Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

# computer kurs


Planungen mit BrainStorm

Flußdiagramme benutzen

Der Psion Organiser

Suchstrategien beim Schach

Heft 45

Ein wöchentliches Sammelwerk

# computer kurs

## Heft 45

## Inhalt


### Hardware

**Gut organisiert** 1233
Der Taschencomputer Psion Organiser

### Tips für die Praxis

**Entscheidungen** 1236
Flußdiagramme erleichtern das Programmieren

**Schrittweise** 1251
Aufbau und Steuerung der Motoren

### Computer Welt

**Vorausschau** 1238
Künstliche Intelligenz beim Schachspiel

**Vielfältig** 1244
Vom Tonbandkopieren zum Softwarevertrieb

### Software

**Listenreich** 1242
„BrainStorm" von Caxton Software

**Relax, do it** 1254
Ein Spiel über Frankie Goes To Hollywood

### Peripherie

**Torch Disk Pack** 1245
Z80-Prozessor und Diskdrive für den Acorn B

### PASCAL

**GOTO ENDE** 1248
Zum Schluß: die Suchbaum-Strategie

### Bits und Bytes

**Register ziehen** 1255
Aufgaben innerhalb des 6809

### BASIC 45

**Befehlseingabe** 1258
Weiter geht's im Abenteuerspiel

---

**Fachwörter von A—Z**


## 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

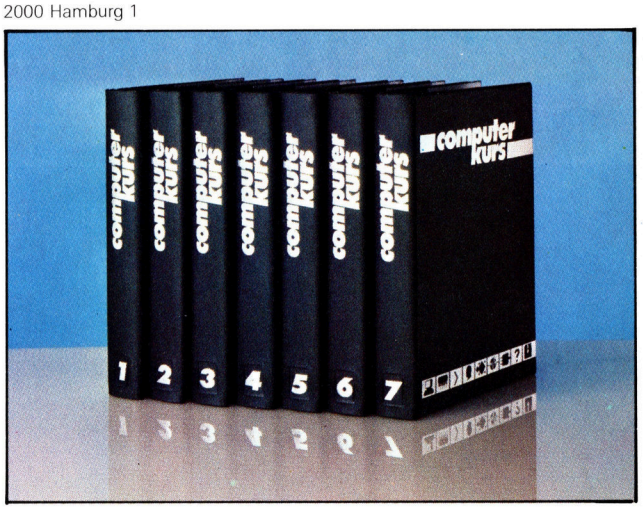
**Schweiz:** Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

## 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

# Gut organisiert

**Psion stellte auf dem Hardwaremarkt den „Organiser“ vor. Dieser Taschencomputer hat vielfältige Möglichkeiten, Daten zu speichern und wieder abzurufen, ist universell einsetzbar und leicht zu bedienen.**


Der Psion Organiser wiegt etwa soviel wie ein elektrischer Rasierapparat und sieht auf den ersten Blick auch so aus. Das stabile Gehäuse umschließt eine Tastatur im Taschenrechnerstil mit 36 flachen Tasten für die Buchstaben des Alphabets (Großbuchstaben) und einige Funktionen. Mit der Shift-Taste lassen sich die Ziffern 0 bis 9, mathematische Funktionen und Satzzeichen eingeben. Über der Tastatur liegt die Flüssigkristallanzeige für 16 Zeichen. Die Anzeige läßt sich im Kontrast verstellen, so daß sie von jedem Blickwinkel gut abgelesen werden kann. Ihre Zeichen bestehen aus einer Fünf-mal-acht-Matrix.

Auf der Rückseite der Maschine liegen zwei Schlitze, in die sich die „Datapaks“ für die Langzeitspeicherung einsetzen lassen. Der Organiser wird mit einem Datapak von acht KByte geliefert, es gibt aber auch 16K-Paks.

Beim Einschalten des Gerätes mit der ON-Taste am linken oberen Rand der Tastatur werden die Zeit im 24-Stunden-Format und das Datum angezeigt. Die vier Hauptfunktionen des Organisers werden nicht eingegeben, sondern lassen sich per Tastendruck abrufen:

- SAVE: Über Tastatur und Anzeige können Datensätze (maximal 200 Zeichen) eingegeben werden. Mit den zwei Cursortasten wird

**Der als Taschencomputer ausgelegte Psion Organiser besitzt eine außerordentlich flexible Datenbank mit schnellem Speicherzugriff. Mit den eingebauten Taschenrechnerfunktionen und der Möglichkeit, wichtige Informationen schnell wiederzufinden, kann der Organiser viele Aufgaben ausführen, die früher nur auf größeren Computern möglich waren.**

editiert, gelöscht oder mit SAVE in einem der beiden Datapaks gespeichert.

● FIND: Diese Taste ruft die Daten der beiden Datenpaks ab. Eine Suche läßt sich für Zeichenketten bis zu 15 Zeichen durchführen. Der Computer vergleicht dabei die per Tastatur eingegebenen Zeichen mit den gespeicherten Datensätzen. Der FIND-Befehl ist sehr schnell: Aus einer Datenbank von 21 KByte holt er einzelne Datensätze in etwa acht Sekunden heraus. Die Datensätze werden in der Reihenfolge ihres Auftretens angezeigt. Mit den Cursortasten lassen sich die Datensätze in beide Richtungen „rollen".

Psions Datapak „Restaurant Guide To London" bietet auf 16 KByte die Daten von 105 Londoner Restaurants. Jeder Eintrag enthält außer dem Namen Informationen über Adresse, Telefonnummer, die nächste U-Bahnstation, die Art des Essens, Öffnungszeiten, Preise und einen allgemeinen Kommentar. Die Suche nach den Zeichen „ARB" brachte ein Restaurant namens „BARBARELLAS", aber auch eins nahe der „MARBLE ARCH"-U-Bahnstation zum Vorschein. Diese Funktion läßt sich durchaus mit der Geschwindigkeit und Anwenderfreundlichkeit vieler Datenbanken auf Microcomputern vergleichen.

● ERASE: Diese Funktion löscht den abgerufenen Datensatz aus dem Datenpak. Der Organiser stellt dafür keine Taste zur Verfügung, sondern eröffnet diese Möglichkeit nur nach Ausführung eines FIND-Befehls. Nach dem Aufblenden des Datensatzes muß erst mit der Mode-Taste die Funktion ERASE aufgerufen und mit Execute aktiviert werden, um den Datensatz zu löschen. Obwohl der Datensatz nun für die CPU unerreichbar ist, steht sein Platz im Datapak nicht für neue Daten zur Verfügung.

● CALC: Der Organiser läßt sich über diesen Befehl auch als Taschenrechner mit vier Funktionen (Addition, Subtraktion, Multiplikation,

## Systeme im Vergleich

Um den Organiser besser bewerten zu können, vergleichen wir ihn mit einem guten Adreßbuch/Terminkalender.

| | Psion | Filofax |
|---|---|---|
| **Größe** | 142 × 78 × 29,3 mm | 180 × 124 × 30 mm |
| **Gewicht** | 225 g | 200 g |
| **Anzeige** | 16 Zeichen Flüssigkristallanzeige | handgeschriebene/getippte Einträge |
| **Speicher** | 10 900 Zeichen im 8K-Datapak | Bis zu 200 Seiten |
| **Suchmöglichkeit** | Eingabe bis zu 15 Zeichen | per Hand, alphabetischer Index |
| **Mögliche Anwendungen** | 8- oder 16K-Datapak, Listen für Wissenschaft, Ingenieur- und Finanzwesen, Buchhaltung, Restaurants | Terminkalender, Telefon/Adreßverzeichnis, Kassenbuch, Kalkulationssystem, Landkarten, Notenzeilen |
| **Weitere Möglichkeiten** | Gibt Zeit und Datum an, als Taschenrechner einsetzbar. | Wird mit einer gut aussehenden Ledertasche geliefert, in der auch Kreditkarten untergebracht werden können. |


**Die Tastatur des Psion enthält alle Buchstaben in alphabetischer Reihenfolge. Die SHIFT-Taste eröffnet den Zugang zu der Zahlentastatur, den Taschenrechnerfunktionen, Cursorbewegungen und eingebauten Datenbankbefehlen.**

Division) einsetzen. Das Ergebnis wird mit bis zu sieben Zeichen hinter dem Komma angezeigt, wobei für sehr große oder sehr kleine Zahlen die wissenschaftliche Schreibweise zur Verfügung steht. Auch hier gibt es keine CALC-Taste. Die Funktion kann mit der Mode-Taste jederzeit (außer nach einer Suche mit FIND) aufgerufen werden.

## Extrem niedriger Stromverbrauch

Dem einfachen Aufbau und der Leichtigkeit der Bedienung entspricht der niedrige Preis. Den Organiser preiswert auf den Markt zu bringen war nur möglich, weil seine Technologie mit wenig Strom auskommt und daher wenig kostet. Die Maschine kann mit einer Neun-Volt-PP3-Batterie bis zu sechs Monaten laufen, da sie mit CMOS-Chips arbeitet und die Datapaks EPROMS enthalten, die beide nur wenig Strom brauchen.

Das Herz des Organisers ist ein Hitachi-630X-Acht-Bit-Prozessor, der mit 0,93 MHz getaktet ist und von 4 KByte ROM gesteuert wird. Weiterhin gibt es 2 KByte RAM, die für die Dateneingabe, Bildschirminformation und als „Arbeitsbereich" des Taschenrechners nötig sind. Die EPROM-Datapaks auf der Rückseite entsprechen dem RAM-Speicher eines Microcomputers. Am einfachsten ist es, ein EPROM als RAM anzusehen, das sich nur einmal beschreiben läßt: Die gespeicherten Daten lassen sich auf normalem Wege nicht wieder löschen, können aber wie im ROM angesprochen werden. Die Funktion ERASE markiert daher einen Datensatz als gelöscht, stellt aber den Speicherplatz nicht wieder zur Verfügung. Auf diese Weise füllen sich die Datenpaks nach und nach mit Daten, bis kein Platz für neue Datensätze mehr zur Verfügung steht. Nur wenn das EPROM intensivem ultraviolettem Licht ausgesetzt wird, werden die Daten gelöscht und der Chip wieder in einen unbeschriebenen Zustand versetzt. Diese Aufgabe übernimmt der Psion Formatter. Neue Datapak-Chips kosten 50 Mark bis 70 Mark (acht bis 16 KByte). Sie können bis zu 100mal formatiert werden.

Im Gegensatz zu RAMs brauchen EPROMs keine Elektrizität, um einmal gespeicherte Daten „behalten" zu können. Sie arbeiten sehr zuverlässig und kosten auf Byteebene nur etwa ein Fünftel der RAMs. Oft verwenden Computerhersteller in Prototypen EPROMs statt ROMs, da die Chips im Falle von Softwarefehlern umprogrammiert werden können. Fehlerhaft programmierte ROMs dagegen sind wertlos. Sie werden nur eingesetzt, wenn die EPROM-Software fehlerfrei läuft.

Über eine RS232-Schnittstelle ist auch die Kommunikation mit anderen Computern möglich. Das Zusatzgerät wird in einen der Datenpakschlitze gesteckt und kann Daten mit einer Geschwindigkeit bis zu 9600 Baud senden und empfangen. Auch das Übertragungsprotokoll läßt sich über die mitgelieferte Software programmieren, so daß für eine Druckerausgabe beispielsweise die Seitenlänge und Zeilenbreite angegeben werden kann.

Der Organiser läßt sich in vielen Bereichen einsetzen, in denen tragbare und flexible Computer gebraucht werden. Mit ihm können Testergebnisse notiert oder Termine abgerufen werden. Die Maschine eignet sich hervorragend für Informationen, die eine große Zahl von Querverweisen haben.

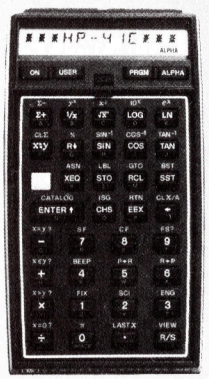
Mit seiner geringen Größe, dem niedrigen Strombedarf, Speicherkapazitäten, die es zuvor nur auf größeren Maschinen gab, und der Flexibilität der eingebauten Datenbank ist der Organiser ein interessantes Gerät für jeden, der einen Microcomputer in der Tasche haben möchte.

## Taschencomputer

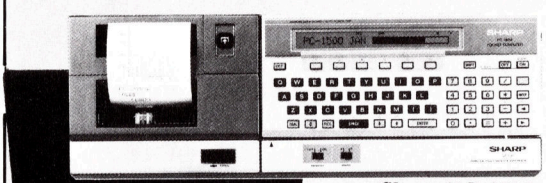
In den letzten Jahren gab es mehrere Versuche, einen Markt für Taschencomputer zu schaffen. Geräte dieser Art wurden jedoch mehr schlecht als recht von der Öffentlichkeit angenommen und erreichten nie die Verbreitung der Taschenrechner.

Einige Taschencomputer sind nur wenig mehr als veredelte Taschenrechner. Sie lassen sich in einer eigenen Sprache programmieren und werden hauptsächlich von Ingenieuren und Wissenschaftlern eingesetzt, die auch außerhalb ihrer Büros komplizierte Berechnungen durchführen müssen. Auf der nächsthöheren Stufe sind Geräte angesiedelt, die sich in BASIC programmieren lassen.

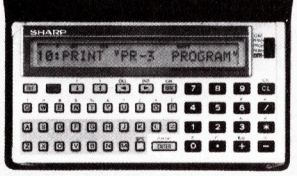
Doch selbst die besten Taschencomputer fanden kaum Verbreitung, da es keine vernünftige Software für das Führen von Terminkalendern oder Adreßverzeichnissen gibt. Weiterhin sind die Speicher der Geräte oft zu klein, um brauchbare Informationsmengen enthalten zu können. Da die winzigen Tastaturen vielen Anwendern zu umständlich sind, greifen die meisten doch lieber zu Papier und Bleistift.



**Hewlett Packard 41C**
Diesen hochentwickelten programmierbaren Taschenrechner gibt es in drei Versionen, HP-41C, HP-41CV und HP-41CX. Der 41CX verfügt über ein halbes K Speicher und die anderen beiden über 2K. Die Tastatur des Rechners ist alphabetisch, aber sie eignet sich nicht für Texteingaben.

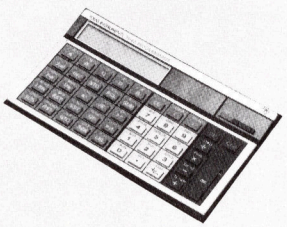


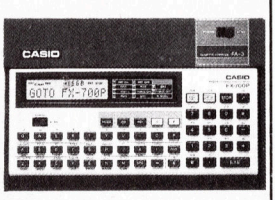
**Sharp PC 1500**




**Sharp PC 1251**
Sharp stellt zwei Taschencomputer her, den PC 1251 und den PC 1500. Das erste Gerät ist der kleinste Taschencomputer des Marktes und das zweite das am weitesten entwickelte. Beide lassen sich in BASIC programmieren, wobei der PC 1500 noch über zusätzliche Befehle verfügt.

**Texas Instruments TI 66**
Dieses Gerät ist im wesentlichen ein programmierbarer Taschenrechner, der hauptsächlich für Wissenschaftler und Ingenieure interessant ist. Der TI 66 kann keinen Text verarbeiten.






**Casio FX700P**
Casio bietet eine ganze Reihe von Taschencomputern, die in BASIC programmiert werden können. Der FX700P hat einen 2K-Speicher und eine winzige alphabetische Tastatur. Das Gerät gibt es auch mit eingebautem Thermodrucker.

# Entscheidungen

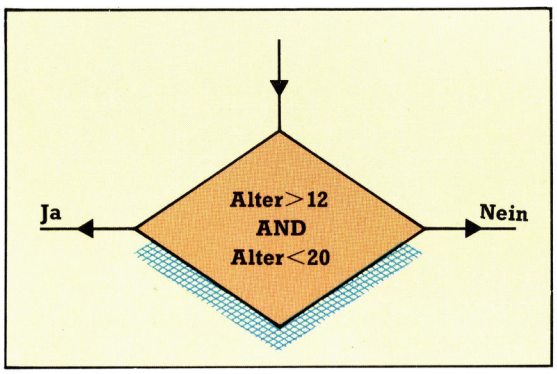
**Wir wollen uns noch etwas gründlicher mit der Technik des Flußdiagramms zur Erleichterung der Programmentwicklung befassen. Heute soll gezeigt werden, wie sich ein komplizierter Entscheidungsprozeß in immer kleinere Elemente zerlegen läßt.**

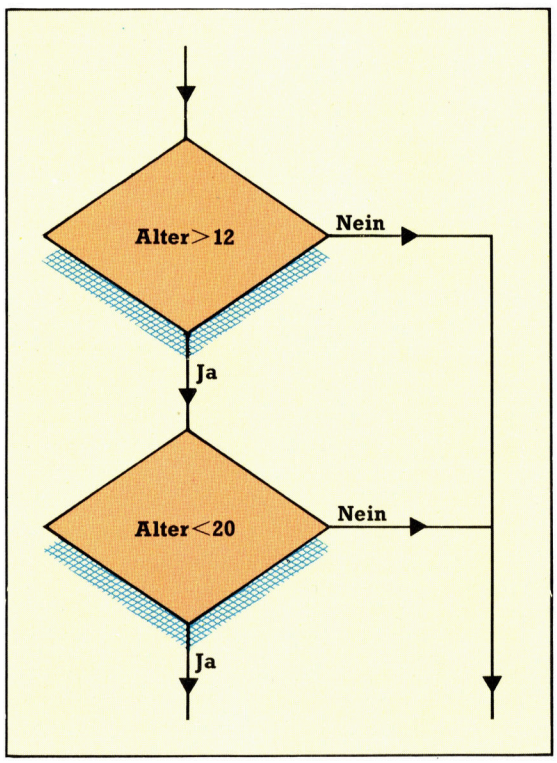
Viele Programmierer verwenden häufig zusammengesetzte Entscheidungen wie:

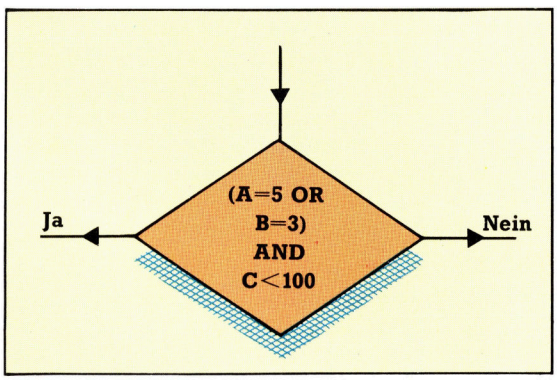
```
IF ALTER>12 AND ALTER<20 THEN
   STATUS="TEENAGER"
```

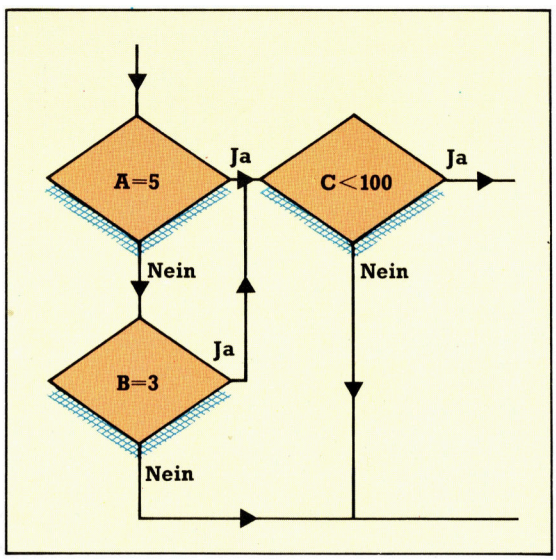
Anweisungen dieser Art sind meist einfacher zu überschauen, wenn die Entscheidungskriterien voneinander gelöst und einzeln betrachtet werden.

Wir haben das BASIC-Beispiel unten in Diagrammform wiedergegeben, um zu zeigen, daß die zusammengesetzte ungünstiger als die einfache Version ist. Das nächste Beispiel besteht aus drei miteinander verbundenen Entscheidungen. Der logische Ablauf im zweiten Diagramm ist dabei wegen der Einzel-Entscheidungen viel übersichtlicher. Es gibt hier eine gewisse Ähnlichkeit mit der Booleschen Logik, die ebenfalls komplizierte Schaltungen aus einfachen logischen Gattern aufbaut.



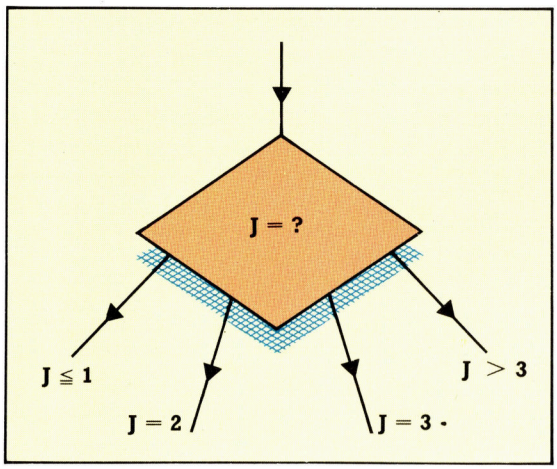



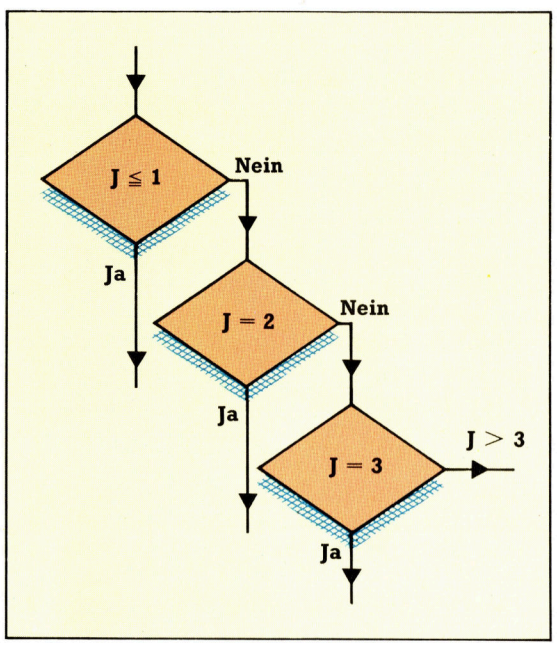




In den Beispielen sind nur Ja Nein-Entscheidungen aufgeführt. Es gibt natürlich auch Fälle, in denen mehr als zwei Möglichkeiten für die Beantwortung einer Frage denkbar sind. So kann etwa bei einem menugesteuerten Programm die Tastatur abgefragt werden, so daß der Programmablauf je nach der gedruckten Taste verzweigt. Dazu gibt es in den meisten Programmiersprachen spezielle Befehle wie CASE .OF in PASCAL oder ON GOTO bzw. ON .GOSUB in BASIC. Die

Regeln sind jedoch überall gleich: Es darf nur jeweils ein Weg aus dem Entscheidungsfeld gewählt werden. Die Beschriftung muß die möglichen Wege eindeutig angeben, und diese müssen alle Möglichkeiten abdecken. Eine kombinierte Entscheidung kann als Verzweigungsfeld mit mehreren Ausgängen gezeichnet werden.




Kombinierte Entscheidungen lassen sich immer durch mehrere Ja/Nein-Entscheidungen ersetzen.

Besonders bei einer Vielzahl kombinierter Entscheidungen kann eine Entscheidungstabelle das Flußdiagramm ersetzen. Eine solche Tabelle besteht aus vier Teilen: Text, der die Bedingungen für die Regelanwendung beschreibt, Text, der angibt, was geschehen soll, ein Raster, das die Bedingungen mit den Regeln verknüpft, und ein zweites Raster, das die Reaktionen mit den Regeln verknüpft. Im Raster „Bedingungen Regeln" stehen Variablenwerte, im unteren Raster „Reaktionen/Regeln" stehen die entsprechenden Reaktionen. Ein Zahlenwert dient als Eingabeparameter für die

| | Regeln | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Bedingungen** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **Feuerknopf gedrückt** | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ |
| **Schwierigkeitsgrad** | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 |
| **Spielstärke** | Anfänger | | Anfänger | | Geübter | Spieler | Geübter | Spieler |
| **Reaktionen** | | | | | | | | |
| **Ausweichmanöver** | | | | | | | ✓ | ✓ |
| **Bombenschutzschilder** | | | ✓ | ✓ | | | ✓ | ✓ |
| **Treibstoff vermindern um** | 1% | 1% | 2% | 2% | 2% | 2% | 4% | 4% |

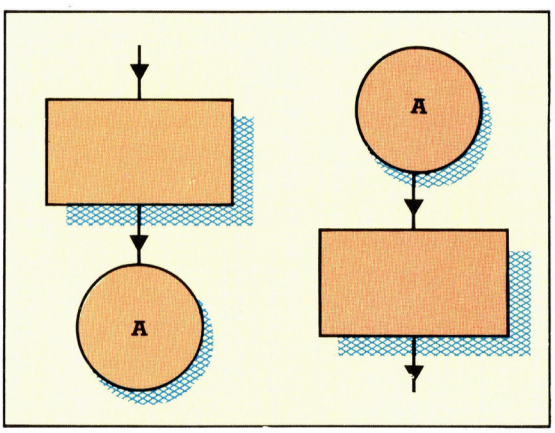
Reaktion. Regel 4 könnte beispielsweise lauten: „Wenn der Feuerknopf gedrückt wird und der Schwierigkeitsgrad 2 von einem Anfänger gespielt wird, dann aktiviere die Bombenschutzschilder und reduziere den Treibstoffvorrat um 2 %".

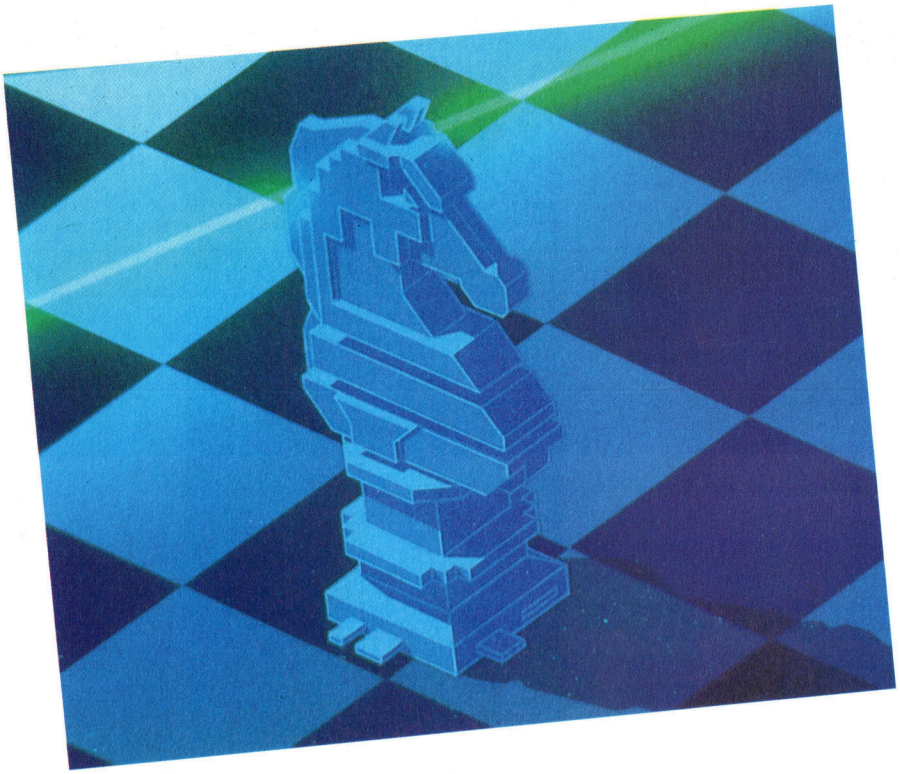
Mit einer solchen Tabelle können auch mehrere Ja/Nein-Entscheidungen zusammengefaßt werden. Sie entspricht damit genau einer Wahrheitstabelle zur Vorhersage der Ausgabe eines Logikgatters.

## Lange Algorithmen zerlegen

Wichtig ist im übrigen, daß sich ein Flußdiagramm nicht über mehr als eine Seite erstrekken sollte – andernfalls verliert man schnell die Übersicht. Zu lange Algorithmen müssen in mehrere Einzelkomponenten zerlegt werden. Dabei kann jeder Algorithmus von einem anderen Algorithmus aufgerufen werden. So läßt sich jede Programmroutine in einen Kasten des Gesamtprogramms hineinschreiben, sogar wenn sie mehrere andere Abläufe aus dem Programm für die eigene Funktion abrufen muß.

Meist geht die Programmentwicklung nicht so glatt, wie man hofft, und das Flußdiagramm muß entsprechend vergrößert werden. In diesem Fall teilt man das Diagramm an einem geeigneten Punkt und verwendet ein Merkzeichen in einem Kreis, mit dem man auf der zweiten Diagrammseite schnell sieht, wo der logische Fluß wieder aufgenommen wird.





**Die meisten intelligenten Schachprogramme haben die Fähigkeit „vorauszublicken". Das heißt, sie können eine Reihe von Zügen mit den nächstmöglichen Zügen vergleichen und so den besten ermitteln. Dies erfolgt durch Konstruktion und Absuchen eines Spielbaums. In den Anfängen der KI-Forschung wurde die Schachspielfähigkeit eines Computers als Maßstab für die Intelligenz der Maschine betrachtet.**

# Voraus-schau

**Auch zu den grundlegenden Konzepten von Schachprogrammen gehören Suchstrategien. Bei unserer Auseinandersetzung mit KI betrachten wir Programmechanismen, die die bestmöglichen Züge für das Schachspiel bestimmen.**

Der Ausdruck „Computerspiel" wird fast immer mit dem Bild von schießenden Außerirdischen im Weltall oder der Suche nach Trollen in unterirdischen Höhlen assoziiert – doch das war nicht immer so. In den Kinderjahren der Computerwissenschaften widmeten sich die Pioniere dieses Bereichs, darunter solche Kapazitäten wie Claude Shannon, John von Neumann und Alan Turing, dem Erstellen von Schachprogrammen.

Schach wurde als das intellektuelle Spiel par excellence betrachtet, und man sah ein erfolgreiches Computerschach-Programm als optimalen Test der „Intelligenz" einer Maschine an. Heute gibt es Computersysteme wie Belle und Cray Blitz, die auf der Ebene internationaler Großmeister spielen.

Die meisten Programme dieser Art basieren auf der Suchbaum-Technik, sind aber dahingehend abgewandelt, daß sie sich auch in die Lage des Gegners versetzen können. Grundlage ist das „Vorausschauen". Das Programm erstellt einen Spielbaum unter Berücksichtigung seiner eigenen Züge, der möglichen Züge seines Gegners, seiner darauffolgenden Züge und so weiter.

Das Diagramm zeigt einen derartigen Baum in einem imaginären Zwei-Personen-Spiel. Wurzel des Baums ist die gegenwärtige Position, bei der MAX am Zuge ist. Die Endknoten stellen jeweils die Position bei Spielende dar. Der Baum wird dazu verwendet, die Entscheidung für einen Zug zu treffen, das ist das sogenannte „Minimaxing", das erstmals 1949 von Claude Shannon durchgeführt wurde. Zunächst werden den Endknoten numerische Werte zugeordnet, beispielsweise Eins für einen Sieg, Null für einen Zug und minus Eins für einen Verlust. Diese Werte werden kombiniert, während der Baum sich verbreitert, davon ausgehend, daß der Spieler (MAX) immer den größeren, sein Gegenspieler (MIN) immer den kleineren Wert erhält, um so Werte für weitere Knoten zu erhalten.

In diesem Beispiel beträgt der Wert der Wurzel 0, womit angezeigt wird, daß ein Zug erfolgt (vorausgesetzt, daß keine der beiden Seiten einen Fehler macht). Die Regeln, nach denen Verzweigungen und Knotenbewertungen erfolgen, werden durch die Regeln des jeweiligen Spiels bestimmt. Nur bei einfachen Spielen ist es möglich, den Spielbaum bis zum Ende durchrechnen zu lassen. Schach dagegen hat einen „Verzweigungsfaktor" von 32, womit die durchschnittlich 32 erlaubten Züge aus jeder Position gemeint sind. In eine sogenannte Halbzugtiefe von „Vier" (zwei Züge nach jeder Seite) vorauszublicken, würde die Betrachtung von über einer Million Knoten bedeuten. Diese „kombinatorische Explosion" verdeutlicht, daß Schachspiel-Programme keinesfalls ganz bis zum Spielende „voraussehen" können.

Deshalb berechnen die Programme nur im Rahmen der gesteckten Grenzen zukünftige Zugmöglichkeiten und bewerten die dort gefundenen Positionen. Das erfordert eine Methode, die Knoten zu bewerten (vorteilhaft oder unvorteilhaft), ohne das Endergebnis zu kennen. Dieser Vorgang wird als „statistische Evaluationsfunktion" bezeichnet und ist zwangsläufig ungenau, da das Endergebnis nur geschätzt werden kann.

Beim Ziehen oder Schlagen im Schachspiel läßt sich folgende einfache Evaluationsfunktion anwenden:

K König-Vorteil
F Figuren-Vorteil
B Beweglichkeit
Z Zentrums-Kontrolle/Beherrschung

Diese Attribute können durch Studium des Spielbretts auf den Computer gebracht wer-

den. Zum Beispiel stellt VK bei K = VK – AK die Anzahl der verteidigenden Könige dar und AK die Menge der angreifenden Könige.

Angenommen, wir bewerten einen König (K) mit dem Dreifachen dessen, was einer normalen Figur (F) an Wert zugesprochen wird, lassen eine Extra-Figur wertmäßig zweieinhalb zusätzlichen Zügen (B) entsprechen und bewerten einen schlechten Zug so, als würden wir ein weiteres Quadrat im Zentrum kontrollieren, so ergäbe sich folgende Evaluationsfunktion der Position:

$$V=15K + 5F + 2B + Z$$

Die Idee, Bestandteilen von Spielzügen Zahlenwerte zuzuordnen und diese in einer gewichteten Summe zu summieren, um die jeweilige Position zu bewerten, hat sich als erfolgreich erwiesen. Die Evaluationsfunktion spielt eine ähnliche Rolle wie die an anderer Stelle erläuterte heuristische Entfernungsmessung bei Problemlösungen.

## Begrenzter Horizont

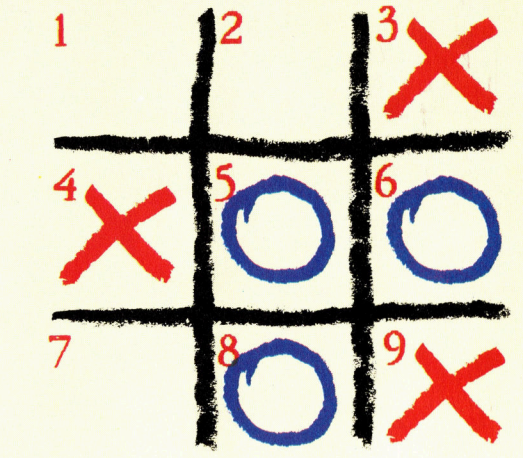
Ein Programm, das nur in eine bestimmte Tiefe vorausschauen kann, gerät bald in Schwierigkeiten. Grund dafür ist, daß einige Spielpositionen „ruhig" sind, andere dagegen sehr „labil". Beim Schach ist der Spielzustand nach einem Schlagen oder Bauernzug sehr labil. Schon beim nächsten Zug kann eine völlige Umkehrung erfolgen. Wenn eine solche Situation gerade einen Schritt außerhalb des „Horizonts" des Programms liegt, dann wird die Voraussage des Programms irreführend sein.

Um diese Auswirkung auszuschließen, haben die meisten Programme keine festgelegte Tiefe bei der Vorausschau. Statt dessen bewerten sie „ruhend", ob das Evaluieren einer Position lohnt. Kann dies nicht eindeutig sichergestellt werden, wird die Suche an anderer Stelle fortgesetzt. Beim Schach wie bei Dame brauchen diese Berechnungen vergleichsweise viel Zeit.

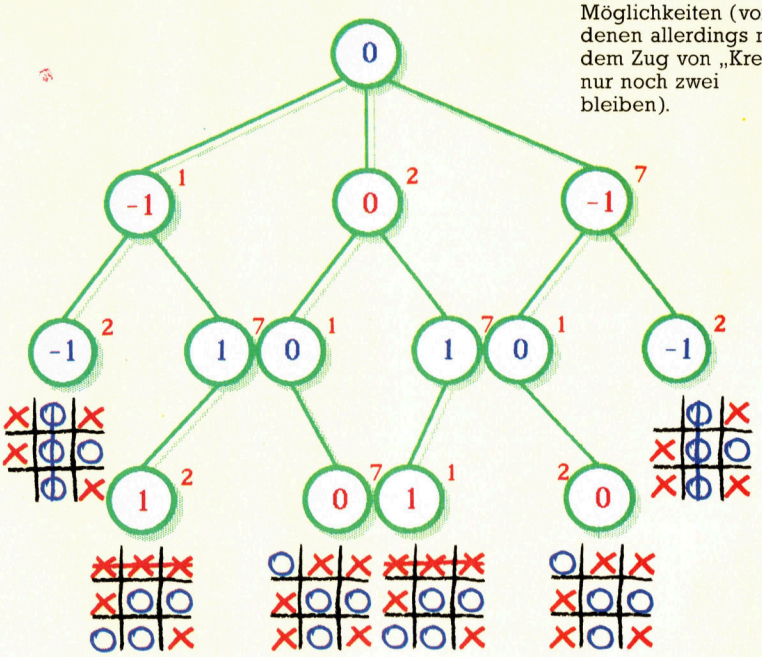
Der Alpha-Beta-Algorithmus tauchte erstmals 1967 in Greenblatts MacHack-Programm auf. Es handelt sich dabei um eine Verbesserung des „Minimaxens" und führt mit weit weniger Aufwand zum selben Ergebnis. Das Diagramm auf der folgenden Seite unten zeigt den Teil eines Spielbaums zwischen zwei mit MINI und MAX benannten Spielern.

Die Buchstaben neben den einzelnen Knoten (A bis L) zeigen, in welcher Reihenfolge der Baum untersucht wird, wobei die Halbzugtiefe zuerst berücksichtigt wird. Die Zahlen sind Evaluationswerte. Die Einzelstriche markieren das sogenannte „Alpha-Abschneiden", Doppelstriche kennzeichnen das „Beta-Abschneiden". Damit werden „Äste" entfernt, die auf das Endergebnis eindeutig keinen Einfluß haben.

**Spielstand**




Das Diagramm zeigt den gegenwärtigen Spielstand eines Spiels mit Kreuzen und Nullen nach sechs Zügen (Kreuz hat den nächsten Zug). Ein einfacher Spielbaum kann konstruiert werden, um die letzten drei Züge des Spiels zu betrachten. Der erste Halbzug hat drei Knoten, entsprechend den drei möglichen Zügen (in die Quadrate 1,2 oder 7), die „Kreuz" machen kann. Wenn „Null" am Zug ist, bleiben nur noch zwei Quadrate, zwischen denen er wählen kann – mit drei Möglichkeiten (von denen allerdings nach dem Zug von „Kreuz" nur noch zwei bleiben).



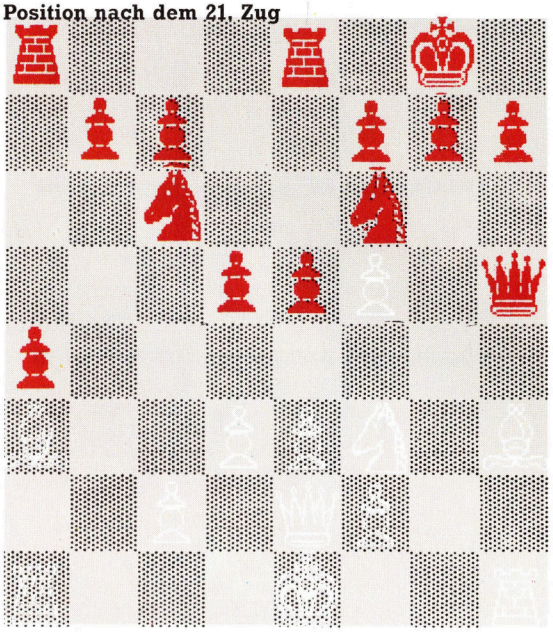

Nach Konstruktion eines Spielbaums kann jedem „terminalen" Knoten ein Wert zugewiesen werden: 1, wenn Kreuz gewinnt, 0 für einen Zug und –1, wenn Null gewinnt. Wir können dann die im Baum entsprechend zugeordneten Werte in Richtung auf die anderen Knoten verfolgen. Betrachten wir zunächst den Knoten rechts außen im ersten Halbzug, gelangen wir zu dem Wert –1. Da Null am Zug ist, wird der kleinere Wert ausgesucht. Durch Rückverfolgen wählt Kreuz den Maximalwert (0) und setzt auf Feld 2.

Ein Alpha-Abschneiden erfolgt an Knoten E, für den keine Evaluation erforderlich ist. Wir wissen, daß Knoten C schon einen Wert von 15 hat, wenn Knoten E erst erreicht wird. Doch der Gegner kann uns an Knoten D bis hinunter auf 10 drücken. Es ergibt sich keine Notwendigkeit festzustellen, ob wir noch weiter gedrückt werden können; denn dieser Weg ist offenbar weniger lohnend als der durch Knoten C. Deshalb können auch die anderen Abzweigungen von Knoten F unberücksichtigt bleiben.

Umgekehrt gilt diese Begründung auch bei der Betrachtung von Knoten I. Bis wir dorthin gelangt sind, hat G bereits einen Wert von 20. Knoten H – Wert 25 – sieht vielversprechender aus, doch MINI – und nicht MAX – wählt

**Diese Schachpartie ist Teil einer ganzen Serie, in der das stärkste Schachprogramm der Welt, Cray Blitz von David Levy (Intelligent Software), 4:0 geschlagen wurde. Grundlage war eine 20 000-Mark-Wette zwischen Levy und den Programmierern von Blitz. Inzwischen besitzt das Programm zwar die Stärke eines Großmeisters, das Spiel zeigt aber, daß noch viel zu tun ist, bis Computer ernstzunehmende Schachgegner für die besten menschlichen Spieler sind.**

**Position nach dem 21. Zug**



zwischen Knoten G und J und wird ganz klar G bevorzugen. Es gibt also keinen Grund, I weiter zu untersuchen, da MAX dort nicht hin gelangen wird.

Wir können diese Überlegungen mit einem Familien-, einem Stammbaum deutlicher machen. MAX ist ein männlicher Chauvinist, der glaubt, daß zum Beispiel Knoten C der Onkel der Knoten D und E ist, die beide Söhne desselben Vaters sind. MINI andererseits ist eine Feministin, und sie ist davon überzeugt, daß G die Tante der Schwestern H und I ist, deren Mutter wiederum J ist. Wenn man sich nicht durch die Geschlechtswandlungen der Knoten in alternativen Halbzugtiefen irritieren läßt, erläutert dieses Beispiel die Alpha-Beta-Regel sehr deutlich:

- Sobald MAX einen Sohn findet, der „schlimmer" als alle seine Onkel ist, ignoriert er die übrigen Brüder dieses Sohnes.
- Sobald MINI eine Tochter findet, die „besser" als alle ihre Tanten ist, ignoriert sie die übrigen Schwestern dieser Tochter.

Im besten Fall überprüft der Alpha-Beta-Algorithmus nur zweimal die Quadratwurzel der Menge der Spitzenknoten des Spielbaums. Im schlechtesten Falle überprüft er so viele wie möglich, und dies dauert entsprechend länger. Um den ersten der beiden Fälle zu vermeiden, ist es wichtig, die Brüder und Schwestern auf jeder Ebene in vernünftiger Reihenfolge zu generieren. Auf den Maximierungsebenen sollte nach dem Prinzip „Bester zuerst" und auf den Minimierungs-Ebenen nach dem Prinzip „Schlechtester zuerst" generiert werden (wobei „Bester zuerst" für den Gegner gilt).

Um die wichtigen Konzepte der Suchbaum-Methode zu illustrieren, stellen wir hier ein Spiel vor, das Suchtechniken enthält. Einzelheiten der Brettdarstellung, der Zug-Generierung und der statistischen Evaluation werden aus Gründen der Einfachheit weggelassen. Das Prinzip des Alpha-Beta-Abschneidens wird jedoch an diesem Beispiel deutlich.

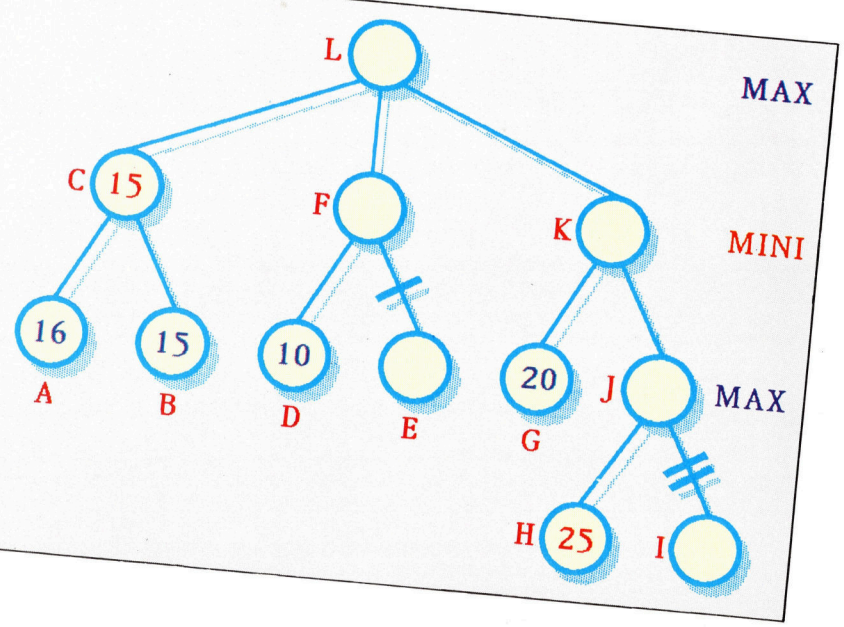
## Das Alpha-Beta-Minimaxen

Es gibt weder ein Spielbrett noch Figuren: Der Spielstatus ist in einer einzigen Zahl dargestellt, die V% genannt wird. Ziel des Computers ist es, den Wert von V% auf 255 zu erhöhen, Sie dagegen müssen ihn unter — 255 bringen.

Bei jedem Zug kann der Spieler zwischen einer von vier Funktionen A,B,C,D wählen, die in den Zeilen 1030 bis 1060 enthalten sind. Man kann sie verändern, wenn man der Abwechslung halber unterschiedliche Versionen des Spiels erhalten und dem Computer die

### Beschneidung

Das Alpha-Beta-Abschneiden verbessert die Minimax-Methode durch Entfernen überflüssiger Zweige vom Spielbaum. Alpha-Schnitte (hier durch einzelnen Balken gekennzeichnet) werden vollzogen, wenn eine Minimierungsstrecke bei der Bewegung von MINI gefunden wird und sich somit weiteres Suchen erübrigt. Umgekehrt werden Beta-Schnitte (doppelter Balken) durchgeführt, wenn bei der MAX-Bewegung eine Maximierungsstrecke gefunden wurde.



Arbeit noch ein wenig erschweren will.

Das Spiel ist sehr einfach, zeigt aber die Suchstrategie klar auf. Zudem verfügt der Computer über einen „eingebauten Vorteil". Das Alpha-Beta-Minimaxen hängt in diesem Fall stark von der Verwendung recursiver Funktionen mit Parametern und lokalen Variablen ab. Die Parameter sind:

VV% Derzeitiger Spielstand
A% Alpha-Bester in dieser Ebene
B% Beta-Schlechtester in dieser Ebene
D% Tiefenindikator

Dieses Programm wurde für den Acorn B geschrieben.

Die Versionen für den C64 und den ZX Spectrum folgen in der nächsten Ausgabe.

## Das Zahlenspiel

```
 10 REM ******************************
 11 REM ** Listing 3.1 :            **
 12 REM ** THE NUMBERS GAME         **
 13 REM ******************************
 50 MODE 7
100 REM -- Game to demonstrate search:
120 GOSUB 1000  : REM initialization
130 GOSUB 1600  : REM instructions
150 REPEAT
160   GOSUB 2000  : REM prepare new game
170   INPUT "Who goes 1st (1=You, 2=Me) ", H1%
180   IF H1%<1 OR H1%>2 THEN GOTO 170
200   REM -- main loop:
210   REPEAT
220     IF H1%=1 THEN GOSUB 3000
230     REM -- person's turn.
240     GOSUB 3500  : REM board display
250     H1%=1 : REM always 1 after 1st cycle
260     GOSUB 4000  : REM test for win
270     IF EG%=0 THEN GOSUB 5000
280     REM -- computer's turn.
290     GOSUB 3500  : REM show game status
300     GOSUB 4000  : REM test for endgame
310     UNTIL EG%<>0 OR M%>33
320   REM -- finale:
330   GOSUB 6000 : REM congratulations
340   PRINT "Another Game (N=No) ";
350   Y$=GET$
360   UNTIL Y$="N" OR Y$="n"
365 PRINT
370 PRINT "So Long, and thanks for the game!"
400 END
444 :
500 DEF FNmaximove(VV%,A%,B%,D%)
505 REM ---------- my turn:
510 LOCAL P%,E%,KEEP%
515 IF D%>=MD% OR ABS(VV%)>HI% THEN =VV% :
    REM static value
520 REM else go deeper:
525 P%=0
530 REPEAT  P%=P%+1
535   H%=P%: V%=VV%: GOSUB 5500: REM make move
536   IF D%=1 THEN PRINT CHR$(H%+64);" = ";
540   E%=FNminimove(V%,A%,B%,D%+1)
545   IF E%>A% THEN A%=E%: KEEP%=P%
548   IF D%=1 THEN PRINT E%;"; ";
550   UNTIL P%>3 OR A%>=B%
555 IF A%>BV% AND D%=1 THEN BV%=A%: HH%=KEEP%
556 REM keep best so far.
560 =A%
565 REM return with max A%
570 :
700 DEF FNminimove(VV%,A%,B%,D%)
710 REM ---------- other's turn:
720 LOCAL E%,P%
730 IF D%>=MD% OR ABS(VV%)>HI% THEN =VV%
740 P%=0
750 REPEAT P%=P%+1: H%=P%
755   V%=VV%: GOSUB 5500: REM make move
760   E%=FNmaximove(V%,A%,B%,D%+1)
770   IF E%<B% THEN B%=E%
780   UNTIL P%>3 OR B%<=A%
790 =B%
796 REM returns with lowest value B%
999 :
1000 REM -- initializing routine:
1001 BL$=""
1002 @%=4
1020 REM -- the 4 functions:
1030 DEF FNA(X%)=2*X% - 7
1040 DEF FNB(X%)=X% DIV 2 + 1
1050 DEF FNC(X%)=-4*X% + 17
1060 DEF FND(X%)=3*X% - 4
1070 LO%=-255: HI%=255
1150 RETURN
1160
1600 REM -- instruction routine:
1610 CLS: PRINT
1620 PRINT "Welcome To The Numbers Game!"
1630 PRINT "If you don't know the rules,"
1635 PRINT "READ THE ARTICLE!"'
1636 PRINT "N.B. I maximize : you minimize."
1640 PRINT "To see the effect of a move type:"
1645 PRINT "A, B, C, or D. To make it, type X."
1650 PRINT : PRINT "Good Luck!";CHR$(7)
1660 RETURN
1670
2000 REM -- Preparation routine:
2010 M%=0 : REM moves
2020 V% = RND(15)-8 : REM initial state.
2050 EG%=0
2060 PRINT'"Initial state = ";V%
2100 RETURN
2110
3000 REM -- Person's Move
3010 M%=M%+1
3020 PRINT
3030 REPEAT
3040   PRINT "Your move is ? ";
3050   H$=GET$: PRINT H$;
3060   IF H$="A" THEN PRINT FNA(V%): H%=1
3070   IF H$="B" THEN PRINT FNB(V%): H%=2
3080   IF H$="C" THEN PRINT FNC(V%): H%=3
3090   IF H$="D" THEN PRINT FND(V%): H%=4
3100   UNTIL H$="X"
3120 GOSUB 5500 : REM do choice H%
3150 RETURN
3160 :
3500 REM -- board display routine:
3520 CLS : PRINT
3522 PRINT "     Move ";M%;" --> ";
3523 IF M%<1 THEN RETURN
3525 PRINT CHR$(64+H%);
3530 PRINT " = ";V%
3535 RETURN
3700
4000 REM -- Win-test Routine (on M$):
4001 IF M%<1 THEN RETURN
4010 EG%=0
4020 IF V%<LO% THEN EG%=-1
4030 IF V%>HI% THEN EG%=1
4040 RETURN
4050 :
5000 REM -- Computer's Move Routine:
5005 W%=V% : REM save current status.
5010 M%=M%+1
5015 MD%=6 : REM max depth
5020 IF M%<4 THEN MD%=4
5030 IF M%>8 THEN MD%=8
5040 GOSUB 5200 : REM --> H%
5045 V%=W% : REM restore status.
5050 GOSUB 5500 : REM do it.
5070 RETURN
5080 :
5200 REM -- Move Selection:
5210 BV%=LO% : D%=0
5220 BV%=FNmaximove(V%,LO%,HI%,1)
5230 H%=HH%
5240 PRINT'"Press any key to go on: ";
5244 C%=GET
5250 RETURN
5270 :
5500 REM -- Make-move Routine (H% : V%):
5505 ON H% GOTO 5510,5520,5530,5540
5510 V%=FNA(V%): RETURN
5520 V%=FNB(V%): RETURN
5530 V%=FNC(V%): RETURN
5540 V%=FND(V%): RETURN
5550 :
6000 REM -- Congratulations Routine:
6010 PRINT "GAME OVER!"
6020 IF EG%>0 THEN PRINT "I won it!!"
6030 IF EG%<0 THEN PRINT "Well done!"
6040 IF EG%=0 THEN PRINT "Game drawn"
6050 RETURN
6600 :
```

# Listenreich

**Innerhalb der kleinen Serie über vertikale Software stellen wir Ihnen diesmal „BrainStorm“ von Caxton Software ausführlich vor.**

BrainStorm wird als „Gedankenprozessor“ bezeichnet. Das Programm soll Hilfestellung beim Ordnen von Ideen geben.

Bei der Planung eines vielschichtigen Projektes werden oft Listen mit allen notwendigen Abläufen angelegt. Aus den Haupt-Listen entstehen Sub-Listen, die die Abläufe einer Haupt-Liste im einzelnen beschreiben. Den Sub-Listen werden weitere Listen untergeordnet, bis der Planer vor lauter Papier den Überblick verliert. Jede Veränderung – beispielsweise die Verschiebung eines Hauptziels auf eine Sub-Liste – kann derart viel Radieren und Ändern bedeuten, daß sich das System nach einiger Zeit nicht mehr verwalten läßt. Mit BrainStorm lassen sich diese Veränderungen und Entwicklungen jedoch ebenso einfach wie das Editieren eines Schriftstücks mit einem Textsystem erledigen.

## Elf Menüoptionen

Die Elemente der Listen oder Sub-Listen können jederzeit zum „Anführer“ einer neuen Subliste „befördert“ werden. Auch lassen sich gleichartige Aktivitäten in verschiedenen Listen durch den gleichen Namen (einen sogenannten „Namensvetter“) miteinander verbinden. In diesem Fall erscheinen alle Elemente, die in die Sub-Liste unter einem Namen eingefügt werden, auch in den anderen Listen mit derselben Überschrift.

Jeder, der mit den Editierbefehlen von Textsystemen wie WordStar vertraut ist, wird sich leicht an die Cursorsteuerung von BrainStorm gewöhnen. So bewegt die gleichzeitige Betätigung der Tasten Control (CTRL) und (S) den Cursor nach links, während CTRL-D, CTRL-E und CTRL-X den Cursor entsprechend nach rechts, auf und ab bewegen. Diese vier Tasten bilden auf der Tastatur ein zusammenhängendes logisches Feld. Die Cursorbewegung kann jedoch auch umdefiniert werden. Etwas gewöhnungsbedürftig ist, daß für Links- und Rechtsbewegungen innerhalb einer Zeile erst der Änderungsmodus (CTRL-A) eingeschaltet werden muß.

Das Eröffnungsmenü von BrainStorm bietet elf Bearbeitungsmöglichkeiten, die über ihren Anfangsbuchstaben aufgerufen werden: **U**se (Eingabe), **L**oad (Laden), **P**rint (Drucken), **ID** Drive (Laufwerk ändern), **C**lear (das aktuelle Modell im RAM löschen), **S**ave (Speichern), **W**rite (Modell auf Diskette speichern), **D**irectory (Inhaltsverzeichnis), **X**it (Programmende), **M**erge (zwei Modelle miteinander verbinden) und **K**ill (Modell auf Diskette löschen).

**U** ruft den Eingabemodus auf, unter dem sich die Ideen zunächst mehr oder weniger geordnet in das Modell eintragen lassen. Ein Modell könnte folgendermaßen aussehen:

Handbuch lesen
Liste eingeben
Sub-Liste eingeben
Liste editieren

Die ?-Taste ruft die Steuerbefehle auf. Jedes Element der Liste kann zum „Anführer“ einer Sub-Liste „befördert“ werden, indem der Cursor darauf eingestellt und CTRL-R gedrückt wird. Ebenso lassen sich auch die Elemente einer Sub-Liste als Kopf einer neuen Sub-Liste kennzeichnen und so weiter – bis der Speicher voll ist. CTRL-C schaltet wieder auf die vorige Liste zurück.

## Programmablauf

Innerhalb der Listen können die einzelnen Elemente frei bewegt werden. Über die Kennzeichnung mit @ und CTRL-G (für Get) oder CTRL-P (für Put) lassen sich die Elemente aber auch in andere Listen und andere Ebenen übertragen. Da das @ nach jedem CTRL-G automatisch auf das nächste Listenelement springt, kann mit wiederholtem Drücken von CTRL-G eine ganze Reihe von Elementen schnell in andere Listen übernommen werden.

Neue Elemente werden in bestehende Listen eingefügt, indem der Cursor an den Anfang einer Zeile gestellt und der Text eingegeben wird. Der Abschluß mit RETURN bewegt dann den Rest der Liste eine Zeile nach unten. CTRL-A schaltet den Änderungsmodus ein.

Wenn in unterschiedlichen Listen identische Namen verwandt werden – beispielsweise ein Datum –, werden diese Elemente zu „Namensvettern", die automatisch aufeinander verweisen. Soll ein geplantes Vorhaben an einem bestimmten Tag stattfinden, genügt es, für alle Ereignisse dieses Tages eine Sub-Liste zu erstellen. Alle anderen Listen mit diesem Datum werden damit automatisch aktualisiert.

## Gedruckte Listen

Es lassen sich beliebig viele „Namensvettern" anlegen. Mit CTRL-S (nächstes Element) und CTRL-D (voriges Element) können sie nacheinander durchgeblättert werden. CTRL-S nach dem letzten Element ruft wieder das erste Element auf.

Die Listen lassen sich selbstverständlich auch drucken, wobei unterschiedliche Ebenen durch Einzüge angezeigt werden:

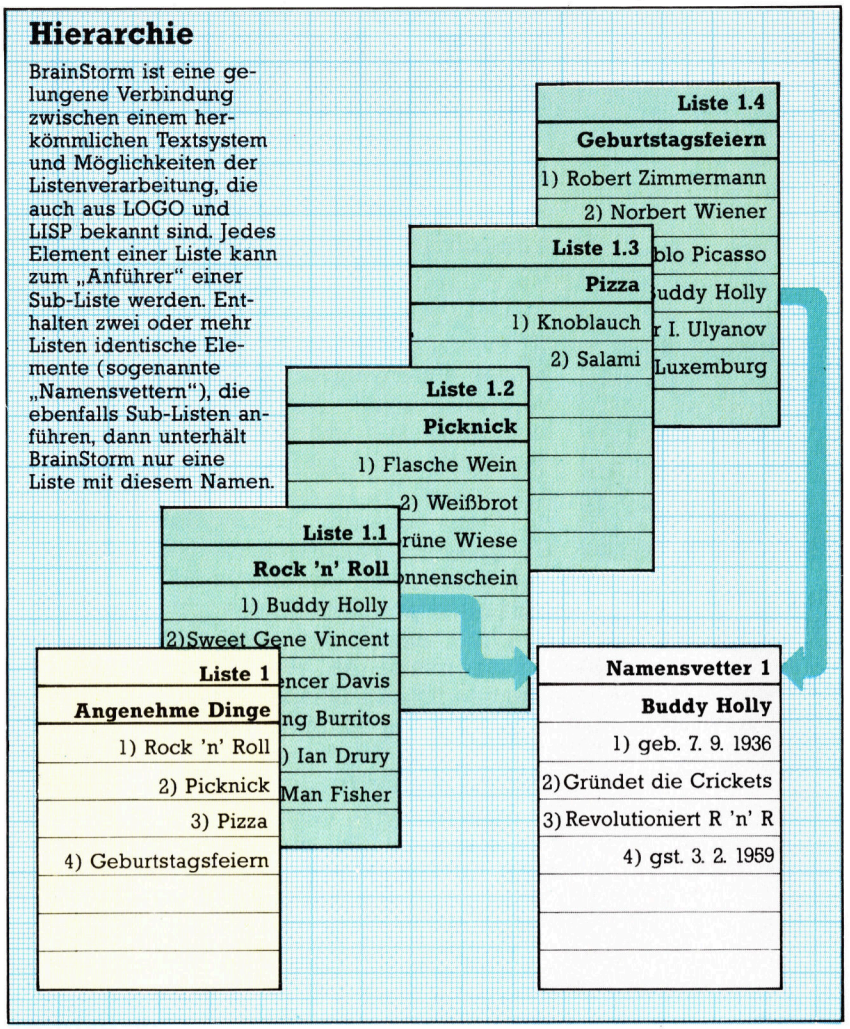
Handbuch lesen
  Handbuch aus der Hülle nehmen
    Inhaltsverzeichnis aufschlagen
      Gewünschte Seite finden
    Seite lesen
  Handbuch wieder in die Hülle stecken
Liste eingeben
  Mit CTRL-R ein Element als Kopf einer Sub-Liste kennzeichnen
    Sub-Liste eingeben
  Mit CTRL-C zurück auf die vorige Liste
Liste editieren
  Mit CTRL-A Änderungsmodus einschalten

Die Tiefe der Einzüge kann der Anwender bestimmen.

Die Modelle lassen sich auch außerhalb von BrainStorm bearbeiten. Nach der Speicherung mit „Write" können die Dateien von Textsystemen wie beispielsweise WordStar gelesen, geändert, gedruckt und wieder gespeichert werden. Beim Schreiben eines Buches ließe sich mit BrainStorm beispielsweise der Ablauf skizzieren und mit WordStar dann aus der Datei „BrainStorm.Doc" übernehmen.

Schon die ersten groben Ideen sollten mit BrainStorm notiert werden. Diese Ideenelemente werden zu Kapitelüberschriften, unter denen wiederum der Inhalt entsteht. Wird ein Element für ein Kapitel zu groß, läßt es sich zu einem eigenen Kapitel „befördern". Umgekehrt können Kapitel, die an Wichtigkeit verlieren, in andere integriert werden.

Sobald sich der grobe Umriß des Buches abzeichnet, können die Stichworte von einem Textsystem übernommen und in das Manuskript umgesetzt werden. Dadurch erspart man sich die etwas umständliche Textaufbereitung in BrainStorm. Douglas Adams, Autor von „The Hitch Hiker's Guide to the Galaxy", entwickelte mit BrainStorm ein Abenteuerspiel auf diese Weise.




Der Lieferumfang des Programms enthält die Datei „SAMPLE.BRN" mit dem Gerüst eines Terminkalenders, einer Namens- und Adreßliste, einer Aufgabenliste (mit Sub-Listen für „wichtig", „sehr wichtig" und „nicht vergessen") und eines Notizbuches. Die Modelle lassen sich mit der Option „Merge" in eigene Modelle einfügen.

Der Umgang mit BrainStorm ist leicht zu erlernen. Das Handbuch ist systematisch aufgebaut und wurde ebenfalls mit BrainStorm erstellt. Da jedoch die Befehle jederzeit auf dem Bildschirm dargestellt werden können, wird es selten gebraucht.

Doch nicht alle Befehle sind leicht zu behalten. Deshalb kann der Anwender mit dem Programm „INSTALLB" andere Befehlsnamen angeben und auch die Menüs umstellen. INSTALLB ist einfach aufgebaut und menügesteuert.

**BrainStorm:** Für zirka 25 verschiedene CP/M-, MS-DOS- und PC-DOS-Maschinen erhältlich, darunter IBM, Sirius und Apricot.
**Herausgeber:** Caxton Software Ltd., 10–14 Bedford Street, London WC2E 9HE
**Autoren:** David Tebutt und Mike Liardet
**Format:** Diskette

# Vielfältig

**Audiogenic hat sich als Vertriebsfirma für Commodore-Programme einen guten Ruf erworben. Früher befaßte sich die Firma ausschließlich mit der Vervielfältigung von Tonbändern.**


**Firmensitz ist Sutton Park, etwas außerhalb von Reading. Als 1984 der Lagerplatz im alten Gebäude in der Innenstadt von Reading nicht mehr reichte, zog die Firma aufs Land.**

Firmengründer Martin Maynard beschreibt das Unternehmen als „Produktions- und Vertriebsgeschäft". Maynard war früher im Musikbereich tätig und gründete Audiogenic in den frühen siebziger Jahren als Tonstudio und Kopierwerk für Bandaufnahmen. 1978 erhielt die Firma dann den Auftrag, Datenbänder für Computer zu duplizieren. Nach der notwendigen technischen Umstellung auf die Massenproduktion von Datenbändern wurde ein Vertrag mit Commodore unterzeichnet – Audiogenic übernahm die Vervielfältigung der Programme für den Heimcomputer PET.

In der Folge führte die Firma den gesamten Vertrieb des Commodore-Programmangebots aus und begann mit dem Verkauf von Büchern, Zeitschriften und anderem Commodore-Zubehör. Nachdem 1981 der VC 20 auf den Markt kam, erwarb Maynard von amerikanischen Softwarehäusern Lizenzen für den Verkauf von VC-20-Programmen. Noch heute besteht das Angebot zu 80 Prozent aus in Lizenz hergestellter Software, auf die auch der größte Teil des Umsatzes entfällt.

Es zeigte sich, daß der reine Vertrieb von Software eher zum Erfolg führt als die Eigenproduktion. Maynard schätzt jedenfalls, daß Audiogenic bisher mehr als eine Million Cassetten fertiggestellt hat. „Unsere Stärke ist die große Auswahl. Dadurch kennen wir auch den Markt", behauptet Maynard. „Die heutige Entwicklung haben wir lange vorausgeahnt. Von den im letzten Jahr neu erschienenen Programmen sind 20–30 Prozent noch unverkauft – die Programmschreiber wissen einfach nicht mehr, was ihre Kunden wollen."

Das Ziel, erprobte und bekannte Programme zu empfehlen, bildet einen scharfen Gegensatz zur Hoppla-jetzt-komm-ich-Mentalität in anderen Softwarehäusern. Produktmanager David Smithson sagt dazu: „Wir kaufen uns keinen teuren Sportwagen. Manche Firmen laufen durch schlechte Geschäftspolitik geradezu mit offenen Augen in die Pleite."

## Auf Bewährtes vertrauen

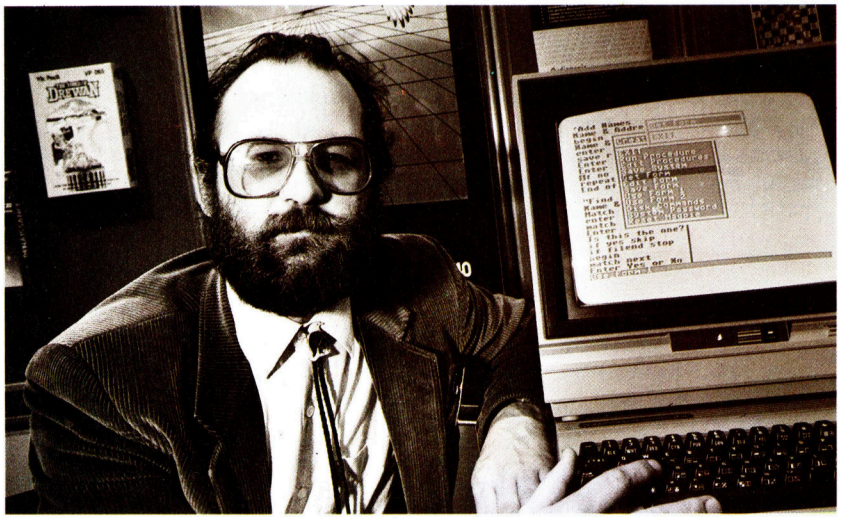
Audiogenic hat 25 Angestellte. Chefprogrammierer Dave Middleton ist als freier Mitarbeiter beschäftigt. Bei Audiogenic neigt man eher dazu, die Eigenprodukte weiter auszubauen, als sich auf den profitversprechenden Markt der Computerspiele zu wagen. Maynard erklärt das so: „Mit Computern wird auch in Zukunft gespielt werden, jedoch geht die Phase des passiven Spielens zu Ende. Computersoftware wird sich zu etwas viel Nützlicherem entwickeln. Wenn Sie von einem Programmpaket zwei- oder dreihundert Stück pro Monat verkaufen, könnte man denken, ‚das verkauft sich nicht gut'. Wenn es aber ein gutes Programm ist, dann werden Sie es auch Jahre später noch gut verkaufen können."

Natürlich lehnt auch Audiogenic den Markt der Computerspiele nicht völlig ab. „Motor Mania" etwa war einer der größten Verkaufserfolge der Firma. Jüngst erschien für den C 64 das Programm „Alice im Videoland", das unter Lizenz entwickelt wurde.

Im März 1984, kurz nach dem Umzug in größere Gebäude, investierte Audiogenic in neue Tonband-Kopiermaschinen. Heute werden die Programme wiederholt auf einem Bandstreifen aufgenommen, der erst nach Aufnahme der Daten geschnitten und in Cassettenform konfektioniert wird. Das geht schneller und preiswerter als das frühere Verfahren, für das große Mengen von C10- und C20-Cassetten gelagert werden mußten, die einzeln mit den Programmen bespielt wurden.

Auch in Zukunft wird Audiogenic in England weiterhin den Vertrieb für fremde Hersteller übernehmen. Aber man will sich auch weiter diversifizieren – ein Schritt in Richtung Hardware ist etwa das Marketing eines Grafiktabletts. Außerdem will man in nächster Zeit auch Software für die neuen Commodore-Geräte und MSX-Rechner aufnehmen.

**Manager, Direktor und Gründer des Unternehmens ist Martin Maynard. Er gründete Audiogenic in den frühen siebziger Jahren.**



# Torch Disk Pack

**Mit dem „Torch Disk Pack“ läßt sich die magere Speicherkapazität des Acorn B um 64 KByte erhöhen. Ferner enthält diese Einheit zwei Diskettenlaufwerke und einen Z80-Prozessor.**

Der Acorn B war von Anfang an so konzipiert, daß er sich zu einem vollwertigen Bürorechner ausbauen lassen sollte. Mit dem Torch Disk Pack kann der Acorn-Besitzer nun auch die Fülle von Software nutzen, die für das CP M-Betriebssystem verfügbar ist. Das ganze Paket kostet nur wenig mehr als ein Paar Acorn-Floppylaufwerke.

Der CP/M-Betrieb setzt einen Z80-Prozessor und Diskettenstationen voraus. Der Acorn B arbeitet zwar mit einem 6502, aber das Torch Disk Pack enthält den erforderlichen Z80, und der überläßt dem 6502 des Acorn nur die Ein- und Ausgabe. Tastatur, Bildschirm und Laufwerke werden also vom 6502 gesteuert, während der Z80 mit dem ganzen 64K-RAM für den CP/M-Betrieb und die Systemorganisation frei ist. Der Datenverkehr zwischen den beiden Prozessoren erfolgt über die Prozessor-Schnittstelle („Tube“).

Der Z80 wird mit dem 64K-RAM auf einer separaten kleinen Leiterplatte geliefert, die in den Acorn B eingesetzt wird. Dessen Netzteil wird nun nicht mehr benötigt; der Rechner wird mit an die Spannungsversorgung der Torch-Laufwerke angeschlossen. Die Laufwerke arbeiten doppelseitig mit 2x80 Spuren, ähnlich wie die 800-K-Byte-Station, die Acorn selbst liefert.

## Einfache Inbetriebnahme

Alternativ bietet Torch das Paket unter dem Namen ZEP 100 auch ohne die Laufwerke an – für Acorn-Besitzer, die bereits über Diskettenstationen verfügen. Im übrigen vertreibt Torch auch eine Reihe von Bürorechnern mit Acorn-Rechnerplatine, Diskettenstation, eingebautem Bildschirm und Modem.

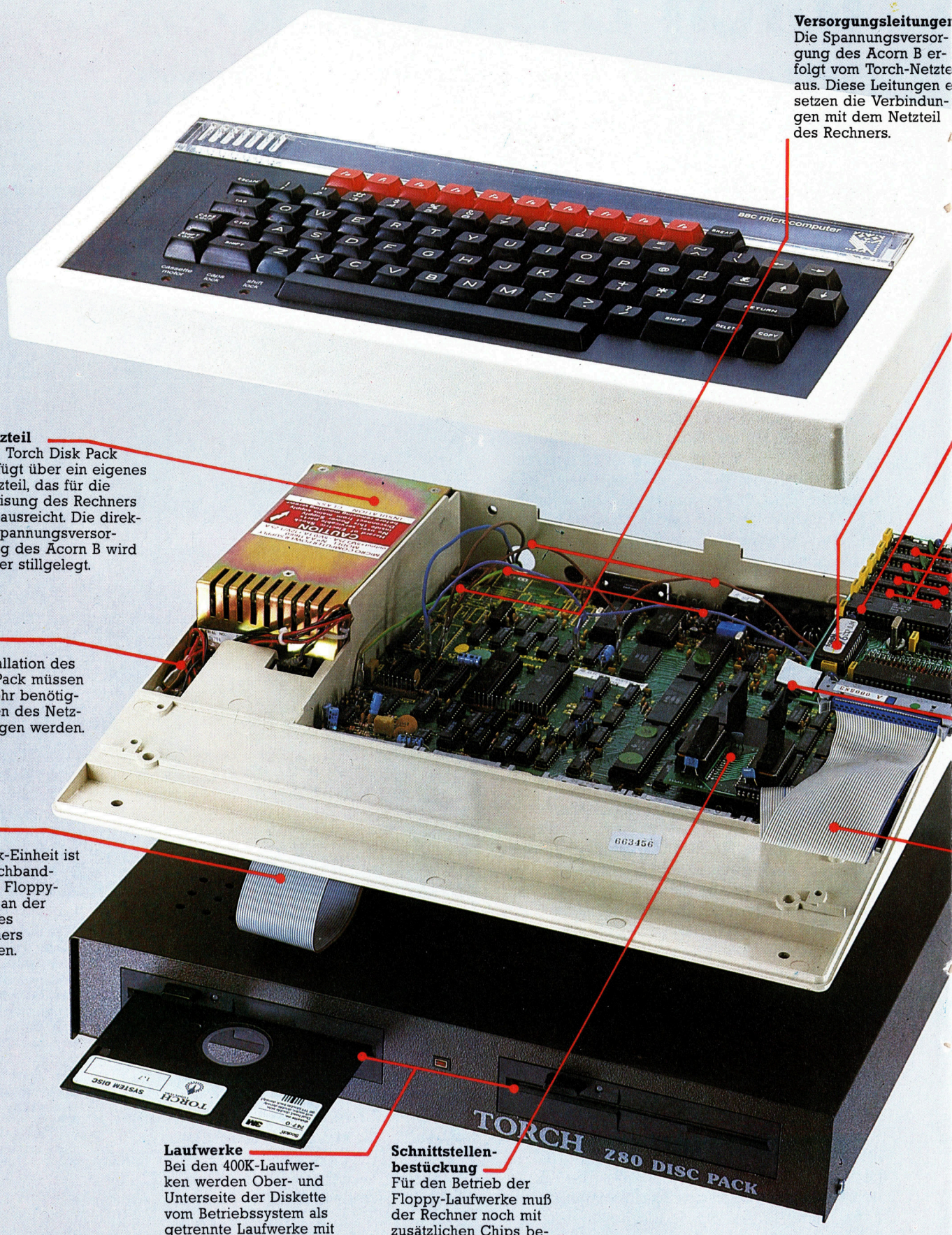
Das Disk Pack kommt direkt unter den Rechner, die Verbindung erfolgt über kurze Kabelstücke. Leider liegt die Tastatur nun allerdings einige Zentimeter über der Tischplatte, was die Bedienung nicht gerade erleichtert.

Alle erforderlichen Teile für die Erweiterung des Acorn sind dem Torch Disk Pack beigefügt – mit einer Ausnahme. Die zusätzliche Chip-Bestückung für die Floppy-Schnittstelle im Rechner gehört nicht dazu. Wer den Acorn nicht gleich in entsprechender Ausstattung gekauft hat und nur die Standardausführung des Acorn B besitzt, muß für den Bausteinsatz knapp vierhundert Mark hinblättern.


**Das „Disk Pack“ der Firma Torch verwandelt den Acorn B in einen professionellen Rechner, weil es einen Z80-Prozessor mit eigenem 64K-RAM enthält, so daß ein CP/M-Betriebssystem eingesetzt werden kann. Das Paket bietet zwei Diskettenlaufwerke zu je 400 KByte.**

Wurden die Komponenten richtig verbunden, so verhält sich die Anlage nach dem Einschalten sofort ganz wie ein CP M-Rechner (im Bildschirm-Mode 3). Das Disketten-Betriebssystem ist zwar eine Torch-Version namens MCP, es ist dem CP M-System aber eng verwandt, und die meisten der vielen hundert CP M-Programme sind damit lauffähig. Ein wesentlicher Unterschied liegt darin, daß das MCP-System in einem ROM im Acorn-Rechner steht und nicht erst geladen werden muß.

Im MCP gibt es viele Befehle, die dem Acorn-Besitzer vertraut sind. Mit MODE können verschiedene Bildschirmdarstellungen gewählt werden, und auch alle anderen Bildschirm- und *FX-Kommandos sind verwendbar. *KEY dient weiterhin zur Belegung der




**Versorgungsleitungen**
Die Spannungsversorgung des Acorn B erfolgt vom Torch-Netzteil aus. Diese Leitungen ersetzen die Verbindungen mit dem Netzteil des Rechners.

**Netzteil**
Das Torch Disk Pack verfügt über ein eigenes Netzteil, das für die Speisung des Rechners mit ausreicht. Die direkte Spannungsversorgung des Acorn B wird daher stillgelegt.

**Unbenutzte Leitungen**
Bei der Installation des Torch Disk Pack müssen die nicht mehr benötigten Leitungen des Netzteils abgezogen werden.

**Laufwerkanschluß**
Die Laufwerk-Einheit ist über ein Flachbandkabel an die Floppy-Schnittstelle an der Unterseite des Acorn-Rechners angeschlossen.

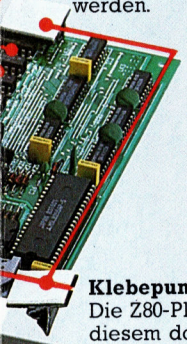
**Laufwerke**
Bei den 400K-Laufwerken werden Ober- und Unterseite der Diskette vom Betriebssystem als getrennte Laufwerke mit je 200K behandelt.

**Schnittstellenbestückung**
Für den Betrieb der Floppy-Laufwerke muß der Rechner noch mit zusätzlichen Chips bestückt werden.

**ROM mit CCCP**
Der CCCP (Cambridge Console Command Processor) ist ein Teil des MCP-Betriebssystems von Torch. Der Rest ist in einem ROM untergebracht, das einen der Leersockel des Rechners belegt.

**Z80-Microprozessor**
Die Aufgaben des 6502-Prozessors werden weitgehend dem Z80 übertragen. Das ermöglicht den Einsatz des Betriebssystems MCP von Torch und damit die Benutzung einer Fülle kommerzieller Software.

**64K-RAM**
Dieses Zusatz-RAM ist dem Z80 zugeordnet, während die 32K des Acorn weiter verwendet werden.



**Klebepunkte**
Die Z80-Platine wird mit diesem doppelseitigen Klebeband einfach innen am Rechnergehäuse befestigt.

**Verbindungskabel**
Die Platine mit dem Z80-Zusatzprozessor hängt über ein Flachkabel an der Prozessor-Erweiterungsschnittstelle (Tube).

Funktionstasten, vier sind bereits mit einer Auswahl nützlicher Befehle versehen.

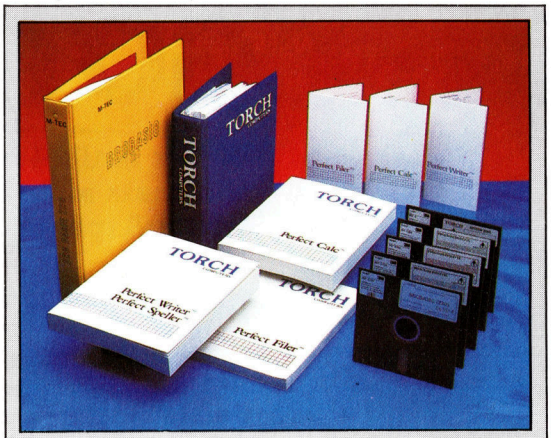
Daneben gibt es viele andere Kommandos für den Umgang mit den Dateien auf Diskette. Maschinenprogramme können zum Beispiel einfach durch Eintippen ihres Namens geladen werden, und ein Befehl lädt spezielle Dateien für den Aufbau von Bildern auf Diskette. PRINT besorgt die Ausgabe von Textdateien auf dem Drucker, TYPE auf dem Bildschirm. Mittels COMMAND wird eine Datei als Folge von Kommandos interpretiert, ähnlich wie es der Befehl EXEC beim Acorn-BASIC bewirkt.

Weitere Dienstprogramme stehen auf einer mitgelieferten Systemfloppy zur Verfügung und können durch Eingabe des betreffenden Programmnamens geladen werden. Dazu gehört eine Routine zum Verändern der Schrifttype auf dem Bildschirm, ein Programm zum Komponieren, ein Fehlerbeseitigungsprogramm für den Maschinencode, ein Disketten-Editor und ein Programm für das Lesen von Acorn-Disketten auf dem Torch-Laufwerk und umgekehrt – sehr nützlich, weil die Formatierung unterschiedlich ist. Das ermöglicht Ihnen beispielsweise, unter Verwendung einer BASIC-Routine ein Textfile zu erstellen und dann mit einem CP/M-Textverarbeitungsprogramm Veränderungen darin vornehmen. Die beiden doppelseitigen Diskettenlaufwerke werden im übrigen von Torch wie von Acorn eigenartigerweise als vier getrennte Laufwerke behandelt.

Ungeachtet aller Aufwertungen können Sie die Anlage auch weiterhin als ganz „normalen" Acorn benutzen. Nach Eingabe von *BASIC hat der Rechner die ganze Sonderausstattung vergessen, einschließlich der Floppy-Laufwerke, und ist auf Cassettenbetrieb eingestellt. Wer ohne MCP mit Disketten arbeiten will, muß daher noch zusätzlich das eigene Diskettensystem (DFS = Disk Filing System) des Acorn B aktivieren. Die Rückumstellung auf Z80-Betrieb ist jederzeit durch Ein- und Ausschalten oder durch Eingabe von *MCP möglich.

Der Acorn B ist zwar auch in der Standardausführung ein Heimcomputer mit vielen Möglichkeiten, aber eben doch nur ein Heimcomputer. Für den kommerziellen Einsatz muß ein Rechner sehr viel mehr bieten, als es der Acorn allein kann. In Verbindung mit dem Torch Disk Pack steht bei Bedarf jederzeit noch der Heimcomputer zur Verfügung, aber die Anlage ist darüber hinaus mit vielen Merkmalen eines Bürorechners ausgerüstet

**Installation des Disk Pack**
Die Ausrüstung Ihres Acorn B mit dem Torch Disk Pack erfordert einige Bastelei. Weil das Acorn-Netzteil nicht mehr benötigt wird, sind alle sieben Leitungen für die Spannungsversorgung auf der Rechnerplatine abzuziehen. Das Laufwerk-Flachkabel wird in den Diskettenanschluß des Acorn B und das ROM mit dem Betriebssystem in einen Leersockel des Rechners gesteckt. Außerdem muß die Prozessor-Platine mit dem RAM im Innern des Acorn B untergebracht werden.

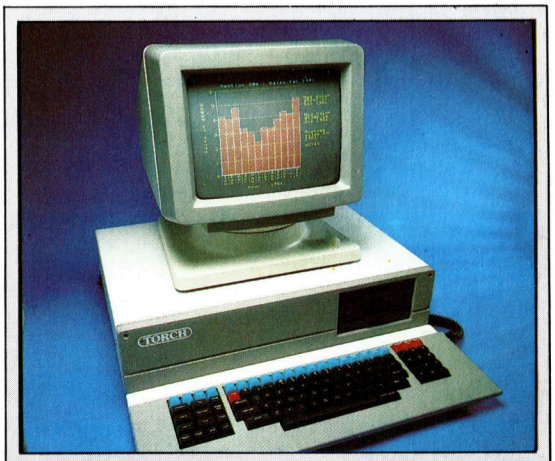



**Mitgelieferte Software**
Im Preis des Torch Disk Pack sind mehrere Softwarepakete enthalten, und zwar vorwiegend Büroprogramme: der „Perfect Writer" (ein Textverarbeitungspaket), der „Perfect Speller" (ein Rechtschreibungs-Programm), der „Perfect Filer" für die Datenbankorganisation und das Programm „Perfect Calc" für Tabellenkalkulationen.

Daneben gibt es eine BASIC-Version für den Z80-Zusatzprozessor. Der Befehlssatz ist der gleiche wie bei der ROM-Version des Acorn B. Statt des 6502-Assemblers ist sinngemäß ein Z80-Assembler vorgesehen. Der Benutzer kann stets über 48K freie Speicherkapazität verfügen, unabhängig vom gewählten Bildschirm-Modus. Das allein ist schon eine erhebliche Verbesserung.

**Der DFS-Baustein**
Die Acorn-Platine müssen Sie für den Diskettenbetrieb mit etlichen Chips zusätzlich bestücken. Der Bausteinsatz gehört nicht zum Lieferumfang des Disk Pack. Weil die Chips nicht speziell für das Torch-Paket zusammengestellt sind, finden Sie dabei einen „DFS"-Chip. Er enthält das Diskettenbetriebssystem von Acorn – damit können Sie auf den Torch-Laufwerken die Acorn-Diskettensoftware für den 6502-Prozessor laufen lassen.

Wegen der Inkompatibilität der Diskettenformate können die Torch-MCP-Programme mit Acorn-Disketten und die Acorn-Programme mit Torch-Disketten nichts anfangen. Für die dafür erforderliche Format-Umsetzung wird jedoch ein Systemprogramm mitgeliefert.




**Rechner fürs Büro**
Der erste Rechner von Torch war ein Bürocomputer für über 13 000 Mark, der inzwischen als „700er Serie" auch in Versionen mit dem UNIX-Betriebssystem hergestellt wird. Dazu gibt es eine 300er-Terminal-Serie für den Aufbau von Netzwerken.

# GOTO ENDE

**Dieser Artikel ist der Abschluß unserer PASCAL-Serie. Wir gehen nochmals auf die Beschreibung der Datenstrukturen ein und berühren einige Aspekte der Sprache, die bisher nicht erwähnt wurden.**

Niklaus Wirth, der Vater von PASCAL, gab einem seiner Bücher den Titel „Datenstrukturen + Algorithmen = Programme" und deutete damit an, wie wichtig korrekte Datenbeschreibungen für die Algorithmen sind, die diese Daten bearbeiten. Wenn wir beispielsweise eine Zeichenfolge in einem *char*-Array unterbringen, brauchen wir für deren Verarbeitung völlig andere Abläufe als beispielsweise bei verketteten Listen.

Nehmen wir als Beispiel die einfache Aufgabe, die Länge eines Strings festzustellen. Bedenken Sie, daß ungenutzte Array-Elemente mit dem ASCII-Zeichen für Null, *chr*(0) ausgefüllt sind beziehungsweise dynamische Listen mit dem Pointerwert NIL abschließen. Die Array-Version sieht nicht kompliziert aus:

```
FUNCTION Laenge (S : String): Cardinal;
  VAR
    N          : 0. .StringLaenge;
    gefunden   : boolean;
  BEGIN
    N          := 0;
    gefunden   := false;
  REPEAT
    N          := N + 1;
    gefunden   := S [ N ] = chr(0)
  UNTIL gefunden OR (N=StringLaenge);
  IF gefunden
    THEN
```

## Das BaumSort-Programm

**Dieses Programm setzt einige Prozeduren des Programms „KreisListe" aus dem vorigen Artikel ein. Ein Datensatz besteht aus einem Namen, dem Schuldenbetrag und anderen Feldern, die – falls gewünscht – hinzugefügt werden können. Die Daten werden über die Tastatur eingegeben und in den „binären Baum" eingefügt.**

**Jedes Strukturelement des Baumes hat zwei Pointer, die es mit den Elementen „darüber" und „darunter" verbinden. Neue Daten lassen sich einfügen, indem sie mit dem Namensfeld jedes Strukturelementes verglichen werden und dann in das entsprechende Verbindungsfeld (NiedrigZweig und HochZweig) verzweigen. Sobald ein leeres Strukturelement gefunden ist (mit einem NIL-Wert in der Verzweigung), wird das neue Element eingesetzt. Die Daten werden dann mit einem einfachen recursiven Vorgang in die Datei geschrieben.**

**Unser Beispiel läßt sich als Grundlage für eine Datenbank verwenden, die Sie genau auf Ihre Zwecke abstimmen können.**

**Für einfache Anwendungen ist die Kreisliste vermutlich besser geeignet. Da es dort keine NIL-Pointer gibt, entfällt bei jedem Vergleich auch der doppelte Test. Die Anzahl der NILs des binären Baums entspricht der Zahl der Strukturelemente + 1.**

```
PROGRAM BaumSort ( input, output, Datei );
CONST
  DateiName     = 'BaumDaten',
  StringLaenge  = 25;
TYPE
  Cardinal      = 0 . . MaxInt;
  StringGroesse = 1 . . StringLaenge;
  String        = PACKED ARRAY [ StringGroesse ]
                    OF char;
  Objekt        = RECORD
                    Name     : String;
                    (* andere Felder . *)
                    Schulden : Cardinal
                  END, (* Objekt *)
  Dateiart      = FILE OF Objekt;
  Baum          = ↑Zweig;
                  (* Vorausblick auf . *)
  Zweig         = RECORD
                    Element     . Objekt;
                    NiedrigZweig,
                    HochZweig . Baum
                  END; (* Zweig *)
VAR
  Daten         : Objekt,
  Datei         : Dateiart;
  Basis         : Baum,
INCLUDE 'Utils.src' (** Quellendatei mit:
                    LeerzeichenUebersprıngen,
                    Einlesen und ZeileLesen;
                    — siehe Programm KreisListe **)
(* ——————————————————————————— *)
PROCEDURE BetragLesen ( VAR Betrag : Cardinal );
  (* feststellen, ob Cardinalwert legal ist *)
  VAR
    OK : boolean;
  BEGIN
    REPEAT
      REPEAT
        write ( 'Betrag ? ' : 20 );
        IF EoLN ( input ) THEN
          ReadLn ( input );
        LeerzeichenUebersprıngen ( ınput )
        (* bei Zeilenende Neueingabe ermoeglichen *)
      UNTIL NOT EoLn ( input );
      Einlesen ( input, Betrag, OK );
      IF NOT OK THEN
        WriteLn ( '——— FEHLER ———' : 20,
                  ' Bitte Neueingabe' )
    UNTIL OK (* Betrag muss legal sein *)
  END; (* BetragLesen *)
(* ——————————————————————————— *)
PROCEDURE Wachsen ( VAR       Blatt : Baum;
    (* an den Baum anhaengen *) Daten : Objekt );
  BEGIN
    new ( Blatt );
    WITH Blatt↑ DO
      BEGIN
        Element     := Daten;
        NiedrigZweig := NIL; (* noch keine weiteren Daten *)
        HochZweig   := NIL (* nicht mehr ın diesem
          Zweig *)
      END
  END, (* Wachsen *)
```

```
        Laenge := N — 1
      ELSE
        Laenge := StringLaenge
  END, (* Laenge *)
```

Hier können jedoch Mißverständnisse und Fehler entstehen. Warum benötigen wir beispielsweise die lokale Boolesche Variable *gefunden?* Und warum wird *Laenge* der Wert N−1 zugewiesen? Dies sind kleine Probleme. Da kompliziertere Algorithmen jedoch ein größeres Fehler-Risiko mit sich bringen, würden Strukturen dieser Art unnötig zusätzliche Verwirrung stiften.

Der Einsatz von „Bit-Flags" (Boolescher Variablentyp) ist einer der ältesten Programmierkniffe. Oft werden diese Typen jedoch nicht als logischer Teil des Algorithmus eingesetzt, sondern nur zur Steuerung des Computers. In unserem Beispiel muß daher die lokale Variable N verwandt werden, nicht *Laenge, da Laenge* Funktionsname ist und keine Variable. Wenn *Laenge* auf der rechten Seite einer Anweisung erscheint, wie:

```
Laenge  = Laenge + 1
```

wird das Funktionsmodul recursiv aufgerufen.

Stellen Sie diesen Ablauf jetzt einer *Laenge*-Funktion für verkettete Listen gegenüber. Bedenken Sie dabei, daß die Typendefinition von *String* bei dynamischen Darstellungen anders ist, da hier Strings von beliebiger Länge entstehen können und Datenbeschreibungen daher oft recursiv definiert werden müssen.

Viele einfache Beispiele für Recursionen lassen sich auch als iterative Algorithmen darstellen. Sehen wir uns dazu die *Laenge*-Funktion für einen recursiven Stringtyp an:

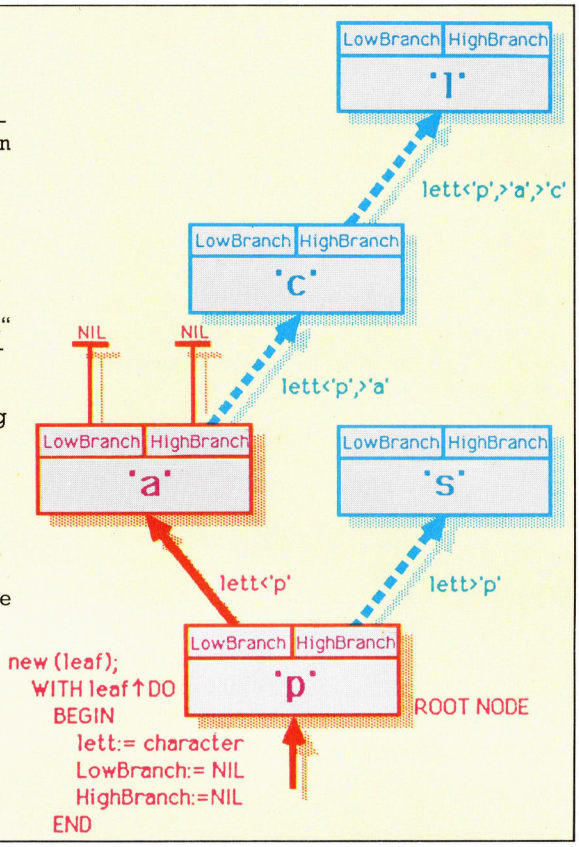
```
FUNCTION Laenge (S :String) : Cardinal;
  BEGIN
    IF S = NIL
      THEN
        Laenge :=0
      ELSE
        Laenge := succ(Laenge(S↑.next))
  END, (* Laenge *)
```

Am Anfang der Liste steht S, während *S↑.next* das Pointer-Feld des nächsten Datensatzes anspricht. Dieser Aufbau läßt sich auch als Liste ansehen, die mit dem nächsten Datensatz anfängt. Wenn wir kein NIL finden, wird die

```
(* ———————————————————————————— *)
PROCEDURE Klettern ( VAR Schoessling      Baum,
    (* Platz zum Einfuegen finden *) Daten   Objekt ),
  VAR
    Stamm    Baum,
    Groesser  boolean,
  BEGIN
    WHILE Schoessling <> NIL DO
      BEGIN
        Stamm    = Schoessling,
        Groesser  = Daten.Name > Stamm↑
                          Element.Name,
        IF Groesser
          THEN (* aufwaerts klettern *)
            Schoessling  = Schoessling↑
              HochZweig
          ELSE (* abwaerts klettern *)
            Schoessling := Schoessling↑
              NiedrigZweig
      END,
    Wachsen ( Schoessling, Daten );
    IF Groesser
      THEN (* darueber einsetzen *)
        Stamm↑.HochZweig   := Schoessling
      ELSE (* darunter einsetzen *)
        Stamm↑.NiedrigZweig := Schoessling
  END, (* Klettern *)
  (* ———————————————————————————— *)
PROCEDURE Schreiben ( VAR      F          Dateiart,
    (* Daten in Datei schreiben *) Wurzel   Baum ),
  BEGIN
    IF Wurzel <> NIL THEN
      WITH Wurzel↑ DO (* Recursion   *)
        BEGIN (* in Reihenfolge schreiben *)
          Schreiben ( F, NiedrigZweig ), (* zuerst
            niedrig, *)
          write ( F, Element ), (* dann die Wurzel *)
          Schreiben ( F, HochZweig ), (* dann hoch *)
        END
  END; (* Schreiben *)
  (* ———————————————————————————— *)
PROCEDURE Wiederholen ( VAR Wurzel   Baum ),
    (* Speicherplatz wieder verfuegbar machen *)
  BEGIN
    IF Wurzel <> NIL THEN (* Recursion *)
      BEGIN (* erst die Zweige — *)
        Wiederholen ( Wurzel↑.NiedrigZweig ),
        Wiederholen ( Wurzel↑.HochZweig );
        dispose ( Wurzel ) (* dann die Wurzel *)
      END
  END, (* Wiederholen *)
  (* ———————————————————————————— *)
BEGIN (* BaumSort — Hauptprogramm *)
  assign ( Datei, Dateiname );
  page ( output ),
  WriteLn ( '=== BaumSort ===' : 25 );
  WriteLn;
  WriteLn ( 'Bitte Daten eingeben (erst Nach-
    name)' );
  write ('Name ? ' ),
  Einlesen ( Daten.Name ),
  BetragLesen ( Daten.Schulden ),
  Wachsen ( Basis, Daten ), (* Baumstamm
    'einpflanzen' *)
  write ('Name ? ' );
  ZeileLesen ( Daten.Name ),
  WHILE Daten.Name [ 1 ] <> chr ( 0 ) DO
    BEGIN
      BetragLesen ( Daten.Schulden ),
      Klettern ( Basis, Daten );
      WriteLn ( 'RETURN = Ende'   40 ),
      write ('Name ? ' );
      ZeileLesen ( Daten.Name )
    END;
  (* Platz fuer weitere Bearbeitung, dann Datei
  schreiben : *)
  WriteLn ( 'Schreiben Datei . ', DateiName );
  rewrite ( Datei ),
  Schreiben ( Datei, Basis ),
  WriteLn;
  WriteLn ( 'Schulden'  8, 'Name'   StringLaenge );
  WriteLn;
  reset ( Datei ), (* Datei lesen und in geordneter *)
                   (* Reihenfolge schreiben : *)
  WHILE NOT EoF ( Datei ) DO
    BEGIN (* Records einzeln lesen *)
      read ( Datei, Daten ),
          (* die Felder auf den Bildschirm bringen *)
      WITH Daten DO
        WriteLn ( Schulden   8, '   ', Name )
    END,
  Wiederholen ( Basis ); (* Speicher freisetzen *)
        (* fuer andere Aufgaben *)
END
```

**Blatt für Blatt**
Das Programm „Baum-Sort" setzt eine binäre Baumstruktur ein, um Daten nach Schlüsselfeldern geordnet speichern zu können. Die einzelnen Elemente werden dabei jeweils nach der alphabetischen Position des Namensfeldes dem hohen oder dem niedrigen Zweig zugeordnet. Die Prozedur „Wachsen" richtet neue Strukturelemente ein, während „Klettern" den Weg in den richtigen NIL-Zweig findet.

Das Diagramm zeigt das Einsortieren des pascal-Strings in eine einfache binäre Baumstruktur, deren Strukturelemente einzelne Zeichen (lett) enthalten. Die rote Markierung zeigt an, wie die Prozedur „Wachsen" den Baum vergrößert.




Laenge dieses Datenelementes bewertet und mit *succ* inkrementiert.

Viele Probleme lassen sich nur mit Recursionen lösen. Das vorstehende Beispiel kann jedoch auch ohne Recursion programmiert werden (wenn auch ein wenig umständlicher):

```
FUNCTION Laenge (S :String) : Cardinal;
  VAR
    N : Cardinal;
  BEGIN
    N := 0;
      WHILE S <> NIL DO
        BEGIN
          N := N + 1;
          S := S↑.next
        END;
    Laenge := N
  END; (* Laenge *)
```

Die Deutlichkeit dieses Algorithmus ergibt sich dabei wie selbstverständlich aus der recursiven Natur der Datenstruktur.

Viele PASCAL-Compiler unterstützen Compileranweisungen, die nicht Elemente der Programmiersprache sind. Der ISO-Standard erkennt dabei eigentlich nur die Anweisung *Forward.* Zwei Prozeduren oder Funktionen sind „wechselseitig recursiv", wenn sie sich gegenseitig aufrufen.

In dieser Situation wird zunächst der Kopf des einen Unterprogramms deklariert und sein „Inneres" durch die Compileranweisung *FORWARD* ersetzt. Nach der vollständigen Definition des zweiten Moduls wird der Kopf des ersten nochmals in abgekürzter Form (ohne Parameterliste) angegeben und dahinter der Inhalt des Moduls aufgeführt. Andere Compileranweisungen beeinflussen oft die Compilerzeit, doch sollte von dieser Möglichkeit nur wenig Gebrauch gemacht werden, wenn die Programme auch mit anderen Compilern funktionieren sollen. Viele Optionen sind nicht übertragbar und werden daher von anderen Compilern auch nicht ausgeführt.

Einige wenige Compiler (beispielsweise von HiSoft) haben eine andere Syntax für die Deklaration des Forward-Pointers. Hier kann nur das Handbuch weiterhelfen. Es gibt jedoch nur wenige PASCAL-Varianten, die nicht mit der ISO-Definition übereinstimmen.

In den Versionen UCSD, TCL/RML, Oxford und HiSoft wurde die Prozedur *dispose* durch die außerhalb des Standards liegenden Prozeduren *mark* und *release* ersetzt.

Eine weitere Datenstruktur von PASCAL muß noch erwähnt werden: die „Variante". Bisher haben wir für das Speichern unterschiedlicher Datentypen einen Record mit entsprechenden Feldern verwandt. Wie aber kann ein Teil oder der gesamte Record flexibel gestaltet werden, wie beispielsweise beim Speichern des Geburtsnamens einer verheirateten Frau? Zuerst wird der „feste" Teil des Records definiert und dann der Teil mit variierender Länge mit CASE und OR eingeleitet:

```
TYPE
  Art      = (maennlich, weiblich);
  Variant = RECORD
    (* erst die gemeinsamen (festen)
    Felder *)
    CASE verheiratet : boolean OF
      false : ();
      true : (Heiratsdatum : String;
              CASE Geschlecht : Art OF
                maennlich : ();
                weiblich  : (Geburtsname :
                             String))
    END; (* Variant *)
```

Beachten Sie, daß die leeren Felder mit Klammern gekennzeichnet sein müssen. Die längste Variante legt den Platzbedarf der Datei fest, doch läßt sich mit Pointern auf die Varianten – *new (p, true, maennlich)* eine Speicherplatzverschwendung verhindern.

Die wenigen Schwächen von PASCAL wurden im Laufe der Zeit behoben. Die Sprache ist außerordentlich flexibel, handlich, leistungsstark, für zahlreiche Aufgaben geeignet und leicht zu erlernen. Wenn Sie innerhalb des ISO-Standards bleiben, gibt es natürlich einige Abläufe auf Systemebene, die man nicht einsetzen kann. Diese Routinen lassen sich jedoch in Assembler oder BCPL schreiben und an das gewünschte PASCAL-Programm hängen. Der größte Vorteil von PASCAL ist jedoch die Möglichkeit, den entwickelten Code auf vielen Micros, Minis oder Großanlagen einsetzen zu können. PASCAL kommt damit einem Esperanto der Computerwelt nahe.

# Schrittweise

**Für den Antrieb des Selbstbau-Roboters wird ein spezieller Elektromotor verwendet, der sogenannte Schrittmotor. Er eignet sich besonders gut für die Ansteuerung durch einen Computer.**

Der Aufbau eines Schrittmotors weicht stark von der Konstruktion eines gewöhnlichen Elektromotors ab. Am Beispiel eines vereinfachten Exemplars sollen die Unterschiede verdeutlicht werden. Betrachten Sie dazu die „Schritt für Schritt"-Anleitung: Der Motor hat zwei getrennte Wicklungen („a" und „b") auf dem Stator sowie zwei Paare magnetischer Pole auf dem Rotor. Unser Robot-Motor arbeitet zwar prinzipiell gleich, hat aber sehr viel mehr Wicklungen und Magnetpole, als das vereinfachte Beispiel zeigt.

Die exakte Regelmöglichkeit des Schrittmotors fordert allerdings auch ihren Preis: Der Motor verbraucht im Stillstand genau so viel Strom wie im Betrieb und ermöglicht keine hohen Drehzahlen, weil sich die einzelnen Wicklungen nicht schnell genug ein- und ausschalten lassen.

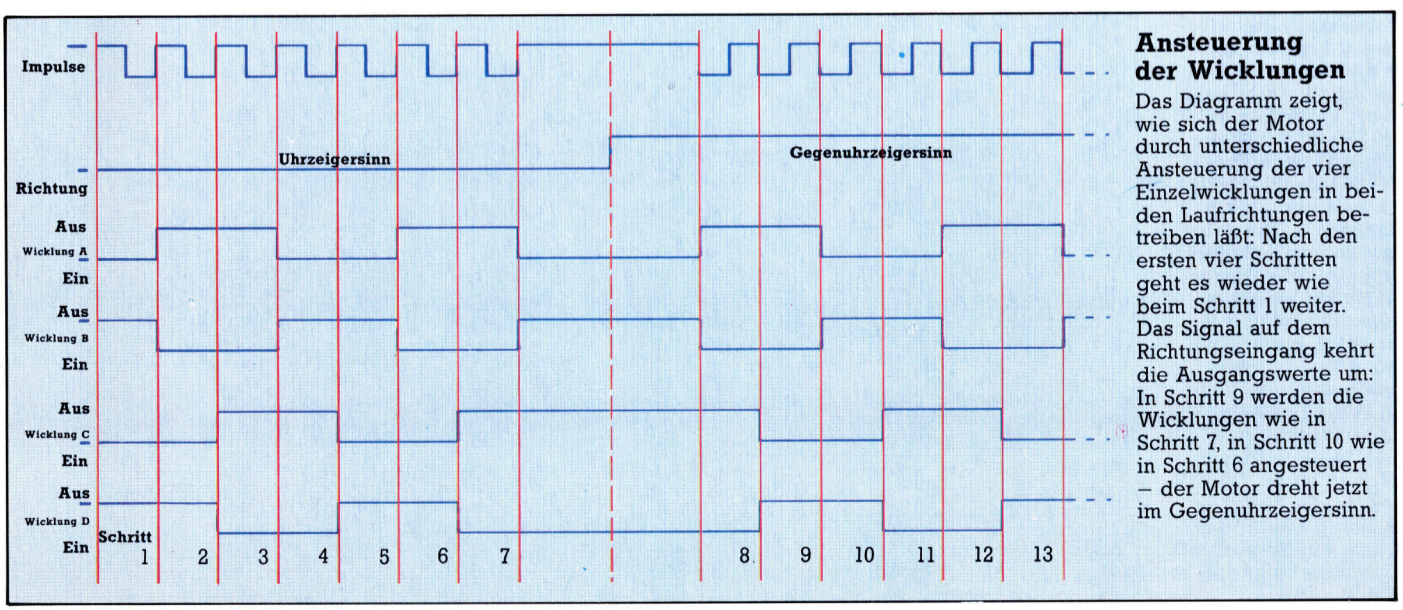
Der vereinfachte Motor könnte nur in 45-Grad-Schritten laufen. Dazu kommt, daß man die Bewegungsrichtung bei diesem Gerät nicht verändern könnte. Anders ist es beim Roboter-Schrittmotor: Er ist mit vier Wicklungen ausgestattet, die paarweise angesteuert werden. Auch der Rotor ist mit zahlreichen Wicklungen ausgerüstet. Durch diesen Aufbau ist es möglich, die Bewegungsrichtung zu kontrollieren, und die Schrittweite kann auf 7,5 Grad reduziert werden. Dazu müssen alle vier Wicklungen getrennt und in einer bestimmten Reihenfolge mit Strom versorgt werden:

**Stromzufuhr zu den Statorwicklungen**

| Schritt | Wicklung A | Wicklung B | Wicklung C | Wicklung D |
|---|---|---|---|---|
| 1 | ein | aus | ein | aus |
| 2 | aus | ein | ein | aus |
| 3 | aus | ein | aus | ein |
| 4 | ein | aus | aus | ein |
| 5 | ein | aus | ein | aus usw. |

Diese Abfolge ließe sich rein softwaremäßig mit vier User-Port-Bits für die vier Wicklungen erzeugen. In BASIC wäre das entsprechende Programm allerdings ziemlich kompliziert und auch nicht schnell genug. Einfacher geht es mit einem speziell für die Steuerung von Schrittmotoren entwickelten IC, dem SAA 1027. Die integrierte Schaltung enthält neben den Ausgangstreibern auch die Logik zur Ansteuerung der einzelnen Motorwicklungen.

Um den Motor einen Schritt weiter zu bewegen, ist außer einem einzelnen Impuls vom User Port noch ein zweites Signal nötig, das die Drehrichtung angibt. Neben dem Reset-Signal verarbeitet das Steuer-IC noch zwei weitere: Ein Impuls auf dem Schritt-Eingang läßt den Motor um einen Schritt vorlaufen, mit einem Signal am Richtungseingang wird die Kombination der Ausgangswerte für die vier Wicklungen umgekehrt. Die Eingangssignale werden mit Hilfe eines bidirektionalen Zählers in die zur Versorgung der Wicklungen nötige Form umgesetzt.




**Ansteuerung der Wicklungen**

Das Diagramm zeigt, wie sich der Motor durch unterschiedliche Ansteuerung der vier Einzelwicklungen in beiden Laufrichtungen betreiben läßt: Nach den ersten vier Schritten geht es wieder wie beim Schritt 1 weiter. Das Signal auf dem Richtungseingang kehrt die Ausgangswerte um: In Schritt 9 werden die Wicklungen wie in Schritt 7, in Schritt 10 wie in Schritt 6 angesteuert – der Motor dreht jetzt im Gegenuhrzeigersinn.

Im SAA 1027 ist bereits ein Ausgangstreiber für eine Leistung von bis zu 500 mW integriert, der Motor kann also ohne zusätzliche Verstärker-Transistoren angeschlossen werden.

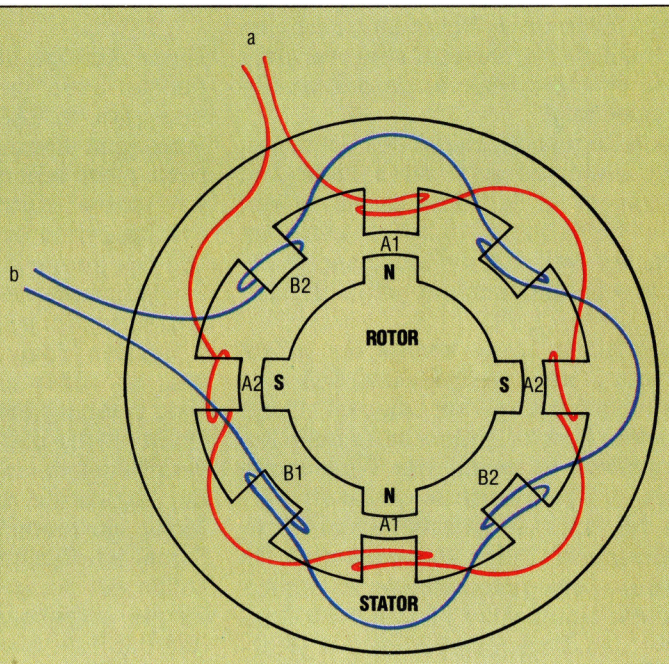
Unsere Schaltung zur Steuerung des Roboters wird recht einfach, weil die wichtigsten Komponenten bereits im Treiber-IC vorhanden sind. Jeder Motor muß einen eigenen Treiber erhalten. Die einzige Schwierigkeit besteht darin, daß die Motortreiber für Eingangssignale von zwölf Volt ausgelegt sind (ein „Low" entspricht 0 Volt, 7,5–12 Volt werden als „High" erkannt). Der User Port liefert nur fünf Volt. Wir brauchen also ein weiteres IC, das die User-Port-Signalspannungen auf ein höheres Niveau umsetzt. Als Konverter wird für diese Aufgabe der 40109-Chip eingesetzt.

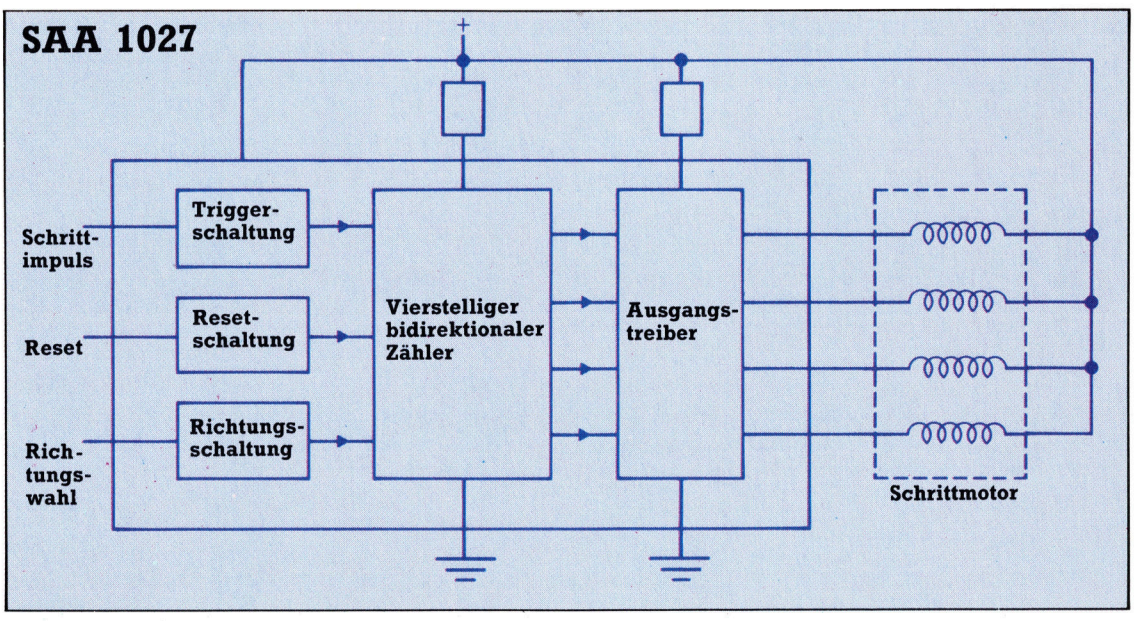
## Liste der Bauteile

| Anzahl | Bauteile |
|---|---|
| 1 | IC 40109 |
| 3 | IC-Sockel, 16-polig |
| 2 | Widerstände, 100 Ohm |
| 2 | Widerstände, 270 Ohm, 0,5 Watt |
| 2 | Kondensatoren, 0,1 µF |
| 1 | Kondensator, 1000 µF, 25 Volt |
| 1 | Lochplatine, 24 Streifen à 50 Löcher |
| 1 | Rolle Schaltdraht |
| 2 | Schrittmotor-Treiber SAA 1027 |

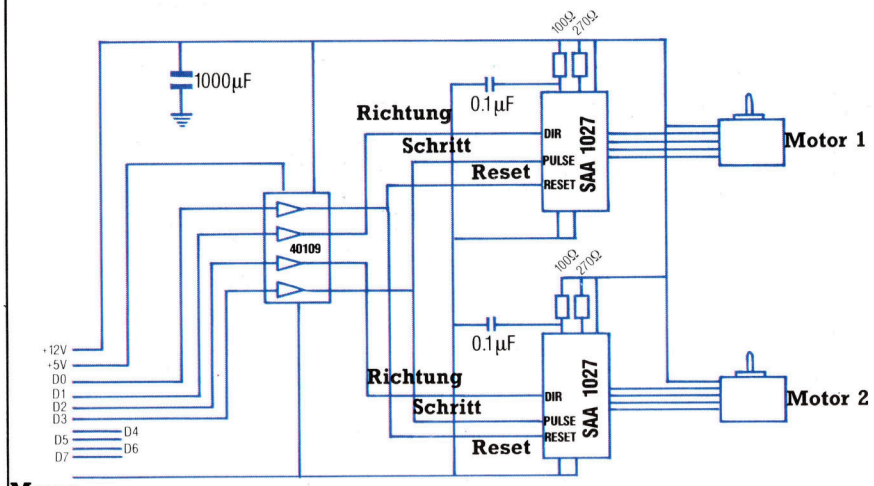
## Schritt für Schritt

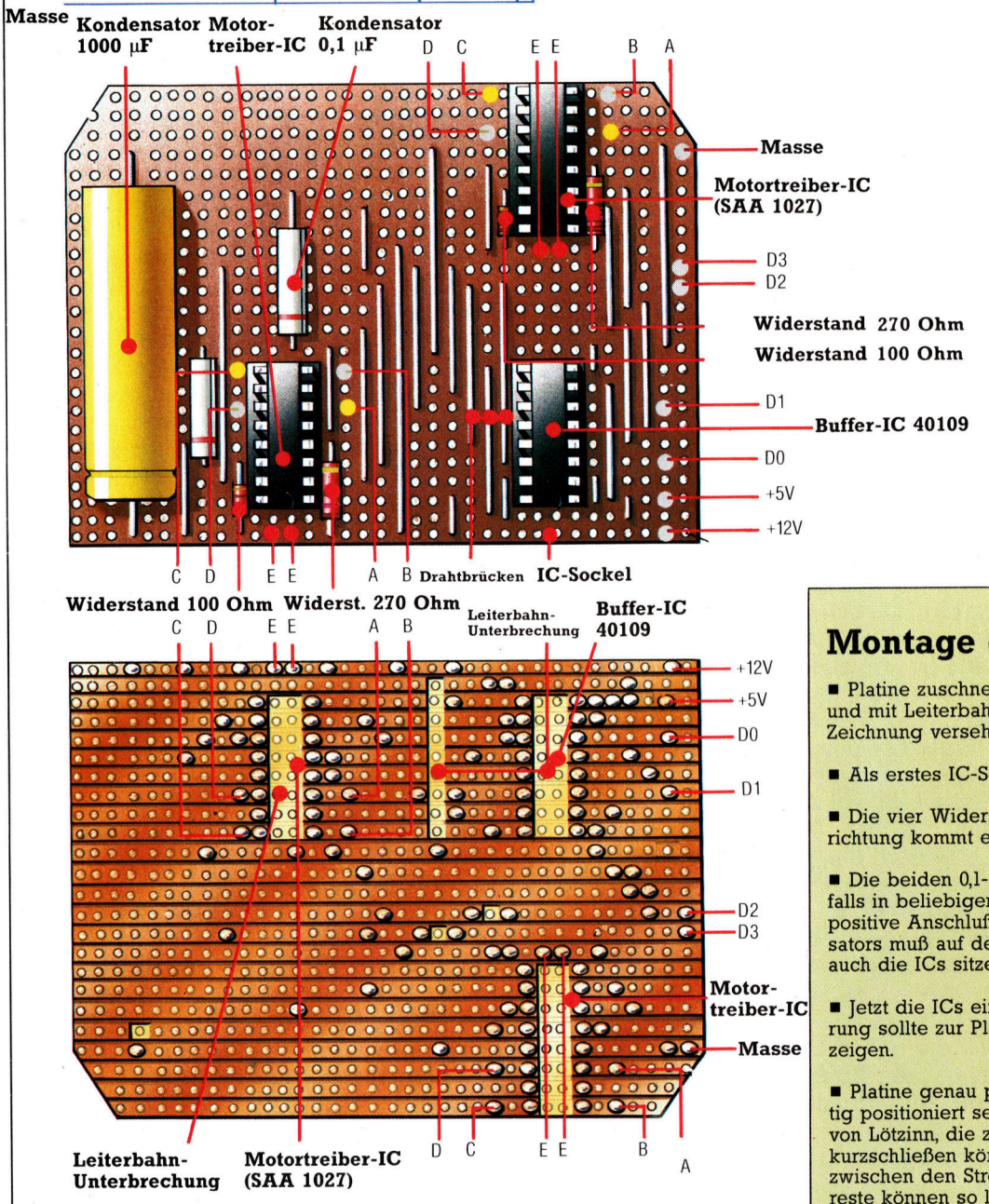
Diese vereinfachte Darstellung zeigt das Schema eines Schrittmotors mit den beiden Wicklungen „a" und „b". Durch abwechselndes Ansteuern der Wicklungen würde sich der Rotor im Uhrzeigersinn drehen. Beachten Sie, daß die Spulen A1 und A2 jeweils in unterschiedlichen Richtungen gewikkelt sind. Geht Strom durch die Wicklung „a", so werden durch die A1-Spulen magnetische Südpole, mit den A2-Spulen aber Nordpole erzeugt. Dieser recht einfache Motor würde pro Schritt um je 45 Grad vorlaufen. Der Motor unseres Selbstbau-Roboters hat mehr Wicklungen und mehr Rotorpole. Er kann sich in Winkeln von 7,5 Grad bewegen.




**Trotz seines komplizierten Aufbaus arbeitet das Treiber-IC für den Schrittmotor nach einem einfachen Prinzip. Damit der Motor richtig läuft, müssen die einzelnen Wicklungen in einer ganz bestimmten Reihenfolge aktiviert werden. Dazu läuft ein bidirektionaler Zähler bei jedem Impuls einen Schritt in der Folge weiter. Mit einem zusätzlichen Richtungssignal wird die Folge in Gegenrichtung durchlaufen – der Drehsinn des Motors kehrt sich um. Ein drittes Signal – Reset – stellt den Läufer des Motors bei Bedarf auf seine Anfangsposition zurück.**




**Der Aufbau der Schaltung für die Ansteuerung der beiden Schrittmotoren ist unkompliziert: Die beiden SAA-1027-ICs versorgen die Motoren mit den richtigen Steuerspannungen. Ein zusätzlicher Buffer (IC 40109) besorgt die Anpassung des User-Port-Niveaus von nur fünf Volt an die für die Motortreiber benötigten Signalspannungen von zwölf Volt. Der nächste Kursabschnitt zeigt, wie die Schaltung mit den Motoren und dem User Port verbunden wird.**

## Montage der Platine

- Platine zuschneiden (24 Streifen à 35 Löcher) und mit Leiterbahnunterbrechungen nach der Zeichnung versehen.
- Als erstes IC-Sockel und Drahtbrücken einlöten.
- Die vier Widerstände einlöten. Auf die Montagerichtung kommt es dabei nicht an.
- Die beiden 0,1-µF-Kondensatoren können ebenfalls in beliebiger Richtung eingelötet werden. Der positive Anschluß des großen 1000-µF-Kondensators muß auf der Seite der Platine liegen, auf der auch die ICs sitzen!
- Jetzt die ICs einsetzen. Die Seite mit der Markierung sollte zur Platinenseite mit den IC-Sockeln zeigen.
- Platine genau prüfen! Alle Bauteile müssen richtig positioniert sein. Achten Sie auch auf Spritzer von Lötzinn, die zwei benachbarte Platinenstreifen kurzschließen könnten. Mit einem scharfen Messer zwischen den Streifen durchkratzen – lose Zinnreste können so leicht entfernt werden.

In „Frankie Goes To Hollywood" müssen Räume nach verschiedenen Gegenständen durchsucht werden. Zieht man eine Schublade, wird der Inhalt auf dem Bildschirm dargestellt. Es ist auch möglich, bestimmte Bilder zu „betreten" und sich mit Arcadespielen zu beschäftigen.

# Relax, do it

**Das von Denton Designs entwickelte Programm „Frankie Goes To Hollywood" enthält die Philosophie von FGTH – von dieser irdischen realen Welt würden sie am liebsten an einen fantastischen Ort des Vergnügens fliehen.**

Frankie Goes To Hollywood hat weniger mit der Musik der gleichnamigen Gruppe, als vielmehr mit ihrer Philosophie zu tun – der Flucht aus der Alltagswelt (von „Mundanesville" nach „Pleasure Dome"), mit dem Ziel, dabei ein 100%iger „richtiger Mensch" zu werden.

Das Programm wurde von Denton Designs, einem Programmierteam aus Liverpool, entwickelt, woher auch die Band stammt. Ursprünglich arbeitete die Mannschaft für „Imagine" und ist nun für eine Reihe von Softwarehäusern auf freier Basis tätig. Das bekannte „Shadowfire" stammt auch von ihnen.

„Frankie" hat viele Ähnlichkeiten mit seinem Vorläufer, ist also auch ein Abenteuerspiel, in dem es eine Reihe von Orten zu erforschen gilt (in diesem Fall eine Reihe von Terrassenhäusern). Während man sich um die Häuser bewegt, kann man die darin befindlichen Möbel nach Gegenständen durchsuchen.

Denton Designs hat die Objekt-Suchmethode adaptiert, die für „Shadowfire" entwikkelt wurde. Hat man einen Gegenstand erst einmal gefunden, kann er durch Führung des Cursors und anschließendes Drücken des Actionsknopfs aufgehoben bzw. abgelegt werden. Im Laufe des Spiels entdeckt man vier kleine Piktogramme am Rande des Bildschirms, die den vier Komponenten der Persönlichkeit des Spielers entsprechen. Diese Elemente werden auf der rechten Seite des Bildschirms gezeigt. Sie wachsen, je weiter man sich zu einem „richtigen Menschen" entwickelt. Die diesen Komponenten entsprechenden Piktogramme sind sogenannte „Freuden-Pillen", die die „Persönlichkeit stärken".

Regelmäßig tauchen „Videos" auf, die den Zutritt zu einem völlig anders gearteten Spielelement erlauben. Viele Häuser sind mit Fernsehgeräten ausgestattet, über die man die Videocassetten abspielen kann. Bewegt man nun seine „Person" in den Videoschirm hinein, wandelt sich das Programm in ein Actionspiel. Während dieser Spielphase ist die Einnahme der Pillen empfehlenswert, da so die Punktzahl maximiert wird. In einigen Häusern hängen Bilder an der Wand. „Betritt" man diese, kann man ebenfalls Actionspiele spielen.

Ferner sind in den Häusern Disketten versteckt. Auf den ersten Blick scheint es, als gäbe es keinen Anwendungszweck dafür, da nirgendwo Computer zu sehen sind, auf denen sie laufen könnten. Doch in einem der Spiele ist ein Computer versteckt, und es geht darum, eben diesen zu finden.

Während man unterwegs ist, entdeckt man, daß ein Mord begangen wurde. Eine zusätzliche Aufgabe besteht deshalb darin, den Mörder zu finden. Indizien, die zum Mörder führen, werden in regelmäßigen Abständen eingeblendet. So kann etwa ein Hinweis sein „DER MÖRDER IST EIN SOZIALIST", wogegen ein anderer lautet „MISS MUNDANE STIMMT IMMER FÜR TORY", womit letztere automatisch von der Liste der Verdächtigen zu streichen ist.

Zwar sind viele Spielelemente von „Frankie Goes To Hollywood" aus anderen Programmen entliehen, und doch stellt dieses Spiel eine selbständige Einheit dar. Die Verwendung so vieler Elemente in einem einzigen Spiel ist derart ungewöhnlich, daß es die Spieler lange begeistern und fesseln wird.

**Frankie Goes To Hollywood:** Für ZX Spectrum und C 64
**Vertrieb:** Rushware
**Autor:** Denton Designs
**Joystick:** Erforderlich
**Format:** Cassette oder Diskette

# Register ziehen

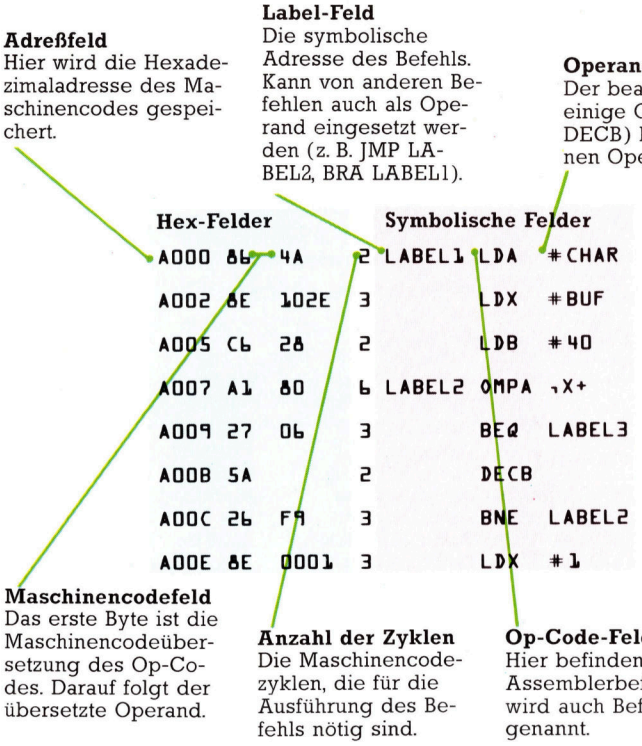
**Bei der Vorstellung des 6809 wurden zunächst die allgemeinen Funktionen des Prozessorregisters erläutert. In diesem Artikel untersuchen wir nun im einzelnen, welche Aufgaben die Register des 6809 ausführen.**

Register sind Speicherstellen, die sich im Prozessorchip befinden. Die Bewegung von Daten vom Arbeitsspeicher in diese Register und umgekehrt steuert den Ablauf der Assemblersprache. Einige Register – hauptsächlich die Adreßspeicher – haben eine Länge von 16 Bits, während die anderen acht Bits lang sind und eine Vielzahl verschiedener Aufgaben ausführen.

- **Indexregister** verändern die Adressen eines Programms.
- **Stapelzeiger** (Stack Pointer) adressieren einen bestimmten Bereich des Hauptspeichers, der für schnelle Zwischenspeicherung eingesetzt wird.
- Der **Befehlszähler** enthält die Adresse des nächsten Befehls. Er kann zur Programmsteuerung verändert werden. Der Befehlszähler entspricht etwa der GOTO-Anweisung.
- Die **Akkumulatoren** werden am häufigsten für mathematische Abläufe eingesetzt.
- Das **Conditions-Code-Register** enthält eine Reihe Flags, die den Zustand des Prozessors anzeigen (zum Beispiel ob das Ergebnis des letzten Vorgangs Null ist). Die Flags können für Verzweigungen oder Schleifen eingesetzt werden und geben der Assemblersprache damit ein Gegenstück zur IF...THEN-Struktur.

Der 6809 besitzt all diese Register. Da er eine Weiterentwicklung des 6800 von Motorola ist (wie auch der 6502), bestehen viele Ähnlichkeiten zwischen den Assemblersprachen beider Prozessoren. Die beiden Versionen sind jedoch nicht kompatibel. Code, der für den einen Prozessor geschrieben wurde, funktioniert nicht auf dem anderen. Viele 6800-Programme sind jedoch im Quellentext kompatibel, das heißt, ein für den 6800 geschriebenes Assemblerprogramm, das für den 6809 assembliert wird, kann durchaus funktionieren. Diese Art der Kompatibilität gibt es nicht bei dem 6502. Durch die Ähnlichkeit der Prozessoren kann jedoch jeder, der mit dem 6502 vertraut ist, leicht dessen Assemblerprogramme in eine entsprechende 6809-Version für folgende Register umsetzen:

- Zwei Acht-Bit-Akkumulatoren mit den Bezeichnungen A und B. Da es als Acht-Bit-Register keine wesentliche Unterschiede zwischen ihnen gibt, können sie beide als Akkumulator eingesetzt werden. Dies bringt den Vorteil, daß die Werte des einen Akkumulators erhalten bleiben, während der andere arbeitet. Beide Acht-Bit-Akkumulatoren lassen sich jedoch auch zusammen als ein 16-Bit-Akkumulator direkt für die Ausführung von 16-Bit-Arithmetik einsetzen. Wegen dieser Eigenschaft wird der 6809 oft als Pseudo-16-Bit-Prozessor bezeichnet. Diese Bezeichnung ist jedoch nicht korrekt. Es ist besser, ihn als hochentwickelten Acht-Bit-Prozessor anzusehen. Die Akkumulatoren werden beim gemeinsamen Einsatz als das 16-Bit-Register D bezeichnet.
- Auch zwischen den beiden Indexregistern X und Y gibt es keine wesentlichen Funktionsunterschiede. Da jedoch einige Befehle, die das Y-Register ansprechen, die Länge von zwei Bytes haben (im Gegensatz zu den entsprechenden Ein-Byte-Instruktionen des X-Registers), sollte, wenn möglich, nur das X-Register eingesetzt werden, da die Ausführzeit auf diese Art kürzer ist.
- Der 6809 hat zwei Stack-Pointer, die mit S und U bezeichnet werden. Alle Stack-Vorgänge des Prozessors laufen über S. Obwohl man auch S ansprechen kann, muß sicherge-




**Assemblerprogramme haben ein anderes Format als BASIC-Programme. So sehen die von einem Assembler (ein Programm, das die Assemblersprache in den Maschinencode übersetzt) gedruckten Listen erst einmal verwirrend aus. Schlüssel zum Verständnis sind die Spalten (oder Felder). Oft sind nur die Spalten für Label, Op-Code und Operand wichtig, während der Rest ignoriert werden kann.**

stellt sein, daß dadurch die Prozessorabläufe nicht gestört werden. Es ist daher sicherer, nur mit U zu arbeiten. Aufgrund der beiden Stack-Pointer eignet sich der 6809 ausgezeichnet für FORTH.

● Der Einsatz der Register X, Y, S und U muß nicht auf ihre vorbestimmte Rolle beschränkt bleiben. So lassen sich S und U auch als Indexregister einsetzen, während alle Register 16-Bit-Zahlen bearbeiten können.

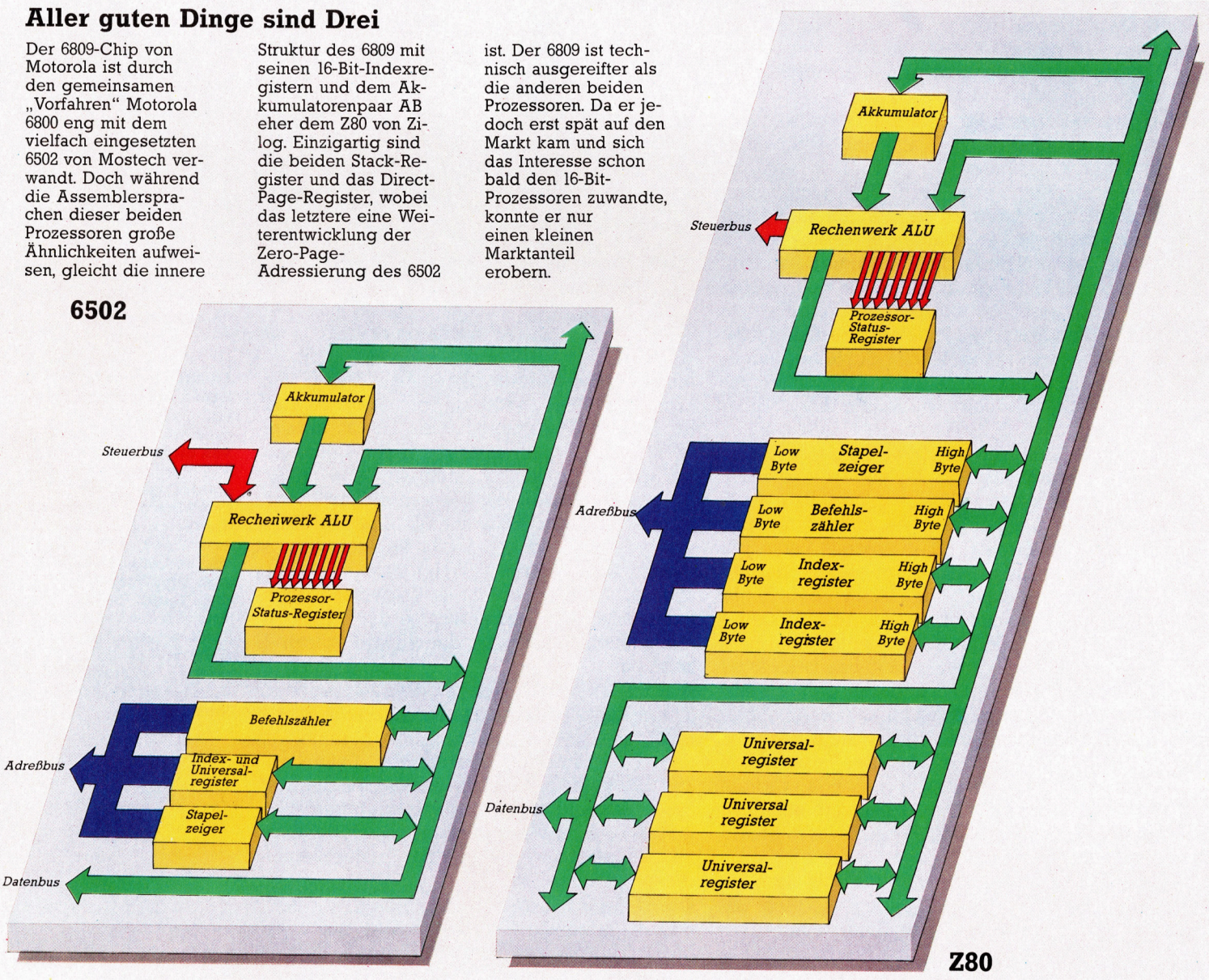
● Der Befehlszähler (PC) ist ein 16-Bit-Register, das vom Prozessor automatisch auf den nächsten Befehl gestellt wird. Die Befehle für Sprünge (JMP) und Verzweigungen (BRA) verändern den Inhalt des Befehlszählers, der auf dem 6809 auch als eine Art Indexregister verwandt werden kann. Wir werden uns später im Kurs noch ausführlich mit dieser Möglichkeit beschäftigen, da durch den Einsatz von Adressen relativ zum Befehlszähler ein Code geschrieben werden kann, der nicht von absoluten Adressen abhängig ist. Mit anderen Worten: Entsprechend programmierte Module können in einem beliebigen Bereich des Speichers abgelegt werden und dort ohne weiteres funktionieren.

● Das Condition-Code-Register (CC) hat acht Bits, die unabhängig voneinander als Flags eingesetzt werden und den Status des Prozessors anzeigen.

Einige Befehle bewegen nur Daten von einem Register zum anderen. Für ein Beispiel wollen wir mit den Assembleranweisungen FCB und FDB eine Anzahl Speicherstellen reservieren. Wenn der Assembler bei einer Übersetzung des Quellentextes in den Maschinencode auf diese Anweisungen trifft, wird dadurch der entsprechende Speicherbereich für das Programm reserviert. FCB und FDB sind keine Prozessorbefehle, und da sie den Op-Codes der Assemblersprache ähnlich sind, werden sie oft Pseudo-Ops genannt. Die Bezeichnung „Assembleranweisungen" bezieht sich jedoch deutlicher auf ihre Funktion,

## Aller guten Dinge sind Drei

Der 6809-Chip von Motorola ist durch den gemeinsamen „Vorfahren" Motorola 6800 eng mit dem vielfach eingesetzten 6502 von Mostech verwandt. Doch während die Assemblersprachen dieser beiden Prozessoren große Ähnlichkeiten aufweisen, gleicht die innere Struktur des 6809 mit seinen 16-Bit-Indexregistern und dem Akkumulatorenpaar AB eher dem Z80 von Zilog. Einzigartig sind die beiden Stack-Register und das Direct-Page-Register, wobei das letztere eine Weiterentwicklung der Zero-Page-Adressierung des 6502 ist. Der 6809 ist technisch ausgereifter als die anderen beiden Prozessoren. Da er jedoch erst spät auf den Markt kam und sich das Interesse schon bald den 16-Bit-Prozessoren zuwandte, konnte er nur einen kleinen Marktanteil erobern.

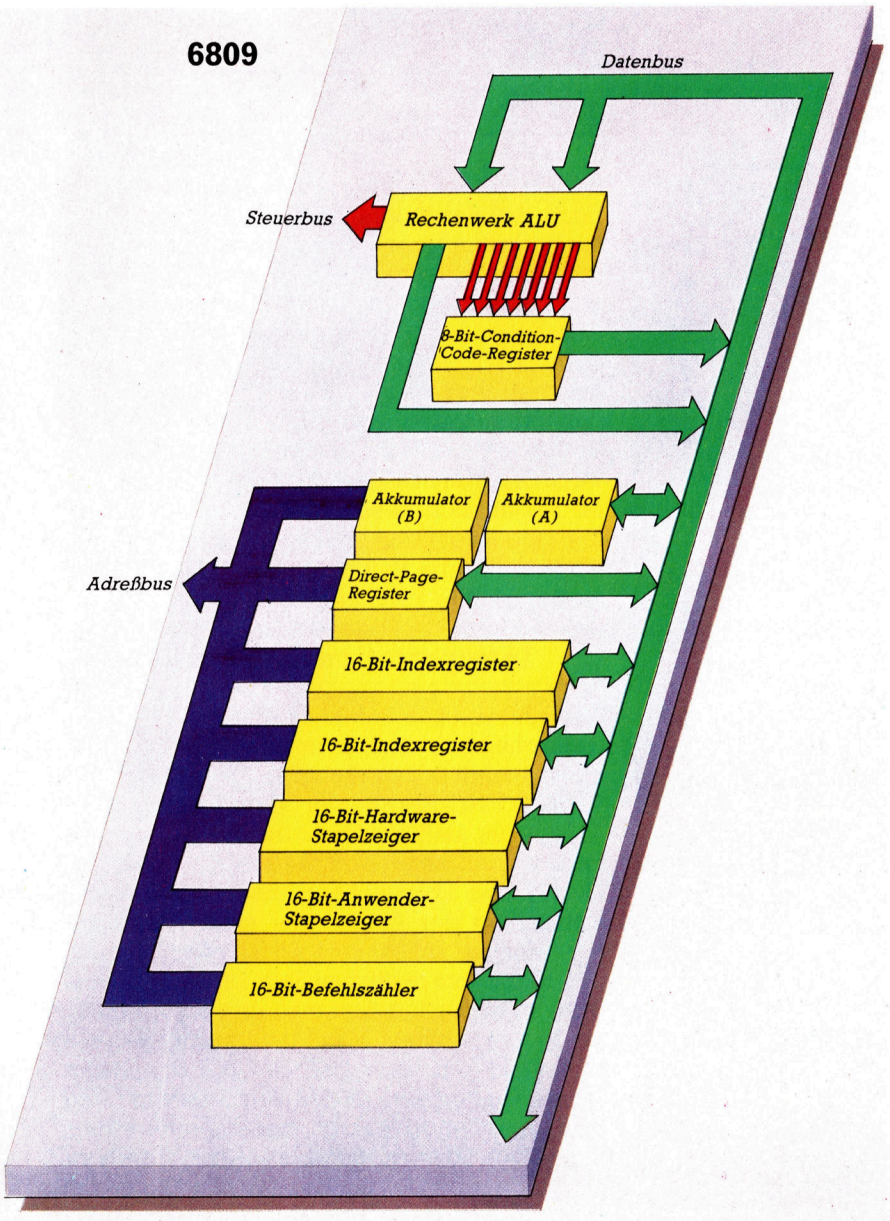


die Steuerung des Assemblers. Wenn die anweisungen mit Namen (Labels) versehen werden, übersetzt der Assembler diese Labels in die entsprechenden Adressen.

```
NUM1 FCB 0      reserviert ein Byte mit dem
                Namen NUM1 und setzt es
                auf den Anfangswert O.
NUM2 FCB 0      gleicher Vorgang wie oben.
NUM3 FCB #A93B  reserviert zwei Bytes für
                die 16-Bit-Zahl #A93B
                (# zeigt in 6809-Assem-
                blern im allgemeinen das
                Hexadezimalformat einer
                Zahl an).
```

Die folgenden Befehle laden die in diesen Adressen gespeicherten Werte in die entsprechenden Register:

```
LDA NUM1    lädt eine Acht-Bit-Zahl aus
            NUM1 in den Akkumulator A.
LDB NUM2    lädt wie zuvor NUM2 in den
            Akkumulator B.
```




```
LDX NUM3
LDY NUM3    Diese Befehle laden die in
LDS NUM3    NUM3 gespeicherte 16-Bit-Zahl
LDU NUM3    entsprechend in die Register X,
LDD NUM3    Y, S, U und D.
```

Auf die gleiche Weise kann der Acht- oder 16-Bit-Inhalt eines Registers wieder im Speicher abgelegt werden:

```
STA NUM1
STB NUM2
STX NUM3
STY NUM3
STS NUM3
STU NUM3
STD NUM3
```

Beachten Sie, daß der Wert von NUM1 beim Laden des Akkumulators nur kopiert, aber nicht verändert wird. Die Speichervorgänge funktionieren auf die gleiche Weise.

Die Inhalte zweier Register lassen sich mit dem Befehl EXG gegeneinander austauschen (vorausgesetzt, ihre Größe ist identisch):

```
EXG A,B vertausche den Inhalt der
        Register A und B
EXG X,S vertausche den Inhalt der
        Register X und S
```

Der Inhalt eines Registers kann auch in ein anderes übertragen werden: TFR Y,U kopiert den Inhalt von Y nach U. Auch hier müssen beide Register die gleiche Länge haben – acht Bit oder 16 Bit.

Nehmen wir für ein Beispielprogramm den Befehl ADD, der den Inhalt einer Speicherstelle auf den Inhalt eines der Akkumulatoren addiert. Er hat folgendes Format:

```
ADDA  NUM1  bedeutet "addiere den Inhalt
            der Speicherstelle NUM1 zum
            A-Register, so daß das A-Re-
            gister das Ergebnis enthält".
```

Wir werden zunächst die beiden in NUM1 und NUM2 abgelegten Acht-Bit-Zahlen addieren, das Ergebnis in NUM1 speichern und den Überlauf ignorieren, wenn das Ergebnis von einer Acht-Bit-Zahl nicht dargestellt werden kann. Danach addieren wir das Ergebnis dieser beiden Speicherstellen nochmals, speichern das 16-Bit-Ergebnis aber in NUM3.

Erstes Beispiel:

```
LDA NUM1    erste Zahl nach A kopieren
ADDA NUM2   zweite Zahl addieren
STA NUM1    Ergebnis in NUM1 speichern
```

Zweites Beispiel:

```
LDB NUM1    erste Zahl nach B kopieren
SEX         die Acht-Bit-Zahl in B in eine
            16-Bit-Zahl in D umwandeln
STD NUM3    D nach NUM3 kopieren
LDB NUM2    die zweite Zahl nach B
            kopieren
SEX         in eine 16-Bit-Zahl in D
            wandeln
ADDD NUM3   die erste 16-Bit-Zahl von
            NUM3 nach D addieren
STD NUM3    das Ergebnis in NUM3
            speichern
```

# Befehlseingabe

**Wir zeigen, wie das Abenteuer-Programm Befehle annimmt.**

Abenteuer sind zumeist so aufgebaut, daß der Spieler von Ort zu Ort gehen kann und dabei Objekte aufnimmt oder ablegt. Dies wird durch eine bestimmte Anzahl an Befehlen ermöglicht (im Kasten rechts zu sehen).

Dabei gibt es Variationen, wie zum Beispiel MOVE anstelle von GO oder GET anstelle von TAKE. Doch das Herausfinden der Wörter, die das Programm versteht, gehört mit zum Spielreiz. Nehmen wir an, der Spieler versucht an einem Fluß den Befehl SWIM. Antwortet das Programm „You can't swim here", so läßt sich daraus schließen, daß es an einem anderen Ort möglich ist.

Die Anzahl an Befehlen, die ein Programm versteht, ist abhängig vom Umfang des Spiels und dem Aufwand, den der Programmierer betrieben hat, um jegliche eventuelle Möglichkeit zu berücksichtigen. Das wichtigste ist jedoch, daß das Programm nicht „abstürzt", wenn ein unvorhergesehener Befehl eingegeben wird. Eine entsprechende Routine, die "I don't understand" ausgibt, ist bereits ausreichend. Allerdings ist es frustrierend, wenn ein Programm beispielsweise den Befehl TAKE LAMP versteht, jedoch bei TAKE THE LAMP mit „I don't understand" antwortet. Doch mit diesem Thema befassen wir uns später ausführlicher, wenn das Spiel ausgereift gestaltet werden soll. Im Augenblick beschäftigen wir uns nur mit der Befehlseingabe und der Entwicklung einer Routine, die die Befehle in eine einfach zu interpretierende Form umsetzt.

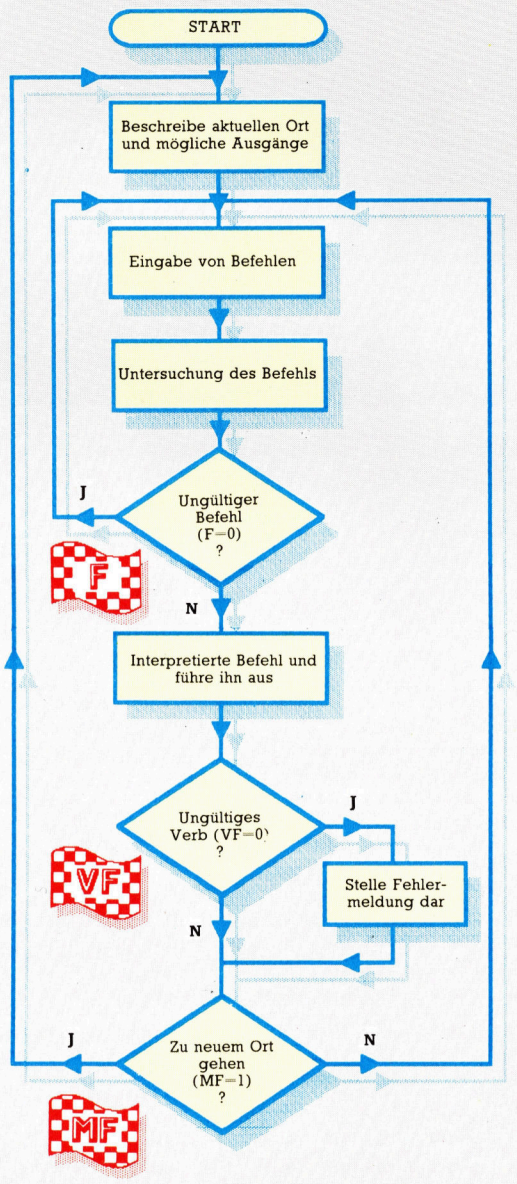
| | |
|---|---|
| **GO (Richtung)** | **Zum Bewegen zwischen Orten** |
| **TAKE (Objekt)** | **Zum Aufnehmen von Objekten** |
| **DROP (Objekt)** | **Zum Ablegen von Objekten** |
| **LIST** | **Zum Auflisten der Objekte** |
| **LOOK** | **Zum Darstellen der aktuellen Ortsbeschreibung** |
| **END** | **Zum Beenden des Spiels** |

## Interpretation der Befehle

Egal um welchen Befehl es sich handelt, ist es wahrscheinlich, daß er in Imperativ-Form formuliert wird – zum Beispiel GO SOUTH TOWARDS THE RIVER. Der Vorteil dieser Satzstruktur ist, daß sie leicht aufzuteilen ist: Das Verb ist das erste Wort, dann folgt direkt das Objekt und abschließend eine genauere Beschreibung der Aktion. Der erste Schritt bei der Analyse eines Befehls ist die Trennung des Verbs vom restlichen Satz. Dies kann leicht erreicht werden, indem man den Satz Zeichen für Zeichen mittels MID$ untersucht, bis eine Leerstelle gefunden wird. Der Teil des Satzes links der Leerstelle ist das Verb und kann VB$ zugeordnet werden. Den restlichen Teil kann man einer zweiten Variablen, NN$, zuweisen. Die folgende Routine wird in dem Programm Haunted Forest verwendet:

## Flags

Flags werden in Programmen mit modularem Aufbau sehr oft verwendet. Bedingungen, die eine Verzweigung des Programmlaufes erfordern, können innerhalb des entsprechenden Moduls ausgeführt werden. Eine Verzweigung aufgrund des Testergebnisses kann jedoch bis zum Rücksprung in die Hauptroutine verzögert werden. Durch Setzen einer Variablen auf einen bestimmten Wert kann dieser später in der Hauptroutine abgefragt werden. Derartige Variablen werden als Flags bezeichnet. Das Flußdiagramm zeigt die bis jetzt entwickelte Hauptprogrammschleife von Haunted Forest. Das Flag F, das innerhalb der Routine zum Interpretieren von Befehlen gesetzt wird, zeigt an, ob ein Befehl gültig ist oder nicht. Die Unterroutine, die zum Interpretieren und Ausführen normaler Befehle dient, verwendet zwei Flags: VF signalisiert, daß das Verb des Befehls erkannt wurde. Wenn bei der Ausführung eines Befehls der Spieler zu einem neuen Ort geht, wird diese Bewegung durch das Flag MF gekennzeichnet. Wenn MF in der Hauptroutine überprüft wird, zeigt ein Wert von 1 an, daß die Routine in die Unterroutine zur Beschreibung des neuen Ortes verzweigen soll.



```
2500 REM **** SPLIT COMMAND S/R ****
2510 IF IS$="LIST" OR IS$="END" THEN VB$=IS$:F=1:RETURN
2515 IF IS$="LOOK" THEN VB$=IS$:F=1:RETURN
2520 F=0
2530 LS=LEN(IS$)
2540 FOR C=1 TO LS
2550 A$=MID$(IS$,C,1)
2560 IF A$<>" " THEN 2590
2570 VB$=LEFT$(IS$,C-1):F=1
2580 NN$=RIGHT$(IS$,LS-C):C=LS
2590 NEXT C
2600 :
2610 IF F=1 THEN RETURN
2620 PRINT:PRINT"I NEED AT LEAST TWO WORDS"
2630 RETURN
```

Bevor die Routine versucht, den Satz aufzuteilen, wird überprüft, ob es sich bei dem Befehl nicht um einen der drei aus einem Wort bestehenden Befehle handelt – LIST, LOOK oder END. Ist es ein derartiger Befehl, wird die Anweisung direkt in VB$ übertragen und die Routine beendet. Im anderen Fall wird eine FOR . . .NEXT-Schleife gestartet, bis eine Leerstelle gefunden wird. Um zu vermeiden, daß man eine FOR . . .NEXT-Schleife abbricht, ohne die NEXT-Anweisung zu durchlaufen, wird nach dem Auffinden einer Leerstelle ein Flag F auf 1 gesetzt. Auch noch den Rest des Satzes zu untersuchen, wäre jedoch Zeitverschwendung.

An dieser Stelle kann die Schleife abgebrochen werden, indem der Schleifenzähler C auf den Endwert LC gesetzt wird. Entsprechend erfolgt beim nächsten NEXT kein Rücksprung zum FOR. Nach Verlassen der Schleife kann nun der Inhalt von F überprüft werden. Ist der Wert 1, so besteht der Befehl aus mehreren Wörtern, und das Programm kann zur Hauptschleife verzweigt werden. Ist der Wert ungleich 1, dann besteht der Befehl aus nur einem Wort. Gleichzeitig bedeutet das jedoch auch, daß es sich bei diesem Befehl nicht um die Anweisungen LIST, LOOK oder END handelt, da die Eingabe bereits daraufhin überprüft wurde. In diesem Fall wird eine Meldung ausgegeben, daß für einen Befehl zwei Wörter eingegeben werden müssen.

Für den größten Teil des Programms gilt, daß der Spieler von Ort zu Ort geht und Objekte aufnimmt oder ablegt. Zu diesem Zweck sind die Befehle GO, TAKE, DROP, LIST, LOOK und END (sowie deren Varianten) völlig ausreichend. Nur an bestimmten Orten muß dem Spieler die Eingabe speziellerer Befehle ermöglicht werden. Normalerweise wäre es natürlich unsinnig, den Befehl KILL zu verwenden, wenn es nichts zu töten gibt. Deshalb kann man eine Programmstruktur entwickeln, in der nur die oben angegebenen sechs Befehle überprüft werden. Kommt der Spieler an einen neuen Ort, überprüft das Programm, ob es sich um einen speziellen Ort handelt. Ist dies der Fall, werden neue Befehle in einer zusätzlichen Unterroutine für diesen Ort behandelt. Die Hauptroutine sollte also nur folgende Aufgaben erfüllen:

1) Orte und Ausgänge beschreiben.
2) Untersuchen, ob es sich hierbei um einen speziellen Ort handelt.
3) Nach einem Befehl fragen und, wenn der Ort nicht „speziell" ist, den normalen Befehl interpretieren.

Außerdem muß die Hauptroutine zwischen einem Befehl, der eine Bewegung zu einem neuen Ort bewirkt, und einem normalen Befehl unterscheiden. Im ersten Fall muß zum Anfang der Schleife verzweigt werden, um den Ort zu beschreiben und festzustellen, ob er „speziell" ist. Im zweiten Fall muß nur nach einem neuen Befehl gefragt werden. Das Problem läßt sich mit einem „Bewegungsflag" MF lösen, das normalerweise auf 0 gesetzt ist. Bewirkt ein Befehl eine Bewegung, wird es auf 1 gesetzt. Der Status des Flags wird am Ende der Hauptschleife überprüft und die entsprechende Programmverzweigung ausgeführt. Fügen Sie die folgenden Zeilen in Haunted Forest ein:

```
270 GOSUB2500:REM SPLIT INSTRUCTION
275 IF F=0 THEN 260:REM INVALID INSTRUCTION.
280 GOSUB3000:REM NORMAL COMMANDS
290 IF VF=0 THENPRINT:PRINT"I DONT UNDERSTAND"
300 IF MF=1 THEN 240:REM NEW LOCATION
310 IF MF=0 THEN 260:REM NEW INSTRUCTION

3000 REM **** NORMAL COMMANDS S/R ****
3010 VF=0:REM VERB FLAG
3020 IF VB$="GO" OR VB$="MOVE" THENVF=1:GOSUB3500
3030 IF VB$="TAKE" OR VB$="PICK"THEN VF=1:GOSUB3700
3040 IF VB$="DROP" OR VB$="PUT"THEN VF=1:GOSUB3900
3050 IF VB$="LIST" OR VB$="INVENTORY"THEN VF=1:GOS
     UB4100
3055 IF VB$="LOOK" THEN VF=1:MF=1:RETURN
3060 IF VB$="END" OR VB$="FINISH" THEN VF=1:GOSUB4
     170
3070 RETURN
```

In der ersten Routine wird ein Flag VF verwendet, um anzuzeigen, ob das Verb verstanden wurde oder nicht. Nur wenn es isoliert werden konnte, wird VF auf 1 gesetzt. Durch Testen von VF kann die Ausgabe der Meldung „I don't understand" veranlaßt werden. Hat VF noch den Wert Null, wurde der Befehl nicht verstanden, und die Meldung wird ausgegeben.

## Beschreibung der Orte

Im nächsten Teil des Kurses werden wir uns mit Routinen zum Aufnehmen, Ablegen und Auflisten von Objekten befassen. Im Moment fügen wir hier nur eine kurze END-Befehl-Routine ein:

```
4170 REM **** END GAME S/R ****
4180 PRINT:PRINT"ARE YOU SURE (Y/N) ?"
4190 GET A$:IF A$<>"Y" AND A$<>"N" THEN 4190
4200 IF A$="N" THEN RETURN
4210 END
```

Die Handhabung des LOOK-Befehls ist sehr einfach. Um den aktuellen Ort noch einmal zu beschreiben, braucht nur das „Bewegungs"-Flag MF auf 1 gesetzt und zur Hauptschleife verzweigt zu werden. Dadurch werden die Routinen zur Beschreibung des Ortes und der Ausgänge aufgerufen. Da der Wert der Ortsvariablen P dadurch nicht verändert wird, wird derselbe Ort beschrieben. Der LOOK-Befehl ist sehr nützlich, wenn durch verschiedene Aktionen die erste Beschreibung vom Bildschirm

verschwunden ist. Durch erneute Eingabe des Befehls lassen sich die entsprechenden Details wieder aufrufen.

Bei der Befehlseingabe gibt es meistens verschiedene Ausdrucksformen. So besagen zum Beispiel GO NORTH, MOVE NORTH und GO TOWARDS THE NORTH dasselbe. Obwohl nicht unbedingt alle Formen vom Programm verstanden werden müssen, wird das Spielen interessanter, wenn der Spieler mehrere Möglichkeiten hat, um den gewünschten Befehl einzugeben.

## Befehlsstruktur

Die drei eben genannten Bewegungsbefehle haben eine gemeinsame Struktur: Sie beginnen alle mit einem Bewegungsverb und enthalten die Richtung. Somit ist es möglich, eine Routine zu entwerfen, die den Satz nach dem Verb und nach der Richtung durchsucht. Dabei werden die Leerstellen des Satzes gesucht und jedes gefundene Wort mit den vier möglichen Richtungen verglichen, bis eine Übereinstimmung gefunden wird.

## Digitaya-Listing

```
1220 GOSUB1700:REM ANALYSE INSTRUCTIONS
1225 IF F=0 THEN 1210:REM INVALID INSTRUCTION
1230 GOSUB 1900:REM NORMAL INSTRUCTIONS
1240 IF VF=0 THENPRINT"I DON'T UNDERSTAND"
1250 IF MF=1 THEN 1160:REM NEW POSITION
1260 IF MF=0 THEN 1210:REM NEW INSTRUCTION

1700 REM **** ANALYSE INSTRUCTION S/R ****
1705 F=0:REM ZERO FLAG
1710 IFIS$="END" OR IS$="LIST" THEN VB$=IS$:F=1:
     RETURN
1720 IF IS$="LOOK" THEN VB$=IS$:F=1:RETURN
1730 :
1740 REM ** SPLIT INSTRUCTION **
1750 VB$="":NN$="":REM ZERO VERB AND NOUN
1770 LS=LEN(IS$)
1780 FOR C=1TO LS
1790 A$=MID$(IS$,C,1)
1800 IF A$=" "THEN VB$=LEFT$(IS$,C-1):NN$=RIGHT$(I
     S$,LS-C):F=1:C=LS
1810 NEXT
1830 IF F=0 THEN PRINT:PRINT"I NEED AT LEAST TWO
     WORDS"
1840 RETURN
1850 :
1900 REM **** NORMAL ACTIONS S/R ****
1910 VF=0
1920 PRINT
1930 IF VB$="GO"ORVB$="MOVE"THENVF=1:GOSUB2000
1940 IF VB$="TAKE"ORVB$="PICK"THEN VF=1:GOSUB2140
1950 IF VB$="DROP"ORVB$="PUT"THENVF=1:GOSUB2360
1960 IF VB$="LIST"ORVB$="INVENTORY"THENVF=1:
     GOSUB2540
1965 IF VB$="LOOK" THEN VF=1:MF=1:RETURN
1970 IF VB$="END"ORVB$="FINISH"THENVF=1:GOSUB2610
1980 RETURN

2015 GOSUB8600:REM SEARCH FOR DIRECTION

2610 REM **** END GAME S/R ****
2620 PRINT:PRINT"ARE YOU SURE (Y/N) ?"
2630 GETA$:IFA$<>"Y"AND A$<>"N"THEN2630
2640 IFA$="N"THEN RETURN
2650 END

8600 REM **** SEARCH FOR DIRECTION S/R ****
8610 NN$=NN$+" ":LN=LEN(NN$):C=1
8620 FORI=1 TO LN
8630 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
8640 W$=MID$(NN$,C,I-C):C=I+1
8650 IF W$="NORTH" OR W$="EAST" THEN NN$=W$:I=LN
8660 IF W$="SOUTH" OR W$="WEST" THEN NN$=W$:I=LN
8670 NEXT I
9680 RETURN
```

```
3630 REM **** SEARCH FOR DIRECTION S/R ****
3640 NN$=NN$+" ":LN=LEN(NN$):C=1
3645 FOR I=1 TO LN
3650 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
3655 W$=MID$(NN$,C,I-C):C=I+1
3660 IF W$="NORTH" OR W$="EAST" THEN NN$=W$:I=LN
3665 IF W$="SOUTH" OR W$="WEST" THEN NN$=W$:I=LN
3670 NEXT I
3675 RETURN
```

Im letzten Teil unseres Projekts haben wir eine Bewegungs-Routine entwickelt. Um nun diese neue Routine zur Bewegungs-Routine hinzuzufügen, muß man nur die folgende kurze Zeile eingeben:

```
3505 GOSUB3630:REM SEARCH FOR DIRECTION
```

Trotzdem kann diese Routine nicht jeden Befehl interpretieren – beispielsweise GO IN A NORTHERLY DIRECTION –, wenn das Richtungswort nicht isoliert werden kann. Selbstverständlich wäre es denkbar, eine entsprechende Routine zu entwickeln, die auch diese Eingabe noch versteht, doch würde sie eine sehr lange Verarbeitungszeit benötigen. Andererseits versteht unser Programm sogar GO NORTHWARDS, wenn von der Bewegungsroutine nur der erste Buchstabe des zweiten Teils der Eingabe verwendet wird. In diesem Fall wird das N von NORTHWARDS als N für NORTH akzeptiert.

## Basic-Dialekte

**Spectrum:**

Verwenden Sie in beiden Programmen I$ anstelle von IS$, B$ anstelle von VB$, und R$ anstelle von NN$.

Ersetzen Sie bei Digitaya die folgenden Programmzeilen:

```
1790 LET A$=I$(C TO C)
1800 IF A$=" " THEN LET B$=I$(TO C-1):LET R$=I$
     (LEN(I$)-LS+C+1 TO):LET F=1:LET C=LS
2630 LET A$=INKEY$:IF A$<>"Y"
     AND A$<>"N" THEN 2630
8630 IF R$(I TO I)<>" " THEN NEXT I:RETURN
8640 LET W$=R$(C TO I-1):LET C=I+1
```

Ersetzen Sie die folgenden Programmzeilen bei Haunted Forest:

```
2550 LET A$=I$(C TO C)
2570 LET B$=I$(TO C-1):LET F=1
2580 LET R$=I$(LEN(I$)-LS+C+1 TO):LET C=LS
3650 IF R$(I TO I)<>" " THEN NEXT I:RETURN
3655 LET W$=R$(C TO I-1):LET C=I+1
4190 LET A$=INKEY$:IF A$<>"Y" AND
     A$<>"N" THEN GOTO 4190
```

**Acorn B:**

Ersetzen Sie bei Digitaya die folgende Zeile:

```
2630 REPEAT:A$=GET$:UNTIL A$="Y" OR
     A$="N"
```

sowie diese Zeile bei Haunted Forest:

```
4190 REPEAT:A$=GET$:UNTIL A$="Y" OR
     A$="N"
```

# Fachwörter von A bis Z

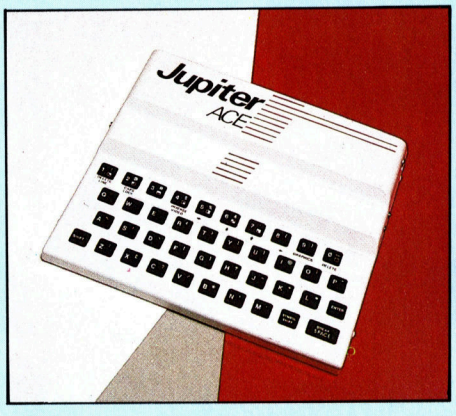
**Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter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

**FORTH = FORTH**
Die Sprache FORTH wurde 1972 von dem Astronomen Charles Moore entwickelt, nachdem er feststellen mußte, daß FORTRAN für seine Teleskop-Steuerprogramme nicht ausreichend war. Die Schwierigkeiten rührten daher, daß Struktur und Verarbeitungsformen bei FORTRAN zu sehr an der mathematischen Lösung wissenschaftlicher Probleme orientiert sind. FORTH bedient sich eines Lexikons von „Primitives" (Stammwörtern) für die elementaren Funktionen und eines Editor/Compiler/Interpreter-Systems.

Der Editor ermöglicht die Definition neuer (unterprogrammähnlicher) Funktionen anhand des vorhandenen Wortschatzes. Die Namen werden dem Lexikon als neue „Wörter" zugefügt. Die Funktionen werden compiliert und gespeichert und sind über den Interpreter jederzeit durch Eingabe des betreffenden Wortes ausführbar. Den RAM-Bereich behandelt FORTH als einen einzigen großen LIFO(Last In First Out)-Stack – den Programmspeicher dagegen als Gebilde aus unabhängigen Abschnitten. Jede Funktion verfügt über eine eigene Unterprogrammadresse. Das Programm springt zur Ausführung auf diese Adresse, greift alle nötigen Parameter vom LIFO-Stack, stapelt die Resultate dorthin zurück und geht zu der nächsten Adresse über.

Beim Programmieren in FORTH kann man für jede Anwendung einen maßgeschneiderten Befehlssatz entwickeln. Die Hauptvorteile sind die Anpassungsfähigkeit und die Geschwindigkeit der Sprache. Der in der Abbildung gezeigte Jupiter Ace wurde mit FORTH als systemeigener Sprache ausgeliefert.

**FORTRAN = FORTRAN**
FORTRAN (FORmula TRANslator = Formelübersetzer) wurde 1956 als erste kommerzielle Hochsprache bei IBM entwickelt. IBM bewies damit, daß auch ein dem gewohnten Englisch nahestehender Programmiercode schnell und effizient compilierbar sein kann. Die Sprache machte




den Rechnereinsatz vor allem für Ingenieure und andere Wissenschaftler interessant. Sie waren eher bereit, etwas dem vertrauten mathematischen Formalismus Entsprechendes zu erlernen, als die Zeit und Geduld für das Programmieren im Maschinencode aufzubringen. FORTRAN wurde ein durchschlagender Erfolg und ist auch heute noch die verbreitetste Hochsprache.

Als große Errungenschaft galt anfangs die Möglichkeit, Bibliotheken mit unabhängig compilierten FORTRAN-Unterprogrammen aufzubauen. Alle Großrechner verfügen über derartige Programmbibliotheken. Wegen ihrer Bedeutung als „Quellen" wurden andere Sprachen – zum Beispiel PASCAL – so ausgelegt, daß sie auf FORTRAN-Bibliotheksprogramme zugreifen können.

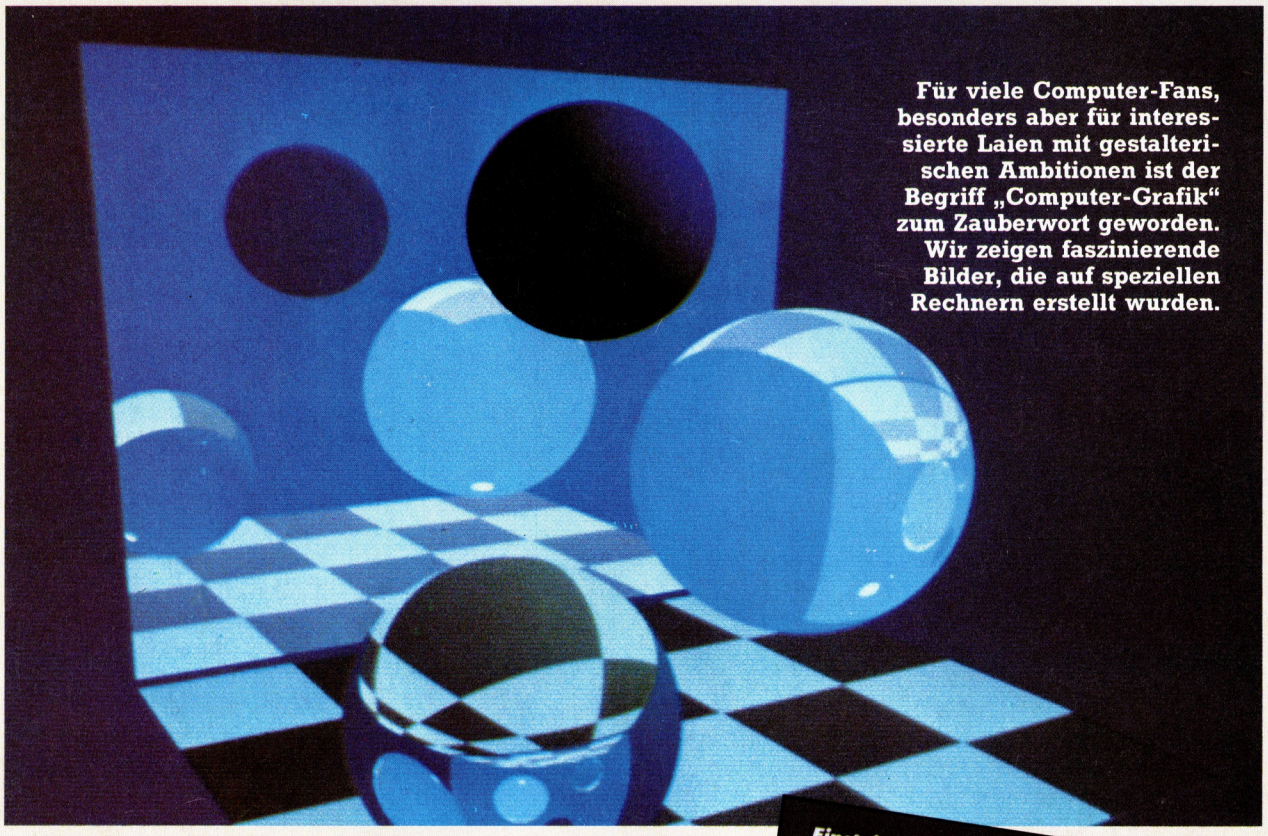
FORTRAN hat eine stolze Zahl von Abkömmlingen, wie ALGOL, PASCAL und BASIC. Das historische Gewicht ist jedoch darin zu sehen, daß FORTRAN den Computer aus den Rechnerlabors der Universitäten herausholte und ihn in Ausbildungsstätten und Fabrikhallen zu einem selbstverständlichen Werkzeug von Wissenschaft und Technik werden ließ. Der Büro- und der Heimcomputer lagen dann nicht mehr fern: FORTRAN machte den Rechner auch dem Nicht-Fachmann zugänglich und leitete einen wichtigen Schritt auf dem Weg zu einer benutzerfreundlichen Datenverarbeitung ein.

**Fourth Generation = Vierte Generation**
In der Rechnerentwicklung umfaßt eine Generation immer etwa ein Jahrzehnt. Es beginnt jeweils mit dem Aufkommen einer aufwendigen neuen Technologie, die mit Ende dieser Zeitspanne alltäglich wird. Die Anfänge der „ersten Generation" lagen vor 1950 bei den ersten speicherprogrammierten Rechnern mit Elektronenröhren. Die „zweite Generation" begann Ende der 50er Jahre mit der diskreten Transistorlogik. Die „dritte Generation" vom Anfang der 60er Jahre (typischer Vertreter ist der IBM 360) ist durch die Verwendung integrierter Schaltungen und die Entwicklung umfänglicher Betriebssysteme gekennzeichnet. Und die „vierte Generation" wurde zu Beginn der 70er Jahre mit der Einführung der LSI- und der VLSI-Technik eingeleitet. Diese Schaltungstechnik wird heute bei Großrechnern wie bei Microcomputern angewandt. Auch die Micros haben einige Entwicklungsstadien durchgemacht, bis sie den heutigen Stand als vollwertige Rechner mit beachtlicher Speicherkapazität, integrierter Software und der Fähigkeit zu Mehrprogramm- sowie Verbundnetz-Betrieb erreichten.

**Bildnachweise**
1233, 1258: Ian McKinnell
1234, 1253, 1256, 1257: Kevin Jones
1236, 1237, 1239, 1240, 1250: Liz Dixon
1238: Wattleworth Silk McConachie, Chris Tucker
1245, 1246, 1247: Chris Stevens
1252: Kevin Jones, Liz Dixon
1254: Liz Heaney

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

# computer kurs Heft 46



**Für viele Computer-Fans, besonders aber für interessierte Laien mit gestalterischen Ambitionen ist der Begriff „Computer-Grafik“ zum Zauberwort geworden. Wir zeigen faszinierende Bilder, die auf speziellen Rechnern erstellt wurden.**

## Link 480Z

Speziell für Anwender im schulischen Bereich wurde dieses Gerät konstruiert. Leider wurde der alte Z80-Prozessor beibehalten.

## Montage

In unserem Selbstbau-Kursus ist nun die Verdrahtung der Motoren an der Reihe: Anschluß an das Userport-Interface.

## Nacht der Wölfe

Nach „Sabre Wulf“ bringt Ultimate jetzt „Knight Lore“. Für Acorn, Schneider und Spectrum.

## Der Prolog

Programming in Logic, kurz PROLOG, ist eine praktische und leicht überschaubare Computer-Sprache. Beginn einer Serie.

## Heiße Eisen

Thermo-Drucker sind kostengünstige Geräte. Wir stellen drei Printer für den Spectrum vor.