basic-Interpreter

Der ABC-80 enthält in seinen ROMs keinerlei Hilfsprogramme für die Programmierung in Maschinensprache. Dafür ist sein Basic-Befehlssatz äußerst leistungsfähig, wie aus der folgenden Aufstellung hervorgeht. ("Kommandos" sind Befehle, die nur direkt von der Tastatur aus, nicht aber innerhalb eines Programmes ausgeführt werden können.)

Kommandos

CLEAR	ED
RUN	SAVE
LIST	LOAD
LIST file	MERGE
REN	NEW/SCR

Befehle

FOR	INPUT
STEP	INPUT£
NEXT	INPUTLINE
ON GOTO	INPUTLINE£
ON GOSUB	DATA
ON RESTORE	READ
ON ERROR GOTO	RESTORE
STOP	DIM
END	GOTO
OPEN AS FILE	IF – THEN
PREPARE AS FILE	ELSE
CLOSE	GOSUB
CHAIN	RETURN
SETDOT	RANDOMIZE
CLRDOT	REM
TRACE	DEF FN
NOTRACE	POKE
LET	OUT
PRINT	GET
PRINT£	

Funktionen

SIN	MID$	SWAP%
COS	LEN	ERRCODE
TAN	ASC	DOT
ATN	CHR$	NOT
LOG	INSTR	OR
LOG10	SPACE	AND

EXP	STRING$	XOR
SQR	VAL	IMP
INT	NUM$	EQV
FIX	PEEK	ADD$
ABS	INP	SUB$
SGN	TAB	MUL$
RND	FN	DIV$
PI	CUR	COMP%
LEFT$	CALL	
RIGHT$		

Die Interpreter-Software sorgt dafür, daß bestimmte Programmierfehler schon bei der Eingabe erkannt werden. Die Indikation von Fehlern erfolgt beim ABC-80 allerdings nicht (wie z.B. beim PET) in Klartext, sondern als Code-Zahl. Dies geschah, weil der Interpreter die Fehler sehr genau differenziert und rund 70 unterschiedliche "Error Codes" ausgeben kann. Normalerweise führen Fehler zum Abbruch des Programmablaufes; wenn sie aber mit ON ERROR GOTO... behandelt werden, wird das Programm z.B. mit einer individuellen Klartext-Fehlermeldung fortgesetzt, die der Anwender selbst schreiben kann. Dies ist allerdings nur bei etwa 20 Fehlerarten möglich, namentlich bei Kassetten- oder Floppy-Operationen sowie bei falschen Eingaben.

Eine Besonderheit des ABC-80-Basic ist schließlich noch, daß nach dem INPUT-Befehl kein auszugebender Text in Anführungszeichen stehen darf. Die Befehlsform

```
INPUT"WERT EINGEBEN";A
```

ist beim ABC-80 zu ersetzen durch:

```
PRINT"WERT EINGEBEN";:INPUT A
```

3.4 PET-2001 und CBM-3001

Der Mikrocomputer PET-2001 war der erste weiter verbreitete Personal Computer auf dem deutschen Markt. Der Hersteller Commodore hatte kurz vor seiner erstmaligen Vorstellung jene Firma übernommen, die den darin enthaltenen Mikroprozessor

6502 entwickelt hatte – nämlich MOS Technology Inc. – und hatte zunächst große Lieferprobleme, weil die Nachfrage nach dem von einem gewissen Chuck Peddle entwickelten Gerät riesengroß war.

Der Erfolg des PET-2001 ist vor allem auf zwei Eigenschaften zurückzuführen: auf den IEC-Bus-Anschluß, der eine Verwendung des Computers als zentrale Steuerung eines Parks von maximal 15 Meßgeräten oder anderer Peripherie gestattet, und seinen "Alles-in-einem"-Aufbau: Das (Störstrahlungen wirksam abschirmende) Metallgehäuse enthält außer dem Computer selbst die Eingabetastatur, den Bildschirm und einen Kassettenrecorder. Gespart wurde beim PET vor allem an der Eingabetastatur, die zwar ein getrenntes Ziffernfeld enthält, deren Anordnung aber ein schnelles Tippen für schreibmaschinen-gewohnte Leute völlig unmöglich machen. Auch die Kleinschreibung ist zwar im Prinzip möglich; im Gegensatz zur Schreibmaschine werden Kleinbuchstaben aber dann erzeugt, wenn die Shift-Taste gedrückt ist.

Commodore erkannte dieses Problem, das die Verwendung des PET für Textverarbeitungs-Aufgaben praktisch unmöglich macht, und brachte 1979 die CBM-Serie heraus. Dabei handelt es sich um Computer, die sich vom "Ur-PET" dadurch unterscheiden, eine Tastatur voller Größe und einen grünleuchtenden Bildschirm enthalten *(Abb. 3.4.1)*. Der Kassettenrecorder hatte keinen Platz mehr im Gehäuse, meist ist aber (bei ernsthafter Verwendung) ohnehin eine Floppy-Disk als Massenspeicher sinnvoller. Die CBM-Computer gestatten auch eine "normale" Groß- und Kleinschreibung und werden mit RAM-Kapazitäten von 16 KByte (CBM 3016) oder 32 KByte (CBM 3032) geliefert. Wie der Vorgänger PET-2001 enthalten auch sie einen IEC-Bus-Anschluß.

Die folgenden Angaben beziehen sich primär auf den sehr weit verbreiteten PET-2001 (Commodore muß zu diesem Gerät aufgrund eines Rechtsstreites in letzter Zeit "Personal Electronic Transactor" sagen). Die CBM-Serie unterscheidet sich vom PET leider etwas in der Speicheraufteilung, ist sonst aber völlig kompatibel.

Abb. 3.4.1 Der CBM-3032 besitzt 32 KByte RAM und erfreut sich großer Verbreitung auch bei industriellen Anwendern: Er verfügt nämlich über einen IEC-Bus-Anschluß

Die Zeilenbreite beträgt bei den Computerserien PET-2001 und CBM-3001 jeweils 40 Zeichen. Dies ist für professionelle Aufgaben oft zu wenig. Commodore brachte daher Mitte 1980 die Serie CBM-8001 heraus, die einen geringfügig größeren Bildschirm verwendet, auf dem sich 2000 Zeichen (25 Zeilen mit je 80 Zeichen) darstellen lassen. Die übrigen Eigenschaften dieser Serie (z.B. CBM-8032 = 32-K-Version) entsprechen weitgehend der CBM-3001-Serie, so daß hier nicht näher darauf eingegangen werden muß. Erwähnt sei lediglich, daß die Speicherbelegungen der 3001- und 8001-Versionen leider wieder einmal nicht identisch sind, so daß die Kompatibilität nur für Basic-, nicht aber für Maschinenprogramme oder PEEKs und POKEs gewährleistet ist.

3.4.1 Hardware

PET-2001 *(Abb. 3.4.2)* und CBM-3032 arbeiten mit dem Mikroprozessor 6502, der mit 1 MHz Taktfrequenz betrieben wird.

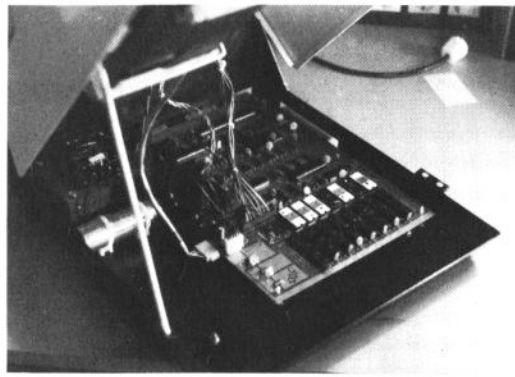


Abb. 3.4.2 Das Innere des Mikrocomputers PET-2001. Auf der linken Platinenseite befinden sich Stecker für den Kassettenrecorder und den Videoteil

Für den Verkehr mit der Außenwelt dient ein "VIA" 6522, das zwei 8-bit-Ports, mehrere Steuerleitungen und zwei Timer enthält. Einer der Timer wird dazu verwendet, im Abstand von 1/60 Sekunde einen Interrupt (IRQ) zu generieren; die dann jeweils angesprungene Interrupt-Routine stellt die Software-Uhr um 1/60 s weiter und fragt auch das Tastenfeld ab, z.B. um festzustellen, ob die Break- bzw. Stop-Taste gedrückt ist.

Der im PET bzw. CBM eingebaute Bildschirm arbeitet mit 60 Hz Bildwechselfrequenz, entspricht also der amerikanischen Fernsehnorm. Anschlüsse für Videosignal und Synchronimpulse sind herausgeführt; wenn man diese Signale entsprechend mischt, kann man direkt den Video-Eingang eines Fernsehgerätes amerikanischer Norm anschließen. Bei deutschen Geräten empfiehlt sich die Verlängerung der Synchronimpulse mit einem Monoflop; durch Verstellen des Vertikal-Bildfangs ist dann meist noch eine Synchronisation mit 60 Hz möglich.

Manchmal erkennt man auf dem PET-Bildschirm ein deutliches, schnelles Zittern des gesamten Bildinhaltes. Dies ist meist

eine Folge magnetischer Einstreuungen auf die Bildröhre, wobei ein Zittern mit der Differenz zwischen der 50-Hz-Netzfrequenz und der 60-Hz-Bildwechselfrequenz auftritt. Man sollte deshalb die Aufstellung von Geräten mit leistungsfähigen Netztrafos direkt neben dem Computer vermeiden.

Im Gegensatz z.B. zum TRS-80 wurden auch bei Dauerbetrieb bislang keine Wärmestau-Probleme beobachtet. Dies ist vor allem auf den großzügigeren Aufbau des PET zurückzuführen. Die Großzügigkeit hört allerdings beim Netzteil auf: Es kann zu Problemen führen, wenn dem PET-Netzteil extern mehr als etwa 200 mA zusätzlich entnommen werden. Andererseits muß man beim Anschluß extern versorgter Peripherie an den PET sehr vorsichtig sein. Ist nämlich der Computer ausgeschaltet, während von außen Spannung an die 6522-Anschlüsse gelangen kann, so fließt in diesem Interface-Baustein u.U. ein Ausgleichsstrom über die integrierten Schutzdioden, der Beschädigungen des VIA hervorrufen kann – und tatsächlich gehört dieser Baustein zu jenen PET-Bauelementen mit der höchsten Ausfallwahrscheinlichkeit.

Während im PET 2001 ein Kassettenrecorder links neben der etwas kleinen Tastatur eingebaut ist, muß ein solcher extern an den CBM 3001 angeschlossen werden, da wegen der größeren Tastatur kein Platz mehr für ihn war. Ein zweiter Kassettenrecorder kann zusätzlich z.B. für komfortable File-Verarbeitung oder für das Kopieren von Kassetten angeschlossen werden.

Das verwendete Kassetten-Aufzeichnungsformat arbeitet mit drei unterschiedlichen Frequenzen. Eine (nur eine!) Schwingung kennzeichnet durch ihre Frequenz drei Möglichkeiten: 1471 Hz dienen zur Kennzeichnung eines Byte-Anfangs; 1961 Hz werden als "Long", 2778 Hz als "Short" bezeichnet. Die Folge Short-Long ergibt ein Null-Bit, die Folge Long-Short ein Eins-Bit. Jedes Byte beginnt mit 1471 Hz und 1961 Hz. Für ein gesamtes Byte ergibt sich dadurch eine Übertragungsdauer von 8,15 ms, was etwa einem 1200-Bd-Kansas-City-Interface gleichkommt und recht schnell ist. Außerdem beinhaltet das Format eine gewisse Redundanz, so daß es sehr störsicher funktioniert.

Probleme beim Austausch von Kassetten-Software mit anderen PET-Anwendern sind praktisch immer auf eine unterschiedliche Neigung des Tonkopf-Spaltes im Recorder zurückzuführen. Das Problem läßt sich beseitigen, wenn man über einen Vorwiderstand von z.B. 1 kΩ einen Lautsprecher an den Ausgang des Kassettenrecorders anschließt und die Taumelschraube des Tonkopfes auf optimale Wiedergabe justiert. Ohne Mithörmöglichkeit ist vom Drehen an dieser Schraube dringend abzuraten, da man sonst u.U. die richtige Einstellung nicht wiederfindet.

Im Gegensatz zu den Computern AIM-65 und PC-100 zeichnet der PET die Basic-Befehle in komprimierter, hexadezimal codierter Form auf. Ferner wird der Arbeitsspeicher nach jedem LOAD-Befehl gelöscht, so daß es nur mit Tricks möglich ist, Programme miteinander zu mischen. Für den Benutzer stellt es eine Vereinfachung dar, nicht NEW drücken zu müssen, wenn noch ein Programm im Speicher steht und ein neues geladen werden soll.

3.4.2 Speicheraufteilung

Wie der AIM-65 arbeitet auch der PET-2001 mit der CPU 6502, die mit ihren 16 Adressenleitungen rund 65 KByte adressieren kann. Die Belegung der "Zero Page" (0000...00FF) wurde bereits bei der Besprechung des AIM-65 dargestellt. Der übrige PET-Speicher sieht folgendermaßen aus:

0200-0202	Uhrzeit (binär, Stunden/Minuten/Sekunden)
0203	Matrix-Koordinaten der zuletzt gedrückten Taste
0204	1 bei gedrückter Shift-Taste
0205-0206	Software-Zähler für Timer
0207	Steuerung von Kassette 1
0208	Steuerung von Kassette 2
0209-020C	Versch. Kassetten-Funktionen
020E	Reverse-Flag f. Zeicheninversion
020F-0218	Tastenfeld-Eingangspuffer
0219-021A	Vektor bei Hardware-Interrupt
021B-021C	Vektor bei Break-Interrupt
225	Cursor-Timing (Blinkzeit)

228-279	Versch. Funktionen f. Kassette und Files
027A-0339	Kassettenbuffer 1
033A-03F9	Kassettenbuffer 2
0400	Start des Basic-Programms
0FFF	RAM-Ende bei der 4-KByte-Version
1FFF	RAM-Ende bei der 8-KByte-Version
7FFF	Ende der maximalen RAM-Erweiterung (32 KByte)
8000-83E7	Video-RAM (dezimal 32768-33767)
9000-BFFF	Frei für ROM-Erweiterungen
C000-E0B0	Microsoft-Basic in ROMs
E0B5-E27D	System-Initialisierung nach Reset
E294-E66A	Bildschirmsteuerungs-Software
E66B-E684	Interrupt-Behandlungs-Software
E685-E758	Uhr-Programm, Tastenfeld-Abfrage (mit 60-Hz-Interrupt)
E75C-E7D4	Umwandlungstabelle Tastenfeld zu Zeichencode
E800-EFFF	PIA- und VIA-Adressen
FD38-FFB2	Diagnostik-Routinen
FFC0-FFEC	Sprungvektoren f. Ein/Ausgabe usw.
FFFA-FFFF	Interrupt-Vektoren (NMI nicht verwendet)

3.4.3 Monitorprogramm

Der PET-2001 besitzt kein Monitorprogramm fest gespeichert. Für die Eingabe von Maschinenprogrammen müssen daher die Basic-Befehle PEEK und POKE verwendet werden, wie das auch beim ABC-80 der Fall ist. Der Hersteller Commodore liefert jedoch das Monitor-Programm "TIM" (Terminal Interface Monitor) auf Kassette; es ist in Maschinensprache geschrieben. Dabei handelt es sich um eine an den PET-2001 adaptierte Version eines Programms, das ursprünglich als IC-6530-003 für 6502-Minimalsysteme von MOS Technology Inc. entwickelt worden war. Er gestattet das Anzeigen und Ändern von Speicherplätzen, das Setzen von sog. Breakpoints sowie das Laden und Abspeichern von Maschinenprogrammen auf Kassette. Dazu dienen folgende Befehle:

M	Display Memory	Speicherinhalt anzeigen
R	Display Registers	Registerinhale der CPU anzeigen
G	Go	Maschinenprogramm ausführen
X	–	Rücksprung zu Basic
S	Save	Programm auf Kassette aufzeichnen
L	Load	Programm von der Kassette laden

Wie gesagt ist TIM nicht fest im PET-2001-ROM gespeichert. Dagegen enthalten die ROMs einige nützliche Routinen, die bei der Programmierung in Maschinensprache oft verwendet werden:

FFD2	Ausgabe eines ASCII-Zeichens auf dem Bildschirm
FFCF	Warten, bis eine Taste gedrückt ist, und Wert einlesen
FFE4	Taste einlesen, ohne das Programm anzuhalten
FFE1	Stop-Taste abfragen (wenn gedrückt, Z-Flag im 6502 setzen)
D0A7	Fließkomma- in Ganzzahl umwandeln (Ergebnis in B3, B4)
D278	Ganzzahl in Fließkommazahl umwandeln (Ganzzahl in A, Y)

Wenn TIM von der Kassette geladen ist, stehen zusätzlich folgende Unterprogramme zur Verfügung:

04F2	CRLF	Neue Zeile
063B	SPACE	Leerraum ausgeben (2 x)
0613	WROB	Byte ausgeben (zwei Hex-Zeichen)
0660	RDOB	Ein Byte von der Tastatur holen (zwei Hexzeichen)
0687	HEXIT	Umwandlung eines ASCII-Zeichens in eine Hex-Ziffer

Das PET-Monitorprogramm TIM steht dabei im Adressenbereich 400...76C und belegt die Zero-Page-Adressen 0004...0022. Solange kein zweiter Kassettenrecorder betrieben wird, schreibt man eigene Maschinenprogramme am besten in den zweiten Kassetten-Buffer ab 033A, da das RAM ab 0400 von Basic be-

legt wird. Der Wiedereintritt in den Basic-Interpreter kann über die Adresse C38B erfolgen, ohne Programme oder Variablen zu löschen. Beim CBM wird TIM mit SYS (1024) gestartet.

3.4.4 Basic-Interpreter

PET-2001 und CBM-3032 sind untereinander absolut softwarekompatibel, und nur einige ROM- und Zero-Page-Adressen sind bei ihnen unterschiedlich belegt. Das PET-Basic wurde von Microsoft entwickelt und ist leider nicht ganz so komfortabel wie z.B. jenes des schon besprochenen ABC-80. Trotzdem muß man zugestehen, daß der PET-Basic-Befehlssatz weit mehr Komfort bietet als der zunächst im Ur-Basic vorgesehene. Dies gilt insbesondere auch für die Textverarbeitung (Strings) – mit einer Ausnahme: Dem PET fehlt ein Befehl zum schnellen Suchen nach einem Stichwort innerhalb eines längeren Strings, wie das bei TRS-80 und ABC-80 mit INSTR möglich ist.

Da im PET mit "Memory-mapped"-I/O-Bausteinen gearbeitet wird, gibt es im Gegensatz zu diesen Computern auch keine speziellen I/O-Befehle für die Ports. Vielmehr lassen sich alle Register der VIA 6522 mit PEEK und POKE ansprechen (Adressen siehe S. 155).

Die Ausgabe von Daten und Steuersignalen über den *IEC-Bus* ist mit PRINT#, gefolgt von der jeweiligen File-Nummer, möglich. Da auch die "intelligente" Floppy-Disk-Einheit von Commodore über den IEC-Bus angeschlossen ist, kann man ohne weitere Disk-Befehle nur mit PRINT# und INPUT# sehr komfortabel Variablen auf Disk abspeichern und zurückladen. Für das Abspeichern und Laden von Programmen dienen die Befehle SAVE und LOAD, gefolgt von dem jeweiligen Programmnamen. Will man z.B. ein Programm von einer frisch eingelegten Floppy laden, so ist folgendes zu tun:

```
OPEN 1,8,15
PRINT#1,"I1"          Disk initialisieren (linkes Laufwerk="1")
LOAD"1:EDITOR",8      Programm "Editor" von der linken
                      Floppy laden
RUN                   Programm starten
```

Recht dialogfreundlich ist beim PET-Basic, daß Fehlermeldungen nicht als schwer zu entschlüsselnder Zifferncode, sondern in (englischem) Klartext ausgegeben werden. Auch findet bei Kassettenoperationen eine regelrechte Bedienerführung mit solchen Texten wie "PRESS PLAY ON TAPE 1", "LOADING" oder "PRESS RECORD AND PLAY ON TAPE 1" statt. Solche Dinge machen den Umgang mit dem PET auch für Laien ausgesprochen problemlos.

Leider hat dieser Computer aber auch seine Tücken. Der PET-2001 besitzt einen achtzig Zeichen langen *Eingangspuffer*. Das bedeutet, daß eine Programmzeile oder eine Eingabe des Benutzers beim INPUT-Befehl maximal 80 Zeichen umfassen darf.

Eine bestimmte Problematik ist in diesem Zusammenhang, daß der Eingangspuffer sozusagen auf dem Bildschirm ist – er ist ein Teil des Video-RAM. Die Folge hiervon ist, daß beim Befehl INPUT M$, der am linken Zeilenrand ein Fragezeichen produziert, unter bestimmten Bedingungen das Fragezeichen mit in die Stringvariable M$ übernommen wird. Dieses Problem tritt insbesondere dann auf, wenn Korrekturen mit dem Cursor durchgeführt werden. Eine Lösung ist folgende Routine für die Texteingabe von M$:

```
420 INPUT M$
425 IFLEFT$(M$,1)=" "ORLEFT$(M$,1)="? "THEN
    M$=MID$(M$,2):GOTO425
```

Die Zeile 425 wird solange ausgeführt, bis alle Leerräume und Fragezeichen, die im eingegebenen String vor dem ersten gültigen Buchstaben stehen, beseitigt sind. (Es sei erwähnt, daß die dezimalen ASCII-Äquivalente von Leerraum und Fragezeichen nicht mit den im PET intern verwendeten Codes übereinstimmen!)

Doch hat der Bildschirm-Eingangspuffer auch seine Vorteile, gestattet er doch ein problemloses Korrigieren (Editieren) von Programmlistings durch Bewegen der Cursorposition. So ist es möglich, einen bestimmten Programmteil (auch mehr als eine Zeile) aufzulisten, mit dem Cursor Fehlerhaftes auszumerzen

und – ohne an das Ende der Zeile gehen zu müssen – durch Drücken der Return-Taste die Korrektur zu beenden. Allerdings muß man jetzt unbedingt mit dem Cursor an das untere Bildschirmende zurückgehen, da sonst versehentlich andere, ebenfalls aufgelistete Programmzeilen geändert werden oder Fehlermeldungen auftreten.

Abschließend seien noch einige *Programmiertricks* angegeben, die sich systemspezifisch auf PET-2001-Adressen beziehen und sich in dieser Form daher nicht bei anderen Computern anwenden lassen. Die Zeile

```
250 GET A$: IF A$=""GOTO 250
```

dient dazu, zu warten, bis eine beliebige Taste am PET gedrückt wird,und deren ASCII-Äquivalent in A$ einzulesen. Einfacher geht das mit:

```
250 WAIT 525,1: GET A$
```

Die Benutzung von *USR* zum Aufruf eines Maschinenprogrammes setzt voraus, daß dessen Adresse vorher in die Zellen 0001 und 0002 gespeichert wurde, und zwar in der Reihenfolge Low/High. Steht das Maschinenprogramm z.B. an der hexadezimalen Adresse 033A, so zerlegt man diese zunächst in 3A (Low) und 03 (High). Dann wandelt man die beiden Teile in ihre dezimalen Äquivalente um, was 58 und 3 ergibt. Vor USR muß also der Befehl stehen:

```
POKE 1,58:POKE 2,3
```

Das Befehlformat von USR lautet B=USR(A). Die Variable A findet sich beim Sprung zum Maschinenprogramm im Fließkommaformat in 00B0...00B5. Will man sie weiterverarbeiten, so ist es oft sinnvoll, sie in das Festkomma-Format umzuwandeln, wobei das höherwertige Byte in 00B3, das niederwertige in 00B4 steht. Zu diesem Zweck muß man im Maschinenprogramm mit dem Befehl JSR D0A7 (also 20 A7 D0) eine im ROM vorhandene Umwandlungsroutine aufrufen. Eine Routine zur Rückumwandlung in eine Fließkommazahl steht an der Adresse D278.

Im Gegensatz zu USR ist es bei SYS nicht möglich, Variablen zu übergeben. Dem steht der Vorteil von SYS gegenüber, daß die Adresse des Maschinenprogramms nicht vorher in die Zellen 0001 und 0002 gespeichert werden muß, sondern dezimal in Klammern hinter SYS angegeben werden kann; z.B. SYS(826) für ein Programm bei 033A.

Das *Abspeichern von Maschinenprogrammen* auf Kassette ist auch ohne die Verwendung des von Commodore angebotenen TIM-Monitors möglich. Dazu bringt man mit POKE das höherwertige Byte der Startadresse in 247, das niederwertige in 248 und die Endadresse in 229 und 230 (dezimal). Das Schreiben auf Kassette erfolgt mit SYS (63153). Will man einen File-Namen mit auf Kassette schreiben, so ist seine Länge in 238 und seine Anfangsadresse in 249 und 250 zu schreiben. Natürlich ist es möglich, dafür den INPUT-Buffer (siehe Zero-Page-Belegung, vgl. AIM-65) oder einen Teil davon zu verwenden.

Will man das Maschinenprogramm wieder von der Kassette laden, so ist das leider nicht mit LOAD möglich. Stattdessen muß man, wie oben geschildert, die Startadresse in 247 und 248 ablegen und die Laderoutine mit SYS (62306) aufrufen.

Oft ist es unerwünscht, daß sich ein Basic-Programm mit der *STOP-Taste* beenden läßt, z.B. auf Ausstellungen, wo manche Besucher mit Vorliebe STOP und NEW drücken. Beim PET-2001 läßt sich diese Taste mit POKE 537,136 unwirksam machen, indem der IRQ-Vektor umgesetzt wird. Allerdings ist dann auch die Uhr außer Betrieb.

Im Gegensatz zu anderen Computern, wie z.B. zum ABC-80, enthält der PET-2001 keinen *MERGE-Befehl,* um mehrere auf Kassette befindliche Programme in den Arbeitsspeicher zu laden, denn LOAD löscht alle bisherigen Programmzeilen. Bei kürzeren Programmen ist eine einfache Abhilfe möglich:

Man lädt zunächst das kürzere Programm und listet es auf dem Bildschirm auf – es muß auf ihm vollständig Platz haben. Dann lädt man das zweite Programm, ohne den Bildschirm zu löschen, geht mit dem Cursor an die erste Zeile des auf dem Bildschirm stehenden ersten Programms und drückt so oft die Return-Taste, bis man die letzte Zeile des auf dem Schirm auf-

gelisteten Programms überschritten hat. Nun kann man den Schirm gefahrlos löschen. Dieses Verfahren funktioniert natürlich nur dann, wenn in den beiden Programmen unterschiedliche Zeilennummern verwendet werden. Ist das nicht der Fall, so kann man gegebenenfalls das auf dem Schirm stehende erste Programm per Cursor so ändern, daß sich keine Überschneidungen mehr ergeben. Hier zeigt sich der große Vorteil, den Bildschirm als Eingangspuffer verwenden zu können.

Im Gegensatz zum ABC-80 bedeutet es bei PET, CBM und AIM-65 keinen merklichen Geschwindigkeitsvorteil, mit *Ganzzahl-Variablen* zu arbeiten (z.B. I% statt I). Zwar ist ihre Verwendung möglich und die Zahlen werden dann tatsächlich in einem 2-Byte-Format abgespeichert (−32768...+32767), das Fließkommaformat arbeitet jedoch bereits so schnell, daß es keine Geschwindigkeitseinbuße mit sich bringt.

3.5 TRS-80

Wenn hier vom TRS-80 die Rede ist, dann ist damit die Low-Cost-Ausführung gemeint und nicht etwa das deutlich teurere Modell II, das vorwiegend für einen Einsatz im professionellen Bereich gedacht ist.

In den USA gehört der TRS-80 nach wie vor zu den am weitesten verbreiteten "Personal Computern" – nicht zuletzt wegen seines verhältnismäßig geringen Preises.

3.5.1 Hardware

Bei Heimcomputern mit relativ geringer Software-Unterstützung seitens des Herstellers werden die Systemkosten zunächst einmal von der Hardware bestimmt, aus der der Computer aufgebaut ist. Der TRS-80 arbeitet mit der CPU Z-80, die 64 KByte mit 16 Adressenleitungen ansprechen kann. Sie wird mit 2 MHz Taktfrequenz betrieben; maximal wären 4 MHz möglich. Der TRS-80 gehört deswegen auch nicht gerade zu den schnellsten Basic-Rechnern.

Der mitgelieferte Bildschirm kann 16 Zeilen mit je 64 Zeichen darstellen; Grafiken sind mit einer Auflösung von 128 Punkten horizontal mal 48 Punkten vertikal möglich, so daß der kleinste darstellbare Punkt auf dem Monitor mit 31 cm Diagonale etwa 1,5 mm x 3 mm groß ist.

Der eigentliche Computer ist vom Monitor absetzbar und innerhalb des Tastatur-Gehäuses untergebracht, das auch das Netzteil enthält, allerdings bis auf den im Netzstecker untergebrachten Transformator. Bei Dauerbetrieb treten zuweilen Wärmeprobleme auf; der Hersteller Tandy bietet daher spezielle Arbeitstische an, die einen nach unten offenen Ausschnitt für das TRS-80-Gehäuse besitzen, so daß eine ausreichende Wärmezirkulation gewährleistet ist.

Ein ganz erheblicher Nachteil des TRS-80 ist die Tatsache, daß im Grundzustand keinerlei Möglichkeit vorhanden ist, direkt periphere Geräte anzuschließen. Dies ist erst nach dem Kauf eines besonderen Erweiterungs-Interface möglich, das auch einen Echtzeit-Taktgeber enthält, mit dem Zeiteinblendungen möglich werden. Ein Steckplatz auf diesem Interface ist z.B. für ein serielles RS-232-Interface frei. Der im TRS-80 selbst vorhandene I/O-Baustein ist mit Tastatur und Kassettenrecorder-Anschluß bereits voll ausgelastet und steht daher dem Anwender nicht zur Verfügung.

Der beim TRS-80 extern anschließbare Kassettenrecorder arbeitet je nach Basic-Version (Level I oder II) mit 250 Baud oder 500 Baud. Das Schreibverfahren arbeitet mit 400 μs langen Impulsen, deren Abstand voneinander bestimmt, ob es sich um log. 1 oder log. 0 handelt. Dabei werden jeweils die Abstände der positiven Flanken ausgewertet. Eine Null umfaßt nur einen 400-μs-Impuls; die Anstiegsflanke des ihm folgenden Impulses muß nach 2 ms auftreten. Eine Eins ist durch zwei 400-μs-Impulse charakterisiert, deren Anstiegsflanken 800 μs voneinander entfernt sind.

Dieses Verfahren ist weder so schnell noch so sicher wie das des PET 2001. Dies ist darauf zurückzuführen, daß der TRS-80 nicht über so komfortable Interface-Bausteine wie der PET 2001 verfügt; bei letzterem Computer kann das VIA 6522 die Inter-

face-Aufgaben ziemlich selbständig wahrnehmen, während beim TRS-80 die Z-80-CPU voll damit belastet wird.

Beim Abspielen von Programmkassetten erfordert der TRS-80 daher eine recht genaue Einstellung der Recorder-Lautstärke, da schon bei geringen Abweichungen Fehler auftreten können. Ein weiteres Problem ist dabei, daß solche Lesefehler manchmal – wenn auch selten – nicht erkannt werden, da nur sparsam mit Redundanz umgegangen wurde. Dies alles macht deutlich, daß die Hardware des TRS-80 in erster Linie auf Preiswürdigkeit "gezüchtet" ist.

3.5.2 Speicheraufteilung

Da es beim Level-I-Basic des TRS-80 kaum Möglichkeiten gibt, Maschinenprogramme zu schreiben, wollen wir uns bei der Betrachtung des TRS-80-Speichers auf die Verhältnisse beim Level-II-Basic beschränken.

Adresse dez.	hexad.	Inhalt
0-	0-	Basic-ROM
12288-	3000-	Reserviert
14302	37DE	I/O-Flag (Status)
14303	37DF	I/O-Flag (Daten)
14304	37E0	Interrupt-Adresse
14305	37E1	Disk-Adresse
14308	37E4	Kassetten-Adresse
14312	37E8	Line-Printer-Adresse
14316	37EC	Disk-Controller-Adresse
14336-	3800-	Tastenfeld-Bereich
15360-	3C00-	Video-RAM
16384-	4000-	RST-1 ... 7-Vektoren
16402-	4012-	System-RAM mit I/O-Adressen
16870-	41E6-	I/O-Buffer-Bereich
17128-	42E8-	Basic-Programm, einfache Variablen, Matrix-Variablen unbenutztes RAM,

.		Stack,
.		Strings,
		Benutzer-Programme in Maschinensprache
20479	4FFF	Ende des 4-KByte-RAM
32767	7FFF	Ende des 16-KByte-RAM

3.5.3 Monitorprogramm

Der TRS-80 besitzt weder in Level I noch in Level II ein fest gespeichertes Monitorprogramm für die Eingabe und das Testen von Programmen, die in der Z-80-Maschinensprache geschrieben sind. Allerdings sind solche Programme sowohl auf Kassette als auch auf Floppy-Disk vom Hersteller zu haben. Bemängelt wird ab und zu, daß die Beschreibung von verwendbaren Unterprogrammen des ROM z.B. zum Schreiben von Zeichen auf den Bildschirm oder für die Tastaturabfrage im TRS-80-Handbuch sehr mangelhaft ist. Dies gilt allerdings – außer für den AIM-65/PC-100 – für praktisch alle heutigen Basic-Computer. Eine praktische Sache sei noch erwähnt: Bei Basic-Befehlen, die Zeichen ausgeben sollen, wird nicht sofort das entsprechende Basic-Interpreter-Programm angesprungen, sondern der Sprung führt über zwei aufeinanderfolgende RAM-Zellen (Vektor). Dadurch hat der Benutzer die Möglichkeit, eigene Ausgabe-Routinen zu implementieren.

3.5.4 Basic-Interpreter

Der TRS-80 läßt sich mit zwei unterschiedlichen Basic-Interpretern bestücken, die vor allem in Rechengeschwindigkeit, Kassetten-Aufzeichnungsdichte und Befehlsvorrat differieren. Für professionelle Anwendungen kommt Level I praktisch kaum in Frage, da hier nur wenige Möglichkeiten zur String-Verarbeitung vorhanden sind und die Rechengenauigkeit nur sechs Stellen beträgt, was schon bei Beträgen von 10 000 DM ein Rechnen auf den Pfennig genau verhindert.

Vergleicht man den Befehlssatz der TRS-80-Basic-Interpreter mit dem des PET-2001, des ABC-80 oder des AIM-65/PC-100,

so stellt man erhebliche Syntax-Unterschiede für ein und denselben Befehl fest – ein Musterbeispiel dafür, wie Dialekte einer Programmiersprache untereinander differieren können, ohne daß dies technisch begründbar ist.

Level-I-Basic

Das in einem 4-KByte-ROM gespeicherte Level-I-Basic des TRS-80 wurde von Tandy geschaffen, um einen Computer mit möglichst geringen Hardware-Kosten auf den Markt bringen zu können, reicht aber für zahlreiche Anwendungen von Heimcomputern völlig aus.

Das Level-I-Basic arbeitet ohne Zwischencode; d.h. die vom Benutzer eingegebenen Basic-Befehle werden in ihrer vollen Länge gespeichert. So belegt das Befehlswort GOTO genau vier Byte – eines pro Zeichen. Um den Speicher trotzdem besser ausnutzen zu können, ist es möglich, häufig gebrauchte Befehlsworte mit einem Punkt abzukürzen, z.B. P. statt PRINT. Die Abspeicherung von Programmen auf Kassette ist mit 250 bit/s möglich, d.h. rund 25 Byte/s. Level I erlaubt Gleitkomma-Rechnungen mit einer Genauigkeit von sechs Stellen, ein numerisches Matrixfeld und zwei String-Variablen. Variablennamen dürfen nur ein Zeichen lang sein (z.B. A, D, G, Z).

Level-II-Basic

In vielen Fällen reicht der Befehlsvorrat von Level-I-Basic nicht aus, z.B. bei komfortabler String-Verarbeitung. Tandy schuf daher einen Level-II-Basic-Interpreter, der etwa dem des PET-2001 gleichwertig ist, wenn man einmal vom Fehlen der IEC-Bus-Anweisungen absieht. Er ist in 8 KByte ROM gespeichert und kann als Ersatz für Level I eingebaut werden. Die mathematischen Funktionen arbeiten zwar immer noch mit nur sechsstelliger Genauigkeit, eine Erweiterung bei den Grundrechenarten auf 16 Stellen ist aber per Programmbefehl möglich. Variablennamen dürfen aus bis zu zwei signifikanten Zeichen bestehen; wenn sie länger sind, werden nur die ersten zwei Zeichen berücksichtigt. Programmzeilen dürfen bis zu 255

Zeichen lang sein, sich also über maximal vier Bildschirmzeilen erstrecken. Die Abspeicherung von Programmen auf Kassette erfolgt doppelt so schnell wie bei Level I, nämlich mit 500 bit/s, was allerdings dazu führt, daß man früher mit Level I geschriebene Programme nicht ohne weiteres in einen Level -II-Computer einlesen kann; hier gibt es aber bereits die erforderliche Umwandlungs-Software auf Kassette.

Der Basic-Interpreter des Level-II-Basic gestattet die Verwendung von drei unterschiedlichen numerischen Variablen, nämlich ganzzahlig, mit einfacher und mit doppelter Genauigkeit. Die Variablen müssen deswegen am Programmanfang definiert werden, falls sie als Ganzzahl-Typ oder mit doppelter Genauigkeit verarbeitet werden sollen; dies geschieht mit DEFINT und DEFDBL. Eine Typenumwandlung ist innerhalb des Programms mit den Befehlen CDBL, CINT und CSNG in doppelte Genauigkeit, Ganzzahltyp und Normvariable mit einfacher Genauigkeit möglich.

Ausgabebefehle, denen ein L vorgestellt ist (z.B. LLIST oder LPRINT), sind auf einen eventuell angeschlossenen Drucker wirksam. Der Befehl SYSTEM schaltet schließlich den TRS-80 in den Maschinensprache-Level um, so daß Z-80-Maschinenprogramme von der Kassette geladen werden können.

Befehlsvorrat des TRS-80-Basic Level I

ABS(X)	INPUT #
CLOAD	INT(X)
CLS	LET
CONT	LIST
CSAVE	MEM
DATA	NEW
END	ON
FOR-TO-NEXT-STEP	POINT
GOSUB	PRINT
GOTO	PRINT AT
IF-THEN	PRINT #
INPUT	READ

REM
RESET
RESTORE
RETURN
RND(1)
RND(X)
RUN
SET
STOP
TAB

Zusätzlicher Befehlsvorrat des TRS-80-Basic Level II

ASC(A$)
ATN(X)
AUTO
CDBL(X)
CHR$(X)
CINT(X)
CLEAR
CLEAR(X)
CLOAD
CLOAD
COS(X)
CSAVE
CSNG(X)
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DIM
EDIT
ELSE
ERL
ERR
ERROR(X)
EXP(X)
FIX(X)
FRE(A$)
FRE(ϕ)
INKEY$
INP (X)
INPUT #-1
LEN(A$)
LEFT$(A$,4)
LLIST
LOG(A)
LPOS(ϕ)
LPRINT
MID$(X$,4,8)
NEW
OUT S,Y
PEEK(X)
POKE X,Y
POS (ϕ)
PRINT@
PRINT USING
RANDOM
RESUME
RIGHT$
SGN(X)
SIN (X)
SQR (X)
STR$(A)
STRING$
SYSTEM
TAN (X)
TRON

TROFF	VAL (A$)
USR (ϕ)	VARPTR (C)

Bei Level-I-Basic erzeugt der TRS-80-Interpreter keinen Zwischencode, d.h. die Befehle werden genauso als Folge von ASCII-Zeichen gespeichert, wie sie auf dem Bildschirm erscheinen. Um Speicherplatz zu sparen, sind folgende Abkürzungen zulässig:

Befehl	Abk.	Befehl	Abk.
PRINT	P.	TAB	T.
NEW	N.	INT	I.
RUN	R.	GOSUB	GOS.
LIST	L.	RETURN	RET.
END	E.	READ	REA.
THEN	T.	DATA	D.
GOTO	G.	RESTORE	REST.
INPUT	IN.	ABS	A.
MEM	M.	RND	R.
FOR	F.	SET	S.
NEXT	N.	RESET	R.
STEP	S.	POINT	P.
STOP	ST.	PRINT AT	P.A.
CONT	C.		

3.6 Apple-II, ITT-2020

3.6.1 Hardware

Der von der amerikanischen Apple Computer Co. konstruierte Mikrocomputer Apple-II ist nach wie vor eines der komfortabelsten und leider auch teuersten Geräte auf dem Markt. Er gestattet (auf einem externen, nicht mitgelieferten) Farbfernsehgerät die Darstellung von 25 Zeilen mit je 40 Zeichen (keine Kleinbuchstaben möglich) ebenso wie die Adressierung von 40 x 48 Grafikpunkten in 15 unterschiedlichen Farben. Mit

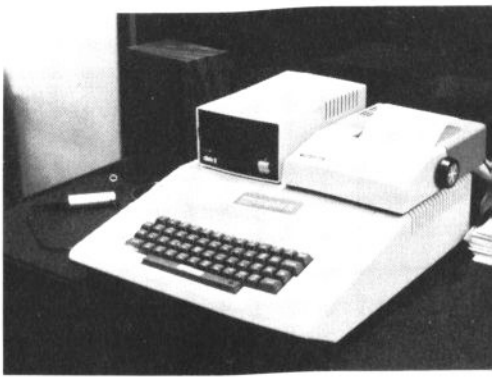


Abb. 3.6 Der Apple-II wird in unterschiedlichen Ausführungen für amerikanische und deutsche Farbfernsehnormen gebaut. Links steht eine kleine Floppy-Station auf dem Apple, rechts ein Drucker

8 KByte RAM (Grundausstattung 4 KByte) hat man schließlich auch die Möglichkeiten hochauflösender Grafik mit 280 x 192 Punkten in vier Farben.

Zunächst wurde der Apple-II *(Abb. 3.6)* nur für Farbfernsehgeräte der amerikanischen Norm gebaut, d.h. für das NTSC-Verfahren mit 60 Hz Bildwechselfrequenz. Mittlerweile steht aber sowohl von Apple selbst als auch von ITT (Pforzheim) eine an europäische Verhältnisse angepaßte Version zur Verfügung. Wenn man auf die Farbe verzichtet, genügt auch ein normaler Schwarzweiß-Fernsehempfänger.

Der Apple-II hat die Größe einer Reiseschreibmaschine; durch die Verwendung eines Schaltnetzteils mit geringer Verlustleistung treten kaum Wärmeprobleme auf. Der Aufbau ist sehr sauber, und intern ist noch eine Speichererweiterung bis zu 48 KByte RAM möglich.

Das eingebaute Kassetteninterface ist ausgesprochen schnell; die Aufzeichnung erfolgt mit 1500 bit/s, und der 10-KByte-Applesoft-Basic-Interpreter läßt sich dadurch in rund einer Minute von der Kassette laden. Ein 50poliger Peripherieanschluß gestattet nicht nur den Anschluß beliebiger I/O-Geräte, sondern auch externen Speicherzugriff (DMA) und komfortable Interrupt-Verarbeitungen.

Wie mehrere andere Mikrocomputer in diesem Buch arbeitet auch der Apple-II (bzw. der von ITT gefertigte ITT-2020) mit der 8-bit-CPU 6502, die mit 1 MHz Taktfrequenz betrieben wird.

3.6.2 Speicheraufteilung

Beim Apple-II gibt es unterschiedliche Möglichkeiten für das Programmieren in Basic; ein Ganzzahl-Interpreter (Integer-Basic) ist fest in ROMs gespeichert, während der komfortable Applesoft-Fließkomma-Interpreter entweder von einer Kassette geladen werden kann oder auch als "Firmware" in Form von zusätzlichen ROMs zur Verfügung steht. Der Applesoft-Interpreter umfaßt 10 KByte; die Kassettenversion setzt daher 16 KByte RAM voraus.

Hex-Adresse	Inhalt
0-1FF	System-RAM (nicht frei verwendbar)
200-2FF	Buffer für Tastatur-Eingabe (255 Zeichen)
300-3FF	steht für Maschinensprache-Routinen zur Verfügung
400-7FF	Video-RAM, Bildschirmseite 1 (Seite 2: 800-BFF)
800-2FFF	Kassettenversion des Applesoft-Interpreters oder Variablenraum bei der Firmware-Version
2000-3FFF	Bildschirmseite 1 für Grafiken hoher Auflösung (nur bei der Firmware-Version von Applesoft)
3000-3FFF	Variablenraum beim Kassetten-Basic
4000-5FFF	Bildschirmseite 2 für Grafiken hoher Auflösung
C000-CFFF	I/O-Bausteine
D000-DFFF	Frei für spätere ROM-Erweiterung
E000-F7FF	Ganzzahl-Basic-Interpreter (Integer Basic) in ROMs

F800-FFFF System-Monitorprogramm mit Disassembler usw.

Das Abspeichern von Zeichen im Video-RAM erfolgt im 6-bit-Format; der Buchstabe A wird z.B. als hex 01 abgespeichert. Bit 6 (das siebte Bit) dient dazu, das Zeichen blinken zu lassen, und Bit 7 gestattet die Darstellung einzelner schwarzer Zeichen auf weißem Grund. Ein inverses A wäre also als hex 81 gespeichert, und ein blinkendes A als hex 41. Bei der Verwendung der Monitor-Unterprogramme zur Zeichen-Ein- und Ausgabe braucht man sich darum jedoch nicht zu kümmern und kann vom gewöhnlichen ASCII-Format ausgehen. Die wichtigsten Monitor-Unterprogramme des Apple-II sind:

Bezeichnung	*Adresse*	*Wirkung*
CLRSCR	F832	Bildschirm löschen
INSTDSP	F8D0	Maschinenbefehl disassemblieren, dessen Adresse in den Zellen 003A, 003B steht
PRNTYX	F940	Inhalt von X- und Y-Register als vier Hex-Ziffern ausgeben
PRBL2	F94C	Leerräume ausgeben (X-Register = Anzahl)
SCROLL	FC70	Bildschirm eine Zeile nach oben "rollen"
CLREOL	FC9C	Rest der Zeile löschen
RDCHAR	FD35	Tastatur abfragen, ASCII-Zeichen holen
GETLIN	FD6A	Eine Zeile vom Tastenfeld in 0200...02FF einlesen
CROUT	FD8E	CRLF ausgeben (neue Zeile)
PRBYTE	FDDA	Akkuinhalt als zwei Hex-Ziffern ausgeben
COT	FDED	ASCII-Zeichen ausgeben
PRERR	FF2D	"ERR" ausgeben, Piepton erzeugen
BELL	FF3A	Piepton erzeugen
RESET	FF59	Kaltstartadresse des Monitors
MON	FF65	Warmstartadresse des Monitors
SWEET16	F689	Interpreter für simulierte 16-bit-CPU (siehe Zeitschrift "Byte" 11/1977, Seite 150-159)

3.6.3 Monitorprogramm

Der Komfort des Apple-II bzw. des ITT-2020 läßt sich bezüglich des Monitorprogramms, das der Programmierung in der 6502-Maschinensprache dient, am besten mit dem des AIM-65/PC-100 vergleichen. Weder PET, CBM, TRS-80 noch ABC-80 besitzen etwas Ähnliches.

Unter den 6502-Anwendern ist seit Jahren die Genialität des Apple-Disassemblers bekannt, eines Programmes, das nur ein halbes KByte umfaßt und in der Lage ist, hexadezimale Operationscodes in mnemonische Befehle zu übersetzen und so für den Benutzer einigermaßen verständlich auszugeben. Das Prinzip des Apple-Disassemblers wurde von den Entwicklern des AIM-65 nahezu unverändert übernommen.

Ebenso erwähnenswert ist der von Steve Wozniak 1976 entwickelte Interpreter für eine simulierte 16-bit-CPU, der im Apple fest im Monitor-ROM gespeichert ist und dem Anwender ermöglicht, die Maschinenbefehle der CPU 6502 durch leistungsfähige 16-bit-Operationen zu ergänzen. Dieser Interpreter trägt den klangvollen Namen "SWEET 16".

Die bei der Programmierung in Maschinensprache verwertbaren ROM-Routinen sind bereits im letzten Abschnitt aufgeführt. Die Monitor-Unterprogramme lassen meist das X-Register der CPU unverändert, nur Akku und Y-Register sind nach dem Aufruf verändert.

3.6.4 Basic-Interpreter

Für Apple-II und ITT-2020 existieren unterschiedliche Programmiersprachen. Zunächst muß darauf hingewiesen werden, daß die im vorletzten Abschnitt aufgeführten Adressenbelegungen nur für den ursprünglichen Applesoft-Fließkomma-Interpreter gelten. Inzwischen ist eine Reihe anderer Interpreter auf den Markt gekommen, für die diese Adressen nicht mehr vollständig gelten.

Im "Urzustand" enthält der Apple-II nur einen Ganzzahl-Basic-Interpreter, dessen Komfort sich noch am ehesten mit dem TRS-80-Level-I-Basic vergleichen läßt. Er rechnet nur mit

ganzen Zahlen und beherrscht die vier Grundrechenarten. Der komfortablere Fließkomma-Interpreter ist unter dem Namen "Applesoft" sowohl auf Kassette als auch in Form von PROMs erhältlich und stellt heute wohl die Standard-Programmiersprache für Apple-II und ITT-2020 dar. Der Befehlssatz geht deutlich über den anderer Basic-Computer hinaus, wie die folgende Aufstellung zeigt:

ABS
ASC
ATN
CALL
CHR$
CLEAR
COLOR
CONT
COS
DATA
DEF FN
DEL
DIM
DRAW
END
EXP
FOR . . .
TO . . .
STEP
FLASH
FRE
GET GOSUB
GOTO
GR
HCOLOR
HGR
HGR2
HIMEM:
HLIN
HOME
HPLOT
HTAB
IF . . .
GOTO
IF . . .
THEN
INPUT
INT
INVERSE
IN #
LEFT$
LEN
LET
LIST
LOAD
LOG
LOMEM:
MID$
NEW
NEXT
NORMAL
NOTRACE
ON . . .
GOSUB
ON . . .
GOTO
ONERR
GOTO
PDL
PEEK
PLOT
POKE
POP
POS
PRINT
READ
RECALL
REM
RESTORE
RESUME
RETURN
RIGHT$
ROT
RND
RUN
SAVE
SCRN
SGN
SHLOAD
SIN
SPC

SPEED	TAB	VAL
SQR	TAN	VLIN
STEP	TEXT	VTAB
STOP	TRACE	WAIT
STORE	USR	XDRAW
STR$		

Der höhere Komfort des Apple-II gegenüber Computern wie dem PET-2001 oder dem TRS-80 ist vor allem in seinen grafischen Möglichkeiten zu sehen, nicht zuletzt aber auch in dem schon erwähnten Monitorprogramm für die Programmierung in Maschinensprache. Die Applesoft-Rechengenauigkeit beträgt neun Mantissenstellen, die Rechengeschwindigkeit ist gegenüber dem PET-2001 oder dem AIM-65 um rund ein Drittel geringer. Dies liegt jedoch ausschließlich an der Auslegung des Applesoft-Interpreters. Der Ganzzahl-Interpreter, der fest in ROMs gespeichert ist, übertrifft dagegen den PET-2001 an Geschwindigkeit deutlich.

3.7 Weitere Basic-Mikrocomputer

Die bisher besprochenen Computer sind natürlich noch längst nicht alle, die heute auf dem Markt sind. Eine umfassende Darstellung wäre wegen des laufend größer werdenden Angebots ohnehin kaum möglich. Die Auswahl der bisher vorgestellten Systeme orientierte sich ausschließlich an deren Verbreitung und stellt keinerlei Qualitätskriterium dar. Natürlich hat eine große Verbreitung bestimmter Computer auch ihre Ursachen, zum Beispiel einen für breite Schichten vertretbaren Preis.

Um nicht den Eindruck zu erwecken, daß es sonst keine Basic-Mikrocomputer mehr gibt, folgt noch eine kurze Übersicht über die wichtigsten anderen Produkte, die allerdings weder vollständig sein kann noch dieses Ziel überhaupt anstrebt. Die Übersicht entspricht der Marktsituation von 1980. Es sei

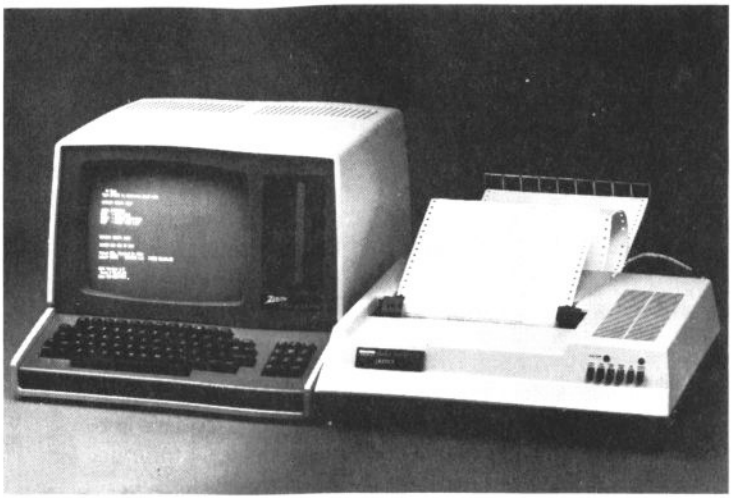


Abb. 3.7.1 Eine Stärke des WH-89 von Heathkit ist die Vielzahl an Programmiersprachen, für die Interpreter und Compiler auf Disketten zur Verfügung stehen

erwähnt, daß zu dieser Zeit noch kein einziger Personal Computer zur Verfügung stand, der von vornherein für die von vielen Anwendern als zukunftsträchtig bezeichnete, wenn auch schwieriger zu erlernden Programmiersprache Pascal gebaut wurde. Damit läßt sich wohl beweisen, daß Basic nach wie vor die Programmiersprache für kleinere Computer darstellt.

3.7.1 WH-89

Der Computer WH-89 *(Abb. 3.7.1)* wurde Ende 1979 von Heathkit auf den Markt gebracht. Er glieder sich intern in zwei Funktionsgruppen, von denen jede mit einem Z-80-Mikroprozessor ausgerüstet ist: den Terminal-Teil, der die gesamte Bildschirm-Organisation übernimmt, und den eigentlichen Computer, der die vom Benutzer eingegebenen Programme ausführt.

Alle Baugruppen des WH-89 sind in einem gemeinsamen Gehäuse untergebracht, und rechts neben dem Bildschirm findet ein Mini-Floppy-Disk-Laufwerk Platz, das im WH-89 eine zentrale Rolle spielt: Der Basic-Interpreter ist hier nämlich

nicht in ROMs fest gespeichert, sondern muß erst von der Floppy-Disk geladen werden. Das hat den Nachteil, daß mehr Adressenraum zur Verfügung steht, wenn man gerade nicht in Basic arbeitet, und den Nachteil, daß die Inbetriebnahme des Basic-Interpreters mehr Zeit und Bedienungsaufwand kostet als z.B. beim PET-2001.

Auf dem Bildschirm des WH-89 lassen sich 25 Zeilen mit je 80 Zeichen darstellen – dies ist mehr als auf den meisten anderen Personal Computern und gestattet zusammen mit dem eingebauten intelligenten Terminal eine sehr komfortable Textverarbeitung. Wer nicht mit einer Floppy-Disk, sondern lieber mit einem Kassettenrecorder arbeitet, dem steht ein Interface zur Verfügung, das nach der Kansas-City-Norm (1200/2400 Hz) mit 1200 bit/s arbeitet, das sind 120 Byte pro Sekunde. Ebenso ist eine Drucker-Schnittstelle vorhanden.

Im Gegensatz zu den anderen hier besprochenen Computern kann man den WH-89 selbst zusammenbauen. Dabei sind allerdings nur die vom Hersteller bereits getesteten Platinen und die einzelnen Baugruppen für Stromversorgung und Videoteil miteinander zu verdrahten und in das (übrigens sehr robuste) Gehäuse einzusetzen.

Speicheraufteilung beim WH-89

0-	System-Monitor-ROM
800-	Reserviertes ROM
1000-	Reserviertes RAM
1400-	Floppy-Disk-RAM
1800-	Floppy-Disk-ROM
2000-	Anwender-RAM
E000-	Reserviert
FFFF	

Befehlssatz des Microsoft-Basic für den WH-89

ABS	AND	ASC	ATN
AUTO	CINT	CSNG	CDBL
CHR$	CLEAR	CLOSE	CONT

COS	DATA	DEF	DEFDBL
DEFINT	DEFSNG	DEFSTR	DEFUSR
DELETE	DIM	EDIT	ELSE
END	ERASE	ERL	ERR
ERROR	FIELD	FIX	FN
FOR	FRE	GET	GOSUB
GOTO	HEX$	IF	IMP
INP	INPUT	INSTR	INT
KILL	LEFT$	LEN	LET
LINE	LIST	LOAD	LIST
LSET	MERGE	MID$	MOD
NAME	NEW	NEXT	NOT
NULL	OCT$	ON	OPEN
OR	OUT	PEEK	POKE
POS	PRINT	PUT	READ
REM	RESET	RESTORE	RESUME
RETURN	RIGHT$	RND	RSET
RUN	SAVE	SGN	SIN
SPACE$	SQR	STOP	STR$
STRING$	SWAP	SYSTEM	TAN
THEN	TO	TROFF	TRON
U3R	VAL	VARPTR	WAIT
WIDTH	XOR		

Der Benutzer kann zwischen mehreren unterschiedlichen Programmiersprachen wählen. Die am ehesten mit den übrigen Computern dieses Buches vergleichbare Sprache ist das Microsoft-Basic, dessen Komfort etwa zwischen dem des PET-2001 und des Apple-II oder ABC-80 eingestuft werden kann.

Es sei noch erwähnt, daß das Gerät auch als "Nur-Terminal" geliefert werden kann. Es enthält dann nur noch einen Z-80-Mikroprozessor, der dem Terminal eine gewisse Eigenintelligenz verleiht.

3.7.2 HP-85

Im Januar 1980 stellte Hewlett-Packard den Basic-Kompaktcomputer HP-85 vor, der in erster Linie auf den professionellen

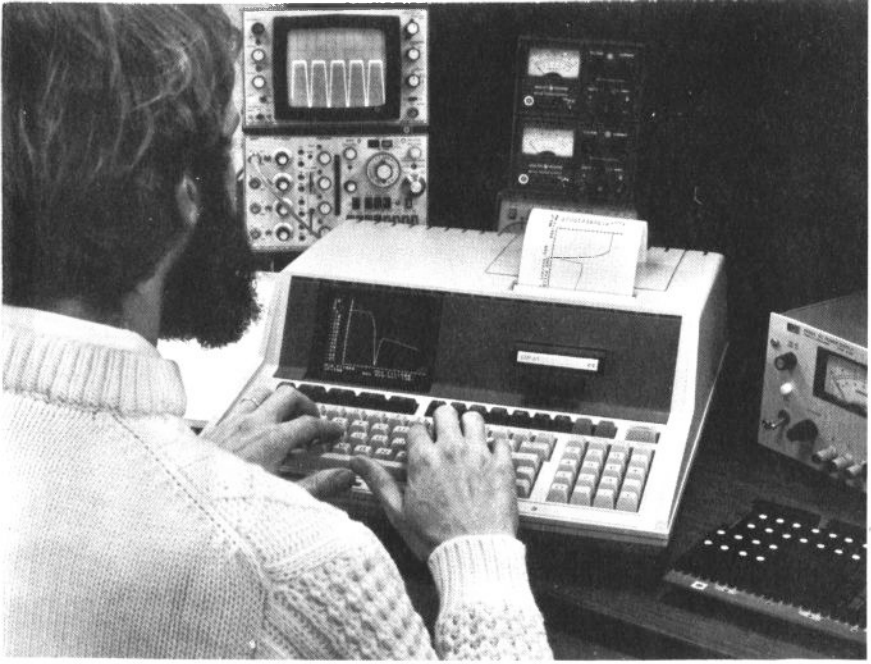


Abb. 3.7.2.1 Schon wegen seines Preises ist der HP-85 vor allem für industrielle Anwender geeignet. Wie PET-2001 und CBM-3001 verfügt er über einen IEC-Bus-Anschluß

Anwender zielt und auch ein Mehrfaches von dem kostet, was man für TRS-80, AIM-65 oder PET-2001 hinblättern muß *(Abb. 3.7.2.1)*.

Trotz des recht hohen Preises besitzt der HP-85 einen für viele Anwender bedeutsamen Nachteil: Er verfügt über keinerlei Möglichkeiten, Programme in Maschinensprache selbst zu schreiben, da er keine PEEK- und POKE-Befehle und auch kein Monitor-Programm besitzt. Beim HP-85 handelt es sich daher um einen "Basic-Rechner in Reinkultur".

Die CPU des HP-85 ist ein schon 1977 entstandener 8-bit-Prozessor mit gemultiplextem Adressen- und Datenbus, so daß der Systembus nur acht Leitungen umfaßt und der CPU-

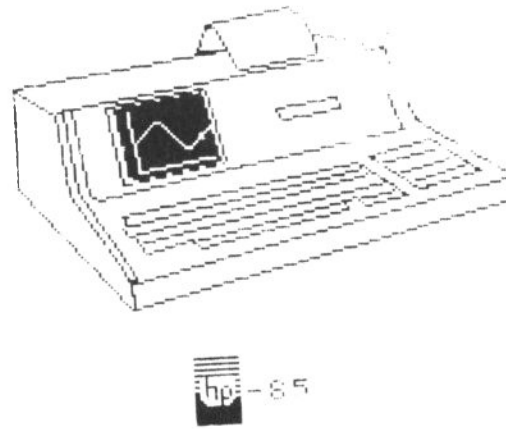


Abb. 3.7.2.2 Hier hat der HP-85 ein Selbstportrait erstellt: Kein Kunststück mit dem eingebauten Grafik-Interpreter

Chip mit 28 Pins auskomint. Der Rechner wird mit 16 KByte RAM geliefert und läßt sich bis zu 32 KByte RAM erweitern. Eine Besonderheit ist auch, daß der Basic-Interpreter nicht mit Binärzahlen, sondern dezimal rechnet, was sowohl der Rechengeschwindigkeit als auch der Genauigkeit zugute kommt. Der Befehlssatz entspricht dem der größeren HP-Rechner. Strings dürfen beliebig lang sein, sie sind nicht auf die sonst üblichen 256 Zeichen beschränkt.

Der HP-85 verfügt über einen kleinen Schwarzweiß-Bildschirm, auf dem sich 16 Zeilen mit je 32 Zeichen darstellen lassen. Der eingebaute Thermodrucker besitzt ebenfalls eine Schriftbereite von 32 Zeichen. Die Möglichkeiten zur Ausgabe von Kurven und Diagrammen (Plotten) sind sehr komfortabel; dabei ist es auch möglich, das Schriftbild auf dem Drucker um 90° zu drehen, um eine horizontale Beschriftung bei Endlos-Diagrammen zu erreichen *(Abb. 3.7.2.2)*.

Wie beim PET-2001 ist der Anschluß externer Geräte über den IEC-Bus-Anschluß möglich. Außerdem verfügt der HP-85

über eine serielle RS-232-Schnittstelle z.B. für Drucker oder ASCII-Fernschreiber.

3.7.3 TI-99/4

Der Mikrocomputer TI-99/4 ist der erste Heimcomputer mit 16-bit-CPU, nämlich mit dem Mikroprozessor TMS 9900. Er ist seit März 1980 auf dem Markt. Der Hersteller Texas Instruments brachte zunächst nur eine Version für das amerikanische Farbfernsehsystem NTSC heraus, so daß mangels eines eingebauten Bildschirms ein Farbfernsehgerät nötig ist, das mit 60 Hz Bildwechselfrequenz und der amerikanischen Farbcodierung arbeitet. Eine Notlösung stellt der Anschluß eines Schwarzweiß-Geräts mit Videoeingang dar, wobei der Bildfangregler von 50 Hz auf 60 Hz verstellt werden muß.

Der TI-99/4 besitzt ähnlich komfortable Farbgrafik-Möglichkeiten wie Apple-II und ITT-2020. Texas Instruments zielte mit ihm auf einen Markt, der erst seit Mitte 1980 eine Rolle spielt: Auf den Markt der "echten" Heimcomputer, deren Benutzer nicht mehr selbst programmieren, sondern im wesentlichen mit fertiger Standardsoftware arbeiten. Solche Standardsoftware ist für zahlreiche Anwendungsgebiete in Heim und Büro in Form von ROM-Moduln erhältlich, die 6...30 KByte enthalten können.

Ein 24 KByte (!) umfassender Basic-Interpreter, der trotz der 16-bit-CPU nicht allzu schnell ist, gibt dem Benutzer die Möglichkeit, Programme auch selbst zu schreiben und auf einen Kassettenrecorder abzuspeichern. Er enthält auch einen Formelrechner, der ebenso einfach zu bedienen ist wie ein Taschenrechner und Nicht-Programmierern die Möglichkeit gibt, Berechnungen durch bloßes Eingeben der Formeln durchzuführen. Erwähnenswert ist die farbige hochauflösende Grafik. Der Mikrocomputer enthält in der Grundversion 16 KByte RAM (trotz des 16-bit-Prozessors besteht auch hier ein Byte aus 8 Bits). Die Tastatur entspricht zwar der Anordnung auf einer Schreibmaschine; die mechanische Ausführung ist jedoch nicht allzu hohen Ansprüchen gewachsen und am besten noch

Abb. 3.7.3 Durch die Verwendung von ROM-Software-Moduln ist der Umgang mit dem TI-99/4 auch Nichtfachleuten möglich, die vom Programmieren nichts wissen wollen. Trotzdem ist der Basic-Befehlssatz recht leistungsfähig

Abb. 3.7.4 Der MZ-80 K von Sharp war praktisch der erste japanische Mikrocomputer auf dem europäischen Markt. Er arbeitet mit der CPU Z-80, und der Basic-Interpreter wird von der Kassette geladen

mit der des PET-2001 zu vergleichen. Der TI-99/4 läßt sich wie der HP-85 nicht in Maschinensprache programmieren.

3.7.4 MZ-80 K

Der Mikrocomputer MZ-80 K *(Abb. 3.7.4)* wird seit Anfang 1980 von Sharp vertrieben. Wie der PET-2001 ist er ein Kompaktgerät mit eingebautem Kassettenrecorder und grünleuchtendem Bildschirm. Die Tastatur besitzt ein getrenntes Nummernfeld. Auf dem Bildschirm lassen sich 24 Zeilen mit je 40 Zeichen darstellen; auch die Ausgabe von Grafiken ist möglich, dabei ergibt sich eine Auflösung von 4000 Punkten.

Der MZ-80 K besitzt keinen eingebauten Basic-Interpreter im ROM. Stattdessen muß der Interpreter erst von der mitgelieferten Kassette geladen werden – das dauert jeweils knapp zwei Minuten. Für den Ladevorgang und für das Starten von Maschinenprogrammen enthält der MZ-80 K ein einfaches ROM-Monitorprogramm. Die Grundversion enthält 20 KByte RAM.

Der Basic-Befehlssatz des MZ-80 K ist dem des PET-2001 recht ähnlich. Trotzdem besitzt dieser japanische Computer einige erwähnenswerte Nachteile.

Zunächst ist es einmal (ähnlich wie beim TRS-80) nicht möglich, irgendwelche externen Geräte damit zu bedienen, solange man nicht den herausgeführten Adressenbus decodiert und den Datenbus über Zwischenspeicher (Latches) puffert, um I/O-Ports zu erhalten. Hat man dies nun geschafft, so muß man feststellen, daß der MZ-80 K keine IN- und OUT-Befehle kennt, so daß man die spezielle I/O-Struktur des eingebauten Mikroprozessors Z-80 (Taktfrequenz 2 MHz) nicht nutzen kann. Stattdessen ist vom Basic-Interpreter aus das Ansprechen externer Geräte nur über "memory-mapped I/O" möglich.

Derartige Dinge stellen bereits ein Problem dar, wenn man nur versucht, zum Beispiel einen Morse-Decoder zu programmieren und nach einer Möglichkeit sucht, eine Morsetaste oder einen Kurzwellen-Empfangskonverter mit dem MZ-80 K zu verbinden. Im Grundzustand des MZ-80 K ist das nämlich nicht möglich!

Das Begleitmaterial, das dem MZ-80 K beigepackt ist, dokumentiert zwar recht gut den Basic-Interpreter mit seinen Befehlen; Sharp gibt jedoch keine Auskunft über verwertbare Monitor- oder Basic-Interpreter-Unterprogramme in Maschinensprache. Der Versuch, den Interpreter auf eigene Faust mit PRINT PEEK (...) zu untersuchen, wird abgeblockt; Basic schützt sich hier selbst vor fremdem Zugriff.

Speicheraufteilung beim MZ-80 K

0000...0FFF	Monitorprogramm-ROM
1000...11FF	Monitor-RAM (reserviert)
1200...5FFF	RAM (Grundversion)
6000...CFFF	RAM-Erweiterung
D000...DFFF	Video-RAM-Bereich
E000...FFFF	Reservierter Bereich

Basic-Befehlssatz des MZ-80 K

TAB	STEP	SIN	LOAD
SPC	NEXT	COS	SAVE
DIM	MUSIC	TAN	VERIFY
STOP	TEMPO	ATN	WOPEN
END	TI$	SQR	PRINT/T
REM	TM$	EXP	ROPEN
USR	PRINT	LOG	INPUT/T
LIMIT	PRINT/P	LN	CLOSE
RESTORE	INPUT	RND	? (=PRINT)
RETURN	GET	LEFT$	π (=3.14...)
DEF FN	PEEK	RIGHT$	NEW
READ	POKE	MID$	CLR
DATA	SET	LEN	RUN
GOTO	RESET	ASC	CONT
GOSUB	ABS	CHR$	BYE
ON	SGN	VAL	SIZE
IF...THEN	INT	STR$	LIST

Die Worte OR und AND wurden hier durch "+" bzw. "*" ersetzt. WOPEN öffnet eine Datei zum Schreiben, ROPEN zum

Lesen. MUSIC und TEMPO sind besondere Befehle für die Erzeugung von Musik. Vergleichsoperatoren usw. entsprechen dem Befehlssatz z.B. des PET-2001.

Monitor-Befehle des MZ-80 K

LOAD	Laden eines Programms in Maschinensprache vom Band
LOAD PGM	Laden eines Programms mit dem Namen PGM (als Beispiel)
GOTO$XXXX	Start eines Programms an der Hex-Adresse XXXX
SG	Einschalten der akustischen Tastendruck-Bestätigung
SS	Hebt SG wieder auf
FD	Aufruf eines Programms bei F000, falls der Inhalt von F000 gleich Null ist
SAVE	Abspeichern eines Programms mit Anfang- und Endadresse usw. in hex 1102...107

3.7.5 PC-1000

Der Personal Computer PC-1000 *(Abb. 3.7.5)* wurde 1980 von Siemens vorgestellt. Es handelt sich dabei um eine deutsche Eigenentwicklung, die jedoch deutlich an den Apple-II angelehnt ist. Der PC-1000 gestattet den Anschluß eines Farbfernsehgerätes als Monitor und besitzt ähnliche Farbgrafik-Möglichkeiten wie der Apple-II.

In dem etwa schreibmaschinengroßen Gehäuse ist auch ein Floppy-Disk-Laufwerk untergebracht; ferner ist ausreichend Platz für einen Drucker und den späteren Einbau von Magnetblasenspeichern enthalten. Der PC-1000 enthält 32 KByte RAM und 24 KByte ROM (inkl. Microsoft-Basic-Interpreter). Die Programmierung der CPU 6502 auf Maschinensprache-Ebene ist offiziell nicht möglich.

Der PC-1000 kann daher vorwiegend Leuten empfohlen werden, die ausschließlich in Basic (oder Pascal) programmieren, besitzt aber einen sehr leistungsfähigen Befehlssatz, der sogar den des ABC-80 übertrifft. Erweiterungen sind mit Steckmoduln möglich.

Abb. 3.7.5 Siemens brachte 1980 den Tischcomputer PC-1000 mit eingebautem Floppy-Disk-Laufwerk heraus

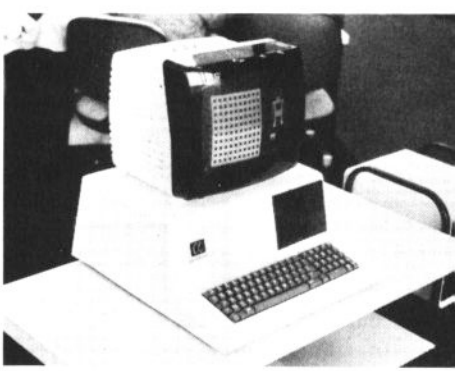


Abb. 3.7.6 Der Mikrocomputer Alphatronic von Triumph-Adler ist in unterschiedlichen Ausbaustufen zu haben – hier sogar mit einem Doppel-Floppy-Laufwerk

3.7.6 Alphatronic

Mit dem Computersystem "Alphatronic" *(Abb. 3.7.6)* trat Triumph-Adler in den Kompaktcomputer-Markt ein. Dabei

handelt es sich um ein modular aufgebautes System mit der von Intel entwickelten und in Deutschland auch von Siemens gefertigten CPU 8085, die softwaremäßig mit dem bekannten 8080 kompatibel ist. Das Alphatronic-System ist in unterschiedlichen Varianten lieferbar, z.B. mit dem Basic-Interpreter entweder im ROM oder auf Floppy-Disk (im letzteren Fall ist ein Floppy-Laufwerk bereits eingebaut). Für die Programmierung in Maschinensprache ist ein Monitorprogramm im ROM vorhanden, das auch dazu dient, den Basic-Interpreter laden und in Betrieb nehmen zu können.

Der Basic-Befehlssatz ist recht komfortabel und etwa mit dem des TRS-80 Level II vergleichbar. Die Alphatronic-Computer sind etwa schreibmaschinengroß und verfügen über einen Video-Ausgang mit dem Benutzer wählbaren Zeilenbreite, die sich an die Auflösung des verwendeten Monitors bzw. Fernsehgerätes anpassen läßt.

Der modulare Aufbau ist einer der größten Vorteile dieses Computersystems. Er ermöglicht eine flexible Anpassung an sich ändernde Bedürfnisse des Benutzers, ohne einen neuen Computer kaufen zu müssen.

3.8 Nachrüstung kleinerer Systeme mit Basic

Es ist eine ganze Reihe von Mikrocomputern auf dem Markt, die zunächst keinen Basic-Interpreter beinhalten. Dazu gehört im Prinzip auch der bereits besprochene AIM-65; allerdings erstreckt sich die notwendige Nachrüstung bei ihm auf das Einstecken von zwei PROM-ICs in schon vorhandene Fassungen.

Für andere Mikrocomputer, z.B. den Einplatinen-Computer KIM-1, ist ein Basic-Interpreter oft auf Kassette lieferbar. Dabei ist zu beachten, daß sich der Arbeitsspeicherbereich (RAM) nicht mehr ausschließlich für das vom Anwender eingetippte Basic-Programm verwenden läßt, da er ja auch den Interpreter selbst enthält. Für einen 8-KByte-Basic-Interpreter auf Kassette sind deshalb zumindest etwa 12 KByte RAM erforderlich.

Das für den KIM-1 lieferbare 8-KByte-Basic besitzt einen

dem AIM-65 sehr ähnlichen Befehlssatz. Wem jedoch 8 KByte allein für den Interpreter zuviel sind, kann auch das sogenannte *"Tiny Basic"* verwenden; es ist ebenfalls auf Kassette lieferbar, belegt aber nur 2 KByte. Allerdings müssen dann erhebliche Einschränkungen hingenommen werden: Es können nur ganze Zahlen mit den vier Grundrechnungsarten verwendet werden; FOR-NEXT-Schleifen sind nicht möglich und müssen durch Zählschleifen ersetzt werden; Strings sind ebenfalls nicht verwendbar. Andere Tiny-Basic-Versionen, z.B. für den Mikrocomputer Nascom-1, lassen zwar FOR-NEXT-Schleifen zu, reichen aber dennoch bei weitem nicht an den Komfort eines 8-KByte-Interpreters heran.

Trotz aller dieser Einschränkungen fanden Tiny-Basic-Interpreter eine große Verbreitung, weil sie auf relativ preiswerten Mikrocomputersystemen lauffähig sind. Der Preisvorteil von Tiny Basic ist jedoch seit dem Erscheinen der Mikrocomputer AIM-65, PC-100, TRS-80 und PET-2001 sehr zusammengeschrumpft.

Eine typische Tiny-Basic-Version ist das "KIM Tiny Basic". Es wird wegen seiner großen Verbreitung nachfolgend etwas näher beschrieben.

Tiny Basic existiert für den KIM-1 in zwei Versionen auf Kassette, nämlich für den Adressenbereich 2000...28E5 und für den Bereich ab 0200. Beide sind nun auch in Deutschland zu haben, z.B. von der Fa. Hofacker (Holzkirchen). Geht man von der erstgenannten Version aus, so läßt sich bei 2003 auch ein "Warmstart" durchführen, der den Programmspeicher nicht löscht. An der Adresse 200F kann das Terminal-ASCII-Zeichen für Back Space programmiert werden, üblicherweise hex 08. Aus einem laufenden Basic-Programm kann man z.B. durch Drücken der Space-Taste am Terminal „aussteigen", da Tiny Basic zyklisch den TTY-Eingang des KIM-1 abfragt.

An Systembefehlen stehen CLEAR (statt NEW), RUN, GOTO (wie RUN, jedoch Start an einer bestimmten Zeilennummer), und LIST in mehreren Varianten zur Verfügung. Basic-Befehle können, wenn man zu Beginn der Zeile eine Zeilennummer wegläßt, auch direkt ausgeführt werden.

Tiny Basic besitzt, wie erwähnt, einige Einschränkungen hinsichtlich des Befehlsvorrats. Folgende Befehle sind im Programm möglich: GOTO, GOSUB, RETURN, LET (kann entfallen), PRINT (Variable, arithm. Ausdruck oder Text), INPUT (Variable), IF...THEN (THEN kann entfallen) und END. Tiny Basic beherrscht die vier Grundrechenarten (+, −, *, /) und die Vergleiche =, > =, < =, >, <, < >, die zusammen mit IF verwendet werden. Die Rechenhierarchie ist „Punkt vor Strich". Die Kurzform von PRINT ist PR.

Gegenüber komfortablen Basic-Interpretern müssen folgende Einschränkungen hingenommen werden: Pro Zeile ist nur ein Befehl möglich; der Rechenbereich umfaßt nur ganze Zahlen von −32762 bis +32762; es gibt nur 26 Variablen, nämlich A...Z, und es sind keine String-Variablen möglich. Dies führt z.B. dazu, daß die INPUT-Funktion nicht mit YES oder NO beantwortet werden kann, sondern z.B. nur mit 1 oder 0 als Antwort auf eine Frage.

```
:LIST

10 PR"WIEVIELE SEK.";
20 INPUT A
30 B=30*A
40 C=0
50 C=C+1
60 IF C<B GOTO 50
70 PR A;" SEK. ";
80 IF A=1 GOTO 110
90 PR"SIND UM"
100 GOTO 10
110 PR"IST UM"
120 GOTO 10

:RUN
WIEVIELE SEK.? 2
2 SEK. SIND UM
WIEVIELE SEK.? 1
1 SEK. IST UM
```

Abb. 3.8 Ein kleines Demonstrationsprogramm in "Tiny Basic"; es dient zur Verzögerung um eine bestimmte Anzahl von Sekunden

Eine weitere Einschränkung ist, daß es keine FOR...NEXT-Schleifen gibt. Wie man dieses Problem umgeht, zeigt *Abb. 3.8*: Es stellt einen einfachen Timer dar, der eine programmierbare Verzögerung produziert. Interessant ist dabei auch die Möglichkeit, bei Print-Befehlen mit einem Strichpunkt am Ende der Programmzeile dafür zu sorgen, daß beim nächsten Print-Befehl oder auch bei INPUT in der gleichen Zeile weitergeschrieben wird. Ähnliches geschieht, wenn statt dem Strichpunkt ein Komma verwendet wird; dann können z.B. Zahlengruppen tabellarisch ausgedruckt werden.

Tiny Basic bietet auch die Möglichkeit, Unterprogramme in Maschinensprache aufzurufen. Dies geschieht mit dem Befehl USR, der jedoch nicht alleine stehen darf, sondern als Wertzuweisung dient, z.B. LETA = USR (256). Dieser Befehl weist der Variablen A einen Wert zu, der vom Unterprogramm, das an der Adresse 256 (dezimal) bzw. 100 (hex) steht, im Akku (LSB) und im Y-Register (MSB) in hexadezimaler Form übergeben wird. Damit sind unter anderem leistungsfähige Input-Output-Routinen realisierbar.

Verwendet man den USR-Befehl, so ist es praktisch, zu wissen, wo die Werte der Variablen gespeichert sind. Tom Pittman hat das recht einfach gelöst: Multipliziert man das ASCII-Äquivalent des Variablennamens mit 2, so ergibt sich die Adresse, wo der entsprechende hexadezimale Wert steht. So ist die Variable A (ASCII hex 41) an den Adressen 0082 (LSB) und 0083 (MSB) gespeichert. Das vom Benutzer eingegebene Basic-Programm beginnt bei der 2000-Version an der Adresse 2900. Jede Zeile beginnt mit der hexadezimal in zwei Bytes verschlüsselten Zeilennummer; es folgt der Zeilentext in ASCII, dann die nächste Zeilennummer usw. Die Zeilentabelle ist mit zwei Null-Bytes abgeschlossen.

Zum Schluß noch eine Bemerkung über den Programmbefehl END. Er ist nur dann erforderlich, wenn das Basic-Programm keine Endlosschleife darstellt. END sorgt nämlich dafür, daß nach Ausführung des Anwenderprogramms richtig in das Tiny Basic zurückgesprungen wird.

4 Begriffe, die Sie sich merken sollten

Das folgende Kapitel enthält eine Reihe von Fachausdrücken, die in Prospekten, Bedienungsanleitungen und Systemhandbüchern häufig vorkommen. Um das schnelle Finden eines Begriffes zu ermöglichen, wurde dabei eine alphabetische Ordnung gewählt. Mit Querverweisen ("s.d." = siehe dort), die nicht immer vermeidbar sind, läßt sich wiederum die Bedeutung der in den Erläuterungen verwendeten Fachbegriffe herausfinden, soweit sie nicht bereits in Kapitel 2 besprochen wurde.

Beim Leser setzen die Erläuterungen der einzelnen Begriffe nur elementare Kenntnisse der Mathematik und der Elektronik voraus. Wo auch in der deutschen Literatur aus dem Englischen stammende Ausdrücke üblich sind, wurde auf eine "gewaltsame" Eindeutschung verzichtet, um Verwirrungen zu vermeiden. Falls ein Wort einmal nicht in der alphabetischen Ordnung gefunden werden kann, so genügt meist ein Blick in das am Schluß des Buches befindliche Sachverzeichnis.

ASCII

ASCII ist die Kurzform von "American Standard Code for Information Interchange". Dabei handelt es sich um einen 7-bit-Code zur Zeichendarstellung. Wie man sich leicht ausrechnen kann, läßt ein 7-bit-Code $2^7 = 128$ unterschiedliche Bitkombinationen zu, die sich auch hexadezimal (s.d.) ausdrücken lassen: Der Bitkombination 0000000 entspricht die Hex-Zahl Null, und der Binärzahl 1111111 läßt sich hex 7F zuordnen. Jeder dieser 128 Bitkombinationen ist nun genau ein Zeichen zugeordnet, z.B. ein Buchstabe, ein Satzzeichen oder eine Ziffer. Die Zuordnung zwischen ASCII- und Hexadezimal-Äquivalent geht aus der *Tabelle* hervor.

Die ersten 32 Zeichen sind "nichtdruckend", d.h. sie dienen nur Steuerzwecken, z.B. um den Cursor (s.d.) über den Bildschirm eines Datensichtgerätes zu bewegen. Da man bei Basic-Computern mit dem Befehl PRINT CHR$(A) das der Dezimalzahl A entsprechende ASCII-Zeichen ausdrucken kann, gibt unsere Tabelle neben den Hex-Werten auch die Dezimal-Äqui-

Tabelle: Umrechnung dezimal/hexadezimal und ASCII-Zeichen

hex	dez.	ASCII	hex	dez.	ASCII
00	0	NUL	20	32	Space
01	1	SOH	21	33	!
02	2	STX	22	34	"
03	3	ETX	23	35	#
04	4	EOT	24	36	$
05	5	ENQ	25	37	%
06	6	ACK	26	38	&
07	7	BEL	27	39	'
08	8	BS	28	40	(
09	9	HT	29	41	)
0A	10	LF	2A	42	*
0B	11	VT	2B	43	+
0C	12	FF	2C	44	,
0D	13	CR	2D	45	–
0E	14	SO	2E	46	.
0F	15	SI	2F	47	/
10	16	DLE	30	48	0
11	17	DC1	31	49	1
12	18	DC2	32	50	2
13	19	DC3	33	51	3
14	20	DC4	34	52	4
15	21	NAK	35	53	5
16	22	SYN	36	54	6
17	23	ETB	37	55	7
18	24	CAN	38	56	8
19	25	EM	39	57	9
1A	26	SUB	3A	58	:
1B	27	ESC	3B	59	;
1C	28	FS	3C	60	<
1D	29	GS	3D	61	=
1E	30	RS	3E	62	>
1F	31	US	3F	63	?

▶

hex	dez.	ASCII	hex	dez.	ASCII
40	64	@	60	96	\
41	65	A	61	97	a
42	66	B	62	98	b
43	67	C	63	99	c
44	68	D	64	100	d
45	69	E	65	101	e
46	70	F	66	102	f
47	71	G	67	103	g
48	72	H	68	104	h
49	73	I	69	105	i
4A	74	J	6A	106	j
4B	75	K	6B	107	k
4C	76	L	6C	108	l
4D	77	M	6D	109	m
4E	78	N	6E	110	n
4F	79	O	6F	111	o
50	80	P	70	112	p
51	81	Q	71	113	q
52	82	R	72	114	r
53	83	S	73	115	s
54	84	T	74	116	t
55	85	U	75	117	u
56	86	V	76	118	v
57	87	W	77	119	w
58	88	X	78	120	x
59	89	Y	79	121	y
5A	90	Z	7A	122	z
5B	91	[	7B	123	{
5C	92	‘	7C	124	'
5D	93	]	7D	125	}
5E	94	↑	7E	126	~
5F	95	—	7F	127	DEL

Bedeutung der wichtigsten Steuerzeichen:
NUL = ohne Wirkung, SOH = start of header, STX = start of text, ETX = end of text, EOT = end of transmission, ENQ = enquiry (Aufforderung der Gegenstation zum Senden), ACK = acknowledge (Bestätigung, Rückmeldung), DLE = data link escape (Umschalten auf eine andere Steuerzeichengruppe), NAK = negative acknowledge, SYN = Synchronisationszeichen, ETB = end of transmission block, VT = vertical tabulating (Cursor nach oben), HT = horizontal tab. (Cursor nach rechts), BS = Back Space (Cursor nach links), LF = Line feed (Cursor nach unten), BEL = bell (akustisches Zeichen, Klingel), CR = Carriage Return, FS = field seperator, GS = group seperator, US = unit seperator, Space = Leerraum, DEL = Delete/Rub out.

valente an und eignet sich dadurch auch zum Umrechnen von Dezimal- in Hex-Zahlen und umgekehrt.

Nicht alle Computer lassen auch den Ausdruck oder die Anzeige von Kleinbuchstaben zu. Auch sind bei bestimmten Personal Computern den Zeichen andere Zahlen zugeordnet, oder die Kleinbuchstaben sind durch Grafiksymbole ersetzt. ASCII hat sich für die serielle Datenübertragung (s.d.) inzwischen allgemein eingebürgert. Hierzulande sagt man dazu auch ISO-7-bit-Code oder CCITT-Alphabet Nr. 5.

In vielen Fällen wird auch dann das ASCII-Format verwendet, wenn nur hexadezimale Daten bzw. Bitfolgen übertragen werden sollen. In diesem Fall faßt man die Daten in Gruppen von je vier Bits zusammen, betrachtet die vier Bits als Hexadezimalziffer (0...F) und macht daraus ein ASCII-Zeichen (hex 30...46) mit sieben Bits. Das achte Bit des seriellen ASCII-Formats kann dann zur Fehlererkennung als Parity-Bit (s.d.) übertragen werden.

ASCII-Arithmetik

Normalerweise ist die Rechengenauigkeit eines Basic-Interpreters auf 6...12 Stellen begrenzt. In bestimmten Fällen kann aber eine höhere Genauigkeit erwünscht sein, z.B. bei sehr hohen Finanzbeträgen, bei denen es auch auf den letzten Pfennig noch ankommt und wo wegen einiger Divisionen im Rechengang Rundungsfehler zu befürchten sind.

Der Computer ABC-80 enthält daher besondere Befehle, die es gestatten, die vier Grundrechenarten mit bis zu 29 Stellen Genauigkeit auszuführen. Die Zahlen werden dabei als Strings betrachtet und dürfen zwar Vorzeichen und Dezimalpunkt, nicht aber einen Exponenten enthalten. Die Befehle lauten ADD$, SUB$, MUL$ und DIV$. Um die beiden vielstelligen Zahlen A$

und B$ zu addieren, braucht man nur zu schreiben: PRINT ADD$ (A$, B$, L). Hierbei gibt L die gewünschte Stellenzahl an. Zur ASCII-Arithmetik gehört auch der Befehl COMP% (A$, B$), der den Wert Null liefert, wenn beide Zahlstrings gleich sind, −1, wenn A$ kleiner als B$ ist, und +1, wenn A$ größer als B$ ist.

Algol

Die "Alogrithmic Language", kurz Algol, wird zum Teil heute noch an Schulen und Hochschulen gelehrt, hat aber im praktischen Leben kaum noch Bedeutung, da sie sich für die modernen "Umgangsformen" bei Computern wie Dialogbetrieb usw. nicht eignet – wenn sie auch eine sehr leistungsfähige Sprache für den technisch-wissenschaftlichen Bereich darstellt. Die Lösung von Textverarbeitungs-Aufgaben ist bei Algol schwieriger zu realisieren als in der Kompaktcomputer-Standardsprache Basic.

Assembler

Ein Assembler dient zum Programmieren in Maschinensprache. Er ermöglicht es, nicht hexadezimale, schwer verständliche Operationscodes, sondern sogenannte mnemonische Befehle einzugeben. Ein Beispiel: Um die Konstante 15 in den Akkumulator des Mikroprozessors 6502 zu laden, müßte man hexadezimal A9 15 eingeben. Mit einem Assembler kann man dagegen eingeben: LDA #15, wobei LDA die leicht zu merkende Abkürzung von "Load Accu" darstellt. Solch ein Assembler ist z.B. im AIM-65 schon fest eingebaut. Er übersetzt die eingetippten Befehle sofort in die hexadezimalen Codes.

Um bei sehr umfangreichen Programmen komplizierte Adressenrechnungen zu vermeiden, gibt es sog. 2-Pass-Assembler. Sie gestatten die Verwendung symbolischer Adressen ("Labels"), die im ersten Assembler-Lauf (Pass 1) realen, hexadezimalen Adressen zugewiesen werden. Erst während Pass 2 werden die hexadezimalen Codes für das endgültige

Maschinenprogramm erzeugt. Voraussetzung für einen 2-Pass-Assembler ist das Vorhandensein eines Editors (s.d.), um das mnemonische Programm mit den symbolischen Adressen (Source-File, Quellenprogramm) überhaupt schreiben zu können. Das nach dem 2-Pass-Übersetzungsvorgang entstandene hexadezimale Programm nennt man "Objektcode". 2-Pass-Assembler sind für den AIM-65 und für die meisten anderen Computer in Form von PROMs oder auf Kassette bzw. Floppy-Disk zu haben.

Backspace

Hat man sich bei der Eingabe einer Zahl, eines Textes oder einer Programmzeile einmal vertippt, so kann man mit der Backspace-Taste den Cursor auf dem Bildschirm ein Zeichen rückversetzen und das zuletzt eingegebene Zeichen korrigieren. Die Backspace-Funktion läßt sich bei Terminals und Computern, die dafür keine eigene Taste besitzen, mit der Tastenkombination CTRL H ausführen, beim AIM-65 bzw. PC-100 mit der Delete-Taste (DEL). Die zuletzt genannten Computer löschen dabei alle Zeichen, über die man mit DEL hinweggeht. Backspace hat den ASCII-Hex-Code 08, DEL den Code 7F.

Baud

1 Baud ist die Einheit für die Schrittgeschwindigkeit bei der seriellen Übertragung von Daten über eine Zweidraht-Leitung, abegekürzt 1 Bd. Bei den üblichen "einwertigen" Übertragungsverfahren entspricht 1 Bd einer Geschwindigkeit von einem Bit pro Sekunde; andere Verfahren sind in der Lage, z.B. durch Tonfrequenzcodierung mehrere Bits gleichzeitig zu übertragen, so daß dann bei 100 Bd Schrittgeschwindigkeit die Datengeschwindigkeit 200 bit/s und mehr sein kann.

Für seriellen Datenverkehr im ASCII-Format sind die Schrittgeschwindigkeiten 110 Bd, 150 Bd, 300 Bd, 600 Bd, 1200 Bd, 2400 Bd, 4800 Bd, 9600 Bd und 19200 Bd genormt. Für die Ansteuerung von Druckern sind 110...300 Bd üblich, für den Datenverkehr mit Terminals sind 600 Bd recht verbreitet. Die Auf-

zeichnung von Daten und Programmen auf Tonbandkassetten geschieht mit Geschwindigkeiten zwischen 110 Bd und – nicht mehr ganz unkritisch – 2400 Bd.

Baudot-Code

Wie ASCII (s.d.) dient auch der Baudot-Code zur Darstellung von Buchstaben, Ziffern und Zeichen als con Computern verarbeitbare Bitfolge. Im Gegensatz zu dem 7-bit-ASCII-Format arbeitet der Baudot-Code mit nur fünf Bits. Da sich mit fünf Bits jedoch nur $2^5=32$ Zeichen darstellen lassen, was für den gesamten erforderlichen Zeichensatz nicht ausreichend wäre, führte man zwei Umschaltzeichen ein: "BU" bedeutet, daß alle folgenden Zeichen zur Buchstaben-"Ebene" gehörten, "ZI" definiert die folgenden Zeichen als zur Ziffern-Ebene gehörig. Das Wort "Ebene" stammt hier aus der Technik mechanischer Fernschreiber, bei denen die Papierwalze tatsächlich je nach Zeichenart höher oder tiefer liegt.

Der Baudot-Code ist heute praktisch nur noch bei Fernschreibern zu finden. Solche Geräte sind gebraucht oft sehr preiswert erhältlich (z.B. Siemens T-100, Lorenz LO-15 usw.) und stellen ein brauchbares Ausgabemedium für Mikrocomputer dar, wenn man den eingeschränkten Zeichensatz inkauf nimmt: Entweder ist nur Kleinschreibung oder nur Großschreibung der Buchstaben möglich, und nicht alle in ASCII üblichen Zeichen, z.B. $, #, <, >, sind im Baudot-Code definiert, so daß sie durch willkürlich gewählte andere Zeichen oder Zeichenkombinationen ersetzt werden müssen.

Die Umcodierung von Baudot auf ASCII und zurück überläßt man sinnvollerweise dem Mikrocomputer; der Software-Aufwand ist relativ gering. Dabei kann der Computer auch die erforderliche Parallel-Serien-Wandlung zur Ansteuerung des Fernschreibers durchführen. Bei Baudot-Fernschreibern sind die Geschwindigkeiten 45 Bd, 50 Bd und 75 Bd üblich; Postfernschreiber sind meist auf 50 Bd eingestellt.

Benchmark-Programm

Benchmark-Programme dienen zum Vergleich unterschiedlicher Mikroprozessoren und Programmiersprachen. Da aber jeder Mikroprozessor und jede Computersprache unter ganz bestimmten, anwendungsorientierten Gesichtspunkten entwickelt wurde, ist ein solcher Vergleich auch mit solchen objektivierten Programmen kaum wirklich objektiv möglich. Es kann sehr wohl sein, daß in bestimmten Anwendungsfällen ein Prozessor, der in Benchmark-Vergleichen miserabel abschneidet, jeden anderen Typ in den Schatten stellt.

Immerhin stellen Benchmark-Programme die einzige Möglichkeit dar, überhaupt solche Vergleiche durchzuführen. Ein typisches Benchmark-Programm ist zum Beispiel, den ersten freien Speicherplatz in einer Texttabelle zu suchen und von einem anderswo befindlichen Input Buffer (s.d.) die neu eingegebenen Zeichen dorthin zu transferieren. Dies ist ein tatsächlich häufig vorkommendes Problem. Wenn man dieses Programmierproblem in einige Mikroprozessortypen implementiert, so gelangt man, wie A. Osborne herausfand, zu folgendem Ergebnis:

CPU	*Befehle*	*Bytes*
8080/85	11	20
Z-80	6	12
6800	11	24
6502	9	16
2650	6	16
SCMP	14	23
9900	5	10

Die Tatsache, daß nicht alle heutigen Kompaktcomputer mit dem 9900 arbeiten, der hier am besten abschneidet (er ist ein 16-bit-Prozessor), ist darauf zurückzuführen, daß bei der CPU-Auswahl in Wirklichkeit ganz andere Dinge eine Rolle spielen als ein Benchmark-Programm, dessen Aufgabenstellung eine bestimmte Qualitätsreihenfolge vortäuscht. Bei unserem Beispiel kommt der Typ Z-80 ganz gut weg, weil der über einen "Block Move"-Befehl verfügt, den andere CPUs durch geeignete Befehle

umschreiben müssen. Trotzdem sind die Basic-Interpreter für den Z-80 nicht schneller als z.B. jene für den 6502.

Man sollte sich darüber im Klaren sein, daß für jeden Prozessor ein Benchmark-Programm gefunden werden kann, bei dem er am besten abschneidet.

Bildschirmtext

Unter Bildschirmtext (englische Bezeichnung: Viewdata) versteht man ein öffentliches Kommunikationsmedium, das über das Fernsprechnetz den Zugriff auf einen von der Deutschen Bundespost betriebenen zentralen Computer gestattet. Als Endgerät kann ein mit einem geeigneten Decoder ausgerüstetes Farbfernsehgerät fungieren, das über ein von der Post zur Verfügung gestelltes 1200-Bd-Modem mit dem Telefon verbunden wird (siehe "Serielle Datenübertragung").

Jeder Bildschirmtext-Teilnehmer besitzt eine eigene Kenn-Nummer, die er angeben muß, wenn er sich in das System einschaltet. Der Zugriff auf den Computer erfolgt seitenweise; d.h. es wird immer eine komplette "Bildschirmseite" vom zentralen Rechner an den Teilnehmer gesandt, und zwar mit 120 Zeichen pro Sekunde. Jede Seite hat eine bestimmte Nummer, die der Benutzer unter Zuhilfenahme eines mit der Tastenfolge *0# abrufbaren Gesamt-Inhaltsverzeichnisses feststellen kann.

Heimcomputer sind teilweise bereits dafür ausgerüstet, als Decoder für Bildschirmtext-Signale zu fungieren. Der Anwender benötigt dann lediglich noch ein geeignetes Betriebsprogramm und das Post-Modem, über das der Telefon-Anschluß erfolgt. Dabei können bei der hohen Geschwindigkeit von 1200 Bd ausschließlich galvanisch bekoppelte Modems verwendet werden; die akustische Ankopplung ist nur bis etwa 300 Bd möglich.

Es sei noch erwähnt, daß "Videotext", auch "Bildschirmzeitung" genannt, den gleichen Decoder benutzt. Hierbei werden ASCII-codierte Textseiten in der Vertikal-Austastlücke von Fernsehsendungen untergebracht und als farbige Schrift auf dem Farbfernsehgerät dargestellt. Die Übertragung geschieht hier also nicht über das Telefon, sondern per Funk über die

Fernsehantenne. Im Gegensatz zu "Bildschirmtext" ist eine Rückantwort des Teilnehmers nicht möglich.

Bit

Ein Bit ist die kleinstmögliche Informationseinheit. Es stellt genau eine Ja-Nein-Entscheidung dar, die sich physikalisch in Form zweier eindeutig definierter Spannungspegel oder logisch in Form zweier Zustände (H, L = High, Low) ausdrücken läßt. Mathematisch betrachtet, kann ein Bit die Zustände 0 und 1 annehmen, also genau zwei Werte. Im Gegensatz zum Dezimal-Zahlensystem, bei dem es zehn zulässige Werte pro Ziffer gibt, nämlich 0...9, handelt es sich hier um ein binäres Zahlensystem. Kombinationen von mehreren Bits bezeichnet man als Byte oder Wort (s.d.).

Entscheidung	binär	Logik-Pegel (pos.Logik)
ja	1	H
nein	0	L

Byte

Als Byte bezeichnet man allgemein eine Folge von acht zusammenhängenden Bits (s.d.). Ein Byte kann also die binären Werte 00000000...11111111 annehmen. Ordnet man den einzelnen Bits, von rechts nach links, die Wertigkeit 2^0 bis 2^7 zu, so kann man mit einem Byte eine Dezimalzahl von 0 bis 255 ausdrücken:

$$11111111 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 =$$
$$= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255.$$

Ein Byte mit der Bitkombination 01100001 hätte also den dezimalen Wert:

$$01100001 = 2^6 + 2^5 + 2^0 = 97.$$

Solche Umrechnungen spielen eine große Rolle, wenn man mit einem Basic-Computer z.B. einen Eingabe/Ausgabe-Port ansprechen möchte, wobei jedem Bit innerhalb des zum Ausgang

transferierten Byte eine Leitung zu einem oder von einem externen Gerät zugeordnet ist. Die Ein- bzw. Ausgangspegel werden als Kombination von acht Bits betrachtet und vom Basic-Computer dezimal verarbeitet.

Als Maß für die Speichergröße eines Computers oder eines externen Speichermediums existieren die Einheiten Kilobyte (1 KByte = 2^{10} Byte = 1024 Byte) und Megabyte (1 MByte = 10^6 Byte).

CRLF

CRLF ist eine vielgebrauchte Abkürzung für Carriage Return/ Line Feed. Dabei handelt es sich um die Aufeinanderfolge der beiden ASCII-Zeichen (s.d.) hex 0D und hex 0A. Ersteres ist das Return-Zeichen, das den Cursor auf dem Bildschirm, den Druckkopf eines Druckers bzw. die Walze des Fernschreibers bis zum linken Anschlag zurückbewegt, also zum Beginn der gerade geschriebenen Zeile. Mit "Line Feed" gelangt man schließlich eine Zeile tiefer, um dort weiterschreiben zu können. CRLF entspricht also in der Wirkung der Wagenrücklauf-Taste einer gewöhnlichen Schreibmaschine.

Da ein Druckkopf oder eine Papierwalze gewöhnlich etwas Zeit für den Rücklauf zum Zeilenbeginn benötigt, werden an die CR- und LF-Zeichen oft noch ein bis fünf ASCII-"NUL"-Zeichen (hex 00, ohne Wirkung) angefügt, bevor der erste Buchstabe der nächsten Zeile ausgegeben wird. Bei Datensichtgeräten ist dies normalerweise jedoch nicht erforderlich, ebenso nicht bei "intelligenten" Druckern.

Die Wirkung einer CRLF-Folge bei Datensichtgeräten, die im Scrolling-Modus arbeiten, ist etwas anders (s. "Scrolling").

CTRL (Control-Taste)

Der ASCII-Zeichenvorrat umfaßt 32 "nichtdruckende" Zeichen (hex 00...1F), die Steuerzwecken vorbehalten sind. Diese Zeichen lassen sich bei den meisten Datensichtgeräten erzeugen, indem man bei niedergedrückter CTRL-Taste bestimmte andere Tasten drückt. Dann wird beispielsweise beim Drücken

der Taste "H" nicht der dem Buchstaben H entsprechende Hex-Code 48 erzeugt, sondern das ASCII-Steuerzeichen BS = Backspace (hex 08). Die Wirkung bestimmter Steuerzeichen ist bei unterschiedlichen Terminals und Computern nicht einheitlich, lediglich die Bewegung des Cursors (s.d.) geschieht meist durch Betätigen der CTRL-Taste zusammen mit H, I, J, K. Ferner kann man mit CTRL C oft die Ausführung eines laufenden Basic-Programms unterbrechen.

Manche Personal Computer besitzen keine eigene CTRL-Taste, da für die notwendigen Steuerfunktionen, z.B. zur Cursor-Bewegung oder zum Anhalten des Programmes, spezielle Tasten zur Verfügung stehen. Bei Computern wie dem AIM-65, bei denen das Tastenfeld per Software decodiert wird, ist zwar eine CTRL-Taste vorhanden, sie liefert aber nicht immer die zu erwartenden Steuerzeichen.

Compiler

Aufgrund ihrer Befehlsstruktur und Wortlänge sind Mikroprozessoren nicht in der Lage, die Befehle einer höheren Programmiersprache wie Basic, Pascal, Fortran, Algol usw. direkt auszuführen. Vielmehr kann die CPU nur die Befehle der relativ primitiven Maschinensprache abarbeiten. Daß man mit Mikroprozessoren dennoch Computer mit höheren Sprachen bauen kann, liegt an der Verwendung von Interpretern (s.d.) und Compilern.

Ein Compiler ist ein in Maschinen- oder einer anderen Sprache geschriebenes Programm, das aus einem vom Benutzer des Computers in einer höheren Sprache eingegebenen Programm ein von der CPU verarbeitbares Maschinenprogramm macht. Diesen Vorgang nennt man auch "Übersetzen". Gewöhnlich ist es dabei notwendig, aus einem einzigen höheren Programmbefehl eine Vielzahl von Maschinenbefehlen zu machen.

Das auf diese Weise entstandene Maschinenprogramm ist nun auf dem Computer sofort lauffähig. Das für den Übersetzungsvorgang notwendige Compilerprogramm wird dann nicht mehr benötigt und kann, wenn es nicht in einem Festwertspeicher, sondern im RAM des Computers steht, ohne weiteres gelöscht werden, um Platz zu gewinnen.

Da Compiler ein direkt lauffähiges Maschinenprogramm erzeugen, sind die mit ihnen geschriebenen Programme wesentlich schneller als Programme, für deren Ausführung ein Interpreter benötigt wird. Auf der anderen Seite brauchen Compiler deutlich mehr Speicherplatz als Interpreter, weshalb sie sich nur bei größeren Computern durchsetzen konnten. Allerdings sind inzwischen auch für manche Personal Computer (z.B. Apple, WH-89) Pascal- und Basic-Compiler zu haben.

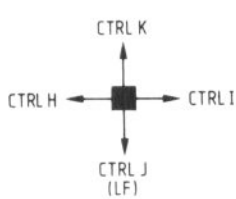
Cursor

Der Cursor markiert auf dem Bildschirm eines Computers oder Datensichtgerätes, an welcher Stelle das nächste Zeichen geschrieben wird. Dabei sind unterschiedliche Anzeigearten des Cursors üblich.

Eine Möglichkeit ist, daß jenes Zeichenquadrat, bei dem sich der Cursor befindet, als kleine weiße Fläche dargestellt wird, so daß es sich vom übrigen Bildschirmhintergrund abhebt. Befindet sich der Cursor an einer Stelle, wo bereits ein Zeichen steht, so wird dieses invertiert, d.h. es erscheint dunkel auf dem hellen Untergrund des Cursors.

In den meisten Fällen erscheint der so dargestellte Cursor nicht als konstant weiße Fläche, sondern blinkt mehrmals pro Sekunde. Dies hat den Vorteil, daß man ihn auch erkennen kann, wenn einige Zeichen auf dem Schirm ohnehin auf weißem Untergrund dargestellt werden.

Eine etwas seltenere Methode der Cursor-Darstellung ist, ihn durch ein dafür reserviertes ASCII-Zeichen (s. ASCII) zu kennzeichnen, z.B. einen nach oben oder nach rechts weisenden



Die aktuelle Schreibposition auf dem Bildschirm wird mit einem Cursor markiert, der sich entweder mit diesen CTRL-Tastenkombinationen oder auch mit speziellen Cursortasten bewegen läßt

Pfeil. Dieses Verfahren ist zwar hardwaremäßig einfacher realisierbar, besitzt aber den großen Nachteil, daß das an der gleichen Stelle wie der Cursor stehende Zeichen nicht lesbar ist.

Mit besonderen Tasten – meist durch Pfeile gekennzeichnet – läßt sich der Cursor und damit die Schreibposition auf dem Schirm in jede gewünschte Richtung bewegen. Bei manchen Terminals, z.B. beim CT-64, ist das Bewegen des Cursors nach oben nur in der Page-Betriebsart (s.d.) zulässig.

Terminals, die keine besonderen Tasten für das Bewegen des Cursors besitzen, lassen durch gleichzeitiges Drücken einer Buchstaben- und der Control-Taste alle Bewegungsrichtungen zu. Üblicherweise verwendet man dafür:

CTRL H	Cursor nach links
CTRL I	Cursor nach rechts
CTRL K	Cursor nach oben

Mit der Taste "Line Feed" (LF) wandert der Cursor schließlich um eine Zeile nach unten, ebenso bei Betätigung von CTRL J.

Disassembler

Als Disassembler bezeichnet man ein Übersetzungsprogramm, das ein hexadezimal im Speicher stehendes Maschinenprogramm in mnemonische, leichter verständliche Befehle übersetzt, z.B. A9 15 in LDA #15. Außerdem rechnet ein Disassembler meist relative Adressen in absolute um. Findet er beispielsweise einen relativen Sprungbefehl um vier Bytes nach vorn, so gibt er die sich ergebende absolute Sprungziel-Adresse als vierstellige Hexadezimalzahl aus.

Disassembler sind entweder Maschinen- oder Basic-Programme. Die Geräte AIM-65 und PC-100 sind derzeit die einzigen Basic-Computer, bei denen Disassembler und Assembler (s.d.) zur Standardausstattung gehören, d.h. sie sind in ROMs auf der Computerplatine untergebracht.

DMA

DMA bedeutet "direct memory access". Dabei handelt es sich um ein Verfahren, das es externen Geräten und Schaltungen ge-

stattet, ohne Umweg über die CPU Informationen in das System-RAM einzuschreiben oder aus ihm auszulesen. Dadurch ist eine sehr hohe Datentransfer-Geschwindigkeit möglich. Um eindeutige Pegelverhältnisse auf dem Adressen- und Datenbus des Computers zu gewährleisten, ist es notwendig, die Bus-Anschlüsse der CPU für die Dauer des externen Zugriffs auf das RAM in den hochohmigen Zustand zu versetzen. Hierfür besitzen viele Prozessoren geeignete Steuerpins. Eine typische Anwendung ist das Auslesen von Daten aus dem Video-RAM (s.d.).

Editor

Als Editor bezeichnet man ein Programm, mit dem es möglich ist, Text mit der Tastatur in den Computer einzuschreiben, irgendwie zu verarbeiten und später wieder auszugeben. Ein Editor sollte es zulassen, einzelne Zeilen zu löschen, neue Zeilen zwischen schon bestehende einzufügen oder an die bisherigen anzufügen und zu korrigieren. Nützlich sind auch Befehle zum alphabetischen Sortieren aller Zeilen, zum formatierten Ausdruck (z.B. mit frei wählbarer Druckbreite und/oder automatischem Randausgleich) sowie zum Abspeichern der Textzeilen auf ein externes Speichermedium wie Kassette oder Floppy-Disk.

Die Mikrocomputer AIM-65 und PC-100 enthalten einen Texteditor bereits als ein in Maschinensprache geschriebenes Programm fest in einem ROM gespeichert. Der Vorteil der Maschinensprache, nämlich die gegenüber Basic unvergleichlich höhere Arbeitsgeschwindigkeit, kommt besonders bei längeren Texten zum Tragen, speziell bei Such- und Sortiervorgängen.

EPROM

Erasable Programmable Read-Only Memory – das ist der Ausdruck für ICs, die eine Sonderform des PROM (s.d.) darstellen und gegenüber diesem den Vorteil besitzen, vom Anwender nicht nur selbst programmiert, sondern bei eventuellen Fehlern auch selbst wieder gelöscht werden zu können. Dies geschieht

mittels einer intensiven Ultraviolett-Bestrahlung durch das Quarzglasfenster dieser ICs hindurch, das sich über dem eigentlichen Halbleiterchip befindet.

Während früher meist mehrere schwierig bereitzustellende Versorgungsspannungen für ein EPROM erforderlich waren, benötigen die heute üblichen Typen 2758 (1 KByte) und 2716 (2 KByte) nur noch + 5 V. Die Zugriffszeit (Zeit zwischen Anlegen der Adresse und der Verfügbarkeit von Daten) liegt dabei bei 450 ns, was gerade zu dem "Timing" heutiger Prozessortypen paßt.

Zum Programmieren von EPROMs ist eine (nur dann benötigte) Hilfsspannung von + 26 V erforderlich. Ferner schreiben die Hersteller ganz bestimmte Zeiten für das Programmieren pro Bit vor. Manche Computerhersteller bieten für ihre Systeme spezielle EPROM-Programmiergeräte oder -platinen an, wobei die notwendigen Impulse und Adressen entweder vom angeschlossenen Computer per Software oder auch – unabhängig vom Computer – per Hardware erzeugt werden.

Maschinenprogramme, die zunächst im RAM des Computers entwickelt wurden und nun nach Fertigstellung und Prüfung in ein EPROM geladen werden sollen, müssen noch eine gewisse Korrektur erfahren, da sie dann ja in einem anderen Adressenbereich arbeiten. Um sie auch im endgültigen Zustand noch testen zu können, vertreibt die Industrie RAM-Box-Geräte, die über ein vielpoliges Kabel und einem dem EPROM-Sockel entsprechenden Stecker als EPROM-Ersatz dienen, bis der letzte Fehler gefunden ist.

Feld

Im Programmier-Sprachgebrauch bezeichnet man eine Gruppe von zusammengehörigen Zahlen oder Variablen als "Feld" (englisch Array). Ein solches Feld kann zum Beispiel eine Gruppe algebraischer Koeffizienten $a_0, a_1, a_2, a_3, a_4, a_5$ sein. In Basic gibt es keine tiefgestellten Ziffern und gewöhnlich keine Kleinbuchstaben, deswegen schreibt man hier A(0), A(1), A(2), A(3), A(4) und A(5). Dabei handelt es sich um ein ein-

dimensionales Feld, da die Variablen nur einen Index enthalten (hier 0...5).

Sechs Variable könnte man aber auch als zweidimensionales Feld definieren, zum Beispiel in der Form A(0,0), A(0,1), A(0,2), A(1,0), A(1,2). Der erste Index nahm hier die Werte 0...1, der zweite 0...2 an, so daß sich genau sechs Kombinationsmöglichkeiten ergeben. Man könnte auch schreiben A(I,J). Läßt man die Indices I und J jeweils die Werte 0...99 durchlaufen, so ergeben sich hier 10 000 Kombinationsmöglichkeiten, das Feld würde also 10 000 Elemente umfassen.

Darüberhinaus sind auch mehrdimensionale Felder denkbar. Albert Einstein hätte an den heutigen Basic-Computern seine wahre Freude gehabt, denn es sind hinter Variablennamen oft bis zu 255 Indices zulässig (!). Damit könnte man nicht nur vierdimensional, sondern zweihundertfünfundfünfzig-dimensional rechnen. In Basic ist es bei der Verwendung von Feld-Variablen wie A(I) notwendig, mit DIM am Anfang des Programms dem Rechner zu sagen, wieviel Speicher er für die Feldelemente reservieren soll.

Firmware

Als Firmware bezeichnet man jene Betriebsprogramme und Interpreterprogramme, die fest in einem ROM gespeichert sind, also nicht erst vom Band oder von der Floppy-Disk in den Comuter geladen werden müssen. Gewöhnlich gehören dazu die für die Tastaturabfrage und Displayansteuerung notwendigen Routinen sowie der Basic-Interpreter. Die Firmware stellt also den Grenzfall dar, daß Software sozusagen als Hardware vorliegt.

File (Datei)

Eine File ist eine beliebige Anzahl zusammengehöriger Variablen, gleich welchen Typs. Die Aufzeichnung einer solchen File z.B. auf eine Kassette oder Floppy-Disk geschieht normalerweise in einem Blockformat. Die einzelnen Blöcke enthalten nicht jeweils genau eine Variable, sondern vielmehr werden zunächst alle Variablen aneinandergekettet, dann in Blöcke aufgeteilt und beim Zurückladen wieder in die ursprüngliche Form zerlegt.

Für die Aufzeichnung einer File kann man ihr gewöhnlich auch einen aus einer begrenzten Anzahl von Buchstaben bestehenden Namen geben, der als "Header" vor die eigentliche File geschrieben wird. Das Aufzeichnungs- und Datenformat von Files ist praktisch bei jedem Computersystem anders, was den Datei-Austausch sehr erschwert. Beispiele für Files sind: Die Lottozahlen vom letzten Freitag; das Inhaltsverzeichnis dieses Buches; eine Kundenkartei. Auch ein auf Kassette gespeichertes Basic-Programm kann als File betrachtet werden, nämlich als Sammlung von Programmzeilen.

Fließkomma-Darstellung

Die meisten Basic-Computer besitzen einen Rechenbereich, der die darstellbare Stellenzahl übersteigt. Die Zahl 100 000 000 000 wird also nicht mit ihren elf Nullen, sondern als 1 E 11 ausgegeben. Die "Mantisse" ist hier die vor dem Exponenten stehende 1, und E 11 bedeutet soviel wie "mal zehn hoch elf". Mantisse und Exponent können auch negative Werte annehmen; −3.654 E-3 bedeutet zum Beispiel −0,003654.

Intern arbeitet der Computer stets im Fließkomma-Format, d.h. die Mantisse ist bei der Interndarstellung stets zwischen 1 und 9,999... und die tatsächliche Lage des Kommas bzw. des Dezimalpunktes wird durch den Exponenten angegeben.

Bei den meisten Computern werden Mantisse und Exponent auch nicht dezimal codiert abgespeichert (BCD, d.h. zwei Dezimalziffern in einem 8-bit-Wort), sondern rein binär. Dies spart zwar Speicherplatz und erhöht bei bestimmten Operationen die Rechengeschwindigkeit, erfordert aber andererseits Dezimal-Binär- und Binär-Dezimal-Umwandlungsprogramme im Interpreter-ROM. Die Umwandlung kostet ihrerseits wiederum eine gewisse Zeit und führt manchmal auch zu kleinen Rechenfehlern, z.B. wenn man erwartet, daß ein Ergebnis Null ist, der Computer aber 1 E-10 herausbekommt.

Im Gegensatz zum Fließkommaformat wird für die Darstellung von ganzen Zahlen (Ganzzahlvariable, z.B. A%) meist das Festkomma-Format verwendet, das im Speicher pro Zahl nur zwei Bytes belegt.

Fortran

Fortran ist eine weitverbreitete Sprache und die Abkürzung von Formula Translation. Dabei handelt es sich um eine problemorientierte Programmiersprache für den technisch-wissenschaftlichen Bereich. Für bestimmte Kompaktcomputer sind Fortran-Compiler lieferbar. Im Gegensatz zu Basic ist bei Fortran jedoch keine direkte Befehlsausführung (ohne Zeilennummer) bzw. ein echter Dialogbetrieb möglich. So sieht beispielsweise ein Fortran-Programm zur Ermittlung der kleinsten der drei Zahlen X, Y, Z aus:

```
100   IF (X-Y) 130, 110, 110
110   IF (Y-Z) 120, 140, 140
120   E = Y
      GOTO 160
130   IF (X-Z) 150, 140, 140
140   E = Z
      GOTO 160
150   E = X
160   RETURN
```

Der Compiler (s.d.) macht aus dieser Befehlsfolge ein Maschinenprogramm, das wesentlich schneller läuft als ein "interpretiertes" Basic-Programm.

Ganzzahl-Variable

Nicht immer ist es sinnvoll und notwendig, mit der vollen Genauigkeit von sechs und mehr Stellen plus dem Exponenten zu rechnen. Wenn man sich auf ganze Zahlen ohne Exponenten und eingeschränktem Wertebereich festlegt, können erhebliche Rechenzeiten eingespart werden, und auch der Speicherbedarf für eine solche Ganzzahl-Variable ist wesentlich geringer als der einer Fließkomma-Variable.

In Basic werden Ganzzahl-Ausdrücke durch ein Prozent-Zeichen gekennzeichnet, z.B. A%, C1%, SWAP% usw. Da sie meist als 15-bit-Zahl binär gespeichert werden – dazu kommt noch ein Bit für das Vorzeichen – ist der zulässige Zahlenbereich

−32768 bis +32767. Der Versuch, einer Ganzzahl-Variablen ein nicht ganzzahliges Rechenergebnis zuzuweisen, führt nicht immer zu einer Fehlermeldung (z.B. bei Tiny Basic).

Hardware

Die Hardware eines Computersystems ist – im Gegensatz zur Software (s.d.) – alles jene, was physikalisch greifbar vorhanden ist, auch wenn man das Computersystem ausschaltet. Dazu gehören die Speicher-ICs, der Mikroprozessor (CPU), die Ein/Ausgabe-Bausteine, die Tastatur, das Sichtgerät, aber auch die Stromversorgung. Programme, die fest in ROM-ICs gespeichert sind, bezeichnet man dagegen als Firmware.

Wenn man seine Programme nicht selbst schreibt, sondern in Software-Büros in Auftrag gibt, muß man damit rechnen, daß der Hardware-Preis eines Computersystems nur einen Bruchteil der tatsächlichen Systemkosten ausmacht.

Hexadezimale Zahlen

Mit vier Bits (s.d.) kann man eine dezimale Ziffer darstellen, wenn man jedem Bit eine Zweierpotenz-Wertigkeit zuordnet. Das von links erste Bit hat die Wertigkeit 2^3, die übrigen folgen mit 2^2, 2^1 und 2^0. Die vier Bits besitzen also die dezimalen Wertigkeiten 8, 4, 2 und 1.

So ist es möglich, z.B. die Dezimalzahl 7 durch die Bitkombination 0111 auszudrücken. Wenn man jedoch umgekehrt die Bitkombination 1011 mit einer Dezimalziffer ausdrücken will, stößt man auf Schwierigkeiten: Die Summe der Bitwertigkeiten ist hier dezimal 11. Da man sich aber bemüht, eine bestimmte Bitanzahl auch mit einer bestimmten, festen Anzahl von Ziffern auszudrücken, z.B. 16 Bits stets mit vier Ziffern, wird in der Computertechnik – speziell beim Umgang mit Maschinensprachen – das sog. Hexadezimalsystem verwendet, zu deutsch "Sechzehnersystem".

Während das Binärsystem nur zwei Ziffern (0 und 1) und das Dezimalsystem zehn Ziffern (0...9) zur Zahldarstellung benutzt, stehen im Hexadezimalsystem – manches Mal auch als

Sedezimal-System bezeichnet – sechzehn Ziffern zur Verfügung. Diese sind:

Hexadez.	Dezimal	Binär	Hexadez.	Dezimal	Binär
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Die Umrechnung von Dezimal- in Hexadezimalzahlen ist am einfachsten, wenn man sich an den Bitwertigkeiten orientiert. Will man z.B. wissen, wie die Dezimalzahl 107 hexadezimal geschrieben wird, so zerlegt man sie zunächst in ihre Zweierpotenzen:

$$107 = 64 + 32 + 8 + 2 + 1 =$$
$$= 0\cdot 2^7+1\cdot 2^6+1\cdot 2^5+0\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0$$

Schreibt man nun die Koeffizienten vor den Zweierpotenzen hintereinander, so ergibt sich die Binärzahl 01101011. Teilt man diese in Vierergruppen auf, so ergibt sich aus obiger Tabelle die Hex-Zahl 6B.

Umgekehrt läßt sich eine hexadezimale Zahl in ihr dezimales Äquivalent umrechnen, wenn man sie als Bitfolge darstellt und dann dezimal die Bitwertigkeiten addiert. Ein Beispiel: Gesucht ist das dezimale Äquivalent von hex FC.

FC läßt sich als achtstellige Bitkombination schreiben, indem man jeder Hex-Ziffer vier Bits zuordnet. Nach unserer Tabelle ergibt sich die Bitfolge 11111100. Schreibt man die Bitwertigkeiten hintereinander, so sieht das folgendermaßen aus:

$$11111100 = 2^7+2^6+2^5+2^4+2^3+2^2 =$$
$$128+64+32+16+8+4 = 252$$

Ein Byte (s.d.) wird gewöhnlich mit zwei solchen Hex-Ziffern

dargestellt. Zur Kennzeichnung, ob es sich bei einer Zahl um hexadezimale oder dezimale Ziffern handelt, stellt man in Assemblerprogrammen (s.d.) den hexadezimalen Zahlen entweder ein Dollar-Zeichen ("$") vor (6502-Assembler) oder ein "H" nach (8080/Z-80-Assembler). Die notwendigen Umrechnungen besorgt der Computer dann meist selbst. In Basic ist die Eingabe hexadezimaler Zahlen (außer in speziellen, dafür vom Anwender geschriebenen Programmen) gewöhnlich nicht zulässig, da 0...9 als Ziffern, alle anderen Zeichen aber als nichtnumerische Codes betrachtet werden.

Für den oben gezeigten Umwandlungsalgorithmus ist es – wenn man "zu Fuß" rechnet – wichtig, die Zweierpotenzen zu kennen. Hier also eine kleine Tabelle.

$2^0 = 1$	$2^{10} = 1024$
$2^1 = 2$	$2^{11} = 2048$
$2^2 = 4$	$2^{12} = 4096$
$2^3 = 8$	$2^{13} = 8192$
$2^4 = 16$	$2^{14} = 16384$
$2^5 = 32$	$2^{15} = 32768$
$2^6 = 64$	$2^{16} = 65536$
$2^7 = 128$	$2^{17} = 131072$
$2^8 = 256$	$2^{18} = 262144$
$2^9 = 512$	$2^{19} = 544288$

Hierarchie

Wie in der Algebra sind manche Operationen gegenüber anderen in Basic "bevorrechtigt". Die bekannteste Stufung dieser Art ist wohl die "Punkt-vor-Strich"-Regel: Die Aufgabe 1+2·3 liefert als Ergebnis nicht 6, sondern 7, weil die Multiplikation bevorrechtigt ausgeführt wird. Operationen mit gleicher Einstufung werden in der Reihenfolge ausgeführt, in der sie im jeweiligen Ausdruck geschrieben sind (z.B. Addition und Subtraktion). In Basic gibt es folgende "Ebenen" von Operationen:

1. Klammerausdrücke werden immer zuerst berechnet;
2. Potenzierung (z.B. 2↑5);

3. Negation, d.h. Vorzeichen vor einer Zahl oder einem Ausdruck;
4. Multiplikation und Division;
5. Addition und Subtraktion;
6. Vergleichsoperatoren: =, <, >, <=, >=;
7. Logisches NOT;
8. Logisches AND;
9. Logisches OR.

Da Operationen der gleichen "Ebene" in der Reihenfolge errechnet werden, wie sie im Programm stehen, kann man mit dem Ausdruck PRINT A-A/100 *100 bei Ganzzahl-Interpretern (z.B. "Tiny Basic") die Variable A auf ihre beiden niederwertigsten Stellen begrenzen; die höherwertigen werden abgeschnitten. (Bei Fließkomma-Interpretern funktioniert dieses Beispiel natürlich nicht.)

I/O-Port

Ein Port ist eine Parallelschnittstelle, über die der Computer Daten empfangen und senden oder auch externe Geräte steuern und abfragen kann. Im Prinzip handelt es sich dabei um Latch-Flipflop-Gruppen, die auf der einen Seite mit dem Datenbus des Mikroprozessors und auf der anderen Seite mit den Portanschlüssen verbunden sind. Durch ein Steuersignal kann der Mikroprozessor die Flipflops veranlassen, die gerade auf dem Datenbus befindlichen Bits zu übernehmen und an den Port weiterzugeben, der dann als Ausgang dient. Die jetzt am Ausgang stehenden Daten bleiben so lange erhalten, bis der Prozessor gezielt andere in die Latch-Flipflops einschreibt (s.a. VIA).

Die Besonderheit des Ports ist jedoch, daß er sich so umschalten läßt, daß er als Eingang dient. Dann werden bei einem bestimmten Steuersignal der CPU die von einem externen Gerät gelieferten Daten auf den Datenbus übernommen und zum Beispiel in den Akkumulator des Mikroprozessors transferiert, so daß sie irgendwie weiterverarbeitet werden können.

Die Prozessoren Z-80 und 8080/8085 verfügen – wie zahlreiche andere CPU-Typen – über die Befehle IN und OUT, die

die entsprechenden Steuersignale für die Ein- und Ausgabe produzieren. Bei den Prozessoren 6800 und 6502 ist dagegen das "Memory Map"-Verfahren üblich: Die Ports werden wie gewöhnliche Speicherzellen behandelt, so daß sich mit ihnen auch direkt arithmetische Operationen ausführen lassen (Beispiel: Daten vom Port zum Akkuinhalt addieren). Die Steuersignale für die Latch-Flipflops werden hier einfach durch Decodieren bestimmter Adressen gewonnen. Besondere Ein- und Ausgabebefehle kennen diese CPUs nicht. Bleibt noch zu erwähnen, daß ein Port stets genauso "breit" wie der Datenbus bzw. die "Wortlänge" des Prozessors ist. Bei 8-bit-Prozessoren verwendet man also Ports, die acht Leitungen als Verbindung zur Außenwelt besitzen. Einem Port sind, wenn die Eingangs/Ausgangs-Umschaltung per Software möglich ist, zwei Adressen zugeordnet, nämlich ein Port-Datenregister, in das Informationen von der CPU eingeschrieben werden und an externe Geräte ausgegeben werden können, und ein Port-Richtungsregister (Direction Register), dessen acht Bits angeben, ob die entsprechenden acht Port-Leitungen Eingänge oder Ausgänge sind.

Erst die I/O-Ports machen es möglich, daß der Computer mit seiner Umwelt verkehrt. Während das bei den Systemen PET-2001 und CBM-3032 kein Problem darstellt, weil sie über einen dem Benutzer zur Verfügung stehenden "User Port" verfügen, ist ein solcher Port in manchen anderen Computern leider nicht eingebaut (TRS-80, ABC-80) und muß bei Bedarf – und wer hat den Bedarf nicht – extern nachgerüstet werden.

Ports ermöglichen auch, obwohl sie ja im Prinzip eine parallele Datenschnittstelle darstellen, die serielle Ein- und Ausgabe (s. serielle Datenübertragung). Dabei wird einfach eine der acht Port-Leitungen, z.B. das niederwertigste Bit, als serieller Ein- oder Ausgang deklariert, indem das entsprechende Bit des Datenrichtungsregisters auf Null oder Eins gesetzt wird. Das "Timing", das Erkennen von Start- und Stopbits und die Seriell/Parallel- bzw. Parallel/Seriell-Umwandlung kann dann der Mikroprozessor selbst übernehmen. Dadurch entfällt das sonst für diesen Zweck benötigte UART (s.d.), und auch eine

automatische Anpassung der Übertragungsgeschwindigkeit an das externe Gerät wird möglich.

IC

In diesem Buch ist an verschiedenen Stellen von ICs die Rede, von kleinen elektronischen Bauelementen mit unterschiedlichsten Funktionen. IC ist die Abkürzung von "Integrated Circuit", Integrierte Schaltung also.

ICs sind kleine Siliziumplättchen von wenigen Millimetern Kantenlänge, die zum Schutz, zur Wärmeableitung und zur besseren Handhabung in ein "unnötig" großes Kunststoff-, Keramik- oder Metallgehäuse eingebaut sind, das bis 64 Anschlüsse (Pins) enthält. Auf dem Siliziumplättchen befinden sich elektronische Bauelemente kleinster Abmessungen, vorzugsweise Transistoren, Dioden und Widerstände (Kondensatoren sind wegen ihres Flächenbedarfs schwieriger zu integrieren). Der 16-bit-Mikroprozessor MC 68000 enthält zum Beispiel über 60000 Transistoren.

Je nach der verwendeten Halbleiter-Technologie spricht man von TTL-, CMOS-, NMOS-, PMOS-, ECL-, IIL-, DTL- oder MNOS-ICs. Für die Herstellung von Mikroprozessoren und Speichern haben in letzter Zeit neben der NMOS- und CMOS-Technik auch HMOS und VMOS Bedeutung erlangt.

IEC-Bus (auch IEEE-488- oder HP-Bus)

Computer, die für ein Zusammenwirken mit externen Meßgeräten u.a. vorgesehen sind, verfügen über eine dafür genormte Parallelschnittstelle, nämlich den IEC-Bus, der 1975 eingeführt und im wesentlichen von Hewlett-Packard entwickelt wurde. Auch die Computer PET-2001 bzw. CBM-3032 benützen diesen Bus.

Die physikalische Steckerausführung des IEC-Bus ist nicht einheitlich, so daß oft Adapter notwendig werden. Das System arbeitet mit TTL-Pegeln und negativer Logik, d.h. 0 V bedeutet "1" und 5 V bedeutet "0". Bis zu 15 Geräte können gleichzeitig, also parallel an den Bus angeschlossen sein, wobei jedes eine

bestimmte Adresse besitzt, die an ihm selbst eingestellt sein muß. Die Geräte können als "Talker" (Datensender) oder "Listener" (Datenempfänger) betrieben werden. Der Bus enthält dazu geeignete Steuerleitungen.

Die Computer PET-2001 und CBM-3032 sowie – mit einem Zusatz versehen – AIM-65 und PC-100 können beliebige Peripherie über den IEC-Bus bedienen. Die beiden ersteren besitzen bereits geeignete Basic-Befehle dafür: Mit OPEN und CLOSE kann man den IEC-Bus "öffnen" und "schließen", mit PRINT# bestimmte Geräte ansprechen und mit INPUT# Daten abfragen.

PET-Kontakt	Busart	IEEE-Bezeichnung	Signal
1	Daten	DI 01	Daten-Ein/Ausgang 1
2	Daten	DI 02	Daten-Ein/Ausgang 2
3	Daten	DI 03	Daten-Ein/Ausgang 3
4	Daten	DI 04	Daten-Ein/Ausgang 4
5	Manager	EOI	End or identify
6	Transfer	DAV	Data valid
7	Transfer	NRFD	Not ready for data
8	Transfer	NDAC	No data accepted
9	Manager	IFC	Interface clear
10	Manager	SRQ	Service request
11	Manager	ATN	Attention
12	Manager	Shield	Masse, Abschirmung
A	Daten	DI 05	Daten-Ein/Ausgang 5
B	Daten	DI 06	Daten-Ein/Ausgang 6
C	Daten	DI 07	Daten-Ein/Ausgang 7
D	Daten	DI 08	Daten-Ein/Ausgang 8
E	Manager	REN	Remote enable (PET: Masse)
F-N	Masse	GND 6-12	Masse für DAV, NRFD, NDAC, IFC, SRQ, ATN und Daten

Die einzelnen Signale des IEC-Bus sind:

ATN
Der Controller (z.B. PET) legt diese Leitung auf Low-Pegel, während er Befehle über den Datenbus sendet. Wenn ATN auf High-Pegel liegt, können vorher angewählte Geräte Daten übertragen.

DAV
Solange DAV auf Low-Pegel liegt, sind die Daten auf dem Datenbus gültig und dürfen ausgewertet werden.

EOI
Wenn das letzte Datenbyte übertragen ist, kann der "Talker", der Datensender, die EOI-Leitung auf Low legen.

IFC
Über die IFC-Leitung initialisiert der Controler (z.B. PET) alle an den IEC-Bus angeschlossenen Geräte, indem er sie bei einem Reset der CPU etwa 100 ms auf Low-Pegel legt.

NDAC
Der Datenempfänger (z.B. ein Drucker) legt diese Leitung auf Low-Pegel, während er Daten empfängt. Wenn das Datenbyte gelesen ist, legt er die Leitung wieder auf High-Pegel und signalisiert damit dem Controller, daß die Daten angenommen wurden.

NRFD
Diese Leitung zeigt durch Low-Zustand an, daß ein oder mehrere periphere Geräte nicht bereit sind, das nächste Datenbyte zu empfangen (Wired-OR-Verknüpfung).

SRQ
Über die Service-Request-Leitung kann ein peripheres Gerät dem Controller signalisieren, daß es Daten empfangen oder senden möchte. Dabei geht SRQ auf Low, bis ein Datenaustausch zustande kommt.

REN
Die Remote-Enable-Leitung wird vom jeweils aktiven Controller auf Low-Pegel gehalten. Der PET-2001 muß der einzige Controller des IEC-Bussystems sein, da er REN konstant auf Masse legt.

DI 01...08
Bei diesen Leitungen handelt es sich um eine Art 8-bit-Parallelschnittstelle für bidirektionalen (beidseitigen) Datenaustausch. Die übrigen Leitungen (s.o.) legen die Datenrichtung fest.

Input Buffer (Eingangspuffer)

Wenn man vom Computer-Tastenfeld aus Befehle, Text oder Zahlen eingibt – sei es im Input-Modus oder während der Ausführung eines INPUT-Befehls – werden die Zeichen nicht sofort in den Programmspeicher oder in Variablen übernommen, sondern werden zunächst in einem besonderen Speicherbereich abgelegt, dem Input Buffer. Erst beim Drücken der Return-Taste wird die Dateneingabe abgeschlossen und der Inhalt des Eingangspuffers in den endgültigen Speicherbereich übernommen. Auf diese Weise sind immer noch Korrekturen der Eingabe möglich, solange nicht Return gedrückt wurde.

Bestimmte Computer, z.B. der PET-2001, verwenden als Eingangspuffer einen Speicherbereich, der gleichzeitig als Video-RAM für die Darstellung von Zeichen auf dem Bildschirm verwendet wird. Der Vorteil dieses Verfahrens ist die Möglichkeit, auf dem Bildschirm durch geeignete Cursor-Bewegungen "herumzuredigieren", z.B. innerhalb eines Programmlistings. Dabei ist allerdings große Vorsicht geboten: Was tatsächlich vom Computer dann in den Eingangspuffer übernommen wird, ist nicht immer klar vorher ersichtlich, da der "Input Buffer" meist länger als eine Bildschirmzeile ist. Beträgt beispielsweise die Anzahl der darstellbaren Zeichen auf dem Schirm 40, die Pufferlänge aber 80, so können beim Drucken der Return-Taste u.U. zwei ganze übereinanderstehende Zeilen übernommen werden.

Ein besonderes Problem ist in diesem Fall auch das Korrigieren längerer Eingaben während eines INPUT-Befehls. Er erzeugt am Anfang einer Zeile ein Fragezeichen und einen Leerraum. Diese beiden Zeichen werden unter bestimmten Bedingungen in den Input Buffer mit übernommen, was vom Computerhersteller keineswegs beabsichtigt war, so daß der INPUT-Befehl hier einen String liefert, der nicht mit dem eingegebenen identisch ist. Bei Computern, die einen vom Video-RAM getrennten Eingangspuffer besitzen, ist zwar das Redigieren auf dem Bildschirm nicht möglich, aber es kann auch nicht zu dem geschilderten Problem kommen.

Der Mikrocomputer PET-2001 besitzt einen Buffer, in dem dank einer Interrupt-Routine auch während eines laufenden

Basic-Programms bis zu zehn Zeichen vorausgeschrieben werden können, noch bevor sie z.B. von einem INPUT-Befehl weiterverarbeitet werden.

Interface

Ein Interface benötigt man in einem Computersystem überall dort, wo eine Datenweiterleitung aus Gründen der unterschiedlichen elektrischen Pegel oder des nicht passenden Datenformats nicht möglich ist. Im ersten Fall genügt eine Pegelanpassung z.B. mit einem Transistor oder einem geeigneten IC, im letzten Fall wird u.U. sogar ein eigenes kleines Mikrocomputersystem als Interface dienen, beispielsweise zur Umwandlung des im Computer verwendeten Zeichencodes (ASCII) auf den Code eines externen Druckers (Baudot bei einem Fernschreiber oder EBCDIC bei einer Kugelkopf-Schreibmaschine).

Interface-Anordnungen, die "nur" das Datenformat anpassen sollen, können oft eingespart werden, wenn der Mikroprozessor im Computersystem schnell genug ist, die Umwandlung selbst per Software durchzuführen. Ein gutes Beispiel ist die Erzeugung von Morsezeichen: Ein langsamer Computer kann an seinem Ausgang nur die Impulse für Striche und Punkte erzeugen, so daß ein externes Interface daraus hörbare Töne machen muß. Ist dagegen die CPU schnell genug, an einem Port (s.d.) die Töne direkt mit Hilfe eines geeigneten Programms zu erzeugen, so kann man das Interface einsparen. Ähnliche Überlegungen gelten für die oben erwähnte Codeumwandlung, für die Programmaufzeichnung mit Kassettenrecordern oder für die Parallel-Serien-Wandlung für die Druckeransteuerung.

Dies ist auch der Grund, warum die Arbeitsgeschwindigkeit eines Mikroprozessors eine weit größere Rolle spielt, als man zunächst annehmen könnte: Eine schnelle CPU hilft, Hardware einzusparen.

Interpreter

Damit ein Mikroprozessor die relativ komplexen Befehle einer höheren Programmiersprache wie Basic ausführen kann, ob-

wohl er nur Maschinensprache "versteht", verwendet man Compiler (s.d.) oder Interpreter.

Ein Interpreter ist selbst ein in Maschinensprache geschriebenes Programm, das – im Gegensatz zum Compiler – auch während der Ausführung des Anwenderprogramms im Computerspeicher vorhanden sein muß. Jeder einzelne Befehl des in einer höheren Sprache geschriebenen Anwenderprogramms wird als Folge zahlreicher Maschinenbefehle "interpretiert" und von der CPU abgearbeitet. Diese Maschinenbefehle werden, wiederum im Gegensatz zum Compiler, nicht erzeugt und abgespeichert, sondern stehen innerhalb des Interpreterprogramms. Das Interpretieren eines Programmes ist zwar verhältnismäßig langsam, hat aber den großen Vorteil, daß außer dem (meist im Festwertspeicher/ROM stehenden) Interpreter und dem Anwenderprogramm mit seinen Variablen kein zusätzlicher Speicherplatz benötigt wird. Aus diesem Grunde verfügen die meisten Basic-Computer heute über Interpreter, während sich Compiler nur in Sonderfällen durchsetzen können.

Basic-Interpreter belegen je nach Befehlsvorrat, CPU und Rechengenauigkeit einen Speicherplatz von 2 KByte ("Tiny Basic", keine Strings, nur ganze Zahlen) bis mehr als 8 KByte (z.B. AIM-65, TRS-80, PET-2001, CBM-3032). Da der Festwertspeicher außer dem Basic-Interpreter meist auch noch Routinen für die Tastaturabfrage, die Display-Ansteuerung und ähnliche Aufgaben enthält, verfügen die meisten Basic-Computer über 12 KByte und mehr ROM-Speicherplatz. Übrigens stammen alle Basic-Interpreter der gerade genannten Systeme von ein und derselben Firma, nämlich Microsoft (USA).

Ein Interpreter kann auch dazu dienen, Maschinenprogramme auszuführen, die für einen anderen CPU-Typ geschrieben sind. So erschien z.B. in Heft 25/1979 der Zeitschrift "Funkschau" ein Interpreter-Programm, das es gestattet, auf einem 6502-Computer (KIM-1, AIM-65 usw.) 8080-Maschinenprogramme auszuführen. Hierbei wird der Prozessor 8080 "simuliert".

Interrupt

Die Programmausführung durch den Mikroprozessor eines Computersystems geschieht normalerweise Befehl für Befehl, d.h. die Adresse des nächsten Befehls geht immer eindeutig aus dem vorhergehenden hervor. Dieser geradlinige Ablauf kann jedoch sozusagen gewaltsam per Hardware unterbrochen werden, nämlich mit einem Interrupt-Signal an einem bestimmten Anschluß der CPU. Ein kurzer Spannungsimpuls genügt, und das Programm wird nicht beim nächsten Befehl fortgesetzt, sondern an einer Adresse, die der Benutzer als Interrupt-Vektor frei definieren kann. Manchmal steht der Interrupt-Vektor auch im ROM (Festwertspeicher); er wurde dann vom Computerhersteller fest auf eine bestimmte Adresse programmiert und kann nicht verändert werden. Für den Interrupt gibt es unterschiedliche Möglichkeiten und "Ebenen":

Reset ist der Interrupt mit der höchsten Priorität. Unabhängig vom Zustand der CPU führt er an eine bestimmte, durch einen meist im ROM stehenden Vektor spezifizierte und somit nicht veränderliche Adresse. An dieser Adresse steht dann ein Programm, das den Computer initialisiert, u.U. auch den Speicher löscht und die Befehlseingabe gestattet. Der Computer enthält daher eine Logik, die beim Einschalten sofort einen Reset-Impuls an die CPU liefert.

NMI (non-maskable interrupt, nicht maskierbarer Interrupt) ist eine Unterbrechung zweiter Priorität; sie ist nur wirksam, wenn nicht gleichzeitig ein Reset auftritt. Der im PET-2001 und CBM-3032 vorhandene Prozessor 6502 gestattet zwar einen solchen Interrupt, er wird jedoch hardwaremäßig hier nicht ausgenutzt.

IRQ (interrupt request) ist eine Unterbrechung mit der niedrigsten Priorität: Sie kann nämlich vom Prozessor durch geeignete Befehle (z.B. SEI, set interrupt disable flag) verhindert werden. Im Gegensatz zum Reset, aber ebenso wie beim NMI, kann man dabei vom Interruptprogramm in das Hauptprogramm zurückspringen, als ob nichts geschehen wäre; beim 6502 lautet der Befehl hierzu RTI (return from interrupt).

Gewöhnlich werden Interrupts von peripheren Geräten ausgelöst, die damit ihre Bereitschaft zum Datenaustausch indizieren und die CPU veranlassen, das für diesen Austausch notwendige Programm anzuspringen. Sobald alle Daten übertragen sind, erfolgt ein Rücksprung mit RTI oder einem entsprechenden Befehl.

Der Interrupt kann auch innerhalb des Computersystems selbst ausgelöst werden, zum Beispiel von einem Timer (s.d.), um eine Software-Uhr zu realisieren.

In den Computern PET-2001/CBM-3032 liefert ein solcher Timer – er ist in das IC 6522 integriert – 60mal pro Sekunde einen Interrupt-Impuls (IRQ) an die CPU 6502. Diese führt das Programm daraufhin an einer speziellen Routine fort, die die interne Uhr um 1/60 s weiterstellt und auch kontrolliert, ob die Break-Taste gedrückt ist. Danach springt die CPU in das vorher verlassene Hauptprogramm, hier in den Basic-Interpreter, zurück. Der IRQ-Vektor kann vom Benutzer geändert werden; er wird beim Einschalten auf seinen Normalwert gesetzt. So läßt sich auch erklären, warum sich der Computer manchmal "aufhängt": Bei Programmierfehlern, speziell in Maschinensprache, läuft das Programm wild im Speicher umher und ändert dabei u.U. den IRQ-Vektor. Der Timer-Interrupt führt dann zu einem Sprung an eine Adresse, an der u.U. kein vernünftiges Programm steht; dann kann der Computer nur noch mit einem Reset bzw. durch Aus- und Einschalten aus dem "Weltraum" zurückgeholt werden. Wenn man auf die Software-Uhr verzichten kann, ist es daher praktisch, während des Experimentierens mit Maschinensprache-Programmen den Interrupt gleich zu Anfang des Programms mit dem Befehl SEI (hexadezimal 78) zu verhindern.

Kansas-City-Standard

Leider gibt es heute von Hersteller zu Hersteller sehr unterschiedliche und in keiner Weise kompatible Methoden, Programme und Daten auf Kassette aufzuzeichnen. Immerhin setzten sich auf Einladung der amerikanischen Zeitschrit Byte Anfang der siebziger Jahre einige Leute in der Stadt Kansas/USA

zusammen, um eine einheitliche Norm dafür aufzustellen, die tatsächlich – wenn auch in Abwandlungen – weite Verbreitung fand.

Der Standard legt fest, daß die Aufzeichnung zunächst mit einem "Vorspann" beginnt, der fünf Sekunden lang einen 2400-Hz-Ton aufzeichnet. Die dann folgenden Daten werden in Form von Nullen (je vier Schwingungen 1200 Hz) und Einsen (je acht Schwingungen 2400 Hz) bitseriell auf Kassette gespeichert. Aus diesem "Timing" ergibt sich automatisch eine Geschwindigkeit von 300 Bd bzw. bit/s. Abarten des Standards arbeiten allerdings mit höheren Geschwindigkeiten bei gleichen Tonfrequenzen; so ist es z.B. möglich, 1200 Bd zu verwenden, wenn eine Null als eine einzige 1200-Hz-Periode aufgezeichnet wird und eine Eins als zwei 2400-Hz-Perioden. Es wurden sogar schon Versuche mit 2400 Bd unternommen, was allerdings schon recht hohe Ansprüche an die Gerätequalität stellt.

In welcher Form nun die Nullen und Einsen aneinandergekettet werden, d.h. wie das Datenformat aussieht, ist vom Kansas-City-Standard nicht verbindlich festgelegt, so daß die Bezeichnung "Kansas-City-Interface" keineswegs die Garantie für die Kompatibilität von Kassetten darstellt, die mit zwei unterschiedlichen Computern aufgenommen wurden. (S.a. serielle Datenübertragung.)

MOS (Metal Oxide Semiconductor)

Die MOS-Technologie spielt bei der Herstellung hochintegrierter ICs, wie Mikroprozessoren, Speicher oder Interface-Bausteine eine dominierende Rolle. Je mehr Bauelemente auf einem Silizium-Chip, d.h. innerhalb eines IC, untergebracht werden, desto mehr muß man nämlich darauf achten, daß diese Bauelemente möglichst wenig Wärme verursachen, d.h. wenig Strom verbrauchen, weil die entstehende Verlustwärme sonst nicht mehr abgeleitet werden kann. Aus diesem Grund war man erst etwa 1970 in der Lage, hochintegrierte Bausteine herzustellen: Die vorher ausschließlich verwendete TTL-Technologie (s.d.) arbeitete mit bipolaren Transistoren als aktive Ele-

mente, die relativ viel Strom verbrauchen, wenn man kleine Schaltzeiten erzielen möchte.

Der Durchbruch zu höheren Integrationsgraden gelang erst, als man MOS-Feldeffekt-Transistoren integrieren konnte. Dabei ergaben sich gleich zwei Vorteile: Die Integrationsfläche pro Transistor wurde kleiner, weil Feldeffekttransistoren (FETs) eine einfachere Struktur als bipolare Transistoren besitzen, und gleichzeitig ging der Leistungsverbrauch – pro Transistor gerechnet – um Zehnerpotenzen zurück, weil FETs rein spannungsgesteuert arbeiten und keinen Steuerstrom benötigen. Der einzige Nachteil der MOS-Technik ist nicht prinzipiell, sondern mehr oder weniger nur durch die technologische Auslegung der ICs bedingt: Die Arbeitsgeschwindigkeit ist geringer als bei TTL-ICs. Die erzielbaren Taktfrequenzen liegen meist weit unter 10 MHz.

Je nachdem, ob man p-leitendes oder n-leitendes Silizium als Grundmaterial für MOS-ICs verwendet, spricht man von der etwas langsameren PMOS- oder der heute vorwiegend verwendeten NMOS-Technologie. Beiden ist gemeinsam, daß die Steuerelektrode der FETs, das Gate, ein dünner leitender Film ist, der vom halbleitenden "Kanal" durch eine hauchdünne Oxidschicht getrennt ist.

Neuere MOS-Abarten sind HMOS und VMOS; sie bieten ein noch günstigeres Verhältnis zwischen Arbeitsgeschwindigkeit und Leistungsaufnahme und werden daher vorwiegend in höchstintegrierten Bausteinen eingesetzt.

Magnetblasenspeicher

Magnetblasenspeicher spielen eine zunehmende Rolle in der Mikrocomputertechnik. Diese "Bubble Memories" nützen den Effekt aus, daß man in bestimmten Materialien sog. magnetische Domänen durch ein von außen angelegtes Feld rotieren lassen kann. Dabei können Informationen bitweise gespeichert werden, indem man jedem Bit eine winzige Feldkonzentration von 1..3 μm Durchmesser zuordnet. Die erzielbare Speicherdichte ist sehr hoch; 1979 waren bereits 256-Kbit-Speicher mit dieser Technologie aus der Serienproduktion verfügbar,

und heute läßt sich sogar 1 MBit in einem einzigen Bubble-Chip unterbringen.

Leider ist der Anschluß eines Magnetblasenspeichers an einen Mikrocomputer nicht so einfach wie das Einstecken eines RAM-Chips in eine Fassung. Der Bubble-Chip erfordert eine umfangreiche Steuerelektronik, die die Speicherorganisation, die Schreib-Lese-Operationen und das Rotieren des Magnetfeldes übernimmt.

Die Zugriffszeit bei Magnetblasenspeichern liegt bei einigen Millisekunden, so daß sie Floppy-Disks überlegen sind. Gegenüber diesen besitzen sie den Nachteil, nicht austauschbar zu sein, und den Vorteil einer extrem hohen Datensicherheit. Bubble-Speicher werden daher in den nächsten Jahren vor allem die Rolle von nichtflüchtigen Massen-Hintergrundspeichern übernehmen. "Nichtflüchtig" bedeutet hier, daß die Informationen auch beim Abschalten der Versorgungsspannung erhalten bleiben.

Für die Mikrocomputer AIM-65 und PC-100 existieren bereits Platinen mit 1 MByte Magnetblasenspeicherkapazität. Dabei dient ein 6502-Mikroprozessor zur Steuerung und Datenformatierung. Der Speicher selbst besteht aus vier 256-K-Bubble-Chips.

Mnemonischer Befehl

Wenn man sich ein in Maschinensprache geschriebenes Programm im Speicher eines Mikrocomputers ansieht, so besteht es nur aus binären bzw. hexadezimalen Zahlen, die irgendeine Bedeutung haben. Wenn man bei der CPU 6502 etwa auf (hex) A9 stößt, so handelt es sich dabei um eine Anweisung, die diesem Byte folgenden Daten in den Akkumulator des Prozessors zu übernehmen, auf neuhochdeutsch "Load Accu". Ein Disassembler (s.d.) ist in der Lage, dieses Byte A9 in eine Abkürzung der Befehlsbedeutung zu übersetzen, nämlich in LDA. Man muß zugeben, daß man mit LDA schon etwas mehr anfangen kann als mit A9; die abgekürzte Befehlsform kann man sich jedenfalls besser merken als eine zweistellige hexadezimale Zahl.

Die abgekürzten Befehle wie LDA, STA usw. nennt man "Mnemonics" – zugegeben, die alten Griechen konnten das "Mn" sicher besser aussprechen als wir.

Die mnemonische Befehlsform wird auch bei der Eingabe eines Programmes mittels eines Assemblers (s.d.) verwendet. Leider sind nicht nur die hexadezimalen, sondern auch die mnemonischen Befehle bei den heutigen Mikroprozessoren völlig unterschiedlich, so daß das Umschreiben von Maschinenprogrammen für andere CPU-Typen äußerst schwierig ist – ein Grund mehr für die Verwendung höherer Programmiersprachen wie Basic.

Modem

Ein Modem ist eine Einrichtung, die die Übertragung von Daten oder Programmen über eine gewöhnliche Niederfrequenzstrecke gestattet. Dabei kann es sich um eine Telefon- oder auch eine Funkverbindung handeln.

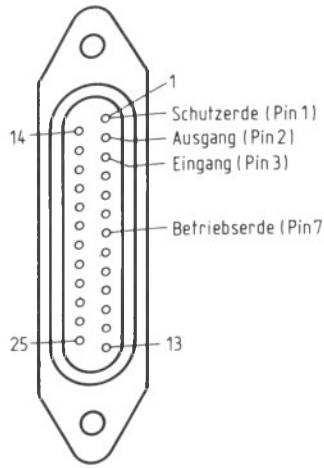
"Modem" ist die Kurzform von Modulator/Demodulator. Der Modulator ordnet den logischen Werten 1 und 0 bestimmte Tonfrequenzen zu, die empfangsseitig vom Demodulator wieder in ein Digitalsignal rückübersetzt werden.

Modem-Frequenzen

Norm	Anwendung	"1"	"0"	Geschwindigkeit
V-21	Telefon	1180 Hz	980 Hz	max. 300 bit/s (Anrufer)
	(Duplex)	1850 Hz	1650 Hz	max. 300 bit/s (Angerufener)
V-23a	Telefon	1700 Hz	1300 Hz	max. 600 bit/s (Hauptkanal)
	(Duplex)	450 Hz	390 Hz	max. 75 bit/s (Hilfskanal)
V-23b	Telefon	2100 Hz	1300 Hz	max. 1200 bit/s (Hauptkan.)
	(Duplex)	450 Hz	390 Hz	max. 75 bit/s (Hilfskanal)
IARU	UKW-Funkfernschreiben (Simplex)	2125 Hz	1275 Hz	45,5/50/75 Baud (Baudot) 300 Baud (ASCII)
Kansas-City (s.d.)	Kassetten	2400 Hz	1200 Hz	300 Baud (ev. auch 600/1200 Baud)


Schnittstellenleitung
Übergabestelle
Stift-Nr.
Datenendgerät (z.B. Terminal)
E1 – Schutzerde
E2 – Betriebserde
D1 – Sendedaten
D2 – Empfangsdaten
S1.1 Übertragungsleitung anschalten
S1.2 DE-Einrichtung betriebsbereit
S2 – Sendeteil einschalten
S3 – Alle Frequenzgruppen verwenden
S4 – Hohe Übertragungsgeschwindigkeit einschalten
S5 – Hohe Sendefrequenzlage einschalten
S6 – Niedrige Empfangsfrequenzlage einschalten
S7 – Empfangsdaten abrufen
S8 – Ersatzbetrieb einschalten
S9 – Bestätigungston senden
S10 – Datenbetrieb ablösen
S11 – Empfangsteil einschalten
M1 – Betriebsbereitschaft
M2 – Sendebereitschaft
M3 – Ankommender Ruf
M4 – Hohe Übertragungsgeschwindigkeit
M5 – Empfangssignalpegel
M6 – Empfangsgüte
M7 – Empfangsdaten-Kennzeichnung
M8 – Ersatzbetrieb
Hilfskanal
HD1 – Sendedaten
HD2 – Empfangsdaten
HS2 – Sendeteil einschalten
HM2 – Sendebereitschaft
HM5 – Empfangssignalpegel
HM6 – Empfangsgüte
T1 – Sendeschrittakt zur DÜ-Einrichtung
T2 – Sendeschrittakt von der DÜ-Einrichtung
T3 – Empfangsschrittakt zur DÜ-Einrichtung
T4 – Empfangsschrittakt von der DÜ-Einrichtung
T5 – Empfangsseitige Abtastmarkierung
Zusatzeinrichtung (Fehlerschutzeinheit, Synchronisiereinheit)
Leitungen für Übergabe von Datensignalen
Datenübertragungs-Einrichtung
A1 – Gesendete Sprachantwort
A2 – Empfangene Sprachantwort
E21 – Schutzerde
E22 – Betriebserde
W21 – Wählbit 1
W22 – Wählbit 2
W23 – Wählbit 3
W24 – Wählbit 4
S21 – Übertragungsleitung belegen
S22 – Wählzeichen übernehmen
M21 – Übertragungsleitung belegt
M22 – Wählzeichen-Übernahmebereitschaft
M23 – Wahl erfolglos
M24 – Gerufene Station angeschaltet
M25 – Funktionsbereitschaft
Leitungen für autom. Wahl
Wähleinrichtung

Die meisten Modems verwenden die RS-232-Norm für die Schnittstelle zum Computer oder zum Terminal. Manchmal ist auch eine automatische Wahl möglich



Die Bezeichnung "Duplex" bedeutet, daß durch die Verwendung zweier Frequenzpaare die Möglichkeit besteht, gleichzeitig (!) in beiden Richtungen Daten zu übertragen. Bei Simplex ist das wegen der Verwendung nur eines einzigen Frequenzpaares nicht möglich.

Bei Geschwindigkeiten von mehr als 1200 bit/s ist es kaum noch möglich, mit Tonfrequenz-Codierung zu arbeiten. Hier verwendet man andere Methoden, z.B. "Phase Encoding" (Phasencodierung) oder NRZ ("No Return To Zero", hier werden die zu sendenden Daten und eine der Bitrate entsprechende Frequenz auf ein Exklusiv-Oder-Gatter geleitet).

Modems enthalten gewöhnlich nicht die zur Serien-Parallel-Wandlung erforderliche Logik (UART). Die Verbindung mit dem Computer geschieht also mit einer seriellen Schnittstelle (siehe "Serielle Datenübertragung"). Die Verwendung eines Telefonmodems ist in Deutschland genehmigungspflichtig und erfordert eine Prüfung durch das Fernmeldetechnische Zentralamt.

Monitorprogramm

Basic-Computer sind in erster Linie für das Arbeiten in Basic konstruiert, und die Eingabe von Programmen in Maschinensprache mittels des POKE-Befehls wäre sehr umständlich und zeitraubend. Besser ist für diesen Zweck die Verwendung eines Monitorprogrammes. ("Monitor" hat hier nichts mit "Bildschirm" zu tun, sondern soll nur ausdrücken, daß man sozusagen in den Computerspeicher hineinsehen kann.)

Während Computer wie der AIM-65 bzw. PC-100 über ein bereits eingebautes, sehr komfortables Monitorprogramm verfügen, ist es bei anderen Geräten erforderlich, ein geeignetes Programm von der Kassette oder Floppy-Disk einzulesen.

Die Hauptaufgaben des Monitorprogrammes sind es, dem Benutzer den hexadezimalen Inhalt beliebiger Speicherzellen des Computers zu zeigen und es zu ermöglichen, diesen Inhalt mit neuen Daten oder Befehlen in Maschinensprache zu überschreiben. Ferner sollte es möglich sein, das fertige Maschinenprogramm irgendwie auf ein externes Speichermedium auszulagern und später wieder zu laden, da mit den Basic-Befehlen SAVE und LOAD dies nicht möglich ist.

PROM

Während ROM-ICs (s.d.) vom Hersteller schon während der Fertigung durch die Maskenauslegung programmiert werden, kann dies der Anwender bei PROMs selbst tun – wenn auch nur einmal, so daß man höllisch aufpassen muß, hierbei keine Fehler zu machen.

Gewöhnlich bestehen PROMs aus einer Diodenmatrix nebst der erforderlichen Adressendecodierlogik. Jede Diode repräsentiert dabei ein Bit. Durch Anlegen einer genügend hohen Spannung kann man die (bei Lieferung durch den Hersteller noch intakten) Dioden zerstören und so gezielt einzelne Bits setzen.

PROMs sind leider etwas teurer als ROMs, wenn man von sehr hohen Stückzahlen ausgeht. Bei mittleren Stückzahlen liegen aber die werksseitigen Programmierkosten von ROMs, bedingt durch die dafür notwendige Maskenänderung, unverhältnismäßig hoch, so daß man dann PROMs verwendet, die

man selbst programmiert. Ist dagegen zu erwarten, daß ab und zu eine Änderung der gespeicherten Programme und Daten erforderlich sein kann, so wird man EPROMs (s.d.) bevorzugen, die sich mehrmals programmieren lassen.

Page-Modus

Als "Page-Modus" bezeichnet man eine Betriebsart von Datensichtgeräten, bei der der Bildschirm Zeichen für Zeichen, von oben links beginnend, vollgeschrieben wird. Wenn das letzte Zeichen (rechts unten) geschrieben ist, beginnt der Schreibvorgang wieder links oben, wobei bei manchen Terminals auch gleichzeitig die gesamte "Bildschirmseite" gelöscht wird. Meist ist auch eine Umschaltmöglichkeit in den oft bevorzugten, praktischen "Scrolling-Modus" (s.d.) vorgesehen. Die heutigen Personal Computer besitzen eine automatische Page/Scroll-Umschaltung: Wenn die unterste Zeile des Bildschirms erreicht wird, schaltet der Computer in den Scrolling-Modus. Erst wenn der Cursor (s.d.) nach oben bewegt wird, erfolgt eine Rückschaltung in die Page-Betriebsart.

Parity (Parität)

Die Verwendung eines zusätzlichen Paritätsbits als Summe der Datenbits stellt bei der Datenübertragung eine einfache und oft ausreichende Methode zur Erkennung von Übertragungsfehlern dar. Ein typischer Anwendungsfall ist die serielle Aussendung von ASCII-Zeichen mit je sieben Bits. Alle sieben Bits werden aufaddiert. Die sich ergebende u.U. mehrstellige Binärsumme wird mit 00000001 (binär) maskiert, so daß nur das niederwertigste Bit der Summe übrigbleibt, das dann als Paritätsbit schließlich mit übertragen wird. Empfangsseitig wird wiederum die Summe der ersten sieben Bits gebildet und ihr niederwertigstes Bit mit dem empfangenen Paritätsbit verglichen. Fällt der Vergleich negativ aus, so trat ein Übertragungsfehler auf.

Die Fehlererkennung mit einem Paritätsbit ist nicht unproblematisch: Traten innerhalb der insgesamt acht übertragenen Bits zwei, vier, sechs oder gar acht Fehler auf, so werden sie

nicht als solche erkannt. Aufwendiger, aber sicherer ist es daher, nicht nur ein Paritätsbit auszustrahlen, sondern zumindest nach jedem größeren Datenblock eine 8- oder 16-bit-Prüfsumme zur Fehlererkennung zu übertragen. Dieses Verfahren ist bei der Kassettenaufzeichnung allgemein üblich.

Pascal

Pascal ist – wie Basic – eine sogenannte "höhere" Programmiersprache. Sie wurde Anfang der 70er Jahre von Prof. N. Wirth in Zürich definiert und stellt eine Art Weiterentwicklung der technisch-wissenschaftlichen Sprache Algol 60 dar. Das Erlernen von Pascal ist schwieriger als von Basic und erfordert mehr Disziplin und Systematik. Wesentliche Eigenschaft ist das Konzept von Datentypen, das auf der Feststellung beruht, daß fehlerfreie Programme leichter entwickelt werden, wenn man jeder Programmvariablen einen während des Laufs prüfbaren und dem Problem möglichst entsprechenden Wertebereich zuordnet. Basis-Datentypen sind "Boolean", "Integer", "Char" und "Real". Benutzerspezifische Datentypen können durch Definition der zulässigen Werte eingeführt werden, z.B. "Type color = (red, green, yellow, blue)". Basisvariablen können mit "Array", "Record", "Set" und "File" zu Gruppen zusammengefaßt werden. Das Pascal-Programm selbst wird – ähnlich wie in Algol – in einer Blockstruktur geschrieben, wobei ein Block auf eine Prozedur, eine Funktion oder das Programm selbst beschränkt ist. Pascal eignet sich also vorzüglich für die "strukturierte" Programmierung.

Ähnlich wie bei Basic kann ein Mikroprozessor Pascal entweder mit einem Interpreter oder einem Compiler verarbeiten, wobei aus Geschwindigkeitsgründen oft dem Compiler der Vorzug gegeben wird.

Pointer

Als Pointer ("Zeiger") bezeichnet man eine oder mehrere Speicherzellen, deren Inhalt eine Adresse angibt und die dadurch einen "indirekten" Zugriff erlauben. Ein Beispiel: Der Befehl

LDA (0054) bedeutet, daß der Akku der CPU mit dem Inhalt jener Speicherzelle geladen werden soll, deren Adresse durch den Inhalt der Zellen 0054 und 0055 spezifiziert ist. 0054 und 0055 sind hier zwei 8-bit-Speicherzellen, die – zusammengehängt – einen 16-bit-Pointer ergeben.

Besonders bei der CPU 6502 haben solche Pointer eine große Bedeutung, weil dieser Mikroprozessor nicht über 16-bit-Register für die indirekte Adressierung verfügt und deshalb die Zero-Page (s.d.) als Ersatz dafür verwendet. Deren 256 Byte lassen sich nämlich als 128 16-bit-Pointer verwenden.

RAM

Random Access Memory, Schreib-Lese-Speicher. In das RAM eines Computers kann der Mikroprozessor Daten einschreiben und später aus ihm wieder auslesen. Welches von beidem gerade geschieht, wird durch den Pegel auf der Read/Write-Steuerleitung bestimmt, den der Prozessor anlegt.

In den heute üblichen Personal Computern werden meist "statische" RAMs verwendet, wobei jedes Bit in einem Flipflop (eine aus zwei oder mehr Transistoren bestehende Kippschaltung) gespeichert wird, z.B. die IC-Typen 2102 (1 Kbit bzw. 1/8 KByte) und 2114 (4 Kbit bzw. $^1/_2$ KByte) mit einer Zugriffszeit (Zeit zwischen dem Anlegen der Adresse durch die CPU und der Verfügbarkeit der Daten) von etwa 450 ns.

RAM-ICs mit sehr hoher Speicherdichte (64 Kbit pro Chip und mehr) lassen sich derzeit noch leichter in einer Technik realisieren, die nur einen Transistor pro Bit benötigt. Solche "dynamischen RAMs" arbeiten praktisch mit der Kapazität zweier voneinander isolierter Siliziumflächen als Speichermedium. Da ein solcher Kondensator die auf ihn aufgebrachte Ladung nicht beliebig lange speichern kann, ist es erforderlich, sie in Abständen von wenigen Millisekunden aufzufrischen ("Refresh"). Dies kann entweder über eine spezielle zusätzliche Schaltung geschehen oder aber, wenn ein dafür geeigneter Prozessortyp verwendet wird (Z-80), durch die CPU selbst. Da jedoch dieser Refresh-Zyklus das "Timing" der CPU und die gesamte Handhabung des Speichers etwas kompliziert, geht der

Trend allgemein zu den statischen RAM-Bausteinen, die allerdings bei vergleichbarer Speicherkapazität noch etwas teurer sind als dynamische RAMs.

In einem Basic-Computer ist ein RAM-Bereich von etwa 4 KByte das vertretbare Minimum, solange keine längeren Texte oder größere Mengen an Daten gespeichert werden sollen. 32 KByte RAM lassen dagegen auch für anspruchsvolle Aufgaben kaum noch Wünsche offen. (Hierbei wird davon ausgegangen, daß der Basic-Interpreter nicht im RAM, sondern im Festwertspeicher bzw. ROM abgelegt ist.)

ROM

ROM ist die Abkürzung von "Read-Only Memory", Nur-Lese-Speicher. Der Anwender hat nicht die Möglichkeit, in diesen Festwertspeicher selbst Daten einzuschreiben. Aus diesem Grund ist auch (im Gegensatz zum RAM, s.d.) keine Read/Write-Steuerleitung vorhanden.

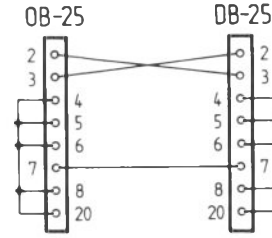
In einem Basic-Computer sind meist mehrere ROM-ICs vorhanden, die den Basic-Interpreter und andere für den Betrieb des Computers erforderliche Festprogramme beinhalten, wie etwa die Routinen für die Tastaturabfrage und die Display-Ansteuerung.

Heute arbeitet man meist mit ROM-Chips von je 4 ... 8 KByte Speicherkapazität. So findet man z.B. im AIM-65 bzw. PC-100 vier ROM-ICs, die zusammen 16 KByte enthalten – 8 KByte als Monitorprogramm (s.d.) und weitere 8 KByte für den Basic-Interpreter. Abarten des ROM sind PROM und EPROM, die sich vom Anwender selbst programmieren lassen (jedoch nicht, wie ein RAM-Baustein, von der CPU des Computers).

RS-232-Schnittstelle

Die RS-232-Schnittstelle ist die am häufigsten vorzufindende Norm für die serielle Datenübertragung (s.d.) innerhalb eines Computersystems, z.B. zwischen dem Computer selbst und einem Datensichtgerät. Sie wurde von der Electronics Industry Association (EIA) festgelegt und umfaßt physikalische und elektrische Größen.

So verbindet man die RS-232-Anschlüsse eines Computers mit den entsprechenden eines Druckers oder Bildschirm-Terminals



Die EIA entwickelte den RS-232-Standard für Übertragungsgeschwindigkeiten bis zu 19200 bit/s bei Leitungslängen von höchstens etwa 30 m. Eine logische Null ist durch einen Pegel zwischen −1,5 V und −36 V, eine Eins durch +1,5 V bis +36 V charakterisiert, bezogen auf das Massepotential. Der "Sender" sollte möglichst niederohmig sein, der Empfänger möglichst hochohmig. Die Verbindung eines Computers und eines Terminals mit RS-232-Norm ist sehr einfach. Es muß erwähnt werden, daß unterschiedliche Steckerbelegungen möglich sind; die in der Abbildung angegebenen Stiftnummern stellen nur eine von mehreren Möglichkeiten dar.

Für höhere Übertragungsgeschwindigkeiten und größere Leitungslängen gibt es noch die Standards RS-422 und RS-423, die jedoch nicht so verbreitet sind. Die meisten käuflichen Telefon-Modems (s.d.) benutzen die RS-232-Schnittstelle, die manchmal auch mit V-24-Schnittstelle bezeichnet wird.

Return

Die mit Return oder auch CR (Carriage Return) bezeichnete Taste ist eine der wichtigsten auf der Computertastatur. Return ist zunächst einmal ein ASCII-Zeichen (s.d.) mit dem hexadezimalen Code 0D bzw. dezimal 13, das dazu dient, den Cursor auf dem Bildschirm an das linke Zeilenende zu setzen. Bei zahlreichen Computern (PET, TRS-80, ABC-80 usw.) wird damit gleichzeitig ein "Line Feed" ausgegeben (s. CRLF), um in die nächste Zeile zu gehen. Dabei wird LF vom Computer selbst erzeugt.

In Basic dient die Return-Taste generell dazu, eine Eingabe abzuschließen, z.B. eine neu eingegebene Programmzeile oder auch eine vom Computer angeforderte Zahl (INPUT-Anweisung). Die einzige Eingabemöglichkeit von der Tastatur her, bei der das Drücken der Return-Taste nicht erforderlich ist, ist der Basic-Programmbefehl GET. Beim TRS-80 ist "Enter" die dem Return-Zeichen entsprechende Taste; sie hat die gleiche Wirkung.

Scrolling

Als Scrolling bezeichnet man das Aufwärtswandern aller Zeilen auf einem Datensichtgerät, sobald die unterste Zeile vollgeschrieben ist. Manche Datensichtgeräte lassen sich von Hand oder mit einem geeigneten ASCII-Steuerzeichen (nicht einheitlich genormt) zwischen der Scrolling- und der "Page"-Betriebsart umzuschalten.

Bei manchen Terminals, z.B. beim CT-64 von COI, sind bestimmte Steuerbefehle wie Bildschirmlöschen oder "Cursor home" im Scrolling-Modus unzulässig, weil eine "Bildschirmseite" dann nicht unbedingt am oberen Bildschirmrand, sondern in einer nicht definierten Zeile beginnt. Dann ist es ratsam, kurz in den Page-Modus zu gehen, den Bildschirm zu löschen und dann auf den Scrolling-Modus zurückzuschalten. (Siehe auch "Page-Modus".)

Serielle Datenübertragung

Um Daten oder Programme über eine Zweidrahtleitung, per Telefon oder gar per Funk übertragen zu können, muß ein serielles Datenformat verwendet werden, das die einzelnen Informationsbits nacheinander auf den Weg schickt. Dafür sind unterschiedliche Verfahren üblich.

Bei der synchronen Datenübertragung handelt es sich um einen fortwährenden Datenfluß, bei dem einfach alle zu übertragenden Bits mit einer bestimmten Schrittgeschwindigkeit (s. Baud) nacheinander gesendet und synchron dazu empfangen werden. Der Abstand von einem Bit zum nächsten ist unter

Jedes Zeichen beginnt mit einem Startbit, damit der Anfang auch dann erkannt werden kann, wenn das erste Bit (2^0) einmal 1 ist, und endet mit einem Stopbit, damit das nächste Startbit in jedem Fall einen Wechsel von 1 auf 0 bewirkt

allen Umständen konstant. Das Verfahren hat den großen Nachteil, z.B. bei Datensichtgeräten nicht anwendbar zu sein, weil man vom "Operator" nicht erwarten kann, daß er die Tastatur mit absolut gleichmäßiger Geschwindigkeit betätigt.

Aus diesem Grunde wird in der Mehrzahl der Fälle ein asynchrones Datenformat verwendet. Die Aussendung geschieht byteweise, d.h. in Gruppen von je acht Bits. Bei den acht Bits kann es sich z.B. um die sieben Bits eines ASCII-Zeichens mit angehängtem Parity-Bit (s.d.) handeln.

Damit das Empfangsgerät weiß, wann ein Zeichen beginnt, stellt man allen Bytes ein sog. Startbit voran, das immer Null und ebenso lang wie alle Datenbits ist. Damit das Startbit überhaupt erkannt wird, muß der Leitungszustand zwischen den einzelnen Zeichen log. 1 sein; dies wird wiederum durch die Verwendung eines Stopbits erzwungen, das log. 1 und ebenfalls genauso lang wie ein Datenbit dauert. Es wird an jedes ausgestrahlte Byte angehängt. Insgesamt werden also immer zehn Bits pro Zeichen übertragen: Acht Datenbits, wovon das letzte ein Parity-Bit sein kann, ein Startbit und ein Stopbit.

Von dieser Regel gibt es zwei Ausnahmen: Bei einer Schrittgeschwindigkeit von 110 Bd arbeitet man mit zwei Stopbits, d.h. die effektive Stopbitlänge entspricht zwei Datenbits; im Baudot-Code (s.d.) werden nur fünf Datenbits übertragen, und das Stopbit dauert 1,5mal so lange wie ein Datenbit, so daß das gesamte Zeichen 7,5 Bits lang ist (obwohl man natürlich nicht von halben Bits sprechen kann – das bezieht sich hier ja nur auf die Zeitdauer).

Für das Zusammenspiel zwischen Computer und Peripheriegeräten ist neben dem richtigen Datenformat auch das Überein-

stimmen der Hardware-Schnittstellenbedingungen, wie Pegel, Polarität usw. ausschlaggebend (s. RS-232, V-24, TTY).

Software

Was man genau unter Software versteht, darüber streiten sich noch die Gelehrten (offenbar haben sie nichts besseres zu tun). Allgemein ist Software ("Weichware") alles jene in einem Computersystem, was man nicht anfassen kann – Programme und Daten also. Natürlich gibt es den Grenzfall, daß man diese Dinge doch anfassen kann – wenn sie nämlich in ROMs (s.d.) fest gespeichert sind; hier hat man sich auf den Begriff "Firmware" geeinigt. Wir können es uns auch einfach machen und zu alledem Software sagen, was in einem Computersystem nicht als "Hardware" bezeichnet wird – letzteres läßt sich nämlich etwas einfacher definieren... Übrigens ist es ausgesprochen teuer, sich Software für bestimmte Anwendungsfälle schreiben zu lassen – das Selbstschreiben lohnt.

Stack

Der Stack ist ein besonderer Speicherbereich im Mikrocomputer, auf den man nicht durch Angabe einer bestimmten Adresse zugreift, sondern der eine Struktur wie ein Getreidesilo hat. Bestimmte Befehle der CPU legen Daten auf dem Stack ab und andere holen diese Daten zurück. Dabei werden die zuletzt abgelegten Daten zuerst zurückgeholt (last in, first out = LIFO-Speicher).

Eine typische Anwendung des Stack sind Unterprogramme. Beim Befehl JSR (Jump Subroutine) bzw. CALL oder GOSUB (letzteres in Basic) wird die momentane Adresse des Hauptprogrammes, wo die Befehlsausführung nach dem Unterprogramm fortgesetzt werden soll, auf dem Stack abgelegt. Der am Schluß des Unterprogramms stehende Befehl RTS bzw. RETURN holt sich diese Adresse wieder vom Stack und setzt das Programm an ihr fort.

Bei Maschinen-Unterprogrammen werden für eine Unterprogramm-Ebene zwei Bytes im Stack benötigt. Das Verfahren

bei der Interrupt-Verarbeitung funktioniert ähnlich, hier wird außer der Rücksprungadresse aber noch meist das Statusregister der CPU mit auf dem Stack abgelegt, so daß hier drei Bytes benötigt werden. Mit besonderen Befehlen (z.B. PHA/PLA) kann der Programmierer zusätzlich noch andere CPU-Register auf den Stack "retten", um sie vor einer Veränderung innerhalb des Unterprogramms oder der Interrupt-Routine zu schützen; dann werden noch mehr Bytes benötigt.

Das Stack-Verfahren gestattet die Verwendung nahezu beliebig verschachtelter Unterprogramme und Interrupts. Bei der CPU 6502 liegt der Stack hardwaremäßig fest im Bereich 0100...01FF; andere Prozessoren erlauben es, jeden beliebigen Speicherbereich als Stack zu benutzen, indem man am Programmanfang den "Stackpointer" entsprechend setzt. Die Abspeicherung beginnt stets am oberen Ende des jeweiligen Bereichs, z.B. bei 01FF beim 6502.

Der Stack wird auch vom Basic-Interpreter benutzt. Hier werden erheblich mehr Bytes pro Unterprogrammebene als bei Maschinenprogrammen abgespeichert. Beim AIM-65 bzw. PC-100 benötigt jede GOSUB-Ebene – solange sie nicht mit RETURN beendet wurde – 6 Bytes. Jede aktive FOR...NEXT-Schleife beansprucht sogar 22 Bytes auf dem Stack; eine Klammeröffnung in einem Ausdruck 4 Bytes und ein vorübergehend benötigtes Zwischenergebnis 12 Bytes. Man kann sich ausrechnen, daß die 255 Bytes eines 6502-Stacks bald erschöpft sein können, wenn man unüberlegt FOR-NEXT-Schleifen mit vielen Klammerausdrücken und Unterprogrammebenen programmiert.

String

Als String bezeichnet man üblicherweise jede Zeichenkette. Dabei kann es sich um einen kurzen Text handeln, eine Folge von Steuerzeichen für ein Terminal oder eine andere Folge von ASCII-Zeichen. Auch eine Zahl kann als String betrachtet werden; so wäre z.B. die Zahl 423, als ASCII-Folge geschrieben, 34 32 33. In Basic kann ein String auch durch eine Variable repräsentiert werden. Ihr Name und jeder andere Basic-Aus-

druck, der einen String darstellt, wird mit einem Dollar-Zeichen versehen, z.B. A$, CHR$, LEFT$ usw., um ihn eindeutig von numerischen Ausdrücken zu unterscheiden. Strings sind bei den heutigen Basic-Mikrocomputern auf eine Länge von max. 255 Zeichen begrenzt.

Eine besondere Problematik der String-Verarbeitung ist, daß u.U. nicht alle im String enthaltenen Zeichen beim Ausdruck auch sichtbar sind. Ein Beispiel dafür ist das Line-Feed-Zeichen (ASCII dezimal 10), das die CBM-Floppy beim Laden eines String an dessen Anfang setzt. Solche unsichtbaren Zeichen lassen sich feststellen, indem man die Zahl der Zeichen im String mit PRINT LEN(String) feststellt und mit der Anzahl der tatsächlich bei PRINT(String) ausgedruckten Zeichen vergleicht. Vermutet man, daß das erste Zeichen im String ein unsichtbares Steuerzeichen ist, so kann man sein dezimales ASCII-Äquivalent mit PRINT ASC(String) feststellen. (Siehe auch "Variablen".) In Basic lassen sich Strings mit "+" zusammenfügen; Beispiel:

A$="A":B$="B":PRINTA$+B$ ergibt AB.

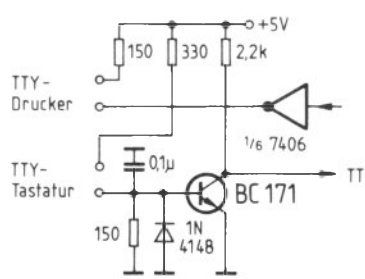
TTL (Transistor-Transistor-Logik)

ICs (s.d.) werden in unterschiedlichsten Technologien gefertigt, die Leistungsverbrauch, Integrationsdichte, Geschwindigkeit und andere Faktoren beeinflussen. In der Computertechnik hat die TTL-Technologie eine große Bedeutung erlangt; sie gestattet Arbeitsfrequenzen bis zu einigen 10 MHz und kommt mit nur einer Versorgungsspannung (+5 V) aus. Der Stromverbrauch pro IC liegt meist zwischen 10 mA und 100 mA. Typische TTL-ICs sind die Vertreter der bekannten "74er"-Serie (7400, 7402, 74123 usw.). Später entwickelte TTL-Technologien sind Schottky (74S...), Low-Power-Schottky (74LS), Low-Power-TTL (74L...) und High-Speed-TTL (74H...). Die TTL-Technik arbeitet zumeist mit positiver Logik. Die binäre Null entspricht dem als "L-Pegel" (Low) bezeichneten Spannungsbereich 0...0,8 V, die binäre 1 dem H-Pegel 2...5 V (High). Ein TTL-Ausgang kann normalerweise zehn TTL-Eingänge treiben. MOS- und CMOS-ICs können dagegen normalerweise nur

einen TTL-Eingang treiben, da sonst wegen der zu hohen Belastung Pegelprobleme auftreten. Eine Sonderform sind TTL-ICs mit "Open-Collector"-Ausgängen, die hochohmig sind, wenn der Ausgang auf log. 1 geht.

TTY-Schnittstelle

Wenn ein Computer über eine TTY-Schnittstelle verfügt, so bedeutet dies, daß sich ein ASCII-Fernschreiber (Teletype) direkt an ihn anschließen läßt. Solche Fernschreiber besitzen vier Klemmen, die zu zwei galvanisch voneinander getrennten Stromkreisen gehören. Beide Stromkreise werden oft auch als 20-mA-Stromschleifen bezeichnet; so viel Strom fließt in jedem von ihnen im Ruhezustand, d.h. wenn kein Zeichen ein- oder ausgegeben wird. Die Tastatur der TTY unterbricht nach einem bestimmten Bitmuster (s. serielle Datenübertragung) den Strom und liefert so eingetippte Zeichen an den Computer; umgekehrt unterbricht der Computer nach einem das jeweilige Zeichen charakterisierenden Bitmuster den Strom, der durch den Druckermagneten der TTY fließt. Teletypes arbeiten meist mit einer Schrittgeschwindigkeit von 110 Bd und dürfen nicht mit den herkömmlichen Postfernschreibern verwechselt werden, die im Baudot-Code arbeiten. Ein modernerer Ersatz für die noch lärmerzeugenden Teletype-Maschinen ist z.B. der Thermodrucker Silent 700 (Texas Instruments) mit ASCII-Tastatur. Außerdem verfügen auch zahlreiche Datensichtgeräte über eine der TTY entsprechende 20-mA-Schnittstelle.



Mit geringem Aufwand läßt sich die 20-mA-Stromschleife der TTY-Schnittstelle an TTL-Ein- und -Ausgänge anpassen

Timer

Zahlreiche Computer verfügen über eine eingebaute "Uhr", die dauernd läuft, vom Mikroprozessor in geringen Abständen weitergestellt wird und im Prinzip nur aus mehreren Speicherstellen für Sekunden, Minuten und Stunden besteht. Diese Speicherstellen können per Programmbefehl, z.B. PRINT TI$, auf den Bildschirm geholt werden.

Im Prinzip wäre das regelmäßige Weiterstellen dieser Software-Uhr natürlich möglich, indem der Prozessor in einer endlosen Schleife läuft. Dieser Weg ist in der Praxis aber unbeschreitbar, der Mikroprozessor hat ja schließlich auch noch andere Aufgaben zu erledigen.

Aus diesem Grund verwendet man einen sog. Interrupt (s.d.), der die CPU z.B. alle Sekunden unterbricht und veranlaßt, daß die Uhr um eine Sekunde weitergestellt wird. Der Interrupt kann entweder von einem Quarzoszillator mit nachfolgender Teilerkette, durch Verwenden der Netzfrequenz oder – und das ist am elegantesten – mit einem Timerbaustein erzeugt werden. Zwei solche Timer besitzen die Computer PET-2001 und CBM-3032 in dem VIA-Baustein 6522 ("Versatile Interface Adapter"). Sie lassen sich auf bestimmte Werte per Programmbefehl vorsetzen und zählen von diesem Wert aus abwärts. Wenn der Zählerstand Null erreicht ist, so wird der Interrupt ausgelöst und der Zähler des Timers wieder mit dem ursprünglichen Wert geladen – das Spiel beginnt von neuem. Das Taktsignal für den Timer wird aus dem CPU-Taktoszillator gewonnen.

Außer für die Realisation einer Software-Uhr läßt sich ein Timer für zahllose andere Aufgaben einsetzen, z.B. für das "Timing" bei serieller Datenübertragung oder zum Messen externer Zeitabläufe.

UART

Ein *U*niversal *A*synchronous *R*eceiver/*T*ransmitter ist eine Schaltung, die in Senderichtung parallel anliegende Daten in Serielle und in Empfangsrichtung serielle und parallele Daten

umwandelt. Die Umwandlung geschieht mit einem von außen anzulegenden Takt, der ein Vielfaches der Schrittgeschwindigkeit (s. Baud) darstellt. Eine Alternative zu einem UART ist die Serien/Parallel- bzw. Parallel/Serien-Wandlung per Software, die eine automatische Anpassung an die Schrittgeschwindigkeit des externen Gerätes ermöglicht (siehe auch "Serielle Datenübertragung").

Unterprogramm

Es kommt häufig vor, daß eine bestimmte Befehlsfolge mehrmals praktisch gleichartig in einem Programm benötigt wird. Es wäre Speicherplatzverschwendung, tatsächlich überall, wo dies der Fall ist, die Befehlsfolge in voller Länge einzusetzen und so kostbaren Speicherplatz zu verschwenden.

Die Alternative ist die Verwendung von Unterprogrammen (Subroutines). Den mehrfach benötigten Programmteil schreibt man irgendwo – möglichst mit abseits liegenden Zeilennummern bzw. Adressen – in den Speicher und schließt ihn mit dem Rücksprungbefehl RETURN ab.

Jedesmal, wenn man die Befehlsfolge im Hauptprogramm benötigt, kann man sie jetzt mit GOSUB aufrufen. Beginnt z.B. ein Unterprogramm zum Errechnen des hyperbolischen Sinus bei der Zeilennummer 2000 im Basic-Programm, so braucht man jedesmal, wenn man den Hyperbelsinus ausrechnen möchte, nur GOSUB 2000 zu schreiben – und das kann im Hauptprogramm beliebig oft geschehen.

Selbstverständlich ist es auch möglich, von einem Unterprogramm ein weiteres aufzurufen, d.h. verschachtelt zu arbeiten. Da aber jedesmal die Rücksprungadresse auf dem Stack (s.d.) gespeichert werden muß, kann die Tiefe dieser Verschachtelung nicht beliebig groß gemacht werden. In einem Basic-Programm belegt eine Rücksprungadresse gewöhnlich 6 Bytes, in einem Maschinenprogramm nur 2 Bytes.

Unterprogramme haben auch eine große Bedeutung bei der sogenannten strukturierten Programmierung. Dabei bemüht man sich, das Problem in kleine, übersichtliche Teilprobleme zu zerlegen und diese dann in Unterprogrammen zu behandeln.

Das Hauptprogramm ist dann ebenfalls sehr kurz und übersichtlich, da es im wesentlichen nur die Unterprogramme für die einzelnen Teilaufgaben aufruft. Diese Programmierideologie nimmt den etwas größeren Speicherplatzbedarf zugunsten der Übersichtlichkeit in Kauf.

VIA

Als "Versatile Interface Adapter" bezeichnen die drei Hersteller MOS Technology, Rockwell und Synertek den Baustein 6522. Dabei handelt es sich um einen Chip, der zu den derzeit komplexesten Eingabe/Ausgabe-Bausteinen gehört. Es würde unsere kurze Betrachtung des VIA bei weitem sprengen, seine zahlreichen Funktionen und Möglichkeiten auch nur annähernd zu beschreiben; hier sei auf die Datenblätter der Hersteller sowie auf einen Aufsatz in dem 1979 erschienenen FUNKSCHAU-Sonderheft "Hobbycomputer 2" von R. Löhr verwiesen.

Der Baustein 6522 enthält 16 Register, von denen jedem eine bestimmte Adresse zugewiesen ist (Memory-Mapped). Diese sind: Zwei Port-Ausgabe-Register (ORA, ORB); zwei Port-Richtungsregister, mit denen bitweise bestimmt werden kann, ob eine bestimmte Portleitung als Ein- oder Ausgang dienen soll (DDRA, DDRB); zwei Port-Leseregister mit einschaltbarem Lese-Zwischenspeicher (IRA, IRB); zwei 16 bit breite Timer zur Zeitmessung, Ereigniszählung, Impulserzeugung oder für den seriellen Datenaustausch mit peripheren Geräten – dafür ist auch das Schieberegister (SR) vorhanden; ein Hilfsregister ACR für die Steuerung von Timer und Port-Zwischenspeichern; ein Steuerregister PCR für die Handshake-Bedienung von Peripherie z.B. via IEC-Bus; ein Interrupt-Flagregister IFR, aus dem die CPU erkennen kann, welcher Teil des VIA einen Interrupt auslöste; und schließlich ein Interrupt Enable Register IER, mit dem den Bausteinen des VIA einzeln gestattet werden kann, Interrupts auszulösen.

Die Mikrocomputer PET-2001, CBM-3016/3032, AIM-65 und PC-100 enthalten ein oder mehrere VIAs, deren Adressen aus der *Tabelle* hervorgehen.

VIA-Registeradressen beim AIM-65 und beim PET-2001

AIM-65 Anwender-VIA hexadezimal	AIM-65 System-VIA hexadezimal	PET-2001 System- und Anwender-VIA hexadezimal bzw. dezimal		Register
A000	A800	E840	59456	ORB Ausgaberegister Port B
A001	A801	E841	59457	ORA Ausgaberegister Port A (mit Handshake)
A002	A802	E842	59458	DDRB Datenrichtungsregister für Port B
A003	A803	E843	59459	DDRA Datenrichtungsregister für Port A
				Timer 1:
A004	A804	E844	59460	T1C-L Lesen Zähler ‚low', Schreiben Vorspeicher 'low'
A005	A805	E845	59461	T1C-H Lesen und Schreiben Zähler ‚high'
A006	A806	E846	59462	T1L-L Lesen oder Schreiben in Vorspeicher ‚low'
A007	A807	E847	59463	T1L-H Lesen oder Schreiben in Vorspeicher ‚high'
				Timer 2:
A008	A808	E848	59464	T2C-L Lesen Zähler ‚low', Schreiben Vorspeicher ‚low'
A009	A809	E849	59465	T2C-H Lesen und Schreiben Zähler ‚high'
A00A	A80A	E84A	59466	SR Schieberegister
A00B	A80B	E84B	59467	ACR Hilfsregister
A00C	A80C	E84C	59468	PCR Steuerregister
A00D	A80D	E84D	59469	IFR Interrupt-Anzeigeregister
A00E	A80E	E84E	59470	IER Interrupt Enable Register
A00F	A80F	E84F	59471	ORA Ausgaberegister für Port A (ohne Einfluß auf Handshakesignal)

Die einfachste VIA-Verwendung besteht darin, über die Ports A und B Signale auszugeben oder abzufragen. Die Ein- oder Ausgangssignale müssen dabei TTL-Pegel sein; jede VIA-Leitung kann nur eine TTL-Last treiben (Fan-Out 1). Jedem Bit der Portregister entspricht eine Leitung; eine 1 im entsprechenden Bit von DDRA/DDRB zeigt an, daß es sich um einen Ausgang handelt, bei Null um einen Eingang. Lesen und Schreiben kann natürlich nur byteweise, d.h. in Gruppen von acht Bits, geschehen; die sich ergebende Dezimalzahl für jedes Register ist die Summe der Bitwertigkeiten.

Variable

Der Ausdruck "Variable" hat in Basic die gleiche Bedeutung wie in der Algebra. Variablen sind Namen für Repräsentanten von Daten. Diese Daten können ganze Zahlen, Fließkomma-Zahlen oder Strings (s.d.) sein. Bei Tiny-Basic können die Variablennamen aus nur einem Buchstaben (A...Z) bestehen, bei komfortableren Interpretern dagegen aus einem oder zwei Buchstaben oder aber aus einem Buchstaben und einer Ziffer. Handelt es sich um eine Stringvariable, also um eine Variable, die einen Text oder eine beliebige ASCII-Zeichenfolge repräsentiert, so ist ihrem Namen ein Dollar-Zeichen nachzustellen (Beispiel: A$). Viele Basic-Computer erlauben das Einsparen von Speicherplatz und Rechenzeit durch die Verwendung von Ganzzahl-Variablen (s.d.); sie sind durch ein Prozentzeichen hinter ihrem Namen zu kennzeichnen (Beispiel: B%). Alle nicht näher bezeichneten Variablentypen werden vom Computer als Fließkommazahlen interpretiert, die den gesamten zulässigen Rechenbereich (z.B. 10^{-37} bis 10^{+37}) umfassen können und je nach Rechnertyp mit 6...13 Mantissenstellen sowie zwei Exponentenstellen plus Mantissen- und Exponenten-Vorzeichen verarbeitet werden.

Wie in der Algebra, so gibt es auch in Basic mehrdimensionale Variablen, die ein Feld darstellen. Dies ist auch mit allen gerade erwähnten Variablentypen möglich. Die eindimensionale Variable A(I) kann dabei mehrere Einzelvariablen er-

setzen, nämlich eine für jeden (stets ganzzahligen, positiven) Wert von I. I darf auch Null sein, so daß die Variable A(I) z.B. 151 Einzelvariablen ersetzt, wenn I maximal 150 werden kann. Dieser maximale Wert von I muß am Anfang des Basic-Programms mit dem Befehl DIM A(151) festgelegt werden, um dem Basic-Interpreter die Möglichkeit zu geben, den zur Verfügung stehenden Speicherplatz optimal aufzuteilen.

Selbstverständlich sind auch mehrere Dimensionen möglich, bei den meisten Basic-Computern maximal 255. Die Variable kann dann z.B. A(I,J,K) lauten, der notwendige DIM-Befehl könnte – wenn I, J und K maximal 20 werden können – DIM A(21,21,21) lauten.

Vektor

Bei Maschinenprogrammen bezeichnet man eine Folge von zwei Bytes, die zusammen eine 16-bit-Sprungadresse enthalten, als Vektor, sofern sie nicht innerhalb des Programms selbst stehen. Ein Beispiel: Wenn in einem Mikrocomputersystem ein Interrupt ausgelöst wird (s.d.), so wird das gerade laufende Programm unterbrochen. Bei der CPU 6502 wird jetzt zu einem anderen Programm gesprungen, dessen Anfangsadresse z.B. in den Speicherzellen FFFE/FFFF (hexadezimal) steht. Die Adressenbelegung beim AIM-65 sieht dann etwa so aus:

.
.
.

E078
Hier beginnt das Programm, das beim Auftreten des Interrupts abgearbeitet wird

.
.
.

FFFE 78
FFFF E0

Diese beiden Speicherstellen geben an, wo der Prozessor bei einem Interrupt die Arbeit fortzusetzen hat und stellen einen "Vektor" dar

Ab und zu führen Interrupt-Vektoren auch zu einem indirekten Sprung über zwei RAM-Zellen, so daß der Benutzer dann selbst definieren kann, wo sein Interrupt-Programm beginnt, obwohl er den Interrupt-Vektor selbst nicht ändern kann, weil er im ROM des Mikrocomputers steht. Wenn wir bei unserem obigen Beispiel bleiben, sieht das so aus:

.
.
.

E078 JMP (A404)

.
.
.

Es erfolgt ein indirekter Sprung über den bei A 404 stehenden zweiten Vektor, der im RAM-Bereich liegt und vom Benutzer auf den Start seiner Interrupt-Routine gesetzt werden kann.

Video-RAM

Die Darstellung von Buchstaben, Ziffern und anderen Zeichen auf dem Bildschirm eines Computers oder Datensichtgerätes erfolgt etwa nach folgendem Prinzip:

Der Schirm ist z.B. in 16 Zeilen mit je 64 Buchstaben aufgeteilt, das sind 1024 Zeichen. Jedem Zeichenfeld des Bildschirms entspricht eine bestimmte Adresse eines Video-RAM-Speichers, wobei sieben, meist aber acht Bits für ein Zeichen verwendet werden. Der Speicher besitzt hier also eine Kapazität von 1024 Byte = 1 KByte. Mittels des DMA-Verfahrens (s.d.) wird aus diesem Speicher mit der notwendigen Bild-Wiederholfrequenz (z.B. 50 Hz oder 60 Hz) der Zeicheninhalt mit Hilfe des Zeichengenerators – einem ROM, das die Darstellungsform für jedes Zeichen enthält – auf dem Schirm sichtbar gemacht.

Jeder Zeichenposition auf dem Bildschirm entspricht also eine ganz bestimmte Adresse im RAM. Wenn der Mikroprozes-

sor des Computersystems in der Lage ist, jede mögliche Zeichenadresse über seine Adressen- und Datenbusse direkt anzusprechen, ist ein sehr schnelles Schreiben von Zeichen an jeder beliebigen Stelle des Schirms möglich, was sehr wichtig für bewegte grafische Darstellungen ist (z.B. Simulation einer Mondlandung u.ä.).

Ist der Computer jedoch über eine serielle Schnittstelle (RS-232, s.d.) mit einem Datensichtgerät verbunden, so ist diese direkte Adressierbarkeit des Bildschirms nicht gegeben. Vielmehr wird der Schirm vom Computer Zeile für Zeile oder notfalls mit Cursor-Bewegungen vollgeschrieben, was erheblich länger dauert.

Wort

Wenn in der Computer-Fachsprache von einem "Wort" die Rede ist, dann ist nicht etwa ein Befehlswort gemeint, sondern eine zusammenhängende Folge von so vielen Bits, wie der Prozessor, das Rechenwerk bzw. die CPU gleichzeitig verarbeiten kann. Bei 8-bit-Prozessoren ist die Wortbreite also gleich 8 bit (und damit gleich einem Byte), bei 16-bit-Prozessoren ist die Wortbreite 16 bit bzw. 2 Byte.

Zero Page

Als "Page" (Seite) bezeichnet man bei der Adressierung von Speichern denjenigen Bereich, bei dem die höherwertigen acht Adressenbits (bei 16 Adressenbits, wie das bei 8-bit-Computern üblich ist) konstant bleiben. Betrachtet man die 16-bit-Adresse als vierstellige Hexadezimalzahl (s.d.), so bedeutet die Bezeichnung "Page 3C", daß die Adressen 3C00 bis 3CFF gemeint sind – insgesamt also $2^8 = 256$ Byte. Sind die acht höherwertigen Adressenbits Null, so spricht man von der "Zero Page" (Seite Null), entsprechend dem Adressenbereich 0000...00FF.

Bei den Mikroprozessoren der Familien 65XX und 68XX hat diese Zero Page eine besondere Bedeutung. Denn während ein Befehl, der eine 16-bit-Adresse als Argument benutzt, in Maschinensprache drei Bytes belegt (ein Byte für den Operationscode

und zwei für die Adresse), so ist in der Zero-Page nur ein Zwei-Byte-Befehl erforderlich (Operationscode plus ein Byte für die 8-bit-Adresse).

Ein Beispiel: Will man den Inhalt der Adresse 346A in den Akku der CPU bringen, so lautet der Befehl in Maschinensprache AD 6A 34. Lautet die Adresse dagegen 006A, so wird der für die Zero-Page-Adressierung passende Operationscode gewählt, und der gleichwertige Befehl lautet nun A5 6A. (Dieses Beispiel bezieht sich auf den 6502; beim 6800 lauten die Operationscodes anders, und die Reihenfolge der beiden Adressenbytes bei der ersten Version ist vertauscht.)

Wegen der schnellen Zugriffsmöglichkeit zu den 256 Speicherzellen der Zero-Page verwendet man diesen Adressenbereich gerne für die Abspeicherung häufig gebrauchter Daten.

Bei anderen Prozessoren, z.B. 8080, Z-80, 8085 usw., liegt die Zero-Page meist im ROM-Bereich, weil ein Reset (s. Interrupt) zu einem Sprung dorthin führt. Ab 0000 findet man dort deshalb das fest gespeicherte Initialisierungsprogramm von Computern mit diesen CPU-Typen.

5 Basic-Befehle

Im folgenden werden alle üblichen Basic-Befehle alphabetisch aufgeführt. Die dabei verwendeten Beispiele sollen dazu dienen, die Wirkung und Anwendung bestimmter Befehlsworte zu verdeutlichen. Wenn von "Direktbefehlen" die Rede ist, so bedeutet dies, daß der Befehl nicht als Programmbefehl verwendet werden kann, sondern nur als Hilfsmittel für den Programmierer dient, wie etwa LIST oder RUN. In diesem Punkt unterscheiden sich die Computertypen zum Teil ein wenig, so daß empfohlen wird, im Zweifelsfall auszuprobieren, ob eine Verwendung innerhalb des Programme möglich ist. Bestimmte Befehle, z.B. GET, können nicht als Direktbefehle, sondern ausschließlich innerhalb eines Programmes verwendet werden.

Jede Zeile, die mit einer Zahl beginnt, wird vom Computer normalerweise als Programmzeile interpretiert, und die am Zeilenanfang stehende Zahl wird als Zeilennummer verwendet. Innerhalb einer Programmzeile können aber auch mehrere Befehle stehen, die mit einem Doppelpunkt oder (seltener) mit einem „\" (Schrägstrich) voneinander getrennt werden.

Leerräume zwischen Befehlen und Ausdrücken dürfen, um Speicherplatz zu sparen, bedenkenlos weggelassen werden. Schon bei der Eingabe ignoriert der Computer alle Leerräume (Spaces), die zwischen der Zeilennummer und dem ersten nichtnumerischen Zeichen stehen. Viele Computer gestatten auch das Abkürzen von zahlreichen Befehlen; diese Abkürzungen sind jedoch systemabhängig, weshalb hier alle Befehle in ihrer Grundform angegeben werden.

Zur Syntax ist zu bemerken, daß Stringausdrücke mit einem Dollarzeichen und Ganzzahlvariable mit "%" gekennzeichnet sind. String-Konstanten müssen außer bei DATA in Anführungszeichen gesetzt werden. Beim ABC-80 von Luxor müssen alle

Dollarzeichen durch '$' ersetzt werden, da dieser Computer einen anderen Zeichensatz besitzt.

ABS

Mit ABS läßt sich das Vorzeichen einer Zahl beseitigen bzw. positiv machen (Absolutwertbildung). Beispielsweise ergibt PRINT ABS(−5.123) den Wert 5.123. Exponent und Mantisse werden nicht verändert. In der Klammer hinter ABS darf eine beliebige Zahl oder numerische Variable stehen.

Da bei vielen Basic-Interpretern der NOT-Befehl beim Argument Null das Ergebnis −1 liefert (statt 1), läßt sich ABS auch einsetzen, um Boolesche Algebra mit den Werten 0 und 1 zu realisieren. PRINT ABS(NOT 0) ergibt dann – wie es sich gehört – den Wert 1.

AND

AND stellt eine Verknüpfung von zwei Entscheidungen oder auch zwei Gruppen von Entscheidungen dar, bei denen jede von beiden "wahr", also erfüllt sein muß. Die Entscheidungsgruppen können auch zwei Bytes sein: Jedes der acht Bits eines Bytes stellt ja eine Ja-Nein-Entscheidung dar. Die Verknüpfung 00000110 AND 11100011 ergibt zum Beispiel 00000010, denn nur für ein Bit gilt die Bedingung, daß beide Einzelentscheidungen "wahr" sind. Natürlich kann man in Basic normalerweise keine binären Zahlen verarbeiten; unser Beispiel würde in Basic, dezimal geschrieben, also lauten 6 AND 227, und das ebenfalls dezimale Ergebnis wäre 2 (siehe auch "hexadezimale Zahlen"). Außer für solche Verknüpfungen kann AND auch in IF-Anweisungen verwendet werden. Hier ein paar Beispiele:

Befehlsfolge	*Wirkung*	*Erklärung*
INPUT A,B: PRINT A AND B	Die binäre Und-Verknüpfung von A und B wird ausgedruckt	A und B werden im rechnerinternen Binärformat Bit für Bit Und-verknüpft

IF A=1 AND B=5 GOTO5	Das Programm verzweigt zu Zeile 5, wenn sowohl A=1 als auch B=5 gilt	Beide Bedingungen nach IF (es können auch mehr sein) müssen erfüllt sein
PRINTA=1AND B=5	0 wird ausgegeben, wenn nicht gleichzeitig A=1 und B=5 sind, andernfalls +/−1	Wenn die Einzelvergleiche A=1 oder B=5 erfüllt sind, liefern sie jeweils +/−1, sonst 0 (siehe „=")

APPEND

Bei Verwendung des Befehls LOAD wird ein Programm aus einem externen Speichermedium (z.B. Kassette) in den Computer-Arbeitsspeicher geladen, wobei das darin vorher befindliche Programm mit allen seinen Variablenwerten gelöscht wird. APPEND entspricht dem LOAD-Befehl, läßt das alte Basic-Programm jedoch im Speicher stehen. (Siehe auch "MERGE"!)

ASC

Mit ASC ist es möglich, den dezimalen Zeichencode (s. ASCII) eines Buchstabens oder anderen Zeichens festzustellen. Beispiel: PRINT ASC("A") liefert den Wert 65. Auch in einer Zuweisung ist ASC verwendbar, z.B. in B=ASC(M$).

Wenn das Argument in der Klammer hinter ASC nicht aus nur einem Zeichen besteht, sondern ein längerer String ist, so wird nur das dezimale Äquivalent des ersten Stringzeichens ermittelt.

AT

Dieser Befehl bewirkt zusammen mit PRINT den Ausdruck von Werten oder Texten an ganz bestimmten Stellen auf dem Bildschirm des Computers. Teilt man den Schirm zeilenweise in Zeichenpositionen auf, so besitzt z.B. der TRS-80 1023 Felder (64 in jeder Zeile). Der Befehl

```
PRINT AT 128,"TEST"
```

druckt das Wort TEST an die Zeichenposition 128, hier den Beginn der zweiten Bildschirmzeile.

ATN

ATN, gefolgt von einem Wert in Klammern, ergibt den Arcustangens dieses Wertes, meist im Bogenmaß. Ein Beispiel:

```
10 INPUT"WERT";A
20 PRINT "WINKEL=";ATN(A)
RUN
? 2
1.10715
```

ATN ist meist die einzige zur Verfügung stehende inverse trigonometrische Funktion in Basic-Interpretern.

AUTO

Normalerweise ist es notwendig, vor jede Befehlszeile eine Zeilennummer zu schreiben. Mit dem Direktbefehl (Kommando) AUTO kann man den Computer dazu bringen, nach dem Drükken der Return-Taste (wenn man eine Zeile eingegeben hat) in die nächste Zeile selbständig eine um 10 erhöhte Zeilennummer zu schreiben, so daß nur noch die Befehle selbst eingegeben werden müssen.

BYE

Das Direktkommando BYE dient dazu, um den Basic-Interpreter zu verlassen und zum Monitorprogramm zu springen, z.B. um ein Programm in Maschinensprache eingeben zu können. In größeren Computersystemen mit Timesharing (mehrere Terminals werden von einem gemeinsamen Rechner in "Zeitscheiben" bedient) kann der Benutzer sich mit BYE vom Rechner abmelden.

CALL

Der Befehl CALL(512) dient zum Ausführen eines Unterprogramms an der dezimalen Adresse 512 (hexadezimal hier 0200) und entspricht dem sonst vorhandenen Befehl SYS (s.d.).

CAT

CAT ist meist Bestandteil von Betriebssystemen für externe Speichermedien (z.B. Floppy-Disk) und gestattet es, einen Katalog (Catalogue) aller zur Verfügung stehenden Programme und Datenfiles abzurufen. Bei Floppy-Systemen wird hierbei einfach das auf jeder Diskette stehende Inhaltsverzeichnis auf den Bildschirm des Terminals oder Computers geschrieben.

CHAIN

CHAIN"PGR1" lädt das Programm mit dem Namen PGR1 von einem externen Speichermedium (z.B. Kassette) und startet es. Es entspricht damit dem Befehl RUN"PGR1" – mit einem wesentlichen Unterschied: CHAIN läßt sich als Befehl innerhalb eines Basic-Programmes verwenden, während RUN meist ein reiner Direktbefehl ist.

CHAIN ermöglicht damit, daß sich ein Basic-Programm selbst löscht, durch ein anderes ersetzt und selbsttätig startet. Damit ist es auch möglich, Programme laufen zu lassen, die nicht auf einmal in den Arbeitsspeicher des Computers passen. Die einzige Schwierigkeit ist dabei, daß außer dem alten Programm auch alle Variablen gelöscht werden, so daß sie vorher z.B. mit POKE in einen geschützten Speicherbereich transferiert werden müssen, wenn sie vom neuen Programm weiterbearbeitet werden sollen.

CHR$

Wenn man wissen will, welches Bildschirm- oder Druckerzeichen einem bestimmten dezimalen Wert (in ASCII) entspricht, so kann man das mit CHR$ feststellen: PRINT CHR$(65) liefert zum Beispiel den Buchstaben A. Eine sehr nützliche Anwendung des Befehls ist die Ausgabe von ASCII-Steuerzeichen, für die der Computer keine Tasten besitzt und die auch nicht auf dem Bildschirm dargestellt werden – etwa, um den Bildschirm zu löschen, die Schriftbreite eines Druckers fernzusteuern oder auch ein "Line-Feed"-Zeichen auszugeben, wie unser Beispiel zeigt:

```
10 PRINT#1,"ERSTE ZEILE";CHR$(10)
20 PRINT#1,"ZWEITE ZEILE"
```

Der Befehlsteil CHR$(10) erzeugt hier einen zusätzlichen Zeilenvorschub auf dem Drucker (File 1), so daß eine Leerzeile zwischen der ersten und zweiten Druckzeile entsteht.

CLEAR oder CLR

CLEAR dient als Direkt- oder Programmbefehl dazu, alle Variablen und die Rücksprungadressen von GOSUB- und FOR-NEXT-Befehlen zu löschen. Das Basic-Programm selbst wird davon nicht beeinträchtigt. In "Tiny Basic" dient CLEAR allerdings statt NEW zum Löschen des Programms.

CLOAD

Beim TRS-80 wird zum Laden eines Programms von der Kassette der Befehl CLOAD verwendet. Er ist in seiner Wirkung mit LOAD (s.d.) identisch. In Anführungszeichen kann ein Name folgen. Mit CLOAD? kann man die Qualität eines Programms überprüfen; dies entspricht dem VERIFY-Befehl.

CLOSE

Wenn man nach OPEN und INPUT# oder PRINT# eine Datei bearbeitet hat, muß sie mit CLOSE, gefolgt von der File-Nummer, wieder geschlossen werden, z.B. mit CLOSE1. Außer der File-Nummer benötigt CLOSE keine weiteren Parameter. Um z.B. beim PET-2001 eine File namens "TEST" vom internen Kassettenrecorder in die Stringvariable A$ zu laden, tut man folgendes:

```
10 OPEN1,1,0,"TEST"
20 INPUT#1,A$
30 CLOSE1
```

Nach CLOSE ist stets die gleiche Filenummer anzugeben, die auch beim vorhergehenden OPEN-Befehl und nach INPUT# verwendet wurde.

CLRDOT

CLRDOT X,Y löscht einen z.B. mit SETDOT auf dem Bildschirm gesetzten Grafikpunkt. X und Y müssen ganzzahlig sein.

CLS

Für das Löschen des Bildschirms ist beim TRS-80 ein eigener Befehl vorhanden, nämlich CLS. Beispiel:

```
10 CLS
```

Bei anderen Computern wird dieser Befehl durch eine Kombination von PRINT und CHR$ ersetzt.

CMD

Mit CMD ist es möglich, externe Geräte z.B. über den IEC-Bus anzusteuern. So kann man etwa beim PET-2001 statt der Bildschirmausgabe (z.B. bei LIST) auch einen Drucker ansteuern, wenn man den IEC-Bus per OPEN-Befehl als File 1 öffnet und dann CMD 1 eintippt. CMD gehört nicht zum Standard-Basic-Befehlsvorrat.

COLOR

Beim Apple-II vorhandener Befehl zum Definieren einer Farbe. COLOR=0 bedeutet schwarz, COLOR=15 ist weiß, und dazwischen liegen alle beim Apple erzeugbaren Mischfarben.

CONT

Mit dem Kommando CONT kann man nach einem STOP-Befehl oder nach einer Programmunterbrechung, die nicht wegen einer Fehlermeldung zustande kam (Break-Taste wurde gedrückt oder beim INPUT-Befehl wurde die Return-Taste ohne Argument gedrückt), das Programm fortsetzen. Nach einer Fehlermeldung führt CONT zu einem "Can't Continue Error".

COPY

Um irgendwelche Daten oder ein Programm von einem Speichermedium in ein anderes zu kopieren, wird COPY benutzt, zum Beispiel bei einem Dual-Floppy-Laufwerk von einer Diskette auf die andere, vom Computer-Bildschirm auf den Drucker oder von einer Kassette auf eine andere. Die Syntax ist sehr systemabhängig. Die meisten Kompaktcomputer verwenden statt COPY andere Basic-Befehle, z.B. CMD, SAVE, PRINT# usw.

COS

Mit COS kann der Cosinus eines meist im Bogenmaß anzugebenden Winkels berechnet werden. Das Argument kann eine beliebige positive oder negative Zahl oder numerische Variable sein und muß dem Befehlswort COS in Klammern folgen. Beispiel:

```
PRINT COS(3.14159)
-1
```

CSAVE

Das beim TRS-80 verwendete Befehlswort für das Abspeichern eines Programms auf Kassette lautet CSAVE. Es besitzt die gleiche Wirkung wie SAVE (s.d.) und kann von einem Programmnamen in Anführungszeichen ergänzt werden.

CUR

Mit CUR(Y,X) kann man die Schreibposition, d.h. den Cursor, auf Zeile Y und Zeichenposition X setzen. X und Y müssen ganzzahlig sein; beim ABC-80 darf X = 0...39 und Y = 0...23 betragen. Beispiel:

```
PRINT CUR(11,12);"X"
```

druckt ein X in die Mitte des Bildschirms.

DATA

Hinter DATA können einige vom Programm benötigte Werte

stehen, wobei mit READ diese Werte Variablen zugewiesen werden können (s. READ). Hinter DATA können auch Strings stehen, die nicht unbedingt in Anführungszeichen gesetzt werden müssen (das ist nur bei wenigen Computern erforderlich). Einige Beispiele:

```
10 DATA MEIER, MUELLER, HUBER
```
Die Leerräume werden nicht wirklich mitgespeichert

```
20 DATA" MEIER"," HUBER"
```
So lassen sich auch Leerräume am Anfang mitspeichern

```
30 DATA"MEIER,HUBER"
```
Jetzt wird auch das Komma gespeichert. MEIER,HUBER ist zusammen ein String

```
40 DATA 413, 456, 819
```
Natürlich eignet sich DATA auch für Zahlen

Die DATA-Werte können auch auf mehrere DATA-Anweisungen verteilt sein, wenn sie wegen ihrer Zahl nicht in eine Zeile passen.

DEF FN

"Define Function" – definiere eine Funktion – ist als Basic-Befehl oft ebenso nützlich wie die Verwendung von Unterprogrammen und hat gegenüber diesen den Vorteil, daß man bei einer mehrmals benötigten Rechenoperation nicht immer die gleichen Variablennamen verwenden muß.

Ein Beispiel: In einem Programm wird mehrmals die Funktion Cotangens gebraucht. Dieser Funktion kann man dann einen Namen zuordnen, der leider nicht COT lauten darf, sondern mit den Buchstaben FN beginnen muß und noch einen Variablennamen enthalten darf, z.B. FN C in unserem Fall. An den Programmanfang schreiben wir nun die aus vorhandenen Basic-Befehlen hergeleitete Cotangens-Funktion mit DEF:

```
10 DEF FN C(X)=1/TAN(X)
```

X ist hier eine "Dummy"-Variable, d.h. sie markiert nur den

Platz einer Variablen, kann aber später durch beliebige andere Variablennamen ersetzt werden. Und so könnte es aussehen, wenn wir der Variablen A den Arcustangens von B zuweisen wollen:

```
50 A=FN C(B)
```

Im gleichen Programm können wir die gleiche definierte Funktion auch mit ganz anderen Variablen verwenden. Einige wenige Basic-Interpreter lassen für solche definierte Funktionen auch Stringvariablen zu. Die Hauptanwendung von DEF FN ist aber in der Implementierung von mathematischen Funktionen zu sehen, die mehrmals in einem Programm für unterschiedliche numerische Variablen angewandt werden sollen.

DIM

Matrixvariablen wie z.B. A(I), B$(K) usw. können meist bis zu 256^2 Elemente enthalten, d.h. I bzw. K können Werte von 0...65535 annehmen. Da der Computer in seinem Speicher für die Werte dieser Variablen Raum reservieren muß, wäre es unsinnig, etwa für A(I) insgesamt 65536 Plätze freizuhalten, wenn z.B. nur zehn Elemente (I=0...9) in Frage kommen. Um wirtschaftlich mit dem Speicherplatz umzugehen, legt man deshalb die Anzahl der maximal zulässigen Elemente für jede Matrixvariable am Programmanfang (!) mit dem DIM-Befehl fest. Da dann der Speicher schon fest "vergeben" ist, darf dieses "Dimensionieren" nicht mehrmals im Programm erfolgen. Beispiele:

```
DIM A(10)
```
Dimensionieren der Matrixvariablen A für Indices von 0...9

```
DIM B$(3,5,10)
```
Dimensionieren der Stringmatrix B$ für die Matrixelemente B$(0,0,0) bis B$(2,4,9)

Man beachte, daß die Indices, d.h. die Werte in der Klammer hinter dem Variablennamen, beim DIM-Befehl um 1 größer sind als die tatsächlich zulässigen Indexzahlen, weil ja auch Null als Index erlaubt ist. Sind mehrere Matrizen zu dimensionieren, so können sie – durch Kommas getrennt – hinter einem gemeinsamen DIM-Befehl stehen, z.B.:

```
DIM A(10), B$(3,5,10)
```

Läßt man DIM am Programmanfang weg, so läßt der Computer meist 10 Elemente pro Matrixvariable zu (Index 0...9).

DO

DO ist Bestandteil der in größeren Computern zu findenden Anweisung IF...THEN...DO...ELSE (s.d.).

DOT

Mit DOT läßt sich prüfen, ob an einer bestimmten Bildschirmstelle ein grafischer Punkt gesetzt wurde (vgl. SETDOT und CLRDOT). Beispiel:

```
150 IF DOT(X,Y) THEN PRINT "PUNKT VORHANDEN":
    GOTO 170
160 PRINT"KEIN PUNKT VORHANDEN"
```

DRAW

Apple-II-Befehl zum Erstellen einer hochauflösenden Grafik; er hat die Form DRAW A AT B,C, wobei B und C die Startkoordinaten der zu erzeugenden Form und A der Index für die mit SHLOAD vorher definierte Form ist.

EDIT

Nach Ausführung des Befehls EDIT wird das Basic-Programm verlassen und ein Editor-Programm aufgerufen, mit dem es möglich ist, Texte zu verarbeiten oder ein im Speicher stehendes Programm mit komfortablen Korrekturmöglichkeiten zu ändern.

ELSE

ELSE dient zusammen mit IF...THEN...DO... zum Ausführen eines Befehls, der zu einer nach IF stehenden Bedingung als Alternative zu dem nach THEN DO stehenden Befehl ausgeführt werden soll. (Siehe auch IF...THEN...DO.)

END

Die END-Anweisung sorgt dafür, daß der Computer nach Abarbeitung des Basic-Programmes in den Direkt-Modus des Interpreters zurückspringt, so daß Befehle wie LIST, RUN usw. eingegeben werden können. Beim AIM-65 kann END dazu benützt werden, um zusätzliche Befehle zu implementieren.

Abgesehen von Tiny Basic ist es gewöhnlich nicht erforderlich, END ins Programm zu schreiben, außer wenn das Hauptprogramm keine Endlosschleife darstellt und von Unterprogrammen gefolgt wird. In diesem Falle läuft das Programm nämlich nach der Hauptprogramm-Abarbeitung in ein Unterprogramm hinein. Da dies ohne GOSUB-Befehl geschah, gibt der Computer eine Fehlermeldung aus, nämlich "Return without Gosub".

EQV

Mit EQV läßt sich eine Verknüpfung zweier Bedingungen realisieren, die wahr ist, wenn entweder beide Bedingungen wahr oder beide falsch sind. Damit stellt EQV genau das Gegenteil von XOR dar (s.d.). Ein Beispiel:

```
150 IF A=B EQV E=F THEN PRINT" BEIDE BEDIN-
    GUNGEN ERFÜLLT ODER BEIDE NICHT ER-
    FÜLLT":END
160 PRINT"NUR EINE VON BEIDEN BEDINGUNGEN
    IST ERFÜLLT"
```

ERRCODE

ERRCODE liefert bei Computern, deren Fehlermeldungen nicht als Klartext, sondern als Zahlencode ausgegeben werden, den Code des zuletzt aufgetretenen Fehlers. Dadurch ist es innerhalb des Basic-Programmes möglich, dem Benutzer einen Fehler im Klartext mitzuteilen, z.B.:

```
200 IF ERRCODE=7 THEN PRINT"ZAHL ZU GROSS"
```

ERL

Mit ERL kann z.B. beim TRS-80 (Level II) diejenige Zeilennummer ermittelt werden, in der zuletzt ein Fehler auftrat:

```
PRINT ERL
35
```

EXP

EXP dient als Basic-Befehl zur Errechnung der e-Funktion, d.h. das nach EXP in Klammern stehende Argument wird als Exponent der Zahl e=2,71828... betrachtet. Beispiel:

```
PRINT EXP(5)
148.413159
```

Das Ergebnis ist e^5.

FLASH

Beim Apple-II vorhandener Befehl, um die Zeichen auf dem Bildschirm blinken zu lassen.

FN

FN sind die beiden ersten Buchstaben eines aus drei Buchstaben bestehenden Funktionsnamens, dem am Programmanfang mittels DEF FN ein bestimmter arithmetischer Ausdruck zugewiesen wurde, z.B. zur Berechnung einer im Basic-Interpreter nicht vorhandenen Funktion. Dabei können andere Variablennamen als bei DEF FN verwendet werden, z.B. PRINT FNA(N). (Siehe DEF FN.)

FOR...TO...STEP...NEXT

Nehmen wir an, wir wollten das Wort TEST zehnmal auf den Bildschirm des Computers schreiben. Dies könnte einfach dadurch geschehen, daß wir zehnmal den Befehl PRINT"TEST" verwenden. Wenn wir TEST hundertmal schreiben wollen, wird das schon etwas mühsam. Eine Alternative wäre zum Beispiel:

```
10 I=0
20 I=I+1:PRINT"TEST"
30 IF I<11 GOTO 20
```

Außer in "Tiny Basic" läßt sich die Aufgabe aber noch einfacher lösen, nämlich mit einer FOR-NEXT-Schleife:

```
10 FOR I=1 TO 10:PRINT"TEST":NEXT I
```

Die Variable I bezeichnet man als "Laufvariable": Sie durchläuft in diesem Programm die Werte 1 bis 10 (inklusive des letzten Wertes 10, so daß hier auch genau 10mal TEST ausgedruckt wird). Die Laufvariable darf eine beliebige numerische Ganzzahl- oder Fließkommavariable sein, allerdings meist keine Matrixvariable wie I(J).

Der Befehl NEXT I erhöht die Laufvariable I um 1 und springt zur FOR-Programmzeile zurück, wenn I den letzten Wert noch nicht überschritten hat. Eine Alternative ist die Verwendung von STEP:

```
10 FOR I=52 TO 70 STEP 2:PRINT"TEST":NEXT I
```

Das Beispiel zeigt, daß wieder zehnmal TEST ausgedruckt wird, weil NEXT I nach STEP 2 bei jedem Durchlauf den Wert von I um 2 erhöht. Hinter STEP dürfen – wie hinter FOR und TO – positive oder negative Zahlen oder Variablen stehen, die nicht ganzzahlig sein müssen. In manchen Basic-Versionen wird statt STEP auch BY verwendet, was die gleiche Wirkung besitzt.

Bei der Verwendung von FOR-NEXT-Schleifen müssen einige wichtige Regeln beachtet werden, die nicht unbedingt selbstverständlich sind:

1. Die Schleife FOR I=5 TO 5 wird genau einmal ausgeführt.

2. Die Laufvariable kann und darf innerhalb der FOR-NEXT-Schleife nicht in einer Anweisung geändert werden. Kommt sie in einer Zuweisung vor, so muß sie stets auf der rechten Seite des Gleichheitszeichens stehen.

3. Es ist zulässig, z.B. mit GOTO aus der Schleife herauszuspringen, auch wenn die Laufvariable ihren Endwert noch nicht erreicht hat. Unzulässig ist es dagegen, von außen in die Schleife

hineinzuspringen, also in eine Programmzeile zwischen FOR und NEXT – sei es mit GOTO oder einer anderen FOR-NEXT-Schleife.

4. Es ist zulässig, statt NEXT I einfach NEXT zu schreiben, d.h. die Laufvariable nur bei FOR, nicht aber nach NEXT zu nennen. NEXT bearbeitet dann stets die zuletzt im Programm nach FOR genannte Laufvariable. Gefährlich wird das nur bei mehreren ineinander verschachtelten FOR-NEXT-Schleifen.

5. Die Schleife wird nicht ausgeführt, wenn der Endwert
a) gleich dem Anfangswert ist und STEP 0 verwendet wird;
b) kleiner als der Anfangswert ist und positive Schrittweiten verwendet werden;
c) größer als der Anfangswert ist und negative Schrittweiten verwendet werden.

FOR-NEXT-Schleifen benutzen den Stack, d.h. denjenigen Speicherbereich des Computers, der auch die Rücksprung-adressen bei Unterprogrammen zwischenspeichert. Da dieser Stack nur eine begrenzte Länge aufweist, ist auch die Anzahl der ineinander verschachtelten FOR-NEXT-Schleifen begrenzt (Größenordnung 10). Die Verschachtelung darf außerdem nicht gegen den obigen Punkt 3 verstoßen.

FRE

Um den im Computer derzeit noch unbenutzten Speicherplatz zu ermitteln, kann die Funktion FRE, gefolgt von einem "Dummy"-Argument in Klammern, verwendet werden. Dabei wird die dezimale Anzahl der noch freien Bytes ermittelt. Beispiel:

```
PRINT FRE(0)
3564
```

Ab und zu ist es auch möglich, die FRE-Anweisung zwischen den insgesamt freien Bytes und den freien Bytes unter Mißachtung der Strings differenzieren zu lassen:
FRE(0) liefert dann die freien Bytes, ohne Strings zu beachten;
FRE(A$) liefert die wirklich freien Bytes.
Diese Differenzierung ist aber bei den meisten Computern

nicht möglich. Normalerweise liefert FRE(0) dann die Anzahl der wirklich freien Bytes im System.

GET

Der GET-Befehl erlaubt in einem Programm, Informationen vom Tastenfeld zu holen, ohne daß die Return-Taste gedrückt werden muß.

GET holt genau ein Zeichen von der im Computer eingebauten Tastatur. Der Befehl ist nicht brauchbar, wenn die Tastatur nicht vom Computer selbst per Software abgefragt wird, sondern z.B. über eine TTY-Schnittstelle mit ihm verbunden ist – es sei denn, diese serielle Schnittstelle wird mit Hilfe einer Interrupt-Routine abgefragt, ohne das laufende Basic-Programm zu unterbrechen.

Der Computer hält mit der Programmausführung nicht an, wenn er auf einen GET-Befehl stößt, unabhängig davon, ob nun eine Taste gedrückt ist oder nicht. Der Programmierer hat also selbst dafür zu sorgen, daß der GET-Befehl genau dann ausgeführt wird, wenn der Benutzer eine Taste drückt. Dies läßt sich am einfachsten erreichen, wenn die Tastenabfrage in einer Programmschleife geschieht, die erst dann verlassen wird, wenn eine Taste gedrückt worden ist:

```
340 GET A$: IF A$="" GOTO 340
```

Das Programm bleibt solange in der Programmzeile 340, bis der Benutzer eine Taste drückt. Das der Taste entsprechende Zeichen ist jetzt in der Variablen A$ verfügbar.

Als Argument hinter GET kann jeder beliebige Variablentyp verwendet werden, also auch numerische oder Matrix-Variablen. Der Befehl kann auch dazu dienen, vom Benutzer einen Text oder eine Zahl abzufragen, deren Längen vorher bekannt sind. Das folgende Programmbeispiel holt ein vierstelliges Wort von der Tastatur, ohne daß die Return-Taste danach gedrückt werden muß, und druckt es anschließend aus:

```
10 GET A$: IF A$="" GOTO 10
20 B$=B$+A$: IF LEN(B$) < 4 GOTO 10
30 PRINT B$
```

Das Beispiel macht auch deutlich, daß bei GET das der gedrückten Taste entsprechende Zeichen – im Gegensatz zum Input-Befehl – nicht auf den Bildschirm geschrieben wird.

GIN

Der GIN-Befehl dient bei Tektronix-Rechnern zur Ermittlung der momentanen Cursor-Position auf dem Bildschirm und wird von zwei Variablen gefolgt, die dann die Koordinatenwerte übergeben. Beispiel: GIN X,Y liefert den horizontalen Wert in X und den vertikalen in Y.

GOSUB

Mit GOSUB werden Basic-Unterprogramme angesprungen. Unterprogramme sind immer dann sinnvoll, wenn ein bestimmter Programmabschnitt mehrmals im Hauptprogramm benötigt wird, zum Beispiel für die Errechnung komplexer mathematischer Funktionen (hier sollte man sich überlegen, ob nicht im individuellen Fall DEF FN günstiger ist).

GOSUB muß von einer existenten Basic-Zeilennummer gefolgt werden; nach einer ON-Anweisung dürfen auch mehrere, durch Kommas voneinander getrennte Zeilennummern verwendet werden (s. ON). Der Unterschied von GOSUB gegenüber GOTO ist, daß der Computer sich merkt, in welcher Zeile der Sprungbefehl stand. Dadurch kann er nach dem mit RETURN abgeschlossenen Unterprogramm in die dem GOSUB-Befehl folgende Zeile des Hauptprogramms zurückkehren.

GOTO

Der GOTO-Befehl gestattet die Fortsetzung eines Programmes an einer neuen Programmzeile. Er muß von einer positiven Zahl (meist 1...65535) gefolgt sein; Variable sind normalerweise nicht zulässig, ebenso nicht existente Zeilennummern. Hinter GOTO dürfen mehrere, voneinander durch Kommas getrennte Zeilennummern stehen, wenn eine ON-Anweisung vorausging (s.d.). Beispiele:

```
10 GOTO 50          Unbedingter Sprung
50 IF A=B GOTO 80   Bedingter Sprung
80 GOTO 80          Endlosschleife
```

Die Endlosschleife in Zeile 80 kann nur durch Drücken der BREAK- bzw. Stop-Taste am Computer verlassen werden, wobei dann "BREAK IN LINE 80" ausgedruckt wird.

GR

Apple-II-Befehl zum Umschalten des Bildschirms in den Grafik-Modus (mittlere Auflösung, 15 Farben möglich).

HGR

Apple-II-Befehl zum Umschalten des Bildschirms für hochauflösende Grafik mit vier Textzeilen am unteren Rand des Schirms. Im Gegensatz zu GR (s.d.) sind nur noch vier Farben möglich.

HIMEM

Mit HIMEM kann der Apple-II-Benutzer festlegen, bis zu welcher RAM-Adresse der Basic-Interpreter Variablen ablegen darf, z.B. HIMEM: 16384. Der Wert wird beim Einschalten automatisch auf den Normalwert gesetzt (wie auch LOMEM, s.d.).

HLIN

Dieser Befehl wird in der Form HLIN A,B AT C beim Apple-II verwendet, um eine Linie zwischen den Punkten mit den Koordinaten A,C und B,C im Grafik-Modus mittlerer Auflösung zu zeichnen. (S.a. VLIN).

HPLOT

Dieser Apple-II-Befehl entspricht dem SET-Befehl (s.d.), wobei im Modus für hochauflösende Grafik ein Punkt auf den Schirm geschrieben wird.

HOME

Manche Computer können mit HOME dazu angewiesen werden, den Cursor auf dem Bildschirm in die linke obere Ecke zu setzen. Bei den meisten Kompaktcomputern ist hierfür eine eigene Taste vorhanden, so daß der Befehl nicht ausgeschrieben werden muß.

IF...THEN...DO...ELSE

Prinzipiell dient IF zum Ausführen von Befehlen abhängig von bestimmten Bedingungen, z.B. abhängig vom Ergebnis des Vergleichs zweier Werte. Beispiele:

```
IF A=B THEN PRINT"BEIDE WERTE SIND GLEICH"
IF A=5 THEN GOTO 550
```

In zweiterem Beispiel kann bei den meisten Basic-Interpretern das THEN auch entfallen, d.h. die Form IF...GOTO ist zulässig. (Näheres über die Vergleiche findet sich unter "Operatoren".) Bei Verwendung von mehreren Befehlen pro Programmzeile, wobei der erste Befehl der Zeile eine IF-Anweisung ist, muß man besonders aufpassen. Wenn die nach IF stehende Bedingung nicht erfüllt ist, springt das Programm nicht etwa zum nächsten Befehl (der hier in der gleichen Zeile wie die IF-Anweisung steht), sondern zur nächsten Programmzeile:

```
50 IF A=B THEN PRINT"GLEICH":GOTO 90
70 PRINT"UNGLEICH"
90 END
```

Wenn A=B ist, wird GLEICH ausgedruckt und zu Zeile 90 gesprungen. Wenn A und B nicht gleich sind, druckt der Programmteil UNGLEICH aus.

Größere Computer gestatten komplexere Anweisungen nach IF, um es dem Anwender zu ermöglichen, ein mehrzeiliges Programm abzuarbeiten, wenn die Bedingung erfüllt ist, und es zu umgehen bzw. zu überspringen, wenn sie nicht erfüllt ist. Das kann etwa so aussehen:

```
50 IF A=B THEN DO
60 PRINT"A IST GLEICH B"
70 DO END
80 ELSE PRINT"A IST UNGLEICH B"
90 DO END
```

Die Befehle DO END dienen hierbei nur dazu, die jeweiligen Anweisungen abzuschließen.

Bei Computern, die ELSE und DO nicht verstehen, dürfen jene Anweisungen, die bei erfüllter IF-Bedingung ausgeführt werden sollen, nur in der Zeile stehen, in der auch IF steht. Diese Anweisungen müssen voneinander mit Doppelpunkten getrennt werden.

IMP

IMP ist eine etwas selten vorkommende Verknüpfung zwischen zwei Bedingungen. Hier gleich ein Programmbeispiel:

```
150 IF A=B IMP E=F THEN PRINT"A=B UND E
UNGLEICH F":END
```

In der Booleschen Algebra nennt sich diese Verknüpfungsart Implizierung. Die Bedingung ist erfüllt, wenn die erste Teilbedingung wahr und die zweite falsch ist.

INKEY

Dieser Befehl – z.B. beim TRS-80 verwendet – entspricht genau dem GET-Befehl (s.d.) und fragt die im Computer eingebaute Tastatur ab, ohne das Programm dabei anzuhalten.

INP

Bei Mikrocomputern, deren Ein- und Ausgabebausteine nicht mit einer 16-bit-Adresse "memory-mapped" angesprochen werden, sondern mit besonderen Steuerleitungen, wie etwa die meisten 8080- und Z-80-Systeme, stehen in Basic besondere Ein- und Ausgabebefehle zur Verfügung. Dies ist bei anderen Computern (AIM-65, PET-2001 usw.) nicht erforderlich, weil sich deren I/O-Adressen mit PEEK und POKE erreichen lassen.

Für 8080- und Z-80-Systeme hat man die Befehle INP und OUT eingeführt, die jeweils ein Byte vom oder zum I/O-Port, also 8 Bits, transferieren. Beispiel:

PRINT INP(6)	gibt den dezimal interpretierten Inhalt des Eingangs-Ports Nr. 6 aus.

INPUT

Der INPUT-Befehl erlaubt dem Benutzer die Eingabe von Variablenwerten. Dabei kann es sich um numerische oder String-Variable handeln. Die meisten Basic-Computer erlauben auch, einen Text in Anführungszeichen hinter INPUT anzugeben, der dann ausgedruckt wird, um den Benutzer des Programms zum Ein Eingeben der benötigten Variablenwerte aufzufordern. Beispiele:

INPUT A	Eingabe eines numerischen Wertes. Am Anfang der nächsten Zeile erscheint ein Fragezeichen, das zur Eingabe auffordert, die mit der Return-Taste beendet werden muß.
INPUT"WERT";A	Der Computer druckt WERT? aus und wartet auf eine Eingabe für den Wert von A.
INPUT"TEXT";A$	Nachdem TEXT? (oder ein beliebiger anderer Text) erscheint, kann man beliebige Zeichen eingeben, die in die Stringvariable A$ übernommen werden.

Drückt man sofort nach dem Erscheinen des Fragezeichens nur die Return-Taste, d.h. gibt man keinen Wert ein, so erfolgt oft ein Abbruch der Programmausführung; man gelangt in den Basic-Kommandomodus. Mit CONT kann man das Programm wieder fortsetzen, wobei man zur Eingabe zurückgelangt (es erscheint wieder ein Fragezeichen).

Es ist auch möglich, mit einem einzigen INPUT-Befehl nacheinander mehrere Variablenwerte einzulesen:

INPUT A, B	Zunächst erscheint ein Fragezeichen, und man kann einen Wert für A eingeben. Sobald man die Return-Taste gedrückt hat, erscheinen zwei weitere nebeneinanderstehende Fragezeichen für die Eingabe von B. Wenn man nach A statt Return ein Komma eingibt, kann man sofort B eintippen; dann erscheint kein weiteres Fragezeichen mehr.

Das Komma dient also zum Trennen mehrerer Eingabewerte beim INPUT-Befehl. Folgerichtig meldet der Computer z.B. bei INPUT A$, wenn GUTEN MORGEN, COMPUTER eingegeben wird: EXTRA IGNORED. Da ein Komma erkannt wurde, nimmt der Rechner an, daß zwei Eingabewerte vorliegen, obwohl nur nach einem gefragt wurde, und übernimmt nur GUTEN MORGEN in die Variable A$. Die Fehlermeldung EXTRA IGNORED ist eine von wenigen, die nicht zu einem Programmabbruch führen. Ebenso wie bei PRINT (s.d.) ist mit INPUT# das Holen von Daten von externen Geräten möglich. INPUT#1,A$ liest den Wert der Stringvariablen A$ von dem Gerät ein, dessen Filenummer vorher mit einem OPEN-Befehl (s.d.) vorher als 1 deklariert wurde. Der AIM-65 und der PC-100 kennen keinen INPUT#-Befehl. Das Problem läßt sich aber lösen, wenn die Speicherzelle hex A412 (INFLG) auf das gewünschte Gerät gesetzt wird (TTY, Kassette usw.). Ein dafür notwendiges kleines Programm ist im Abschnitt "AIM-65 und PC-100" beschrieben.

Wenn bei einem Input-Befehl Anführungszeichen verwendet werden, so kann deren Bedeutung unterschiedlich sein. Steht bei der Eingabe eines Strings der gesamte Text in Anführungszeichen, so darf er auch Kommas enthalten; die Anführungszeichen werden jedoch dann nicht in die Stringvariable übernommen. Stehen die Anführungszeichen dagegen mitten im Textstring, so werden sie mitgespeichert. Leerräume am Stringbeginn werden nur zwischen Anführungszeichen gespeichert. Vorsicht ist auch bei der Verwendung von Doppelpunkten geboten. Bei manchen Computern werden sie wie Kommas als

Trennungszeichen behandelt, was zur Ausgabe von "EXTRA IGNORED" und zum Abtrennen eines Stringteils führt, z.B. beim AIM-65.

Das Problem, daß Strings beim INPUT-Befehl bestimmte Zeichen nicht enthalten dürfen, löst der manchmal vorhandene Befehl INPUTLINE. Er liest eine komplette Bildschirmzeile bis zum Drücken der Return-Taste in eine Variable ein (z.B. beim ABC-80).

INSTR

Manche Basic-Interpreter verfügen über den zuweilen recht nützlichen Befehl INSTR (Instring), mit dem man innerhalb eines längeren Strings nach einem kurzen String suchen kann:

```
P=INSTR(B,A$,S$)
```

liefert in P die Position des Suchbegriffs S$ in A$. Gesucht wird ab der Position B in A$. Das Ergebnis (hier P) bezieht sich auf das erste übereinstimmende Zeichen. P ist Null, wenn S$ nicht in A$ gefunden wurde.

Die Wirkung des Befehls geht aus folgendem Beispiel hervor:

```
10 INPUT A$                    Eingabe eines Textes
20 PRINT INSTR(7,A$," ")       Suche nach einem Leer-
RUN                            raum nach dem 7. Zeichen
                               im Text.
? DIES IST EIN TEST            Der Leerraum wird als 9.
9                              Zeichen gefunden.
```

INT

Mit INT läßt sich der Nachkommaanteil einer Fließkommazahl abschneiden, so daß sich eine ganze Zahl ergibt:

```
PRINT INT(34.25)
34
```

Das ist natürlich noch keine Rundung zur nächsten ganzen Zahl. Dafür wäre folgendes Mini-Programm notwendig:

```
10 INPUT"ZAHL";Z
20 R=INT(Z+0.5)
30 PRINT"GERUNDET:";R
```

Durch vorhergehende Multiplikation mit 100 und nachfolgende Division durch 100 läßt sich z.B. auf zwei Nachkommastellen runden:

```
10 INPUT Z:Z=Z*100
20 R=INT(Z+0.5):R=R/100
30 PRINT R
```

INVERSE

Beim Apple-II vorhandener Befehl, um Zeichen auf dem Bildschirm invers, d.h. schwarz auf weißem Hintergrund, darzustellen.

KILL

Das als Programm- und Direktbefehl verwendbare KILL löscht eine bestimmte Datei oder ein Programm auf der Floppy-Disk. Beim ABC-80 muß hierbei außer dem Namen der Datei innerhalb der Anführungszeichen – von einem Punkt abgetrennt – auch der Dateityp angegeben werden, z.B. BAS für ein Basic-Programm:

```
KILL"PRGR1.BAS"
```

LEFT$

Mit LEFT$ kann man einen String auf eine bestimmte Anzahl von auf seiner linken Seite stehenden Zeichen begrenzen. Der zu bearbeitende String und, durch ein Komma getrennt, die Anzahl der gewünschten Zeichen sind in Klammern hinter LEFT$ zu setzen:

```
10 INPUT A$
20 PRINT LEFT$(A$,3):GOTO 10
RUN
? BASIC
BAS
```

Die Stringvariable A$ selbst bleibt dabei unverändert. Noch ein paar Beispiele:

```
10 A$=LEFT$(F$,K)
50 Z$(I)=LEFT$(Z$(I),1)
70 C$=LEFT$(A$+B$,2)
```

Die gewünschte Länge des Resultats muß größer als Null und meist kleiner als 256 sein, andernfalls tritt ein "Function Error" auf. Ist die Stringvariable im Argument ebensolang oder kürzer als die gewünschte Länge, so bleibt sie unverändert.

LEN

Um die Länge eines Strings zu ermitteln, kann man LEN verwenden:

```
10 INPUT A$
20 L=LEN(A$)
30 PRINT"DER TEXT IST";L;"ZEICHEN LANG"
```

LEN ist auch sehr nützlich, um festzustellen, ob ein String "unsichtbare" ASCII-Steuerzeichen enthält, die man auf dem Computer-Bildschirm nicht zu sehen bekommt. Dann liefert nämlich PRINT LEN(A$) eine größere Zahl, als man durch Abzählen der nach PRINT A$ ausgegebenen Zeichen ermittelt. Normalerweise kann LEN Werte bis 255 erreichen – so lang dürfen Strings bei den meisten Kompaktcomputern sein.

LET

LET steht im Standard-Basic vor einer Zuweisung, z.B.

```
10 LET A=5
20 LET B$="TEST"
```

Bei den meisten Basic-Interpretern kann LET weggelassen werden, da der Computer selbständig erkennt, ob es sich bei dem Gleichheitszeichen "=" um einen Vergleich (z.B. nach IF) oder um eine Zuweisung handelt.(Siehe auch "Operatoren".)

LIST

Der Direktbefehl LIST ermöglicht es, das gesamte Programm, das sich momentan im Arbeitsspeicher befindet, oder auch Teile davon aufzulisten. Ein paar Beispiele machen deutlich, welche Möglichkeiten es hierbei gibt.

LIST	listet das gesamte Basic-Programm auf
LIST 100-	listet alle Zeilen ab 100 auf
LIST 40-80	listet alle Zeilen von 40 bis 80 auf
LIST -100	listet alle Zeilen von 1 bis 100 auf
LIST 40,80	gleichbedeutend mit LIST 40-80

In diesem Zusammenhang sei noch erwähnt, daß auf zahlreichen Computern die Zeilennummer Null nicht zulässig ist, da sie ebenso abgespeichert würde wie das vom Rechner intern verwendete Kennzeichen für das Programmende.

LOAD

Mit LOAD kann man ein Programm von einem externen Speichermedium wieder einlesen, das vorher mit SAVE aufgezeichnet wurde. Außer bei AIM-65 und PC-100 wird dabei zunächst ein eventuell schon im Arbeitsspeicher stehendes Programm gelöscht, und SAVE darf von einem in Anführungszeichen stehenden Programmnamen gefolgt sein, nach dem dann auf der Kassette oder Diskette gesucht wird. LOAD ist gewöhnlich nur als Direktbefehl verwendbar. Beim AIM-65 und PC-100 entspricht der LOAD-Befehl eher dem sonst üblichen MERGE, da das neue Programm mit dem alten gemischt werden kann, und der Programmname wird erst nach dem Drücken der Return-Taste eingetippt.

LOG

Mit LOG errechnen Basic-Computer den natürlichen Logarithmus des in Klammern danach angegebenen Arguments, d.h. den Logarithmus zur Basis e. Beispiel:

```
PRINT LOG(5)
1.60944
```

Die Befehlsform darf nicht mit der bei Taschenrechnern üblichen Errechnung des dekadischen Logarithmus (LOG-Taste) verwechselt werden. Der natürliche Logarithmus entspricht der Taschenrechner-Taste LN.

LOMEM

Dieser Apple-II-Befehl erlaubt es dem Programmierer, den für Variablen zur Verfügung stehenden Speicherplatz an beliebiger Stelle im RAM beginnen zu lassen, z.B. LOMEM;2051.

MEM

MEM ist beim TRS-80 (Level I) ein direkter Ersatz für FRE(0). Dabei wird die Anzahl der noch freien Bytes im Arbeitsspeicher des Computers ermittelt. Beispiel:

```
PRINT MEM
```

Dabei ist zu bedenken, daß niemals die wirklich zur Verfügung stehende Zahl von freien Bytes ausgegeben werden kann, weil ja MEM selbst schon Speicherplatz benötigt.

MERGE

Wenn ein Programm mit LOAD von der Kassette oder einem anderen externen Speichermedium geladen wird, so wird automatisch das vorher im Arbeitsspeicher stehende Basic-Programm gelöscht (Ausnahme: AIM-65/PC-100). Mit MERGE ist es nun möglich, ohne diese Löschung ein Programm zu laden und mit dem bereits vorhandenen zu mischen. Wenn bestimmte Zeilennummern allerdings sowohl schon gespeichert sind als auch von außen geladen werden, so werden die gespeicherten Programmzeilen überschrieben. Üblicherweise kann nach MERGE wie nach LOAD ein Programmanme angegeben werden. Beispiel:

```
MERGE"PGR1"
```

MID$

In der Klammer hinter MID$ können entweder zwei oder drei Argumente stehen, wovon das erste stets ein String bzw. eine

Stringvariable ist:

MID$(A$,I) liefert den rechten Teil des Strings A$, beginnend beim I-ten Zeichen (von links gezählt)

MID$(A$,I,J) liefert den mittleren Teil des Strings A$, beginnend beim I-ten Zeichen und mit der Länge J

Beispiele:

```
PRINT MID$("BASIC",2)
ASIC
PRINT MID$("BASIC",2,2)
SI
A$=MID$(F$,K,I)
```

Die angegebene Startposition (hier K) und die Länge (hier I) müssen größer als Null und meist kleiner als 255 sein, andernfalls tritt ein "Function Error" auf. Überschreitet die gewünschte Länge die Länge des ursprünglichen Strings, so wird er von der Startposition K bis zu seinem letzten Zeichen ausgegeben. Wenn die Startposition K größer ist als die Länge des ursprünglichen Strings, so ist das Ergebnis ein "Leerstring", d.h. es enthält kein Zeichen.

NAME

Mit NAME kann man ein auf der Diskette stehendes Programm umbenennen, ohne es zu ändern. Das dabei verwendete Format ist:

```
NAME"Alter Name"AS"Neuer Name"
```

Dieser Befehl läßt sich nicht nur als Direktbefehl, sondern auch innerhalb eines Programmes anwenden.

NEW

NEW – bei manchen Computern auch SCR – löscht das gesamte im Arbeitsspeicher (RAM) stehende Basic-Programm und alle Variablen. Ferner werden eventuell geöffnete Files (z.B.)

(z.B. Kassettenrecorder-Dateien) geschlossen. NEW ist außer als Direktbefehl oft auch als Programmbefehl verwendbar – ein Programm kann sich also auch selbst löschen.

NORMAL

Das Video-Display des Apple-II kann mit NORMAL in den normalen Wiedergabe-Modus geschaltet werden, d.h. kein Blinken und nicht invers.

NOT

NOT dient zur binären Negation, d.h. eine Zahl kann bitweise invertiert werden. Die Wirkung von NOT ist von der internen Zahlendarstellung des Basic-Interpreters abhängig. Bei AIM-65 und PET-2001 liefert PRINT NOT 0 den Wert –1. Eine Anwendung sind z.B. bedingte Sprünge:

```
10 INPUT A
20 IF NOT A=5 THEN PRINT"NEIN"
```

Hier wird NEIN ausgedruckt, wenn A ungleich 5 ist. Die Zeile 20 könnte natürlich ebensogut lauten:

```
20 IF A<>5 THEN PRINT"NEIN"
```

NOTRACE

Dieser Befehl bewirkt das Ausschalten des Trace-Modus, der vorher mit TRACE eingeschaltet wurde (s.d.). Beispiel:

```
730 NOTRACE
```

Auch eine Verwendung als Direktbefehl ist möglich, zum Beispiel nach einem Abbruch der Programmausführung mit der Break- bzw. Stop-Taste.

NUM

Manche Computer enthalten zusätzlich zu VAL den Befehl MUM. Wenn das der Fall ist, so sind hinter VAL nur Ziffern, nicht aber Vorzeichen zulässig. Anders bei NUM: Hier dürfen im Argument – stets ein String – auch Vorzeichen vorkom-

men. Wie VAL wird NUM dazu verwendet, den numerischen Wert eines aus Ziffern und u.U. Vorzeichen bestehenden Strings zu ermitteln. Ein Beispiel:

```
10 LET B=NUM("-3.532"):PRINT B
RUN
-3.532
```

ON

ON dient zusammen mit GOTO oder GOSUB zur Realisation einer Mehrfachverzweigung. Der Befehl

```
10 ON K GOTO 40,60,55,90
```

sprint zur Zeile 40, wenn K=1 ist, zur Zeile 60, wenn K=2 ist usw. Wenn K nicht ganzzahlig ist, wird der Dezimalteil ignoriert. Statt der Variablen nach ON ist auch ein beliebiger arithmetischer Ausdruck zulässig.

ON ERROR GOTO

Computer, die Fehlermeldungen nicht als Text ausgeben, sondern in Form von Fehlercode-Zahlen, gestatten oft die Fehlerbehandlung innerhalb des Basic-Programmes, ohne daß dieses abgebrochen wird. Ein Beispiel hierfür ist die Fehlermeldung bei einer Division durch Null:

```
40 C=B/A
50 ON ERROR GOTO 200
60 PRINT C:END
200 PRINT"DIVISION DURCH NULL":END
```

Will man herausfinden, um welche Art von Fehler es sich gehandelt hat, so kann man dieses mit ERRCODE tun (s.d.).

OPEN

OPEN dient zum Öffnen einer Datei. Dabei kann folgende Befehlsform verwendet werden:

```
OPEN I,J,K,"NAME"
```

I ist die File-Nummer (Dateinummer) und darf üblicherweise 0...255 betragen; J dient als Gerätenummer, und K bestimmt, ob gelesen oder geschrieben werden soll. Der in Anführungszeichen stehende NAME wird z.B. bei einer Kassetten-Datei mit aufgezeichnet, um die Datei später wiederfinden und identifizieren zu können.

Beim PET-2001 gelten z.B. folgende Vereinbarungen: Maximal 10 Files können gleichzeitig offen sein. Die Gerätenummern werden wie folgt vergeben: 0 = Tastatur, 1 = eingebaute Kassette, 2 = externe Kassette, 3 = Bildschirm, 4...15 = IEC-Bus. K ist beim Lesen Null und beim Schreiben 1 oder 2. Bei K=2 wird an die Datei ein zusätzliches End-of-Tape-Zeichen (ASCII hex 04) angehängt. Wenn nicht alle Parameter genannt werden, z.B. bei OPEN1, so setzt der PET J=1 und K=0, so daß von der internen Kassette gelesen wird. Beim verwendeten Filenamen muß man aufpassen, daß er – wenn man von der Kassette liest – nicht innerhalb eines längeren anderen Namens vorkommt, da dann u.U. die falsche Datei geladen wird, z.B. diejenige mit dem Namen "PET-PRGR", wenn nach "PRGR" gesucht wird. Vor einem INPUT#-Befehl muß die entsprechende Datei stets mit OPEN geöffnet worden sein, ebenso vor PRINT#.

OPEN...AS FILE

Nicht alle Rechner verfügen über einen IEC-Bus. Daher kann und braucht beim Öffnen einer Datei keine Gerätenummer angegeben werden. Beim ABC-80 wird zudem eine andere Syntax für die Datei-Eröffnung verwendet, nämlich:

```
OPEN"Dateiname"AS FILE N
```

N ist dabei die Dateinummer. Statt des ausgeschriebenen Dateinamens darf (wie beim OPEN-Befehl des PET-2001) auch eine Stringvariable verwendet werden. Zu beachten ist, daß mit OPEN AS FILE beim ABC-80 eine Datei nur zum Lesen geöffnet werden kann; der entsprechende Befehl für das Schreiben lautet PREPARE (s.d.).

Operatoren

„+"

Das Plus-Zeichen dient der Addition zweier beliebiger Fließkomma- oder Integer-Zahlen, dem Zusammenfügen von zwei Strings (Vorsicht: der sich ergebende String darf meist nicht länger als 255 Zeichen sein!) oder – in der Booleschen Algebra – als Zeichen für das logische "Oder" (gleichwertig mit OR in Basic). Beispiele:

Befehlsfolge	Ergebnis
PRINT 255+15.5	270.5
PRINT"TEST"+ "STRING"	TESTSTRING
INPUT A,B:IF(A=1)+ (B=2) THEN PRINT "RICHTIG"	RICHTIG wird ausgedruckt, wenn A=1 oder B=2 eingegeben wird. "+" kann hier durch OR ersetzt werden.

„–"

Mit dem Minus-Zeichen kann man zwei Zahlen bzw. Variablen voneinander subtrahieren. Für Strings und in der Booleschen Algebra ist "–" nicht verwendbar.

„*"

Der Stern entspricht dem Multiplikationszeichen. Er läßt sich zum Multiplizieren von Zahlen und Variablen sowie als "Und"-Verknüpfung einsetzen. Beispiele:

Befehlsfolge	Ergebnis
PRINT 10*23.7	237
IF(A=1)*(B=2)GOTO300	Das Programm sprint zur Zeile 300, wenn A=1 und B=2 ist. (Wenn man den Stern durch AND ersetzt, können die Klammern entfallen.)

„/"

Als Divisionszeichen wird in Basic meist der Querstrich verwendet. Er gestattet nur das Dividieren von Zahlen und numerischen Variablen und läßt sich weder für Strings noch in der Booleschen Algebra einsetzen.

„="

Das Gleichheitszeichen kann mehrere Bedeutungen haben, die aus folgenden Beispielen hervorgehen.

Befehlsfolge	Ergebnis	Erklärung
A=1: PRINT A	1	„=" dient hier als Zuweisung
A=A+5	A wird um 5 erhöht	Hier wird der Unterschied der Zuweisung gegenüber der Algebra deutlich!
IF A=5 GOTO 300	Programmverzweigung zu Zeile 300, wenn A=5 ist	„=" dient hier als Vergleich
INPUT A: PRINT A=5	Ausdruck von 0, wenn A=5, sonst +/−1	Der Ausdruck A=5 ist Null, wenn der Vergleich "unwahr" ist

Auf der linken Seite des Gleichheitszeichens muß bei Zuweisungen immer eine Variable stehen; die Zuweisung 1=A führt zu einer Fehlermeldung. Der Vergleich IF 5=A ... ist dagegen zulässig. Natürlich kann man mit „=" auch Stringvariablen einen Text zuweisen, z.B. A$="STRINGTEXT". Auch hier muß links aber immer eine Variable stehen; Ausdrücke wie MID$(A$,4,1)= "F" sind also – obwohl formal richtig – unzulässig!

„↑"

Auf Computerbildschirmen gibt es normalerweise keine Möglichkeit, einen Exponenten hochzustellen, z.B. die 5 bei 2^5. Stattdessen schreibt man deshalb 2↑5. Beispiel:

```
PRINT 2↑10
1024
```

Der ABC-80 verwendet „**" statt „↑".

>

Das "Größer-als"-Zeichen dient in Basic zum Vergleich zweier Werte. Dabei kann es sich um numerische, bei vielen Basic-Interpretern aber auch um Stringvariable handeln. Zusammen mit IF

ergibt sich eine bedingte Anweisung. Zwei Beispiele machen dies deutlich:

```
10 INPUT A
20 IF A>5 THEN PRINT"A IST GROESSER ALS 5":
   GOTO 40
30 PRINT"A IST KLEINER ODER GLEICH 5"
40 END
```

Das zweite Beispiel arbeitet mit String-Vergleichen:

```
10 REM ALPHABETISCHER VERGLEICH
20 INPUT"ERSTE ZEILE";A$
30 INPUT"ZWEITE ZEILE";B$
40 IF B$>A$ GOTO 60
50 C$=A$:A$=B$:B$=C$:REM VERTAUSCHEN
60 PRINT A$:PRINT B$
```

Die Variablen A$ und B$ werden hier in alphabetischer Reihenfolge geordnet und ausgegeben. Der Vergleich erstreckt sich hier auf die volle Stringlänge, wobei A$ und B$ auch unterschiedlich lang sein dürfen.

<

Der Umgang mit dem "Kleiner-als"-Zeichen ähnelt dem Zeichen ">", hat eben nur die umgekehrte Bedeutung. Es braucht hier deshalb nicht noch einmal erläutert werden.

<>

Die beiden hintereinandergestellten Vergleichsoperatoren drükken "ungleich" aus. Wieder wird aus einem kleinen Beispiel klar, wie das zusammen mit IF funktioniert:

```
10 A=RND (1)*10+1:A=INT(A)
20 INPUT"IHRE SCHAETZUNG";B
30 IF B<>A THEN PRINT"FALSCH GESCHAETZT":
   GOTO 20
40 PRINT"GRATULATION!"
```

Hier handelt es sich um ein Mini-Programm, bei dem der Benutzer eine Zahl zwischen 1 und 10 erraten muß.

<= und >=

Außer dem Ungleich-Symbol gibt es noch zwei weitere Mischungen einzelner Vergleichszeichen. Ihre Bedeutung ergibt sich eigentlich aus der Schreibweise; das Gleichheitszeichen schließt im Gegensatz zu den Einzelzeichen die exakte Gleichheit ein. Solche Grenzwert-Überlegungen spielen eine große Rolle in bedingten Schleifen u.a.

OR

Die OR-Verknüpfung ist ein logisches "Oder" zwischen zwei Entscheidungen oder Entscheidungsgruppen. Die Bedingung A=1 OR B=5 ist bereits erfüllt, wenn eine der beiden Teilbedingungen erfüllt ist. Verknüpft man mit OR zwei Zahlen, so werden ihre Bits paarweise Oder-verknüpft. 01010101 OR 10101010 (binär) liefert so zum Beispiel 11111111; dezimal wäre das: 85 OR 170 ergibt 255. Die Beispiele der AND-Verknüpfung lassen sich äquivalent auf OR anwenden.

OUT

OUT entspricht dem Gegenteil von INP (s.d.) und existiert nur bei solchen Mikrocomputern als Befehl, die Ein- und Ausgangsbausteine mit besonderen Steuerleitungen, also nicht "memory-mapped" ansprechen. Beispiel:

OUT 6,64 gibt die Dezimalzahl 64 als Byte an den Ausgangsport Nr. 6 aus.

Mikrocomputer mit Memory-mapped-I/O benötigen keinen OUT-Befehl, da sich alle Ausgänge mit POKE ansprechen lassen.

PDL

Der Apple-II verfügt über Analog-Eingänge für "Paddles", Spiel-Steuerknüppel also. A=PDL(5) gibt der Variablen A den Wert, der dem Stand von Paddle 5 entspricht.

PEEK

PEEK(A) liefert das als Dezimalzahl (0...255) interpretierte Byte an der dezimalen Adresse A. Damit ist es möglich, sozusagen in den Computerspeicher direkt "hineinzusehen". Beispiele:

```
10 PRINT PEEK(65535)
20 A=PEEK(256)
```

PI

Nicht alle Computer haben den Wert der Kreiszahl π fest gespeichert. Jene, die diesen Komfort bieten, gestatten den Abruf der Zahl mit einer besonderen Taste π oder aber mit dem Ausdruck PI. Beispiel:

```
PRINT PI
3.141592654
```

Sollte ein Computer nicht über den Wert für π verfügen und braucht man diesen häufig im Programm, so ist es sinnvoll, am Programmanfang zu schreiben:

```
10 PI=3.141592654
```

Dann kann die Kreiszahl genauso wie z.B. beim ABC-80 abgerufen werden, der sie fest gespeichert enthält. Die einmalige PI-Definition am Programmanfang ist hinsichtlich der Rechengeschwindigkeit und des Speicherplatzbedarfs günstiger, als mehrmals den Zahlenwert zu schreiben.

PLOT

Der Befehl PLOT ist beim Apple-II zum Setzen eines Bildpunktes im Grafik-Modus vorhanden und entspricht dem Set-Befehl (s.d.). Die Farbe des Punkts wird durch einen vorher angewendeten COLOR-Befehl festgelegt.

POINT

POINT dient zur Abfrage, ob ein Grafik-Punkt z.B. mit SET an- oder mit RESET ausgeschaltet wurde. Beispiel:

```
400 IF POINT(X,Y)=1 THEN PRINT"PUNKT AUF DEN
    KOORDINATEN ";X;",";Y
```

Der Wert von POINT ist entweder 0 oder 1. Ein ähnlicher Befehl ist DOT (s.d.).

POKE

Mit POKE lassen sich bestimmte Speicherzellen vom Basic-Programm her neu belegen. POKE(A,D) transferiert die Daten D (0...255) an die Adresse A (0...65535). A und D müssen dezimal angegeben werden. Bei der Verwendung dieses Befehls ist größte Vorsicht geboten; überschreibt man nämlich versehentlich Speicherzellen, in denen Teile des Basic-Programms oder Variablenwerte stehen, so kann u.U. das ganze System zusammenbrechen, so daß man es aus- und wieder einschalten muß. Es ist vor der Anwendung von POKE daher erforderlich, anhand der Speicherbelegung (s. Kap. 3) herauszufinden, wo man Daten gefahrlos hinspeichern darf.

POP

POP korrigiert den "Stackpointer", indem eine Rücksprungadresse nach GOSUB vom Stack geholt wird, ohne den Rücksprung auszuführen. Dies ist sinnvoll, wenn man z.B. von einem Unterprogramm direkt in eine bestimmte Zeile des Hauptprogramms springen möchte.

POS

POS benötigt ein Blindargument in Klammern (ähnlich wie FRE) und ermittelt die momentane Cursorposition (vgl. GIN). Beispiel:

```
PRINT CUR(0)    druckt eine Dezimalzahl aus,
                die der Position entspricht, an der sich
                der Cursor vor diesem Befehl befand.
```

PREPARE

Dieser Befehl ähnelt dem OPEN-Befehl und wird z.B. beim ABC-80 verwendet. Hier gleich ein Beispiel:

PREPARE "PRGR1" AS FILE 1

eröffnet die Datei namens PRGR1 mit der Dateinummer 1.

Der PREPARE-Befehl dient ausschließlich dazu, eine Datei zum Schreiben vorzubereiten. Zum Lesen dient auch beim ABC-80 der Befehl OPEN. Da dieser Rechner nicht über einen IEC-Bus verfügt, braucht und kann keine Gerätenummer angegeben werden.

PRINT

PRINT dient – als Befehl oder Kommando – zur Ausgabe beliebiger Zahlen, Texte oder Zeichen. Einige Beispiele:

PRINT	Löst einen Zeilenvorschub aus (beim AIM-65 und PC-100 ist dazu PRINT" " erforderlich)
PRINT"BASIC"	Druckt BASIC aus und geht dann in eine neue Zeile
PRINT"BASIC";	Wie oben, der Cursor (d.h. die Schreibposition) bleibt aber in der gleichen Zeile, hier nach dem "C"
PRINT A, B, C$	Druckt die Variablenwerte von A, B und C$ tabellarisch in eine Zeile
PRINT A; B; C$	Druckt die Werte von A, B und C$ in eine Zeile, läßt jedoch keinen Abstand dazwischen. Bei positiven Werten von A und B ergibt sich nur jeweils ein Leerraum an der Stelle des Vorzeichens.
PRINT CHR$(10);	Druckt ein Line-Feed-Zeichen (ASCII dezimal 10), z.B. für einen Zeilenvorschub beim Drucker

Das nach PRINT verwendete Argument darf auch ein beliebiger arithmetischer Ausdruck sein, zum Beispiel PRINT 234+A/9.

Beim AIM-65 bestimmt die Speicherzelle hex A413 ("OUTFLG"), welches Ausgabegerät von einem PRINT-Befehl angesprochen wird, z.B. Display, Drucker oder auch Kassette.

Bei anderen Computern gibt es für den Fall, daß die Ausgabe auf externe Geräte gewünscht wird, den PRINT#-Befehl:

PRINT#1, A, B, C$ Ausgabe der Variablen A, B, C$ auf das Gerät, dem im OPEN-Befehl die File-Nummer 1 zugeordnet wurde (s. OPEN)

Normalerweise ist es mit dem PRINT-Befehl nicht möglich, doppelte Anführungszeichen (") auszugeben. Eine Möglichkeit hierfür ist jedoch die Verwendung von PRINT CHR$(34). Beim Anschalten eines externen Druckers an den Computer ist von großer Bedeutung, ob beim Zeilenwechsel vor oder nach PRINT außer dem Return-Zeichen (ASCII dez. 13) auch ein Line-Feed-Zeichen generiert wird; notfalls muß dieses per Programm hinzugefügt werden, wie oben gezeigt.

Beim PET-2001 können in den Anführungszeichen nach einem PRINT auch besondere Steuerzeichen z.B. für Cursorbewegungen oder für das Löschen des Bildschirms stehen. Sie werden zwischen zwei Anführungszeichen nicht ausgeführt, sondern zur Kontrolle als inverse Grafikzeichen auf den Bildschirm geschrieben. Deshalb ist es hier nicht möglich, nach PRINT" einen falsch eingegebenen Text zu korrigieren, indem man mit dem Cursor zurückgeht. Eine Alternative zum völligen Neuschreiben der Zeile ist es dann, nach dem Schreiben eines fehlerhaften Zeichens wieder ein Anführungszeichen einzugeben, jetzt mit dem Cursor zurückzugehen, die Korrektur auszuführen und das nur zu Korrekturzwecken eingegebene zweite Anführungszeichen dann einfach wieder zu überschreiben, um es nach dem auszugebenden Text an seine Sollposition zu setzen. (Das gleiche Verfahren ist auch bei Schreibfehlern in dem einem INPUT-Befehl folgenden Text anzuwenden.) Der PRINT-Befehl läßt sich durch USING (s.d.) auf ein bestimmtes Anzeigeformat festlegen.

Beim TRS-80 kann mit PRINT@ an der aktuellen Cursorposition weiter geschrieben werden, ohne – wie sonst üblich – eine neue Zeile zu beginnen. LPRINT entspricht beim TRS-80 dem PRINT-Befehl, ist aber auf den angeschlossenen Drucker bezogen.

RANDOMIZE

Die von einem Software-Zufallsgenerator (s. RND) erzeugten Zahlen sind eigentlich nicht ganz zufällig, da sie ja einem festgelegten Algorithmus entstammen. RANDOMIZE dient nun dazu, aus irgendwelchen im System zufällig vorhandenen Parametern (Uhrzeit, bestimmte Speicherinhalte usw.) einen Anfangswert für die RND-Funktion zu ermitteln. Würde das nicht geschehen, so würde die RND-Funktion nach jedem Programmstart mit RUN immer die gleiche Folge von "Zufallszahlen" liefern, weil sich jede Zahl aus der vorhergehenden durch den Zufallsalgorithmus ergibt.

READ

Mit READ können die in DATA-Anweisungen definierten Werte oder Strings Variablen zugewiesen werden. Dies geschieht immer in der Reihenfolge, wie die Werte und Strings hinter den DATA-Anweisungen stehen. Ein Beispiel:

```
10 DATA MEIER, MUELLER, HUBER, SCHULZ
20 FOR I=1 TO 4
30 READ A$: PRINT A$,:NEXT I
RUN
MEIER MUELLER HUBER SCHULZ
```

Will man im gleichen Programm die hinter DATA stehenden Werte mehrmals verwenden,so gibt es zwei Möglichkeiten. Einmal ist es möglich, den internen Zähler für die DATA-Werte mit dem RESTORE-Befehl rückzusetzen, so daß er hier nach RESTORE wieder auf MEIER zeigt. Zum anderen – und das ist eine übliche Methode – kann man die DATA-Werte zunächst in eine Matrixvariable einlesen:

```
10 DATA MEIER, MUELLER, HUBER, SCHULZ
20 FOR I=0 TO 3
30 READ A$(I):NEXT I
```

Jetzt kann man jeden einzelnen DATA-Wert beliebig oft aufrufen und verwenden, indem man statt READ nun die entsprechende Matrixvariable benutzt. So liefert PRINT A$(2)

zum Beispiel den Ausdruck HUBER. Selbstverständlich ist die Verwendung von numerischen DATA-Werten und Variablen bei READ ebenso möglich.

Versucht man, mehr Werte mit READ einzulesen, als in DATA-Anweisungen definiert sind, so meldet der Computer "OUT OF DATA ERROR"

REM

REM gestattet das Einfügen von Kommentaren in das Programm, die später nicht ausgedruckt werden, sondern ausschließlich dazu dienen, bei später eventuell notwendigen Programmänderungen dem Benutzer die Übersicht zu erleichtern. Einige Beispiele:

```
10 REM BEISPIEL
20 A=A+B:REM AUFSUMMIEREN
30 REM ENDE
```

REM-Anweisungen werden im Programm vom Computer überlesen. Der ihnen folgende Text darf alle Zeichen enthalten, auch Doppelpunkte. Die Form 10 REM ANFANG:PRINT"TEST" ist unzulässig, da das Wort TEST niemals ausgedruckt würde. Der REM-Text kann also nur durch Drücken der Return-Taste des Computers abgeschlossen werden.

REN (oder RENUMBER)

Wie schon erwähnt, schreibt man Basic-Programme gewöhnlich mit Zeilennummern-Abständen von 5 oder 10, um nachträglich im Notfall noch zusätzliche Zeilen einfügen zu können. Allerdings kann man dabei sehr schnell an die natürliche Grenze gelangen: Bei 10er-Abständen lassen sich nur 9 Zeilen einfügen. Abhilfe schafft hier der REN-Befehl. Er numeriert alle Programmzeilen neu mit einer Schrittweite von z.B. 10 und korrigiert dabei, damit das Programm auch wieder laufen kann, alle Sprünge (GOTO, GOSUB usw.). Die Befehlsform ist, soweit dieser Befehl überhaupt vorhanden ist, recht unterschiedlich. Beim Tektronix-Basic bewirkt REN eine Neunumerierung, die bei der alten Programmzeile 100 beginnt, mit der Schrittweite 10 und der ersten Zeilennummer von 100. REN 2000, 5, 90

beginnt bei der alten Programmzeile 90, bewirkt eine Neunumerierung mit einer Schrittweite von 5 und setzt die erste Zeilennummer auf 2000. Beim ABC-80 von Luxor dient REN zur Neunumerierung des gesamten Programms mit einer Schrittweite von 10, und REN 150,20 ergibt ein Programm, das bei 150 beginnt und in Schritten von 20 numeriert ist. REN ist nur als Direktbefehl (Kommando) verwendbar.

RESET

Dieser beim TRS-80 benutzte Befehl hat die gleiche Bedeutung wie CLRDOT (s.d.). RESET (X,Y) löscht einen z.B. mit SET gesetzten Grafikpunkt auf dem Bildschirm. (Im Gegensatz zum Basic-Befehl RESET besitzt ein an manchen Computern zu findender Knopf mit der Bezeichnung "Reset" eine gänzlich andere Wirkung: Der Programm- und Variablenspeicher des Rechners wird u.U. gelöscht.)

RESTORE

Wendet man den READ-Befehl an, so werden unter DATA abgelegte Werte in der gleichen Reihenfolge in Variablen eingelesen, wie sie in den Programmzeilen stehen. Mit jedem READ wird der "DATA-Zeiger" deshalb um 1 erhöht.

Will man die DATA-Werte mehrmals im Programm lesen, so muß man diesen Zeiger irgendwie auf den ersten Wert zurücksetzen können, so daß man wieder von vorn beginnen kann. Dies geschieht mit dem Befehl RESTORE. Auf Variablen und Programm hat er sonst keinen Einfluß. Beispiel:

```
10 DATA MEIER, HUBER, MUELLER
20 FOR I=1 TO 3:READ A$:PRINT A$:NEXT
50 RESTORE
60 FOR I=1 TO 3:READ A$:PRINT A$,:NEXT
```

Dieses kleine Programm druckt die drei Namen einmal untereinander und einmal nebeneinander. Um den zweiten Lesevorgang zu ermöglichen, setzt RESTORE in Zeile 50 den DATA-Zeiger auf MEIER zurück.

RESUME

Der nur im Programm verwendbare RESUME-Befehl kann am Ende einer Fehlerbehandlungs-Routine (nach ON ERR GOTO...) verwendet werden, um das Programm in der nächsten Zeile fortzusetzen.

RETURN

RETURN muß am Ende jeden Unterprogrammes stehen. Der Befehl holt sich die von GOSUB auf dem "Stack" des Rechners abgelegte Rücksprungadresse und springt in die dem GOSUB-Befehl (s.d.) folgende Programmzeile. RETURN ohne GOSUB führt zu einer Fehlermeldung.

RIGHT$

RIGHT$ hat eine ähnliche Wirkung wie LEFT$ (s.d.), wirkt aber auf den rechten Teil des Strings im Argument:

```
10 INPUT A$
20 PRINT RIGHT$(A$,3):GOTO10
RUN
? BASIC
SIC
? COMPUTER
TER
```

Die gewünschte Resultatlänge muß größer als Null und meist kleiner als 256 sein. Die Stringvariable im Argument wird unverändert übernommen, wenn sie kürzer oder ebenso lang wie die gewünschte Länge des Resultats ist.

RND

Mit RND ist es möglich, Zufallszahlen zwischen 0 und 1 zu erzeugen (gleichmäßige Verteilung von 0...1). Die Befehlsform von RND ist nicht einheitlich. Bei Computern, die kein Argument hinter RND benötigen, ist meist der RANDOMIZE-Befehl vorhanden (s.d.), der irgendeinen Anfangswert für den Software-Zufallsgenerator ermittelt, damit nicht jedesmal die gleiche Zahlenfolge mit RND entsteht:

```
10 RANDOMIZE                    Anfangswert ermitteln
20 FOR I=1TO4:PRINT RND:NEXT I  Vier Zufalls-
                                werte ausgeben
```

Andere Computer benötigen ein Argument in Klammern hinter RND, das selbst einen Anfangswert darstellt. Sinnvollerweise verwendet man hierfür irgendeine Variable, die von vorherigen (nicht konstanten) Benutzereingaben belegt wurde:

```
20 PRINT RND(A)
```

Die entstehenden Zahlen zwischen 0 und 1 lassen sich leicht in andere Bereiche umrechnen. Ein Würfel-Programm, das Zahlen von 1...6 liefert, sieht z.B. so aus:

```
10 INPUT"WIEVIELE ZAHLEN";N
20 FOR I=1 TO N: PRINT INT(6*RND(N)+1):NEXT
```

ROT

Die Anweisung ROT=A dreht eine in der hochauflösenden Apple-II-Grafik wiedergegebene Form, wobei A=16 zum Beispiel 90° im Uhrzeigersinn bedeutet.

RUN

Mit dem Direktbefehl RUN kann man ein im Arbeitsspeicher stehendes Programm an seiner ersten Zeile starten. Läßt man nach RUN eine Zeilennummer folgen, so beginnt die Ausführung erst an der entsprechenden Programmzeile. Manche Computer (z.B. der ABC-80) gestatten es auch, nach RUN einen Programmnamen zu schreiben: RUN"PRGR1" lädt das Programm mit dem Namen PRGR1 zunächst von der Kassette und startet es automatisch, was der Befehlsfolge LOAD/RUN entspricht.

Es ist nicht zulässig, nach RUN eine Zeilennummer anzugeben, die zu einem Unterprogramm gehört. RUN löscht alle Variablen, setzt den DATA-Zähler, den Unterprogramm-Rücksprungspeicher (Stack) und alle aktiven FOR-NEXT-Schleifen zurück. Will man dies vermeiden, so muß man den Befehl CONT verwenden (s.d.).

SAVE

Mit SAVE kann man ein Basic-Programm in voller Länge, jedoch ohne seine Variablenwerte, auf ein externes Speichermedium retten, z.B. auf eine Tonbandkassette oder auf eine Diskette. Oft darf dabei unmittelbar nach SAVE in Anführungszeichen ein Programmname stehen, der mit abgespeichert wird und später beim Wiedereinlesen zur eindeutigen Identifikation des Programms dient. SAVE ist gewöhnlich nur als Direktbefehl verwendbar. Bei AIM-65 und PC-100 muß man vor dem Gebrauch dieses Befehls den Hex-Wert 20 in die Zelle A409 speichern, um ein fehlerfreies Wiedereinlesen zu ermöglichen.

SCALE

SCALE=A gibt z.B. beim Apple-II an, welcher Maßstab beim DRAW-Befehl zur Wiedergabe hochauflösender Grafiken verwendet werden soll.

SCR

Der von "Scratch" abgeleitete Direktbefehl SCR hat die gleiche Wirkung wie NEW: Er löscht das im Arbeitsspeicher stehende Basic-Programm und alle Variablen. Ferner werden eventuell geöffnete Files (z.B. Kassettenrecorder-Dateien) geschlossen.

SCRN

Der Befehl A=SCRN(B,C) liest die Farbe des Bildpunktes mit den Koordinaten B,C in die Variable A ein und wird u.a. bei der mittleren Grafik-Auflösung des Apple-II verwendet.

SET oder SETDOT

SET X,Y bzw. SETDOT X,Y setzt auf dem Bildschirm im Grafik-Modus einen Punkt mit den Koordinaten X und Y. Der Befehl eignet sich daher ideal zum Plotten, d.h. zum Schreiben von Kurven oder Grafiken. X und Y müssen meist ganzzahlig sein bzw. werden gerundet. Der gesetzte Punkt kann mit CLRDOT X,Y wieder gelöscht werden. Bei manchen Computern

dient SET allerdings dazu, den Winkelfunktionen-Modus z.B. von Grad auf Bogenmaß umzuschalten.

SGN

Die Funktion PRINT (SGN(A) liefert folgendes Ergebnis:

0, wenn A Null ist;
1, wenn A positiv ist;
−1, wenn A negativ ist.

Bei Computern, die die SGN-Funktion nicht besitzen, läßt sie sich durch A/ABS(A) ersetzen, wenn A nicht Null ist.

SHLOAD

Dieser Befehl dient beim Apple-II bzw. ITT-2020 zum Laden einer hochauflösenden Grafik von der Kassette in den Arbeitsspeicher.

SIN

Die Sinus-Funktion errechnet den Sinus (−1...+1) eines meist im Bogenmaß (3.1415 statt 180°) anzugebenden Winkels, der hinter SIN in Klammern stehen muß. Ein kleines Beispiel:

```
10 FOR K=0 TO 3.12159 STEP 0.2
20 PRINT K,SIN(K):NEXT K
```

Dieses Programm liefert eine Sinus-Wertetabelle für 0...180°

SPACE oder SPC

Mit diesem Befehl kann eine bestimmte Anzahl von Leerräumen erzeugt werden (Spaces). Das Argument hinter SPC bzw. SPACE darf eine ganze Zahl oder eine ganzzahlige numerische Variable sein. PRINT SPC(5);"*" druckt einen Stern in die sechste Zeichenposition der nächsten Zeile. Es fällt eine gewisse Ähnlichkeit zu TAB auf; der Unterschied ist, daß TAB sich stets auf den linken Zeilenrand bezieht, SPC hingegen auf die zuletzt gedruckte Zeichenposition. Das wird aus folgendem Beispiel deutlich:

```
10 PRINT TAB(3);"A";TAB(5);"B"
20 PRINT SPC(3);"A";SPC(5);"B"
RUN
...A.B.....
...A.....B...
```

Die Punkte dienen hier nur zur Kenntlichmachung der erzeugten Leerräume und werden nicht wirklich ausgedruckt.

SPEED

Mit SPEED können z.B. Apple-II-Benutzer festlegen, mit welcher Geschwindigkeit Zeichen ausgegeben werden sollen. SPEED läßt sich wie eine Variable behandeln; die höchste Geschwindigkeit wird mit SPEED=255 erreicht.

SQR

Da die Tastaturen von Basic-Computern über kein spezielles Wurzelzeichen verfügen, verwendet man zur Quadratwurzelberechnung SQR, gefolgt von einem in Klammern stehenden positiven Argument:

```
PRINT SQR(2)
1.41421356
```

Negative Argumente führen zu einer Fehlermeldung des Computers. Zur Errechnung der Kubikwurzel oder höherwertiger Wurzeln kann man sich eines inversen Exponenten bedienen, z.B. liefert B = A↑(1/3)die Kubikwurzel der (positiven oder negativen) Zahl A.

ST

Mit PRINT ST läßt sich beim PET-2001 abfragen, ob beim Laden eines Programms ein Fehler auftrat; wenn nein, liefert PRINT ST den Wert Null. ST ist die Abkürzung von "Status".

STEP

Mit STEP kann man nach FOR...TO... die Schrittweite der

Laufvariablen bestimmen. Das hinter STEP stehende Argument kann eine beliebige Zahl oder numerische Variable sein; Klammern sind nicht nötig (siehe FOR...).

STOP

STOP hat weitgehend die gleiche Wirkung wie END. Allerdings gibt der Computer "BREAK IN LINE..." aus, also die Zeilennummer des STOP-Befehls. Außerdem ist es möglich, mit CONT das Programm von Hand wieder zu starten. Ein Demonstrationsbeispiel:

```
10 FOR I=1 TO 1000:NEXT     Verzögerungsschleife
20 STOP                     Programmstop
30 PRINT"FERTIG"            Test-Text
RUN
BREAK IN 20                 Abbruch beim STOP-Befehl
CONT                        Manuelle Fortsetzung
FERTIG                      Zeile 30 wird bearbeitet
```

STR$

Normalerweise kann man einer numerischen Variable keine Stringvariable zuweisen und umgekehrt – außer mit STR$ und VAL (s.d.). STR$(N) liefert einen aus Ziffern und Vorzeichen bestehenden String, der der numerischen Variablen N entspricht:

```
10 LET A$=STR$(N)
```

STRING$

Der z.B. beim ABC-80 vorhandene Befehl STRING$(A,B) liefert eine Kette von A Zeichen des ASCII-Dezimalcodes B. Um zum Beispiel zehnmal den Buchstaben B auszudrucken, schreibt man:

```
PRINT STRING$(10,66)
```

Es handelt sich dabei also um eine Art "multiplizierten" CHR$-Befehl.

SWAP%

Dieser u.a. beim ABC-80 vorhandene Befehl wirkt auf eine aus zwei Bytes bestehende ganze Zahl (16 Bits). Dabei werden die beiden Bytes vertauscht:

```
10 SWAP%(A%)
```

Die Prozentzeichen dienen lediglich zur Kennzeichnung, daß hier mit ganzen Zahlen gearbeitet wird, die nur zwei Bytes belegen.

SYS

Nicht immer ist Basic die ideale Programmiersprache, besonders bei zeitkritischen Problemen. In diesem Falle wird man Teile des Programmes in der prozessor-spezifischen Maschinensprache ausführen und diese von Basic her aufrufen. Wenn dabei keine Variablen übergeben werden sollen, kann dies am einfachsten mit SYS geschehen, wobei in Klammern eine dezimale Adresse folgen muß, die die Startadresse des Maschinenprogrammes angibt. Der Rücksprung von der Maschinenroutine in das Basic-Programm erfolgt durch einen Befehl wie RTS (z.B. 6502) bzw. RET (8080).

TAB

Mit TAB(I) kann man in einer Zeile um I Zeichen nach rechts weitergehen. Beispiel:

```
PRINT TAB(19);"X"
```

druckt ein X 19 Zeichenquadrate vom linken Bildschirmrand entfernt.

Man beachte den Strichpunkt vor "X"; er sorgt dafür, daß das X sofort an die sich ergebende Cursorposition und nicht erst in die nächste Zeile oder – wenn man den Strichpunkt durch ein Komma ersetzt – an die im Rechner vorgesehene Standard-Tabellenposition gedruckt wird.

TAN

TAN liefert den Tangens einer in Klammern folgenden und im Bogenmaß anzugebenden Zahl. Beispiel:

```
10 INPUT"WINKEL";W              Eingabe in Grad
20 B=W/180*3.14159              Umrechnung ins Bodenmaß
30 PRINT"TANGENS=";TAN(B)       Ausgabe des Tangens-Wertes
```

TEXT

Dieser Befehl dient beim Apple-II zum Rückschalten aus dem Grafik- in den Textanzeige-Modus des Bildschirms. Der Cursor geht dabei in die letzte Zeile.

THEN

THEN muß nach einer IF-Bedingung stehen, um bestimmte Befehle ausführen zu können, wenn diese Bedingung erfüllt ist. Die meisten Basic-Computer erlauben das Weglassen von THEN, wenn nach IF ein GOTO-Befehl folgt. (Siehe IF...THEN...ELSE ...DO.)

TRACE

TRACE kann als Kommando oder als Basic-Befehl verwendet werden, um während der Programmausführung ab der gerade durchlaufenen Zeile in den Trace-Modus zu gehen. Dabei werden alle Zeilennummern, die von Programm gerade bearbeitet werden, zur Kontrolle auf den Bildschirm oder Drucker ausgegeben. Beispiel:

```
420 TRACE
```

Das Abschalten des Trace-Modus ist mit NOTRACE möglich.

TROFF

Mit TROFF kann der Trace-Modus abgeschaltet werden. Der Befehl entspricht in seiner Wirkung NOTRACE (s.d.).

TRON

Einschalten des Tracing-Modus. Dieser Befehl wird beim TRS-80 statt TRACE verwendet und hat die gleiche Wirkung (s.d.).

UNSAVE

Der Direktbefehl UNSAVE dient z.B. beim ABC-80 dazu, eine Datei bzw. ein Programm auf der Diskette zu löschen. Der Name der Datei ist hinter UNSAVE ohne Anführungszeichen zu schreiben. Beispiel:

```
UNSAVE PRGR1
```

USING

Dieser Ausdruck wird zusammen mit PRINT dazu benutzt, um Strings zusammen mit Zahlen auszudrucken, wobei ein Format entsteht, das nicht (wie sonst) von der Zahlenlänge, sondern nur von einer vorhergehenden Stringanweisung bestimmt wird. Ein Beispiel:

```
10 I=2.15
20 PRINT USING "####.##";I;"MAL ZWEI ERGIBT"
30 I=I*2
40 IF I < 999 GOTO 20
RUN
   2.15 MAL ZWEI ERGIBT
   4.30 MAL ZWEI ERGIBT
   8.60 MAL ZWEI ERGIBT
  17.20 MAL ZWEI ERGIBT usw. usw.

550.40 MAL ZWEI ERGIBT
READY
```

Mit USING kann man also recht bequem eine Tabelle mit unterschiedlich langen Zahlen im Festkommaformat sauber formatiert ausgeben.

USR

Wie mit SYS können auch mit USR Maschinenprogramme aufgerufen werden, die sich bei zeitkritischen Problemen besser als

Basic-Routinen eignen. Im Gegensatz zu SYS ist das in Klammern folgende Argument bei USR aber nicht die Adresse der Maschinenroutine, sondern irgendeine Zahl oder Variable, die von dieser Routine bearbeitet werden soll. Die Startadresse des Maschinenprogramms muß vorher mit POKE-Befehlen an bestimmte (systemspezifische) Adressen gespeichert werden. Es wird empfohlen, wegen der großen Unterschiede der einzelnen Computer bei Befehlen wie SYS und USR das mitgelieferte Handbuch zu konsultieren.

VAL

Der Versuch, einer Stringvariablen eine Zahl bzw. eine numerische Variable zuzuweisen, führt normalerweise zu der Fehlermeldung "TYPE MISMATCH". Mit VAL kann dieses Problem umgangen werden, da diese Funktion einen aus Ziffern und Vorzeichen bestehenden String in die entsprechende numerische Variable umwandelt. Das kann ausgenutzt werden, um zusammen mit STR$ (s.d.) eine numerische Variable z.B. auf drei Stellen zu begrenzen und die höherwertigen Stellen abzuschneiden:

```
10 INPUT N
20 A$=STR$(N)
30 A$=RIGHT$(A$,3)
40 N=VAL(A$):PRINT N
```

VARPTR

Die Werte von Variablen werden von Basic-Interpreter an bestimmte Adressen im RAM des Mikrocomputers abgespeichert; numerische Variablen meist binär bzw. hexadezimal codiert und String-Variablen im ASCII-Format. Mit VARPTR läßt sich feststellen, wo die Anfangsadresse einer Variablen steht:

PRINT VARPTR (C)	druckt diejenige Adresse dezimal, an der der Wert der Variablen C abgespeichert ist.

VLIN

Während HLIN beim Apple-II eine horizontale Linie zeichnet, dient VLIN dazu, eine vertikale Linie auf den Schirm zu bringen.

VERIFY

Wenn man sich nicht sicher ist, ob das Programm, das man gerade auf Kassette abgespeichert hat, lesbar oder identisch mit dem noch im Arbeitsspeicher befindlichen ist, kann das mit VERIFY"NAME" überprüft werden. NAME ist dabei der Programmname, mit dem das Programm auf Kassette aufgezeichnet wurde. Dieser Befehl ist normalerweise nur als Direktbefehl verwendbar.

WAIT

WAIT ist eine elegante Methode, ein Basic-Programm anzuhalten, bis z.B. ein externes Gerät über eine I/O-Leitung gemeldet hat, daß es mit irgendetwas fertig ist.

WAIT(A,M,Z) hält das Programm an, bis die Daten an der dezimalen Adresse A den Zustand Z erreicht haben. Dabei werden nur die mit M maskierten Bits ausgewertet.

Wenn nur zwei Argumente hinter WAIT stehen, also Z weggelassen wird, so wird Z als Null angenommen, so daß gewartet wird, bis die mit M gekennzeichneten interessierenden Bits Null geworden sind. WAIT ist im allgemeinen nur auf Computern zu finden, deren Ein- und Ausgangsports "memory-mapped" ausgeführt sind, d.h. wir normale Speicherzellen adressiert werden. Die Adresse A darf normalerweise 0...65535 und die übrigen zwei Argumente 0...255 betragen. Selbstverständlich dürfen numerische Variablen als Argument ebenso Verwendung finden wie Zahlen.

Da alle Argumente in ihrer dezimalen Form angewendet werden müssen, ist es erforderlich, die jeweiligen Bitmuster mittels der Zweierpotenzen $2^0 ... 2^8$ in Dezimalzahlen umzu-

rechnen. Hierzu finden sich Beispiele in Kapitel 4 unter "Hexadezimale Zahlen". Manche Rechner warten nach WAIT auch auf einen Interrupt.

XDRAW

Dieser Apple-II-Befehl hat fast die gleiche Wirkung wie DRAW (s.d.), als Farbe wird jedoch das Komplement der vorher vorhandenen verwendet.

XOR

Exklusiv-Oder-Verknüpfung zweier numerischer Variablen oder Zahlen, auch in Vergleichsanweisungen, z.B.:

```
150 IF A=B XOR E=F GOTO 170
160 PRINT"ENTWEDER SIND BEIDE AUSSAGEN
    FALSCH ODER BEIDE RICHTIG":END
170 PRINT"ENTWEDER IST A=B ODER E=F, ABER
    NICHT BEIDES"
```

XOR kann auch dazu benützt werden, um einzelne Bits in einem Byte gezielt zu invertieren, z.B. das niederwertigste an der dezimalen Adresse 515:

```
A=PEEK(515) XOR 1: POKE(515,A)
```

Die Zahl 1 entspricht 2^0 und "maskiert" daher das niederwertigste Bit des Ergebnisses von POKE(515).

6 Über das Schreiben von Programmen

6.1 Von der Problemstellung zum Flußdiagramm

Um ein Programmierproblem optimal lösen zu können, darf man mit der Denkarbeit nicht erst dann beginnen, sobald das halbe Programm schon geschrieben ist, wenn man eine solche Vorgehensweise manchen fertigen Programmen ab und zu auch allzu deutlich ansieht.

Gewöhnlich liegt die Problemstellung in einer ziemlich unüberschaubaren, ja chaotischen Form vor. Wenn wir uns einmal das in Kapitel 7 besprochene Mischprodukte-Programm ansehen, so fällt es schwer, die Dinge, die man mit ihm anfangen kann, in irgendeine logische Sequenz zu bringen. Genau das ist aber stets der erste Schritt beim Programmieren: Das Problem muß in eine zeitliche Folge von kleinen Schritten zerlegt werden. Der Computer kann ja nicht mehrere Dinge gleichzeitig tun! Eine wichtige Hilfe leistet uns dabei ein Flußdiagramm. Es besteht aus Symbolen für unterschiedliche Tätigkeitsbereiche – für Ein- und Ausgabeoperationen, logische Entscheidungen und andere Anweisungen. Verzweigungen und Programmschleifen können dabei sehr übersichtlich dargestellt werden *(Abb. 6.1.1)*.

Anhand eines Beispiels wird deutlich, wie wichtig es ist, zunächst einen groben Ablauf vorzugeben. Es soll eine Methode gefunden werden, den Wochentag eines beliebigen Datums festzustellen. Dabei soll der Gregorianische Kalender als Grundlage dienen.

Seine Regeln lauten – kurz gesagt – wie folgt: Die durch vier teilbaren Jahreszahlen sind Schaltjahre. Die durch 100 teilbaren Jahre sind keine Schaltjahre, die durch 400 teilbaren aber doch. 1600, 2000, 2400 sind also Schaltjahre, 1700, 1800, 1900 jedoch nicht. Ein Jahr hat rund 365,25 Tage.

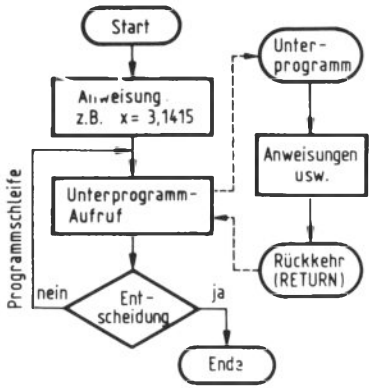


Abb. 6.1.1 Typischer Aufbau eines Flußdiagrammes. Kommt eine bestimmte Befehlsfolge mehrmals vor, so schreibt man sie zweckmäßig als Unterprogramm. Dann läßt sie sich vom Hauptprogramm aus beliebig oft aufrufen

Diese Regeln in einer einzigen Formel auszudrücken und daraus z.B. die Wochentags-Nummer von 1...7 zu errechnen, ist ziemlich aussichtslos. Man muß die Problemstellung also in eine logische, zeitliche Folge von Anweisungen und Entscheidungen umformulieren, deren Ergebnis *Abb. 6.1.2* zeigt.

6.2 Vom Flußdiagramm zum Basic-Programm

Das Flußdiagramm ist noch weitgehend davon unabhängig, in welcher Programmiersprache später das jeweilige Problem gelöst werden soll, wenn es auch nur solche Anweisungen enthalten kann, die sich irgendwie in der zur Verfügung stehenden Sprache ausdrücken lassen. Prinzipiell läßt sich aus ein und demselben Flußdiagramm ein Assembler-, ein Basic-, ein Pascal- oder auch ein Fortran-Programm machen.

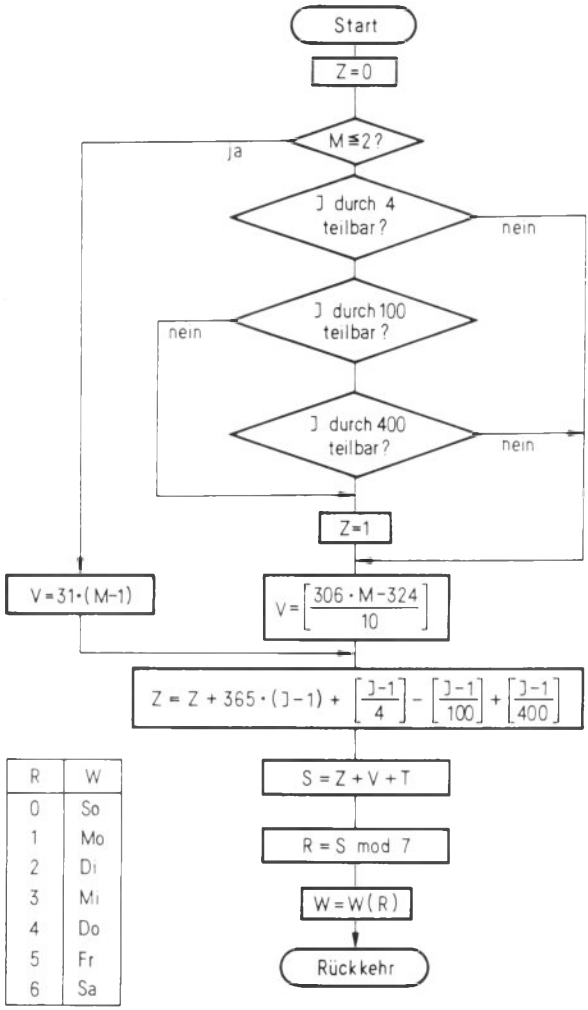


Abb. 6.1.2 Flußdiagramm zur Bestimmung des Wochentags eines beliebigen Datums

```
Ø11Ø INPUT T,M,J
Ø12Ø Z=Ø
Ø122 IF M<=2 THEN 296
Ø13Ø IF J/4<>INT(J/4) THEN 292
Ø132 IF J/1ØØ<>INT(J/1ØØ) THEN 29Ø
Ø134 IF J/4ØØ<>INT(J/4ØØ) THEN 292
Ø29Ø Z=1
Ø292 V=INT((3Ø6*M-324)/1Ø)
Ø294 GO TO 3ØØ
Ø296 V=(M-1)*31
Ø3ØØ Z=Z+(J-1)*365+INT((J-1)/4)
Ø3Ø1 Z=Z-INT((J-1)/1ØØ)+INT((J-1)/4ØØ)
Ø34Ø S=Z+V+T
Ø35Ø R=S-INT(S/7)*7
Ø37Ø FOR I=1TO7
Ø38Ø READ W$(I)
Ø39Ø NEXT I
Ø4ØØ PRINT W$(R+1)
Ø51Ø DATA SONNTAG,MONTAG,DIENSTAG,MITTWOCH
     DONNERSTAG,FREITAG,SAMSTAG
```

Abb. 6.2 Aus dem Flußdiagramm entstandenes Basic-Programm. Nach dem Start mit RUN muß man das Datum im Format Tag, Monat, Jahr eingeben. Nach "Return" wird der Wochentag ausgegeben

Wir wollen uns hier natürlich darum kümmern, wie wir ein Basic-Programm erstellen können, da diese Sprache bei den preiswerteren Kompaktcomputern am verbreitetsten ist.

Ein Basic-Programm ist grundsätzlich in Zeilen organisiert. Diese Zeilen können nach Belieben von Null (manchmal ab 1) bis maximal 65535 (manchmal 32768) numeriert werden. Nun sind sogar Programmierer keine Genies, und es kann schon vorkommen, daß man ein Programm später mit zusätzlichen Zeilen erweitern muß. Um nun das Einfügen solcher zusätzlicher Zei-

len zu gestatten, läßt man die Zeilennummern nicht einfach aufeinanderfolgen, sondern verwendet Abstände von z.B. 5 oder 10.

Abb. 6.2 zeigt ein aus dem gerade erwähnten Flußdiagramm entstandenes Basic-Programm. Es macht auch deutlich, daß man für die Variablen möglichst sinnvolle Namen verwenden sollte, um das Verständnis des Ablaufs zu erleichtern, z.B. T für Tag, M für Monat, J für Jahr usw. Durch Hinzufügen von komfortableren Eingaberoutinen (z.B. INPUT"TAG";T usw.) kann unser Programm schließlich noch benutzerfreundlicher gestaltet werden, so daß man keine schriftliche Bedienungsanleitung dazu benötigt.

Immerhin liegt uns jetzt aber bereits ein lauffähiges Programm vor. Sinnvollerweise tragen wir, damit wir uns nach einigen Monaten auch noch darin auskennen, die wichtigsten Zeilennummern in das vorher erstellte Flußdiagramm ein, so daß der Ablauf auf einen Blick sichtbar wird.

6.3 Fehlermeldungen

Eine der nützlichsten Eigenschaften von Basic-Computern ist es, bestimmte Fehler selbständig zu erkennen. Dabei handelt es sich natürlich nie um logische Fehler im Programmablauf, sondern ausschließlich um Fehler in der Formulierung von Befehlen und Ausdrucken, in der Überschreitung der Speicher- oder Rechenkapazität usw.

Schon ein Blick in den Fehlerkatalog des Computer-Handbuches zeigt, welche unterschiedlichen Fehler ("Errors") erkannt werden können.

Die meisten Computer erkennen Fehler erst, wenn das Programm mit RUN gestartet wurde. Wenige andere, z.B. ABC-80 und HP-85, erkennen manche Fehler bereits bei der Eingabe der jeweiligen Programmzeile. Fehler führen gewöhnlich zum sofortigen Abbruch des laufenden Programms; manche Geräte erlauben es dem Programmierer aber z.B. mit dem Befehl ON ERROR GOTO, automatisch zu einem anderen Programmteil (z.B. zu einer Fehlerbehandlungs-Routine) zu verzweigen.

Nur zwei Fehlermeldungen führen bei praktisch allen Basic-Computern nicht zum Programmabbruch: "REDO FROM START" und "EXTRA IGNORED". Erstere Meldung wird ausgegeben, wenn der Computer nach einer INPUT-Anweisung eine Zahl erwartet, der Benutzer aber nichtnumerische Zeichen (z.B. Buchstaben) eingetippt hat. EXTRA IGNORED bedeutet, daß mehr Daten eingegeben wurden, als der Computer verlangte.

Wird etwa bei dem Befehl

```
INPUT A$
```

die Zeichenfolge

```
GUTEN TAG, HIER BIN ICH
```

eingegeben, so betrachtet der Rechner GUTEN TAG und HIER BIN ICH als zwei voneinander durch ein Komma eindeutig getrennte Strings. Da er aber nur einen erwartete, ignoriert er HIER BIN ICH und gibt deshalb konsequenterweise auch EXTRA IGNORED aus. Das Programm läuft danach normal weiter.

Natürlich sind die bei Fehlern ausgegebenen Meldungen nicht bei allen Computern gleich. AIM-65 und PC-100 geben nur die beiden letztgenannten Fehler als Klartext aus; alle anderen bestehen aus zwei Buchstaben (z.B. SN für "Syntax") und dem Wort ERROR (s.u.). PET-2001 und CBM-3032 sind hier vorbildlich: Alle Fehlermeldungen werden im Klartext auf den Bildschirm geschrieben. Beim ABC-80 werden dagegen Zahlencodes benutzt, so daß erst in einer Tabelle nachgesehen werden muß, um welchen Fehler es sich nun handelt.

Im folgenden sind die bei AIM-65 und PC-100 vorhandenen Fehlermeldungen aufgeführt. Dazu wäre noch zu sagen, daß die Werte der Variablen nach einem Programmabbruch durch Fehlermeldungen zwar erhalten bleiben, ein Fortfahren mit CONT aber erneut zu einem Fehler führt, also nicht möglich ist. Da ein Programmstart mit RUN aber alle Variablenwerte löscht, empfiehlt es sich, mit POKE 134,2 das Error-Flag gewaltsam zu löschen und es nun mit CONT zu versuchen (nur bei AIM-65 und PC-100 so möglich).

Fehlercode	Bedeutung
BS	Falsche Indizierung (Bad Subscript). Es wurde versucht, sich auf ein Matrix-Element zu beziehen, das außerhalb der Dimension der Matrix liegt. Dieser Fehler kann auch auftreten, wenn die falsche Dimension als Matrix-Bezug gewählt wird. z.B. LET A(1,1,1)=Z,wenn aber A mit DIM(2,2) dimensioniert wurde.
CN	Folgefehler (Continue Error). Versuch, ein nicht existierendes Programm, ein Programm nach einem Fehler oder nachdem eine neue Zeile in das Programm eingetragen wurde, fortzusetzen.
DD	Doppelte Dimensionierung (Double Dimension). Nachdem eine Matrix dimensioniert wurde, wurde noch eine Dimensionsanweisung für die gleiche Matrix entdeckt. Dieser Fehler tritt dann oft auf, falls einer Matrix die vorher definierte Dimension 10 gegeben wurde, da auf eine Anweisung wie A(I)=3 gestoßen und erst später im Programm ein DIM A(100) gefunden wurde.
FC	Funktionsaufruffehler (Function Call Error). Der Parameter, der an eine mathematische oder String-Funktion übergeben wurde, war außerhalb des zulässigen Bereichs.
ID	Direkt Verboten (Illegal Direct). Sie dürfen eine INPUT-, DEF- oder GET-Anweisung nicht als direktes Kommando verwenden.
LS	Zu langer String (Long String). Mit dem Verkettungsoperator „+" wurde versucht, einen String mit mehr als 255 Zeichen herzustellen.
NF	NEXT ohne FOR. Die Variable in einer NEXT-Anweisung gehört zu keiner vorher ausgeführten FOR-Anweisung.
OD	Keine Daten da (Out of Data). Eine READ-Anweisung wurde ausgeführt, aber alle DATA-Zuweisungen im Programm sind schon gelesen worden. Das Programm versuchte, zuviele Daten zu lesen, oder zu wenig Daten wurden in das Programm eingeschlossen.
OM	Speicher voll (Out of Memory). Programm zu groß, zu viele Variable, zu viele NEXT-FOR-Schleifen, zu viele GOSUB's, zu komplizierter Ausdruck oder eine Kombination dieser Fälle.

OV	Überlauf (Overflow). Das Ergebnis einer Berechnung war zu groß und im Basic-Zahlenformat nicht mehr darstellbar. Falls ein Unterlauf auftritt, wird als Ergebnis Null angenommen und die Programmausführung ohne Fehlermeldung fortgesetzt.
RG	RETURN ohne GOSUB. Es wurde auf eine RETURN-Anweisung gestoßen ohne vorherige Ausführung einer GOSUB-Anweisung.
SN	Syntax-Fehler. Fehlende Klammern in dem Ausdruck, nicht zulässige Zeichen in einer Zeile, unkorrekte Satzzeichen usw.
ST	Zeitweiliger Stringfehler (String Temporaries). Ein Stringausdruck war zu komplex. Machen Sie zwei kürzere Strings daraus.
TM	Variablentyp falsch (Type dismatch). Die linke Seite einer Zuweisung war eine numerische Variable, und die rechte Seite war ein String oder umgekehrt; oder einer Funktion, die ein String-Argument erwartete, wurde eine Zahl übergeben oder umgekehrt.
UF	Undefinierte Funktion. Man bezog sich auf eine Anwender-Funktion, die nie definiert wurde.
US	Undefinierte Anweisung (Undefined Statement). In einem GOTO, GOSUB oder THEN wurde eine Anweisung eingegeben, die nicht existiert.
/0	Division durch Null.

6.4 Optimierung von Basic-Programmen

Die Tatsache, daß ein selbstgeschriebenes Programm zum erstenmal läuft und richtige Ergebnisse liefert, ist sicher eines der größten Erfolgserlebnisse im Leben eines Programmierers. Es wäre aber fehl am Platze, nun anzunehmen, das Programm wäre nicht mehr verbesserungsfähig. In vielen Fällen – eigentlich fast immer – sind noch Bemühungen um zwei Dinge angebracht:

- weniger Speicherplatzbedarf;
- weniger Rechenzeit.

Nicht selten kommt es vor, daß man beide Faktoren um 30 Prozent verbessern kann, ohne einen Nachteil hinsichtlich der übrigen Programmeigenschaften hinnehmen zu müssen. Allerdings geht dabei oft ein gewisses Maß an Übersichtlichkeit des Ablaufes verloren. So ist es im Interesse des Speicherbedarfs und der Geschwindigkeit z.B. angebracht, alle REM-Kommentare und auch alle Leerräume (Spaces) im Programm zu beseitigen. Der Anwender muß selbst wissen, wo er höhere Prioritäten setzt.

6.4.1 Schnellere Programme

Die interpretative Arbeitsweise ist dafür verantwortlich, daß Basic-Programme meist deutlich langsamer arbeiten als vergleichbare Maschinenprogramme. Dies kann jedoch bei vielen Rechen- und Textverarbeitungsaufgaben zugunsten einer leichteren Programmierbarkeit hingenommen werden. Lediglich bei zeitkritischen Aufgaben, z.B. bei der Steuerung von Peripheriegeräten oder bei solch komplexen Dingen wie einem Schachspiel, wird man auf die Möglichkeit zurückgreifen, Unterprogramme in der prozessor-spezifischen Maschinensprache zu verwenden und eventuell mit SYS oder USR vom Basic-Programm her abzurufen.

An einem kleinen Beispiel wollen wir uns einmal ansehen, wie man bei Basic-Programmen die Geschwindigkeit erhöhen kann. Die dabei angegebenen Ausführungszeiten beziehen sich auf das 6502-Microsoft-Basic des AIM-65, das etwa mit dem des PET 2001 übereinstimmt. Nehmen wir an, wir hätten folgendes Programm vorliegen:

```
10 FOR I=0 TO 20000
15 REM SCHLEIFE
20 NEXT I
```

Starten wir es mit RUN, so meldet der Computer nach etwa 36 Sekunden, daß er damit fertig ist. Eine erste Maßnahme zur Verringerung der Ausführungszeit (und des Speicherbedarfs) ist es, alle Leerräume wegzulassen:

```
10 FORI=0TO20000
15 REM SCHLEIFE
20 NEXTI
```

Ein Versuch zeigt, daß sich die Laufzeit nur etwa um eine Sekunde auf 35 s verkürzt hat – eine kaum nennenswerte Verbesserung. Der nächste Schritt bringt schon mehr: Wenn wir die Zeile 15 mit der REM-Anweisung weglassen, meldet sich der Computer schon 27 s nach RUN wieder. Eine weitere Verbesserung können wir erzielen, wenn wir schreiben:

```
10 FORI=0TO20000
20 NEXT
```

Jetzt benötigt das Programm nur noch 22 s – einfach durch Weglassen von I nach NEXT!

Bei „Tiny Basic" läßt sich das Programmbeispiel leider nicht nachvollziehen, da es hier keine FOR-NEXT-Anweisung gibt. Die (gleichwertige) Schleife

```
10 I=0
15 I=I+1
20 IFI < 20000GOTO15
30 END
```

dauert in Tiny Basic auf einem 6502-System knapp zehn Minuten – eine Folge des direkten Interpretierens ohne Zwischencode.

Benötigt man z.B. die Zahl π mehrmals in einem Programm, so ist es zweckmäßig, diese Zahl irgendeiner Variablen zuzuweisen und später die Variable statt der Konstanten aufzurufen. Folgendes Programmbeispiel benötigt eine Ausführungszeit von etwa 17 Sekunden:

```
10 FOR I=0 TO 500
20 B=3.14159265
30 NEXT
```

17 Sekunden werden also benötigt, die Konstante 3,14159265 fünfhundertmal der Variablen B zuzuweisen und sie dabei in das interne Fließkomma-Format umzurechnen. Die folgende,

gleichwertige Programmversion läßt die Ausführungszeit auf etwa 1,2 Sekunden schrumpfen:

```
 5 A=3.14159265
10 FOR I=0 TO 500
20 B=A
30 NEXT
```

Die Umrechnung in das Fließkomma-Format muß hier nur einmal geschehen, und 500mal wird der Variablen B der Wert der Variablen A (und damit wiederum π) zugewiesen.

Dieser Programmiertrick ist allerdings nur dann sinnvoll, wenn es sich wirklich um eine Fließkomma-Konstante handelt. Will man dagegen einer Variablen solche Werte wie 0, 1, 2 oder 5 zuweisen, so ergibt sich kein Geschwindigkeitsvorteil der zweiten Programmversion.

Bei Tiny Basic ist es in bezug auf die Ausführungsgeschwindigkeit günstiger, die Form LET A=... zu verwenden, da der Befehl dann schneller decodiert wird, als wenn man LET wegläßt. Bei anderen Basic-Interpretern spielt dies keine Rolle, da mit und ohne LET ein gleichwertiger Zwischencode erzeugt wird.

Einige nützliche Kleinigkeiten: Der Ausdruck A+A wird schneller berechnet als 2*A, weil die Multiplikation langsamer funktioniert als die Addition. Benötigt man innerhalb eines Programmes mehrmals z.B. SIN(1.5), so sollte man diesen Wert nicht jedesmal neu errechnen, sondern einer Variablen zuweisen, die später beliebig oft aufgerufen werden kann. Das Schreiben mehrerer Befehle in eine statt getrennte Zeilen wirkt sich auf die Geschwindigkeit praktisch nicht aus.

Bei der Abfrage von I/O-Leitungen verwendet man statt des PEEK-Befehls mit nachfolgendem Vergleich besser den Befehl WAIT, wenn auf einen bestimmten Logikzustand gewartet werden soll. Seltsamerweise enthält das PET-Handbuch keinen Hinweis auf diesen Befehl, obwohl er auch beim PET funktioniert.

Es spielt auch eine wesentliche Rolle, welche Stringvariablen in einem Programm als erste vorkommen, da sie dann nicht erst umgeschachtelt werden müssen, wenn andere Strings ihre Länge

ändern. Stringvariablen mit variabler Länge sollte man daher möglichst als letzte definieren. Besonders bei längeren Programmen bzw. umfangreichen Texten spielt dies eine enorme Rolle.

6.4.2 Weniger Speicherplatz

Die Verwendung von LET ist bei Tiny Basic zwar für die Geschwindigkeit vorteilhaft, kostet aber mehr Speicherplatz; hier muß der Programmierer selbst entscheiden, wo er Prioritäten setzt. REM-Kommentare im Programm kosten Zeit und Speicherplatz, erleichtern aber das spätere Verständnis für den logischen Ablauf. Ebenso schaffen Leerräume ein übersichtlicheres Programm, verschwenden aber je ein Byte. Ein „ausgekochtes" Programm geht also immer zu Lasten der Verständlichkeit.

Die Verwendung der END-Anweisung ist bei Tiny Basic stets nötig, bei anderen Interpretern nur vor Unterprogrammen. Außerdem ist es möglich, in eine Zeile mehrere, von einem Doppelpunkt getrennte Befehle zu schreiben, so daß weniger Speicherplatz für die Zeilennummern benötigt wird. Selbstverständlich sollte man auch darauf achten, Speicher für ein- und mehrdimensionale Variablen am Anfang des Programms den wirklichen Erfordernissen nach zu reservieren und auch die Null-Indices, z.B. A(0), zu verwenden, da auch sie erlaubt sind. Nach Möglichkeit sollte man Variablennamen mehrmals im Programm verwenden, wenn es nicht darauf ankommt, den alten Wert zu retten; auf diese Weise braucht kein zusätzlicher Speicherraum reserviert zu werden. Wenn zum Beispiel die Variable K$ verwendet wird, um das Tastenfeld mit dem GET- oder INPUT-Befehl abzufragen, so kann dies später im Programm wieder mit K$ geschehen.

Computer wie der PET-2001 oder der TRS-80 gestatten für viele Befehle Abkürzungen, z.B. ein Fragezeichen statt "PRINT" Diese Abkürzungen haben auf den Speicherplatzbedarf der Programme keinerlei Auswirkung, da der gleiche Zwischencode wie bei der „normalen" Befehlsform erzeugt wird; sie verkürzen lediglich die Eingabe. Listet man ein so eingetipptes Programm auf, erscheinen die Befehle wieder in voller Länge.

6.4.3 Benutzerfreundlichkeit

Programme, die als Selbstzweck oder nur zur Verwendung des Programmierers selbst geschrieben wurden, können bedenkenlos nach den oben gegebenen Ratschlägen optimiert werden. Programme, die auch in anderer Leute Hände gelangen können, erlauben diese Durchforstung aber nur in begrenztem Umfang, besonders, was die Verwendung von REM-Kommentaren angeht.

Ein wichtiger Aspekt der Benutzerfreundlichkeit ist die Möglichkeit von Basic, im Dialog mit dem „Operator" zu kommunizieren; dies sollte man auch ausnutzen. Wenn man Programme schreibt, bei denen bestimmte Funktionen nur mit Schlüsselworten ausgelöst werden können, so sollte man diese Schlüsselworte und ihre Bedeutung am Anfang des Programms erst einmal auf den Bildschirm schreiben, um die „Spielregeln" zu definieren. Bei komplexen Programmen, die den Computer u.U. minutenlang beschäftigen, sollte man dem Benutzer mitteilen, daß es etwas dauern wird, um zu vermeiden, daß dieser voller Ungeduld die Break-Taste drückt und wieder von vorne anfangen muß. Allerdings: Je komfortabler dieser Dialog-Betrieb wird, desto mehr Speicherplatz belegt das Programm und desto langwieriger wird auch für den Benutzer der Umgang mit ihm. Möge jeder sein Maß finden!

6.4.4 "Radikalkuren"

In bestimmten Fällen ist der Programmierer gezwungen, jede Rücksichtnahme auf Änderungsmöglichkeiten, Editierbarkeit, Übersichtlichkeit und System-Kompatibilität fallen zu lassen, um ein recht umfangreiches Programm "mit Gewalt" in einen Computer mit eingeschränktem Speicherplatz zu pressen. Dem Leser sei empfohlen, bei seinen Programmen die im folgenden geschilderten Maßnahmen nur dann anzuwenden, wenn dies wirklich notwendig ist – wenn man also das Programm bereits nach allen anderen Gesichtspunkten optimiert hat und trotzdem an die Speicherbereichsgrenzen gelangt ist. Sehen wir uns die Radikalmaßnahmen einmal näher an:

Wie schon erwähnt, ist es günstig, möglichst viele Befehle in einer einzigen Programmzeilen-Nummer unterzubringen, da auch die Zeilennummern Speicherplatz benötigen. Computer wie der PET-2001 begrenzen eine Programmzeile auf zwei Bildschirmzeilen, das sind 80 Zeichen. Diese Begrenzung ist jedoch nur bei der Eingabe, nicht bei Speicherung und Ausgabe wirksam. Also kann man Befehlsabkürzungen verwenden, z.B. "? " statt "PRINT", "Ve" statt "VERIFY" usw., zwei Bildschirmzeilen mit diesen Abkürzungen als eine Programmzeile deklarieren und dann beim Auflisten erstaunt feststellen, daß die Ausgabe auf dem Bildschirm mit "PRINT" bzw. "VERIFY" durchaus auch vier Zeilen umfassen kann. (Versuchen Sie, in Programmzeile 10 einfach zwei Bildschirmzeilen voll Fragezeichen und Doppelpunkte in den PET oder CBM zu tippen und dann Zeile 10 aufzulisten!). Erwähnt sei noch, daß der CBM/PET für fast alle Basic-Befehle Abkürzungen zuläßt, die meist aus dem ersten Befehlsbuchstaben ohne Shift und dem zweiten mit Shift bestehen.

Eine weitere Eigenart des PET ist auch nützlich, um Speicherplatz zu sparen. Statt

```
10 PRINT"WERT:";A  kann man auch schreiben

10 ?"WERT:"A
```

Den Strichpunkt darf man nämlich nach einem Anführungszeichen weglassen, wenn es nicht am Zeilenende steht.

Bei Programmen, die relativ viele PRINT-Befehle mit Textausgabe enthalten, z.B. um dem Benutzer des Computers genaue Anweisungen oder Erläuterungen zu geben, kann man mit folgendem Trick Speicherplatz sparen: Alle Worte, die häufig im Text vorkommen und die wenigstens vier Zeichen umfassen (ev. mit einem Leerraum davor und dahinter), deklariert man am Anfang des Programms als String-Variable. Ein kleines Beispiel:

```
10 D$="DER WERT DER VARIABLEN ":
   B$=" BETRÄGT
250 PRINTD$"A"B$;A
260 PRINTD$"B"B$;B
```

usw. (Anführungszeichen am Zeilenende dürfen entfallen). Je nach Zusammensetzung der auszugebenden Texte kann man mit dieser Methode oft 30 % Speicherplatz einsparen, wobei allerdings die Übersichtlichkeit völlig verloren geht.

Schließlich muß noch bedacht werden, daß in allen GOTO- und GOSUB-Befehlen die Zeilennummern der Sprungziele in ASCII-Form gespeichert werden. Infolgedessen muß es auch Bestandteil unserer "Radikalkur" sein, diese Zeilennummern möglichst kurz zu halten. Die sicherste Methode dabei ist selbstverständlich, die Zeilennummern des gesamten Programms nicht in Zehner-, sondern in Einerschritten zu staffeln. Abhängig von der Zahl der Sprungbefehle kann man damit wieder einige Bytes einsparen, macht aber das spätere Einschieben von Zeilen unmöglich (bei Programmen, die den Speicherplatz voll ausnützen, ist das aber ohnehin unmöglich).

Programmieranfängern sei die Verwendung der in diesem Abschnitt gebrachten Speicherspar-Methoden abgeraten, da sie Übersichtlichkeit und Änderbarkeit von Programmen stark einschränken.

7 Einige Beispiel-Programme in Basic

Die nachfolgenden Programme sollen dem Anwender Programmiertechniken und -tricks zeigen, aber ihm auch die Möglichkeit geben, seinen Computer nutzbringend einzusetzen, ohne bereits vertiefte Kenntnisse des Basic-Programmierens zu besitzen. Deshalb finden sich hier ausschließlich sogenannte "nützliche" Programme und keine Spielereien.

Wie bereits in Kapitel 3 gezeigt, sind die heutigen Personal Computer leider nicht ganz syntax-kompatibel. Die meisten folgenden Programme sind für die Computer PET-2001, CBM-3032, AIM-65, PC-100, ITT-2020 und Apple-II ausgelegt, die alle das von der amerikanischen Firma Microsoft entwickelte 6502-Basic verwenden. Die für andere Computer in Einzelfällen notwendigen Syntaxänderungen lassen sich leicht in Kapitel 5 nachschlagen. Es soll keinesfalls der Eindruck erweckt werden, die hier veröffentlichte Software sei absolut optimal und ließe keinerlei Verbesserungen mehr zu. Es ist nicht übertrieben, wenn behauptet wird, daß sich j e d e s Programm noch verbessern läßt – und sei es durch Verkürzen um nur wenige Byte Speicherplatz. Im Vordergrund soll hier jedoch die Art und Weise der Problemlösung stehen, nicht das Feilschen um das letzte Byte.

7.1 Langsame Textausgabe

Aus dem kurzen Programm in *Abb. 7.1* kann man schon allerhand lernen. Es dient dazu, fünf Textzeilen nacheinander mit einer Verzögerung von etwa einer halben Sekunde auszugeben. Das ist besonders bei dem einzeiligen LED-Display von AIM-65 und PC-100 nützlich.

Der Satz "Ein wackerer Bayer vertilgt bequem zwo Pfund Kalbshaxe" gehört zu jenen kuriosen Gebilden, die alle Buchstaben des Alphabets enthalten, wenn man einmal von den Um-

Abb. 7.1 So erreicht man bei Mikrocomputern mit einzeiligem Display eine langsame Textausgabe. Der Testsatz enthält übrigens alle Buchstaben des Alphabets

```
10 DATA EIN WACKERER BAYER
15 DATA VERTILGT BEQUEM ZWO
20 DATA PFUND KALBSHAXE
25 DATA UND JETZT NOCHMAL
30 DATA "ALLES VON VORN:"
40 RESTORE
50 FOR I=0 TO 4
60 READ A$:PRINT A$
70 FOR J=0 TO 1000:NEXT
80 NEXT I:GOTO 30
```

lauten und Ziffern absieht. Die einzelnen Textzeilen stehen hier in Data-Statements, wobei die Anführungszeichen in Zeile 30 nur deshalb da sind, weil sonst der Doppelpunkt nach "vorn" nicht ausgedruckt würde – dieses Zeichen dient ja normalerweise zur Trennung von Befehlen und würde ignoriert, wenn man die Anführungszeichen wegläßt.

Da das Programm die Zeilen mit READ-Befehlen liest, würde ohne den RESTORE-Befehl in Zeile 40 nach einem Durchlauf ein "Out of Data Error" auftreten. Zeile 70 dient zur Verzögerung zwischen der Ausgabe der einzelnen Textteile; das NEXT bezieht sich hier auf die Variable J – ohne Argument nach NEXT wird stets die zuletzt bei FOR genannte Variable verwendet.

Das GOTO 30 in Zeile 80 beweist, daß man auch auf eine DATA-Zeile springen darf, ohne daß eine Fehlermeldung auftritt; es könnte aber genausogut GOTO 40 heißen.

Hat man das kleine Programm einmal mit RUN gestartet, so läßt es sich nur noch mit der Break- bzw. Stop-Taste anhalten, bei manchen Computern auch mit der Tastenkombination CTRL C. Es lohnt sich, einige Änderungen am Programm selbst zu erproben, z.B. die Zeile 50 auf FOR I=312 TO 316 zu ändern – es ist erstaunlich, was man alles ändern kann, ohne daß sich irgendwelche Beeinflussungen des Ablaufes ergeben.

Damit wird auch gleich deutlich, daß es für ein und dasselbe Problem viele unterschiedliche Software-Lösungen geben kann. Stellt man zehn Programmierer vor ein Problem, so kommen zwanzig verschiedene Programme dabei heraus...

7.2 Fragebogen-Auswertung

Auch mit einem vergleichsweise kleinen Basic-Rechner wie dem AIM-65 bzw. PC-100 lassen sich schon erhebliche Arbeitszeit-Einsparungen bei statistischen Aufgaben erzielen, wie dieses Beispiel zeigt. Das Programm in *Abb. 7.2* dient dazu, eine größere Anzahl von Fragebogen auszuwerten, die jeweils eine bestimmte Anzahl von Fragen mit Antworten in Form irgendeiner numerischen Bewertung enthalten (z.B. Punkte oder Noten, 0...9,99). Nach dem Abschluß der Eingabe aller Bewertungen erfolgt die Ausgabe der errechneten Mittelwerte zu den einzelnen Fragen. Enthalten manche Fragebogen ungültige, nicht auswertbare Antworten, so ist statt der Bewertungszahl lediglich ein Minuszeichen "–" einzugeben. Die entsprechenden Antworten werden dann ignoriert und gehen in den Mittelwert nicht ein.

Das Problem enthält einige bemerkenswerte Befehlsfolgen. Um statt einer Zahl bei ungültigen Antworten auch ein nichtnumerisches Zeichen eingeben zu können, erfolgt die Abfrage mit einem String-INPUT, der bei gültiger Antwort mit VAL in eine numerische Variable umgewandelt wird. Umgekehrt dient der Befehl STR$ bei der Ausgabe der errechneten Mittelwerte dazu, Zahlen variabler Länge in einen String umzuwandeln. Handelt es sich um eine ganze Zahl, so wird ein Dezimalpunkt hinzugefügt, und in jedem Fall werden noch so viele Nullen angehängt, wie es zum Erreichen eines Festkomma-Formats mit zwei Stellen hinter dem Dezimalpunkt erforderlich ist. (Bei Computern mit dem Befehl PRINT USING ginge das wesentlich einfacher.) Das Hinzuaddieren von 0.005 in Zeile 270 sorgt für eine richtige 4/5-Rundung für dieses Festkommaformat.

Die im Programm verwendeten Variablennamen sind:

A = Zahl der Fragen pro Fragebogen
B = Laufvariable für 1...A (Nr. der Frage auf dem Bogen)
C = Laufvariable für die Fragebogen-Nummer
A$ = Stringvariable für Ein- und Ausgabe
D(B) = Zahl der bisherigen Antworten bei Frage B
E(B) = Summe der Bewertungen bei Frage B

```
<6>
LIST

 100 INPUT"WIEVIELE FRAGEN";A
 110 DIM D(100),E(100):C=0
 120 IF C<2 GOTO 160
 130 PRINT"NOCH EIN BOGEN? J/N"
 140 GET A$:IF A$="N"GOTO 220
 150 IF A$<>"J"GOTO 140
 160 C=C+1:PRINT"-- BOGEN NR.";C;"--"
 170 FOR B=1TO A:PRINT"FRAGE";B;:INPUT A$
 180 IF A$<CHR$(48)GOTO 200
 190 D(B)=D(B)+1:E(B)=E(B)+VAL(A$)
 200 NEXT B:REM NAECHSTE FRAGE
 210 GOTO 120:REM NAECHSTER BOGEN
 220 PRINT C;"FRAGEBOGEN MIT"
 230 PRINT" JE";A;"FRAGEN"
 240 PRINT"M I T T E L W E R T:"
 250 FOR B=1 TO A
 260 PRINT"FRAGE";B;":";
 270 A$=STR$(E(B)/D(B)+.005)
 280 IF LEN(A$)=2 THEN A$=A$+"."
 290 IF LEN(A$)<5 THEN A$=A$+"0":GOTO 290
 300 PRINT LEFT$(A$,5):NEXT B
 310 PRINT"-- F E R T I G --"
RUN
WIEVIELE FRAGEN? 3
-- BOGEN NR. 1 --
FRAGE 1 ? 2
FRAGE 2 ? 4
FRAGE 3 ? 6
-- BOGEN NR. 2 --
FRAGE 1 ? 3
FRAGE 2 ? -
FRAGE 3 ? 2
NOCH EIN BOGEN? J/N
-- BOGEN NR. 3 --
FRAGE 1 ? 4
FRAGE 2 ? 3
FRAGE 3 ? 5
NOCH EIN BOGEN? J/N
 3 FRAGEBOGEN MIT
 JE 3 FRAGEN
M I T T E L W E R T:
FRAGE 1 : 3.00
FRAGE 2 : 3.50
FRAGE 3 : 4.33
-- F E R T I G --
```

Abb. 7.2 Basic-Programm zum Auswerten von Fragebögen. Man beachte die Verwendung von VAL zum Umwandeln eines Strings in eine numerische Variable

Das Programm begrenzt in Zeile 110 die Zahl der Fragen pro Bogen auf 100. Bei ausreichendem Speicherplatz ist es ohne weiteres möglich, dieses Limit zu erhöhen.

7.3 Primfaktoren-Zerlegung

In der Kürze liegt die Würze: Das Programm in *Abb. 7.3* zerlegt eine (ganzzahlige) Zahl Z in ihre Primfaktoren. Zeile 20 enthält dabei einen recht ungewohnt aussehenden Ausdruck, der je nach dem momentanen Wert von P diesen um 1 oder 2 erhöht. Dabei wird der Microsoft-Basic-Effekt benützt, daß der Ausdruck P>2 Null ist, wenn die Bedingung nicht erfüllt ist, andernfalls ist er −1. Die Variable L legt eine obere Grenze fest, bis zu der Primfaktoren errechnet werden; es genügt nämlich, bis zur Wurzel der zu untersuchenden Zahl zu rechnen. Ferner genügt es bei Primfaktoren größer als 2, nur noch ungerade Faktoren zu testen (dazu erhöht Zeile 20 den Wert von P um 2, wenn er größer als 2 ist).

Es sei darauf hingewiesen, daß der Vergleichstrick in Zeile 20 nicht mit allen Basic-Interpretern funktioniert; getestet

```
LIST

 1 REM PRIMFAKTOREN-ZERLEGUNG
 10 INPUT"ZAHL";Z:P=1:L=SQR(Z)
 20 P=P-(P>2)+1
 30 A=Z/P:IF A>INT(A) GOTO 50
 40 PRINT P;: Z=A:GOTO 30
 50 IF P<L GOTO 20
 60 IF Z>1 THEN PRINT Z;
 70 PRINT"FERTIG":GOTO 10
RUN
ZAHL? 1234567
 127          9721
FERTIG
ZAHL? 54321
 3            19
 953         FERTIG
ZAHL?
```

Abb. 7.3 Die Primfaktoren-Zerlegung ist z.B. nützlich, wenn man mit Hilfe eines hochfrequenten Quarzes und einer Teilerkette eine bestimmte Niederfrequenz erzeugen möchte. Die Primfaktoren stellen dabei die nötigen Teilerfaktoren dar

wurde er auf den Geräten PET, CBM und AIM-65/PC-100. Eine Alternative stellt die Verwendung der Zeile

```
20 P=P+1:IF P>3 THEN P=P+1
```

dar, was natürlich ein wenig mehr Speicherplatz kostet und auch die Rechengeschwindigkeit vermindert.

```
 10 PRINT"DEZIMAL-BINAER- UND
 20 PRINT"BINAER-DEZIMAL-UMWANDLUNG"
 30 PRINT"WOLLEN SIE DEZIMAL-"
 40 PRINT"ODER BINAERZAHLEN EINGEBEN (D/B)?"
 50 GETH$:IFH$="B"GOTO100
 60 IFH$="D"GOTO200
 70 GOTO50
 100 INPUT"BINAER";B$
 110 A=1:C=0
 120 FORI=LEN(B$)TO1STEP-1
 130 IFMID$(B$,I,1)="1"THENC=C+A
 140 A=A+A:NEXT
 150 PRINT"DEZIMAL:";C:GOTO30
 200 INPUT"DEZIMAL";A:H$=" BINAER"
 210 A=A/2
 220 IF A=INT(A)THEN H$="0"+H$:GOTO240
 230 H$="1"+H$
 240 A=INT(A):IF A>0 GOTO210
 250 PRINT H$:GOTO30
RUN
DEZIMAL-BINAER- UND
BINAER-DEZIMAL-UMWANDLUNG
WOLLEN SIE DEZIMAL-
ODER BINAERZAHLEN EINGEBEN (D/B)?
DEZIMAL? 678976564
10100001111000010111000011O100 BINAER
WOLLEN SIE DEZIMAL-
ODER BINAERZAHLEN EINGEBEN (D/B)?
BINAER? 1000000110100001
DEZIMAL: 65953
WOLLEN SIE DEZIMAL-
ODER BINAERZAHLEN EINGEBEN (D/B)?
```

Abb. 7.4 Nahezu beliebig lange Binärzahlen kann dieses Umwandlungsprogramm dank der Verwendung von String-Befehlen verarbeiten

7.4 Dezimal-Binär- und Binär-Dezimal-Umwandlung

Wie an anderer Stelle dieses Buches besprochen, erfolgt die Steuerung externer Geräte über I/O-Leitungen stets bitweise; jede Leitung ist einem Bit des Datenwortes zugeordnet, das zum I/O-Port geschickt wird. Dies geschieht z.B. mit dem POKE-Befehl. Dabei muß das Bitmuster jedoch als Dezimalzahl angegeben werden, so daß eine Zahlensystem-Umwandlung erforderlich wird. Umgekehrt liefert der PEEK-Befehl bei der Abfrage eines Datenports eine Dezimalzahl, die der Programmierer irgendwie als Bitmuster interpretieren muß.

Das Programm in *Abb. 7.4* zeigt eine Möglichkeit, wie ein Basic-Computer die Umrechnung selbst übernehmen kann. Die Länge der Binärzahl ist praktisch nur durch die Länge des Computer-Eingangspuffers bestimmt (80 Zeichen beim PET-2001, 60 Zeichen beim AIM-65, auf dem dieses Programm entwickelt wurde). Die Umwandlung geschieht durch Zerlegung der Dezimalzahl in ihre Zweierpotenzen. Das Programm zeigt auch, wie sich String-Befehle nutzbringend auch bei arithmetischen Aufgaben einsetzen lassen.

7.5 Mischprodukte von Oszillatoren

Bei der Dimensionierung von Empfängern und Sendern ist es von großer Bedeutung, die Frequenzen der dafür erforderlichen Oszillatoren so zu wählen, daß ihre Grundfrequenzen und Oberwellen (Harmonische) ebensowenig wie die sich ergebenden Kombinationsfrequenzen (Mischprodukte) in den Empfangs- bzw. Sendefrequenzbereich fallen.

Hierbei ergibt sich die Schwierigkeit, daß eine sehr große Anzahl von Mischprodukten auftreten kann, da sich ja auch die Oszillator-Oberwellen mischen können. Die Berechnung von Hand erfordert deshalb sehr viel Zeit.

Mit dem Programm in *Abb. 7.5* verläuft die Errechnung der Mischprodukte automatisch, und es werden nur jene Ergebnisse ausgegeben, die in den interessierenden Frequenzbereich fallen, z.B. in den Abstimmbereich eines Empfängers.

```
10 PRINT"sQQ·  *** MISCHPRODUKTE ***":INPUT"QQQUNTERE GRENZE";U
20 INPUT"OBERE GRENZE";O
30 INPUT"OSZILLATOR-ANZAHL";VZ
40 VZ=VZ-1:REM WEIL AUCH 0-INDEX MOEGL.
50 INPUT"MAXIMALE HARMONISCHEN-ORDNG.";EW
60 DIMA(VZ):DIMF(VZ)
70 FORI=0TOVZ:A(I)=-EW:PRINT"F";I+1;:INPUTF(I):NEXT
75 A(VZ)=0
80 Z=0:REM INDEX
90 S=0:REM QUERSUMME
100 FORI=0TOVZ:S=F(I)*A(I)+S:NEXT
110 S=ABS(S):IFS>UANDS<OTHENPRINTS,
120 A(Z)=A(Z)+1:IFA(Z)>EWGOTO140
130 GOTO90
140 A(Z)=-EW
150 Z=Z+1:IFZ>VZTHENPRINT"FERTIG":END
160 A(Z)=A(Z)+1
170 IFA(Z)>EWGOTO140
180 Z=0:GOTO90
```

Abb. 7.5 Wer Empfänger oder Sender baut, weiß dieses Programm zu schätzen. Es errechnet alle denkbaren Mischprodukte mehrerer Oszillatorfrequenzen und derer Harmonischen. Die Besonderheit ist hier, daß eine Gleichung mit variabler Gliederzahl berechnet werden muß – je nachdem, wieviele Oszillatoren vorhanden sind (Q = Cursor-Steuerzeichen nach unten; s = Bildschirm löschen)

Eine Besonderheit ist die Tatsache, daß die Anzahl der Oszillatoren frei gewählt werden kann (max. 255). Die Berechnung erfolgt nach folgendem Schema:

$$f_m = a_1 f_1 \pm a_2 f_2 \pm a_3 f_3 \pm \ldots \pm a_n f_n$$

f_m ist dabei das jeweilige Mischprodukt, $f_1 \ldots f_n$ sind die Grundfrequenzen der n Oszillatoren. Die Faktoren a kennzeichnen die jeweilige Oberwelle des zugehörigen Oszillators und stellen Laufvariablen dar, deren Wert von Null bis zur maximalen Harmonischen-Ordnungszahl durchläuft. Die Berechnung erfordert alle denkbaren Kombinationen von $a_1 \ldots a_n$. Um den zusätzlich noch notwendigen Vorzeichenwechsel einzusparen, läßt das Programm die Faktoren $a_2 \ldots a_n$ nicht von Null, sondern von der negativen maximalen Harmonischen-Ordnungszahl ab laufen.

Das Durchlaufen der Faktoren $a_1 \ldots a_n$ kann man sich wie bei den einzelnen Stellen eines Bandzählwerkes vorstellen. Es ist leicht einsehbar, daß wegen der großen Zahl der möglichen Kombinationen die Rechenzeit mit der Zahl der Oszillatoren und der maximalen Harmonischen-Ordnungszahl schnell zunimmt. Das Programm macht deutlich, wie sich indizierte Variablen für Gleichungen einsetzen lassen, deren Gliederzahl variabel ist.

7.6 Hex-Monitor

"Monitor" bedeutet hier nicht etwa Bildschirm, sondern "Programm zur Eingabe von Routinen in Maschinensprache". Zahlreiche Mikrocomputer wurden speziell für das Programmieren in Basic konstruiert und enthalten daher im Urzustand keine Möglichkeit zur Eingabe von Programmen in Maschinensprache. Obwohl in diesem Buch nicht näher auf die Maschinenbefehle unterschiedlicher CPU-Typen eingegangen wird, sei doch ein einfaches, in Basic geschriebenes Monitorprogramm angegeben. Die Verwendung wird jedoch nur fortgeschrittenen Programmierern empfohlen, weil man durch unbeabsichtigtes Ansprechen von Adressen, die vom Basic-Interpreter benutzt werden, das gesamte Computersystem "aufhängen" kann, so daß es nur

```
10 REM ◆◆◆ HEX MONITOR ◆◆◆
20 REM COMMANDS: POKE,LIST,RUN
30 DIMH$(16):T=100
40 DATA0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
50 FORI=0TO15:READH$(I):NEXT
60 INPUT"CMD=";A$
70 INPUT"DEC.ADR.:FROM=";B
80 IFA$="LIST"GOTO200
90 IFA$="POKE"GOTO400
100 IFA$="RUN"GOTO600
110 GOTO60
200 INPUT"TO=";C
210 FORI=BTOC:D=PEEK(I)
220 E=INT(D/16):F=DAND15
230 PRINTI;H$(E)+H$(F)
240 FORJ=0TOT:GETD$
250 IFD$="0"THENT=10
260 IFD$="S"THENT=200
270 IFD$="X"GOTO60
280 NEXTJ
290 NEXTI
300 GOTO60
400 PRINT" ":PRINTB;:GOSUB500
410 D=16◆I:GOSUB500
420 D=D+I:POKEB,D:B=B+1:GOTO400
500 GETD$:IFD$="X"THENCLEAR:GOTO30
510 IFD$<"0"ORD$>"F"GOTO500
520 FORI=0TO15:IFD$=H$(I)GOTO540
530 NEXT
540 PRINTD$;:RETURN
598 REM AIM USR ADR = 4, 5
599 REM PET USR ADR = 1, 2(CHANGE600)
600 POKE5,INT(B/256):POKE4,BAND255
610 INPUT"ARGUMENT";A
620 PRINT"RESULT:";USR(A):GOTO60
RUN
CMD=? POKE
DEC.ADR.:FROM=? 3000

 3000 A9
 3001 33
 3002 A0
 3003 56
 3004 4C
 3005 D1
 3006 C0
 3007 CMD=? LIST
DEC.ADR.:FROM=? 3000
TO=? 3003
 3000 A9
 3001 33
 3002 A0
 3003 56
CMD=? RUN
DEC.ADR.:FROM=? 3000
ARGUMENT? 0
RESULT: 13142
CMD=?
```

7.6 Wem Basic nicht mehr genügt, kann mit diesem Hex-Monitor seine Programme in Maschinensprache schreiben. Dabei ist allerdings äußerste Vorsicht angebracht...

durch Aus- und Einschalten wieder aus dem Weltraum zurückgeholt werden kann.

Unser Monitorprogramm in *Abb. 7.6* gestattet das Arbeiten in drei Betriebsarten: POKE zur Eingabe eines Maschinenprogrammes als Folge hexadezimaler Bytes ab einer bestimmten, dezimal anzugebenden RAM-Adresse; LIST zum hexadezimalen Auflisten von Bytes in beliebigen Speicherbereichen, und RUN zum Ausführen eines Maschinenprogramms ab einer bestimmten dezimalen Adresse. In der Betriebsart POKE dient die Taste X dazu, in den Befehlseingabe-Modus zurückzukehren. Die Betriebsart RUN kann dem Maschinenprogramm im Fließkomma-Akku (siehe Computer-Handbuch) einen beliebigen Zahlenwert zur Weiterverarbeitung als Argument übergeben. Das Maschinenprogramm liefert beim Rücksprung in das Basic-Monitorprogramm ebenfalls einen Zahlenwert, der hier in Akku und Y-Register der CPU 6502 übergeben wird. In der abgedruckten Form läuft das Programm auf den Mikrocomputern AIM-65 und PC-100; beim PET sind die Ziffern 4 und 5 in Zeile 600 durch 1und 2 zu ersetzen, dort wird die Einsprungadresse des Maschinenprogramms für USR abgelegt. Es wird davor gewarnt, das als Beispiel angegebene Maschinenprogramm bei anderen Computern in Betrieb zu nehmen, da sich u.U. das System "aufhängt". Hingewiesen sei noch auf eine notwendige Syntaxänderung beim PET-2001 in Zeile 500: Hier muß es CLR statt CLEAR heißen.

In der Betriebsart LIST kann man den Ausdruck mit S langsamer und mit Q schneller machen. Die Zeitschleife ermöglicht auch bei dem einzeiligen Display des AIM-65 noch ein bequemes Mitlesen.

7.7 Formatierte Zahlenausgabe

Bereits in unserem Fragebogen-Auswertungsprogramm (7.2) trat das Problem der formatierten Festkomma-Ausgabe von Zahlen variabler Länge auf. Dort konnte das Problem relativ einfach gelöst werden, weil wir uns auf Zahlen mit nur einer Stelle vor dem Dezimalpunkt festlegten.

```
10 INPUT A:REM FUER TEST
20 IF A<.006 THEN A=0
25 A$=STR$(INT(.5+A*100))
30 PRINT TAB(19-LEN(A$));
40 PRINT LEFT$(A$,LEN(A$)-2);
50 PRINT".";RIGHT$(A$,2)
100 GOTO10
```

Abb. 7.7 Dieses kurze Programm demonstriert die formatierte Festkomma-Ausgabe von Zahlen veränderlicher Länge – eine Fähigkeit, die vielen Basic-Computern leider fehlt

Läßt man jedoch einen größeren Zahlenbereich zu, z.B. ein- bis sechsstellige Zahlen, so stellt *Abb. 7.7* eine Alternative dar. Auch hier wird der STR$-Befehl verwendet; für die Festkomma-Ausgabe mit zwei Stellen hinter dem Dezimalpunkt ist es erforderlich, die Zahl zuerst mit 100 zu multiplizieren, in einen String umzuformen und in diesen dann künstlich den Dezimalpunkt an richtiger Stelle "hineinzumogeln"

Man beachte, daß die Ausgabe stets so erfolgt, daß alle Dezimalpunkte ausgedruckter Zahlen exakt untereinander stehen. Zu diesem Zweck justiert der TAB-Befehl die Position der ersten Stelle je nach der Länge der Zahl. Die Konstante 19 sorgt für eine rechtsbündige Ausgabe bei einer Zeilenbreite von 20 Zeichen (z.B. AIM-65/PC-100) und kann bei Bedarf entsprechend modifiziert werden.

Zeile 20 sorgt in unserem Programm dafür, daß sehr kleine Zahlen nicht in Exponentialform dargestellt werden (z.B. 4.329E-5), da dies das Ausgabeformat durcheinanderbrächte. Der Benutzer hat aus dem gleichen Grunde darauf zu achten, daß keine zu großen Zahlen (z.B. 123456787654321) verarbeitet werden müssen.

7.8 Zinseszins-Berechnung

Das recht kurze Programm in *Abb. 7.8* enthält eigentlich keine besonderen Programmiertricks, macht aber die arithmetische

```
 10 INPUT"SPARRATE/JAHR";R
 20 INPUT"ANF-KAPITAL";K
 30 INPUT"ZINSSATZ/%";P
 40 INPUT"ZEITRAUM(J.)";N
 50 Q=1+P/100:A=(R*Q*(Q^N-1))/(Q-1)
 60 PRINT"ENDKAPITAL";:K=K*Q^N+A
 70 PRINT INT(K*100)/100
RUN
SPARRATE/JAHR? 500
ANF-KAPITAL? 2000
ZINSSATZ/%? 5.5
ZEITRAUM(J.)? 10
ENDKAPITAL 10208.03
```

Abb. 7.8 Basic-Programm zum Errechnen des Endkapitals bei Sparkonten. Handelt es sich nicht um Einzahlungen, sondern um Abhebungen, so ist ein negativer Wert einzugeben

Hierarchie von Basic deutlich. Man sollte sich ruhig einmal ansehen, welche Operationen in den einzelnen Formeln zuerst ausgeführt werden – es existieren sozusagen "unsichtbare" Klammern.

Das Programm berechnet das Endkapital auf einem Konto mit periodischen Einzahlungen und einer konstanten Zinsrate über mehrere Jahre hinweg. Handelt es sich nicht um Einzahlungen, sondern um regelmäßige Abhebungen, so ist ein negativer Betrag einzugeben. Der Endbetrag des Kapitals wird auf ganze Pfennige abgerundet.

7.9 Ein bißchen Statistik

Eine der Hauptaufgaben größerer Datenverarbeitungsanlagen war es bisher, statistische Auswertungen vorzunehmen. Das kann auch bei relativ umfangreichen Datenmengen heute bereits mit Basic-Tischcomputern geschehen.

Das Programm in *Abb. 7.9.1* erwartet die Eingabe einer praktisch beliebig großen Zahl von numerischen Daten, wobei man 9999 dazu benützt, um dem Rechner mitzuteilen, daß die Ein-

```
 100 REM STATISTIK-FUNKTIONEN
 110 REM DATENEINGABE MIT 9999 ABSCHLIESSEN
 120 S=0:S1=0:N=0
 130 INPUT X:IF X=9999 GOTO 150
 140 S=S+X:S1=S1+X*X:N=N+1:GOTO 130
 150 M=S/N:V=(N*S1-S*S)/N/(N-1)
 160 S3=SQR(V):S4=SQR(V/N)
 170 PRINT"ANZAHL=";N:PRINT"SUMME=";S
 180 PRINT"QUADRATSUMME=";S1:PRINT"MITTELWERT";M
 190 PRINT"VARIANZ=";V:Q=S3/M
 200 PRINT"STANDARDABWEICHUNG=";S3
 210 PRINT"STANDARDFEHLER=";S4
 220 PRINT"VARIATIONSKOEFFIZIENT=";Q
RUN
? 123.6
? 432.3
? 654.3
? 894.2
? 9999
ANZAHL= 4
SUMME= 2104.4
QUADRATSUMME= 1429862.38
MITTELWERT 526.1
VARIANZ= 107579.18
STANDARDABWEICHUNG= 327.992653
STANDARDFEHLER= 163.996326
VARIATIONSKOEFFIZIENT= .623441651
```

Abb. 7.9.1 Programm zum Errechnen der wichtigsten Statistikfunktionen eines Datenfeldes beliebiger Länge

gabe jetzt beendet ist. Das Programm ermittelt dann selbständig Anzahl, arithmetischen Mittelwert, Varianz, Standardabweichung, Standardfehler und Variationskoeffizient des Datenfeldes. Die Programmzeile 120 kann man normalerweise weglassen, da die meisten Rechner nach RUN ohnehin alle Variablen löschen.

Die Berechnung des manchmal recht aussagekräftigen Korrelations-Koeffizienten zweier Datenmengen gestaltet sich schon

```
 100 REM KORRELATIONS-KOEFFIZIENT
 110 REM VON DATENPAAREN X,Y
 120 REM QUELLE: DARTMOUTH COLLEGE
 130 REM KIEWIT COMPUTATION CENTER
 180 S1=0:S2=0:S3=0:S4=0:S5=0
 190 INPUT"ZAHL DER PAARE";N
 200 FOR I=1 TO N:PRINT"X";I;:INPUT X
 210 PRINT"Y";I;:INPUT Y:S1=S1+X
 220 S2=S2+Y:S3=S3+X*Y:S4=S4+X*X
 230 S5=S5+Y*Y:NEXT I
 240 A=N*S3-S1*S2
 250 B=SQR((N*S4-S1*S1)*(N*S5-S2*S2))
 260 A=INT(A/B*1000+.5)/1000
 270 PRINT"KORRELATIONSKOEFFIZIENT =";A
RUN
ZAHL DER PAARE? 5
X 1 ? 1
Y 1 ? 5
X 2 ? 2
Y 2 ? 3
X 3 ? 3
Y 3 ? 0
X 4 ? 4
Y 4 ? -5
X 5 ? 5
Y 5 ? -11
KORRELATIONSKOEFFIZIENT =-.978
```

Abb. 7.9.2 Ermittlung des Korrelations-Koeffizienten als Maß für die Qualität einer Geraden-Anpassung an ein Feld aus x-y-Datenpaaren

etwas schwieriger und ist in *Abb. 7.9.2* wiedergegeben. Der Korrelationskoeffizient (−1...+1) ist ein Maß dafür, wie gut sich an N Wertepaare x, y eine Gerade anpassen läßt; eine optimale Anpassung liegt bei −1 oder +1 vor. Die Eingabe der x- und y-Werte muß nicht numerisch geordnet erfolgen. Die Zeile 180 kann normalerweise entfallen, wenn man das Programm stets mit RUN startet.

7.10 Drucker plottet Funktionen

Einer mathematischen Funktion sieht man ihren Verlauf normalerweise nicht direkt an, und es ist deshalb recht hilfreich, sie irgendwie schwarz auf weiß ausgeben zu können. Das Programm in *Abb. 7.10* entstand mit einem am AIM-65 angeschlos-

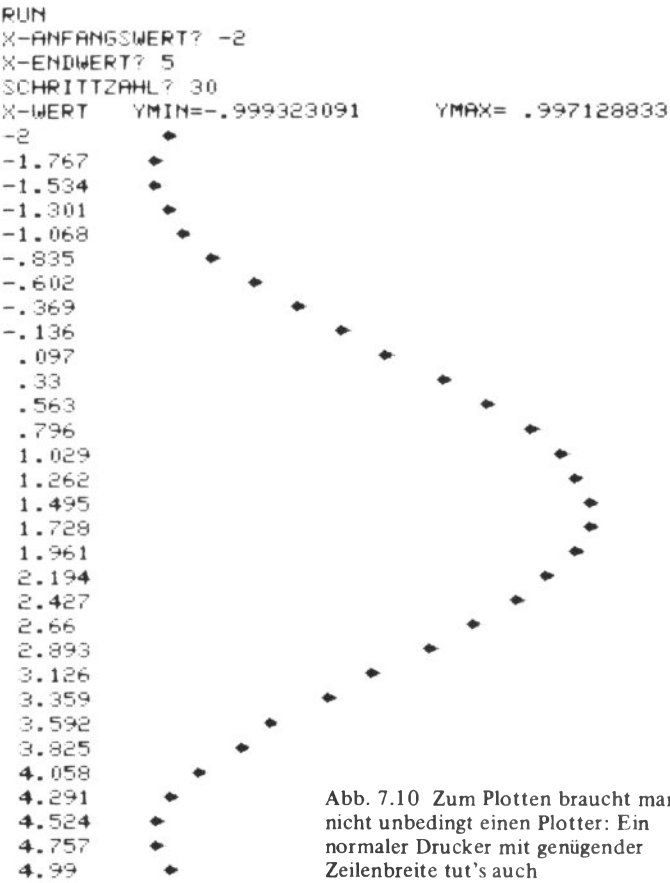


Abb. 7.10 Zum Plotten braucht man nicht unbedingt einen Plotter: Ein normaler Drucker mit genügender Zeilenbreite tut's auch

```
LIST

 1 REM TTY-PLOTTER                                    zu Abb. 7.10
 10 BR=30:REM PAPIERBREITE
 20 INPUT"X-ANFANGSWERT";X1
 30 INPUT"X-ENDWERT";X2
 40 INPUT"SCHRITTZAHL";W
 50 W=(X2-X1)/W
 60 X=X1:GOSUB500:Y1=Y:Y2=Y
 70 X=X1:REM Y-MIN/MAX ERMITTELN
 80 GOSUB 500
 90 IF Y<Y1 THEN Y1=Y
 100 IF Y>Y2 THEN Y2=Y
 110 IF X<=X2 GOTO 80
 120 PRINT"X-WERT    YMIN=";Y1;TAB(BR);"YMAX=";Y2
 130 X=X1:REM KURVE AUSDRUCKEN
 140 PRINT X;:GOSUB 500
 150 PRINT TAB((Y-Y1)/(Y2-Y1)*BR+10.5);
 160 PRINT"*":IF X<=X2 GOTO 140
 170 END
 500 Y=SIN(X):REM TESTFUNKTION
 600 X=INT((X+W)*1E3+.5)/1E3:RETURN
```

senen Matrixdrucker (Silent-700 von TI), wobei zu beachten ist, daß bei der Initialisierung des Basic-Interpreters ein passender Wert für "Line Width" (hier 64) einzugeben ist.

Das hier wiedergegebene Plotter-Programm verwendet in Zeile 150 den TAB-Befehl zur Ausgabe einzelner Sternchen als Funktionswerte. Im Gegensatz zu sonst üblichen Darstellungen wird die x-Koordinate hier senkrecht, die y-Achse waagrecht gedruckt. Für jedes Sternchen wird links der zugehörige x-Wert (in Zeile 600 auf drei Nachkomma-Stellen gerundet) ausgegeben.

Eine Besonderheit ist, daß der Benutzer keinen y-Maßstab oder -Bereich angeben muß, da das Programm diesen selbst in einem "Probedurchlauf" ermittelt und auch den minimalen und maximalen y-Wert im vorgegebenen x-Bereich ausgibt.

Die zu plottende Funktion steht in Zeile 500 und kann bei Bedarf auch mehrere Programmzeilen umfassen, z.B. um durch Abfrage von Bedingungen unstetige Funktionen darzu-

stellen. Allerdings muß sie stets eindeutig sein, d.h. zu einem x-Wert darf es nur einen y-Wert geben (ein geschlossener Kreis läßt sich also mit diesem Programm nicht plotten).

Die Definition der Variablen BR (Zeile 10) paßt den Ausdruck der jeweils verfügbaren Zahl von Zeichen pro Zeile an; selbstverständlich sieht die erzeugte Kurve umso besser aus, je größer diese Zahl beim verwendeten Drucker sein kann.

Es ist dem Benutzer überlassen, dafür zu sorgen, daß die in Zeile 500 stehenden Funktion zu keinen Fehlermeldungen des Computers führt (Division durch Null usw.).

7.11 Erzeugung von Lottozahlen

Die Erzeugung einer Folge von Zufallszahlen ohne Wiederholung, wie z.B. Lottozahlen, setzt voraus, daß sich der Computer "merkt", welche Zahlen bisher erzeugt wurden. In *Abb. 7.11* geschieht das durch ein Feld B(1...49), das man sich

```
<6>
LIST

 10 PRINT"♦♦♦ LOTTOZAHLEN ♦♦♦"
 20 DIM B(50):FOR I= 1 TO 7
 30 Z=INT(49♦RND(1)+1)
 40 IF B(Z)=1 GOTO 30
 50 B(Z)=1:NEXT I
 60 FOR I=1 TO 49
 70 IF B(I)=1 THEN PRINT I;
 80 NEXT I
RUN
♦♦♦ LOTTOZAHLEN ♦♦♦
 1  6  11  35  37  40  43
RUN
♦♦♦ LOTTOZAHLEN ♦♦♦
 10  15  21  25  27  32  43
RUN
♦♦♦ LOTTOZAHLEN ♦♦♦
 4  14  19  23  27  28  41
```

Abb. 7.11 Erzeugen von Zufallszahlen ohne Wiederholungen und deren numerisch sortierter Ausdruck

wie einen Lottoschein vorstellen kann; jede ermittelte Zahl wird hier durch Eintragen einer 1 in das entsprechende Matrixfeld 1...49 gekennzeichnet. Falls eine schon "angekreuzte" Zahl generiert wurde, so wird das in Zeile 40 festgestellt, und es erfolgt ein Sprung nach Zeile 30, um eine neue Zufallszahl zu erzeugen.

Der sortierte Ausdruck der sieben Lottozahlen erfolgt ab Zeile 60, wobei sozusagen alle angekreuzten Kästchennummern, also alle Matrix-Indices mit einer 1 im zugehörigen Element, ausgegeben werden.

Für den Leser ist es sicher ein leichtes, den Ausdruck so zu ändern, daß ein quadratischer Lottoschein mit Sternchen in den angekreuzten Kästchen entsteht.

7.12 Verwendung von DEFFN für neue Funktionen

Das kleine Demonstrationsprogramm in *Abb. 7.12* zeigt die Verwendung der Anweisung DEFFN. Viele Rechner lassen Variablen und damit auch Funktionsnamen mit drei und mehr

```
 998 REM DEMONSTRATION VON DEFFN
 999 GOTO1020:REM SIN+COS SCHON VORHANDEN
1000 DEFFN SIN(X)=TAN(X)/SQR(1+(TAN(X)^2))
1010 DEFFN COS(X)=1/SQR(1+(TAN(X)^2))
1020 DEFFN SEC(X)=SQR(1+(TAN(X)^2))
1030 DEFFN CSC(X)=SQR(1+(TAN(X)^2))/TAN(X)
1040 DEFFN ASN(X)=ATN(X/SQR(1-X*X))
1050 DEFFN ACS(X)=ATN(SQR(1-X*X)/X)
2000 REM *** TESTPROGRAMM ***
2010 INPUT X
2020 Y=FN CSC(X)
2030 PRINT"ERGEBNIS=";Y
RUN
? 1.4
ERGEBNIS= 1.01476511
```

Abb. 7.12 Definition zusätzlicher Funktionen mit der Anweisung DEFFN. Zu beachten ist, daß eine Funktion immer vor ihrem Aufruf definiert werden muß und keine reservierten Worte enthalten darf

Buchstaben zu, wobei allerdings nur die beiden ersten relevant sind. So ist es möglich, etwa den Arcussinus mit ACS zu bezeichnen und später mit FN ACS aufzurufen.

Zu beachten ist aber, daß Variablennamen keine "reservierten Worte" des Basic-Interpreters sein dürfen. Deswegen kann keine Funktion SIN definiert werden, wenn sie der Interpreter ohnehin schon enthält. Da das Programm auf einem AIM-65 entwickelt wurde, der SIN und COS beherrscht, wird die Definition dieser Funktionen mit GOTO 1020 übersprungen – andernfalls würde eine Fehlermeldung auftreten.

Nicht vergessen sollte man auch, daß jede Funktion stets *vor* ihrem Aufruf definiert werden muß – im Gegensatz zu DATA-Statements, die auch am Ende des Programms stehen können.

8 Ausblick

Dieses Buch widmete sich primär der Programmiersprache Basic – einer Computersprache, deren Bedeutung derzeit noch eher zunimmt als von neueren Konkurrenten, wie etwa Pascal, beeinträchtigt wird. Der Leser wird aber schon bald feststellen, daß sich nicht alle Programmierprobleme in Basic optimal lösen lassen. Es sei ihm daher dringend geraten, sich auch ein wenig mit der Maschinen- oder Assembler-Sprache seines Computers zu befassen, worüber schon eine Menge Literatur erschienen ist. Erst damit ist es möglich, z.B. Morsezeichen bei Geschwindigkeiten von 150 Buchstaben pro Minute und mehr zu decodieren, per Software Frequenzen bis zu 50 kHz zu messen oder zu erzeugen und ohne zusätzlichen Hardware-Aufwand z.B. einen Fernschreiber anzusteuern. Leider sind die Möglichkeiten zum Programmieren in Maschinensprache bei den heutigen Computern recht unterschiedlich – das eine Extrem ist wohl der AIM-65 alias PC-100, der ein 8-KByte-Monitorprogramm speziell hierfür besitzt, und das andere Extrem wird vom HP-85 repräsentiert, der jede Möglichkeit zur Maschinenprogrammierung gänzlich unterbindet.

Wegen der Systemabhängigkeit der Maschinensprache konnte sie in diesem Buch nicht näher behandelt werden, wenngleich sie eine wertvolle Ergänzung auch von Basic-Programmen darstellt: Routinen, die in Maschinensprache geschrieben sind, lassen sich auch innerhalb von Basic-Routinen aufrufen, um z.B. Aufgaben durchzuführen, bei denen es auf eine möglichst hohe Geschwindigkeit ankommt.

Beim Erscheinen neuer, leistungsfähigerer Computer sollte man sich darüber im klaren sein, daß auch den Möglichkeiten der bisher schon erhältlichen Geräte kaum Grenzen außer dem Einfallsreichtum des Programmierers gesetzt sind. Verbesserun-

gen betreffen meist in erster Linie die Hardware: In den nächsten Jahren wird es Personal Computer mit mehr Speicherplatz, höherer Arbeitsgeschwindigkeit und besserer Bedienbarkeit geben, die nicht mehr (aber auch nicht viel weniger) als heute kosten. Um eines kommen aber auch deren Besitzer nicht herum – nämlich um das Erlernen einer logisch-exakten, konsequenten und schrittweisen "Denkungsart" als elementare Voraussetzung für den Umgang mit dem Kollegen Computer.

9 Literatur

9.1 Allgemeine Einführungen

Osborne, A.: Mikrocomputer-Grundwissen. Te-Wi-Verlag, München.

Osborne, A.: Einführung in die Mikroprozessor-Technik. Te-Wi-Verlag, München.

Osborne, A.: An Introduction To Microcomputers (engl., Zusammenstellung aller heutigen Mikroprozessor-Typen). Te-Wi-Verlag, München.

Pelka, H.: Was ist ein Mikroprozessor? RPB 82, Franzis-Verlag, München.

Mikrocomputer – eine Einführung in kleinen Schritten. Sonderheft der FUNKSCHAU, Franzis-Verlag, München.

Dem Mikroprozessor aufs Bit geschaut – Bauanleitung für einen Mikrocomputer mit der CPU 2650. Sonderheft der ELO, Franzis-Verlag, München.

Hobbycomputer 1 (Systembeschreibungen, Grundlagen). Sonderheft von ELO, FUNKSCHAU und ELEKTRONIK. Franzis-Verlag, München.

9.2 Grundlagen der Programmiersprache Basic

Lien, D.A.: The Basic Handbook (engl.). Compusoft Publishing/Hofacker-Verlag, Holzkirchen.

Schneider, W.: Basic-Einführung für Techniker. Vieweg-Verlag, Wiesbaden.

Schneider, W.: Programmieren von Heimcomputern 1: Einführung in Basic. Vieweg-Verlag, Wiesbaden.

Schwill, W.D.; Weibezahn, R.: Einführung in die Programmiersprache Basic. Vieweg-Verlag, Wiesbaden.

Spencer, D.D.: Anleitung zum praktischen Gebrauch von Basic. Oldenbourg-Verlag, München.

Tracton, K.: The Basic Cookbook (engl.). Tab Books/Hofacker-Verlag, Holzkirchen.

9.3 Basic-Programmsammlungen

Lorenz, C.: Basic-Programmier-Handbuch. Hofacker-Verlag, Holzkirchen.
Lorenz, C.: PET-Programmier-Handbuch. Hofacker-Verlag, Holzkirchen.
Tracton, K.; Lorenz, C.: 57 praktische Basic-Programme. Hofacker-Verlag, Holzkirchen.
Hobbycomputer 2. Sonderheft der FUNKSCHAU, Franzis-Verlag, München.
Programme für Kleincomputer und Taschenrechner. Sonderheft der FUNKSCHAU, Franzis-Verlag, München.
The First Book of 80-US (engl., TRS-80). Hofacker-Verlag, Holzkirchen.

9.4 Maschinensprache-Bücher

Camp, R.C.; Smay, T.A.; Triska, C.J.: Microprocessor Systems Engineering (engl., AIM-65/PC-100). Matrix Publishers/Interface Age Europe, München.
Feichtinger, H.: Anwendungsbeispiele für den Mikroprozessor 6502. RPB 173, Franzis-Verlag, München.
Forster, C.C.: Programming A Microcomputer (6502, engl.). Addison-Wesley Publications, Micro-Shop Bodensee.
Klein, M.: Z-80-Applikationsbeispiele. Franzis-Verlag, München.
Osborne, A.: Assembly Programming (engl.; mehrere Bände für 8080, 8085, 6800, Z-80, 6502). Te-Wi-Verlag, München.
Programme für Kleincomputer und Taschenrechner. Sonderheft der FUNKSCHAU, Franzis-Verlag, München.
Hobbycomputer 2. Sonderheft der FUNKSCHAU, Franzis-Verlag, München.

9.5 Hardware-Bücher

Bacher, W.; Grunow, D.; Schierenbeck, F.: Datenübertragung – Eigenschaften der Verbindungswege, Technik und Geräte. Siemens-Verlag, München.
Lesea, A.; Zaks, R.: Mikroprozessor-Interface-Techniken. Sybex/Micro-Shop Bodensee.
Osborne, A.: Some Real Support Devices (engl., Peripherie-Bausteine). Te-Wi-Verlag, München.
Pelka, H.: Praxis mit Mikroprozessoren (Bauanleitung für ein 8080-System). Franzis-Verlag, München.
Care and Feeding of your Commodore PET (engl. Service-Anleitung). Hofacker-Verlag, Holzkirchen.
IEC-Bus-Handbuch. Hofacker-Verlag, Holzkirchen.

Sachverzeichnis

Herwig Feichtinger
Basic für
Mikrocomputer

Das kleine Praktikum

Dieses praxisorientierte Buch ist Einführung und Nachschlagewerk zugleich: Begriffe aus der Computer-Fachsprache wie ASCII, RS-232-Schnittstelle oder IEC-Bus werden ebenso ausführlich erläutert wie alle derzeit üblichen Befehlsworte der Programmiersprache Basic. Marktübliche Basic-Rechner werden einander gegenübergestellt – einerseits, um vor dem Kauf die Qual der Wahl zu erleichtern, andererseits, um das Anpassen von Programmen an den eigenen Rechner zu ermöglichen. Und schließlich findet der Leser handfeste Tips für das Erstellen eigener Programme und Beispiele fertiger Problemlösungen für typische Anwendungsfälle.

Der Autor befaßt sich seit etwa 1975 mit der Mikrocomputer-Technik. Im Rahmen seiner Tätigkeit als FUNKSCHAU-Redakteur arbeitete er mit den meisten der hier beschriebenen Geräte selbst, was einen objektiven Vergleich möglich machte.

ISBN 3-7723-6821-2

F'