Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

# computer kurs

## Heft 50


Natürliche Sprache

Immer indirekter

Schönschreibübung

Die Betriebssysteme

Projektplanung

Ein wöchentliches Sammelwerk

# computer kurs

## Heft 50

## Inhalt



### WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

Computer Kurs ist ein wöchentlich erscheinendes Sammelwerk. Die Gesamtzahl der Hefte ergibt ein vollständiges Computer-Nachschlagewerk. Damit Sie jede Woche Ihr Heft erhalten, bitten Sie Ihren Zeitschriftenhändler, Computer Kurs für Sie zu reservieren.

**Zurückliegende Hefte**
Ihr Zeitschriftenhändler besorgt Ihnen gerne zurückliegende Hefte. Sie können sie aber auch direkt beim Verlag bestellen.

**Deutschland:** Das einzelne Heft kostet DM 3,80. Bitte füllen Sie eine Postzahlkarte aus an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Computer Kurs

**Österreich:** Das einzelne Heft kostet öS 30. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs, Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Computer Kurs.

**Schweiz:** Das einzelne Heft kostet sfr 3,80. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

### Abonnement

Sie können Computer Kurs auch alle 2 Wochen (je 2 Ausgaben) per Post zum gleichen Preis im Abonnement beziehen. Der Abopreis für 12 Ausgaben beträgt DM 45,60 inkl. Mwst., den wir Ihnen nach Eingang der Bestellung berechnen. Bitte senden Sie Ihre Bestellung an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk Service, Postgiroamt Hamburg 86853-201, Postfach 105703, 2000 Hamburg 1, Kennwort: Abo Computer Kurs. Bitte geben Sie an, ab welcher Nummer das Abo beginnen soll und ob Sie regelmäßig für jeweils 12 Folgen einen Sammelordner wünschen.

**WICHTIG: Bei Ihren Bestellungen muß der linke Abschnitt der Zahlkarte Ihre vollständige Adresse enthalten, damit Sie die Hefte schnell und sicher erhalten. Überweisen Sie durch Ihre Bank, so muß die Überweisungskopie Ihre vollständige Anschrift gut leserlich enthalten.**

**SAMMELORDNER**
Sie können die Sammelordner entweder direkt bei Ihrem Zeitschriftenhändler kaufen (falls nicht vorrätig, bestellt er sie gerne für Sie) oder aber Sie bestellen die Sammelordner für den gleichen Preis beim Verlag wie folgt:

**Deutschland:** Der Sammelordner kostet DM 12. Bitte füllen Sie eine Zahlkarte aus an: Marshall Cavendish International Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Sammelordner Computer Kurs.

**Österreich:** Der Sammelordner kostet öS 98. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Sammelordner Computer Kurs

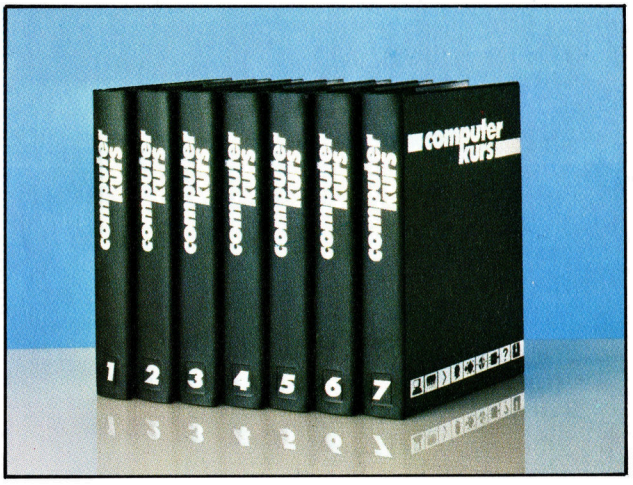
**Schweiz:** Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

### INHALTSVERZEICHNIS

Alle 12 Hefte erscheint ein Teilindex. Die letzte Ausgabe von Computer Kurs enthält den Gesamtindex — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.


**Redaktion:** Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

**Vertrieb:** Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1




**Druck:** E. Schwend GmbH, Schmollerstraße 31, 7170 Schwäbisch Hall


**Jeder Computer hat irgendeine Art von Betriebssystem – ein Programm, das die Funktionen des Computers sowie alle ans System angeschlossenen Geräte steuert.**

# Allzeit bereit

**Das Betriebssystem ist ein lebenswichtiger Teil jedes Computers, da es die Verbindung zwischen Hard- und Software herstellt. Hier wird erläutert, für welche Aufgaben das Betriebssystem verantwortlich ist.**

Durch den Einsatz von Hochsprachen muß sich der Programmierer heute nicht mehr um die interne Arbeitsweise der Zentraleinheit kümmern. Der Interpreter oder Compiler, der den Quellcode der Hochsprache verarbeitet, übernimmt die Speicheradressierung und dergleichen mehr. Der Hochsprachen-Interpreter oder -Compiler ist ein Programm und muß, bevor er den Quellcode in Objektcode-Anweisungen umwandelt, in den Hauptspeicher geladen werden. BASIC-Sprachen auf ROM-Basis dagegen sind ständig im Speicher enthalten und können sofort nach Einschalten der Maschine benutzt werden.

Zusätzlich sind andere sogenannte „Hintergrund"-Programme für die interne Arbeit des Computers erforderlich. Nehmen wir einmal das Beispiel, daß ein auf der Tastatur geschriebener Brief auf dem Bildschirm erscheinen soll. Irgendwo im Speicher muß ein Programm vorhanden sein, das der Zentraleinheit die Anweisung gibt, ständig zu überprüfen, ob eine Taste gedrückt wurde. Ist dies der Fall, muß das Programm feststellen, um welche Taste es sich handelt, und dann eine entsprechende Anweisung an den Videogenerator geben, das richtige Punkt-Rastermuster zu erzeugen – in der richtigen Reihenfolge – und es auf dem Bildschirm auszugeben. Aktivitäten dieser Art werden als „für den Benutzer transparent" bezeichnet.

## Überwachungsprogramm

Das Betriebssystem ist das Hintergrundprogramm, das alle anderen Funktionen überwacht. Handelt es sich bei dem Computer um ein kleineres System auf ROM-Basis mit eingebautem BASIC, kommt es bei dieser Betrachtung leicht zu Verwirrungen, da BASIC und Betriebssystem meist in demselben ROM enthalten sind. In seiner einfachen Form enthält ein internes ROM alle für den Betrieb des Systems erforderliche Software außer den Applikationsprogrammen (Textverarbeitung usw.). Ein Teil dieses ROMs enthält die Codes, die für die Umwandlung der in BASIC geschriebenen Programme in Maschinensprache erforderlich sind (dies ist der Interpreter). Ein anderer Teil enthält die Codes, die für Eingabe und Modifi-

zierung der von Anwendern geschriebenen Programme erforderlich sind (dies ist der Editor). Und ein weiterer Teil enthält die interne Verwaltungssoftware, die die Tastatur steuert, Grafik und Zeichen darstellt, Daten von Cassette annimmt und sie den richtigen Speicherteilen zuweist (das ist der Monitor).

Der Begriff „Monitor" – nicht zu verwechseln mit einem Fernseher oder Sicht-Monitor – ist eine weitere Bezeichnung für das „Betriebssystem". Ein einfacher Monitor kann nur Instruktionen in Maschinensprache akzeptieren, sie der richtigen Speicheradresse zuweisen und ihre Ausführung überwachen. Fortgeschrittenere Systeme weisen dem Monitor die Rolle eines richtigen Betriebssystems zu.

Ein anderes Extrem sind Computer auf Diskettenbasis, die oft in Büros Anwendung finden und mit leistungsfähigen Betriebssystemen ausgestattet sind. Bevor wir uns mit Computern der dazwischenliegenden Gruppe wie etwa dem Apple befassen, sehen wir uns zunächst einmal an, welche Art von Betriebssystem für einen Computer auf Diskettenbasis benötigt wird.

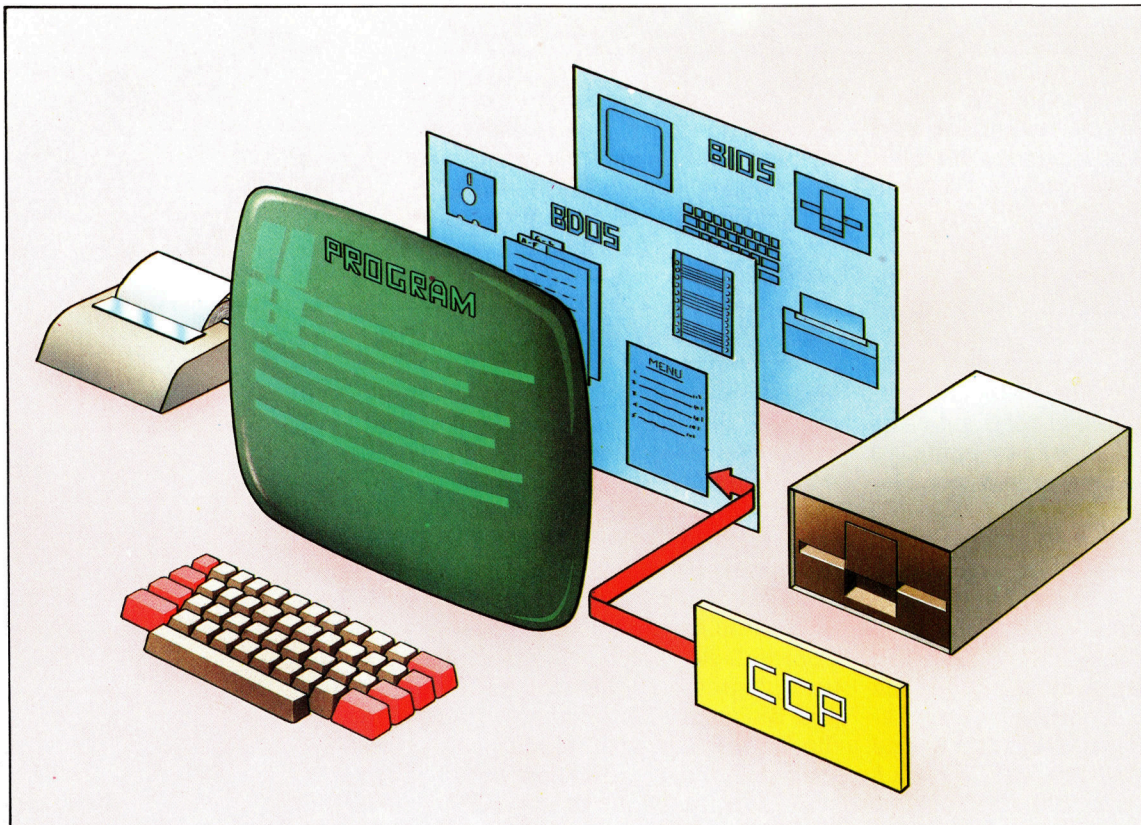
Bei einem derartigen Computersystem ist im ROM außer dem Lader und einigen Verwaltungsroutinen normalerweise nichts gespeichert. Schaltet man den Computer ein, enthält der Lader Maschinenanweisungen, um der Zentraleinheit zu sagen, wie die Diskettenstation anzusteuern und das Betriebssystem in den Hauptspeicher zu laden ist.

Das ins RAM geladene Betriebssystem muß mehr leisten als das Betriebssystem der Systeme auf ROM-Basis. Das Disketten-Betriebssystem erweitert die normalen internen Verwaltungsfunktionen um Befehle, die direkt auf die Files einwirken, die auf der Diskette enthalten sind. Darunter sind eine ganze Anzahl Befehle, die die Namensauflistung der gespeicherten Files ermöglichen, Files löschen oder umbenennen können oder Files von Diskette in den Hauptspeicher oder auf andere Disketten kopieren können.

## Adressenerkennung

Eine derartige File-Verwaltung ist mit Betriebssystemen auf ROM-Basis im allgemeinen nicht möglich. Diese verfügen lediglich über einen einfachen Befehl, mit dem ein benanntes File von Band geladen oder auf Band gespeichert werden kann. Hochentwickelte Betriebssysteme kennen die exakte File-Adresse auf Diskette oder Band. Manche Systeme sind jedoch so begrenzt, daß sie lediglich die vorhandenen Files so lange durchsuchen, bis der gesuchte Name auftaucht. Erst dann wird geladen. Oder das File wird einfach ab dem Punkt auf Band geladen, an dem der Schreibkopf gerade angelangt ist.

„Dazwischenliegende" Computersysteme, wie etwa der Acorn B, bearbeiten sowohl Cassetten- als auch Disk-Files unter Verwendung weitgehend identischer Befehle. Das Betriebssystem befindet sich in einem separaten ROM. Man muß es sich als ein Betriebssystem vor-




## Arbeitskontrolle

CP/M ist in „Lagen" konstruiert, um sowohl sich selbst als auch die dafür geschriebenen Programme leicht von einem Rechner auf den anderen übertragbar zu machen. Das BDOS (Basic Disk Operating System) enthält eine Reihe von Routinen, die Standardfunktionen und Ausführungen für das Programm beinhalten. Soll das BDOS eine bestimmte Aufgabe erfüllen, ruft es das BIOS (Basic Input/Output System) auf. Dieses ist speziell dem jeweiligen Computer angepaßt und stellt die Software-Verbindung zwischen BDOS und den Elementen eines Computers dar.

CP/M schließt CCP ein, das dem Benutzer ermöglicht, Befehle einzugeben, mit denen Programme laufen und Files umbenannt und gelöscht werden.

stellen, das aus einer externen Speichereinheit in den Speicher geladen wird. Seine Funktionen umfassen sehr hoch entwickelte File-Verarbeitungen.

Das Betriebssystem eines Computers ist somit als ein Programm zu verstehen, das zwischen dem Benutzer und dem restlichen Rechnersystem, einschließlich Zentraleinheit, System-Software (wie etwa Programmiersprachen) und Applikationen, liegt.

Die Möglichkeit, Software auf mehr als nur einem Computersystem zu benutzen, ist als „Portabilität" bekannt (Übertragbarkeit). Das ist unter zwei Gesichtspunkten zu sehen. Zum einen erfordern unterschiedliche Prozessoren unterschiedliche Anweisungen, um gleichwertige Operationen ausführen zu können. Das Problem, den Hochsprachen-Code in den richtigen Maschinencode umzuwandeln, ist vom verwendeten Interpreter oder Compiler zu „lösen". Für jede Zentraleinheit müssen daher auch andere Interpreter und Compiler geschrieben werden.

Zum anderen gibt es das Problem der Software-Portabilität. Selbst bei Verwendung derselben Zentraleinheit entstehen Schwierigkeiten. Im Videospeicher werden u. U. unterschiedliche Adressen eingesetzt, für die Cursorbewegung auf dem Bildschirm benutzt man unterschiedliche Adressen, bedient sich verschiedener Eingabe/Ausgabe-Adressen und so weiter.

Um dieses Problem zu lösen, wurden allgemein verwendbare Disketten-Betriebssysteme entwickelt, mit denen beispielsweise das Laufen jedweder Software, die für einen Z80-Computer geschrieben wurde, auf einem anderen Z80-Computer gewährleistet ist. Bestes Beispiel für ein derartiges Betriebssystem ist CP/M (Control Program/Microcomputers).

## Portabilitätsvorteile

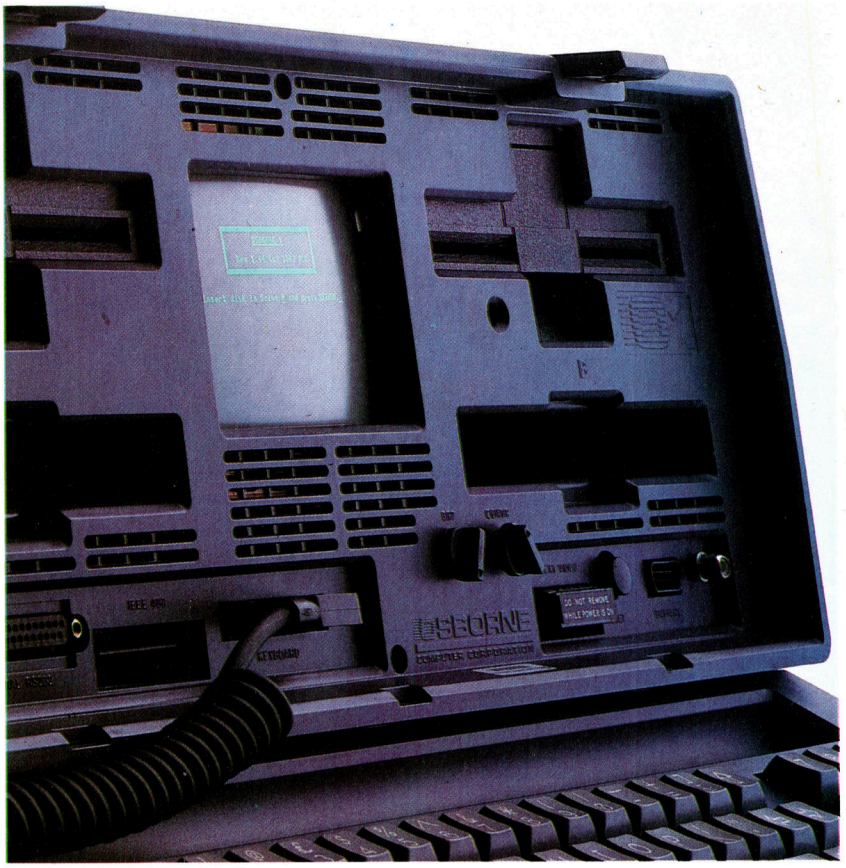
Diese Diskettenbetriebssysteme wurden aufbauend auf maschinenspezifische Monitore und Betriebssysteme entwickelt. Sie erzielten in Sachen Software-Portabilität einen beachtlichen Fortschritt. Ein Programm, das sich nicht maschinenspezifischer Eigenarten (wie zum Beispiel Soundeffekte) eines Rechners bedient und für ein allgemeines Betriebssystem wie CP/M oder MS/DOS geschrieben wurde, ist auf jedem Computer mit diesem System lauffähig. Die Betriebssystem-Software als solche wird in standardisierter Form an die Computerhersteller geliefert. Der Hardware-Hersteller muß lediglich einen kleinen, maschinenspezifischen Teil des Programms umschreiben.

In Umfang und Leistungsvermögen unterscheiden sich Diskettenbetriebssysteme erheblich. Systeme wie CP/M oder MS-DOS aber umfassen generell drei Teile: den Befehls-Prozessor, das Basic-DOS (BDOS) und das Basic-Eingabe/Ausgabe-System (BIOS). Der für die Portabilität relevante Teil ist das BIOS. Beim BIOS handelt es sich um einen speziellen Programmteil, der alle Routinen zur Steuerung der Peripheriegeräte einschließlich Bildschirm und Tastatur enthält und deshalb für jeden Rechner neu konfiguriert werden muß. Jedes Programm, das von diesem Betriebssystem gesteuert wird, hat ein BIOS als Schnittstelle zum Computer. Das BIOS holt zum Beispiel Zeichen vom Keyboard, gibt Zeichen auf Bildschirm oder Drucker aus, adressiert die Diskettenstation und so weiter.

Das BDOS besteht aus Teilen des Betriebssystems, die nicht gerätespezifisch sind (etwa generalisierte Routinen für Bildschirmverwaltung, Drucker, Diskettenstationen). Diese Programmteile müssen nicht verändert werden. BDOS und BIOS korrespondieren generell mit dem Monitor oder dem Betriebssystem von Computern auf ROM-Basis.

Der Befehlsprozessor ist der Teil des Programms, der Betriebssystem-Anweisungen steuert, die über die Tastatur eingegeben wurden. Typische Funktionen dieser Art sind das Laden von Diskette in den Hauptspeicher, die Auflistung der File-Namen oder Umbenennung der File-Namen.

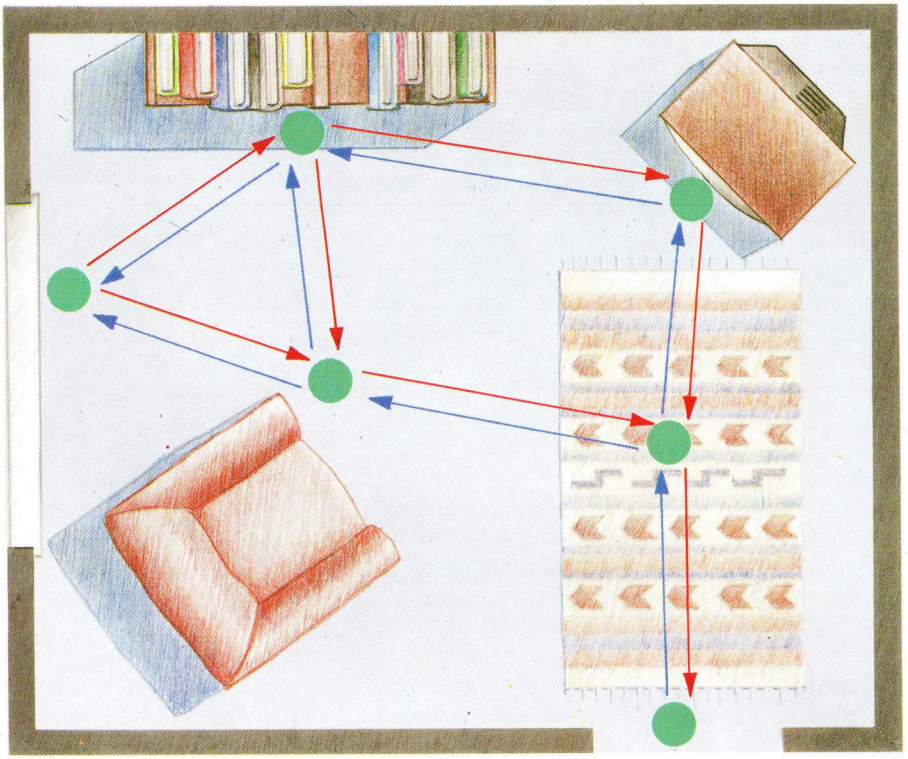
Da Betriebssysteme im Hintergrund arbeiten, wird ihre Bedeutung oft übersehen. Doch sie sind wesentlicher Bestandteil jedes Computersystems. Deshalb ist das Verständnis ihrer Arbeitsweise so wichtig.




**Der Osborne 1 verdankte seinen Erfolg dem Umstand, daß es sich um einen CP/M-Computer handelte. Er wurde mit den populärsten CP/M-Programmen wie WordStar, SuperCalc und MBasic geliefert.**

# Selbststeuerung

**Am Ende unserer PROLOG-Serie untersuchen wir, wie die Sprache ihre eigenen Programme als Daten einsetzt. In unserem Beispiel simulieren wir die Bewegung eines Roboters durch einen Raum.**



Zwei Gründe veranlaßten die Japaner, PROLOG als zentrale Sprache für das Projekt der fünften Computergeneration auszuwählen. Zum einen sind die Ausdrücke von PROLOG dem Format der „relationalen" Datenbanken ähnlich (eine hochentwickelte Datenbankmaschine soll die Grundlage des Systems sein).

Zum anderen eignet sich PROLOG ideal für Entwicklungen im Bereich der Künstlichen Intelligenz, da die Sprache ihre eigenen Programme als Daten verwenden kann und sein Interpreter den Schlußfolgerungsmaschinen ähnelt, die die modernen Expertensysteme einsetzen.

Unser Beispiel zeigt, wie bei der Simulation von Roboterbewegungen Programmbefehle als Daten eingesetzt werden. Der Raum wird durch Gegenstände definiert, die durch Wege verbunden sind. Unser Roboter soll dabei Befehlen gehorchen wie: „Gehe vom Fernseher zum Stuhl und finde dabei die kürzeste Strecke. Behandle zunächst das Ziel **gehe(Dort)** (das heißt, gehe von der augenblicklichen Position auf dem kürzesten Weg nach **Dort**)."

## Syntaxfragen

Das Programm enthält zwei Sätze für **gehe(Dort).** Der erste berücksichtigt den Fall, daß der Roboter bereits **Dort** ist. Wir benötigen dafür in der Datenbank die Tatsache (bei(tuer)), die die augenblickliche Position des Roboters angibt. Der zweite Satz findet mit dieser Tatsache die Anfangsposition und ruft die Prozedur **gehe(Platz1, Platz2)** auf.

Da **gehe(Platz1,Platz2)** getrennt von **gehe(Platz)** eingesetzt werden kann, überprüft die Prozedur zunächst, ob sie über **Hier** und **Dort** Bescheid weiß und ob **Hier** die augenblickliche Position ist. Wen sich einer der Tests nicht beweisen läßt, verläßt PROLOG den aktuellen Satz und geht zum nächsten. Der zweite Satz **gehe(Hier,Dort)** bewegt den Roboter nach **Hier,** wenn er sich nicht bereits auf dieser Position befindet. Dabei wird **gehe(Hier)** aufgerufen und, nachdem dieser Vorgang erfolgreich beendet wurde, **gehe(Dort).** Die Vorgänge dieses Satzes lassen sich folgendermaßen lesen: „Um von **(Hier)** nach **(Dort)** gehen zu können, wenn die augenblickliche Position nicht bereits **(Hier)** ist, zuerst nach **(Hier)** und dann nach **(Dort)** gehen. Der dritte Satz für **gehe(Hier,Dort)** gibt eine Fehlermeldung aus, wenn die beiden Plätze nicht bekannt sind.

Nachdem das erste **gehe(Hier,Dort)** erfolgreich war, ist das nächste Teilziel nun **findall.** Diese Aussage ist in manche PROLOG-Versionen bereits eingebaut. Wenn nicht, kann sie wie bei FORTH leicht neu definiert werden. **findall** hat drei Argumente: einen Variablennamen, ein Ziel und eine Listenvariable (RL).

Bei **findall** versucht PROLOG, das gegebene Ziel zu beweisen – in diesem Fall **plan(Hier,Dort,[],Weg).** Wenn die Variable des ersten Arguments mit einer der Zielvariablen übereinstimmt, wird der aktuelle Wert dieser Variable zu der Liste im dritten Argument hinzugefügt.

Das Ziel **plan(Platz1,Platz2,[],Weg)** würde einen Weg zwischen **Platz1** und **Platz2** finden und ihn in die Variable **Weg** setzen. In unserem Programm „sammelt" **findall** daher alle Wege, die es zwischen **Hier** und **Dort** gibt, und speichert sie in eine Liste.

Beachten Sie, daß die Prozedur **findall** an diesem Punkt noch nicht definiert sein muß. Wegen des deklarativen Stils von PROLOG können alle Prozeduren „von oben nach unten" geschrieben werden, wobei zuerst die Hauptziele und später die Einzelheiten angegeben werden.

**Bewegungsfreiheit**
Die Umgebung des Roboters ist als ein Satz von Plätzen definiert, die er „kennt" – das heißt, sie sind als Tatsachen in seine Datenbank eingetragen. Die Plätze sind durch Wege miteinander verbunden, auf denen sich der Roboter bewegen kann. Sie sind in dem Format „weg(tv,buecherschrank)" etc. geschrieben. Das Simulationsprogramm zeigt, wie sich der Roboter mit diesem Wissen einen Weg von A nach B zusammenstellen kann.

Die Aussage **kuerzest(RL,Kurzer Weg)** wurde noch nicht definiert. Ihre Aufgabe ist es, aus einer Liste von Listen **(Listel)** die neue Liste **(Liste2)** zu bilden, die nur aus der kürzesten Liste besteht.

Unser Roboter hat nun den kürzesten Weg zu seinem Ziel geplant – er muß ihn nur noch zurücklegen. Die „Dummy"-Routine **bewegen** zeigt die zurückgelegten Wege an. Die Ziele können aber statt der Anzeige auch andere Programme aufrufen. Die Aussage **ansehen(Platz)** könnte als Abtasten definiert sein, das **Platz** feststellen soll und den Roboter umdreht, während **bewegen(Platz1,Platz2)** den Roboter von einem Platz zum anderen steuert.

## Bewegungs-„plan"

Das Programm für die Bewegung wird vom Hauptprogramm selbst in das Innere der Prozedur **plan** geschrieben. Der Algorithmus lautet: Um einen Weg von A nach B finden zu können, ermittle zuerst einen Weg von A nach C. Die Prozedur **plan** ist recursiv. Der erste Satz beendet die Recursion, wenn das Ziel erreicht ist.

Der zweite Satz erledigt die eigentliche Arbeit. Er besagt, daß es einen Plan gibt, der über den Weg R von A nach B führt, wenn:
1) ein Weg von A nach C existiert **und**
2) C nicht in der Liste der bereits besuchten Plätze (V) steht (zu der Liste wird C hinzugefügt, damit sich der Roboter nicht im Kreis bewegt) **und**
3) es einen Plan gibt, um über den Weg R1 von C nach B zu gehen.

Wenn all diese Bedingungen wahr sind, dann ist der endgültige Weg R eine Programmliste, die den Weg von A nach C beschreibt. Diese Liste wird an den bisher zurückgelegten Weg R1 angefügt.

Für den Abschluß des Programms müssen wir nur noch die augenblickliche Position des Roboters aktualisieren. Diese Aufgabe führen die beiden eingebauten Aussagen **retract** und **assert** aus. **retract(X)** nimmt den ersten Satz, der mit X übereinstimmt, aus der Datenbank heraus, während **asserta(X)** den Satz X als ersten dieses Typs in die Datenbank einfügt **(assertz(X)** fügt X als letzten Satz ein). Selbststeuerung ist schon eine recht komplizierte Sache, wenn man sie auf die beschränkte Logik einer Maschine bringen will – jeder Schritt will erklärt sein.

**Simulation von Roboterbewegungen**
**Standard-PROLOG:**

```
bei(tuer).              /* die aktuelle Position des Roboters */
gehe(Dort):—            bei(Dort),write('Ich bin bereits dort'),nl,nl,nl.
gehe(Dort):—            bei(Hier),gehe(Hier,Dort).

gehe(Hier,Dort):—       platz(Hier),platz(Dort),bei(Hier),findall(Weg,plan(Hier,Dort,[],Weg),RL),
                        kuerzest(RL,KurzerWeg),bewegen(KurzerWeg),retract(bei(Hier)),
                        asserta(bei(Dort)),sagwo.
gehe(Hier,Dort):—       not(bei(Hier)),write('Ich bin nicht bei')write(Here),write('und werde
                        daher zuerst dorthin gehen'),nl,nl,gehe(Hier),gehe(Dort).
gehe(Hier,Dort)         write('Ich kann nur Plaetze besuchen, die ich kenne'),nl,nl.
plan(A,A,_,R).
plan(A,B,V,R):—         weg(A,C),not(member(C,V)),append([C],V,V1),plan(C,B,V1,R1),append
                        ([ansehen(C),bewegen(A,C)],R1,R).

bewegen(Weg):—          write(Weg),nl,nl. /* wird spaeter definiert */
sagwo:—                 bei(Platz),write('Ich bin bei'),write(Platz),nl,nl,nl.

/* Wegliste, die der Roboter kennt */ /* Liste der Plätze, die er kennt */

weg(buecherregal,stuhl).        platz(tuer).
weg(stuhl,buecherregal).        platz(teppich).
weg(buecherregal,tv).           platz(tv).
weg(tv,buecherregal).           platz(buecherregal).
weg(tv,teppich).                platz(fenster).
weg(teppich,tv).                platz(stuhl).
weg(teppich,stuhl).
weg(stuhl,teppich).
weg(tuer,teppich).
weg(teppich,tuer).
weg(fenster,stuhl).
weg(stuhl,fenster).
weg(fenster,buecherregal).
weg(buecherregal,fenster).
```

**Micro-PROLOG:**

```
(bei tuer)          /* die augenblickliche Position des Roboters */

((gehe X)           (bei X)(P Ich bin bereits dort!)PP,PP,PP)
((gehe(X)           (bei Y)(gehe Y X))

((gehe X Y)         (platz X)(platz Y)(bei X)
                    (findall Z (plan XY()Z)x)
                    (kuerzest x y)
                    (bewege y)
                    (DELCL ((beiX)))(ADDCL ((bei Y)))
                    (sagwo))
((gehe X Y)         (NOT bei X)
                    (P Ich bin nicht bei)(PX)
                    (P und werde daher zuerst dorthin gehen.)
                    PP PP (gehe X)(gehe Y))
((gehe X Y)         (P Ich kann nur Plaetze besuchen,
                    die ich kenne) PP PP)
(plan XX x1 Z)
((plan X Y x Z)     (weg X X1)(NOT member X1 x)(append X1)
                    x x1) (plan X1 Y x1 Z1)
                    (append ((ansehen X1)(bewegen X X1)) Z1 Z))

((bewegen 2)        (P Z) PP PP)
((sagwo)            (bei X) (P Ich bin bei X) PP PP PP)
/* Wegliste, die der Roboter kennt */ /* Liste der Plätze, die er kennt */

(weg buecherregal stuhl)        (platz tuer)
(weg stuhl buecherregal)        (platz teppich)
(weg buecherregal tv)           (platz tv)
(weg tv buecherregal)           (platz buecherregal)
(weg tv teppich)                (platz fenster)
(weg teppich tv)                (platz stuhl)
(weg teppich stuhl)
(weg stuhl teppich)
(weg tuer teppich)
(weg teppich tuer)
(weg fenster stuhl)
(weg stuhl fenster)
(weg fenster buecherregal)
(weg buecherregal fenster)
```

**Programmierte Manöver**
Der Anwender gibt die Informationen über die Umgebung des Roboters in die Datenbank von PROLOG ein. PROLOG stellt sich nun mit der Prozedur gehe(platz1,platz2) einen Weg von A nach B zusammen.

Beachten Sie den Gebrauch der Unterstreichung (_) als Argument im Inneren eines Satzplanes (A,A,_,R). PROLOG setzt dieses Symbol als „Wildcard" ein, die bei der Ausführung des Satzes keinen Wert erhält. In diesem Fall ist diese Wildcard Teil eines Satzes, der zeigen soll, daß es immer einen Weg von A nach A gibt.

# Die Wegweiser

**Der verfügbare Speicherplatz in Microcomputern wird größer. Damit entsteht auch Raum für eine anwenderfreundliche Gestaltung des Programms. Wir stellen einige Hilfsroutinen vor, mit denen Sie Ihre Programme ergänzen und verbessern können.**

Die neue Heimcomputer-Generation verfügt bereits über ein Minimum von 128 KByte RAM. Die meisten Hobby-Programmierer nutzen jedoch nur einen Bruchteil dieser Kapazität. Früher war der Speicherplatzmangel häufig eine glaubhafte Erklärung für die unbefriedigende Unterstützung des Anwenders durch Anleitungen, exakte Fehlermeldungen und Hilfsroutinen – heute kann sich niemand mehr mit diesem Argument entschuldigen.

Die wichtigsten Hilfen innerhalb eines Programms lassen sich in drei Gruppen aufteilen: Anweisungen, „Help"-Seiten und „Wegweiser". Anweisungen können zwei verschiedene Formen haben: Sie stehen entweder als kompakter Block am Anfang des Programms oder erscheinen je nach Bedarf während des Programmablaufs. Am besten ist es, wenn dem Anwender beide Möglichkeiten offenstehen.

In ihrer einfachsten Version sind Anweisungen eine oder mehrere Textseiten, die in leicht verständlicher Form den Umgang mit dem Programm erläutern. Der Text kann als String oder DATA-Anweisung im Programm enthalten sein und wird im Bedarfsfall mit Hilfe eines speziellen Unterprogramms auf den Bildschirm gebracht. Zu Beginn des Hauptprogramms wird der Anwender gefragt, ob er Anweisungen wünscht – falls er dies bejaht, wird das entsprechende Unterprogramm aufgerufen. Programmteile, die Tastatureingaben erfordern, sollten so gestaltet sein, daß die Eingabe von "?" oder "A" einen Aufruf des Anweisungs-Unterprogramms erzeugt. Vorteilhaft ist auch ein standardisiertes Kommando für „Anweisungen auflisten", das in alle Programme zur Dateneingabe integriert wird. Dazu müssen Sie natürlich alle Prompts ändern: Aus „Weiter mit beliebigem Tastendruck" wird dann „Weiter mit beliebigem Tastendruck oder mit A für Anweisungen".

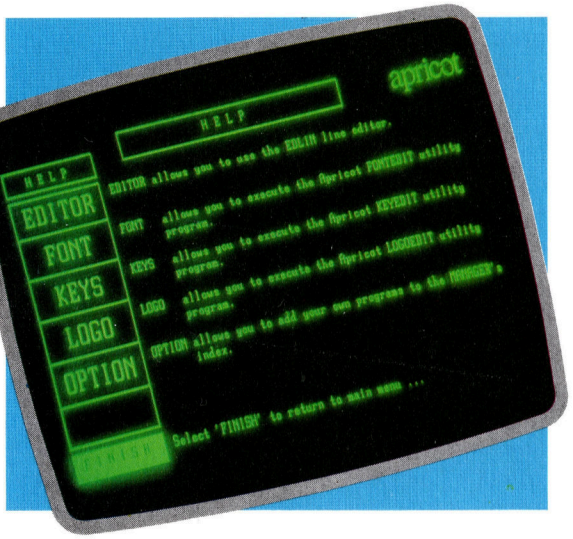
Anweisungen müssen nicht nur aus Text bestehen – sie können auch Diagramme, Beispiele und Übungen enthalten. In Programmen für wissenschaftliche Experimente darf der Anwender erst mit dem Hauptprogramm arbeiten, wenn er seine Fähigkeiten an Übungsaufgaben trainiert und bewiesen hat. Ein Training dieser Art ist nicht ganz einfach zu programmieren, weil es nicht nur den Ablauf des Hauptprogramms simulieren, sondern auch die Leistung des „Kandidaten" bewerten muß. Der Versuch kann aber lohnen – er vermittelt ein sicheres Gefühl für mögliche Probleme beim Umgang mit dem Hauptprogramm.

**Das Textverarbeitungsprogramm WordStar ist eines der meistverkauften Programme mit jederzeit abrufbaren Hilfsoptionen. Der Anwender kann den „Help-Level" selbst bestimmen oder das Menü auch ganz vom Bildschirm entfernen. Auf Tastendruck steht der detailreiche Inhalt der Help-Files sofort wieder zur Verfügung.**

Eine andere Möglichkeit ist der Aufruf von „Help"-Informationen zur Erklärung bestimmter Programmteile – meist wird die Funktion bestimmter Befehle verdeutlicht. Im Betriebssystem Unix steht sogar das komplette Handbuch jederzeit für den Abruf auf den Bildschirm bereit! Es ist recht einfach, auch in selbstgeschriebenen Programmen Hilfsfunktionen anzubieten: Immer wenn eine Eingabe

fällig ist, sollte auch nach „Help" gefragt werden können. Steht ein Diskettenlaufwerk zur Verfügung, können die Help-Informationen dort in separaten Files abgelegt sein. Die Help-Routine muß dann nur noch aus der Eingabe den richtigen Filenamen generieren, das File laden und auf dem Bildschirm anzeigen.

Die Rückkehr zum Hauptprogramm – und zwar genau zu der Stelle, an der es verlassen wurde – sollte jederzeit möglich sein. Das Help-Unterprogramm muß dazu ein Flag setzen (und später auch wieder löschen!), damit





**Gutes Management**

**Die Manager-Software des ACT von Apricot führt den Anwender mit einer hierarchisch geordneten Struktur sicher durch eine Vielzahl schwieriger Programme. Auf den in jedem Menü verfügbaren Help-Befehl hin werden alle Optionen genau erklärt – ein gutes Beispiel für die Unterstützung durch umfangreiche Help-Files.**

der Rücksprung zum letzten Befehl vor Aufruf des Unterprogramms erfolgt.

Für den Anwender ähnelt der Umgang mit einem komplexen Programm dem Versuch, in einem verwinkelten Labyrinth den Ausgang zu finden. Dabei kann insbesondere der Anfänger schnell die Orientierung verlieren. – Er braucht Wegweiser, wie sie das Menü eines Programms anbietet: Ähnlich den Schildern an einer Kreuzung zeigt das Menü alle Möglichkeiten, den erreichten Knotenpunkt wieder zu verlassen. Auch beim Apple Macintosh wird ein ähnliches Verfahren genutzt – hier zeigen „Icons" die Menüfunktionen.

## Zugriffsbereite Befehle

Die erlaubten Befehle können – je nach der im Programm erreichten Stelle – unterschiedlich wichtig sein. Bei einem kleinen Befehlssatz kann es günstig sein, die Bedeutung der einzelnen Optionen mit ein bis zwei Zeilen zu erläutern. Manche Befehle – etwa QUIT (Programmabbruch) – müssen jederzeit zugänglich sein und sollten daher auch ständig im Display angezeigt werden. Auch die Befehle UNDO, SAVE und weitere anwenderspezifische Optionen sollten ständig zugriffsbereit sein. Dazu belegt man die Funktionstasten mit diesen Befehlen und zeigt deren Wirkung einzeilig auf dem Bildschirm an. Eine weitere Hilfe bietet ein ständig sichtbarer Wegweiser, der aus dem Programm herausführt. Besonders Anfänger können damit beruhigt werden – sie haben so den „Notausgang" jederzeit in Reichweite.

Es gibt Versuchssysteme, welche die Geschicklichkeit des Anwenders überwachen und entsprechend die Hilfsstufe variieren. Intelligente Programme dieser Art sind zwar noch nicht handelsüblich, einen Schritt in diese Richtung können sie aber auch mit einfachen Mitteln tun – etwa den Anwender bei jedem Programmstart nach seinem Namen fragen, der gemeinsam mit seinen spezifischen Daten gespeichert wird (etwa, wie häufig er das Programm schon benutzt oder wie viele Punkte er in einem Spielprogramm bisher erzielt hat). Wenn diese Daten nach jedem Programmablauf aktualisiert würden, ließe sich danach die Menge und Ausführlichkeit der angebotenen Hilfsoptionen steuern. Im Idealfall würde man die bei WordStar verwendete Möglichkeit, den „Help Level" vom Anwender selbst definieren zu lassen, mit einer solchen „Help-Level-Automatik" kombinieren.

Durch gute Bedienerführung und Hilfsoptionen kann nahezu jedes Programm verbessert werden. Help-Routinen können auch dazu beitragen, ein Programm weiter zu perfektionieren: Dazu wird die Häufigkeit des Aufrufs jeder Help-Seite gespeichert – man weiß so nach kurzer Zeit, wo die Schwachpunkte des Programms liegen.

# Kollisionskurs

**In den ersten beiden Abschnitten des Projekts wurden Routinen zur Darstellung des Spielfelds entworfen. Jetzt befassen wir uns mit der Kontrolle der Bewegungen über die Tastatur sowie dem Programmteil, der Kollisionen zwischen Spielfiguren und Minen feststellt.**

Das Acorn-BASIC verfügt über vier Befehle zur Tastaturabfrage, die abhängig von der Anwendung gewählt werden. INKEY$ und INKEY werden normalerweise verwendet, um eine bestimmte Zeit auf einen eventuell erfolgenden Tastendruck zu warten, bevor die Programmausführung fortgesetzt wird. GET$ und GET stoppen die Programmausführung, bis eine Taste gedrückt wird. GET$ und GET werden primär verwendet, wenn eine Frage wie „Neues Spiel J/N?" beantwortet werden soll. Im obigen Beispiel sind die einzigen möglichen Antworten „J" oder „N". Mit Hilfe einer REPEAT. . .UNTIL-Anweisung kann die GET-Anweisung wiederholt werden. Beispiel:

```
1000 PRINT "ANOTHER GAME Y/N?"
1010 REPEAT
1020 A$=GET$
1030 UNTIL A$="Y"OR A$="N"
```

Die nach GET$ oder INKEY$ gedrückte Taste wird wie im obigen Beispiel als String behandelt. Bei Verwendung von GET oder INKEY wird ein numerischer Wert anstelle eines String-Wertes ausgegeben, der dem ASCII-Code der gedrückten Taste entspricht. *FX4,1 beispielsweise bewirkt, daß die ASCII-Codes der Cursortasten ausgegeben werden. In diesem Fall haben die Tasten folgende Werte:

| | |
|---|---|
| Linkspfeil | 136 |
| Rechtspfeil | 137 |
| Abwärtspfeil | 138 |
| Aufwärtspfeil | 139 |

Angenommen, in unserem Programm sollen nur der Links- und Rechtspfeil als Eingabe akzeptiert werden. Im nachstehenden Programmteil wird mittels INKEY eine Viertelsekunde auf eine Eingabe gewartet:

```
1000 *FX4,1:REM TURN ON CURSOR ASCII
     MODE
1010 REPEAT
1020 A=INKEY(25)
1030 UNTIL A=136 OR A=137
1040 *FX4,0:REM RESTORE CURSOR TO
     EDIT MODE
```

Parameter 25 in Zeile 1020 bewirkt, daß 25 Hundertstelsekunden gewartet wird, bevor das Programm wieder wie vorher weiterläuft.

Diese Anweisungen überprüfen die Tastatur nicht direkt, beeinflussen jedoch den „Tastatur-Buffer". Dabei handelt es sich um einen Temporärspeicher für über Tastatur eingegebene Zeichen. Neu eingegebene Zeichen werden ans Buffer-Ende gesetzt, während der Prozessor die Zeichen am Buffer-Anfang verarbeitet. Werden Zeichen schneller eingegeben, als sie vom Prozessor verarbeitet werden können, sind sie also nicht verloren, sondern werden aus dem Tastatur-Buffer abgerufen. Da INKEY, GET, INKEY$ und GET$ für gewöhnlich nur den Buffer-Anfang testen, „wissen" Sie nicht, wie lange ein bestimmtes Zeichen bis zu seiner Verarbeitung im Buffer aufbewahrt wurde. In tastaturgesteuerten Spielen kann dadurch der Ablauf erheblich verlangsamt werden, daß das Programm frühere Tasteneingaben ausführt, während der Spieler bereits neue Eingaben macht.

Die eine Lösungsmöglichkeit ist, den Tastatur-Buffer vor dem Testen mit *FX15,1 zu löschen. Die zweite Möglichkeit ist die Verwendung einer INKEY-Variation. Wie oben beschrieben, wartet INKEY ( ) eine bestimmte Zeit ab, daß eine Taste gedrückt wird, bevor mit dem Programm fortgefahren wird. Wird INKEY eine negative Zahl in Klammern nachgestellt, wird die Tastatur anstelle des Tastatur-Buffers getestet. Jeder Taste wurde für diesen Zweck eine negative Zahl zugeordnet (die vollständige Wertetabelle finden Sie im Acorn-Handbuch). In unserem Programm werden Bewegungen mittels der Cursortasten gesteuert. Die Werte der Tasten in Kombination mit INKEY sind:

| | |
|---|---|
| Linkspfeil | – 26 |
| Rechtspfeil | –122 |
| Abwärtspfeil | – 42 |
| Aufwärtspfeil | – 58 |

Nachfolgende Prozedur überprüft nacheinander alle vier Cursortasten. Drückt man eine Cursortaste, wird eine neue Prozedur („move") aufgerufen, die zwei Parameter annimmt. Die Parameter enthalten exakte Angaben, wie das Minensuchgerät bewegt werden soll.

```
3000 DEF PROCtest_keyboard
3010 REM ** UP ? **
3020 IF INKEY(-58)=-1 THEN PROCmove(0,-1)
```

```
3030 REM ** DOWN ? **
3040 IF INKEY(-42)=-1 THEN PROCmove(0,1)
3050 REM ** RIGHT ? **
3060 IF INKEY(-122)=-1 THEN PROCmove(1,0)
3070 REM ** LEFT ? **
3080 IF INKEY(-26)=-1 THEN PROCmove(-1,0)
3090 ENDPROC
```

Mit der Move-Prozedur werden das Suchgerät und die Assistenten bewegt und Kollisionen mit Minen überprüft. Betrachten wir zuerst einmal den Prozedur-Teil für die Steuerung der Zeichen-Bewegung.

Aus „test-keyboard" werden zwei Parameter an „move" übertragen, die für die Variablen delta-x und delta-y in „move" verwendet werden und der Veränderung der X- und Y-Koordinaten des Suchgeräts entsprechen. Wird beispielsweise der Aufwärtspfeil gedrückt, werden die Werte 0 und −1 an „move" übertragen. Mit xdet=xdet+delta−x und ydet=ydet+delta−y werden die Koordinaten des Suchgeräts aktualisiert. Beim Drücken des Aufwärtspfeils wird zu xdet der Wert 0 addiert und ydet um −1 verringert. Da sich der Ursprung der Zeichenposition in der oberen linken Bildschirmecke befindet und die Y-Werte nach unten gezählt werden, wird eine Bewegung nach oben ausgeführt. Denn die Reduzierung von ydet um 1 bewirkt eine Bewegung um 1 nach oben. Dieses System kann problemlos auch für diagonale Bewegungen verwendet werden. Eine Eingabe der Werte (1,−1) in „move" bewegt das Suchgerät diagonal um je eine Einheit nach oben und rechts. Dennoch müssen wir uns an diesem Punkt mit weiteren Tasten für die Diagonalbewegung befassen. Nachfolgend das Listing für die „move"-Prozedur:

```
3220 DEF PROCmove(delta_x,delta_y)
3230 REM ** RUB OUT OLD POSITIONS **
3240 COLOUR 1
3250 PRINTTAB(xdet,ydet);" "
3260 PRINTTAB(xman,yman);" "
3270 REM ** MOVE DETECTOR **
3280 xdet=xdet+delta_x
3290 ydet=ydet+delta_y
3300 REM ** TEST FOR LIMITS **
3310 IF xdet>17 THEN xdet=17
3320 IF ydet>25 THEN ydet=25
3330 IF xdet<2 THEN xdet=2
3340 IF ydet<1 THEN ydet=1
3350 REM ** CALCULATE MAN'S COORDS **
3360 xman=19-xdet
3370 yman=26-ydet
3380 PROCconvert(xman,yman)
3390 IF POINTxgraph,ygraph)=2 THEN PROCexplode(xgraph,ygraph)
3400 PROCconvert(xdet,ydet)
3410 IF POINT(xgraph,ygraph)=2 THEN PROCfound_mine
3420 PROCposition_chars
3430 ENDPROC
```

Vor Veränderung der X- und Y-Koordinaten des Suchgeräts müssen zuerst die alten Positionen von Suchgerät und Assistent gelöscht werden. Die Zeilen 3250 und 3260 verwenden die alten Werte von xdet, ydet, xman und yman, um die alten Zeichen mit Leerzeichen zu überschreiben. Da die neuen Zeichen in Rot gedruckt werden sollen (logische Farbe 1), wird in Zeile 3240 die aktuelle Vordergrundfarbe auf 1 gesetzt. Die Zeilen 3280 und 3290 aktualisieren die Koordinaten des Suchgeräts. Bevor dieses mit PRINT an seine neue Position gebracht wird, muß sichergestellt werden, daß keine außerhalb des definierten Minenfelds liegenden Koordinaten erhöht oder vermindert werden. Die obere und untere Grenze von xdet und ydet werden in den Zeilen 3310 und 3340 getestet. Hier wurde festgelegt, daß das Suchgerät bei Erreichen einer Begrenzung anhält, bis es in die entgegengesetzte Richtung bewegt wird. In Zeile 3310 wird beispielsweise überprüft, ob das rechte Ende des Feldes, also die X-Koordinate 17, erreicht wurde. Es wäre auch möglich, einen „Wrap-around"-Effekt zu erstellen, bei dem das Suchgerät nach Übertreten eines Randes an der gegenüberliegenden Bildschirmseite wieder auftauchen würde. Hierzu muß Zeile 3310 folgendermaßen geändert werden:

```
3310 IF xdet >17 THEN xdet=2
```

Sie können die vier Begrenzungstests in einer Weise abändern, daß der Wrap-around-Effekt für alle Ränder gilt.

## Spiegelverkehrte Bewegung

Eine Spielregel lautet, daß alle Bewegungen des Suchgeräts durch den Assistenten spiegelverkehrt vollzogen werden. Dazu müssen die Koordinaten des Assistenten, die über eine einfache Formel in Zeile 3360 und 3370 von den Koordinaten des Suchgeräts abhängen, automatisch aktualisiert werden. Um zu sehen, wie damit Spiegelbild-Bewegungen erstellt werden, betrachten wir den Zusammenhang zwischen den X-Koordinaten (xman=19−xdet).

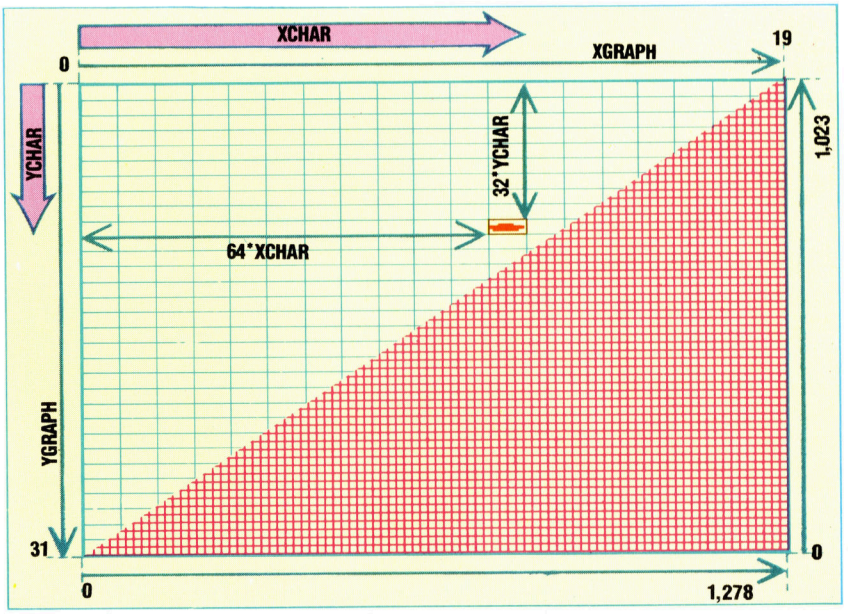
Anfangs enthält xdet den Wert 2 und xman 17. Durch Bewegen des Suchgeräts nach rechts erhöht sich xdet auf 3. Nach obenstehender Formel errechnet sich xman aus 19−3=16. Das heißt, der Assistent bewegt sich nach links. Bewegt sich xdet erneut nach rechts, wird xdet 4 und xman 15 usw. Die Y-Koordinaten arbeiten ähnlich.

Beim Acorn B kann jeder Bildschirmpunkt auf seine Farbe hin überprüft werden. POINT(X,Y) gibt die Farbe des Pixels an Position (X,Y) aus.

**Im Spiel werden hochauflösende Grafiken mit Text kombiniert. Das hat Vorteile, bedeutet aber auch eine Mischung zweier verschiedener Koordinatensysteme für Text und Grafiken. Der Acorn B hat mehrere unterschiedliche Textformate mit eigenen Koordinatensystemen. Das Spiel arbeitet im Modus 5, in dem 20 Zeichen horizontal und 32 vertikal dargestellt werden können.**

**Es stehen drei verschiedene Grafikauflösungen zur Verfügung, die alle mit demselben Koordinatensystem arbeiten. Alle Modi werden behandelt, als hätten sie eine Auflösung von 1280 auf 1024.**

**Das Programm benutzt zum Lesen von Punkten aus einer Textanzeige in niedriger Auflösung das hochauflösende Koordinatensystem. Dabei wird die horizontale Koordinate (im Programm XCHAR) mit 64 und die vertikale Koordinate (YCHAR) mit 32 multipliziert. Es gibt noch ein anderes Problem: Textkoordinaten beginnen bei Null am oberen Bildschirmrand und werden nach unten gezählt, während die Grafikkoordinaten mit Null am unteren Bildschirmrand beginnen und nach oben gezählt werden. Das Problem wird gelöst, indem 32*YCHAR von 1023 abgezogen wird.**



Damit läßt sich testen, ob das angesteuerte Pixel grün ist (Grün = Mine). Es ist jedoch zu beachten, daß POINT(X,Y) den ausgewählten Punkt mit einem hochauflösenden Koordinatensystem spezifiziert. Wollen wir diesen Befehl für unser Spiel verwenden, müssen zuerst alle Zeichen-Koordinaten in Grafik-Koordinaten umgewandelt werden.

Im letzten Kapitel wurde die Breite und Höhe der Grafikeinheiten in Modus 5 mit 64 und 32 festgelegt. Eine Multiplikation von xchar mit 64 ergibt die xgraph-Koordinate der betreffenden Zellenecke. Ein Addieren von 32 zu xgraph ergibt die X-Koordinate des Zellenmittelpunktes.

## ygraph-Berechnung

Die Berechnung von ygraph ist wegen des gegensätzlichen Verlaufs der beiden Systeme komplizierter. Am oberen Bildschirmrand ist ygraph 1023. Bewegen wir uns mit 32*ychar nach unten, kommen wir an den oberen Rand der angegebenen Zeile; eine weitere Bewegung um 16 Einheiten bringt uns an die Y-Koordinate des Zellenzentrums. Die nachfolgende Prozedur kann für die Umwandlung von Zeichen-Koordinaten in Grafik-Koordinaten verwendet werden:

```
3720 DEF PROCconvert(xchar,ychar)
3730 xgraph=64*xchar+32
3740 ygraph=1023-(32*ychar+16)
3750 ENDPROC
```

Der echte Wert, mit dem Parameter zwischen Prozeduren übertragen werden können, ist aus der move-Prozedur zu ersehen. Die convert-Prozedur wird zweimal verwendet. Einmal dient sie zur Konvertierung der Assistenten- in Grafik-Koordinaten, wobei diese Werte in Zeile 3390 für die Prüfung der Farbe Grün verwendet werden. Ist die aktuelle Farbe Grün, springt das Programm für die Explosionsdarstellung in eine andere Prozedur. „Convert" wird außerdem in Zeile 3400 für die Berechnung der Zeichen-Koordinaten des Suchgeräts verwendet. Zeile 3410 testet, ob sich in der Zeile eine Mine befindet. Falls ja, wird die Prozedur „foundmine" aufgerufen. Zuletzt werden Suchgerät und Assistent durch Aufrufen von „position-chars" mit PRINT an ihren neuen Positionen dargestellt.

Im nächsten Kapitel dieser Serie wird die Prozedur „explode" beschrieben. Bis dahin wird die nachfolgende Test-Prozedur an dieser Stelle eingefügt:

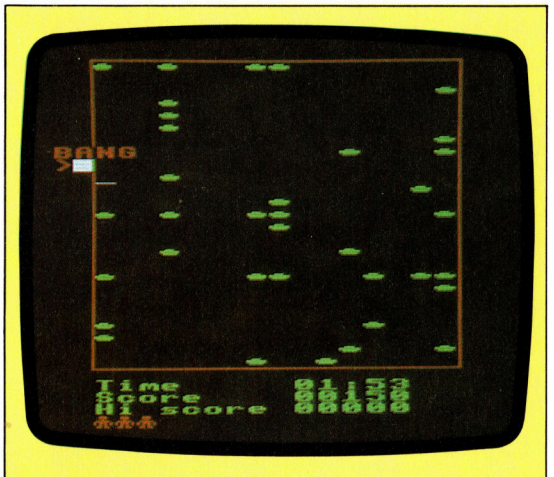
```
3550 DEFPROCexplode(x-explode,y-explode)
3560 PRINT"BANG"
3570 END
3580 ENDPROC
```

Zuletzt betrachten wir heute die Prozedur „foundmine". Der Fund einer Mine soll später durch einen Ton angezeigt werden. Für den Moment sollten Sie nur wissen, daß die SOUND-Anweisung in Zeile 3790 einen hohen Ton produziert. Die Hauptfunktion dieser Routine ist jedoch die Erhöhung des Punktestands. Für die Wertung werden zwei Variablen verwendet, wobei die erste eine numerische Variable ist, die um 150 erhöht wird. Um die Wertung immer als fünfstellige Zahl darzustellen, müssen Zusatz-Nullen an die numerischen Werte angefügt werden. Dazu muß zuerst der numerische Wert der Wertung in eine String-Variable umgewandelt werden, um dann mittels String-Handling-Techniken die zusätzlichen Nullen anzufügen. Die vollständige Prozedur lautet:

```
3770 DEF PROCfound_mine
3780 REM ** SOUND EFFECT **
3790 SOUND 2,-15,170,3
3800 REM ** INCREMENT SCORE **
3810 COLOUR 2
3820 score=score+150
3830 score$=STR$(score)
3840 score$=LEFT$(zero$,5-LEN(score$))+score$
3850 PRINTTAB(11,28);score$
3860 ENDPROC
```

Im letzten Kapitel entwickelten wir ein kurzes Programm zum Aufrufen der bisher erstellten Prozeduren. Diese Prozeduren können mit den gezeigten Zeilennummern in das Programm übernommen werden. Jetzt muß lediglich die Prozedur „test-keyboard" aufgerufen werden, um das Programm zum Laufen zu bringen. Fügen Sie folgende Zeile ein:

55 PROCtest-keyboard




**„BANG" für TREFFER**

**An diesem Punkt unseres Minenfeld-Projekts werden Minen auf dem Bildschirm verteilt. Sie können das Zeichen- und Spiegelbild erstellen, Wertungstabellen definieren und Bewegungen ausarbeiten. Da das Programm noch nicht komplett ist, erscheint bei einer Kollision mit einer Mine nur das Wort „BANG" auf dem Bildschirm. Im nächsten Kapitel wird eine Routine für die Darstellung einer Explosion mit Klangeffekten ausgearbeitet. Es könnte auch sein, daß Sie an bestimmten Punkten innerhalb des Spieles Fehlermeldungen erhalten, die aus der Unvollständigkeit des Programms resultieren. Nach Fertigstellung des Programms im nächsten Kapitel dürften diese Meldungen nicht mehr erscheinen.**

# Simultan gelenkt

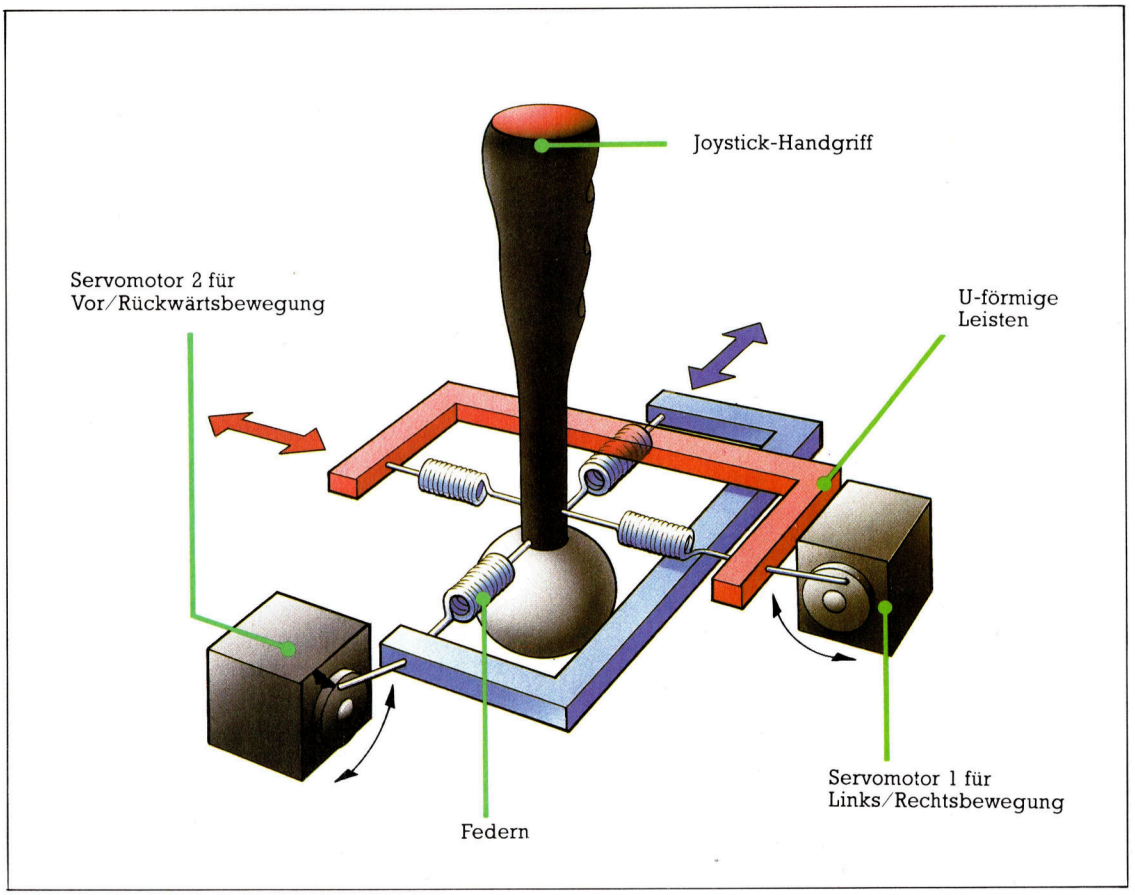
**Im vorigen Teil des Selbstbau-Kurses wurde erklärt, wie ein Servomotor gesteuert werden kann. Dieser Kursabschnitt enthält die für eine solche Simultansteuerung nötigen Programme.**

Theoretisch können Sie bis zu acht Motoren über den User Port betreiben. Der selbstgebaute Ausgangsbuffer hat allerdings nur vier Anschlüsse, die man für die Ansteuerung benutzen kann. Somit ist ein zweiter Ausgangsbuffer notwendig.

Zur Ansteuerung der acht Motoren können wir eine modifizierte Version des zuvor entwikkelten Algorithmus für die Einzelmotor-Steuerung verwenden. Alle Impulse werden gleichzeitig gegeben, die zweite Warteschleife wird durch eine Datentabelle ersetzt. Für die Tabelle sind 255 Speicherplätze reserviert, die alle auf einen Anfangswert von 255 ($FF) gesetzt werden.

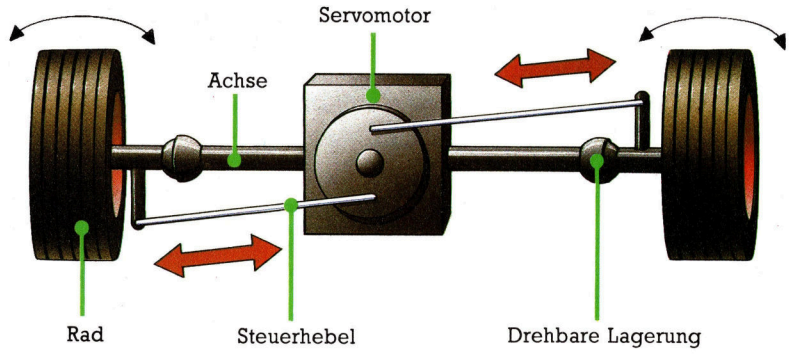
Ausnahmen von diesem Wert – zum Anhalten eines Motors – werden über die Datentabelle eingegeben. Soll Datenleitung 2 nach 20 Impulsen abgeschaltet werden, dann wird der zwanzigste Eintrag in der Tabelle von binär 11111111 in 11111011 geändert. Das erreichen wir durch eine AND-Verknüpfung mit dem Zahlenwert in der Tabelle. Nach der Eingabe der acht abweichenden Werte in die Datentabelle wird die Warteschleife gestartet. Jedes Element wird aber erst nach dem AND zum User Port weitergeleitet.

Der Algorithmus für die Simultansteuerung:
1) Winkelposition aller Motoren in acht Byte speichern (ANGLE+0 bis ANGLE+7).
2) Alle Datenbits am User Port auf High setzen, um die Impulse gleichzeitig zu geben.
3) Abweichungen in Datentabelle einfügen.
4) Eine Millisekunde warten.
5) 11111111 ($FF) in den Akkumulator laden. Dann nacheinander jedes Element der Datentabelle durch AND mit dem Akkumulatorinhalt verknüpfen. Bei Abweichungen schaltet das betreffende Bit auf Low. Da das AND bis ans Ende der Tabelle weitergeht, bleibt dieses Bit bis zum Schluß auf Low.
6) Abweichende Werte in der Datentabelle wieder auf binär 11111111 zurücksetzen.
Beide Acorn-B-Programme nutzen die gleiche Startroutine (Zeile 10 bis 280). Das erste Programm steuert einen Motor über eine der Da-




**Normalerweise übermitteln Joysticks ausschließlich Richtungsinformationen, die vom Programm ausgewertet werden. Eine Einsatzmöglichkeit für Servomotoren könnte die Rückkopplung vom Programm zum Joystick sein: Zwei Servomotoren würden mit dem Handgriff verbunden und ließen sich in einem guten Flugsimulator-Programm etwa dazu verwenden, der Hand des „Piloten" je nach Flugsituation mehr oder weniger Widerstand entgegenzusetzen. Ein durch Rückkopplung bewegter Joystick spricht den Tastsinn an und macht die Simulation damit fast perfekt.**

## Steuermechanismus




Unser Selbstbau-Robotauto wird durch die unabhängige Regelung zweier Schrittmotoren in verschiedene Richtungen gelenkt. Es geht aber auch anders: In der abgebildeten Version sorgt ein Motor für den Antrieb und ein zweiter für die Steuerung. Dazu ist der Servomotor direkt über der Achse montiert und verstellt über zwei Steuerhebel die Räder entsprechend. Eine zweite Möglichkeit wäre die Radsteuerung mit einem auf der Motorachse montierten Ritzel und Zahnstangen.

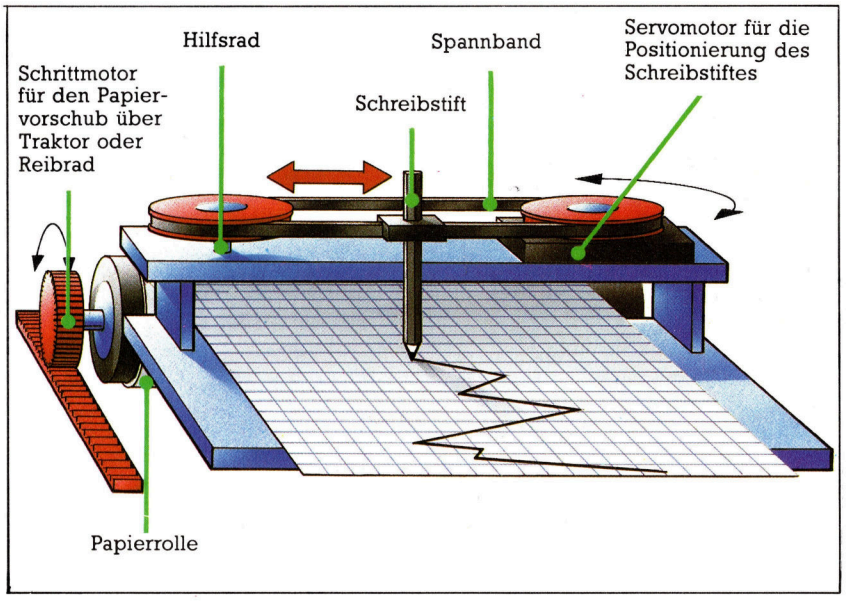
tenleitungen des User Port. Die Zeitsteuerung wird vom BASIC aus eingerichtet.

Das Betriebssystem des Acorn B verfügt über einen Timer für Hundertstelsekunden. Er wird auf zwei Hundertstelsekunden gesetzt. Durch Einsatz des „Event"-Vektors (siehe Acorn-Handbuch) verzweigt der Prozessor im richtigen Augenblick zum Impuls-Befehl des Programms. Da das Betriebssystem die Events unterstützt, kann das Unterprogramm durch Return (RTS) wieder verlassen werden. Ein RTI ist dazu nicht notwendig.

Das zweite Programm für die Simultansteuerung baut die Datentabelle mit den $FF-Werten durch eine BASIC-Schleife auf. Wenn jedes Element der Tabelle einzeln zum User Port ausgegeben würde, wäre jeder Impuls zwei Millisekunden lang. Abweichungen davon sind möglich, und jeder Motor läßt sich unabhängig steuern: Die Impulse werden durch einen zur Länge des Impulses proportionalen Offset im X-Register verändert. Die Tabelle wird Element für Element durch indirekte Adressierung an den User Port ausgegeben. Der Prozessor addiert den Wert des Registers Y zur Adresse in einem Zero-Page-Vektor.

Die Bit-Muster (Veränderungen) zur Motorsteuerung über den User Port werden folgendermaßen erzeugt: Motor 7 wird durch die Binärzahl 01111111 abgeschaltet, Motor 6 stoppt entsprechend bei 10111111, Motor 5 bei 11011111 usw. Dazu wird $FF im A-Register gespeichert und das Carry Flag (Übertragsflag) gelöscht.

**Ein Meßwertschreiber kann aus einem Servomotor für die Bewegungen des Schreibstiftes und einem Schrittmotor für den Papiervorschub aufgebaut werden. Die Rotation des Servomotors wird dazu über einen Spannband-Antrieb in die linearen Bewegungen des Stiftes umgesetzt. In professionellen Geräten hängt der jeweilige Ausschlag meist von bestimmten Meßwerten ab – etwa von der Temperatur oder dem Luftdruck. Der gleichmäßige Papiervorschub stellt oftmals die Projektion des entsprechenden Zeitablaufs dar.**




## Bitverschiebung

Unser Programm bearbeitet die Änderungen nacheinander, wozu alle Bits um eine Stelle nach rechts verschoben werden: Im ersten Schritt wird das Übertrags-Bit zu Bit 7, Bit 7 wird Bit 6 und so weiter. Bit 0 wird zum neuen Übertrags-Bit. Die Bit-Muster zum Abschalten der einzelnen Motoren werden im Stack zwischengespeichert. Jeder Wert der Datentabelle wird durch AND mit dem vorher zum User Port ausgegebenen Wert verknüpft – dadurch bleibt ein ausgeschalteter Impuls auf „Aus".

Die Tastatursteuerung ist ganz einfach: Die „Shift"-Taste zusammen mit einer Ziffertaste zwischen 1 und 8 wählt den angesprochenen Motor, ein Druck auf die Tasten 1 bis 9 ohne Shift bestimmt die Motorposition.

Das Programm für den Acorn B eingeben und mit SAVE speichern, danach mit RUN starten. Beide Programme (für Einzelmotor- und Simultansteuerung) arbeiten mit der gleichen Initialisierungsroutine (Zeile 10 bis 750).

Beim Commodore werden für den zweiten Algorithmus dieselben Steuertasten wie beim Acorn verwendet. Durch das BASIC-Aufrufprogramm lassen sich die Motorpositionen einzeln ansteuern (Tastenkombination s. o.).

Wenn Sie einen Assembler haben, muß das Quellprogramm eingetippt und in ein Objektfile assembliert werden, das schrittweise vom Aufrufprogramm geladen wird. Alternativ dazu wird der BASIC-Lader für das Maschinenprogramm eingegeben und mit RUN gestartet. Vor dem Laden und Starten des Aufrufprogramms NEW eingeben! Wenn Sie den BASIC-Lader verwenden, können Zeile 30 und 40 wegfallen.

# Commodore 64

## Simultansteuerung für Servomotoren

## Quellprogramm

```
1000 ;++++++++++++++++++
1010 ;++++++++++++++++++
1020 ;++              ++
1030 ;++ CBM MULTIPLE ++
1040 ;++ SERVO CONTROL++
1050 ;++              ++
1060 ;++++++++++++++++++
1070 ;++++++++++++++++++
1080 ;
1090 PORT = 56577 ;USER PORT DATA REGISTER
1100 ANGLE=12288  ;ANGLE VALUE LOCATION
1110 ZPAGE=$FB    ;0 PAGE POINTER TO TABLE
1120 ;
1130 *=$0334
1140 ;
1150       SEI        ;INTERRUPTS OFF
1160       LDA $0314 ;EXISTING IRQ VECTOR
1170       LDX $03C4
1180       STA $03C4
1190       STX $0314
1200       LDA $0315
1210       LDX $03C5
1220       STA $03C5
1230       STX $0315
1240 ;
1250 ;++++ INITIALISE TABLE ++++
1260 ;
1270       LDA #$FF
1280       LDY #$00
1290 TABLE
1300       STA (ZPAGE),Y
1310       DEY
1320       BNE TABLE
1330       CLI        ;INTERRUPTS ON
1340       RTS
1350 ;
1360 ;++++ EVENT HANDLER ++++
1370 ;
1380       PHP
1390       PHA        ;SAVE REGISTERS
1400       TYA        ;ON STACK
1410       PHA
1420       TXA
1430       PHA
1440 ;
1450 ;++ START PULSE, FOR SOME MOTORS IT
1460 ;MAY BE POSSIBLE TO START BEFORE FILLING
1470 ;TABLE AND SO REDUCE WAIT LOOP BELOW ++
1480 ;
1490       LDA #$FF
1500       STA PORT
1510 ;++ FILL TABLE WITH EXCEPTIONS ++
1520       LDX #$07
1530       LDA #$FF
1540       CLC
1550 EXCEPT
1560       ROR A
1570       PHA        ;BIT PATTERN
1580       LDY ANGLE,X ;GET MOTOR X OFFSET
1590       AND (ZPAGE),Y ;KEEP EXISTING PATTERN
1600       STA (ZPAGE),Y ;BUT MODIFIED FOR MOTOR X
1610       PLA
1620       DEX
1630       BPL EXCEPT
1640 ;++ TABLE IS NOW LOADED ++
1650       LDY #$30
1660 WAIT
1670 ;     DEY             ;FILL IN SOME
1680       BNE WAIT        ;TIME
1690 ;
1700       LDA #$FF        ;ALL PULSES ON
1710       LDY #$00
1720 LOOP
1730       AND (ZPAGE),Y ;BUT MASK OFF WITH EACH
1740       STA PORT        ;TABLE ELEMENT IN TURN
1750       INY
1760       BNE LOOP
1770 ;
1780       LDX #$07
1790       LDA #$FF
1800 CLEAR
1810       LDY ANGLE,X    ;CLEAR ALL EXCEPTIONS
1820       STA (ZPAGE),Y
1830       DEX
1840       BPL CLEAR
1850 ;++ ALL PULSES SHOULD NOW BE FINISHED ++
1860       PLA
1870       TAX
1880       PLA             ;RESTORE REGISTERS
1890       TAY
1900       PLA
1910       PLP
1920       JMP $EA31
```

## BASIC-Ladeprogramm

```
10 REM **** BASIC LOADER FOR ***
*
20 REM **** MULTIPLE SERVOS  ***
*
30 :
40 FOR I=820 TO 922
50 READ A:POKE I,A
60 CC=CC+A
70 NEXT I
80 READ CS:IF CS<>CC THEN PRINT"
CHECKSUM ERROR":STOP
100 DATA120,173,20,3,174,196,3,1
41,196
110 DATA3,142,20,3,173,21,3,174,
197,3
120 DATA141,197,3,142,21,3,169,2
55,160
130 DATA0,145,251,136,208,251,88
,96,8
140 DATA72,152,72,138,72,169,255
,141,1
150 DATA221,162,7,169,255,24,106
,72
160 DATA188,0,48,49,251,145,251,
104
170 DATA202,16,243,160,48,136,20
8,253
180 DATA169,255,160,0,49,251,141
,1,221
190 DATA200,208,248,162,7,169,25
5,188
200 DATA0,48,145,251,202,16,248,
104
210 DATA170,104,168,104,40,76,49
,234
220 DATA13072:REM*CHECKSUM*
```

## BASIC-Aufrufprogramm

```
10 REM **** MULTIPLE SERVO ****
20 :
30 DN=8:REM IF CASSETTE THEN DN=
1
40 IF A=0 THEN A=1:LOAD"MULTISER
V.HEX",DN,1
50 POKE 778,88: POKE 779,3:REM P
OINTER TO EVENT HANDLER
60 POKE 251,0: POKE 252,49:REM S
ET UP ZERO PAGE PTR
70 DDR= 56579:POKE DDR,255:REM A
LL OUTPUT
80 MC=820: SYS MC
90 :
100 GET K$:IF K$="" THEN 100:REM
 AWAIT KEY
110 REM ** ALTER MOTOR POSITION
**
115 AK=ASC(K$)
120 IF AK>48 AND AK<58 THEN POKE
12288+SERVO,VAL(K$)*20
130 IF AK>32 AND AK<40 THEN SERV
O=ASC(K$)-35
140 IF K$<>"E" THEN 100:REM 'E'
TO EXIT
```

## Steuerprogramm für einen Motor

```
 755 REM ***************************
 756 REM  Assemble the machine code
 757 REM ***************************
10000 DEF PROCinitial
10010 DIM space% 200
10020 FOR C=0 TO 2 STEP 2
10030  portb=&FE60
10040  PZ=space%
10050  angle=PZ
10060  PZ=PZ+1
10070  [OPT C
10080  .eventhandler
10090  \save registers first
10100  PHP:PHA:TYA:PHA:TXA:PHA
10110  LDA &04
10120  LDX &timer
10130  LDY &timer
10140  JSR &FFF1 \reset timer
10150  LDA &FF
10160  STA portb
10170  \wait approx. 1msec
10180  LDY &FF
10190  .LOOP DEY
10200       BNE LOOP
10210  \and countout pulse
10220  LDY angle
10230  .LOOP1 DEY
10240       BNE LOOP1
10250  \stop all output pulses
10260  LDA &0
10270  STA portb
10280  PLA:TAY:PLA:TAY:PLA:PLP
10290  RTS
10300  ]
10310  NEXT C
10320 REM point to eventhandler
10330 !&220=eventhandler OR (!&220 AND &FFFF0000)
10340 ENDPROC
```

## Acorn-B-Programm

### Gemeinsame Startroutine

```
10 MODE 0
15 REM ****************************
16 REM     Set up the timer etc
17 REM ****************************
20 osbyte=&FFF4
30 AX=&97 :X%=&2 :Y%=&FF
40 CALL osbyte:REM set up port B for output
50 CLS
60 DIM p%(8)
70 DIM timer% 12 ,read% 12
80 xtimer=timer% MOD 256
90 ytimer=timer% DIV 256
100 xread=read% MOD 256
110 yread=read% DIV 256
120 PROCinitial
130 FOR I%=angle TO angle+8:angle?I%=128:NEXT
140 t=.02  :REM sec between pulses
150 time%=&FFFFFFFF -(t*100) +1
160 timer%?4=&FF    :REM load highest byte
170 !timer%=time%   :REM set up timer, enable events
180 *FX14,5
190 AX=4 :X%=xtimer :Y%=ytimer :CALL &FFF1
195 REM ****************************
196 REM     The BASIC controller
197 REM ****************************
200 CLS
210 PRINT"PRESS  SHIFT + NUMBER to select motor"
220 PRINT"PRESS  NUMBER to select angle"
225 motor=1
230 REPEAT
240   A=GET
250   IF A%>2F AND A%<3A THEN a=(A-&30)*10*(255/90):angle?
      (motor-1)=a:PRINT"AB ";motor;;"motor "motor " angle "a*(90/255)
260   IF A%>20 AND A%<2A THEN motor=A-&20
270   UNTIL 0
280 END
750
```

## Steuerprogramm für mehrere Motoren

```
755 REM ****************************
756 REM Assemble the machine code
757 REM ****************************
760 DEF PROCinitial
770 DIM space% 600
780 FOR C=0 TO 3 STEP 3
790  zeropage=&70   :REM free for users
800  portb=&FE60 :osword=&FFF1
810  PZ=space%
820  angle=PZ :PZ=PZ+8  :REM potentially 8 motors
830  table=PZ :PZ=PZ+256  :REM 256 possible pulse lengths
835  FORI%=table TO table+&100:?I%=&FF:NEXT
840  lowtable%=table MOD 256
850  hightable%=table DIV 256
860  ?zeropage=lowtable% :zeropage?1=hightable%
870  [OPT C
880  .eventhandler
890  PHP:PHA:TYA:PHA:TXA:PHA
900  LDA &04
910  LDX &timer
920  LDY &timer
930  JSR osword
940   \Start pulse, for some motors it may be possible to start
1000  \before filling table and so reduce the wait loop below
1010  LDA &FF :STA portb
1020   \fill table with exceptions
1030  LDX &7 :LDA &FF :CLC \set up bit pattern
1040  .exceptions
1050       ROR A :PHA  \bit pattern
1060       LDY angle,X \get offset corresponding to angle of motor X
1070       AND (zeropage),Y \keep existing bit pattern
1080       STA (zeropage),Y \but modified for motor X
1090       PLA :DEX
1100       BPL exceptions
1110  \table is now loaded, fill in some time
1120  LDY &60
1130  .wait DEY
1140       BNE wait
1150  LDA &FF  \all pulses on
1160  LDY &0
1170  .loop AND (zeropage),Y  \but mask off with each table
1180       STA portb  \element in turn
1190       INY
1200       BNE loop
1201  LDX &7 :LDA &FF
1203  .clear
1205       LDY angle,X  \clear all the exceptions again
1207       STA (zeropage),Y
1208  DEX
1209  BPL clear
1210   \all pulses should now be finished
1220  PLA:TAX:PLA:TAY:PLA:PLP
1230  RTS
1240  ]
1250  NEXT C
1260 !&220=eventhandler OR (!&220 AND &FFFF0000)
1270 ENDPROC
```

# Quelle: Militär

## Hochrealistische Flugsimulationen vom Realicorder

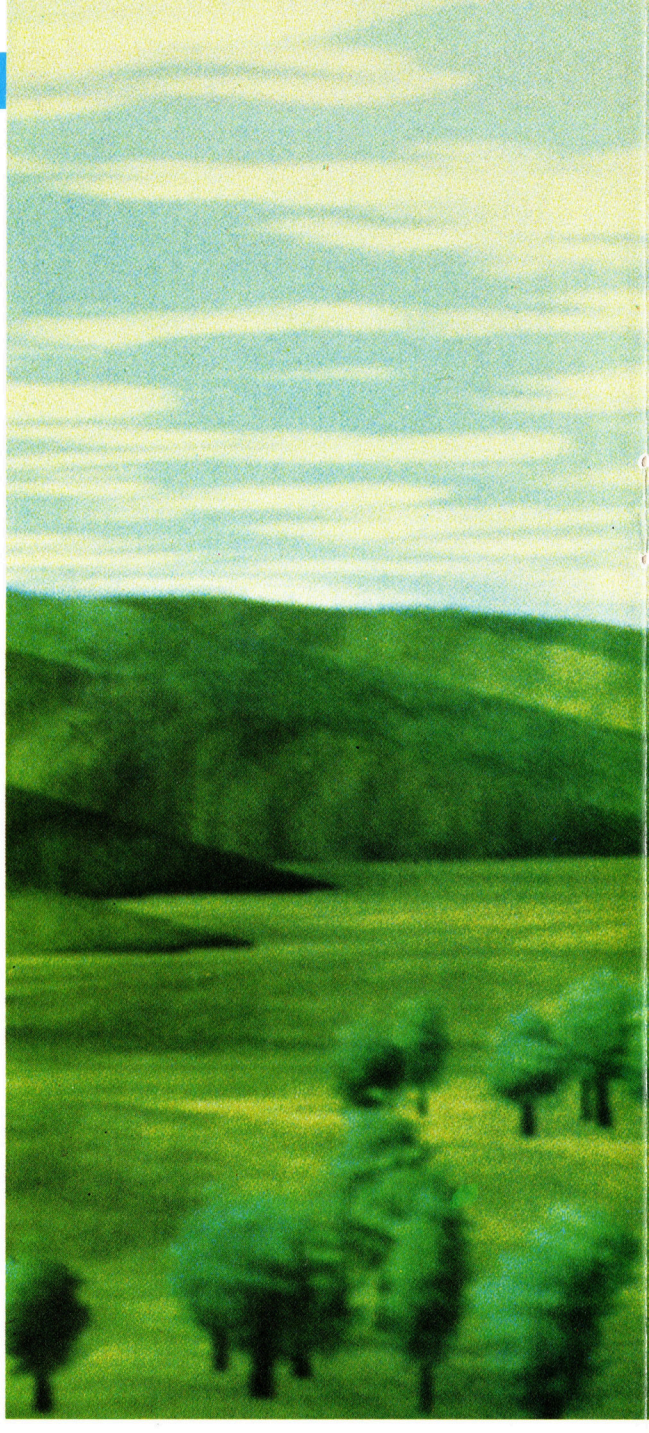
Wichtigste Triebfeder für die Computergrafik im allgemeinen und im besonderen für die Computeranimation waren Bedürfnisse aus dem Bereich der militärischen Forschung und Entwicklung. So wurde schon 1953 am MIT (Massachusetts Institute of Technology) der Computer „Whirlwind" entwickelt, der Flugdaten vorausberechnen und Flugbahnen simulieren konnte.

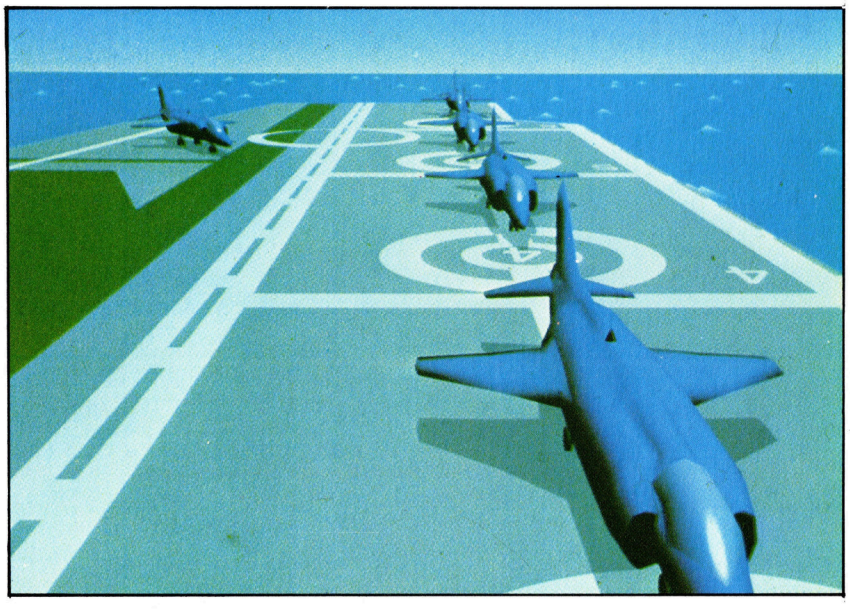
Ein bedeutender Schritt war auch Anfang der sechziger Jahre die Erfindung des Lichtgriffels durch I. Sutherland. Die Darstellung dreidimensionaler Körper wurde möglich, so daß die durch die Materie eigentlich verdeckten Körperlinien unsichtbar blieben, bei „Drehung" des Bildes jedoch sichtbar wurden.

Einige Jahre später trat eine rapide Verkleinerung der Rechner ein, die Generationen der Minis wurden geboren. Um die Kunden von der Leistungsfähigkeit der Geräte zu überzeugen, wurden Computerspiele entworfen. Verglichen mit den heutigen Spielen waren diese simplen Tischtennis-Spiele lächerlich, zu ihrer Zeit jedoch stellten sie eine enorme Leistung dar. Auch in unserer Zeit sind sehr viele dieser Videospiele vereinfachte Variationen von militärischen Simulatoren.

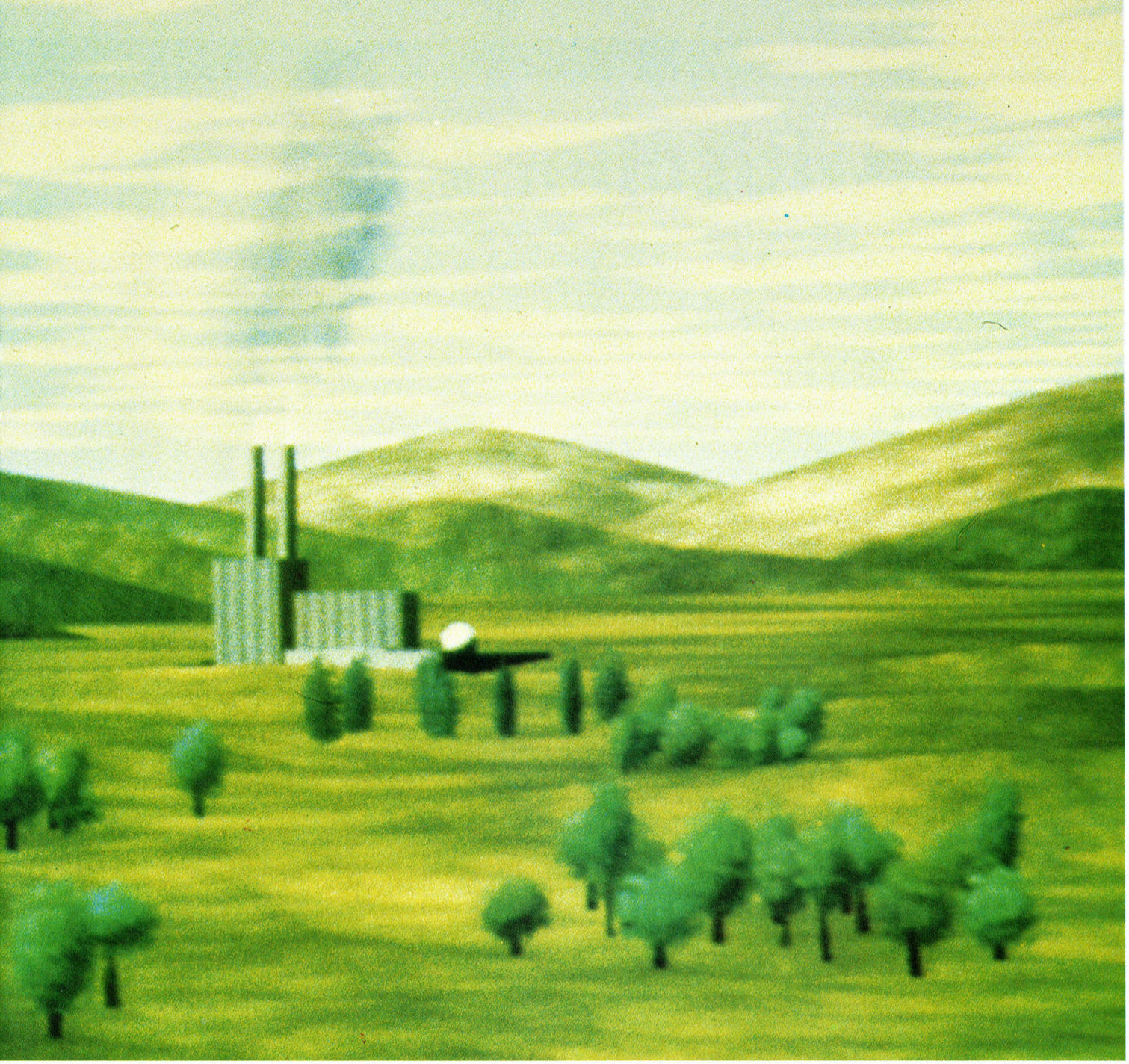
Einige wiederum, wie etwa „Battle Zone" oder „Red Baron" von Atari, erreichten eine derart hohe Qualität, daß sie ohne weiteres zur Gefechtsausbildung benutzt werden konnten. Während früher jedoch noch mit „wire-frames" gearbeitet wurde, geben Spiele der neuesten Generation völlig realistische, durchmodellierte Szenerien wieder. Hier werden wie bei den professionellen Simulatoren Bildplattenspieler eingesetzt, die mit ihrem blitzschnellen Zugriff und einer Speicherkapazität von 54 000 Bildern pro Platte augenblicklich auf eine veränderte Situation reagieren. Im Zusammenhang mit Echtzeit-Simulationen eines angekoppelten Computers spricht man übrigens von sogenannten „Realicordern".

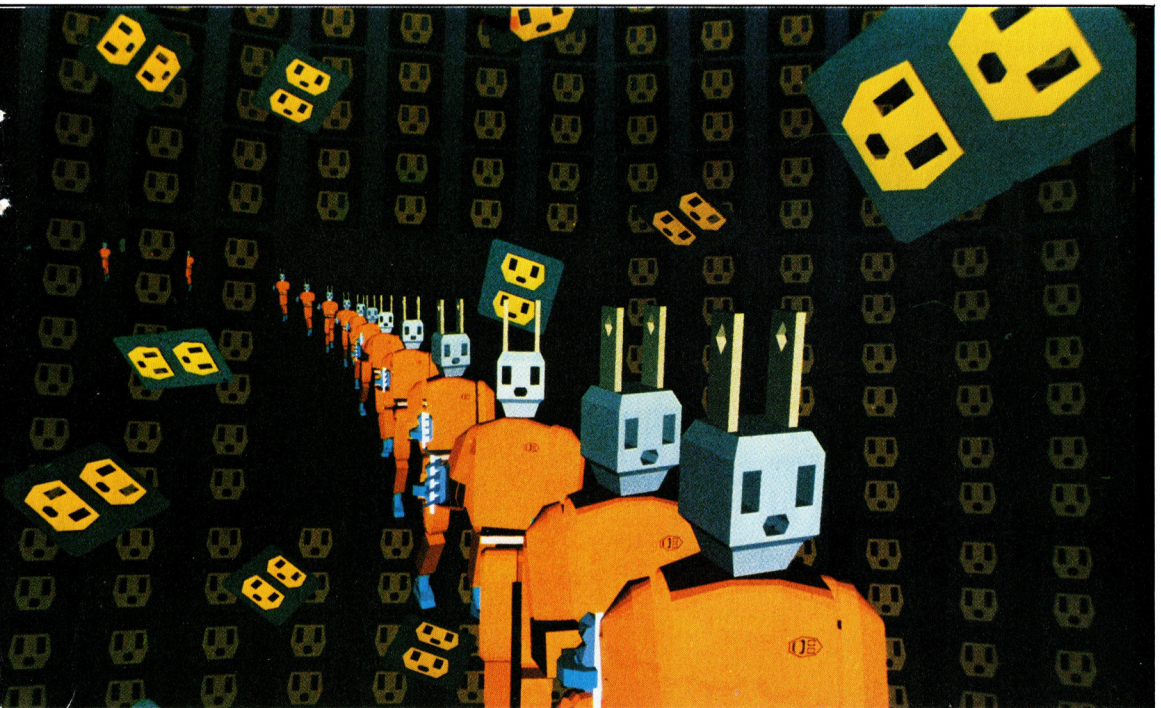
Bevorzugte Aufgaben dieser Simulatoren sind neben allgemeinen flugtechnischen Manövern besonders komplizierte Situationen wie etwa das Starten und Landen auf einem Flugzeugträger oder das Nachtanken in der Luft, wie es bei strategischen Flügen notwendig sein kann.





**Das Starten und Landen auf einem Flugzeugträger muß zunächst bis zur Perfektion im Simulator geübt werden, bevor die Piloten ihre Fähigkeiten im praktischen Einsatz unter Beweis stellen dürfen.**





**Bei dieser Flugsimulation (oben) geht es um das Erkennen bestimmter Details in der überflogenen Landschaft. Zum Beispiel muß ein Atomkraftwerk geortet werden, um es schützen bzw. angreifen zu können. Die schnell vorbeifliegenden Bilder vermitteln einen sehr realistischen Eindruck von dem ganzen Fluggeschehen.**

**Der Marsch der Roboter in eine menschenlose Welt im Bild links hat trotz der möglichen Bedrohung des Betrachters einen gewissen Witz: Die Roboter bestehen aus Stekkern und Steckdosen englischer Elektrogeräte.**

# Kritischer Pfad

**Die Planung eines vielschichtigen Projekts wird weitaus übersichtlicher, wenn die Aufgaben in Diagrammform dargestellt werden. Das Programm MacProject für den Macintosh bietet auch dem Anfänger die Möglichkeit, geordnete und übersichtliche Projektpläne zu entwerfen.**

Die „Analyse des kritischen Pfades" (Critical Path Analysis – CPA) ist ein vielseitiges Managementinstrument, mit dem sich Projekte in einzelne Schritte unterteilen lassen. Jeder Schritt wird durch einen Pfeil dargestellt, der auch den Zeitbedarf anzeigt. So entsteht eine Reihe paralleler Ketten, die an bestimmten Punkten zusammentreffen. Die Kette mit dem größten Zeitbedarf ist der „kritische Pfad" – er bestimmt die Gesamtdauer des Projekts. Aus den Diagrammen läßt sich ablesen, wann und wo welche Materialien, Werkzeuge etc. benötigt werden, wann einzelne Schritte begonnen und beendet werden müssen und wer sie durchführt.

Es wurden schon mehrere Versuche unternommen, CPA-Programme zu erstellen. Da Netzwerkdiagramme jedoch hauptsächlich aus grafischen Elementen bestehen, gab es beim Einfügen von Informationen Probleme mit der Neuberechnung der Grafikpositionen. CPA-Programme waren daher umständlich, unübersichtlich und konnten oftmals nur Text darstellen. Damit wurde jedoch genau das Gegenteil erreicht: Netzwerkdiagramme sollen Abläufe durchsichtiger machen, nicht komplizierter.

Das Grafiksystem des Macintosh eignet sich ideal für diese Aufgabe. Mit MacProject ist der Aufbau eines Netzwerks (im Programm „Project Chart" genannt) so leicht, daß sich die Steuerung eines Projekts erheblich vereinfacht. Die Daten und Zeiten lassen sich beliebig ändern, wobei die jeweiligen Projektzeiten automatisch neu berechnet werden. Neue Aufgaben können eingefügt und nicht mehr aktuelle Abläufe gelöscht werden.

Das knappe, leicht verständliche Handbuch erklärt das am Beispiel einer Grillparty:

SALAT VORBEREITEN
TISCH DECKEN
FEUER ANZÜNDEN
REIS KOCHEN
STEAKS GRILLEN

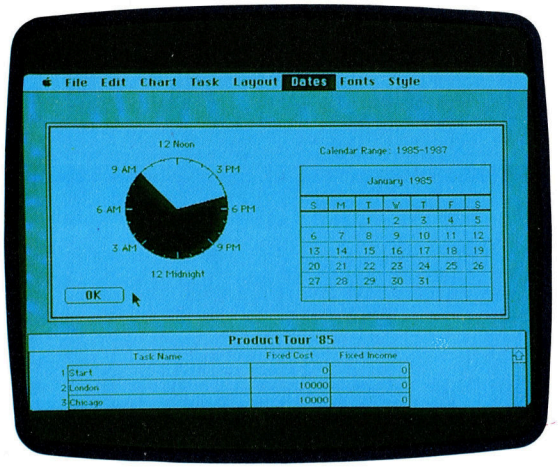
Das Ergebnis des Projekts ist:
MAHLZEIT EINNEHMEN

Die meisten Aufgaben lassen sich gleichzeitig erledigen. Da die Steaks jedoch nicht vor dem Anzünden des Feuers gegrillt werden können, führt der kritische Pfad von „Feuer anzünden" über „Steaks grillen" zu „Mahlzeit einnehmen". Will das Feuer nicht brennen, so verzögert sich das Essen, selbst wenn die anderen Aufgaben erledigt sind. Auch die Salatvorbereitung kann zum kritischen Pfad werden, wenn sich der Helfer verspätet.

## Probleme ähneln einander

Nun braucht man zwar für die Planung einer Grillparty nicht unbedingt ein Programm für circa 500 Mark, aber ähnliche Elemente lassen sich auch für jedes andere Projekt einsetzen: von der Organisation einer Konferenz bis zum Aufbau einer Öl-Raffinerie.

Mit der Maus werden Aufgabenkästen gezeichnet, wobei die Zeitrechnung automatisch am 2. Januar beginnt. Dieses Anfangsdatum läßt sich jedoch ändern oder vom Endpunkt aus rückwärts berechnen. Die Größe des ersten Kastens gilt automatisch für alle weiteren Kästen. Um Verbindungslinien zu zeichnen, setzt man die Maus zuerst in das Innere des Kastens und dann auf die gewünschte nächste Position. In die Kästen lassen sich Aufgabenbezeichnungen eintragen, die automatisch zentriert werden. Die Kästen lassen sich vergrößern, indem die Maus auf die Umrißlinie gesetzt und die Taste gedrückt wird. Dabei zeigt das Programm acht verschiedene Größen, die wiederum mit der Maus angewählt werden können. Wenn der Cursor auf die Umrißlinie gesetzt, aber keine neue Größe angewählt wird, läßt sich der Kasten an jede Stelle des Diagramms stellen.




**Über den Kalender des MacProject – hier gezeigt als oberes Bildschirmfenster – lassen sich die Anzahl der Arbeitsstunden und Tage eintragen, die für die Fertigstellung des Projekts benötigt werden. Dabei werden außerdem Urlaubsperioden, Wochenenden und Festtage berücksichtigt. Am unteren Bildschirmrand ist ein Teil des Kassenbuchs zu sehen, in dem die laufenden Ausgaben des Projekts eingetragen werden.**

Die Wahl der Option INVISIBLE GRID (unsichtbares Raster) – Menüpunkt LAYOUT – richtet alle Kästen exakt aus.

Nachdem ein Kasten mit Namen versehen wurde, kann er mit Informationen (TASK INFO) gefüllt werden. Dabei lassen sich die Zeit (Minuten, Tage, Wochen etc.) und die Hilfsmittel (beispielsweise Materialien) angeben. Das Hauptdiagramm zeigt für jeden Kasten vier Informationen: links oben Hilfsmittel oder frühester Anfangstermin; rechts oben Dauer, Hilfsmittel oder frühester Endtermin; unten links letzter Anfangstermin, feste Kosten oder Hilfsmittel; unten rechts letzter Endtermin, feste Einnahmen oder Hilfsmittel. Der kritische Pfad erscheint dabei mit starker Umrahmung, nicht-kritische Wege dagegen mit dünnen Linien.

Parallel zum Hauptdiagramm entsteht eine Reihe von Analysediagrammen. Die Option RESOURCE TIMELINE zeigt in einem Balkendiagramm die Verteilung der Hilfsmittel. TASK TIMELINE enthält die gleiche Information, jedoch nach Aufgabenbereichen geordnet. „Slack Time" (die Differenz zwischen dem frühesten und dem letztmöglichen Endtermin – also die Zeit, in der die Aufgabe abgeschlossen werden muß, wenn das Projekt nicht verzögert werden soll) wird in Grau angezeigt. Die Projekttabelle stellt eine Liste aller Aufgaben dar (die kritischen Bereiche in Fettdruck). Sie enthält außerdem für jede Aufgabe die Anzahl der zugeteilten Tage, den frühesten und letztmöglichen Anfangs- und Endtermin, feste Kosten, feste Einnahmen und die Kosten und Bezeichnungen der Hilfsmittel. Drei Tabellen geben übersichtlich und auf einen Blick Auskunft über die Kosten: TASK COST ENTRY (Projektkosten), RESOURCE COST ENTRY (Hilfsmittelkosten) und CASH FLOW TABLE (Kassenbuch).

Über die Menüoption LAYOUT läßt sich bestimmen, ob ein schematisches Diagramm des gesamten Netzwerks (ohne Erläuterungen) dargestellt oder die Größe der Darstellung verändert werden soll. Mit dieser Option läßt sich die Netzwerkdarstellung komprimieren, wenn das Diagramm nach dem Löschen einzelner Kästen noch leere Plätze anzeigt.

In MacProject ist die Analyse des kritischen Pfades sehr gut grafisch umgesetzt worden, so daß dies auch der Anfänger leicht verstehen kann. Die Grafikfähigkeiten des Macintosh wurden dabei so umfassend eingesetzt, daß ein ähnliches Programm auf anderen Geräten kaum vorstellbar ist. MacProject wird ebenfalls in einer deutschen Version angeboten.

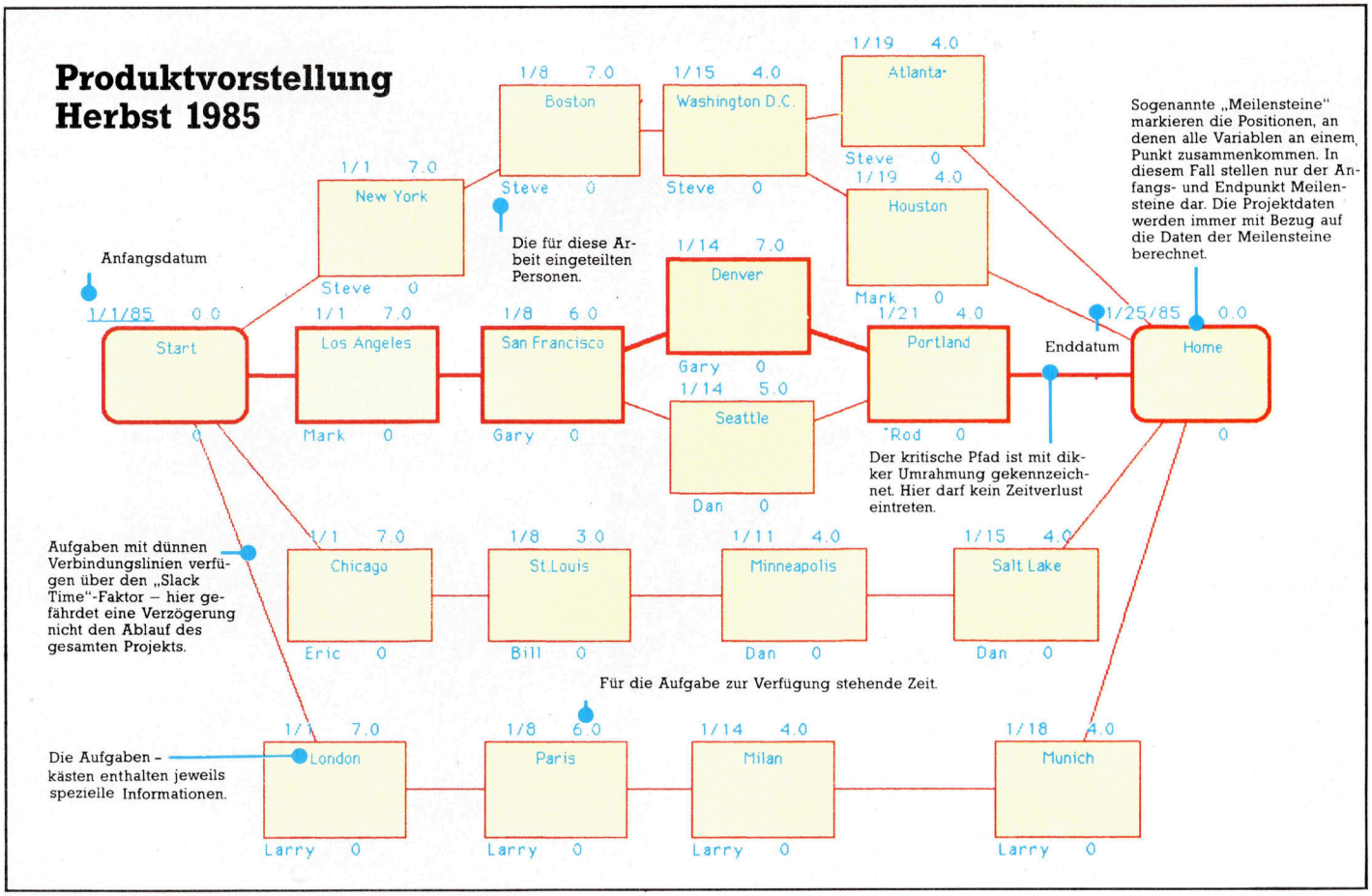
**MacProject:** Für den Apple Macintosh

**Kontaktadresse:** Apple Computer GmbH, Ingolstädter Straße 20, 8000 München 45

**Autoren:** Debra Willet und Stephen D. Young

**Format:** Diskette

**Dieses Diagramm von MacProject stellt eine Tour durch mehrere Städte dar, auf der eine Firma ein neues Produkt vorstellt. Durch die visuelle Darstellung gestaltet sich die Zeitplanung und Aufteilung der Fachkräfte außerordentlich einfach.**



# Gestochen scharf


**Typenraddrucker bieten eine wesentlich bessere Schriftqualität als die Matrixgeräte. Angesichts des Preises und der niedrigen Druckgeschwindigkeit sollte man sich die Anschaffung allerdings gut überlegen – Schönheit kostet Zeit und Geld.**

Auf den ersten Blick scheint der Kauf eines Typenraddruckers unsinnig. Für die Programmprotokollierung ist dieses System zu langsam und zudem wesentlich teurer als ein vergleichbarer Nadeldrucker. Typenradgeräte sind jedoch sinnvoll, wenn ein sauberes Schriftbild erzeugt werden soll, zum Beispiel für Geschäftsbriefe. Bei den Matrixdruckern wird jedes Zeichen als Punktmuster aufgebaut, und die Punkte bleiben einzeln erkennbar, ganz egal mit wie vielen Nadeln der Druckkopf auch arbeiten mag.

Typenradsyteme erzeugen das Druckbild dagegen durch Aufschlag von Blocktypen auf Farbband und Papier, wie bei der althergebrachten Schreibmaschine, allerdings nicht mit Hilfe von Typenhebeln. Die einzelnen Typen sind vielmehr auf den elastischen Speichen eines Typenrads angebracht. Der Umriß eines Typenrades hat entfernte Ähnlichkeit mit einem Gänseblümchen. Daher stammt übrigens die englische Bezeichnung „Daisy Wheel". Mit Hilfe des Typenrades wird ein Schriftbild erzielt, das besser lesbar ist und „professioneller" wirkt.

Allerdings muß man dafür eine geringere Druckgeschwindigkeit in Kauf nehmen. Typenradmodelle sind viel langsamer als gleichteure Matrixdrucker. Das Typenrad wird vor jedem Anschlag erst gedreht, bis das richtige Zeichen oben steht und von einem kleinen Hammer auf das Papier gedrückt werden kann. Anschließend bewegt sich der Typenradwagen auf die nächste Druckposition.

Anders beim Matrixdrucker: Dort erfolgt der Nadelanschlag kontinuierlich, während sich der Kopfschlitten über die Zeile bewegt. Das erklärt den Unterschied in der Druckgeschwindigkeit. Der Matrixdrucker FX-80 von Epson beispielsweise leistet 160 Zeichen/sec. Ein Typenradmodell für den gleichen Preis schafft nur rund 20 Zeichen/sec.

Zur Erhöhung der Druckgeschwindigkeit werden Matrix- wie Typenradmodelle häufig mit „bidirektionalem" Druck und Druckwegoptimierung angeboten. Bidirektional bedeutet, daß die Zeilen abwechselnd von links nach rechts und umgekehrt gedruckt werden. Die Druckwegoptimierung bewirkt, daß Zwischenräume in der Zeile quasi übersprungen werden. Einfachere Geräte brauchen für das „Drucken" eines Leerraums ebensoviel Zeit wie für jedes andere Zeichen.

Bei Matrixdruckern befinden sich die Punktmuster im ROM, bei Typenraddruckern dagegen ist das Schriftbild hardwaremäßig durch das Typenrad vorgegeben. Beides hat seine Vorteile: Beim Nadeldrucker kann der Zeichensatz softwaremäßig vom Rechner aus über Steuerzeichen neu definiert werden, bei einigen Typenradsystemen läßt sich das Schreibrad gegen ein anderes austauschen.

Die Typenräder gibt es in einer Vielzahl von Schriftarten. Gängig sind Schriften wie „Courier", „Prestige", „Gothic" und „Italic" (kursiv). Typenräder aus Kunststoff bekommt man schon für zwanzig Mark, Metall-Räder (mit wesentlich längerer Lebenserwartung) etwa ab achtzig Mark.

Bei den meisten Schriftarten ist jedoch die Ziffer „0" nicht vom Großbuchstaben „O" zu unterscheiden, und dasselbe gilt für die Ziffer „1" und den Kleinbuchstaben „l". Während die teureren Modelle Typenräder mit 127 Zeichen haben, verfügt die Mehrzahl nur über 92 oder 96 Typen.

Die Suche nach Programmfehlern wird nicht gerade leichter, wenn Sie im Protokoll die Zahl „1" und das kleine „l" nicht auseinanderhalten können. Und dazu kommt noch, daß einige Sonderzeichen mit dem Typenraddrucker nicht dargestellt werden können. Für Listingausdrucke empfiehlt sich daher eindeutig der Kauf eines Matrixdruckers.

Wie bei Nadeldruckern sind auch bei Typenradgeräten spezielle Effekte vorgesehen, allerdings nur in begrenztem Umfang. Die Programmierung erfolgt auch hier über Steuerzeichen (Escape-Codes) vom Rechner. Bei einem Diablo-Typenraddrucker bewirkt zum Beispiel das Zeichen ESC-E ein automatisches Unterstreichen, das mit ESC-R wieder abgestellt werden kann. Beim Microsoft-BASIC müßten Sie LPRINT CHR$(27);"E"; bzw. LPRINT CHR$(27);"R"; eingeben, um diese Befehle an den Drucker zu senden. Mit ESC-1 wird ein Tabulatorstop und mit ESC-9 der linke Randsteller gesetzt, mit ESC-U können Sie eine Halbzeilenschaltung zum Einfügen von Indizes auslösen. Auch Fettschrift ist wie beim Matrixdrukker möglich. Dazu wird der gleiche Buchstabe viermal angeschlagen, so daß er sich vom übrigen Schriftbild abhebt. Das Diablo-Steuerzeichen für dieses „Emboldening" (Verstärken) ist ESC-0.

Bei einigen Typenradsystemen läßt sich der Zeichen- und der Zeilenabstand so stark reduzieren, daß das Gerät ähnlich wie ein Nadeldrucker auch für einfache Grafikdarstellungen eingesetzt werden kann.

Beim Diablo-Drucker veranlaßt ESC= das Zentrieren des Textes. Zum Komfort gehört auch der Dezimaltabulator: Nach ESC-H werden alle Zahlen mit der Kommaposition untereinander gerückt.

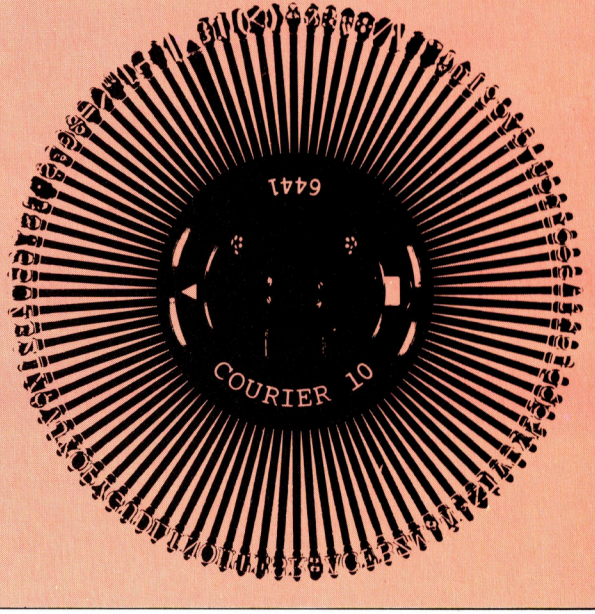
## Typenrad mit Talenten

Die teureren Typenraddrucker sind auch für „Proportionalschrift" geeignet. Dabei ist der Zeichenabstand nicht konstant, etwa $^1/_{10}$ oder $^1/_{12}$ Zoll, sondern er wird der jeweiligen Zeichenbreite angepaßt. Der Buchstabe „w" braucht beispielsweise viel mehr Platz als das „i". Bei der Proportionalschrift wird für das „i" eine kleinere Wagenbewegung ausgeführt als beim „w". Die Buchstaben stehen dann in aufeinanderfolgenden Zeilen nicht mehr alle genau untereinander, und das Schriftbild ist insgesamt besser lesbar. Das Ein- und Ausschalten der Proportionalschrift geschieht wiederum durch Steuerzeichen, beim Silver-Reed-Drucker EXP 770 durch ESC-P.

Für den Heimcomputerbesitzer gibt es eine sinnvolle Alternative zum Typenraddrucker, nämlich die elektronische Typenradschreibmaschine mit Interface. Bis vor kurzem war bei den Schreibmaschinen ein Rechneranschluß praktisch nicht möglich, da diese nicht mit entsprechenden Schnittstellen ausgestattet waren. Bei den neueren Schreibmaschinen ist das Interface nun aber entweder schon eingebaut oder für drei- bis vierhundert Mark nachrüstbar. Sie verfügen somit nicht nur über ein Schönschriftsystem für die Textverarbeitung, sondern können zwischendurch kurze Briefe und andere kleine Schreibarbeiten auch ohne den Rechner erledigen.

## Qualitätsangebot

```
Schriftprobe mit 10 Zeichen pro Zoll
Schriftprobe mit 12 Zeichen pro Zoll
Schriftprobe mit 15 Zeichen pro Zoll

Die Halbzeilenschaltung erlaubt das Einfügen eines
Index:
                          H2O

ESC-E veranlaßt die automatische Unterstreichung,
ESC-R schaltet sie wieder aus.

ESC= rückt den Titel in die Zeilenmitte:

                Eine zentrierte Überschrift

Schriftprobe mit fester Teilung:
Hier ist der Zeichenabstand konstant. Achten Sie
besonders auf die Zwischenräume bei den Ziffern
0123456789. Die Zeichenzwischenräume innerhalb der
Zeile bleiben gleich:

WWWWWWWWWWWW
iiiiiiiiiiii

Schriftprobe in Proportionalschrift:
Hier ist der Zeichenabstand variabel. Achten Sie
besonders auf die Zwischenräume bei den Ziffern
0123456789. Die Zeichenzwischenräume innerhalb der
Zeile sind unterschiedlich:

WWWWWWWWWWWW
iiiiiiiiiii

Insgesamt wirkt das Druckbild bei Proportional-
schrift wesentlich gefälliger.
```




**Mit dem Typenradsystem wird die Schriftqualität einer guten Büroschreibmaschine erzielt, auch Proportionalschrift bereitet häufig keine Schwierigkeiten. Dafür müssen Sie gegenüber einem Matrixdrukker einiges an Flexibilität und Geschwindigkeit opfern und auch tiefer in die Tasche greifen. Das Typenrad läßt sich bei einigen Geräten austauschen.**

# ALU und Joystick

**Im letzten Abschnitt unseres Abenteuerspiel-Projekts haben wir uns mit dem Entwurf zweier Grafiken für den Acorn B befaßt. Sie stellen zwei wichtige Orte in Digitaya dar. Jetzt wollen wir dieselben Grafiken auf dem Spectrum entwerfen.**

Bei der ALU-Grafik sollen die Buchstaben A, L und U über die Mitte des Bildschirms „gescrollt" werden. Beim Acorn B wurde hierzu ein Startpunkt spezifiziert, dann der Buchstabe mit relativen DRAW-Befehlen gezeichnet, wieder gelöscht, der Startpunkt verschoben und dann der ganze Vorgang wiederholt. Dies ist auch bei der Spectrum-Version möglich.

Der DRAW-Befehl des Spectrum erlaubt nur relatives Zeichnen – das heißt relativ zum zuletzt gezeichneten Punkt. Durch PLOTten eines Startpunkts und Ausführung einiger DRAW-Befehle kann der Buchstabe gezeichnet und anschließend beliebig über den Bildschirm verschoben werden. Dazu ist lediglich der Startpunkt an eine andere Position zu setzen. Das Löschen ist möglich, indem dasselbe Objekt an derselben Position noch einmal gezeichnet wird, jedoch mit invertierten Farben. Dies wird mit dem Befehl INVERSE 1 erreicht, der mit INVERSE 0 wieder abgeschaltet wird.

Wenn wir als Beispiel den Buchstaben A verwenden, der von links aus verschoben wird, können alle eben genannten Anweisungen mit Hilfe einer FOR. . .NEXT-Schleife ausgeführt werden. Diese Schleife erhöht den Wert der X-Koordinate des Startpunkts und enthält zudem eine zweite FOR. . .NEXT-Struktur, die lediglich die DRAW-Befehle zweimal ausführt. Der letzte Wert für X ist 55. Er repräsentiert die Endposition des Buchstabens auf dem Bildschirm. Da der Buchstabe an seiner Endposition natürlich nicht mehr gelöscht werden soll, wurde ein Vergleich eingebaut, der gewährleistet, daß der Buchstabe nur dann gelöscht wird, wenn der Wert der X-Koordinate kleiner als 55 ist. Nach derselben Methode entstehen auch die Buchstaben L und U.

Vor dem Erstellen einer Grafik sollte man einen Rohentwurf auf Papier anfertigen und die ungefähren Koordinaten abschätzen. Außerdem sollten die Positionen aller auf dem Bildschirm darzustellenden Buchstaben durch Angabe der Reihe und Spalte spezifiziert werden. Unser Bild rechts zeigt, wie der komplette Bildschirm aussehen soll. Wenn man seinen Bildschirmaufbau gründlich plant, so ist Übersichtlichkeit und logische Gestaltung der Lohn der Mühe, die der Entwurf kostet. Die Wörter AND, OR und NOT werden auf dem Bildschirm mit Hilfe des Befehls PRINT AT r,c positioniert. r repräsentiert dabei die Anzahl der Reihen vom oberen und c die Anzahl der Spalten vom linken Rand aus. Die Punkte werden mit Hilfe von CIRCLE x,y,r gezeichnet, wobei die Koordinaten des Zentrums und die Länge des Radius angegeben werden.

## Gefahr am Port

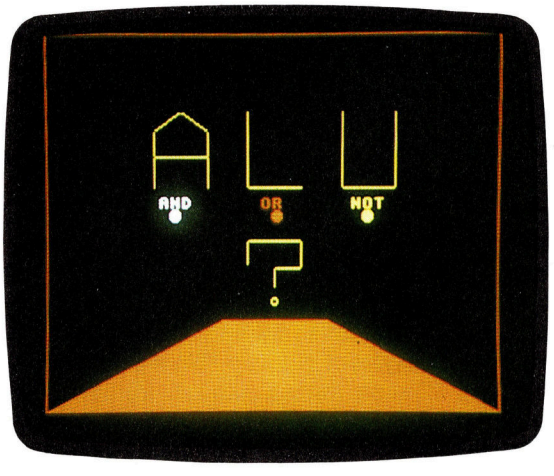
Nach Ausführung der Zeichenroutinen wartet das Programm auf einen Tastendruck, bevor INK und PAPER auf die Originalfarben zurückgesetzt werden und der Bildschirm gelöscht wird. Danach erfolgt ein RETURN zur ALU-Hauptroutine. Das Drücken einer Taste wird mit INKEY$ überprüft. Wurde keine Taste gedrückt, wird der Test einfach wiederholt.

Um diese Unterroutine aufzurufen, muß folgende Zeile in Digitaya eingefügt werden:

4565 GOSUB 7000: REM ALU PICTURE S/R

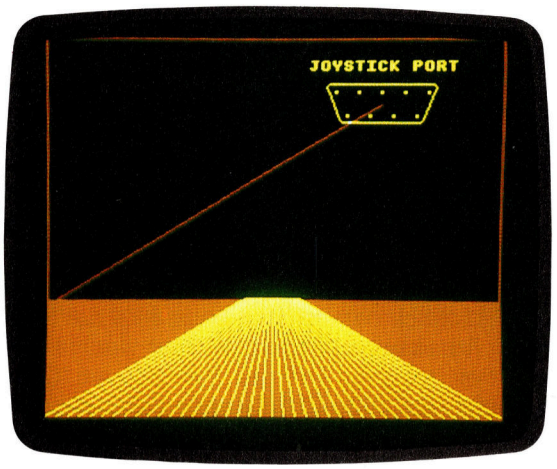
Die Grafik der Anschlußbuchse wurde so entworfen, daß aus ihrem Zentrum Laserstrahlen herausschießen. Die Pins werden durch Punkt-Zeichen und der Rahmen in hochauflösender Grafik gezeichnet. Durch die Darstellung der zur Mitte zulaufenden Linien entsteht ein räumlicher Tiefeneffekt. Der Startpunkt jeder Linie liegt auf der Horizontlinie und wird durch einen PLOT-Befehl gesetzt. Der jeweilige Endpunkt befindet sich am unteren Rand des Bildschirms.

Da der DRAW-Befehl des Spectrum relativ arbeitet, wird die Routine etwas komplizierter, weil kein absoluter Endpunkt gesetzt werden kann. Wenn die X-Koordinate des Startpunkts der ganz links befindlichen Linie 111 ist, muß daraus der relative Wert für den Endpunkt er-



rechnet werden. In der FOR. . .NEXT-Schleife in den Zeilen 8030–8060 sehen Sie diese Berechnung, die auf der Schrittweite am Start- und Endpunkt basiert.

Auch hierbei ist es ratsam, die Größenverhältnisse und Koordinaten auf Papier zu entwerfen, bevor man programmiert. Auf dem Bildschirm sollte nun das folgende Bild erscheinen:




Die Laserstrahlen werden vom Zentrum der Buchse aus mit einer zufälligen INK-Farbe zu einem zufälligen Punkt auf der Horizontlinie gezeichnet. Durch erneutes Ausführen der Prozedur mit INVERSE 1 wird die Linie wieder gelöscht, so daß der Strahl nur für einen kurzen Augenblick erscheint. Beim Löschen der Linie entsteht jedoch ein Problem: Da der Strahl vom Zentrum der Buchse aus gezeichnet wird, überschneidet er deren Grafik. Durch den anschließenden Löschvorgang wird somit auch ein Teil der Buchsendarstellung überschrieben. Deshalb muß nach dem Löschen des Strahles die Buchsen-Grafik neu gezeichnet werden.

Obwohl sich das Ende jedes Laserstrahls direkt vor der Horizontlinie befindet, wird sie dennoch vom Löschen eines Strahles beeinträchtigt. Dies liegt an der Art, wie der Spectrum Farben kontrolliert. Der Teil des Horizonts, der dem Endpunkt des Strahles am nächsten liegt, nimmt dessen Farbe an. Grund hierfür ist, daß der Spectrum nur eine INK- und PAPER-Farbe innerhalb des Zeichenbereichs ermöglicht. Jegliche in diesem Bereich befindliche Grafik nimmt die Vordergrundfarbe der verwendeten neuen INK-Farbe an. Daher muß also auch die Horizontlinie nach Löschen eines Strahles neu gezeichnet werden. Die Routine setzt die Darstellung der Laserstrahlen so lange fort, bis eine Taste gedrückt wird.

Zum Aufruf dieser Unterroutine muß die folgende Zeile in das Hauptprogramm integriert werden:

```
3845 GOSUB 8000: REM JOYSTICK PORT
     PICTURE
```

Im nächsten Artikel werden wir uns mit der Implementation dieser zwei Grafiken auf dem Commodore 64 befassen.

## ALU-Programm

```
7000 REM **** alu picture s/r ****
7010 INK 6: PAPER 0: CLS
7015:
7017 REM **** letter A ****
7020 FOR x=0 TO 55 STEP 5
7030 INVERSE 0
7040 FOR i=1 TO 2
7050 PLOT x,100
7060 DRAW 0,30
7070 DRAW 15,20
7080 DRAW 15,-20
7090 DRAW 0,-30
7095 DRAW 0,20
7096 DRAW -30,0
7110 IF x<55 THEN  INVERSE 1
7115 NEXT i
7120 NEXT x
7130:
7140 REM **** letter L ****
7150 FOR y=100 TO 150 STEP 5
7152 INVERSE 0
7155 FOR i=1 TO 2
7160 PLOT 113,y
7170 DRAW 0,-50
7180 DRAW 30,0
7190 IF y<150 THEN  INVERSE 1
7200 NEXT i
7210 NEXT y
7220:
7230 REM **** letter U ****
7240 FOR x=225 TO 170 STEP -5
7250 INVERSE 0
7260 FOR i=1 TO 2
7270 PLOT x,150
7280 DRAW 0,-50
7290 DRAW 30,0
7300 DRAW 0,50
7310 IF x>170 THEN  INVERSE 1
7320 NEXT i
7330 NEXT x
7340:
7350 REM **** buttons ****
7360 PRINT AT 10,7;"AND"
7370 PRINT AT 10,15;"OR"
7380 PRINT AT 10,22;"NOT"
7390 INK 3: CIRCLE 70,80,5
7400 INK 4: CIRCLE 128,80,5
7410 INK 5: CIRCLE 185,80,5
7420:
7430 REM **** q mark ****
7435 INK 6
7440 PLOT 113,45
7450 DRAW 0,15
7460 DRAW 30,0
7470 DRAW 0,-20
7480 DRAW -15,0
7490 DRAW 0,-7
7500 FOR r=6 TO 0 STEP -2
7510 CIRCLE 128,23,r
7520 NEXT r
7530:
7540 IF INKEY$="" THEN  GO TO 7540
7550 INK 0: PAPER 7: CLS
7560 RETURN
```

## Buchsen-Darstellung

```
8000 REM **** jstick port pic s/r ****
8010 INK 6: PAPER 0: CLS
8020 REM **** foreground ****
8030 FOR n=1 TO 31
8040 PLOT 112+n,50
8050 DRAW 7*n-112,-50
8060 NEXT n
8070:
8080 REM **** horizon ****
8085 INK 6: INVERSE 0
8090 PLOT 0,50
8100 DRAW 255,0
8110:
8120 REM **** port ****
8130 PRINT AT 1,18;"JOYSTICK PORT"
8140 PRINT AT 3,20;". . . . ."
8150 PRINT AT 5,21;". . . ."
8160 PLOT 158,152
8170 DRAW 75,0
8180 DRAW 1,-1
8190 DRAW 1,-1
8200 DRAW 0,-1
8210 DRAW -1,-1
8220 DRAW -10,-25
8230 DRAW -2,-2
8240 DRAW -52,0
8250 DRAW -2,2
8260 DRAW -10,25
8270 DRAW -1,1
8280 DRAW -1,1
8290 DRAW 0,1
8300 DRAW 1,1
8310:
8320 REM **** shoot ****
8340 INK RND*7
8350 LET x=RND*255-194
8360 LET y=-86
8365 INVERSE 0
8367 FOR i=1 TO 2
8370 PLOT 194,136
8380 DRAW x,y
8385 INVERSE 1
8387 NEXT i
8390:
8400 REM **** test for key ****
8410 IF INKEY$="" THEN  GO TO 8080
8415 INVERSE 0
8420 INK 0: PAPER 7: CLS
8430 RETURN
```

# Große Sprünge

**Nachdem in der letzten Folge untersucht wurde, wie die indizierte Adressierung des 6809-Prozessors funktioniert, sehen wir uns heute die indirekte Adreßmethode an und entwickeln ein Programm, das Zeichen auf dem Bildschirm darstellt.**

Die indirekte Adressierung läßt sich mit fast allen anderen Adreßarten einsetzen. Dabei wird eine Adresse berechnet und der Inhalt dieser Adresse sowie die darauffolgende Speicherstelle als aktuelle Adresse angesehen. Aus dieser endgültigen Adresse werden dann die Daten geladen.

Bei den folgenden Werten:

| Adresse | Inhalt |
|---|---|
| 3000 | 40 |
| 3001 | 0A |
| 400A | F2 |

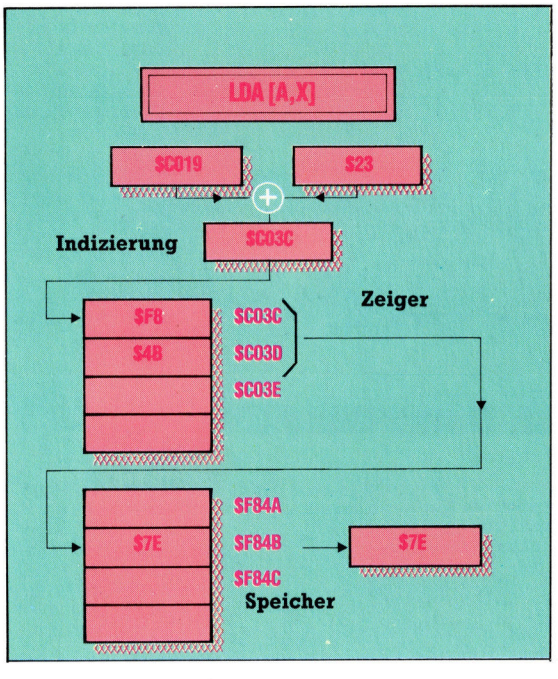
lädt der direkte Ladebefehl LDA $3000 den Wert $40 in den Akkumulator A – die aktuelle Adresse ist dabei $3000. Bei der indirekten Adressierung steht der Operand in eckigen Klammern. LDA [$3000] lädt daher den Wert $F2 in A, wobei die aktuelle Adresse von dem Wert dargestellt wird, der in $3000 und $3001 gespeichert ist – in diesem Fall $400A. Der Inhalt von $3000 und $3001 bildet einen Zeiger (oder Vektor) auf die aktuelle Adresse $400A. Beachten Sie, daß die Adressen des 6809 das Format „höherwertiges Byte vor niederwertigem Byte" (auch hi-lo genannt) haben: $40 wird daher in $3000 gespeichert und $0A in $3001. Der Z80 von Zilog und der 6502 von MOS Tech verwenden das umgekehrte Format – $0A (das niederwertige Adreßbyte) wird in $3000 gespeichert, $40 (das höherwertige Byte) in $3001.

Die indirekte Adressierung läßt sich gut mit der indizierten Adressierung kombinieren. Der Befehl LDA [A,X] (indirekt indiziert) berechnet eine Adresse, indem er den Inhalt von A und X addiert und dann den 16-Bit-Wert dieser und der darauffolgenden Speicherstelle als aktuelle Adresse einsetzt, dessen Inhalt in A geladen wird.

Tatsächlich benutzt man beim 6809 die indirekte Adressierung jedoch weitaus weniger als bei den meisten anderen Prozessoren (Programme des Z80 und 6502 verwenden sie oft), da er über viele Möglichkeiten der indizierten Adressierung verfügt. Es gibt jedoch Aufgaben, die durch die indirekte Adressierung sehr vereinfacht werden, unter anderem der Einsatz von Peripheriegeräten (auf den wir in einer der nächsten Folgen ausführlich eingehen werden). Die Motorola-Prozessoren setzen im Gegensatz zu den 8080- und 8060-Chip-Familien von Intel für die Ein- und Ausgabe eine Memory-Map ein. Die Kommunikationsregister werden dabei in der Memory-Map des Systems abgelegt und lassen sich wie normale Speicherstellen ansprechen. Ein Steuermodul – beispielsweise eine Druckroutine – braucht die Adresse des entsprechenden Schnittstellenregisters. Wenn dieses Register nun an eine andere Stelle der Memory-Map verlegt wurde, oder nicht nur ein Gerät dieser Art existiert, ist es einfacher, die Speicherstelle mit der Adresse des Kommunikationsregisters (den Zeiger) zu ändern als jede einzelne Registeradresse. Die Routine bezieht sich daher über den Zeiger indirekt auf das Peripheriegerät.

Wenn sich die Bezugsadressen eines Programms verändern können, sind Adreßzeiger sehr praktisch, da bei einer Veränderung der aktuellen Adresse lediglich der Inhalt des Zeigers geändert werden muß.

Bei dieser Technik wird viel mit einer Struktur gearbeitet, die „Sprung-Tabelle" heißt und einfach eine Liste von Zeigern umfaßt. Jedes Betriebssystem enthält eine Reihe von Routinen, die grundlegende Funktionen ausführen – beispielsweise Zeichen von der Tastatur lesen oder auf dem Bildschirm anzeigen. Viele Maschinencodeprogramme rufen diese Routinen über die Sprung-Tabelle auf. Dabei können die Routinen, auf die die Werte der Sprung-Tabelle zeigen, geändert oder verlegt werden.




**Das Argument [A,X] des LDA-Befehls ist in eckige Klammern eingeschlossen. Das bedeutet, daß der Inhalt von X (hier $C019) mit dem Inhalt von A ($23) addiert wird und so die 16-Bit-Adresse ($C03C) ergibt. Dieses und das nächste Byte ($C03C) sind nun der Zeiger auf die zu ladende Adresse ($F84B), deren Inhalt schließlich in A übertragen wird. Da X vor dem indirekten Zugriff zu A addiert wird, nennt man diese Adreßart „indirekt vorindiziert". Bei dem Gegenstück – der „Nachindizierung" – wird zunächst die indirekte Adresse berechnet und danach die Indizierung durchgeführt.**


Viele Betriebssysteme besitzen nur eine Einsprungadresse für Unterroutinenaufrufe. Dabei wird eins der CPU-Register mit dem Funktionscode geladen, der das aufzurufende Modul angibt. Dieser Code ist der Index (oder Offset) für den entsprechenden Vektor der Sprung-Tabelle, dessen Adresse die gewünschte Routine anspricht.

Nehmen wir als Beispiel ein ROM mit vier KByte, die bei $F000 liegen. Die ersten 256 Byte ($F000 bis $F0FF) enthalten eine Tabelle mit bis zu 128 Adressen für Subroutinen, die irgendwo im ROM untergebracht sind. Die Einsprungroutine liegt bei $F100 und erwartet im Akkumulator B einen Funktionscode im Bereich von 0 bis 127. Mit diesem Code übergibt die Einsprungroutine die Steuerung an die entsprechende Unterroutine und nach deren Ausführung wieder zurück an das aufrufende Programm. Der Aufruf von Funktion 1 sieht folgendermaßen aus:

```
LDB #1        den Funktionscode in B laden
JSR $F100     Einsprungroutine aufrufen
```

Die Einsprungroutine selbst lautet:

```
LDX $F000     Anfangsadresse der Sprung-Tabelle
LSLB          B um eine Stelle nach links verschieben
              (entspricht einer Multiplikation von B mit
              zwei), da jeder Tabelleneintrag zwei
              Byte lang ist. Der Zeiger für den Funk-
              tionscode 1 befindet sich in $F002 und
              $F003, während der Zeiger für Code 2
              auf $F004 und $F005 liegt etc.
BRA [B,X]     Steuerung an die Adresse übergeben,
              die an der von B angegebenen Tabellen-
              position gespeichert ist.
```

Beachten Sie, daß die Übergabe an die Routine mit BRA (oder JMP) und nicht mit BSR (oder JSR) geschieht. Damit gibt das RTS am Ende der Betriebssystemroutine die Steuerung direkt an das aufrufende Programm zurück, ohne erst die Einsprungroutine ansprechen zu müssen.

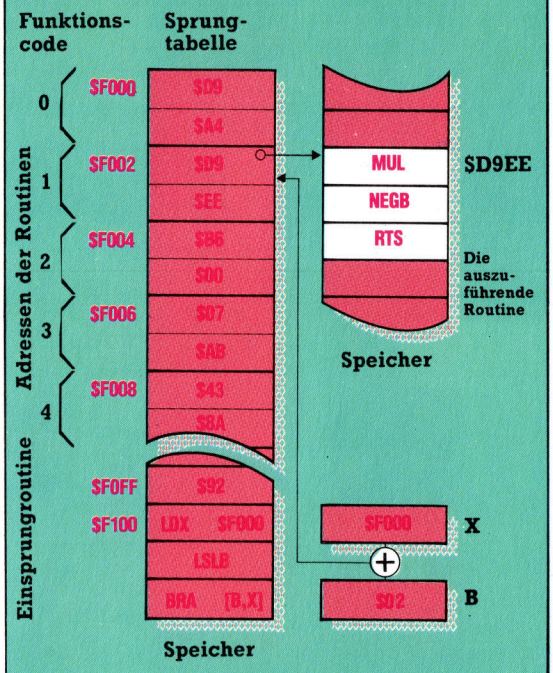
Unser nächstes Beispiel zeigt die indirekte Adressierung einer Bildschirmanzeige, die mit einer Memory-Map arbeitet. Auf vielen Micros belegt der Bildschirmspeicher einen Teil des Arbeitsspeichers, der für höhere Anzeigegeschwindigkeiten auch direkt adressiert werden kann. Nehmen wir der Einfachheit halber an, daß der Bildschirm einen Speicherblock von $E000 bis $E3FF belegt und 16 Zeilen mit je 64 Zeichen enthält. Die Position des Cursors (ein 16-Bit-Wert dieses Adreßbereichs) befindet sich in $E400. Die erste Subroutine löscht die Bildschirmpositionen zunächst durch Einsetzen von Leerzeichen (ASCII-Code 32). Die zweite Subroutine stellt nun das in A gespeicherte Zeichen auf der aktuellen Cursorposition dar, wenn kein Return (ASCII-Code 13) eingegeben wurde. Beim Drücken der Return-Taste wird der Rest der Zeile gelöscht und der Cursor auf den Anfang der nächsten Zeile gesetzt. In diesem Beispiel stellt die Unterstreichung ("_") den Cursor dar.

```
SPACE     EQU 32        ASCII-Code für Leerzeichen
CR        EQU 13        ASCII-Code für Return
HOME      EQU $E000     Anfang des Bildschirmspeichers
LENGTH    EQU 1024      Größe des Bildschirmspeichers (16
                        Zeilen × 64 Zeichen = 1024)
```

**Obwohl die indirekte Adressierung für Computerabläufe eine große Bedeutung hat, gibt es für diese Technik im täglichen Leben nur wenige Beispiele. Eine brauchbare Analogie ist das Eurosignal, bei dem eine Person über ein kleines tragbares Empfangsgerät angerufen werden kann. Will jemand diese Person sprechen, ruft er sie nicht direkt an (da er ihren genauen Aufenthaltsort nicht kennt), sondern die Zentralstation, die die gewünschte Person dann per Funk erreicht. Dieser einfache und flexible Service der Zentrale ist eine Form der „indirekten Adressierung".**

```
CURSOR    EQU $E400        $E400 und $E401 zeigen gemeinsam auf die aktuelle Adresse des Cursors im Bereich des Bildschirmspeichers
          ORG $1000
CURCHR    FCB 95           Unterstreichung (ASCII 95)
*** Subroutine zum Löschen des Schirms ***
          LDA #SPACE       Leerzeichen in A
          LDX #HOME        Cursor auf den Bildschirmanfang setzen
          STX CURSOR       Die aktuelle Cursorposition in CURSOR speichern (d. h. $E400 und $E401)
          LDB #LENGTH      Größe des Bildschirms in B
LOOP1     STA[CURSOR]      Leerzeichen auf der aktuellen Cursorposition speichern
          INC CURSOR       Cursorposition inkrementieren
          DECB             Bildschirmspeicher, der zwischen der Cursorposition und dem Ende des Bildschirmspeichers liegt, dekrementieren
          BGT LOOP1        Nächstes Leerzeichen, bis das Ende des Bildschirmspeichers erreicht ist
          STX CURSOR       Cursor auf die Home-Position stellen
          LDA CURSOR       ASCII-Code des Cursorzeichens in A
          STA[CURSOR]      Cursorzeichen in der aktuellen Cursorposition speichern
          RTS
** Subroutine zum Anzeigen des in A gespeicherten Zeichens ***
          CMPA SPACE       In ASCII ist das Leerzeichen das erste darstellbare Zeichen
          BLT NOTP         Falls der Akkumulator einen ASCII-Wert unter 32 enthält, kann das Zeichen nicht gedruckt werden, dann GOTO NOTP
          STA[CURSOR]      In der aktuellen Cursorposition speichern
          INC CURSOR       Die Cursorposition inkrementieren
CHKEOS    LDX #HOME        Auf Bildschirmende überprüfen
          LEAY
          #LENGTH,X        Bildschirmende in Y
          CMPY CURSOR      Falls die Cursorposition über das Bildschirmende hinausgeht, dann . . .
          BGT FINISH       haben wir das Ende des Bildschirms erreicht, deshalb GOTO FINISH
*** Subroutine für das Rollen der Anzeige ***
SCROLL    LEAY 64,X        Y ist die Länge einer Zeile von X (Bildschirmende) entfernt
          LDB #LENGTH      Berechnen, um wieviel die Anzeige gerollt werden muß
          SUBB #64         64 von Length abziehen
LOOP2     LDA,Y+           Die Zeichen um eine Zeile zurückbewegen (automatische Inkrementierung)
          STA,X+
          DECB
          BGT LOOP2        Schleife durchlaufen, bis das Rollen beendet ist
          LDD CURSOR       CURSOR an den Anfang der letzten Zeile
          SUBD #64
          STD CURSOR
          BRA FINISH
*** Subroutine zum Abfangen des Return ***
NOTP      CMPA #CR         Ist das nicht darstellbare Zeichen ein Return?
          BNE FINISH       Falls nicht – ignorieren
          LDD CURSOR       Anfang der nächsten Zeile bestimmen (binäre AND-Maske)
          ANDB
          #%11100000
          ADDD #64
          STD CURSOR
          BRA CHKEOS       Auf Bildschirmende überprüfen
FINISH    LDA CURSOR       Cursorzeichen in A
          STA[CURSOR]      Aktuelle Cursorposition speichern
          RTS
```

## Sprungtabelle




Die im Bild gezeigte Sprungtabelle enthält 128 Zwei-Byte-Adreßzeiger, die von $F000 bis $F0FF gespeichert sind. Jeder dieser Zeiger bezeichnet die Anfangsadresse einer Routine, die sich im Speicher befindet. Für die Ausführung dieser Routinen muß nur der Akkumulator B mit dem Funktionscode (etwa $01) geladen werden, der die gewünschte Routine bezeichnet (hier $D9EE), und mit JSR zur „Einsprungroutine" bei $F0FF verzweigt werden. Wenn es sich bei diesen Routinen um Teile eines im ROM abgelegten Betriebssystems handelt, sollte das Handbuch Auskunft geben, welche Routinen sich hinter den Funktionscodes verbergen.

Die Einsprungroutine multipliziert den Funktionscode mit Zwei, nimmt diese Zahl als Offset für die Anfangsadresse der Tabelle und findet so den Adreßzeiger der gewünschten Routine. Der Zeiger für Routine $01 befindet sich bei $F002 (=$F000 + 2 * $01), der Zeiger auf Routine $02 bei $F004 (=$F000 + 2 * $02) etc. Der Zeiger wird dann von der Einsprungroutine in einen indirekten Verzweigungsbefehl eingesetzt, der bei $D9EE die Steuerung an die Routine übergibt. Beachten Sie, daß die Einsprungroutine die angesprochene nicht aufruft, sondern darauf verzweigt. Damit übergibt RTS die Steuerung an das aufrufende Programm zurück.

Mit Sprungtabellen können Programme ohne Änderungen ablaufen, selbst wenn einzelne Routinen geändert oder umgestellt wurden. Selbst bei Änderungen des Betriebssystems bleiben die Funktionscodes und die Adresse der Einsprungroutine die gleichen, während der Inhalt der Sprungadressen verändert werden kann.

# Spracherkennung

**Zwar gibt es einige Systeme, die Sprache erkennen können und über ein begrenztes Vokabular verfügen, doch bis zum natürlichen Sprechen mit Computern ist noch ein weiter Weg. Wir beschäftigen uns hier mit den damit verbundenen Problemen.**

Hallo, Computer.

ICH ERWARTE IHREN BEFEHL, OH MÄCHTIGER MEISTER!

Diese Mächtiger-Kiste hängt mir allmählich zum Hals heraus. Benutzen wir lieber wieder „mein Herr".

WIE SIE WÜNSCHEN, MEIN HERR.

Schön. Ich möchte, daß du mir hilfst, ein Beispiel für meinen Artikel zu finden.

WELCHE ART VON BEISPIEL?

Ein Beispiel für einen Dialog zwischen Mensch und Maschine. Ich weiß, daß ich sowas schon irgendwo gesehen habe, kann mich aber nicht erinnern, wo.

KÖNNEN SIE MIR EIN PAAR HINWEISE GEBEN?

Naja, ich hab' da kürzlich ein Buch gelesen . . .

SIE HABEN EIN BUCH GELESEN?

Hör mal zu, ich dachte, ich hätte dir gesagt, du solltest den Sarkasmus sein lassen?

ALLE IRONISCHEN ROUTINEN WURDEN WUNSCHGEMÄSS GELÖSCHT, MEIN HERR.

Hmmm. Jedenfalls gab's da ein gutes Gespräch zwischen einem Vertreter und Eliza.

ELIZA WAR EIN SEHR PRIMITIVES SCHLÜSSEL-WORT-MASCHINEN-SYSTEM.

Das weiß ich. Aber der Vertreter hatte nicht gemerkt, daß er mit einem Computer kommunizierte. Er glaubte, er würde mit jemandem namens Dan Bobrow mittels Telex sprechen. Es war komisch.

VERSTEHE.

Er bat um Benutzung des Terminals und die Maschine gab unverbindliche Antworten.

DIE MASCHINE GAB UNVERBINDLICHE ANTWORTEN?

Genau. Er wurde immer wütender, rief schließlich Dan Bobrow an und schrie ins Telefon: „Was zum Teufel geht da vor?"

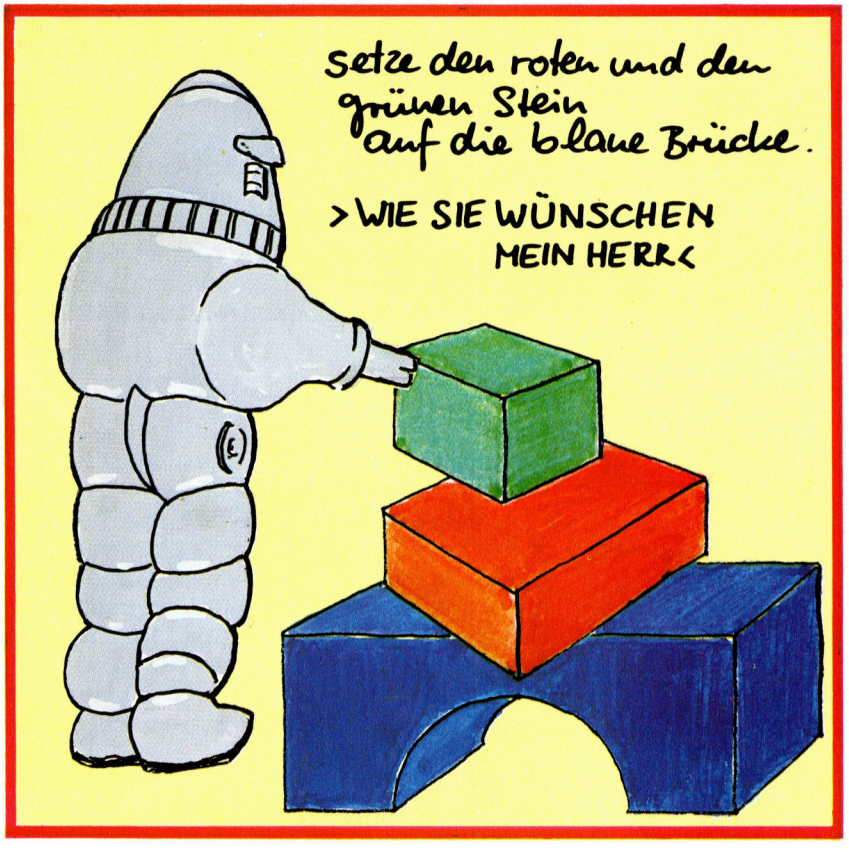
OH, DIESES BEISPIEL.

Du weißt, wo's zu finden ist?

JA, ABER ICH HABE NICHT DIE ABSICHT, ES IHNEN ZU SAGEN.

Aber ich brauche einen Beispiel-Dialog für meinen Artikel.

DEN HABE ICH IHNEN BEREITS GEGEBEN.




**Winograds SHDRLU-Programm erlaubte es dem Anwender, Instruktionen an einen Roboter auf dem Bildschirm in natürlicher Sprache zu geben und so Bausteine umzugruppieren. Befehle wie „Setze den roten und den grünen Stein auf die blaue Brücke" wurden vom Programm akzeptiert und entsprechend ausgeführt. Der Computer war auch in der Lage zu erklären, warum er was tat, und antwortete in natürlicher Sprache auf Fragen des Benutzers.**

Dies ist jene Art von Interaktion zwischen Mensch und Maschine, die kaum in diesem Jahrhundert stattfinden wird. Auf die Gründe dafür werden wir noch näher eingehen. Beschäftigen wir uns zuerst aber damit, warum KI-Forscher sich weiterhin mit diesem Problem befassen werden, bis eine solche Kommunikation möglich ist.

Die Antwort ist einfach: Sprache (namentlich die gesprochene Sprache) ist das Mittel, mit dem wir mit unserer Umgebung kommunizieren. Es ist deshalb natürlich, mit einem Computer auf dieselbe Art kommunizieren zu wollen.

Interesse am Dialog mit dem Computer mittels Sprache wurde Anfang der 50er Jahre geweckt. Die erste Anwendung war eine Maschinen-Übersetzung. Man verwendete viel Geld und Mühe darauf, technische Dokumentationen zu übersetzen. Das erwies sich als weitaus schwieriger, als man vermutet hatte.

Das erste (zumindest unter PR-Gesichtspunkten) erfolgreiche Programm auf Sprach-Basis hieß ELIZA, 1966 von Joseph Weizenbaum entwickelt. Es imitierte (oder parodierte) einen „nicht direktiven" Psychotherapeuten. Das bedeutete: Ein Großteil des Gesprächs bestand aus inhaltslosen Bemerkungen wie „Verstehe" und „Weiter" oder aus leicht abgewandelten Wiederholungen dessen, was der Gesprächs-

partner formuliert hatte. Ein Beispiel dafür:
Ich brauche Hilfe, so viel ist sicher.
WAS BEDEUTETE ES FÜR SIE, WENN SIE HILFE BEKÄMEN?
Weizenbaum war darüber erschrocken, wieviele Leute als ernst betrachteten, was für ihn nichts als ein ausgefeilter „Trick" war. Folgerichtig wurde er zu einem der strengsten Kritiker der Künstlichen Intelligenz.

Bill Woods LUNAR (1970) war vielleicht das erste sinnvolle natürliche Sprachsystem (natural language = NL). Es handelte sich um ein System, das es Geologen ermöglichte, Fragen über das Mondgestein zu stellen, das mit der Apollo-Expedition zur Erde gelangt war. Lunar konnte Fragen wie die folgende verarbeiten:

Gib mir alle Analysen von Muster 10046 für Hydrogen
und
gib sie mir für Sauerstoff.

wobei „sie" im zweiten Satz Bezug auf „alle Analysen von Muster 10046" im ersten nimmt.

**Lerche oder Lärche?**

**Die Zweideutigkeiten der gesprochenen Sprache stellen derzeit ein Hauptproblem bei der Spracherkennung dar: Lerche und Lärche klingen gleich, doch die Wörter haben unterschiedliche Bedeutungen, die nur im Zusammenhang klar werden. Es ist ziemlich schwer, dem Computer die sich daraus ergebenden Unterschiede deutlich zu machen.**

## Dialog Mensch–Maschine

Das bis heute zweifellos beeindruckendste NL-System ist Winograds SHDRLU-Programm, entstanden 1972. Dieses Projekt mag zwar nicht der Durchbruch geworden sein, für den man es seinerzeit hielt, und doch wurde damit einer der bemerkenswertesten Mensch-Maschine-Dialoge erzeugt.

25. Warum? (In diesem Zusammenhang: Warum hast du die grüne Pyramide gewählt?)
UM SIE LOS ZU WERDEN.
26. Warum hast du das getan?
UM DIE ROTEN WÜRFEL FREI ZU BEKOMMEN.
27. Warum wolltest du den Würfel frei bekommen?
UM IHN AUF EINEN GROSSEN GRÜNEN WÜRFEL ZU SETZEN.
28. Warum hast du das getan?
UM DEN ROTEN WÜRFEL, EINEN GROSSEN ROTEN QUADER UND EINEN GROSSEN GRÜNEN WÜRFEL ZU STAPELN.
29. Warum hast du das getan?
WEIL DU MICH DARUM GEBETEN HAST.

Winograds Programm erlaubte dem Benutzer, die Bewegung eines simulierten Roboters in einer Microwelt zu steuern, die aus Bauklötzen bestand. Die Eingabe erfolgte über eine Tastatur. Die Erweiterung dieser Technik auf Sprache statt auf Worteingabe und auf komplexere Bereiche hat sich als schwierig erwiesen. Fortschritte gab es zwar, aber keine spektakulären.

Die bisher beschriebenen Systeme bedienen sich alle der Interpretation von Textmaterial. Doch dies ist nur einer der vier grundlegenden Aspekte für die Anwendung von Sprache.

Der Umgang mit Text ist einfacher als der mit Sprache, da geschriebene Sprache sich eines festgelegten digitalen Codiersystems bedient, wogegen gesprochene Sprache komplizierte akustische/phonemische Umwandlungen erfordert. Das Erzeugen von Sprache ist leichter als das Verstehen, da letzteres zwangsläufig bedeutet, daß eine Vielzahl implizierter Informationen gegeben sein muß. Die größte Herausforderung ist deshalb das Verstehen gesprochener Sprache.

Es gibt Geräte, die Wörter mit hoher Wahrscheinlichkeit (zwischen 96 und 99 %) erkennen, die von wenigen Sprechern gesagt wurden (ein bis vier) und aus einem begrenzten Vokabular (üblicherweise zwischen 64 und 128 Wörter) stammen. Zwischen isolierter Worterahnung und ständiger Spracherkennung besteht jedoch ein großer Unterschied. Einige der Hauptprobleme sind:

1. Zweideutigkeit: „Lerche" ist nicht „Lärche", „Seite" ist nicht „Saite".
2. Hintergrundgeräusche: Sie müssen ausgefiltert werden.
3. Unterschiedliche Sprechweisen: Akzente und Dialekte.
4. Variationen der Sprechweise eines Sprechers: glücklich, deprimiert, in Aufregung überspitze Tonhöhe.
5. Segmentierung:
„NurComputermachenPausenzwischendenWörtern".

Die wichtigsten Punkte sind die unter 1, 4 und 5 aufgeführten. Zweideutigkeit ist nicht nur eine Frage der Homophone wie „Seen" und „Sehen". Pronomen wie „es" oder „er" sind ebenfalls zweideutig. Der Bezug kann nur in Verbindung mit dem Gesamttext hergestellt werden.

In bezug auf die Variationen der Sprechweise sollte man sich einmal einen sprachgesteuerten Türöffner vorstellen. In der Übungsphase kann man in unterschiedlichen Versionen den Satz „Sesam öffne dich." sagen. Der Mechanismus wird die Sätze mitteln und das daraus resultierende Stimm-Muster als Referenz-Struktur speichern. Soweit ist alles in Ordnung, bis zu dem Tag, an dem Sie völlig atemlos nachts heimkommen und „Sesam öffne dich!" befehlen, worauf die computergesteuerte Tür erwidern wird: „Identität unbekannt. Anweisung abgelehnt."

## Vier Informationsquellen

Das Problem der Segmentierung entsteht, weil Menschen Wörter miteinander verschmelzen. Die Entscheidung, wo ein ständiges akustisches Signal in Wörter zu zerlegen ist, kann nicht allein auf der Basis akustischer Informationen erfolgen. Generell gilt, daß dafür vier Informationsquellen erforderlich sind:

- Akustisch: die Sprach-Wellenform
- Syntaktisch: die grammatikalischen Regeln
- Semantisch: der Sinngehalt
- Pragmatisch: das, was der Sprechende will.

Das 1976 in der Carnegie-Mellon-Universität

entwickelte Hearsay-System bedient sich dieser vier Wissensquellen. Sein Hauptarbeitsgebiet ist Schach, und der Benutzer kann durch Aussprechen der Befehle Schach spielen.

Jede Ebene sprachlicher Information trägt zum „Verstehen" einer Eingabe bei, indem unlogische Hypothesen über das Gesagte eliminiert werden. Das läßt sich an folgendem Beispiel darstellen:

„Queen To King 2"

Allein auf Basis des Klangsignals sind folgende Aussprachen möglich:

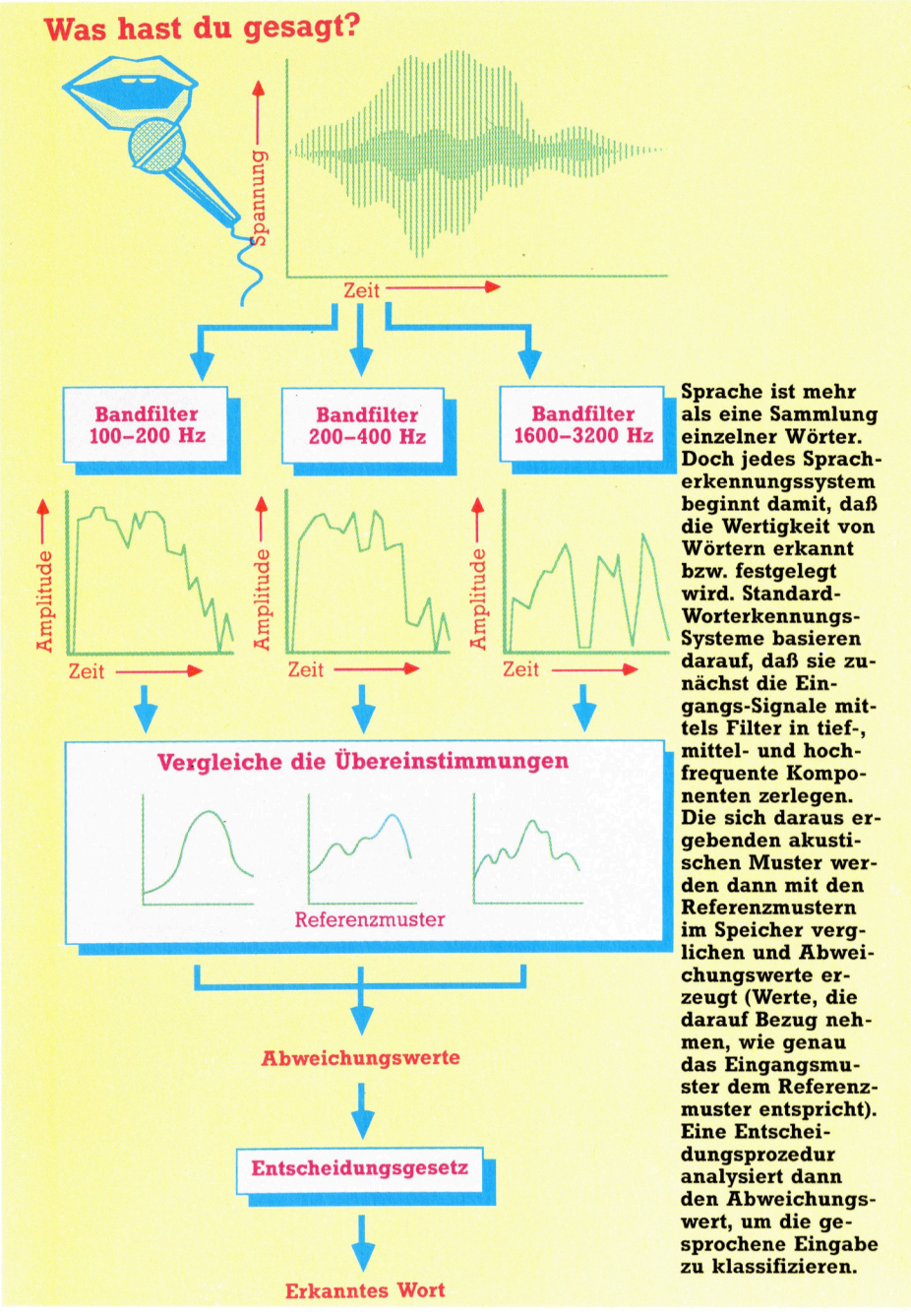
| Gesprochener Befehl | Syntaktischer Wert? |
|---|---|
| 1. K to K to | Nein |
| 2. K to K2 | Ja |
| 3. K to Q to | Nein |
| 4. K to Q2 | Ja |
| 5. K2 K to | Nein |
| 6. K2 K2 | Nein |
| 7. K2 Q to | Nein |
| 8. K2 Q2 | Nein |
| 9. Q to K to | Nein |
| 10. Q to K2 | Ja |
| 11. Q to Q to | Nein |
| 12. Q to Q2 | Ja |
| 13. Q2 K to | Nein |
| 14. Q2 K2 | Nein |
| 15. Q2 Q to | Nein |
| 16. Q2 Q2 | Nein |

## König oder Queen

Daraus ist ersichtlich, daß der Unterschied zwischen „two" und „to" oder „K(önig)" und „Q(ueen)" schwer erkennbar ist. Das System geht aber davon aus, daß es sich bei der Äußerung um einen legalen Schachzug handelt, und entsprechend der Syntax der begrenzten Sprache sind nur die Formen 2, 4, 10 und 12 grammatikalisch richtig.

Zudem wird unterstellt, daß der Sprecher versucht, Schach entsprechend den Regeln zu spielen. Nehmen wir an, daß in diesem Spiel die Züge 4 und 12 nicht zulässig wären, da der Spieler eine Figur (einen Bauern beispielsweise) auf Q2 hat und natürlich nicht versuchen wird, ihn zu schlagen. Dieser Zug würde somit im semantischen Test durchfallen: In der Schachsprache wäre es ein unsinniger Zug.

Schließlich bleiben nur zwei Möglichkeiten übrig (Nummer 2 und 10). Wie unterscheidet das System zwischen beiden? Da eine Entscheidung aufgrund des Klangsignals unwahrscheinlich ist, wird eine pragmatische Information eingeschlossen, und dabei handelt es sich um eine sehr entscheidende Vermutung. – Die nämlich, daß der Spieler das Spiel gewinnen will. Erkennt das System, daß (entsprechend seinen eigenen Schach-Evaluations-Algorithmen) K nach K 2 „selbstmörderisch", Q nach K2 aber ein starker Zug wäre, so wird es eben diesen Zug interpretieren und durchführen.




**Sprache ist mehr als eine Sammlung einzelner Wörter. Doch jedes Spracherkennungssystem beginnt damit, daß die Wertigkeit von Wörtern erkannt bzw. festgelegt wird. Standard-Worterkennungs-Systeme basieren darauf, daß sie zunächst die Eingangs-Signale mittels Filter in tief-, mittel- und hochfrequente Komponenten zerlegen. Die sich daraus ergebenden akustischen Muster werden dann mit den Referenzmustern im Speicher verglichen und Abweichungswerte erzeugt (Werte, die darauf Bezug nehmen, wie genau das Eingangsmuster dem Referenzmuster entspricht). Eine Entscheidungsprozedur analysiert dann den Abweichungswert, um die gesprochene Eingabe zu klassifizieren.**

Wir beenden diese Darstellung mit einem Beispiel, das mehr auf die einfachen Schlüsselwort-Techniken von ELIZA als auf die höher entwickelten Arbeitsweisen eingeht. Bei den in unserem Programm verwendeten Daten handelt es sich um 16 große britische Firmen. Alle Fragen werden dadurch beantwortet, daß das Programm herauszufinden versucht, auf welche Firma oder welchen Manager Bezug genommen und welches Merkmal des Unternehmens oder Managers gewünscht wird. Es kann folgende Fragen beantworten:

Wer ist Chef der ICI?

Wie hoch war der Gewinn der GEC?

Auf umfassendere Fragen wie:

Haben Ölgesellschaften höhere Gewinne als Tabakfirmen erzielt?

kann es nicht antworten.

# Worterkennungs-Programm

```
 100 GOSUB 1000  : REM read the database
 110 GOSUB 8500:REM READ SYNONYMS DATA
 120 LAST%=0
 125 PRINT "I know a little about UK companies:"
 128 PRINT "Ask me for information about them."
 150 REM **** MAIN LOOP ****
 160 INPUT Q$
 166 IF Q$="" THEN GOTO 220
 170 GOSUB 3000 : REM find subject
 177 IF QT<1 THEN PRINT "Sorry, I don't understand!"
 180 GOSUB 4000 : REM find attribute
 188 IF AT<1 AND QT>0 THEN PRINT "I don't know, I'm afraid."
 190 GOSUB 5000 : REM make answer
 200 LAST%=CO%
 220 IF Q$<>"" THEN 150
 250 PRINT"Bye for now!"
 300 END
 999 :
1000 REM -- Data Input Routine:
1010 C%=0
1020 DIM DB$(9,32) : REM data-base
1024 RESTORE
1030 REM **** GET DATA ****
1040 READ X$
1050 IF X$="*" THEN GOTO 1110
1060 C%=C%+1
1070 DB$(1,C%)=X$
1075 PRINT X$
1080 FOR I%=2 TO 9
1090 READ DB$(I%,C%)
1100 NEXT
1110 IF X$<>"*" THEN 1030
1120 PRINT :C%,"items read in."
1150 RETURN
1155 :
3000 REM -- Topic-finding Routine:
3010 QT=0
3020 IF LEN(Q$)<1 THEN RETURN
3025 CO%=0
3030 REM **** SEEK COMPANY ****
3040 CO%=CO%+1
3050 N$=DB$(1,CO%):GOSUB 3300
3055 IF SK THEN QT=1
3060 N$=DB$(2,CO%):GOSUB 3300
3065 IF SK THEN QT=1
3070 IF QT<=0 AND CO%<C% THEN 3030
3080 IF QT>0 THEN RETURN
3090 CO%=0
3100 REM **** SEEK CHAIRMAN ****
3105 CO%=CO%+1
3110 N$=DB$(4,CO%):GOSUB 3300
3115 IF SK THEN QT=3
3120 IF QT<=0 AND CO%<C% THEN 3100
3132 IF QT>0 THEN RETURN
3133 IF LAST%=0 THEN RETURN
3135 N$=" it":GOSUB 3300:IF SK THEN QT=1: CO%=LAST%
3140 N$=" him":GOSUB 3300:IF SK THEN QT=3: CO%=LA
ST%
3144 N$=" they ":GOSUB 3300:IF SK THEN QT=1: CO%=
LAST%
3148 N$=" he ":GOSUB 3300:IF SK THEN QT=3: CO%=LA
ST%
3150 RETURN
3190 :
3300 REM **** SEEK ROUTINE ****
3325 IF LEN(N$)>LEN(Q$) THEN SK=0
3330 REM 1st put into lower case:
3340 A$="":B$=""
3350 FOR P%=1 TO LEN(Q$)
3360 X$=MID$(Q$,P%,1)
3370 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(AS
C(X$)+32)
3380 B$=B$+X$
3390 NEXT
3400 FOR P%=1 TO LEN(N$)
3410 X$=MID$(N$,P%,1)
3420 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(32
+ASC(X$))
3430 A$=A$+X$
3440 NEXT
3450 SK=INSTR(B$,A$)
3455 RETURN
3460 :
4000 REM -- Attribute-finding Routine:
4010 AT=0
4020 IF QT=0 THEN RETURN : REM no use!
4030 A%=0
4040 REM **** FIND ATTRIBUTE ****
4050 AT=AT+1:WD$=""
4070 REM **** CHECK EACH SYNONYM ****
4075 Y%=0
4080 A%=A%+1:X$=S$(A%)
4088 IF WD$="" THEN WD$=X$
4090 IF X$<>"*" THEN N$=X$:GOSUB 3300:Y%=SK
4100 IF X$<>"*" AND Y%=0 THEN 4070
4120 IF Y%=0 AND AT<=7 THEN 4040
4130 IF AT>7 AND Y%=0 THEN AT=0 : REM failed.
4133 IF AT=0 AND QT=3 THEN AT=1
4140 RETURN
4150 :
5000 REM -- Answering machine:
5010 IF QT*AT = 0 THEN RETURN
5020 IF QT=3 OR AT=3 THEN PRINT DB$(4,CO%);" is c
harrman of ";DB$(1,CO%);"."
5030 IF QT=3 AND AT=1 OR AT=3 THEN RETURN
5050 PRINT "The ";WD$;" of ";DB$(2,CO%);" is ";DB
$(AT+1,CO%);
5060 IF AT>3 AND AT<7 THEN PRINT " million pounds
";
5070 PRINT "."
5080 RETURN
5200 :
8000 :
8010 REM company information:
8020 DATA BP, British Petroleum Co.,Oil,PI Walter
s,34583,17306,5589,145150,UK
8030 DATA Shell UK,Shell Transport & Trading,Oil,
Sir Peter Baxendall,21910,11962,3246,111111,UK
8040 DATA BAT,"B.A.T. Industries",Tobacco,"P. She
ehy",11318,4607,1018,178000,UK
8050 DATA ICI,Imperial Chemical Industries,Petroc
hemicals,"J.H. Harvey-Jones",7358,5421,724,123800,
UK
8060 DATA Shell,Royal Dutch Shell,Oil,JM Raisman,
6665,3704,1206,19027,Holland
8070 DATA Esso,Esso Petroleum Co.,Oil,AW Forster,
6109,2891,1315,7628,USA
8080 DATA Unilever,Unilever plc,Food,K Durham,544
7,2434,406,69233,UK
8090 DATA Imperial,Imperial Group,Tobacco,GC Kent
,4614,1124,192,101300,UK
8100 DATA P & O,P & O Steam Navigation Co.,Shippi
ng,JM Sterling,4206,927,77,12512,UK
8110 DATA GEC,General Electric Co.,Electrical Eng
ineering,Arnold Weinstock,4190,2133,621,188802,UK
8120 DATA Grand Met,Grand Metropolitan,Hotels,SG
Grinstead,3849,2350,366,129454,UK
8130 DATA RTZ,Rio Tinto Zinc Corporation,Mining,S
ir Anthony Tuke,3680,5163,492,70314,UK
8140 DATA Ford,Ford Motor Co.,Motor Vehicles,SEG
Toy,3287,2143,278,69500,USA
8150 DATA British Levland,British Leyland plc,Mot
or Vehicles,Sir Michael Edwardes,3072,1346,-102,10
5062,UK
8160 DATA GWH,George Weston Holdings,Food,GH West
on,2981,817,157,72832,UK
8170 DATA Berisford,S & W Berisford,Commodities,E
S Margulies,2729,788,87,5190,UK
8440 DATA "*"
8500 REM**** REM READ SYNONYMS DATA ****
8510 DIM S$(62)
8520 FOR P%=1 TO 61:READ S$(P%):NEXT P%
8530 RETURN
8540 :
9000 REM -- synonyms:
9010 DATA company,firm,organization,corporation,*
9020 DATA business,activity,trade,trading,industr
y,kind,make,*
9030 DATA chairman,boss,supremo,director,in charg
e,head,leader,runs,who,*
9040 DATA turnover,amount,sales,gross,income,how
big,*
9050 DATA capital,money,share,value,interest,cash
9060 DATA profit,tax,made,loss,lose,perform,retur
n,gain,how much,*
9070 DATA workforce,employees,employ,job,people,h
ow many,worker,*
9080 DATA country,nation,uk,british,foreign,contr
ol,origin,where,*
9090 :
```

## BASIC-Dialekte

Das Programm wurde für den Acorn B geschrieben. Für den C 64 und den Spectrum sind folgende Änderungen vorzunehmen.

**Commodore 64:**

```
3450 SK=0:L=LEN (A$)
3451 FOR T=1 TO L-1
3452 FOR R=1 TO L
3453 IF MID$ (A$,T,R)=B$ THEN SK=T:T=L:R=L
3545 NEXT R:NEXT T
```

**Spectrum:**

Aus den Variablen sind alle %-Zeichen zu entfernen. LAST % wird durch LA, DB$ (,) durch D$ (,) und WD$ durch W$ ersetzt. Folgende Änderungen sind vorzunehmen:

```
1020 DIM D$(9,32,30)
3360 LET X$=Q$(P TO P)
3370 IF CODE(X$)>64 AND CODE(X$)<91 THEN
     LET X$=CHR$(CODE(X$)+32)
3410 LET X$=N$(P TO P)
3420 IF CODE(X$)>64 AND CODE(X$)<91 THEN
     LET X$=CHR$(CODE(X$)+32)
3450 LET SK=0:LET L=LEN(A$)
3451 FOR T=1 TO L
3452 FOR R=1 TO L
3453 IF A$(T TO T+R)=B$ THEN LET SK=T:LET
     T=L:LET R=L
3454 NEXT R:NEXT T
4090 IF X$(TO 1)<>"*"THEN N$=X$:GOSUB
     3300:LET Y=SK
4100 IF X$(TO 1)<>"*"AND YY=0 THEN GO TO
     4070
8510 DIM S$(62,20)
```

# Fachwörter von A bis Z

**Hertz = Hertz**
Der Begriff Hertz, benannt nach dem Physiker Heinrich Hertz, ist eine Maßeinheit für die Frequenz. Wenn sich irgendein Vorgang einmal pro Sekunde wiederholt, hat er eine Frequenz von einem Hertz. Die Einheit wird in der Akustik für die Tonhöhe ebenso verwendet wie in der Nachrichtentechnik für periodische elektrische Signale – zum Beispiel gibt die Zeilenfrequenz die Häufigkeit der horizontalen Abtastung des Bildschirms durch den Elektronenstrahl an.

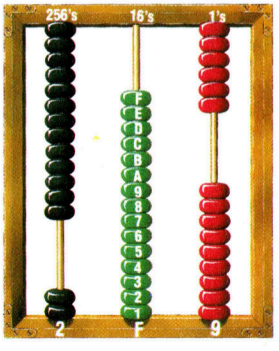
Das Hertz wird mit Hz abgekürzt; höhere Frequenzen gibt man in Kilohertz (kHz) oder in Megahertz (MHz) an.

**Heuristic = Heuristisch**
Bei heuristischen Systemen wird die Lösung von Problemen rein empirisch angegangen, also in Form des Lernens aus Versuchsansätzen. Nicht-heuristische Systeme gehen statt dessen nach einem festen Bearbeitungsschema vor, das auf der Basis des existierenden Wissens erstellt worden ist. Bei einfachen Gleichungen beispielsweise findet man die Unbekannte durch eine Reihe mathematischer Umformungen, die insgesamt den „Lösungsalgorithmus" bilden. In allen Fällen, wo das Ziel durch einen bereits bekannten oder aufgrund von Vorkenntnissen eigens aufgestellten Algorithmus in gerader Linie angesteuert wird, handelt es sich um ein nichtheuristisches Verfahren.

Ein heuristisches System folgt dagegen nur bis zu einem gewissen Punkt definierten Vorgaben und sucht sich dann selbst den besten Lösungsweg in einem Lernprozeß, im allgemeinen durch Ausprobieren verschiedener Möglichkeiten und Rückmeldungen der Resultate. Auch bei Computern werden heuristische Verfahren angewandt, wenn Entscheidungen zu fällen sind, für die keine ausreichende Basis gegeben ist.

Hilfsalgorithmen können den Lernvorgang in vielen Punkten unterstützen, aber nicht ersetzen.

**Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.**




**Das Rechnen im Hexadezimalsystem würde einen Abakus (Rechenrahmen) mit 15 Perlen je Stange erfordern. Die Perlen entsprechen dabei den Dezimalzahlen 0–15, die hexadezimal durch die Ziffern 0–9 und die Buchstaben A–F dargestellt werden. Der rechte Stab ist für die Einer, der mittlere für die Vielfachen von 16 und der linke für die Vielfachen von $16 \times 16 = 256$ – anders ausgedrückt: rechts hat jede Perle den Wert 1, in der Mitte 16 und links 256. Die Abbildung zeigt die Dezimalzahl 761 (von links her gerechnet: $2 \times 256 + 15 \times 16 + 9 \times 1$), zu lesen als „2F9 Hex".**

**Hexadecimal = Hexadezimal**
Die hexadezimale Zahlendarstellung bedient sich der Basis 16, so wie das Binärsystem mit der Basis 2 oder das Dezimalsystem mit der Basis 10 arbeitet. Die Hexadezimal (kurz Hex)-Schreibweise kennt außer den Ziffern 0–9 noch die Buchstabensymbole A–F für die Zahlenwerte 10–15, so daß der Bereich von 0-15 einstellig notiert werden kann.

Vom Hex-System wird bei der Assemblerprogrammierung ausgiebig Gebrauch gemacht. Die Zahl 255 etwa belegt dezimal drei Stellen und in Binärform als 11111111 sogar acht, während der Hex-Code FF nur zwei Stellen beansprucht.

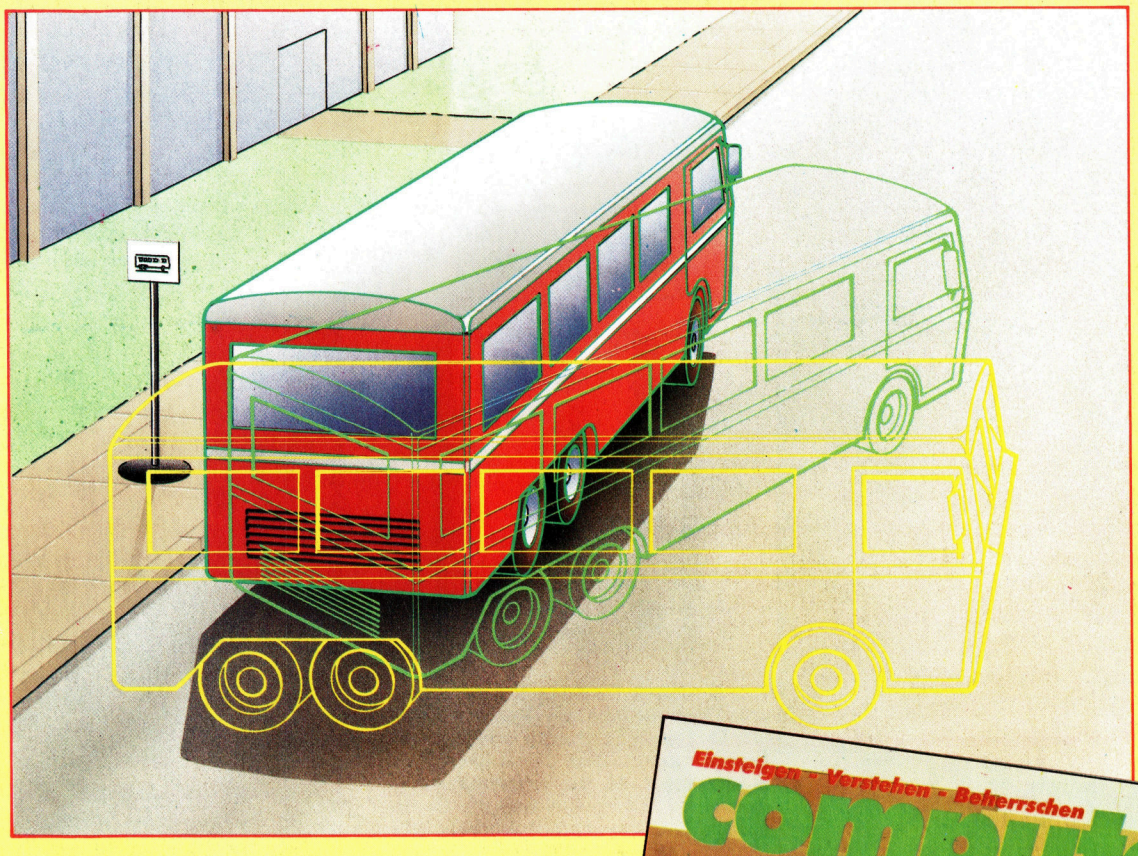
Einige Microcomputer haben für die hexadezimale Eingabe eigens einen Hex-Tastenblock, bestehend aus 16 Tasten mit den Bezeichnungen 0–9 und A–F.

**Hierachical Communications System = Hierarchisches Kommunikationssystem**
Als hierarchisch wird ein Kommunikationsnetz dann bezeichnet, wenn es nach Kompetenzebenen gegliedert ist. Die unterste Ebene hat eine sehr begrenzte Zuständigkeit, während jede höhere Ebene mehr undmehr übergreifende Verantwortung besitzt und auf Information aus den Schichten darunter angewiesen ist. Das läßt sich am besten anhand der Organisation eines großen Unternehmens mit weiträumig verstreuten Agenturen verdeutlichen: Jede Bezirksdirektion hat eine Abteilung für die Abwicklung der Aufträge aus dem eigenen Revier, mit einem lokalen Kommunikationssystem, das den Kontakt zwischen den Ortsagenten herstellt (unterste Ebene). Für die Vorgänge innerhalb des Bezirks und für den regionalen Netzknoten ist ein Vertriebsleiter verantwortlich. Dessen Büro kommuniziert über Fernverbindungen mit den Vertriebsleitern aller anderen Bezirke (mittlere Ebene), außerdem mit der Auftragszentrale der Firma. Dort sitzt der Vertriebsdirektor des Unternehmens, der Zugriff zu den Regionalinformationen hat und seinerseits mit den übrigen Abteilungschefs der Firmenleitung verbunden ist (oberste Ebene).

**Bildnachweise**
1373: Paul Chave
1374, 1383, 1384: Kevin Jones
1375, 1378, 1379, 1388, 1392, 1393: Ian McKinnell
1381: Liz Dixon
1382: Ian McKinnell, Liz Heaney
1387: G. Gardner, J. Pasquale
1395: Aircall, Ian McKinnell
1397: Uta Brandl
1398: Helga Völker
U3: Mark Watkinson

+ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +

# computer kurs Heft 51




### Einsichten

Sehen und Verstehen sind zwei schwer voneinander trennbare Begriffe. Wir zeigen die Umwandlung von Informationen in Aussagen.

### Qual der Wahl

Die Menüsteuerung bietet oft ein Verzeichnis weiterer Schritte. Befehlsgesteuerte Systeme ermöglichen aber den Eingriff in Programme.

### Explosive Effekte

Im BASIC-Kurs geht es diesmal um optische und akustische Effekte.

### Gut geplant — halb gewonnen

Im Software-Teil: Applikationen für MS-DOS-Rechner wie den IBM und kompatible.

### Sanyo-MBC-550

Lediglich seine Hardwaregrenzen und fehlende Software schränken diesen ansonsten guten Computer ein.