Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 56

Intelligente Maschinen

Roboter lernen Sehen

Kartei anlegen

Erweitert: Dragon 64

Debugger-Routinen

Ein wöchentliches Sammelwerk

computer kurs

Heft 56

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

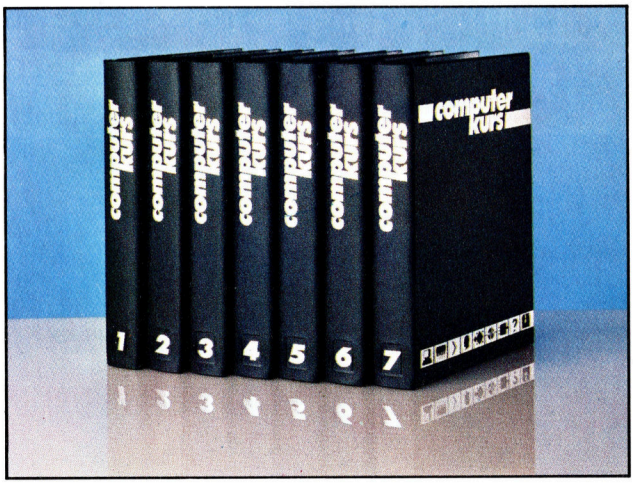
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

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

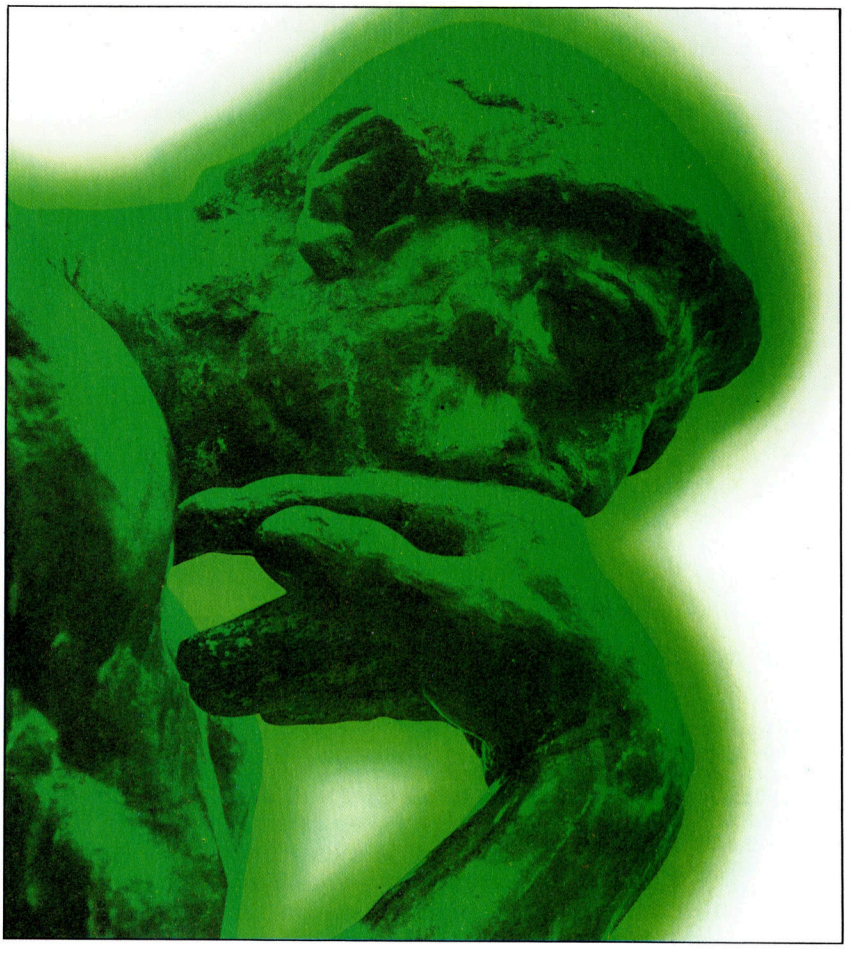
Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



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

Schöne neue Welt?

Zum Abschluß unserer Serie über KI betrachten wir die langfristigen Aussichten der Herstellung von Maschinen mit kreativer Intelligenz. Welche Auswirkung solche Maschinen auf die Evolution des Menschen haben werden, ist noch nicht klar.



Betrachten wir die Computerentwicklung als Leiter, können wir, allgemein gesagt, vier Sprossen zunehmender Komplexität unterscheiden. An der untersten Sprosse ist die Buchhaltung angesiedelt – eine allgemeine Aufgabe, die der Computer gut erfüllt. Nur einen Schritt weiter werden die Dinge bereits interessanter. Hier stehen Programme, die Menschen helfen, intelligente Entscheidungen zu treffen – Finanzplanungs- und Tabellenkalkulationssysteme.

Auf der dritten Ebene finden wir Applikationen, die aus menschlicher Erfahrung abgeleitet sind. Ein Beispiel für solche Software ist das PROSPECTOR-Expertensystem, in dem die codierten Erfahrungen von mehreren Prospektoren enthalten sind. Durch Anwendung dieser Regeln wurden die Programmautoren mit der Entdeckung eines gewaltigen, bisher unbekannten Molybdän-Vorkommens im Staate Washington belohnt. Danach wurde ein weiteres in Kanada gefunden.

Die Erfolge von PROSPECTOR sind sicher beeindruckend, auf die vierte Leitersprosse haben sie uns jedoch nicht gebracht: Die Herstellung von Maschinen mit kreativer Intelligenz steht noch aus.

Produktivität hängt von der Befolgung von Gesetzen ab, und Computer sind nun einmal „gesetzestreu". Sie aber mit Kreativität auszustatten, hat sich als extrem schwierig erwiesen. Um wirklich kreativ sein zu können, müßten Computerprogramme wiederum eigene Gesetze erstellen können.

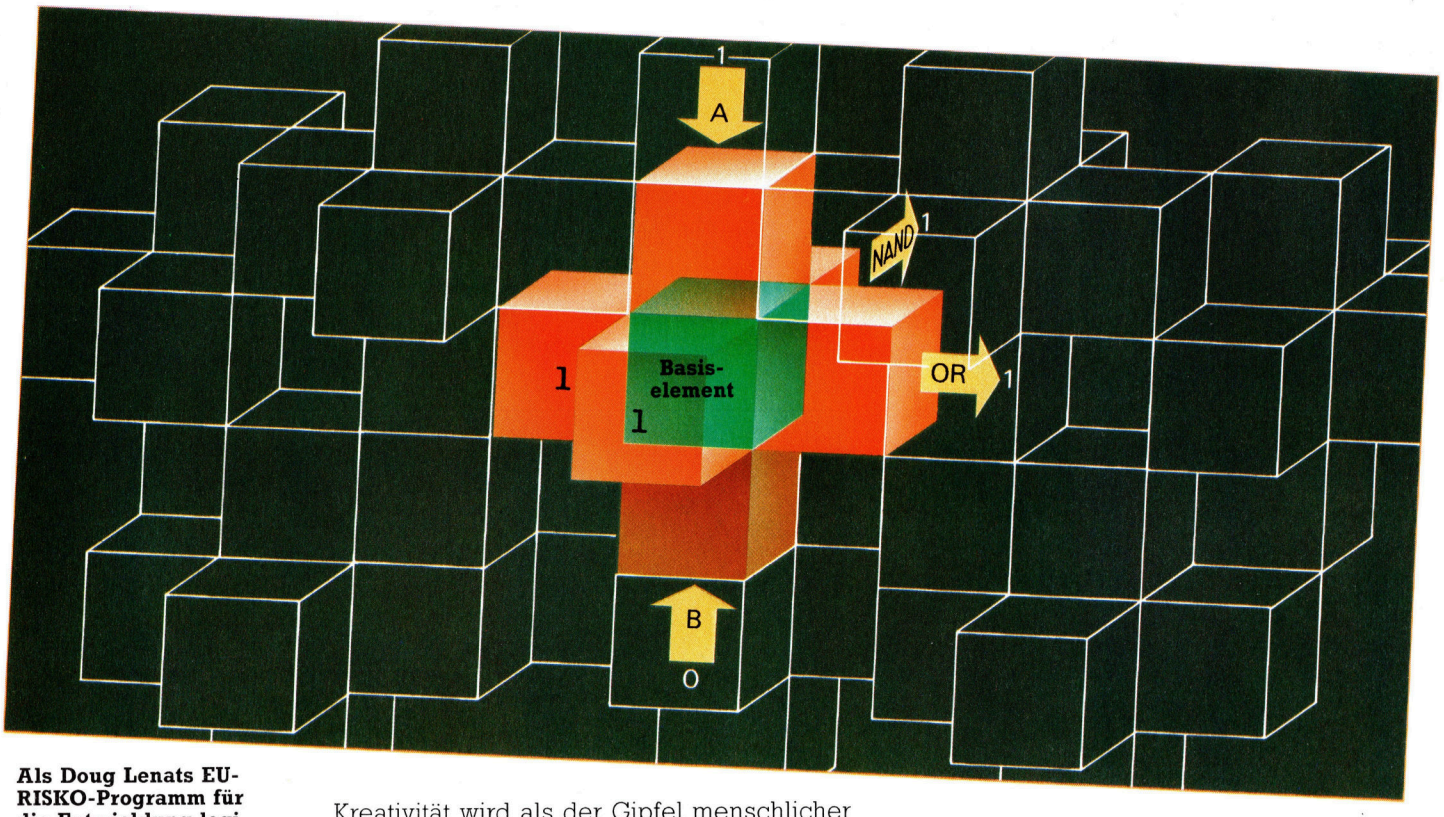
Ein Programm, das immerhin einen Weg zur vierten Sprosse weist ist EURISKO, entwickelt von Doug Lenat an der Stanford Universität. EURISKO ist ein Entwicklungsprogramm, das an anderer Stelle dieser Serie bereits behandelt wurde. Es wurde in unterschiedlichsten Bereichen eingesetzt – in der maritimen Kriegsführung wie auch in der VLSI-Platinen-Entwicklung. EURISKO enthält eine Sammlung heuristischer Regeln und Konzepte und paßt sie den jeweils gewählten Fachbereichen an. Die eigentliche Innovation aber liegt in der Tatsache, daß das Programm seine eigene Heuristik (= Erkenntnis) modifizieren kann. Damit kann das System grundlegende Regeln so adaptieren und spezialisieren, daß es in der Lage ist, neue Situationen zu bewältigen.

Genial, Herr Rechner!

EURISKO hat eine Entdeckung gemacht, die sofort als „patentwürdig" galt. Es handelt sich um eine neue Gestaltung einer 3-D Logik-Schaltung (ein NAND/OR-Gatter).

Als EURISKO für das VLSI-Design angewandt wurde, verfügte es bereits über ein heuristisches Gesetz, das aus einer vorherigen Aufgabe entwickelt worden war. Dies besagte: Wenn ein Konzept interessant ist, versuche, es zu verbessern. Bei einer 2-D-Aufgabe angewandt führte das zu einer optimierten 3-D-Version des Bauteils, das unlängst erfolgreich hergestellt wurde. Nach Lösung der Produktionsprobleme kann diese 3-D-Version dichter gebaut werden und größere Kapazität bieten.

Der Fortschritt bei der Entwicklung der Bio-Technologie könnte zu dem beängstigenden Mensch-Maschine-Hybriden führen – in dem sich die intiutiven Möglichkeiten des menschlichen Gehirns mit dem logischen und mathematischen Potential des Microprozessors vereinen. Sollte dies Wirklichkeit werden, wäre die menschliche Evolution dem Einwirken der Natur entrissen und gänzlich der Kontrolle durch Mensch-Maschinen unterworfen, die zu selektiver Selbst-Reproduktion imstande wären.



Als Doug Lenats EURISKO-Programm für die Entwicklung logischer Schaltungen angewandt wurde, erzeugte es ein völlig neues Design für ein 3-D-Logik-Gatter. EURISKO vollzog diese Neuentwicklung durch Anwendung eines heuristischen Gesetzes bei einem herkömmlichen MOS-Logik-Gatter. Neben der Möglichkeit, jedes gegebene Paar von NAND- und OR-Operationen gleichzeitig zu berechnen, ist mit dem neuen Gatter eine dichtere Bauweise möglich.

Kreativität wird als der Gipfel menschlicher Intelligenz betrachtet. EURISKO aber sollte uns zum Nachdenken anregen. Es ist ein Beispiel für eine geniale Entdeckung, die aus der Anwendung eines einzigen Gesetzes resultiert. Der kreative Computer ist näher, als die meisten Menschen glauben.

Bisher betrachteten wir KI als „Pionier" der Computerwissenschaft – als fruchtbare Quelle neuer Ideen und Tricks. Sind diese Ideen erfolgreich, werden sie in andere Bereiche der Computerwissenschaft Einzug halten. Dies war bei der Listen-Verarbeitung ebenso der Fall wie beim „sprechenden" Computer. Gegenwärtig geschieht dies bei Systemen auf Wissensbasis.

Die mittelfristigen Aussichten für diese Arbeit sind gut. Es mag Rückschläge geben, und einige Vorhersagen mögen nicht eintreffen. Am Ende dieses Jahrhunderts aber werden wir grundlegende Fortschritte gemacht haben: In den Bereichen Computer-Sehen, automatische Übersetzung, Maschinenlernen und Systeme auf Wissensbasis.

Die japanische Initiative der fünften Generation hat diesem KI-Aspekt beachtlichen Antrieb gegeben. Mit ihren weitreichenden Plänen haben die Japaner die Spielregeln der Computerindustrie behutsam neu definiert. Sie rechnen damit, in den neunziger Jahren Wissens-Informations-Prozessoren entwickeln zu können, die auf einer völlig neuen Parallelcomputer-Architektur basieren. Die Software, die derartige Systeme lauffähig macht, basiert auf der KI-Forschung.

Die Konzentration der Japaner auf KI-Techniken hat den Rest der Welt dazu veranlaßt, sich wieder mit diesem Bereich zu befassen. Regierungen haben Milliarden investiert, um in diesem Rennen nicht als Letzter durchs Ziel zu gehen und den Anschluß zu verpassen.

Es gibt etwas, das viele Menschen bei der KI als negativ betrachten. Wie beeindruckend auch die KI-Erfolge sonst sein mögen, es gibt Anwendungen, die mißfallen. Dies gilt besonders für den militärischen Bereich. Über die Hälfte der KI-Anwendungen ist kriegerischer, militärischer Art. Dazu gehören unter anderem:

- „Intelligente" Unterseeboote
- „Intelligente" Munition
- Cruise Missiles
- Zielsuchende Panzer
- Auf Wissen basierende Suchsysteme
- Zielsuchsysteme für Torpedos
- Radar-Abwehrsysteme

Weit beängstigender sind jedoch jene Bilder, die die Entwicklung der „Ultra Intelligenten Maschinen (UIM) darstellen, Maschinen mit übermenschlicher Intelligenz. Viele Menschen sehen eine von den UIMs beherrschte Zukunft. Ohne tatsächlich „denken" zu können, können diese Maschinen fast alles, was auf logischem Denken beruht, besser tun als Menschen. In mathematischen wie naturwissenschaftlichen Disziplinen werden sie uns weit übertreffen. In der Industrie, so führt die Vision weiter aus, werden sie menschliche Experten ersetzen und so die Wirtschaft kontrollieren. Zwar können sie nicht gut Tennis spielen, werden aber bessere Schachspieler als der Mensch sein. Wo ist die „Notbremse", die wir ziehen können, bevor die Maschinen uns endgültig überlegen sind?

Die Idee der Mensch-Maschine-Hybriden

gehört seit langem zu den Lieblingsthemen von Science-fiction-Autoren. Kürzlich erst gelang es, diese phantastische Idee in ein langfristiges Forschungsobjekt umzuwandeln. Grund dafür ist die Verschmelzung zweier bisher nicht als vereinbar geltender wissenschaftlicher Forschungsbereiche, nämlich genetische Konstruktion und Wissens-Konstruktion.

Genetische Konstruktion bedeutet, den in lebendem Material enthaltenen genetischen Code so zu „programmieren", daß Folgegenerationen genetische Verbesserungen erfahren. Genetische Ingenieure haben bereits Techniken perfektioniert, die es ihnen erlauben, wünschenswerte Charakteristiken auf Microorganismen zu übertragen, um so Drogen zu erzeugen, die zuvor entweder zu teuer waren oder aber in großen Mengen nicht erzeugt werden konnten. Man geht davon aus, daß bereits in naher Zukunft genetische Ingenieurskunst so weit fortgeschritten sein wird, daß sogenannte „Bio-Chips" erzeugt werden können. Durch Programmieren der Aktionen von Enzymen, die Moleküle teilen und aufbauen, kann man dann logische Schaltungen wachsen lassen bzw. züchten. Die so gewonnene Größenreduzierung der gewachsenen Schaltungen aus Protein-Molekülen wäre revolutionär. Die Idee, daß lebendes Material codierte elektronische Botschaften enthält, ist nicht neu: Auf diesem Basismechanismus funktioniert das menschliche Nervensystem.

Segen und Gefahr

Gegenwärtig verändern Genetik-Ingenieure die genetische Struktur des Weizens, um ihn gegen Schädlinge widerstandsfähiger zu machen. Allerdings kann dies genetische Mißbildungen zur Folge haben. Nachdem aber erste Schritte bei der Gen-Manipulation des Menschen getan wurden, werden weitere Experimente folgen. So könnten Wissenschaftler beispielsweise versuchen, ein Bio-ROM zu entwickeln und dieses, versehen mit einer Datenbank, auf ein menschliches Gehirn plantieren. Das mag verrückt klingen, doch ist es Wissenschaftlern bereits gelungen, elektrische Geräte an das Gehirn anzuschließen. Einer schwerhörigen Oxford-Studentin wurde ein elektronischer Impuls eines Mikrofons direkt ins Hörzentrum des Gehirns gesandt. Nach kurzer Übungszeit konnte sie die Signale entziffern und Töne „hören".

Einige Menschen sind der Ansicht, daß in ferner Zukunft unsere Abkömmlinge eine Art Hybrid Mensch-Maschine sein werden. Da der Evolutionsprozeß sich jedweden verfügbaren Materials bedient – Flossen entwickelten sich zu Beinen, Fischkiemen wurden zu Lungen –, scheint es durchaus einleuchtend, daß diese Hybrid-Organismen aus einem menschlichen Hirn, umgeben von unterschiedlichsten Geräten technischer Errungenschaft, umgeben sein könnten. Und das menschliche Hirn in seiner derzeitigen Gestalt entstand aus einer Reihe von Teilen, die während unserer Evolution entwickelt wurden.

Die Anwendungen, die sich aus diesen neuen Technologien entwickeln, sind noch unbestimmbar. Ob die hier aufgeführten Hypothesen jemals Wirklichkeit werden, hängt mehr davon ab, ob es unter soziologischen Gesichtspunkten betrachtet wünschenswert ist, als von den technischen Möglichkeiten. Und es muß gewährleistet sein, daß ein gesellschaftspolitisches Kontrollinstrumentarium besteht, das die Welt vor den Gefahren der Hochtechnisierung in wirkungsvoller Weise zu schützen imstande ist.

HAL, der im Film 2001 vorgestellte und hier in der Fortsetzung 2010 gezeigte Computer, ist mit Sprachfähigkeit und einem komplexen Gesetzessatz ausgestattet, der ihm erlaubt, den Erfolg der Mission zu beurteilen. Solche Eigenschaften werden zunehmend in die Hardware von heute eingearbeitet. In dem Buch 2010 geht Autor Arthur C. Clarke noch einige Schritte weiter, indem er das Konzept maschineller Intelligenz dahin führt, daß HAL seinen mechanischen Status überwindet: Er wird zum übermenschlichen Wesen. Damit wird die Annahme in Frage gestellt, daß Maschinen sich ohne ihren Schöpfer nicht weiterentwickeln können.

Alles an Bord

Nachdem wir uns mit den Prinzipien von Simulationsspielen befaßt haben, beginnen wir jetzt mit dem Programmieren. Die Handelsexpedition geht nach Amerika. Geschrieben in Microsoft-BASIC, ist das Programm so entworfen, daß die in jedem Artikel erstellten Module unabhängig voneinander funktionieren.

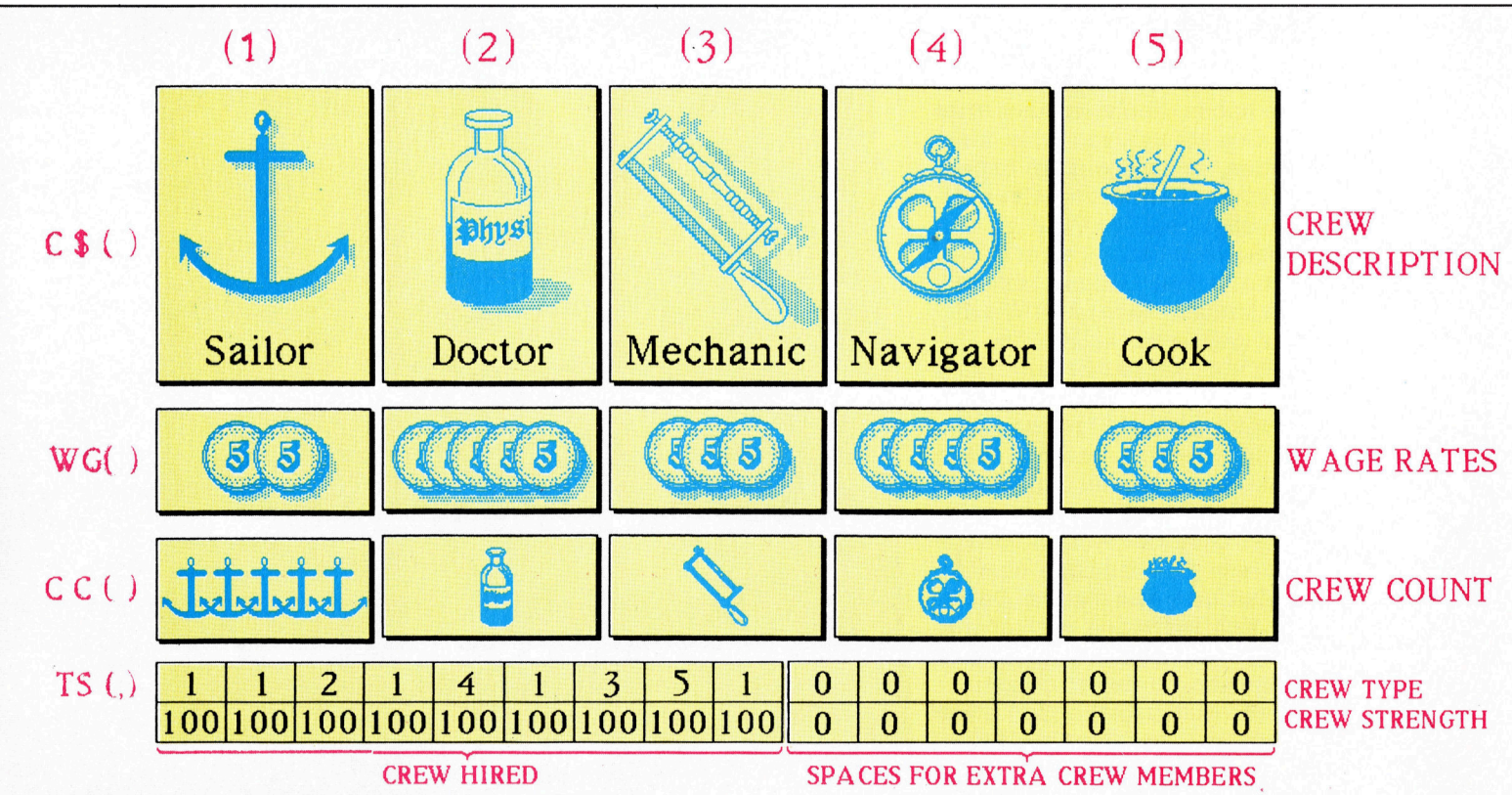
In diesem Simulationsspiel werden Sie in die Rolle eines Händlers im 15. Jahrhundert zurückversetzt, der eine Expedition in die Neue Welt organisiert. Sie müssen die Reise planen und leiten und schließlich mit den Eingeborenen möglichst profitabel Handel treiben.

Zu Beginn besitzt der Spieler ein Handelsschiff und 2000 Goldstücke. Die erste Aufgabe besteht im Anheuern einer Mannschaft, indem Sie einen angemessenen wöchentlichen Lohn für die Expeditionsdauer anbieten. Von dessen Höhe werden die Fähigkeiten der Besatzung abhängen, die eine Vielzahl von Aufgaben erfüllen muß. Insgesamt können 16 Männer auf dem Schiff untergebracht werden. Heuern Sie keine zu große Mannschaft an, da dies höhere Kosten und Nahrungsverbrauch zur Folge hat, so daß weniger Gold zum Handeln übrig bleibt. Ist die Mannschaft jedoch zu klein, so ist man nicht in der Lage, mit Zwischenfällen während der Reise fertig zu werden, wie zum Beispiel Piratenangriffen.

Das erste Modul dient zum Initialisieren von Variablen und Arrays, die während des Spiels benutzt werden, sowie zum Anheuern der Crew. In den ersten beiden Abschnitten des Moduls werden Arrays DIMensioniert, in denen wichtige Informationen über die Mannschaft gespeichert werden. C$() speichert die „Crew Description" (Beschreibung der Mannschaftstypen). Da es fünf Kategorien gibt, wird das Array in Zeile 16 auf fünf Elemente DIMensioniert. Die aktuellen Beschreibungen werden dann jedem Element individuell zugeordnet. Die Mannschaftsbeschreibungen lassen sich über die Elementnummern innerhalb des Arrays abrufen.

Der Status der Besatzung variiert im Verlauf des Spiels. Um den Zustand der Mannschaft und die Stärke zu verfolgen, wird ein zweidimensionales Array TS(,) verwendet. Das Array wird auf 16 Spalten und zwei Reihen DIMensioniert. Jede Spalte repräsentiert ein Mannschaftsmitglied; die obere Reihe den „Crew

Zum Speichern der Mannschaftsdaten werden vier Arrays verwendet. C$(), WG() und CC() speichern die Charakteristiken jedes Typs: Beschreibung, Lohn und Gesamtanzahl. TS(,) enthält die aktuelle Zusammensetzung der für die Fahrt gewählten Crew bis zu einem Maximum von 16 Männern. Jeder Typ kann mittels einer Zahl abgefragt werden. Diese Zahl ist einer Adresse in den drei Arrays zugeordnet.



Type" (Mannschaftstyp) und die untere Reihe die „Crew Strength" (Stärke). Am Anfang hat jedes Mitglied eine „Stärke" von 100.

Der Lohn für jeden Mannschaftstyp („Wage Rates") kann in einem Array WG() mit fünf Elementen gespeichert werden. Will man also den Lohn des Bordarztes (Typ 2) wissen, braucht man nur unter WG(2) nachzusehen. Matrosen erhalten zehn Goldstücke pro Woche, Ärzte 25, Navigatoren 20 und Köche fünfzehn Goldstücke. Um über das vom Spieler verbrauchte Gold Buch zu führen, wird in Zeile 12 die Variable MO auf 2000 gesetzt und dann jeweils um die Ausgaben verringert.

Später muß ermittelt werden, wie viele Männer jeden Mannschaftstyps angeheuert wurden, so daß die Effizienz der Besatzung errechnet werden kann, falls unvorhergesehene Dinge eintreten. Der Zustand der Crew kann auch jederzeit bestimmt werden, indem die obere Reihe des Arrays TS untersucht wird. Es ist jedoch einfacher und schneller, ein anderes Array CC() („Crew Count") zu initialisieren, um die Anzahl jedes Typs der Mannschaft zu speichern. Auch dieses Array erfordert nur fünf Elemente und wird in Zeile 18 DIMensioniert.

Fahrtdauer und Löhne

Nachdem diese Arrays zum Speichern der Daten über die Mannschaft initialisiert sind, gibt das Programm-Modul Informationen über die Dauer der Fahrt und das Geld, das zur Zahlung der Löhne bereitgestellt werden muß. Diese Informationen werden nicht auf einmal dargestellt, sondern mit einer speziellen Routine langsam über den Screen gerollt. Die Unterroutine ab Zeile 9100 legt den auszugebenden Satz in der Variable S$ ab und gibt ihn Buchstabe für Buchstabe aus.

Die Zusammenstellung der Mannschaft erfolgt in der Unterroutine ab Zeile 1000. Gültige Eingaben werden in TS(,) in Zeile 1156 gespeichert, wobei über CN die Anzahl der bisher angeheuerten Mitglieder gezählt wird. Mit jedem angeheuerten Seemann wird der wöchentliche Gesamtlohn WT dargestellt. Er wird errechnet, indem in Zeile 1158 das relevante Element des Arrays WG() zum vorherigen Wert von WT addiert wird. Dabei wird jeweils die aktuelle Zusammenstellung der Schiffsbesatzung durch den Programmteil von Zeile 1160 bis 1190 dargestellt. In Zeile 1185 wird überprüft, ob die Anzahl eines bestimmten Typs größer als 0 oder 1 ist. Wenn ja, wird ein S an das Ende der Beschreibung angehängt.

In diversen Programmteilen wird der Spieler aufgefordert, eine Taste zu drücken. Die Variable K$ wird verwendet, um die Meldung „PRESS ANY KEY TO CONTINUE" zu speichern. Ihr Inhalt wird in S$ übertragen, sobald die Meldung benötigt wird. Nach Rückkehr aus der „Anheuerungs"-Unterroutine endet das erste Modul.

Erstes Modul

Das erste Modul unseres Handelsspiels dient dem Anheuern einer Mannschaft. Zu diesem Zweck werden diverse Arrays initialisiert, und der Spieler kann bis zu 16 Mitglieder für die Reise auswählen.

```
9 K$="PRESS ANY KEY TO CONTINUE"
10 DIM TS(16,2): REM CREW TYPE/STRENGTH
11 CN=0: REM NO. OF CREW
12 MO=2000: REM START MONEY
13 DIMWG(5):WG(1)=10:WG(2)=25:WG(3)=15:WG(4)=20:WG(5)=15: REM WAGES
14 WT=0: REM WEEKLY WAGE BILL
15 :CM=16:REM MAX CREW
16 DIMC$(5):C$(1)="SAILOR":C$(2)="DOCTOR":C$(3)="MECHANIC"
17 C$(4)="NAVIGATOR«:C$(5)="COOK"
18 DIMCC(5): REM COUNT OF EACH CREW TYPE
80 PRINTCHR$(147):S$=" NEW WORLD TRADING GAME*":GOSUB9100:PRINT
81 GOSUB9200
82 S$="YOU ARE THE CAPTAIN OF A SHIP*" :GOSUB9100:PRINT
83 S$="SAILING TO THE NEW WORLD. THE*":GOSUB9100:PRINT
84 S$="JOURNEY IS SAID TO TAKE EIGHT*":GOSUB9100:PRINT
85 S$="WEEKS, BUT MAY TAKE LONGER. YOU*":GOSUB9100:PRINT
86 S$="MUST HIRE A CREW,PAY THEM, BUY*":GOSUB9100:PRINT
87 S$="PRVISIONS,EQUIPMENT AND GOODS*":GOSUB9100:PRINT
88 S$="FOR TRADING. YOU HAVE 2000 GOLD*":GOSUB9100:PRINT
89 S$="PIECES TO SPEND.*":GOSUB9100:PRINT:GOSUB9200
90 PRINT:S$=" GOOD LUCK!*":GOSUB9100:GOSUB9200:PRINT
91 S$=K$:GOSUB9100
94 GETP$:IFP$=""THEN94
95 GOSUB9200
500 GOSUB1000
999 END
1000 PRINTCHR$(147):PRINT" STAGE 1 — HIRING CREW"
1010 PRINT" ----------"
1012 PRINT
1015 GOSUB9200
1020 PRINT:PRINT"CREW TYPES AVAILABLE:"
1025 GOSUB9200
1030 PRINT
1040 PRINT"TYPE DESCRIPTION WAGES OER WEEK"
1050 PRINT"------- ---------- ----------"
1060 PRINT" 1 SAILOR 10 GOLD PCS"
1070 PRINT" 2 DOCTOR 25 GOLD PCS"
1080 PRINT" 3 MECHANIC 15 GOLD PCS"
1090 PRINT" 4 NAVIGATOR 20 GOLD PCS"
1100 PRINT" 5 COOK 15 GOLD PCS"
1105 GOSUB9200
1110 PRINT:PRINT:PRINT
1120 S$="ENTER CREW TYPE REQUIRED(1—5)*":GOSUB9100
1122 S$="OR ‚F' TO FINISH HIRING*:GOSUB9100:PRINT:INPUTP$
1125 CT=VAL(I$)
1128 IFLEFT$(P$,1)="F"THENPRINT:PRINT"END OF CREW HIRE.":GOSUB9200
     :GOTO1310
1130 IFCT>0ANDCT<STHEN1150
1139 PRINT:PRINT
1140 PRINTP$;:S$=" IS NOT A CREW TYPE*":GOSUB9100
1142 GOSUB9200
1145 S$="PLEASE ENTER AGAIN"
1146 GOSUB9100
1147 GOTO1300
1150 PRINT:PRINT
1155 CN=CN+1:REM CREW HIRED SO FAR
1156 TS(CN,1)=CT:REM CREW TYPE
1157TS(CN,2)=100:REM STARTING STRENGHT
1158 WT=WT+WG(CT):REM TOTAL WAGES
1159 CC(CT)=CC(CT)+1:REM CREW TYPE COUNT
1160 S$="CREW SO FAR:"
1170 FORT=1TO5
1180 PRINTS$;CC(T);" ";C$(T)
1185 IFCC(T)>1ORCC(T)=0THENPRINT"S":GOTO1189
1186 PRINT " "
1189 S$="
1190 NEXT
1195 PRINT: PRINT"TOTAL WEEKLY WAGE BILL ";WT
1200 IFCN=CM-1THENPRINT:S$="ONLY ONE MORE CREW*":GOSUB9100:GOTO1295
1202 IFCN=CMTHENPRINT:S$=" SHIP NOW FULL!!":GOSUB9100:GOTO1310
1295 REM
1300 GOTO1015
1310 PRINT: S$=K$:GOSUB9100:PRINT: GOSUB9200
1320 GETIP$:IFP$=""THEN1320
1999 RETURN
9100 REM SLOW PRINT OF ST$
9110 FOR S3=1TO32
9115 IFMID$(S$,S3,1)="*"THENS3=32:GOTO9140
9120 PRINTMID$(S$,S3,1);
9130 FORS4=1TO25:NEXT
9140 NEXT:PRINT
9199 RETURN
9200 REM DELAY LOOP
9210 FORS5=1TO1000:NEXT
9299 RETURN
```

BASIC-Dialekte

Spectrum
Führen Sie die folgenden Änderungen aus:

```
16 DIM C$(5,9):C$(1)="SAILOR":C$(2)=
   "DOCTOR":C$(3)="MECHANIC"
80 CLS:LET S$="NEW WORLD TRADING
   GAME*":GOSUB 9100: PRINT
94 LET P$=INKEY$:IF P$="" THEN GO TO 94
1005 CLS:PRINT" STAGE 1 — HIRING CREW"
1128 IF P$(1 TO 1)="F" THEN PRINT"END OF
   CREW HIRE."GOSUB 9200:GOTO 1310
9115 IF S$(S3 TO S3)="*" THEN S3=32:
   GOTO 9140
9120 PRINT S$(S3 TO S3);
```

ACORN B
Ändern Sie die folgenden Zeilen:

```
80 CLS:S$="NEW WORLD TRADING
   GAME*":GOSUB 9100:PRINT
94 P$=GET$
1005 CLS:PRINT" STAGE 1 — HIRING CREW"
```

K

Kartei anlegen

Viele Datenbankprogramme verfügen über eine spezielle Sprache, die den Zugang zu den gespeicherten Informationen vereinfachen soll. Am Beispiel des Programms „dBase II" von Ashton Tate wollen wir die Möglichkeiten für einen effektiven Einsatz leistungsfähiger Datenbanksysteme untersuchen.

Bevor wir die vom Anwender definierten Prozeduren erklären, werfen wir zuerst einen Blick auf die allgemeine Befehlsstruktur der Datenbanken. Meist erfährt das Programm über die Befehle SEARCH, LOCATE, DISPLAY, NEXT, LAST usw., was der Anwender eigentlich möchte. Danach wird die Datei geöffnet und sequentiell (vom ersten bis zum letzten Eintrag) durchgearbeitet. Die gewünschten Datensätze werden sozusagen „im Vorbeigehen" – wie eine Karteikarte – entnommen.

Neben der direkten Eingabe bieten manche Datenbanksysteme – Archive von Psion und dBase II von Ashton Tate sind gute Beispiele dafür – zusätzlich Möglichkeiten der Programmierung fester Abläufe, mit denen der Anwender schneller zur gewünschten Information kommt. Dabei kann es sich durchaus um mehr als nur eine Kombination aus Standard-Befehlen handeln – die Datenbanken enthalten als Teil des Systems vollständige Programmiersprachen. Einige ähneln in ihrer Struktur bekannten Sprachen und sind leicht erlernbar. Bei Archive ist die verwendete Sprache mit dem SuperBASIC von Sinclair identisch.

Die Befehlsformate von dBase II decken sich zwar nicht mit einem bekannten „Computer-Dialekt", die leicht faßlichen und schnell erlernbaren Befehle erinnern aber an strukturiertes BASIC. Der Aufbau ist einfach, es stehen Befehle wie DO WHILE...ENDDO, DO CASE...ENDCASE und IF...ENDIF zur Verfügung. Insgesamt gibt es circa 100 Befehle, von denen einige aus BASIC bekannt, andere für die Datenverwaltung maßgeschneidert sind. Vertraut sind etwa CALL (Aufruf eines Maschinenprogramms), NOTE (entspricht der BASIC-Anweisung REM), STORE <Ausdruck> to <Variable> (entspricht dem LET <Variable> = <Ausdruck>). Weniger bekannt sind Befehle wie SKIP (zum Vor- und Rückwärtsbewegen durch die einzelnen Datensätze), PACK (um überflüssige Datensätze zu entfernen) und FIND <String>.

Auch eine Vielzahl von Funktionen erinnert stark an BASIC. Dazu gehören etwa CHR <Ausdruck> (entspricht CHR$(X), LEN <String>, <Ausdruck> (wie BASIC-LEN), TYPE <Ausdruck> (ergibt den Typ eines Ausdrucks) und DATE() (Systemvariable mit dem aktuellen Datum).

Die Annehmlichkeiten eines selbstdefinierten Ablaufs im Vergleich zur Eingabe einer Befehlsfolge über die Tastatur wollen wir anhand eines Beispiels verdeutlichen, dBASE II soll darin als Lager- und Inventurverzeichnis genutzt werden. Wir wollen auch zeigen, wie sich bestimmte Informationen mit einem kurzen Programm sehr schnell ermitteln lassen. Die Datensätze in unserer Datei sind unkompliziert, sie enthalten nur vier numerische Felder und ein Zeichenfeld. So sieht der typische Eintrag aus:

```
HERSTELLERNUMMER                06116
KATALOGNUMMER                    3995
LAGERBESTAND                       86
PREIS                           34,75
BESCHREIBUNG      Stutzen mit Ventilklappe
```

Jedes Feld muß einen Namen und eine definierte Länge haben, außerdem muß unbedingt festgelegt werden, ob es sich um Zeichen-, numerische oder logische Felder handelt. Dazu genügt folgendes:

```
HERSTELLER        :5  Stellen;numerisch
KATALOG           :5  Stellen;numerisch
BESTAND           :3  Stellen;numerisch
PREIS             :6  Stellen;numerisch
BESCHREIBUNG      :40 Stellen;Zeichen
```

Nach dem Programmstart meldet sich dBase II betriebsbereit und wartet auf Eingaben. Mit CREATE wird eine neue Datei angelegt, für die zuerst ein Name angegeben werden muß. Wir verwenden den Namen TEILE. Der gesamte Vorgang sieht so aus:

```
.create teile <CR>
ENTER RECORD STRUCTURE AS FOLLOWS:
FIELD, NAME, TYPE, WIDTH, DECIMAL PLACES
001 hersteller,n,5,0
002 katalog,n,5,0
003 bestand,n,3,0
```

```
004 preis,n,6,2
005 beschreibung,c,40
006 <CR>
```

dBase II wandelt die kleinen Buchstaben in großgeschriebene Lettern um. Wichtig ist, daß bei numerischen Feldern neben der Feldgröße auch die Zahl der Nachkommastellen angegeben werden muß. Mit der Return-Taste wird die Phase des Dateiaufbaus abgeschlossen.

Als nächstes müssen wir mit dem Befehl APPEND die ersten Daten eingeben. APPEND löscht den Bildschirm und zeigt das Dateiformat mit Leerstellen für die Daten:

```
HERSTELLER  :      :
KATALOG     :      :
BESTAND     :   :
PREIS       :      :
BESCHREIBUNG:
```

Nun können die entsprechenden Werte jedes Feldes über die Tastatur eingegeben werden. Return schließt die Eingabe in das einzelne Feld ab und übernimmt die Daten.

Nach der Dateneingabe möchte man die Informationen natürlich auch nutzen. Angenommen, Sie brauchen bei einer Inventur den Gesamtverkaufspreis aller gelagerten Waren. Dazu könnte man entweder jeden Eintrag einzeln auf den Bildschirm holen und sich Preis sowie Lagerbestand notieren. Dann müßten die Einzelpreise mit der Stückzahl multipliziert und alle Ergebnisse zusammengerechnet werden. Bei einer konventionellen Karteiblatt-Ablage wäre dies der einzige Weg zum korrekten Ergebnis.

Zum Glück läßt sich dieser zeitraubende Vorgang bei fast allen Datenbank-Systemen abkürzen. Bei dBase II geht das so:

```
.use teile
.sum bestand * preis
37870.58
```

Mit der ersten Befehlszeile wird dBase II angewiesen, die TEILE-Datei aufzurufen. Die nächste Zeile bedeutet „Multipliziere den Inhalt der BESTAND-Felder aller Datensätze mit den dazugehörigen PREIS-Feldern und addiere die Ergebnisse." Zeile 3 gibt das Resultat der beschriebenen Operation an. Verglichen mit der Dauer einer solchen Berechnung mit Hilfe normaler Karteikarten brechen Datenbanksysteme hier jeden Rekord.

Das Beispiel zeigt, was sich bereits mit einfachen Befehlen aus einem Datenbanksystem herausholen läßt. Die Speicherung einer Befehlsfolge als Programm kann diese Leistungen aber noch erheblich steigern. Unser einfaches Programm zur Bewertung des Lagerbestandes kann bei dBase II unter dem Namen WERT gespeichert werden. Dazu muß man eingeben:

```
.modify command
ENTER FILE NAME: wert
set talk off
use teile
sum bestand * preis
set talk on
cancel
```

Dieses Kurzprogramm kann auf Diskette gespeichert und bei Verwendung von dBase II jederzeit mit

```
.do wert
```

aufgerufen werden. Das Laden und Starten des Ablaufs geschieht dann automatisch. Nach der Programmausführung geht es wieder im Normalmodus von dBase II weiter.

Wer ist Buchhalter?

```
*Zeigt alle Buchhalter
SET TALK OFF
USE MITGLIEDER
DO WHILE.NOT.EOF
   IF BERUF—"Buchhalter"
      DISPLAY NAME
   ENDIF
   SKIP
ENDDO
```

Dieses Programm würde aus einer mit dBase II geführten Liste – von Vereinsmitgliedern etwa – alle Buchhalter heraussortieren und ihre Namen auf dem Bildschirm zeigen. Der Stern vor der ersten Zeile bedeutet, daß es sich um einen Kommentar handelt.

Der Befehl USE MITGLIEDER gibt an, in welcher Datenbankdatei gesucht werden soll. Durch die Schleife innerhalb der DO WHILE . . .ENDDO-Befehle werden alle Datensätze geprüft.

Innerhalb der Schleife steht die Anweisung, daß die Namen der Datensätze angezeigt werden sollen, in denen das Programm einen „BUCHHALTER" entdeckt. Die IF-Bedingung muß auf jeden Fall mit ENDIF abgeschlossen werden. Das Programm gelangt dann zum Befehl SKIP, der dBase II zum folgenden Eintrag weiterleitet.

Klare Sicht

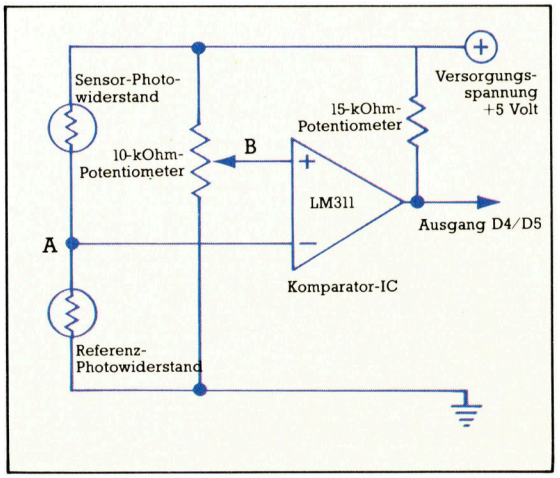
Durch den Einbau einer Reihe von Photowiderständen lernt unser Roboter sehen – so gut, daß er einer gezeichneten schwarzen Linie folgen kann. Programme für die Lichtsensor-Steuerung laufen auf dem Acorn B, dem Commodore 64 und dem aufgerüsteten Spectrum.

Die „Sehwerkzeuge" des Roboters bestehen aus zwei identischen Schaltungen, die über je eine Datenleitung mit dem User Port verbunden sind. Wichtigstes Element der Schaltungen ist das Komparator-IC LM311, das zwei anliegende Spannungen vergleichen kann. Ist die Spannung am positiven Eingang höher als am negativen, schaltet der Ausgang des ICs von 0 Volt auf die Betriebsspannung um. Die Ausgabe des Chips ist also bei identischen Eingangsspannungen 0.

Zwei Photowiderstände (LDR = Light Dependent Resistors) bilden einen Spannungsteiler zwischen der Versorgungsspannung und der Masse. Wenn beide Photowiderstände der gleichen Lichtmenge ausgesetzt sind, haben sie den gleichen Widerstand – die Spannung am Punkt A der Schaltung liegt dann bei circa 2,5 Volt (halbe Versorgungsspannung). Mit dem Potentiometer wird am Punkt B die gleiche Spannung eingestellt. Solange beide Eingangsspannungen gleich sind, bleibt die Ausgangsspannung des Komparator-Chips bei 0 Volt. Abnehmende Belichtung des Sensor-LDR – wenn er sich etwa über einer schwarzen Linie befindet – läßt seinen Widerstand und damit die Spannung an Punkt A ansteigen. Der Komparator-Ausgang schaltet dann auf die Versorgungsspannung (logisch 1) um.

Mit zwei symmetrischen Schaltungen kann der Computer erkennen, nach welcher Seite der Roboter von der schwarzen Linie abgewichen ist. Die Widerstandspaare bestehen aus dem Referenz-Photowiderstand zur Messung der Umgebungshelligkeit und dem Sensor-Photowiderstand für die Prüfung der Lichtstärke über der Leitlinie.

Nach Aufbau und Montage der Platine muß die Schaltung kalibriert werden. Nehmen Sie die Einstellung bei Lichtverhältnissen vor, die den späteren Betriebsbedingungen ähneln. Die beiden Potentiometer auf der Platine können Sie mit folgendem Programm exakt und bequem justieren.



Die Hardware zum Verfolgen von Linien besteht aus zwei identischen Schaltungen, die je ein Paar Photowiderstände (LDRs) enthalten. Wenn die LDRs unterschiedlich starke Lichtwerte empfangen, liefert das Komparator-IC ein digitales Signal. Der Roboter „weiß" dann, daß er sich nicht mehr direkt über der Leitlinie befindet.

LDR-Testprogramm

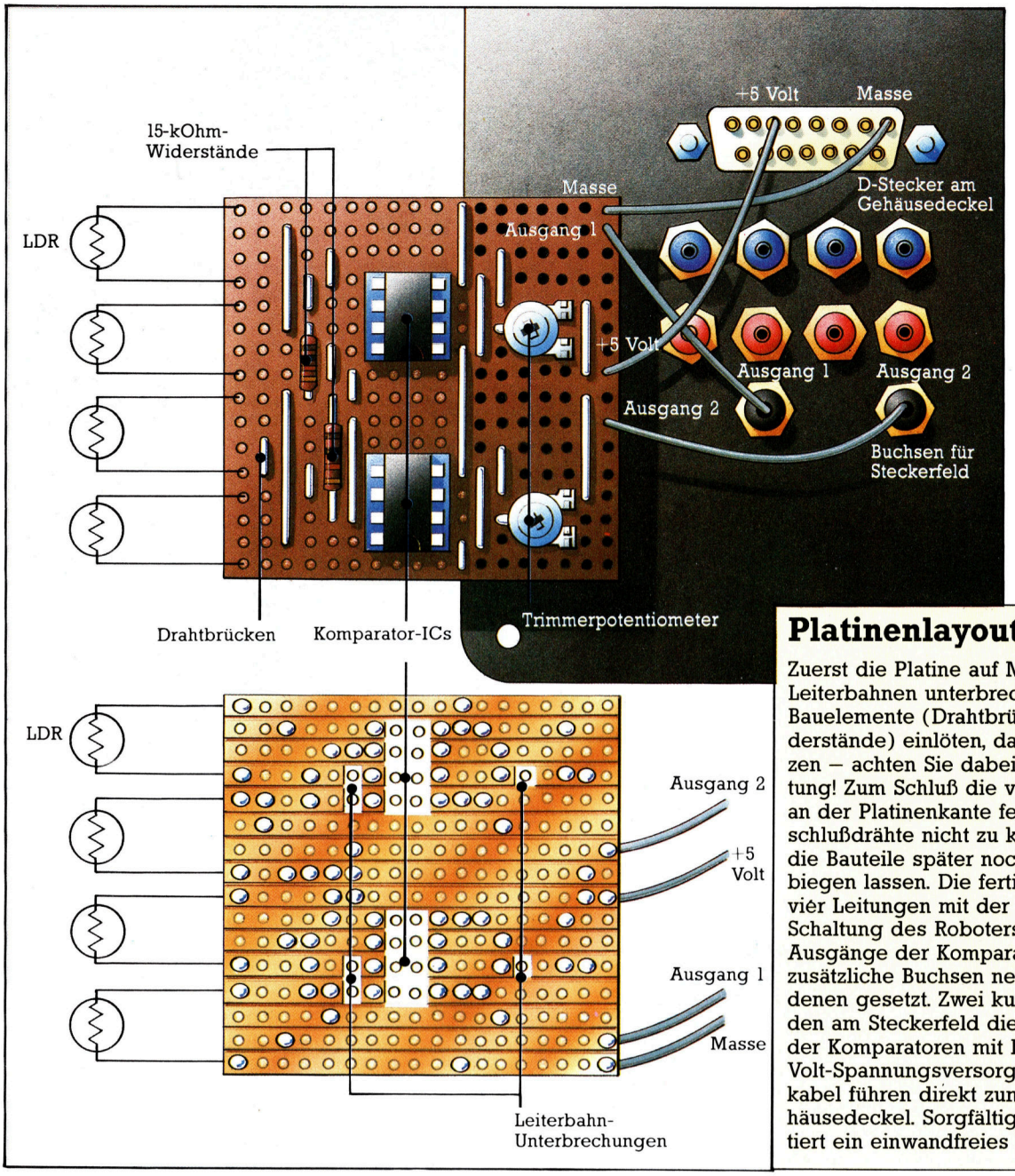
```
1000 REM **** BBC LDR TEST ****
1010 DDR=&FE62:DATREG=&FE60:?DDR=15:REM LINES
     0-3 OUTPUT
1020 REPEAT
1030 contents=?DATREG
1040 IF(contents AND 16)=0 THEN PRINT TAB(5)
     "LEFT";
1050 IF(contents AND 32)=0 THEN PRINT TAB(15)
     "RIGHT";
1060 PRINT
1070 UNTIL FALSE
```

```
1000 REM **** CBM LDR TEST ****
1010 DDR=56579:DATREG=56577:POKE DATREG,15
1020 CN=PEEK(DATREG)
1030 IF(CN AND 16)=0 THEN PRINTTAB(5)"LEFT";
1040 IF(CN AND 32)=0 THEN PRINTTAB(15)"RIGHT";
1050 PRINT
1060 GOTO 1020
```

```
1000 REM **** SPECTRUM LDR TEST ****
1010 CLEAR 32499:LET ST=32500: GO SUB 3000
1020 LET NM=16:GO SUB 2000:IF USR ST=0 THEN PRINT
     TAB 5;"LEFT";
1030 LET NM=32:GO SUB 2000:IF USR ST=0 THEN PRINT
     TAB 5;"RIGHT";
1040 PRINT
1050 GO TO 1020
1060 :
2000 REM **** PERFORM AND ****
2010 POKE ST+1,IN 31
2020 POKE ST+3,NM:RETURN
2030 :
3000 REM **** MACHINE CODE LOADER ****
3010 FOR I=ST TO ST+8
3020 READ A:POKE I,A
3030 NEXT I
3040 DATA 62,0,14,0,161,6,0,79,201
3050 RETURN
```

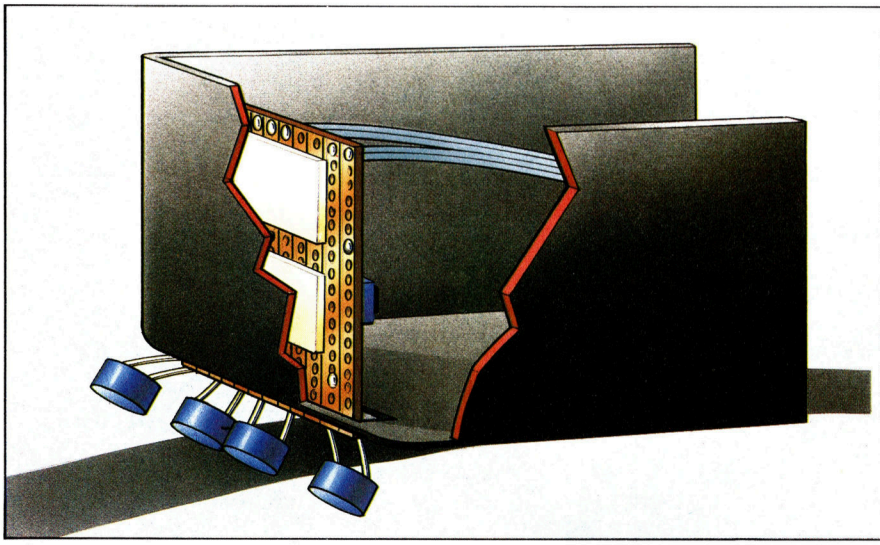
Liste der Bauteile

Anzahl	Bauteil
2	Komparator-ICs LM311
2	8polige IC-Sockel
2	2-mm-Buchsen
2	Widerstände, 15 kOhm
2	Trimmerpotentiometer
4	Photowiderstände ORP12
1	Lochplatine (Veroboard), 24 Bahnen à 50 Löcher
1	Stück Doppelklebeband
1 Meter	Flachbandkabel, 4polig
1 Meter	verzinnte Litze, ohne Isolierung



Platinenlayout

Zuerst die Platine auf Maß schneiden und die Leiterbahnen unterbrechen. Die passiven Bauelemente (Drahtbrücken, Sockel und Widerstände) einlöten, dann erst die ICs einsetzen – achten Sie dabei auf die korrekte Richtung! Zum Schluß die vier Photowiderstände an der Platinenkante festlöten. Dabei die Anschlußdrähte nicht zu kurz wählen, damit sich die Bauteile später noch an die richtige Stelle biegen lassen. Die fertige Platine wird durch vier Leitungen mit der bereits vorhandenen Schaltung des Roboters verbunden. Für die Ausgänge der Komparator-ICs werden zwei zusätzliche Buchsen neben die acht vorhandenen gesetzt. Zwei kurze Leitungen verbinden am Steckerfeld die Ausgangsbuchsen der Komparatoren mit Leitung 4 und 5. Die 5-Volt-Spannungsversorgung und das Massekabel führen direkt zum D-Stecker am Gehäusedeckel. Sorgfältige Verarbeitung garantiert ein einwandfreies Funktionieren.



Im Blick

Die Platine wird auf der vorderen Innenseite des Roboter-Gehäuses so festgeklebt, daß die vier Photowiderstände durch einen Schlitz im Boden (6 × 1 cm) herausstehen können. Die Anschlußdrähte vorsichtig in die abgebildete Form biegen. Die beiden mittleren LDRs sollten so nahe wie möglich beieinanderliegen.

Auf einen Bogen weißes Papier zeichnen Sie sich eine 2,5 cm breite schwarze Linie. Wenn der Roboter jetzt mit den beiden mittleren Photowiderständen direkt über der Linie steht, müssen Sie die beiden Trimmer so einstellen, daß der Computer kein Signal gibt. Die Bildschirmausgabe „RIGHT" oder „LEFT" darf nur bei einer entsprechenden Verschiebung des Roboters über der schwarzen Linie erfolgen. Nach der Kalibrierung wird der Roboter langsam von links nach rechts über die Linie hinweggeschoben. Dabei sollte folgendes angezeigt werden: Wenn sich alle vier LDRs über dem weißen Papier links neben der Linie befinden, „LEFT RIGHT". Wird der Roboter näher an die Linie herangeführt, erscheint „LEFT", und wenn er direkt darüber steht, sollte der Bildschirm leer bleiben. Etwas weiter rechts zeigt der Screen dann entsprechend „RIGHT", mit allen vier LDRs rechts neben der Linie wechselt die Anzeige erneut zu „LEFT RIGHT". Für das Programm ist der linke Lichtsensor an Leitung 4, der rechte an 5 angeschlossen.

Achten Sie bei Kalibrierung und Inbetriebnahme auf konstante Lichtverhältnisse. Sie lassen sich leicht rekonstruieren, wenn mit einer hellen Lampe ausgeleuchtet wird.

Das Programm „Linie verfolgen" nutzt die Routine zum Bewegen und Drehen des Roboters. Zu Beginn muß der Roboter auf der Linie stehen. Eine Programmschleife bewegt den Roboter um jeweils einen Millimeter vorwärts und prüft, ob er vom Weg abgekommen ist. Das ist der Fall, wenn Bit 4 oder 5 des User-Port-Datenregisters auf Low wechseln. Low von Bit 4 zeigt eine Abweichung nach links. In diesem Fall würde der Roboter vor dem Weiterfahren eine 5-Grad-Kurve nach rechts beschreiben. Bei Rechtsabweichung (Bit 5 steht auf Low) ist der Vorgang spiegelbildlich gleich. Bei einem Low auf beiden Bits endet das Programm mit dem Ende der Linie.

Den Kurvenwert von 5 Grad haben wir experimentell ermittelt. Der Wert muß möglicherweise der Strichstärke und Enge der Kurven angepaßt werden. Auch Abwandlungen des Programms selbst sind möglich – wenn der Roboter die Linie zum Beispiel erst einmal finden soll. Mit dem Steckerfeld können Sie auch andere Sensor-Kombinationen einstellen.

Der nächste Kursabschnitt behandelt Aufbau und Steuerung eines Roboterarmes.

Programm „Linie verfolgen"

Acorn B

```
 10 REM **** BBC LINE FOLLOWER ****
 20 :
 30 PROCinitialise
 40 PROCfollow_line
 50 END
 60 :
 70 DEF PROCfollow_line
 80 REPEAT
 90 PROCmove(forwards,1)
100 PROCtest_ldr
110 UNTIL endflag=1
120 ENDPROC
130 :
140 DEF PROCtest_ldr
150  ldrval=?DATREG AND 48
160 IF ldrval=32 THEN PROCturn(right,5)
170 IF ldrval=16 THEN PROCturn(left,5)
180 IF ldrval=0 THEN endflag=1
190 ENDPROC
200 :
210 DEF PROCinitialise
220 DDR=&FE62:DATREG=&FE60:?DDR=15
230 ?DATREG=1
240 forwards=4:backwards=2:left=6:right=0
250 pd_ratio=3.34446:pa_ratio=379/90
260 endflag=0
270 ENDPROC
280 :
290 DEF PROCmove(dir,distance)
300 ?DATREG=(?DATREG AND 1)OR dir
310 pulses=pd_ratio*ABS(distance)
320 FOR I=1 TO pulses:PROCpulse:NEXT I
330 ENDPROC
340 :
350 DEF PROCturn(dir,angle)
360 ?DATREG=(?DATREG AND 1)OR dir
370 pulses=pa_ratio*angle
380 FOR I=1 TO pulses:PROCpulse:NEXT I
390 ENDPROC
400 :
410 DEF PROCpulse
420 ?DATREG=(?DATREG OR 8)
430 ?DATREG=(?DATREG AND 247)
440 ENDPROC
```

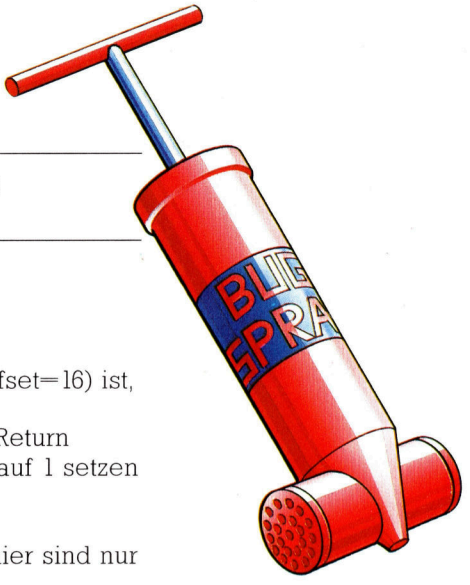
Commodore 64

```
10 REM **** CBM LINE FOLLOWER ****
20 :
30 GOSUB2000:REM INIT
40 GOSUB1000:REM FOLLOW LINE
50 END
60 :
1000 REM **** FOLLOW LINE ****
1010 DR=FW:DS=1:GOSUB2500:REM MOVE
1020 GOSUB1500:REM TEST LDRS
1030 IF EF<>1 THEN 1000
1040 RETURN
1050 :
1500 REM **** TEST LDRS ****
1510 LV=PEEK(DATREG)AND 48
1520 IF LV=32 THEN DR=RT:DS=5:GOSUB3000:REM TURN
1530 IF LV=16 THEN DR=LF:DS=5:GOSUB3000:REM TURN
1540 IF LV=0 THEN EF=1
1550 RETURN
1560 :
2000 REM **** INIT ****
2010 DDR=56579:DATREG=56577:POKEDDR,15
2020 POKEDATREG,1
2030 FW=4:BW=2:LF=6:RT=0
2040 PD=3.34446:PA=379/90
2050 EF=0:REM ENDFLAG
2060 RETURN
2070 :
2500 REM **** MOVE (DR,DS) ****
2510 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
2520 PL=PD*ABS(DS)
2530 FOR I=1 TO PL:GOSUB 3500:NEXT I
2550 RETURN
2560 :
3000 REM **** TURN (DR,DS) ****
3010 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
3020 PL=PA*ABS(DS)
3030 FOR I=1 TO PL:GOSUB 3500:NEXT I
3040 RETURN
3050 :
3500 REM **** PULSE ****
3510 POKE DATREG,PEEK(DATREG)OR 8
3520 POKE DATREG,PEEK(DATREG)AND 247
3530 RETURN
```

Sinclair Spectrum

```
10 REM **** SPECTRUM LINE FOLLOWER ****
12 REM DELETE LINES 2010,2020,2510,3010 OF CBM
   VERSION
13 REM AND MAKE THESE CHANGES
15 CLEAR 32499:ST=32500:GOSUB 4500
1510 LET NM=48:GO SUB 4000:LET LV=USR ST
3510 OUT 31,DR+9
3520 OUT 31,DR+1
4000 REM **** PERFORM AND ****
4010 POKE ST+1,IN 31
4020 POKE ST+3,NM:RETURN
4500 REM **** MACHINE CODE LOADER ****
4510 FOR I=ST TO ST+8
4520 READ A:POKE I,A
4530 NEXT I
4540 DATA 62,0,14,0,161,6,0,79,201
4550 RETURN
```

Korrekturmodul



Wir vervollständigen die Ein- und Ausgaberoutinen unseres Debuggers und setzen das vollständige Modul zusammen.

Für das Ein- und Ausgabemodul brauchen wir die folgenden vier Routinen: GETHX2, GETHX4, PUTHEX und PUTCR. GETHX2 holt eine zweistellige Hexadezimalziffer von der Tastatur und GETHX4 eine vierstellige Zahl. Wir müssen uns dabei entscheiden, ob wir nur Eingaben mit zwei oder vier Ziffern zulassen wollen oder aber weniger Zeichen, die mit einem Return beendet werden. Weiterhin muß festgelegt werden, ob das Zeichen für „Backspace" bereits Eingegebenes löschen soll.

Für GETHX4 nehmen wir die einfachste Methode: Es müssen vier Ziffern eingegeben werden, und ein Backspace ist nicht möglich. Der 16-Bit-Wert (die Adresse) wird im D-Register gespeichert.

Bei GETHX2 können Probleme entstehen: Für die Untersuchung und Veränderung des Speichers (Befehl M) müssen Acht-Bit-Werte eingegeben werden, und der Zugriff auf eine Adresse muß gewährleistet sein. Der Inhalt dieser Adresse wird dann angezeigt, und der Anwender kann entweder Return eingeben (um sich die darauffolgende Speicherstelle anzusehen) oder eine aus zwei Ziffern bestehende Hexzahl (die an dieser Adresse gespeichert wird) oder irgendein anderes Zeichen (beispielsweise einen Punkt), das ihn wieder auf die Befehlsebene zurückbringt. Die beiden zusätzlichen Zeichen lassen sich an die Liste der gültigen Hexzeichen anfügen. GETHX2 akzeptiert nun entweder zwei Hexziffern oder ein Return oder einen Punkt. Der Acht-Bit-Wert wird in B gespeichert, und A zeigt an, welche Situation eingetreten ist. Wenn eine zweistellige Zahl eingegeben wurde, erhält A den Wert 0, bei Eingabe eines Return eine 1 und bei einem Punkt eine −1. Bei diesen Werten können wir das Register A testen, ohne es mit einem anderen Wert vergleichen zu müssen.

Nehmen wir an, daß für dieses Modul folgende Deklarationen gemacht wurden:

```
HEXCHS FCC'0123456789ABCDEF'
PUNKT  FCB'.
RETURN FCB 13 (ASCII-Code für Return)
```

Wir übergeben 16 als Stringlänge an GETHX4 und 18 an GETHX2 (wo auch die beiden zusätzlichen Zeichen gebraucht werden).

Daten:

Nächstes-Zeichen ist der ASCII-Code in A
Offset in die Tabelle gültiger Zeichen in B
Hex-Wert ist ein Acht-Bit-Wert, der in B aufgebaut wird
Flag ist entweder 0, 1 oder −1 in A

Ablauf:

```
Nächstes-Zeichen holen
IF das Zeichen ein Punkt (Offset=16) ist,
   dann Flag auf −1 setzen
ELSE wenn das Zeichen ein Return
   (Offset=17) ist, dann Flag auf 1 setzen
ELSE
   Offset zwischenspeichern
   Nächstes-Zeichen holen (hier sind nur
      Hexziffern gültig)
   Hex-Wert zusammensetzen
ENDIF
```

Der endgültige Code von GETHX2 ist auf den nächsten Seiten abgedruckt. Die Codierung von GETHX4 ist einfacher, da wir dafür Teile von GETHX2 einsetzen können. Wenn wir HX4 als zweiten Einsprungspunkt in GETHX2 einsetzen, wird sichergestellt, daß nur gültige Hexziffern akzeptiert werden (vorausgesetzt, B wird zuvor auf 16 gesetzt). Auf diese Weise ist die Programmierung der Eingabe von vier Hexziffern weniger kompliziert.

Daten:

Hex-Zahl ist ein 16-Bit-Wert, der in D gespeichert wird
Höchstwertiges-Byte und
Niederwertiges-Byte sind Acht-Bit-Werte, die in B gespeichert werden

Ablauf:

```
Höchstwertiges-Byte holen
Höchstwertiges-Byte zwischenspeichern
Niederwertiges-Byte holen
Hex-Zahl zusammensetzen
```

Der Code für GETHX4 befindet sich ebenfalls auf der nächsten Seite.

Das Anzeigen der Zeichen ist weitaus weniger kompliziert. Wir gehen davon aus, daß sich bei der PUTHEX-Routine die benötigte Acht-Bit-Zahl in B befindet.

Daten:

Zahl ist ein Acht-Bit-Wert in B
Offset ist der an HEXCHS übergebene Vier-Bit-Offset

Ablauf:

```
   Die vier höchstwertigen Bits von Zahl
      als Offset herausziehen
   HEXCHS (Offset) anzeigen
   Die vier niederwertigen Bits von Zahl
      herausziehen
   HEXCHS (Offset) anzeigen
```

Die PUTCR- ist die letzte Routine für die Ein- und Ausgabe. Der Code ist einfach und erklärt sich eigentlich selbst. Wir können nun das Ein- und Ausgabemodul zusammenbauen.

Ablauf:

Befehl-holen liefert den Offset in B, der dann als Offset in eine Sprungtabelle eingesetzt wird

Adresse-holen speichert die Rücksprungadresse in D

Wert-holen speichert den Rücksprungwert in B und Flag in A

Wert-anzeigen wird an B übergeben

Adresse-anzeigen wird an D übergeben

Der endgültige Code für das Ein- und Ausgabemodul steht auf der folgenden Seite. Wir wenden uns nun wieder den Unterbrechungspunkten zu, für die wir in der letzten Folge bereits den Code für das Anlegen der Punkte abgedruckt hatten. Wir mußten dabei den ersten Prozeß – das Einsetzen der Unterbrechungspunkte – zurückstellen, da uns die notwendige Adresse fehlte. Die obenstehenden Routinen erledigen nun diese Aufgabe, und der entsprechende Code kann somit nun endlich vervollständigt werden.

Beachten Sie, daß der Befehl INC NUMBP,PCR eine 1 zu der Anzahl-der-Unterbrechungspunkte addiert. Der Wert von A ist an dieser Stelle um eins kleiner als der von Anzahl-der-Unterbrechungspunkte, der den Offset in der Unterbrechungstabelle angibt. Da die Adresse an D zurückgegeben und damit der Wert von A überschrieben wird (das D-Register ist aus A und B zusammengesetzt), müssen wir die aktuelle Adresse in Y ablegen.

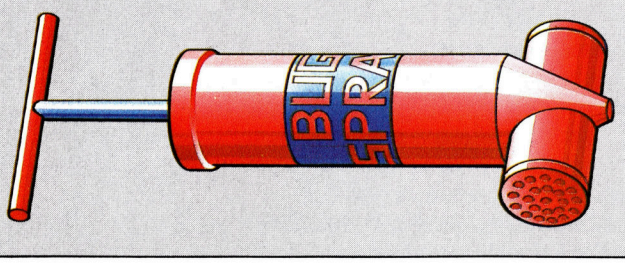
Nach der Codierung des ersten Prozesses fehlen noch drei Arbeitsgänge. Zwei davon sind die Umkehr der beiden bisher programmierten Routinen: Unterbrechungspunkt-

Die GETHX2-Routine

```
GETHX2  LDB   #18          Zahl-der-Zeichen-OK
HX4     PSHS  X            Eingesetzte Register sichern
        LEAX  HEXCHS,PCR   Adresse von Zeichen-OK in X setzen
        BSR   GETCH        Nächstes Zeichen holen
IFOO    CMPB  #16          Wenn Offset = 16
        LDA   #$FF         Flag auf -1 setzen (im Zweierkomplement)
        BRA   ENDFOO
        CMPB  #17          Wenn Offset = 17
        LDA   #1           Flag auf 1 setzen
        BRA   ENDFOO
        LSLB               B vier Bits nach links verschieben, um
        LSLB               die höchstwertige Ziffer zu erhalten; B
        LSLB               enthält den Offset für HEXCHS (binär)
        LSLB
        PSHS  B            B zwischenspeichern
        LDB   #16          Jetzt gelten nur Hexziffern
        BSR   GETCH        Nächstes-Zeichen
        ADDB  1,S+         Acht-Bit-Zahl zusammensetzen, zwischengespeichertes B löschen
        PULS  X,PC
```

Die PUTCR-Routine

```
PUTCR   PSHS  A            A sichern
        LDA   #13          ASCII-Code für Return
        BSR   OUTCH        Anzeigen
        PULS  A,PC
```



Die GETHX4-Routine

```
GETHX4  LDB   #16
        BSR   HX4          Höchstwertiges Byte holen
        PSHS  B            Höchstwertiges Byte zwischenspeichern
        LDB   #16
        BSR   HX4          Niederwertiges Byte in B holen
        PULS  A            Höchstwertiges Byte nach A
        RTS                Der Ergebniswert jetzt in D
```

Die Routine ‚Unterbrechungspunkt-anzeigen'

```
BPLABS  FCC   '1 2 3 4 5 6 7 8 9101112131415 16'
SPACE   FCB   32           ASCII-Code für Leerzeichen
DISPBP  PSHS  A,B,X,Y
        LEAX  BPTAB,PCR    Adresse für Unterbrechungstabelle
        LEAY  BPLABS,PCR   Adresse für Labels
        CLRB               Nächster-Unterbrechungspunkt auf Null-Offset setzen
WHIL01  CMBP  NUMBP,PCR    WHILE Nächster-Unterbrechungspunkt <= Zahl-der-Unterbrechungspunkte
        BGT   ENDW01
        LDA   ,Y+          Label anzeigen
        BSR   OUTCH
        LDA   ,Y+
        BSR   OUTCH
        LDA   SPACE,PCR    Leerzeichen anzeigen
        BSR   OUTCH
        PSHS  B            B zwischenspeichern
        LDD   ,X++         Adresse sichern
        BSR   DSPADD
        PULS  B            B zurückladen
        BRA   WHIL01
ENDW01  PULS  A,B,X,Y      Restore und Return
```

löschen löscht einen Unterbrechungspunkt aus der Tabelle, während Unterbrechungspunkt-zurücksetzen den SWI-Op-Code wieder durch den Befehl ersetzt, der ursprünglich an dieser Stelle stand. Mit diesen beiden Routinen beschäftigen wir uns in der nächsten Folge. Die letzte Routine stellt alle Unterbrechungspunkte dar:

Daten:

Zahl-Unterbrechungspunkt ist ein Acht-Bit-Zähler (in B), der die Punkte der Unterbrechungstabelle einzeln anspricht

Aktueller-Unterbrechungspunkt ist die Adresse, die dargestellt werden soll

Name-des-Unterbrechungspunktes ist eine zweistellige (Dezimal-)Zahl, die die Adressen bei der Darstellung mit einer laufenden Nummer versieht

Leerzeichen trennt den Namen von der Adresse

Ablauf 3: Unterbrechungspunkte-darstellen

```
Zahl-Unterbrechungspunkt auf 1 setzen
  (Offset von Null)
WHILE Zahl-Unterbrechungspunkt
<= Anzahl-der-Unterbrechungspunkte
  Name-des-Unterbrechungspunktes
    (Zahl-Unterbrechungspunkt)
    anzeigen
  Unterbrechungstabelle
    (Zahl-Unterbrechungspunkt)
    anzeigen
  Zahl-Unterbrechungspunkt inkrementieren
ENDWHILE
```

Ende von Ablauf 3

Die PUTHEX-Routine

```
PUTHEX  PSHS   A,B,X      Eingesetzte Register sichern
        PSHS   B          B zwischenspeichern
        LEAX   HEXCHS,PCR Adresse von HEXCHS in X
        LSRB              Vier Rechtsverschiebungen für
        LSRB              die vier höchstwertigen Bits
        LSRB
        LSRB
        LDA    B,X        Entsprechendes Zeichen in A setzen
        BSR    OUTCH      Anzeigen
        PULS   B          B zurückladen
        ANDB   #%00001111 Die vier höchstwertigen Bits
                          „ausmaskieren“
        LDA    B,X        Zweites Zeichen
        BSR    OUTCH
        PULS   A,B,X,PC   Restore und Return
```

Die Routine ‚Unterbrechungspunkt-einsetzen‘

```
BP01    PSHS   A,B,X,Y    Eingesetzte Register sichern
        LDX    BPTAB      Adresse der Unterbrechungstabelle
        LDA    NUMBP,PCR
IF01    CMPA   MAXBP,PCR  Wenn Zahl-der-Unterbrechungspunkte
                          < Max
        BGE    ENDF01
        INC    NUMBP,PCR  1 auf Zahl-der-Unterbrechungs-
                          punkte addieren
        LSLA              Das Offset für die 16-Bit-Tabelle
                          mit 2 multiplizieren
        LEAY   A,X
        BSR    GETADD     Adresse holen
        STD    ,Y         Adresse in der Unterbrechungstabelle
                          speichern
ENDF01  PULS   A,B,X,Y    Register wiederherstellen und Return
```

Das Ein- und Ausgabemodul

```
HEXCHS  FCC    '0123456789ABCDEF'
DOT     FCB    '.
RETURN  FCB    13         ASCII-Code für Return
COMNDS  FCC    'BUDSGRMQ'
GETCOM  PSHS   A,X        Inhalt von A und X sichern
        LEAX   COMNDS,PCR Adresse der Befehlszeichen in X
        LDB    #8         Zahl-der-Zeichen-OK
        BSR    GETCH      Zeichen holen
        PULS   A,X,PC     Return
GETADD  BSR    GETHX4
        BSR    PUTCR
        RTS
GETVAL  BSR    GETHX2
        BSR    PUTCR
        RTS
DSPVAL  BSR    PUTHEX
        BSR    PUTCR
        RTS
DSPADD  PSHS   B          B zwischenspeichern
        TFR    A,B        Höchstwertiges Byte in B
        BSR    PUTHEX
        PULS   B          B zurückladen
        BSR    PUTHEX
        PULS   B
        BSR    PUTHEX
        BSR    PUTCR
        RTS
```

Ausgewachsener Drache

Der Dragon 64 ist die erweiterte Version des 32er-Modells. Er verfügt über mehr Speicherkapazität und eignet sich ideal als Einsteiger-Gerät.

Der Dragon 64 ist mit der gleichen Tastatur ausgestattet wie der Dragon 32. Sie ist durchaus recht solide, hat aber keine Funktionstasten, auch keine Escape-, Tab- oder Control-Taste.

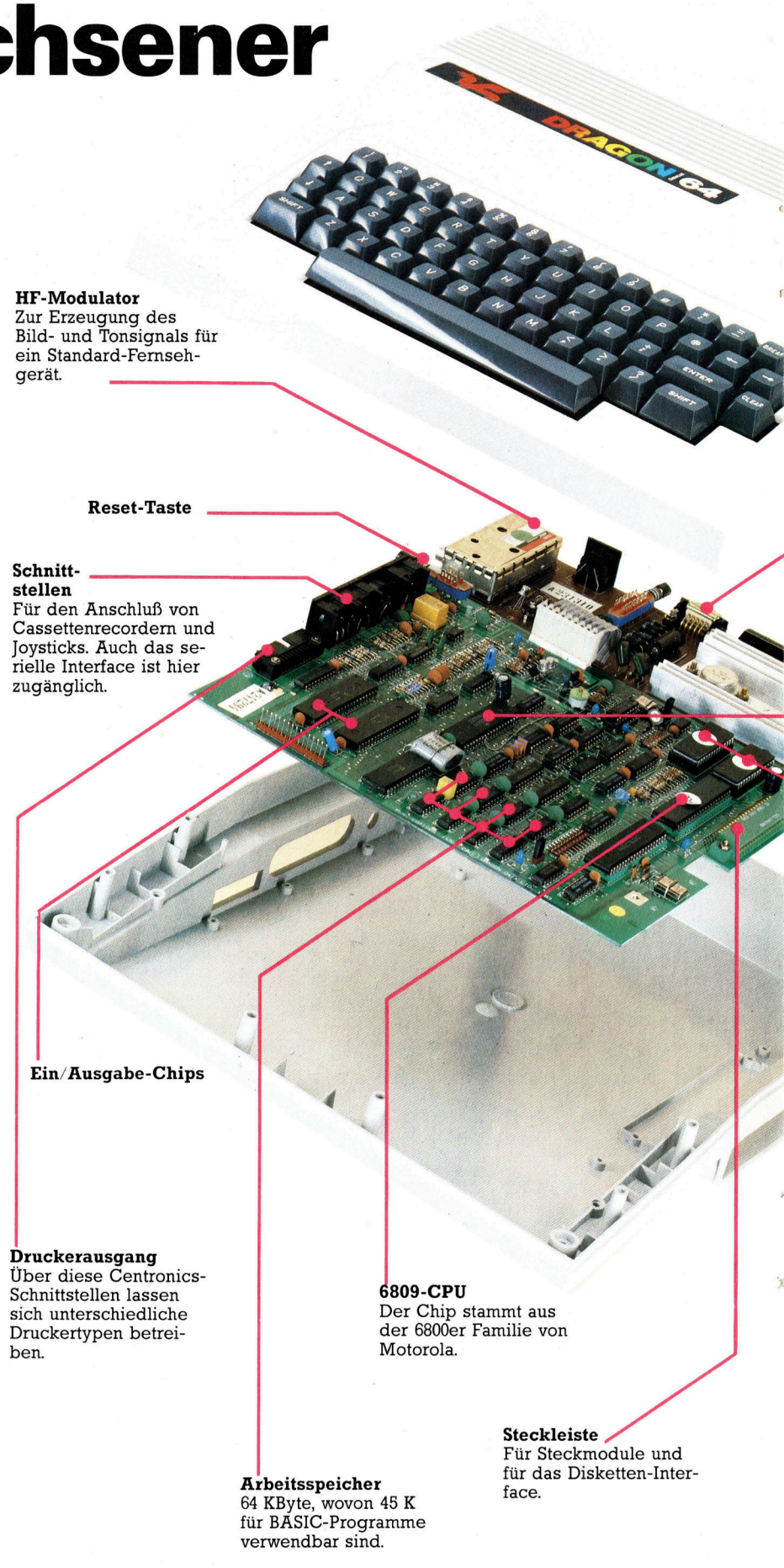
Nach dem Einschalten wird der Modus 32 aktiviert, in dem 30 KByte RAM zur Verfügung stehen. Mit dem Befehl EXEC 48000 wird das BASIC-ROM abgeschaltet, und damit sind die zusätzlichen 32 K RAM zugänglich. Der Interpreter wird dann ins RAM hochkopiert, so daß insgesamt 45 KByte frei verfügbar sind. Sie können allerdings nur bei Benutzung des OS9 oder bei Maschinencode-Programmierung angesprochen werden, denn bei Anschluß des Dragon-DOS-Steckmoduls wird automatisch wieder auf den 32er-Mode umgeschaltet – bei Diskettenbetrieb ohne OS9 sind daher auch beim 64er nur insgesamt 23 KByte für BASIC-Programme frei.

Platz für mehr RAM

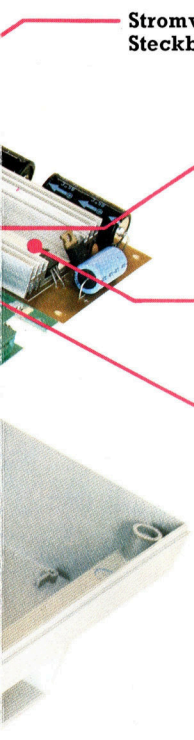
Die Leiterplatte ist beim erweiterten Modell nicht wiederzuerkennen, obwohl es sich im Prinzip um das gleiche Modell handelt; durch die veränderte Anordnung der Bausteine wurde Platz für die serielle Schnittstelle und die zusätzliche RAM-Kapazität geschaffen.

Als CPU dient der 8-Bit-Prozessor 6809 von Motorola, der aufgrund seines späten Erscheinens nicht die Verbreitung wie der 6502 und der Z80 gefunden hat. Darauf abgestimmt ist der Video-Chip 6847 für die Bilderzeugung via Fernsehapparat oder Composite-Video-Monitor.

Der 6847 ist dafür verantwortlich, daß im Textmodus nur 16 Zeilen zu 32 Zeichen auf dem Bildschirm dargestellt werden können (auf grünem oder orangefarbenem Grund). Beim Dragon 32 war das noch hinzunehmen, aber wer den 64er beispielsweise für die Textverarbeitung einsetzen will, ist damit doch stark eingeengt. Auch der Gebrauch der Diskettensoftware in Verbindung mit dem OS9-Betriebssystem unterliegt aus diesem Grund erheblichen Einschränkungen.



Dragon-Diskettenstation
Die Diskettenstation ist mit ein oder zwei 175-KByte-Laufwerken erhältlich. Mitgeliefert wird ein Interface-Steckmodul, das das DOS enthält.



Stromversorgungs-Steckbuchse

Video-Steuerbaustein 6847
Wahlweise kann ein Fernsehgerät oder ein Composite-Video-Monitor angeschlossen werden.

Kühlkörper
Dient zur Abführung der Verlustwärme des Spannungsreglers.

BASIC-EPROMs
Sie enthalten den 16 KByte umfassenden Microsoft-BASIC-Interpreter.

Der Dragon 32 ist eine „verkleinerte" Version des 64er-Modells.

Die Grafikauflösung reicht von 128 × 96 Punkten (vierfarbig) bis zu 256 × 192 Punkten (zweifarbig). Das ist im Vergleich zu anderen Rechnern nicht gerade umwerfend, zumal auch die Wiedergabequalität teils sehr zu wünschen übrig läßt. Über spezielle Software läßt sich jedoch im Grafikmode eine 24zeilige Schriftausgabe mit 51 Zeichen je Zeile erzeugen, wodurch auch Tabellenkalkulation und Textverarbeitung durchgeführt werden kann.

Extended BASIC

Der Dragon ist mit dem 16-KByte-Microsoft-Extended-BASIC ausgestattet, das auch über leistungsfähige Sound- und Grafikbefehle verfügt. Sie können damit eine größere Anzahl von Grafiken parallel im Speicher halten. Natürlich kann man bei Bedarf den Grafik-Speicherbereich auch für Programme nutzen.

Der Dragon hat alle erforderlichen Schnittstellen, eingeschlossen zwei Analog/Digital-Wandler und ein Centronics-Interface, an das die meisten Paralleldrucker anschließbar sind. Die serielle Schnittstelle ist ebenfalls für Drukker und ähnliche Peripherie, aber auch für die Vernetzung gedacht.

Ferner können Sie am Dragon einen Standard-Cassettenrecorder mit Bandlauf-Fernsteuerung vom Programm her betreiben; die Tonwiedergabe ist über den Lautsprecher des Fernsehgeräts möglich. Das BASIC enthält unter anderem mehrere Befehle für die Handhabung von Cassetten-Dateien. Die Dragon-Diskettenstation wird mit einem zusätzlichen ROM-Betriebssystem und zusätzlichen BASIC-Befehlen geliefert.

Der Dragon 64 läßt sich wegen der kargen Textwiedergabe und Tastaturausstattung kaum zu einem ernsthaften Bürosystem ausbauen, ist aber für den Computer-Neuling geeignet.

Mehrprogramm/Mehrplatz-Betriebssystem OS9
Der 6809-Prozessor des Dragon erlaubt in Verbindung mit 64 K RAM und Diskettenlaufwerken den Einsatz des OS9-Systems, das speziell für den 6809 entwickelt wurde und über weitreichende Möglichkeiten verfügt.

Das OS9 erlaubt ähnliche Techniken wie UNIX, das bis zu den größten Bürorechnern hin Verwendung findet. So ist die gleichzeitige Bearbeitung mehrerer Programme (Multi-Tasking) und der simultane Betrieb mehrerer Terminals an einem Rechner (Mehrplatzbetrieb) vorgesehen. Programm- und Datenfiles werden unter OS9 in einer festen Struktur organisiert und nicht hintereinander auf Diskette abgelegt. Mit Paßwörtern und Zugriffscodes können die Dateien gegen unbefugte Benutzung gesichert werden.

Die meisten privaten Benutzer werden diese Möglichkeiten kaum wahrnehmen; der kleine Dragon-Rechner käme auch an seine Grenzen, wenn man das OS9 voll ausreizen wollte. Immerhin bietet sich hier eine preisgünstige Gelegenheit, mit einem raffinierten Betriebssystem zu experimentieren. Außerdem unterstützt das OS9 anspruchsvolle Programme und Computersprachen wie PASCAL und C.

Dragon 64

ABMESSUNGEN
380×330×90 mm

CPU
6809

SPEICHERKAPAZITÄT
64 K RAM, davon bis zu 45 K für BASIC-Programme verfügbar; 16 K ROM

BILDSCHIRM-DARSTELLUNG
Bei Textwiedergabe 16 Zeilen zu 32 Zeichen. Auflösung im Grafikmodus: 128×96 Punkte (vierfarbig) oder 256×192 Punkte (zweifarbig)

SCHNITTSTELLEN
Anschlüsse für zwei Joysticks, serielles Interface, Parallelschnittstelle, Composite-Video-Monitor und Fernseher; Modul-Steckplatz

DISKETTENLAUFWERKE
Maximal zwei 175-KByte-Laufwerke, Betriebssystem: Dragon-DOS oder OS9

SPRACHEN
BASIC, FORTH, 6809-Assembler; mit OS9 auch PASCAL, C und strukturiertes BASIC

TASTATUR
Schreibmaschinentastatur mit 53 Tasten

DOKUMENTATION
Leider ist das Dragon-Handbuch lückenhaft und enthält nur die allernötigsten Informationen.

STÄRKEN
Der Dragon 64 stellt mit seiner großen Speicherkapazität und der Vielzahl von Schnittstellen ein gutes System für Einsteiger dar.

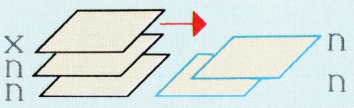
SCHWÄCHEN
Weniger glücklich sind die knappe Tastaturausstattung und die Bildschirmdarstellung. Für einige der OS9-Anwenderprogramme ist der 6809-Prozessor mit 1 MHz-Takt zu langsam.

Rechnen mit UPN: 2+3 gleich 23+

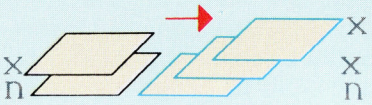
Gesteuerter Stapel

Es ist nicht immer möglich, den Stapel so aufzubauen, daß alle Zahlen in der richtigen Reihenfolge stehen. FORTH bietet daher eine Reihe von Wörtern, die die Werte des Stapels umstellen. Hier ein paar Beispiele:

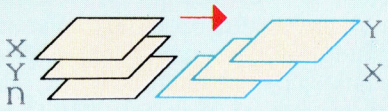
- **DROP** x–– (löscht den obersten Stapelwert)



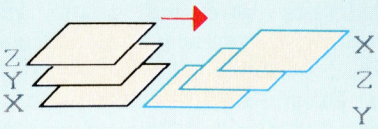
- **DUP** x––x, x (dupliziert den obersten Stapelwert)



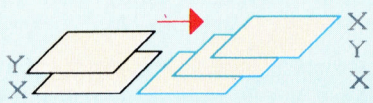
- **SWAP** x, y––y, x (tauscht die beiden obersten Stapelelemente gegeneinander aus)



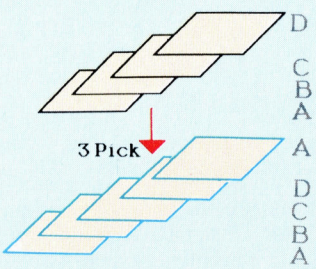
- **ROT** x, y, z –– y, z, x (rotiert die ersten drei Stapelelemente, so daß die zuunterst liegende Zahl oben liegt)



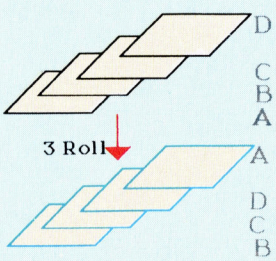
- **OVER** x, y –– x, y, x (kopiert die zweitunterste Nummer und legt sie oben auf den Stapel)



- **PICK** xn, ... x2, x1, x0, n––xn, ..., x2, x1, x0, xn (wie OVER. Sie müssen jedoch eine zusätzliche Zahl (n) angeben, die besagt, welche Zahl kopiert werden soll. 1 PICK entspricht beispielsweise OVER, 2 PICK kopiert die drittunterste Zahl, während 0 PICK wie DUP arbeitet)



- **ROLL** xn, ..., x2, x1, x0, n –– ..., x2, x1, x0, xn (wie ROT, aber mit einer zusätzlichen Zahl wie PICK. 2 ROLL entspricht ROT, während 3 ROLL vier Zahlen rotiert. 1 ROLL entspricht SWAP).



In FigFORTH fehlen PICK und ROLL. In FORTH-79 sind sie enthalten, doch spricht die Zusatzzahl einen um Eins höheren Wert an, als in FORTH-83. In FORTH-79 entspricht 3 ROLL daher ROT.

In dieser Folge untersuchen wir die Logik der „Umgekehrten Polnischen Notation" (UPN) und wollen zeigen, wie diese Schreibweise mathematische Abläufe und die Bearbeitung von Daten vereinfacht.

Es ist selbstverständlich, daß wir die Summe zweier Zahlen nur erhalten können, wenn wir beide Zahlen kennen. Bei der Eingabe von 2+3 (das "+" steht in der Mitte), wartet ein Computer daher erst auf die zweite Zahl, bevor er sich mit dem + beschäftigt. Die Maschine versteht die Aufgabe als 2 3 +.

Wir sind daran gewöhnt, mathematische Operatoren wie +, –, * und / in der Mitte („Infix Notation") zu schreiben (zum Beispiel 2+2). Die Schreibweise entspricht der Umgangssprache (zwei plus zwei ergibt vier) und trennt die beiden Operanden deutlich voneinander.

Die Infix Notation bringt bei Spracherweiterungen jedoch große Nachteile. Zunächst muß der Operator exakt zwei Operanden (Argumente oder Parameter) haben – auf jeder Seite einen. Werden mehrere Operanden gebraucht, muß eine funktionelle Schreibweise wie FNf(a, b, c, d, e) verwandt werden. Weiterhin sind Ausdrücke wie 2+3*5 nicht eindeutig: Wird hier das + zuerst ausgeführt, oder das *? Das Problem läßt sich nur durch zusätzliche Regeln lösen, die beispielsweise besagen, daß * eine höhere Priorität hat als +. Wenn Sie bei der Infix Notation neue Operatoren einführen wollen, müssen Sie daher auch die Regeln erweitern.

FORTH bietet hier eine Lösung, die den Fähigkeiten des Computers am besten entspricht: Bei allen Operatoren oder Funktionen (also allen Wörtern) werden zuerst die Operanden geschrieben und dann das Wort (beispielsweise 2 3 +). Die Schreibweise funktioniert wie ein Kochrezept: Zuerst holen Sie alle Zutaten, und dann verarbeiten Sie diese. Die technische Bezeichnung dieser Methode lautet „Umgekehrte Polnische Notation" (UPN). In der Direkten Polnischen Notation steht der Operator vor den Operanden. LOGO verwendet diese Schreibweise bei neuen Funktionen.

Nehmen wir an, Sie haben einen Ausdruck mit Infix Notation wie 2*3+8/4. Da nach den üblichen Prioritätsregeln das + zuletzt ausgeführt wird, ist das Endergebnis die Summe von 2*3 und 8/4. Für die UPN-Schreibweise wird

der Ausdruck umgeformt:
(2*3) (8/4) +

Hier haben jedoch 2*3 und 8/4 noch die Infix Notation. Die zweite Umformung ergibt daher:
(2 3 *) (8 4 /) +

Die UPN braucht keine Klammern, da sie in jedem Fall eindeutig ist. FORTH setzt Klammern außerdem nur für Kommentare ein. Hier das endgültige UPN-Format von 2*3+8/4:
2 3 * 8 4 / +

Bedenken Sie, daß in FORTH die Leerzeichen unbedingt notwendig sind.

Die Umgekehrte Polnische Notation kann also die beiden Probleme der Infix Notation lösen: Bei der UPN sind pro Operator nicht nur zwei Operanden möglich, sondern mehrere (gefolgt von einem Operator). Beispielsweise hat in FORTH der Operator */ drei Operanden: Er multipliziert die ersten beiden und dividiert das Ergebnis durch den dritten Operanden.

Das zweite Problem der Infix Notation sind die Prioritätsregeln und Klammern, durch die die Reihenfolge der Einzelberechnungen festgelegt wird. Die UPN kommt ohne Regeln und Klammern aus – sie teilt dem Computer exakt mit, wie das Ergebnis berechnet wird.

Ergebnis im Speicher

Sehen wir uns an dem einfachen Beispiel 2 3 + einmal an, wie die UPN ausgeführt wird. FORTH trifft zuerst auf die 2 und speichert sie. Danach findet es die 3, die ebenfalls gespeichert wird. Bei + nimmt der Computer an, daß er sich bereits zwei Zahlen „gemerkt" hat. Er findet sie, addiert sie und legt das Ergebnis wieder im Speicher ab. Dort steht es für die weitere Bearbeitung zur Verfügung.

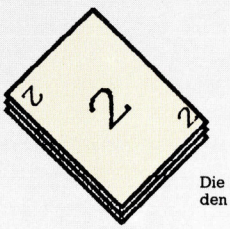
In der letzten Folge erwähnten wir kurz, daß FORTH sich an Zahlen „erinnert", indem es diese auf den Stapel schiebt (nach der LIFO-Methode – Last In First Out). Dieses Konzept ist Maschinencodeprogrammierern sicherlich vertraut, doch kann es für Anfänger schon recht ungewohnt sein. Wir geben daher hier ein weiteres Beispiel. Stellen Sie sich einen Stapel Spielkarten vor. Um eine Nummer zu „speichern", schreiben Sie sie auf eine neue Karte, die Sie oben auf den Stapel legen. Für das + nehmen Sie die beiden obersten Karten herunter und addieren die beiden darauf notierten Zahlen, verändern jedoch nicht den Rest des Stapels. Der Kasten zeigt die Entwicklung des Stapels bei der Bearbeitung von 2 –3 * 8 4 /+ ((2 * –3) + 8 / 4)). Beachten Sie, daß in Schritt 6 die Karte –6 von der Teilung 8 / 4 nicht berührt wird.

Zwar zeigt dieser Ablauf, wie der Stapel mit Ganzzahlen funktioniert, in FORTH werden Variablen jedoch als Adressen und nicht als Ganzzahlen behandelt. Die Adressen werden

Von Oben nach Unten

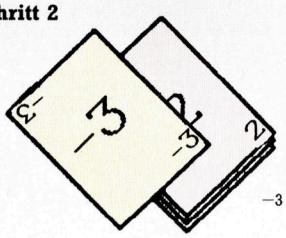
Hier ein weiteres Beispiel, wie bei der Ausführung von 2 –3 * 8 4 / + (Infix Notation: 2* –3 + 8/4) Zahlen auf den Stapel „geschoben" und „heruntergezogen" werden.

Schritt 1



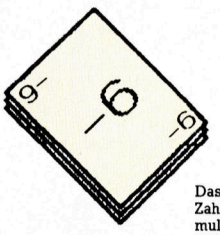
Die Zahl 2 wird zuoberst auf den Stapel gelegt...

Schritt 2



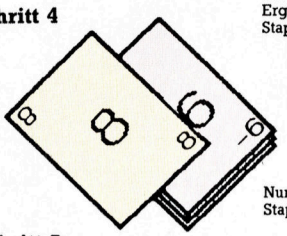
–3 wird auf die 2 gelegt

Schritt 3



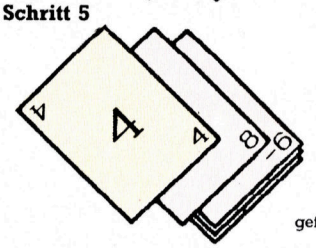
Das Wort * zieht die beiden Zahlen vom Stapel herunter, multipliziert sie und legt das Ergebnis wieder auf den Stapel.

Schritt 4



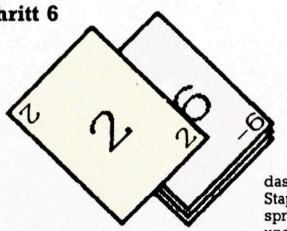
Nun wird 8 zuoberst auf den Stapel gelegt

Schritt 5



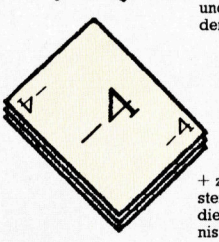
gefolgt von 4

Schritt 6



das / nimmt zwei Zahlen vom Stapel (–6 wird nicht angesprochen), verarbeitet sie und legt das Ergebnis wieder zuoberst auf den Stapel.

Schritt 7



+ zieht nun die beiden obersten Zahlen vom Stapel, addiert sie und legt das Ergebnis wieder auf den Stapel.

eingesetzt, um (mit @, dem Befehl für „holen") die Variablenwete anzugeben oder (mit !, dem Befehl für „speichern"), um geänderte Werte auf die Adressen zu schreiben.

Unser Beispiel zeigt deutlich, wie exakt die Umgekehrte Polnische Notation arbeitet. FORTH unterhält eine eigene Version eines „Kartenstapels", so daß der Computer bei jedem Wort die entsprechenden Abläufe ausführen kann, ohne sich darum kümmern zu müssen, was zuvor oder danach passiert.

Hier ein weiteres Beispiel für den Einsatz des Stapels: Das Wort 2* ersetzt den letzten Wert des Stapels durch den doppelten Wert dieser Zahl. Es ist folgendermaßen definiert:

```
:2* (n —— n*2)
2 *
;
```

Beachten Sie, daß Klammern nur Kommentare umschließen. (Das Symbol —— wird in dem nebenstehenden Kasten erklärt.) Hier ein Beispiel für das neue Wort 2*:

```
19 2*.
```

zeigt als Ergebnis 38. Das Symbol . ist das FORTH-Wort für PRINT. Es nimmt den obersten Wert des Stapels und zeigt ihn auf dem Bildschirm an. Das *-Zeichen in der Definition von 2* erwartet mindestens zwei Zahlen. Die zweite Zahl ist dabei die in der Definition enthaltene 2, während sich die erste (in unserem Beispiel 19) bereits auf dem Stapel befinden muß, wenn 2* eingesetzt wird.

Stapelvorgänge dieser Art haben den Vorteil, daß sie mehr als ein Ergebnis erzeugen können. So beläßt das Wort /MOD beispielsweise zwei Werte auf dem Stapel: den „Quotienten" (das Ergebnis) und den „Rest" einer Division. Mit —— läßt sich dieser Vorgang aber auch folgendermaßen ausdrücken:

```
m, n —— Rest, Quotient von m/n
```

Bei der Infix Notation wäre dieser Vorgang nicht möglich, bei der UPN ist er normal.

Die direkte Programmierung ist zwar eine der beeindruckendsten Eigenschaften von FORTH, kann aber auch Probleme verursachen. So stehen die Zahlen des Stapels zuweilen nicht in der richtigen Reihenfolge und müssen umorganisiert werden. Der nebenstehende Kasten enthält einige Standardwörter, die für diesen Zweck eingesetzt werden. Das nachfolgend definierte Wort */ ist zwar nicht so gut wie die Standarddefinition (die auch die richtige Antwort bietet, selbst wenn das Ergebnis zu groß für den Stapel ist), zeigt aber, wie ROT und SWAP funktionieren:

```
: */ (x,y,z——x*y/z)
  ROT ROT *
  SWAP /
;
```

Hier der Ablauf am Beispiel von 4 3 6 */:

Vorgang */	Stapel (oben → unten)
Am Anfang von */	4,3,6
ROT	3,6,4
ROT	6,4,3
*	6,12
SWAP	12,6
/	2
	Leer (2 wird dargestellt)

Normalerweise werden die Befehle zur Stapelmanipulation jedoch nur im Inneren von Wortdefinitionen eingesetzt.

Vorsicht

Sie müssen sorgfältig darauf achten, genau die Anzahl an Werten auf den Stapel zu schieben, die das Wort benötigt. Wenn Sie zu wenig anlegen, setzt das Programm selbständig Werte ein. Wurden zu viele eingesetzt, dann stimmt das Ergebnis ebenfalls nicht.

Für die genaue Wort-Definition können Sie die Schreibweise "——" einsetzen. Dabei wird die Liste der Werte, die ein Wort braucht, durch "——" von den Werten getrennt, die es auf dem Stapel zurückläßt. Hier einige Beispiele:

Wort:	Auswirkung:
+	m,n———m+n
.	m ——
*/	x, y, z——x*y/z

Die Schreibweise "——" ist kein Bestandteil von FORTH, sondern soll nur die Schreibweise übersichtlicher machen. Bei der Verwendung von Klammern sollten Sie beachten, daß nach der ersten Klammer ein Leerzeichen stehen muß. Die Klammer ist ein Wort, das besagt: Ignoriere von hier an alles, bis eine geschlossene Klammer auftaucht.

Abfrage des Stapels

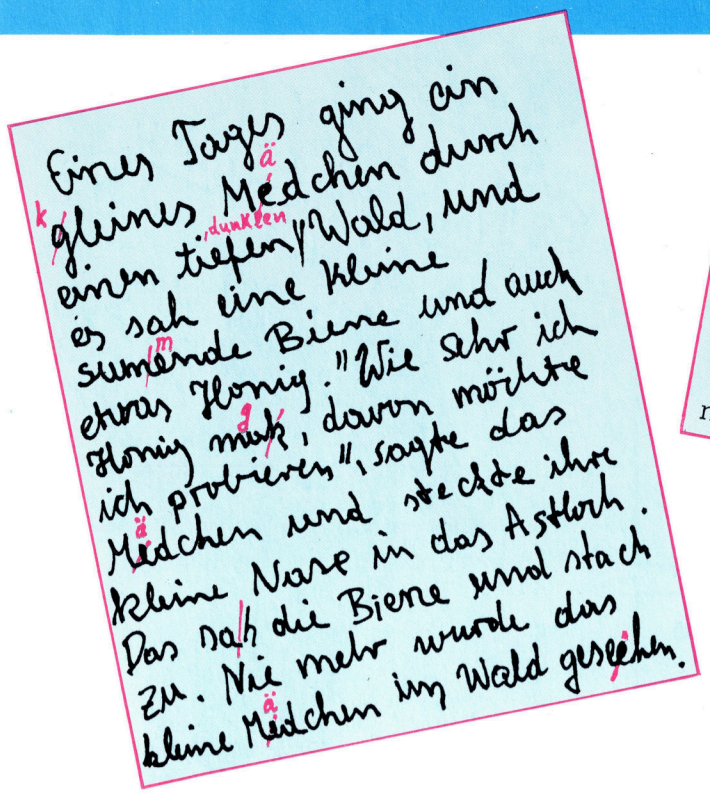
Ein sehr praktisches Wort ist .S, das den Stapelinhalt anzeigt. Es ist im Standard-FORTH zwar nicht enthalten, in FORTH-83 aber definiert als:

```
: .S (——)
    (zeigt den Stapel von unten nach oben
      an)
    0 DEPTH 1 — DO
      I PICK.
    —1 + LOOP
;
```

In FORTH-79 funktioniert PICK anders. Die dritte Zeile muß dann lauten

```
  1 DEPTH DO
```

DEPTH (—— Stapeltiefe) informiert Sie darüber, wie viele Zahlen auf dem Stapel abgelegt waren, bevor Sie DEPTH aufgerufen haben. Da FigFORTH jedoch weder über DEPTH noch über PICK verfügt, funktionieren diese Definitionen dort natürlich nicht.



Eines Tages ging ein kleines Mädchen durch einen tiefen, dunklen Wald, und es sah eine kleine summende Biene und auch etwas Honig. "Wie sehr ich Honig mag, davon möchte ich probieren", sagte das Mädchen und steckte ihre kleine Nase in das Astloch. Das sah die Biene und stach zu. Nie mehr wurde das kleine Mädchen im Wald geseehen.

Eines der Hauptprobleme, mit denen Kinder konfrontiert werden, ist die Unzulänglichkeit ihrer Handschrift. Denn sie hält meist nicht mit den Gedanken Schritt. Beim Versuch, Briefe richtig zu schreiben, gehen Gedanken nur allzuoft verloren. Aus diesem Grunde wirkt die Handschrift von Kindern oft so zerrissen. Textverarbeitung allein kann dieses Problem nicht lösen: Es dauert ebenso lange, eine Taste zu finden, wie einen Buchstaben zu Papier zu bringen. Doch mit guten Schreibmaschinen-Fähigkeiten kann ein Kind dieses Problem bewältigen.

Schriftbilder

In dieser Folge unserer Serie über englische Ausbildungsprogramme beschäftigen wir uns mit dem Einsatz von Micro-Computern beim Schreiben- und Zeichnenlernen. Hierbei handelt es sich um eine entscheidende, zugleich kontroverse Entwicklung. Beide Fähigkeiten beinhalten ausgeprägte kreative Elemente. Deshalb scheint es angebracht, die Verwendung von Computern in diesen Bereichen mit Vorbehalten zu betrachten.

Häufig stehen die Menschen technischen Neuerungen kritisch gegenüber und weigern sich, neuen Ideen zu folgen, bis diese sich als erfolgreich bewiesen haben. Das Übungsheft, eines der grundlegenden Ausbildungsmittel von heute, stellte eine revolutionäre Neuerung dar, als es die Schiefertafel ersetzte. Zuvor war Papier für geschäftliche Transaktionen, für wichtige Arbeiten und Dokumente bei Universitäten und Behörden verwendet worden. Da es teuer war, hielt man seine Benutzung durch Kinder für Verschwendung, die – so die Meinung – ohnehin nicht wußten, wofür es zu benutzen sei und allzu großzügig damit umgingen.

Die Einführung von Übungsheften gab den Schülern eine kontinuierliche Übersicht ihrer Arbeit. Sie konnten sich, ohne vorherige Aufgaben zu löschen, mit unterschiedlichen Aufgaben beschäftigen. Diese Neuerung war teuer, aber nutzbringend. Was nun jedoch die ökonomischen Gesichtspunkte des Einsatzes von Kleincomputern bei der Ausbildung anbelangt, so gibt es ähnliche Vorbehalte wie seinerzeit bei den Übungsheften. Aus Kostengründen scheint das Konzept „Für jedes Kind einen Computer" nicht tragbar; denn ein bestimmender Faktor der Ausbildungspolitik waren schon immer die Kosten.

Technikfeindlichkeit

Dazu kommt, daß viele Lehrer die neue Technik ablehnen, da sie durch das „Programmierte Lernen" bereits negativ eingestellt sind. Bei näherer Betrachtung des Computers im Klassenzimmer werden jedoch viele Vorurteile zerstreut, nicht allein deshalb, weil der Computer Effizienz beweist und so die reinen Kosten reduziert, sondern auch, weil er wertvolle Beiträge zu kreativem Lernen leistet.

Betrachten wir zunächst einmal die Anwendungsmöglichkeiten des Micros bei der Entwicklung der Schreibfähigkeiten. Hat man ein

Der Einsatz eines Computers bei Textverarbeitung und Zeichnen fordert dieselben Fähigkeiten wie das Schreiben und Zeichnen von Hand. Doch richtig benutzt kann der Computer viele Fähigkeiten besser vermitteln oder erweitern, als es mit den herkömmlichen Lehrmethoden möglich ist, er trägt sogar zur Entwicklung zusätzlicher Fähigkeiten bei. Das Diagramm gibt eine Übersicht der kognitiven Fähigkeiten, die Computer lehren oder verbessern.

Schreib- (Er)-Kenntnisse

- Feinmotorik (Finger- und Handbewegungen)
- Buchstabenerkennung
- Buchstabenform (das Zeichenformat auf dem Bildschirm und dem Drucker vermittelt die richtige Art des Buchstabenschreibens)
- Korrekturlesen und Fehlerbeseitigung
- Rechtschreibung (eine Fülle spezifischer Fähigkeiten)
- Gliederung (Ereignisse in die richtige Reihenfolge bringen)
- Visuelles Gedächtnis (Erlernen der Tastenposition, um nicht ständig auf die Tastatur blicken zu müssen)

Grafische (Er)-Kenntnisse

- Augen-Hand-Koordination
- Feinmotorik
- Mustererkennung und -erstellung
- Konturenveränderungen
- Ursachen-Wirkungs-Relation
- Geläufigkeit (rasches Entwickeln vieler Ideen)
- Flexibilität (Querdenken: bekannte Dinge in neuem Licht sehen)
- Einbildungskraft/Phantasie

Konzept gefunden, muß eine Geschichte oft umgeschrieben, überarbeitet und verbessert werden, ehe sie vollendet ist. In diesem Sinne ist das Schreiben dem Malen und der Bildhauerei sehr ähnlich.

Leider ist der Lernprozeß, mit dem viele Kinder zum Schreiben ausgebildet werden sollen, dem häufig völlig entgegengerichtet. Eine Hauptforderung der Lehrer ist die schnellstmögliche Beendigung der Arbeit, die zudem sauber – ohne Streichungen und Verbesserungen – vorgelegt werden soll. Aufgrund dieser Forderung werden das Überarbeiten, Neuschreiben und somit künftig ernsthaft kreatives Schreiben auf dem Vorwege nicht gefördert, sondern eher behindert.

Erlaubt man einem Kind, einen Computer für die Textverarbeitung zu benutzen, so kann es seine Arbeit problemlos korrigieren und verbessern und liefert schließlich einen sauberen Ausdruck ab. Der Text kann ebenso auf einer Diskette gespeichert werden, die das Kind dem Lehrer überreicht, der sie wiederum in seinem Computer zu Hause oder in der Schule benutzt, um die Arbeit in Ruhe durchzugehen. Bei Verwendung eines Computer-Netzwerks kann der Lehrer die Files der Kinder direkt aufrufen und die Arbeit korrigieren. Das Endergebnis ist ein sauber bedrucktes Stück Papier, auf das auch der sonst unbeholfenste Schreiber stolz sein kann.

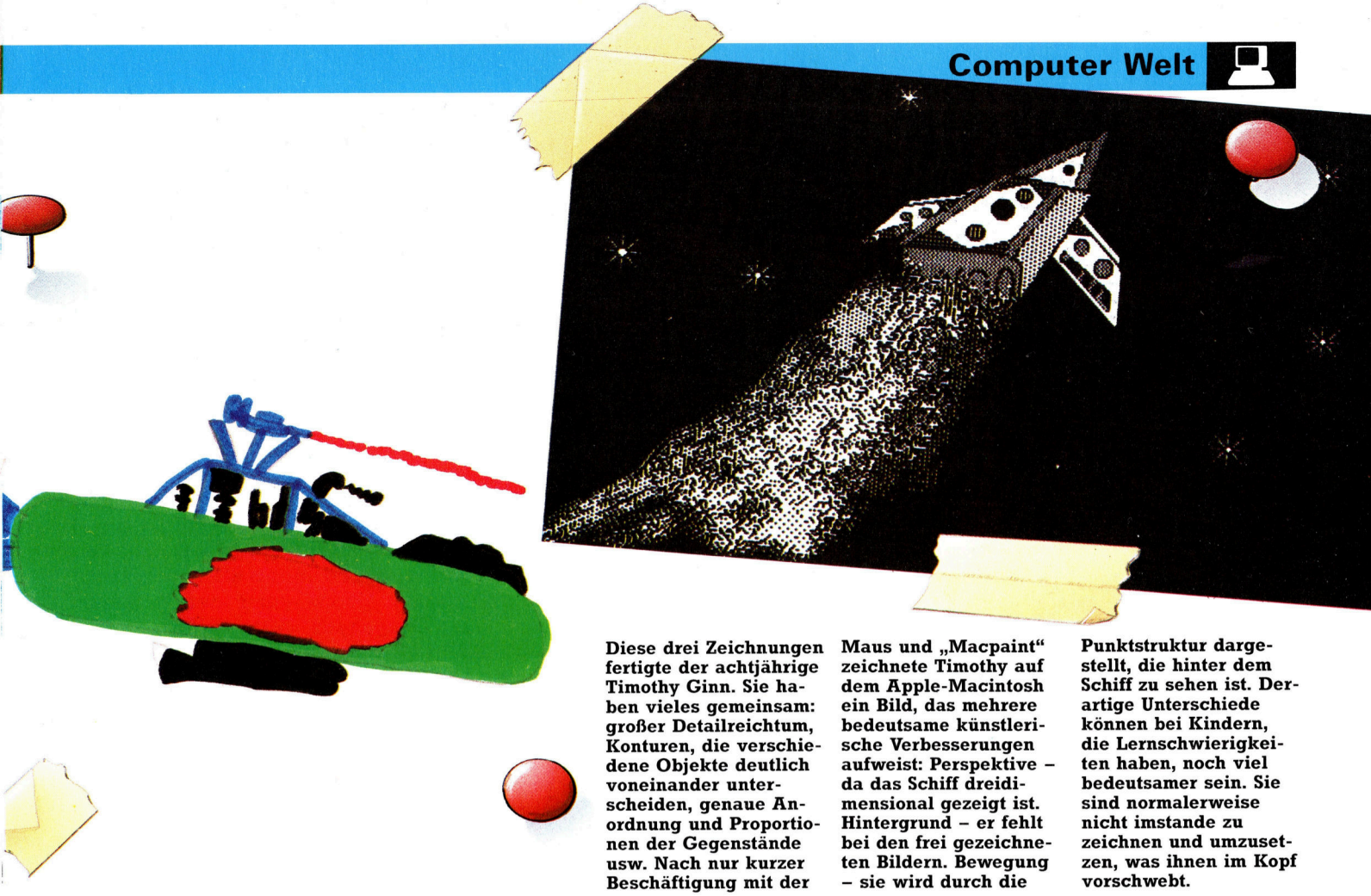
Ein möglicher Nachteil dieses Verfahrens ist, daß ein Kind sich – wenn es den korrigierten Text sieht – nicht in dem Maße seiner Fehler bewußt wird, wie es in einem korrigierten Übungsheft der Fall wäre.

Ein Kind, das an einem Computer ausgebildet wird, hat – sagen die Gegner – das „Hindernis Tastatur" zu überwinden. Vom „Hindernis Bleistift" aber ist nie die Rede. Eine der Hauptaufgaben von Lehrern an Grundschulen besteht darin, den Kindern das Schreiben beizubringen. Zunächst müssen diese die richtige Haltung des Stiftes lernen, dann die Konturen der Buchstaben und schließlich die Reproduktion dieser Konturen auf Papier beherrschen. Das Kind muß Wörter in eine einheitliche Größe bringen und darauf achten, daß die Wörter gerade geschrieben sind. Es dauert Jahre, um all diese Fähigkeiten zu beherrschen, und viele Kinder lernen genaues und richtiges Schreiben nie.

Handschrift oder Tippen

Vergleichen wir einmal diesen Vorgang mit dem Tippenlernen. Man muß die unterschiedlichen Buchstabenformen erlernen, ihre Lage auf der Tastatur und dabei die Schreibmaschinentasten mit den entsprechenden Fingern bedienen. Das ist wesentlich leichter, obwohl viele Leute zu Recht argumentieren könnten, daß die Herausforderung des Schreibenlernens wertvoll sei und deshalb nicht ignoriert werden dürfe.

Natürlich ist Textverarbeitung kein Ersatz für



Diese drei Zeichnungen fertigte der achtjährige Timothy Ginn. Sie haben vieles gemeinsam: großer Detailreichtum, Konturen, die verschiedene Objekte deutlich voneinander unterscheiden, genaue Anordnung und Proportionen der Gegenstände usw. Nach nur kurzer Beschäftigung mit der Maus und „Macpaint" zeichnete Timothy auf dem Apple-Macintosh ein Bild, das mehrere bedeutsame künstlerische Verbesserungen aufweist: Perspektive – da das Schiff dreidimensional gezeigt ist. Hintergrund – er fehlt bei den frei gezeichneten Bildern. Bewegung – sie wird durch die Punktstruktur dargestellt, die hinter dem Schiff zu sehen ist. Derartige Unterschiede können bei Kindern, die Lernschwierigkeiten haben, noch viel bedeutsamer sein. Sie sind normalerweise nicht imstande zu zeichnen und umzusetzen, was ihnen im Kopf vorschwebt.

die Handschrift, doch das spätere Leben diktiert immer mehr die Benutzung einer Tastatur, sowohl im Beruf als auch in anderen Situationen des täglichen Lebens. Maschineschreiben wird jungen Leuten oft später in Berufsschulkursen vermittelt. Das Erlernen dieser Fähigkeit bereits auf der Schule wäre daher ein Vorteil. Mit dem Vormarsch der Computer und ihrer immer breiteren Anwendung wird sich dieser Trend sicherlich fortsetzen.

Ein anderer wichtiger Aspekt, unter dem Microcomputer bei der schulischen Ausbildung zu sehen sind, ist seine überlegene Möglichkeit der grafischen Darstellung. Diese interessante Anwendungsmöglichkeit der Micros kam bisher aus Kostengründen noch nicht in vollem Umfang zum Tragen, da Grafikverarbeitung ausgefeiltere und höher entwickelte Hard- wie Software voraussetzt. Doch die Wirtschaftlichkeitsrechnung einer Schule ändert sich, wenn die Rechner gleichzeitig leistungsfähiger und preiswerter werden.

Zunächst erlaubten die Grafik-Tabletts oder Maltafeln unterschiedliche künstlerische Funktionen, die auf dem Monitor dargestellt werden konnten. Deren Vorteile sind in die „Maus-Technologie" integriert. Durch Wahl von „Piktogrammen" hat der Benutzer exakte und sofortige Kontrolle über eine Fülle von Aktivitäten, die vom Bilderzeichnen und Vergrößern und Verkleinern von Abbildungen über das Ausfüllen von Flächen bis hin zu allen nur vorstellbaren Techniken reicht.

Überdies erlaubt der Computer dem Schüler, anderes Arbeitsmaterial zu integrieren (etwa Text und Grafik) und miteinander zu verbinden. Dies ist leichter, als es mit herkömmlichen Methoden möglich ist. Ein fertiges Bild kann „ausgeschnitten" und an beliebiger Stelle in einen zuvor erstellten Text „geklebt" werden. Wenngleich Grafikprogramme das künstlerische Arbeiten im Klassenzimmer nicht ersetzen können, so bieten sie doch ein neues Medium mit neuen Techniken, an dem Kinder experimentieren, lernen und sich ausdrücken können.

Kinder wollen lernen – ihre ständige Neugier veranlaßt sie, jede wache Stunde zum Suchen, Forschen und Lernen zu nutzen, ihre Umwelt zu begreifen und ihre Eindrücke einem kreativen Impuls folgend auch auszudrücken. Sie greifen nach Bleistift und Kreide, nach Papier oder der nächst erreichbaren Wand, auf der sie malen können – und so werden diese fundamentalen Fertigkeiten entwickelt.

Zweijährige, die Zugang zu einem Computer haben, machen ihre Entdeckungen daran mit ebensoviel Freude, indem sie alle Tasten drükken und sehen, wie Muster auf dem Bildschirm entstehen. Sie akzeptieren die neue Technik leichter als Erwachsene, und ein Kind benutzt Computer ganz selbstverständlich. Es ist lediglich eine Frage der Zeit, bis Schulen den Computer als wichtiges Erziehungsmittel in die Ausbildung einbeziehen, statt sie nur als teures Spielzeug zu betrachten.

Neue Zeichen

Nachdem Sie gesehen haben, wie man auf dem C 64 eigene Zeichen definiert, zeigen wir Ihnen hier zwei Programme für den Acorn B und den Spectrum. Beide Computer verfügen über einen speziellen Speicherbereich für frei definierbare Zeichen.

Beim Spectrum stehen dem Anwender 168 Byte Speicher für eigene Zeichendefinitionen zur Verfügung, die vom Betriebssystem reserviert werden (man kann sie jedoch auch für Maschinenroutinen oder Daten verwenden). In diesem Bereich können 21 Zeichendefinitionen gespeichert werden, und das Betriebssystem ordnet ihnen CHR$-Werte im Bereich von 144 bis 164 zu. Die definierten Zeichen werden vom Rechner wie die Buchstaben „a" bis „u" im Grafik-Modus (G) behandelt. Die UDG-System-Variable (Adresse 32600) zeigt auf das erste Byte des Speicherbereichs, doch brauchen Sie zur Ermittlung der Startadresse einer Zeichendefinition keine komplizierten Berechnungen auszuführen. Der Befehl LET address=USR"A" ermittelt die Adresse des ersten Byte für die Definition des A.

Das Programm wurde direkt von der Commodore-Version (Seite 1533) übertragen. Mit den Cursortasten des Spectrum (SHIFT 5, SHIFT 6 usw.) wird der Editier-Cursor gesteuert. Die ungeshifteten Tasten 6, 7 und 8 sind Befehlstasten zum Umschalten einer Zelle, Ändern des zu editierenden Zeichens und Übertragen eines Zeichens in das Textfenster. Das Ausrufungszeichen ist der Ende-Befehl.

Wenn Sie Ihre neuen Zeichen definiert haben, kann der UDG-Bereich mit folgendem Befehl gespeichert werden:

```
SAVE "udgfile" CODE (USR "A"),168
```

Ein erneutes Laden ist möglich mit:

```
LOAD "udgfile" CODE
```

Die Möglichkeiten zur Handhabung anwenderdefinierter Zeichen beim Acorn B sind denen des Spectrum ähnlich. Zwischen den Adressen &0C00 und &0CFF sind 256 Byte für die Definition der ASCII-Code-Zeichen 224 bis 255 reserviert. Ist der Zeichensatz eingeschränkt, sind diese Definitionen auch den ASCII-Codes 128 bis 159, 160 bis 191 und 192 bis 223 zugeordnet. Im erweiterten Modus kann der gesamte druckbare Zeichensatz (von CHR$(32) bis CHR$(255)) umdefiniert werden, jedoch auf Kosten von Speicherplatz. Für die meisten Zwecke sollten 32 Zeichen ausreichen.

Die Funktionen entsprechen denen der C-64- und Spectrum-Version. Die Pfeil-Tasten kontrollieren den Cursor, und die Funktionstasten f1 bis f3 geben Zugriff auf die Befehle Umschalten, Umdefinieren und Übertragen. Mit dem ! wird das Programm beendet.

Beachten Sie die OS-Aufrufe in den Zeilen 70 und 180. Sie blockieren und aktivieren die COPY-Funktionen, so daß die Pfeiltasten im Programm verwendet werden können. Wenn Sie das Programm nicht mit dem „!"-Befehl beenden, müssen Sie *FX4,0 am Anfang einer neuen Zeile eingeben, um die Editier-Tasten wieder zu aktivieren.

Beim Acorn B können Sie zur Zeichendefinition den Befehl VDU23 einsetzen. In diesem Fall ist es jedoch einfacher, die einzelnen Adressen zu errechnen und individuell zu PEEKen und POKEn.

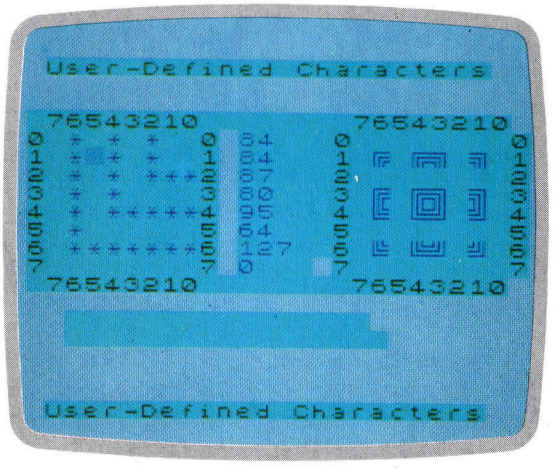
Zum Speichern Ihrer speziellen Zeichen geben Sie folgende Befehle ein:

```
*SAVE "Dateiname" 0C00 0CFF
```

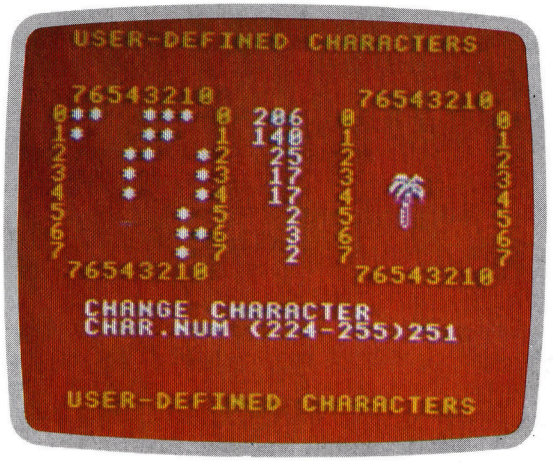
Zum erneuten Laden geben Sie ein:

```
*LOAD ""
```

Spectrum



Acorn B



Spectrum

```
  19 REM **********************
  20 REM *       spectrum      *
  21 REM *usr defined char-gen*
  22 REM **********************
 100 GO SUB 1000: REM initialise
 110 FOR j=0 TO 1 STEP 0
 120 GO SUB 2500: REM input
 130 GO SUB 3000: REM validate
 140 GO SUB pointer: REM obey
 150 NEXT j
 200 STOP
 999 REM **********************
1000 REM *   initialise       *
1001 REM **********************
1020 DIM b(8,8):   DIM c$(2,2):
  DIM o(2,7): DIM r$(8): DIM d$(
1): DIM t(8,8)
1100 LET nullkey=2000: LET movec
rsr=3500: LET command=3000: LET
update=6500
1200  REM ****init screen****
1220 LET r0=4: LET c0=3: LET r1=
8: LET c1=8: LET of=16
1230 LET black=0: LET blue=1: LE
T cyan=5: LET white=7
1240 PAPER cyan: INK black
1250 LET z$="User-Defined Charac
ters": PRINT AT 1,4;z$: PRINT AT
 20,4;z$
1260 LET z$=" 76543210": PRINT A
T r0,c0;z$;: PRINT TAB c0+of;z$
1270 PRINT AT r0+c1+1,c0;z$;: PR
INT TAB c0+of;z$
1280 FOR r=r0+1 TO r0+c1
1290 LET z$=STR$ (r-r0-1): LET z
$=z$+"          "+z$
1300 PRINT AT r,c0;z$;AT r,c0+of
;z$
1310 NEXT r
1420 REM ****cursor offset*****
1440 DATA -1,+1,0,0
1445 DATA 0,0,+1,-1
1450 DATA 0,0,+1,-1
1460 FOR k=1 TO 2: FOR l=1 TO 4
1470 READ o(k,l): NEXT l: NEXT k
1600 REM ***init text window***
1620 FOR r=1 TO r1
1640 FOR c=1 TO c1
1660 LET t(r,c)=32
1680 NEXT c: NEXT r
1800 LET rpos=1: LET cpos=1: LET
 cn=144: LET e$="A"
1820 GO SUB 6000
1990 RETURN
1999 REM **********************
2000 REM * invalid keypress   *
2001 REM **********************
2020 BEEP .2,-3: RETURN
2120 RETURN
2499 REM **********************
2500 REM * crsr @ rpos,cpos   *
2501 REM **********************
2520 LET rp=r0+rpos: LET  cp=c0+
cpos: LET cf=1+b(rpos,cpos): LET
 d$=" *"(cf)
2540 PRINT AT rp,cp; FLASH 1;d$
2600 REM  LET t$=INKEY$: IF t$<>
"" THEN  GO TO 2600
2620 LET t$=INKEY$: IF t$="" THE
N  GO TO 2620
2640 PRINT AT rp,cp; FLASH 0;d$
2700 RETURN
2999 REM **********************
3000 REM * validate input     *
3001 REM **********************
3020 IF t$="!" THEN  LET pointer
=nullkey: LET j=2: RETURN
3040 LET key=CODE (t$)-7: LET k=
key-45: LET pointer=nullkey
3060 IF (key)=1 AND key(=4) THEN
  LET pointer=movecrsr: RETURN
3080 IF (t$)="6" AND t$(="8") TH
EN  LET key=VAL (t$)-4 :    LET
  pointer=command+key*500: RETUR
N
3100 RETURN
3499 REM **********************
3500 REM * move the cursor    *
3501 REM **********************
3520 LET ny=rpos+o(2,key):  LET
nx=cpos+o(1,key)
3540 IF  (ny(1 OR ny)r1)  THEN
  RETURN
3560 IF  (nx(1 OR nx)r1)  THEN
  RETURN
3580 LET rpos=ny: LET cpos=nx
3600 RETURN
3999 REM **********************
4000 REM * toggle a cell      *
4001 REM **********************
4020 LET tg=1-b(rpos,cpos): LET
d$=" *"(1+tg)
4040 PRINT AT rpos+r0,cpos+c0;d$
4060 LET b(rpos,cpos)=tg
4120 LET pe=PEEK (mpos+rpos)
4140 LET pe=pe+(tg*2-1)*(2^(8-cp
os))
4160 POKE (mpos+rpos),pe
4200 PRINT AT r0+rpos,c0+c1+3;PE
EK (mpos+rpos);"  "
4220  NEXT r
4300 RETURN
4499 REM **********************
4500 REM * define new char    *
4501 REM **********************
4520 PRINT  AT  r0+r1+3,c0+2;
 FLASH 1;"Change Character"
4530 GO SUB update
4540 PRINT  AT  r0+r1+4,c0+2;
;"Hit A Key (a - u)"
4550 FOR z=1 TO 1
4560 LET e$=INKEY$: IF e$<>""
 THEN  GO TO 4560
4570 LET e$=INKEY$: IF e$=""
THEN  GO TO 4570
4580 IF (e$("a" OR e$)"u") THEN
 GO SUB nullkey: LET z=0
4590 NEXT z
4600 LET cn=CODE (e$)+79
4610 IF cn)164 THEN  LET cn=cn-3
2
4620 PRINT  AT  r0+r1+3,c0+2;
 FLASH 0;"              "
4630 PRINT AT  r0+r1+4,c0+2;
"              "
4640 GO SUB 6000
4990 RETURN
4999 REM **********************
5000 REM * place a char       *
5001 REM **********************
5020 PRINT AT rpos+r0,cpos+c0+
of;CHR$ (cn)
5040 LET t(rpos,cpos)=cn
5490 RETURN
5999 REM **********************
6000 REM *  display a char     *
6001 REM **********************
6020 LET mpos=USR (e$)-1:
 INK blue
6040 FOR r=1 TO c1: LET pe=PEEK
(mpos+r): LET p=pe: LET z$=""
6060 FOR c=c1 TO 1 STEP -1: LET
n=INT (pe/2): LET q=pe-2*n
6080 LET b(r,c)=q: LET pe=n
6100 LET r$(c)=" *"(q+1): NEXT c
6120 PRINT AT r0+r,c0+1;r$;AT r0
+r,c0+c1+3;p;"  "
6130 NEXT r
6140 RETURN
6499 REM **********************
6500 REM * update text window *
6501 REM **********************
6520 FOR r=  TO 8: LET y=r0+r
6540 FOR c=1 TO 8: LET x=c0+c+of
6560 PRINT AT y,x;CHR$ (t(r,c))
6580 NEXT c: NEXT r
6600 RETURN
```

Acorn B

```
  19 REM**********************
  20 REM*    BBC Micro       *
  21 REM**********************
  22 REM* USER-DEF CHGEN     *
  50 MODE 1
  60 @%=3
  70 *FX4,1
 100 PROCinitialise
 110 REPEAT
 120 PROCget_command
 130 PROCvalidate_command
 140 PROCobey(COMMAND)
 160 UNTIL COMMAND=QUIT
 180 *FX4,0
 200 END
 999 REM**********************
1000 DEFPROCinitialise
1001 REM**********************
1040 DIM BD(8,8),C$(2,2)
1045 DIMOFST(2,7),D$(1),TX(8,8)
1060 D$(0)=" ":D$(1)="*"
1080 CGEN=&0C00-1
1100 QUIT=-1:NULLKEY=0:MVCRSR=1:
1105 TGGLE=2:DFINE=3:PLACE=4
1200 REM*****INIT SCREEN*****
1220 R0=4:C0=3:RL=8:CL=8
1225 C9=CL+3:OF=16
1230 BLACK=0:RED=1:YELLOW=2
1235 WHITE=3:BACKGR=128
1240 COLOUR BACKGR+RED
1245 COLOUR YELLOW:CLS
1250 Z$="USER-DEFINED CHARACTERS"
1255 PRINTTAB(4,1)Z$;TAB(4,20)Z$
1260 Z$=" 76543210"
1264 PRINTTAB(C0,R0)Z$
1268 PRINTTAB(C0+OF,R0)Z$
1270 Y=R0+RL+1:X=C0
1274 PRINTTAB(X,Y)Z$
1278 PRINTTAB(X+OF,Y)Z$
1280 FOR R=R0+1 TO R0+RL
1290 Z$=STR$(R-R0-1)
1295 Z$=Z$+"          "+Z$
1300 PRINTTAB(C0,R)Z$
1304 PRINTTAB(C0+OF,R)Z$
1310 NEXT R
1320 COLOUR WHITE
1420 REM****CURSOR OFFSET****
1440 DATA -1,+1,0,0
1450 DATA 0,0,+1,-1
1460 FOR K=1 TO 2:FOR L=1 TO 4
1470 READ OFST(K,L):NEXT L,K
1500 REM****KEY SETTINGS*****
1520 *KEY1"T"
1540 *KEY2"D"
1560 *KEY3"P"
1600 REM***INIT TEXT WINDOW**
1620 FOR R=1 TO RL
1640 FOR C=1 TO PL
1660 TX(R,C)=32
1680 NEXT C,R
1800 RPOS=1:CPOS=1:CN=224
1805 PROCdisplay_character
1990 ENDPROC
1999 REM**********************
2000 DEFPROCnullkey
2001 REM**********************
2020 SOUND 1,-15,48,10
2120 ENDPROC
2499 REM**********************
2500 DEFPROCget_command
2501 REM**********************
2520 RP=R0+RPOS:CP=C0+CPOS
2540 PRINTTAB(CP,RP);
2620 T$=GET$
2700 ENDPROC
2999 REM**********************
3000 DEFPROCvalidate_command
3001 REM**********************
3020 COMMAND=MVCRSR
3040 KEY=ASC(T$)-135
3060 IF KEY(1 THEN COMMAND=NULLKEY
3065 IF KEY)4 THEN COMMAND=NULLKEY
3080 IF T$="T"THEN COMMAND=TGGLE
3100 IF T$="D"THEN COMMAND=DFINE
3120 IF T$="P" THEN COMMAND=PLACE
3150 IF T$="!" THEN COMMAND=QUIT
3160 ENDPROC
3299 REM**********************
3300 DEFPROCobey(command)
3301 REM**********************
3320 IF command=QUIT THEN
PROCnullkey
3340 IF command=MVCRSR THEN
PROCmovecursor(KEY)
3360 IF command=TGGLE THEN
PROCtoggle_a_cell
3380 IF command=DFINE THEN
PROCdefine_character
3400 IF command=PLACE   THEN
PROCplace_character
3420 IF command=NULLKEY THEN
PROCnullkey
3450 ENDPROC
3499 REM**********************
3500 DEFPROCmovecursor(key)
3501 REM**********************
3520 NY=RPOS+OFST(2,key)
3525 NX=CPOS+OFST(1,key)
3540 IF NY)=1 AND NY(=RL THEN
RPOS=NY ELSE PROCnullkey
3560 IF NX)=1 AND NX(=CL THEN
CPOS=NX ELSE PROCnullkey
3600 ENDPROC
3999 REM**********************
4000 DEFPROCtoggle_a_cell
4001 REM**********************
4020 TG=1-BD(RPOS,CPOS)
4025 Y=RPOS+R0:X=CPOS+C0
4040 PRINTTAB(X,Y)D$(TG)
4060 BD(RPOS,CPOS)=TG
4120 PE=?(MPOS+RPOS)
4140 PE=PE+(TG*2-1)*(2^(8-CPOS))
4160 ?(MPOS+RPOS)=PE
4200 PRINTTAB(C0+C9,R0+RPOS)PE
4220 PROCupdate_text
4300 ENDPROC
4499 REM**********************
4500 DEFPROCdefine_character
4501 REM**********************
4510 REPEAT
4520 PRINTTAB(C0+2,R0+RL+3);
4525 PRINT"                  "
4540 PRINTTAB(C0+2,R0+RL+4);
4545 PRINT"                  "
4550 PRINTTAB(C0+2,R0+RL+4);
4560 INPUT"NUMBER (224-255)"CHNUM
4580 UNTIL CHNUM)223 AND CHNUM(256
4600 CN=CHNUM
4620 PROCdisplay_character
4900 ENDPROC
4999 REM**********************
5000 DEFPROCplace_character
5001 REM**********************
5015 X=CPOS+C0+OF:Y=RPOS+R0
5020 PRINTTAB(X,Y);CHR$(CN)
5040 TX(CPOS,RPOS)=CN
5400 ENDPROC
5999 REM**********************
6000 DEFPROCdisplay_character
6001 REM**********************
6020 MPOS=(CN-224)*8+CGEN
6040 FOR R=1 TO CL
6045 PE=?(MPOS+R):P=PE:Z$=""
6060 FOR C=CL TO 1 STEP -1
6065 N=INT(PE/2):Q=PE-2*N
6080 BD(R,C)=Q:PE=N
6100 Z$=D$(Q)+Z$:NEXT C
6120 PRINTTAB(C0+1,R0+R)Z$
6125 PRINTTAB(C0+C9,R0+R)P
6130 NEXT R
6140 ENDPROC
6499 REM**********************
6500 DEFPROCupdate_text
6501 REM**********************
6520 FOR R=1 TO RL:Y=R0+R
6540 FOR C=1 TO RL:X=C0+C+OF
6560 PRINTTAB(X,Y)CHR$(TX(C,R))
6580 NEXT C,R
6900 ENDPROC
```

Strukturvergleiche

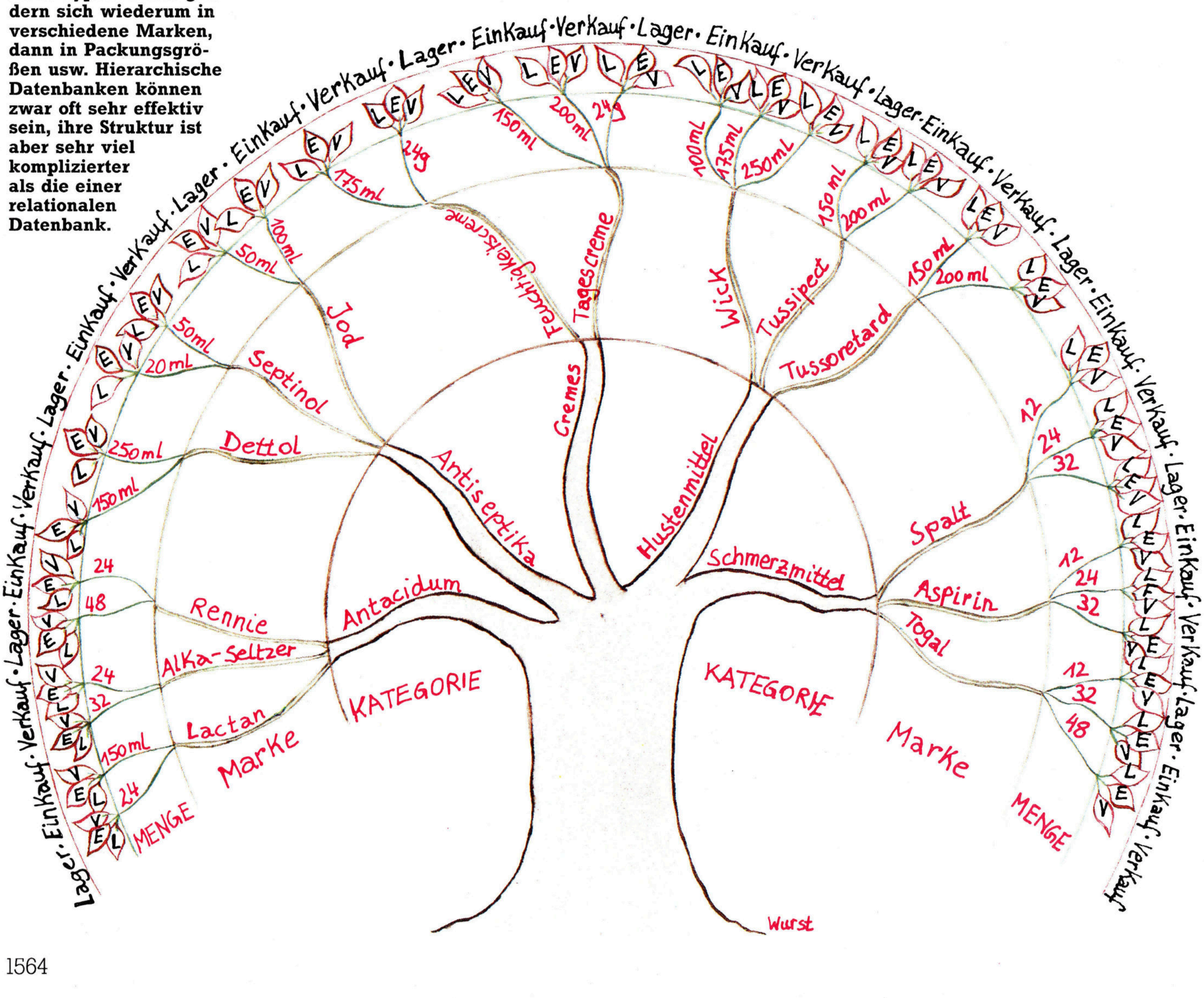
Informationen werden in Heimcomputern meist als sogenannte „relationale“ Datenbank organisiert. Aber es gibt auch Alternativen zu dieser Struktur. Hierarchische und Netzwerk-Datenbanken können oft effektiver sein als ein relationaler Aufbau.

Fast alle Datenbanksysteme für Heimcomputer sind relational aufgebaut. Das bedeutet, daß die Datensätze in der Datei in Form einer Tabelle aus Zeilen und Spalten aufgebaut sind. Die vielfältigen Möglichkeiten der Darstellung von Informationen lassen zwar eine komplexere Struktur vermuten, im Grunde steckt aber kaum mehr dahinter als ein einfaches Raster. Der Aufbau aller Datensätze ist gleich, und auch die Felder müssen eine feste Länge haben.

Der Ausdruck „relational“ bezieht sich auf die Tatsache, daß jede Zeile der Datenbank in einer eindeutigen Beziehung zu jeder Spalte steht. Besonders flexibel ist diese Struktur nicht, sie reicht jedoch für einfachere Datenbanksysteme aus.

Ein von der tabellarischen Organisation völlig verschiedener Ansatz ist die Anordnung der Daten in hierarchischer Form. Hierbei sind die Informationen in Baumform miteinander verknüpft – sie verzweigen sich immer weiter, bis sie zum Schluß in einzelne „Blätter“ übergehen. Als anschauliches Beispiel kann uns hier die Datenbank für einen Zulieferbetrieb dienen, die wir zuerst in relationaler, dann in

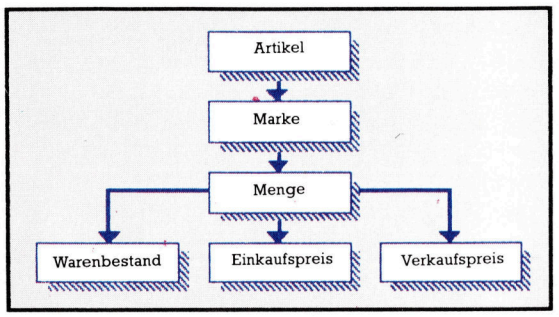
Im Gegensatz zu einer „relationalen“ Datenbank, die Informationen in tabellarischer Form speichert, werden die Daten in einer hierarchischen Datenbank in „Baumform“ organisiert. In unserem Beispiel führen „Zeiger“ von der Hauptkategorie „Medikamente“ zu den unterschiedlichen Medikament-Typen. Diese gliedern sich wiederum in verschiedene Marken, dann in Packungsgrößen usw. Hierarchische Datenbanken können zwar oft sehr effektiv sein, ihre Struktur ist aber sehr viel komplizierter als die einer relationalen Datenbank.



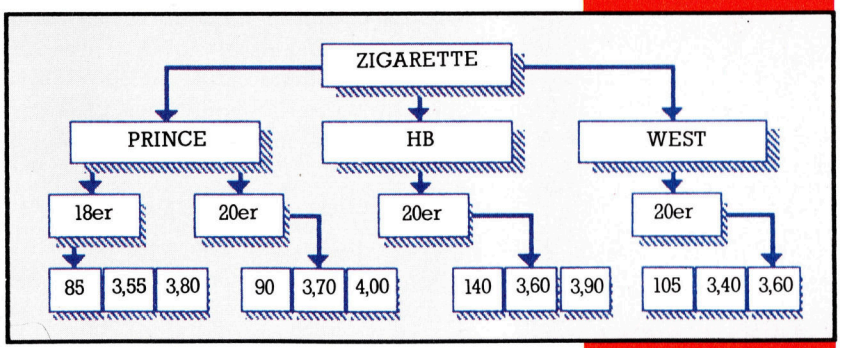
hierarchischer Form darstellen:

ARTIKEL	MARKE	MENGE	BESTAND	EINKAUF	VERKAUF
ZIGARETTE	PEER 100	20	350	3,60	3,90
ZIGARETTE	HB	20	140	3,65	3,90
ZIGARETTE	PRINCE	19	85	3,55	3,80
ZIGARETTE	WEST	18	105	3,40	3,60
SÜSSWAREN	MARS	1	106	0,50	0,70
SÜSSWAREN	BOUNTY	1	95	0,40	0,55
SÜSSWAREN	TREETS	1	25	0,70	0,95
ZEITUNG	BILD	1	35	0,33	0,50
ZEITUNG	WELT	1	12	0,62	1,00
ZEITUNG	STERN	1	62	2,80	3,50
GETRÄNK	COLA	330	35	0,44	0,70
GETRÄNK	FANTA	330	25	0,47	0,70
GETRÄNK	SPRITE	330	20	0,43	0,65

Hierarchisch aufgebaut sähe das so aus:



Der Datensatz ist hier in Datensegmente aufgeteilt, wobei jedes Segment mehr als nur ein Feld umfassen kann. Die hierarchisch geordneten Einträge zum Begriff ZIGARETTE haben etwa diese Form:

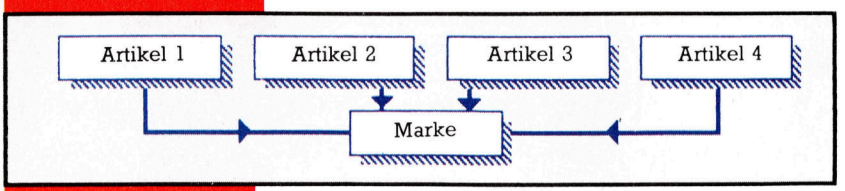


Wie in der Darstellung ist jede Teilinformation den anderen Daten entweder über- oder untergeordnet. Zwischen den Datensegmenten werden nur „Zeiger" benötigt. In diesem Beispiel benötigen wir nur ein Segment ZIGARETTE, daß Zeiger zu den unterschiedlichen Marken und weitere Zeiger zur Packungsgröße, den Kosten, Preisen usw. enthält.

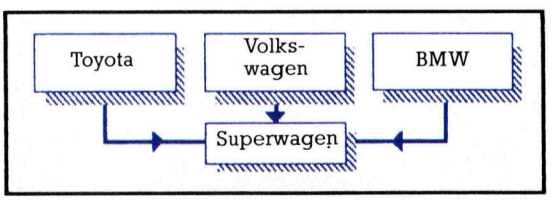
Angenommen, unser Betrieb hätte nur hundert verschiedene Artikel, aber tausend Informationen über deren Ein- und Verkauf. In einer relationalen Datenbank bräuchte man dafür tausend Datensätze – für jede Information einen Eintrag. In der hierarchischen Datenbank reichen hundert Segmente zur Verwaltung aller Artikel. Die Struktur der Datenbank wäre zwar komplizierter, dafür ließe sich aber das mehrfache Aufführen der gleichen Information vermeiden.

Eine weitere Organisationsform für Datenbanken ist das CODASYL- oder Netzwerksystem. CODASYL wurde von der Projektgruppe Datenbank der „COnference in DAta SYstem Languages" entwickelt. Die Schwierigkeit hierarchisch aufgebauter Datenbanken ist, daß die Verknüpfung der Informationen nur eine Flußrichtung erlaubt. In unserem Beispiel könnten aber etwa „EMBASSY-FEHLFARBEN" sowohl ein Zweig von „ZIGARREN" als auch vom Markenzeichen „EMBASSY" sein. Dieses Problem taucht auch dann auf, wenn Gegenstände mehr als einen Hersteller haben.

CODASYL baut ein Netzwerk aus „Gruppen"-Daten auf, die jeweils aus mehreren Datensätzen bestehen. Im Unterschied zu den Datensätzen einer relationalen Datenbank ist der Umfang der Einträge unbeschränkt. Jeder Datensatz kann außerdem mehreren Datensätzen zugeordnet werden. Bei CODASYL kann eine Gruppe auch aus nur einem Datensatz bestehen, einzelne Datensätze können allerdings innerhalb einer Gruppe nicht mehrfach auftauchen. Eine Struktur wie diese wäre also in diesem Fall nicht erlaubt:



Das ist natürlich eine Einschränkung. Auch wenn es kaum ZIGARETTEN, ZEITUNGEN und GETRÄNKE gibt, die alle PHANTASIA heißen – eine Firmenkonstellation wie in diesem Beispiel ist nicht gerade ungewöhnlich:



Hierarchische und Netzwerk-Datenbanken sind zwar flexibler als relationale Versionen, ihre Komplexität erfordert aber eine Rechnerleistung, die Heimcomputer überfordert. Es bleibt also in diesem Bereich bei der relationalen Verwaltung. Damit wird ein Warenverzeichnis immer dann zum Problem, wenn es für einen bestimmten Artikel mehr als lediglich einen Lieferanten gibt. Man muß den Datensatz des jeweiligen Artikels durch einen weiteren Datensatz ergänzen, in dem die dazugehörigen Lieferanten verzeichnet sind. Hier helfen Datenbanksysteme, bei denen mehrere Dateien gleichzeitig offen sein können.

Unterbrechungen

Bei der Entwicklung unseres Debuggers vervollständigen wir das Modul zur Bearbeitung der Unterbrechungspunkte.

In unserem Unterbrechungspunktmodul fehlen noch die Subroutinen für das Löschen von Unterbrechungspunkten und das Wiedereinsetzen von Op-Codes, die zeitweilig durch SWI-Op-Codes ersetzt wurden. Als erstes entwickeln wir die Routine „Unterbrechungspunktlöschen" (aus der bereits bekannten Unterbrechungstabelle).

Die Unterbrechungstabelle (BPTAB) kann bis zu 16 Unterbrechungspunkte enthalten. Um einen bestimmten Punkt löschen zu können, müssen wir daher eine Zahl (zwischen 0 und 15) haben, die den Offset in der Tabelle angibt. Der Löschvorgang setzt alle auf diesen Punkt folgenden Einträge um eine Position (zwei Bytes) zurück und dekrementiert die Zahl-der-Unterbrechungspunkte.

Daten:

Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert.

Zahl-Unterbrechungspunkt – ein Acht-Bit-Zähler

Unterbrechungstabelle – ein Array von 16-Bit-Adressen

Eintrag-zum-Löschen – ein Acht-Bit-Offset (mit einem Wert zwischen 1 und 16)

Ablauf:

```
Zahl-der-Unterbrechungspunkte dekrementieren
IF Eintrag-zum-Löschen (= Zahl-der-Unterbrechungspunkte(vorletzter) THEN
   FOR Zahl-Unterbrechungspunkt = Eintrag-zum-Löschen TO Zahl-der-Unterbrechungspunkte(vorletzter)
      Unterbrechungstabelle(Zahl-Unterbrechungspunkt + 1) auf Unterbrechungstabelle(Zahl-Unterbrechungspunkt) verschieben
      Entfernte-Werte(Zahl-Unterbrechungspunkt + 1) auf Entfernte-Werte (Zahl-Unterbrechungspunkt + 1) verschieben
   ENDFOR
ENDIF
```

Ende des Ablaufs

Der Parameter Eintrag-zum-Löschen kann an B übergeben werden. Dadurch erhält der Zähler, Zahl-Unterbrechungspunkt, der ebenfalls dort gespeichert wird, automatisch den richtigen Anfangswert. Er wird nach dem Vergleich mit Zahl-der-Unterbrechungspunkte automatisch dekrementiert, um den Offset in der Acht-Bit-Tabelle Entfernte-Werte bilden zu können. Danach wird er nach links verschoben (mit Zwei multipliziert), um den Offset in der 16-Bit-Unterbrechungstabelle anzugeben. Den Acht-Bit-Offset können wir in B speichern und den 16-Bit-Offset in A. Wenn die Adressen der Tabelleneinträge in X und Y gelegt werden, lassen sich die beiden Tabellen mit dem Auto-Inkrementbefehl durcharbeiten. Dabei können zwar die 16-Bit-Einträge über U versetzt werden, doch müssen wir für die Acht-Bit-Werte wieder A verwenden.

Die letzte Prozedur dieses Moduls nimmt den Unterbrechungspunkt aus dem Programm heraus, indem sie den SWI-Op-Code gegen den ursprünglichen, in der Tabelle Entfernte-Werte gespeicherten Op-Code austauscht.

Daten:

Zahl-Unterbrechungspunkt – ein Acht-Bit-Offset in die Unterbrechungstabelle

Ablauf:

```
Den Inhalt von Entfernte-Werte(Zahl-Unterbrechungspunkt) holen
In der Adresse der Unterbrechungstabelle (Zahl-Unterbrechungspunkt) speichern
```

Wir setzen voraus, daß der Parameter Zahl-Unterbrechungspunkt als Zahl zwischen 1 und 16 an B übergeben wurde und zum Tabellenoffset umgewandelt werden kann.

Wir beginnen nun mit dem Aufbau des Moduls, das die acht Steuerbefehle des Systems (je ein Zeichen) ausführt. Einige Befehle könnten die bereits codierten Routinen zwar direkt einsetzen, aus Gründen der Einheitlichkeit rufen wir sie im Steuermodul jedoch mit dem Befehl CALL auf.

Der Befehl B – das Einsetzen eines Unterbrechungspunktes – wird vollständig von der Routine Unterbrechungspunkt-Einsetzen (BP01) erledigt. Das Modul muß daher nur den folgenden Aufruf

```
CMDB BRA BP01
```

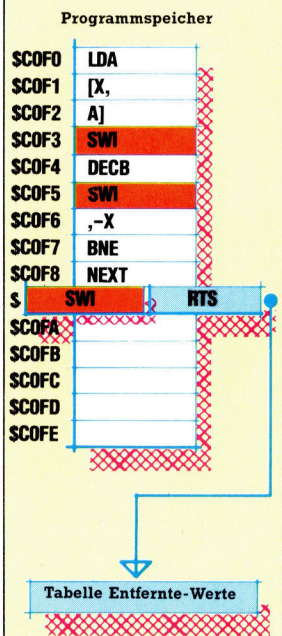
erhalten. Auch der Befehl U wird fast vollständig von der soeben entwickelten Routine (BP04) erledigt. Wir müssen dafür jedoch zuerst die Adresse des zu löschenden Unterbrechungspunktes erhalten und dann die Unterbrechungstabelle nach dieser Adresse durchsuchen. Ist die Adresse dort nicht vorhanden, wird der Befehl ignoriert; andernfalls übergeben wir den Offset ganz einfach an die Subroutine bei BP02.

Daten:

Prompt – wird dargestellt

Adresse-des-Unterbrechungspunktes – die Eingabe

Code(unter)brecher



Der Debugger setzt Unterbrechungspunkte in den analysierten Objectcode ein. Er speichert dafür zuerst den Code des Unterbrechungspunktes in der Tabelle Entfernte-Werte und inkrementiert den Zähler Zahl-der-Unterbrechungspunkte. Danach überschreibt er den Inhalt des Unterbrechungspunktes mit dem SWI-Op-Code. Beim Löschen wird der entfernte und in der Tabelle gespeicherte Wert wieder in den Programmcode eingesetzt und das überflüssige Tabellenbyte gelöscht, indem alle darauffolgenden Einträge um eine Position rückwärts verschoben werden.

Unterbrechungstabelle
Zahl-Unterbrechungspunkt
Ablauf:
Prompt darstellen
Adresse-des-Unterbrechungspunktes holen
Zahl-Unterbrechungspunkt auf 16 setzen
WHILE Unterbrechungstabelle(Zahl-Unterbrechungspunkt) <> Adresse-des-Unterbrechungspunktes AND Zahl-Unterbrechungspunkt > 0
Zahl-Unterbrechungspunkt dekrementieren
IF gefunden THEN
Unterbrechungspunkt herausnehmen

Die Adresse-des-Unterbrechungspunktes bleibt in Y, damit X als Zeiger auf die Tabelle eingesetzt werden kann. Zahl-Unterbrechungspunkt bleibt in B.

Der Befehl D zeigt die Unterbrechungspunkte. Er wird von der Routine DISPBP erledigt und durch den Aufruf einer Subroutine angesprochen:

CMDD BRA DISPBP

Der Befehl S löst den Start des Programms aus und aktiviert die Unterbrechungspunkte. Dabei wird der SWI-Op-Code in die Adressen der Unterbrechungstabelle eingesetzt, der ursprüngliche Op-Code dieser Position in Entfernte-Werte gespeichert und die Steuerung an die Anfangsadresse des Programms übergeben. Beachten Sie, daß der nächste Unterbrechungspunkt die Zahl 1 ist. Hier der Ablauf für den Programmstart:

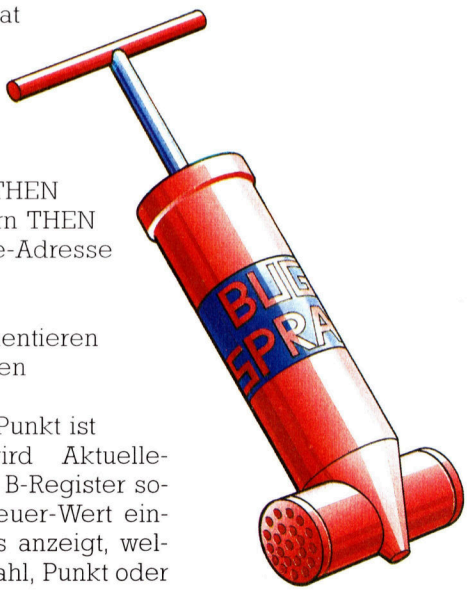
Daten:
Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert
Unterbrechungstabelle
Entfernte-Werte
Zahl-Unterbrechungspunkt – ein Acht-Bit-Zähler
Nächster-Unterbrechungspunkt – ein Acht-Bit-Wert
SWI-Op-Code – ein Acht-Bit-Wert
Anfangsadresse – die 16-Bit-Adresse des analysierten Programms
Ablauf:
Zahl-Unterbrechungspunkt auf Zahl-der-Unterbrechungspunkte setzen
WHILE Zahl-Unterbrechungspunkt > 0
Unterbrechungspunkt-Einrichten(Zahl-Unterbrechungspunkt)
ENDWHILE
Nächster-Unterbrechungspunkt auf 1 setzen
Auf Anfangsadresse springen

Für diese Aufgabe können wir die bereits codierte Routine Unterbrechungspunkt-einsetzen verwenden, die den Wert Zahl-Unterbrechungspunkt (minus Eins, damit er als Tabellenoffset dienen kann) in A erwartet. Wir werden daher A dekrementieren, bevor wir Unterbrechungspunkt-einsetzen aufrufen. Auf der nächsten Seite ist der Code dieser Routine abgedruckt.

Das Ende der Routine ist erklärungsbedürftig. Da beim Ablauf des getesteten Programms keine zusätzlichen Werte auf dem Stack liegen dürfen, müssen wir sicherstellen, daß der Stack bei Übergabe der Steuerung an das Programm leer ist. Nun kann das Hauptmodul zwar alle überflüssigen Werte leicht vom Stack herunterziehen, doch läßt sich nicht verhindern, daß bei einem BSR-Aufruf (den wir aus Gründen der Einheitlichkeit einsetzen) die Rücksprungadresse auf den Stack geschoben wird. Wenn wir diesen Wert dort belassen und das Programm oft neu starten, wächst der Stack ständig. Zur Lösung dieses Problems löschen wir diese Adresse in dem Moment, in dem wir auch die Steuerung an das Programm zurückgeben. Wir ersetzen dabei einfach die Rücksprungadresse durch die Anfangsadresse des Programms. RTS zieht dann als Rücksprungadresse die Startadresse vom Stack: Die Steuerung ist übergeben, während gleichzeitig der Stack bereinigt wird.

Mit dem Befehl M lassen sich Speicherstellen untersuchen und verändern. Dafür muß eine Eingabeadresse geholt und der Inhalt dieser Adresse auf dem Bildschirm angezeigt werden. Der Anwender kann entweder eine neue zweistellige Hexzahl auf dieser Adresse speichern oder ein Return oder einen Punkt eingeben. In den beiden ersten Fällen müssen wir die darauffolgende Speicherstelle ansprechen, während ein Punkt den Ablauf abbricht.

Daten:
Aktuelle-Adresse – die 16-Bit-Adresse der untersuchten Speicherstelle
Aktueller-Wert – der Inhalt von Aktuelle-Adresse (ein Acht-Bit-Wert)
Neuer-Wert – ersetzt Aktueller-Wert. Er hat ebenfalls ein Acht-Bit-Format
Ablauf:
Aktuelle-Adresse holen
REPEAT
Aktueller-Wert anzeigen
Neuer-Wert holen
IF Neuer-Wert kein Punkt THEN
IF Neuer-Wert kein Return THEN
Neuer-Wert in Aktuelle-Adresse speichern
ENDIF
Aktuelle-Adresse inkrementieren
Aktuelle-Adresse anzeigen
ENDIF
UNTIL Aktuelle-Adresse ein Punkt ist



Für diese Befehlsroutine wird Aktuelle-Adresse in X abgelegt und das B-Register sowohl für Aktueller-Wert und Neuer-Wert eingesetzt. A enthält das Flag, das anzeigt, welche der drei Alternativen (Hexzahl, Punkt oder Return) eingegeben wurde.

In der nächsten Folge codieren wir die letzten drei Befehle G, R und Q und untersuchen den dabei eingesetzten Interrupt-Mechanismus. Außerdem entwickeln wir das Hauptmodul unseres Debuggers.

Die Routine „Unterbrechungspunkt-löschen“

```
BP04    PSHS    A,B,X,Y,U       Eingesetzte Register sichern
        DEC     NUMBP,PCR       Zahl der-Unterbrechungspunkte
                                dekrementieren

IF02    CMPB    NUMBP,PCR       IF Eintrag-zum-Löschen <
        BGT     ENDF02            Zahl-der-Unterbrechungspunkte
        DECB                    B in Offset umwandeln
        TFR     B,A             B nach A kopieren
        LSLA                    A in Offset umwandeln
        LDX     BPTAB,PCR       Basisadresse der Unterbrechungs-
                                tabelle
        LEAX    A,X             Unterbrechungstabelle(Zahl-
                                Unterbrechungspunkt)
        LDY     REMTAB,PCR      Basisadresse von Entfernte-Werte
        LEAY    B,Y             Entfernte-Werte(Zahl-
                                Unterbrechungspunkt)
FOR00   LDU     2,X             Eintrag der Unterbrechungs-
                                tabelle holen, der verschoben wird
        STU     ,X++            Um eine Position zurücksetzen
        LDA     1,Y             Den Eintrag von Entfernte-Werte
                                holen, der verschoben werden soll
        STA     ,Y+             Um eine Position zurücksetzen
        INCB
        CMPB    NUMBP,PCR       War das der letzte?
        BLT     FOR00           Nächster
ENDF02  PULS    A,B,X,Y,U,PC    Register wiederherstellen und
                                Rücksprung
```

Die Routine „Unterbrechungspunkt-herausnehmen“

```
BP05    PSHS    A,B,X
        DECB                    In den Offset für Entfernte-Werte
                                umwandeln
        LDX     REMTAB,PCR      Basisadresse von Entfernte-
                                Werte
        LDA     B,X             Wert holen, der verschoben wird
        LSLB                    In den Offset für die Unterbre-
                                chungstabelle umwandeln
        LDX     BPTAB,PCR       Basisadresse der Unterbre-
                                chungstabelle
        STA     [B,X]           Auf Tabellenadresse speichern
        PULS    A,B,X,PC        Register wiederherstellen und
                                Rücksprung
```

Der Befehl U

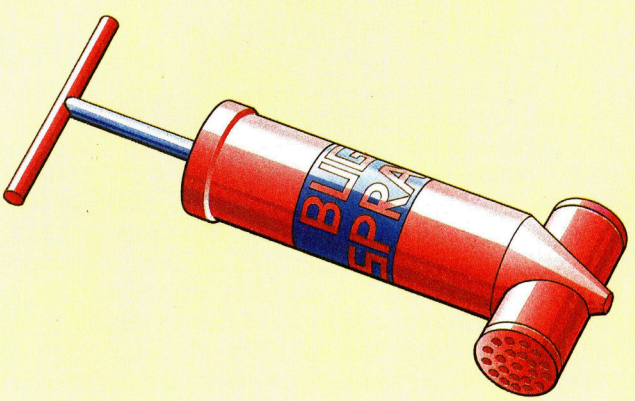
```
PROMPT  FCB     '>
CMDU    PSHS    A,B,X,Y         Eingesetzte Register sichern
        LDA     PROMPT,PCR
        BSR     OUTCH           Prompt anzeigen
        BSR     GETADD          Adresse holen
        TFR     D,Y             Adresse in Y speichern
        LDB     MAXBP,PCR       Größtmögliche Zahl von Unterbre-
                                chungspunkten (16)
        LDX     BPTAB,PCR       Basisadresse der Unterbrechungs-
                                tabelle
        TFR     B,A
        LSLA                    A ist Offset für das Ende der
                                Unterbrechungstabelle
        LEAX    A,X             X zeigt nun über das Tabellenende
                                hinaus
        TSTB                    Je nach Inhalt von B Flags setzen
WHIL02  BLE     ENDW02          WHILE B > 0
        CMPY    ,--X            (Bedenken Sie, daß X zuerst dekre-
                                mentiert wurde)
        BEQ     ENDW02          AND die Adresse-des-
                                Unterbrechungspunktes nicht
                                vorhanden ist
        DECB                    Zahl-Unterbrechungspunkt dekre-
                                mentieren
        BRA     WHIL02
ENDW02  TSTB                    Gefunden, wenn B > 0
IF03    BLE     ENDF            IF gefunden THEN
        BSR     BP04            Unterbrechungspunkt-heraus-
                                nehmen
ENDF03  PULS    A,B,X,Y
```

Der Befehl S

```
START   RMB     2               Startadresse
CMDS    LDA     NUMBP,PCR       Zahl-Unterbrechungspunkt auf
                                Zahl-der-Unterbrechungspunkte
WHIL03  TSTA                    Den Wert von Zahl-Unterbre-
                                chungspunkt testen
        BLE     ENDW03          WHILE Zahl-Unterbrpkt. > 0
        DECA                    Zahl-Unterbrpkt. dekrementieren
        BSR     BP02            Unterbrechungspunkt-anlegen
        BRA     WHIL03          Nächster Unterbrechungspunkt
ENDW03  LDA     #1              Nächster-Unterbrechungspunkt auf
                                1 setzen
        STA     NEXTBP,PCR
        STD     1,S
        RTS
```

Der Befehl M

```
PROMPT  FCB     '>
SPACE   FCB     32              ASCII-Code für Leerzeichen
CMDM    PSHS    A,B,X           Eingesetzte Register sichern
        LDA     PROMPT,PCR
        BSR     OUTCH           Prompt anzeigen
        BSR     GETADD          Aktuelle-Adresse holen
        TFR     D,X             Auf X verschieben
REPT01  LDB     ,X              Aktueller-Wert holen
        BSR     PUTHEX          Aktueller-Wert anzeigen
        LDA     SPACE,PCR
        BSR     OUTCH           Leerzeichen anzeigen
        BSR     GETVAL          Neuer-Wert holen
IF03    TSTA                    IF Neuer-Wert kein Punkt
        BLT     UNTL01
        BGT     ENDF03          IF Neuer-Wert kein Return
        STB     ,X              Neuer-Wert in Aktuelle-Adresse
                                speichern
ENDF03  LEAX    1,X             Aktuelle-Adresse inkrementieren
        TFR     X,D             Aktuelle-Adresse anzeigen
        BSR     DSPADD
        BRA     REPT
UNTL01  PULS    A,B,X,PC
```



Fachwörter von A bis Z

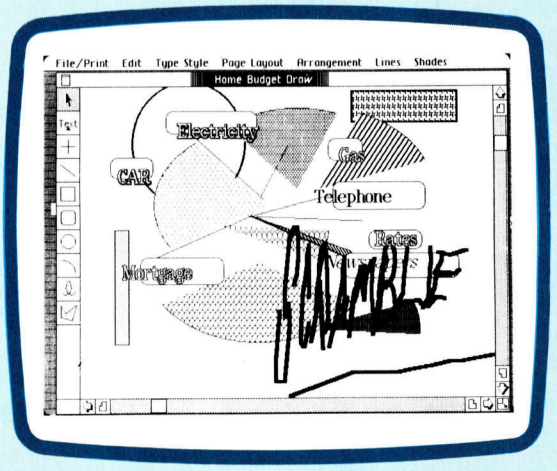
Integrated Circuit = Integrierte Schaltung
Als integrierte Schaltung oder ‚IC' wird ein elektronischer Schaltkreis bezeichnet, der in Fotoätztechnik auf einem Siliziumscheibchen aufgebaut ist. Dabei sind viele tausend Dioden- und Transistorfunktionen auf einer Fläche von wenigen Quadratmillimetern „integriert". ICs haben die gesamte Elektronikindustrie revolutioniert. Es gibt ICs für die verschiedensten Aufgaben; Hauptvorteil der Miniaturisierung gegenüber der Verwendung einzelner Bauelemente sind der geringe Leistungsbedarf und erhöhte Geschwindigkeit.

Die ersten ICs wurden Ende der fünfziger Jahre von Jack Kilby bei Texas Instruments entwickelt. Es folgten vier IC-„Generationen": Small Scale Integration (SSI), Medium Scale Integration (MSI), Large Scale Integration (LSI) und Very Large Scale Integration (VLSI).

Bei den ICs sind im wesentlichen zwei Gruppen zu unterscheiden: In Microcomputern werden bevorzugt MOS-Typen (Metall-Oxid-Semiconductor) verwendet, die sich durch hohe Packungsdichte und geringen Leistungsbedarf auszeichnen und extrem kostengünstig herzustellen sind. Bei größeren Rechnern werden dagegen vorwiegend bipolare ICs eingesetzt, bei denen die Schaltung auf dem Siliziumplättchen mit positiven und negativen Transistorelementen aufgebaut wird – dieses Verfahren ist wegen der unterschiedlichen Halbleiterdotierung aufwendiger als die MOS-Technologie. Die Packungsdichte ist erheblich geringer, der Leistungsbedarf größer; bipolare ICs sind jedoch schneller als MOS-Bausteine.

Intelligent Terminal = Intelligentes Terminal
Mit einem intelligenten Terminal lassen sich in begrenztem Umfang Daten bearbeiten – die Ergebnisse können dann an Großrechner weitergegeben werden. Diese Vorverarbeitung, zum Beispiel das Editieren, wird durch einen eigenen Prozessor möglich, der damit den Hauptrechner für andere Aufgaben freihält.

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.



Lisadraw ist ein interaktives Grafikprogramm-Paket. Jedes Pixel auf dem Bildschirm kann einzeln angesprochen werden, so daß sich eine hohe Grafikauflösung ergibt.

Interactive Graphics = Interaktive Grafik
Von ‚interaktiver' Grafik spricht man, wenn ein System die Erstellung und Modifizierung von Bildern und Grafik in unmittelbarem Wechselspiel zwischen Rechner und Benutzer erlaubt. Diese Technik findet bei Microcomputern zunehmend Anwendung. Bekannt ist das integrierte Software-Paket ‚Lotus 1-2-3', das die sofortige Darstellung von Tabellendaten als Kurven oder Torten- bzw. Balkendiagramm ermöglicht.

Interface = Schnittstelle
Eine Schnittstelle kann sowohl eine Anpassungsschaltung sein, die den Datenaustausch zwischen zwei Systemen ermöglicht, als auch eine spezielle Routine für die Kommunikation zwischen unterschiedlichen Programm-Modulen (‚Software-Interface'). Aufgabe der Schnittstelle ist die Umsetzung von Informationen in eine Form, die vom Empfängersystem verstanden wird.

Zu einer Hardware-Schnittstelle gehören neben der Elektronik auch Verbindungselemente wie Kabel und Stecker; ein Software-Interface muß als „Brücke" zwischen Programmteilen den Austausch von Parametern und Variablen vermitteln.

Interpreter = Interpreter
Die CPU eines Rechners verarbeitet Bitmuster nach vorgegebenen logischen Regeln – Menschen drükken ihre Gedanken in Begriffen, Wörtern und Symbolen aus. Diesen Unterschied sollen Programmiersprachen lösen, indem sie einerseits dem Benutzer entgegenkommen und andererseits für den Rechner unter Zwischenschaltung eines Übersetzungsprogramms verständlich sind.

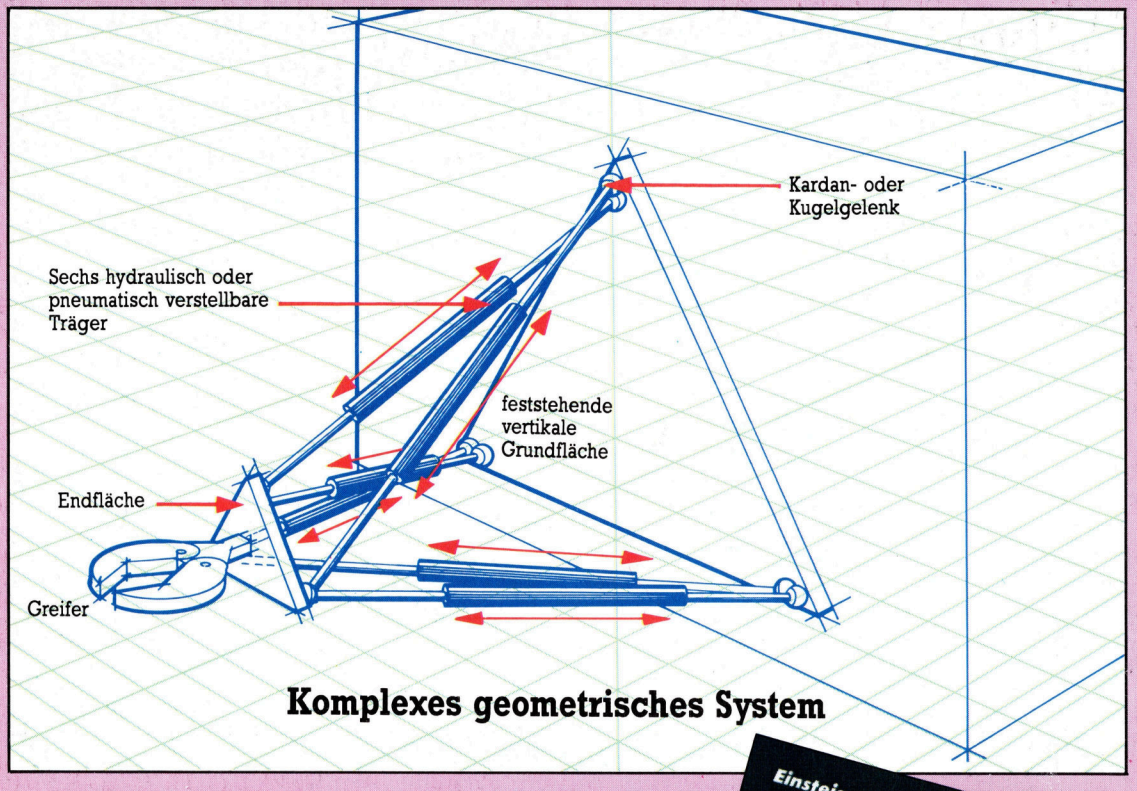
Bei der Eingabe eines Programms wandelt der Interpreter die einzelnen Anweisungen in eine kompakte Kommandoform um. Beim Programmlauf werden die Kommandos dann vom Interpreter in Maschinenbefehle umgesetzt und unmittelbar ausgeführt, ohne daß der erzeugte Code gespeichert wird.

Diese Arbeitsweise macht den Interpreter besonders geeignet für die Entwicklung von Programmen und für Testläufe. Sie sorgt allerdings, etwa beim BASIC-Interpreter, für die weithin bekannte Langsamkeit dieser Programmiersprache.

Bildnachweise
1541: Helen Zahorodnyj
1542: Steve Cross
1543: Courtesy of U.I.P.
1544, 1555, 1562 Ian McKinnell
1549: Kevin Jones
1557: Caroline Clayton
1559: Jojo
1560: Timothy Ginn
1564: Gabrielle Izen, Hotte Wurst
1565: Liz Dixon

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

computer kurs Heft 57



Komplexes geometrisches System

Entwurf

Neues Selbstbauprojekt für den Commodore, Spectrum und Acorn B: Miniroboter.

Die letzten Drei

Die Steuerung zwischen dem Debugger und dem analysierten Programm wird unter die Lupe genommen: Interruptmechanismus.

Hardware: Das „As“

„Jupiter Ace“, der preisgünstige Heimcomputer mit FORTH als Standardsprache.

Kreuzverweise

Bei der Verwaltung von Datenbanken sind Kreuzverweise oft sehr nützlich. Wir zeigen, wie es funktioniert.

Lichtschalter

Licht kann Informationen schneller übertragen als Strom. Gibt es in der Zukunft „Lichtcomputer“?