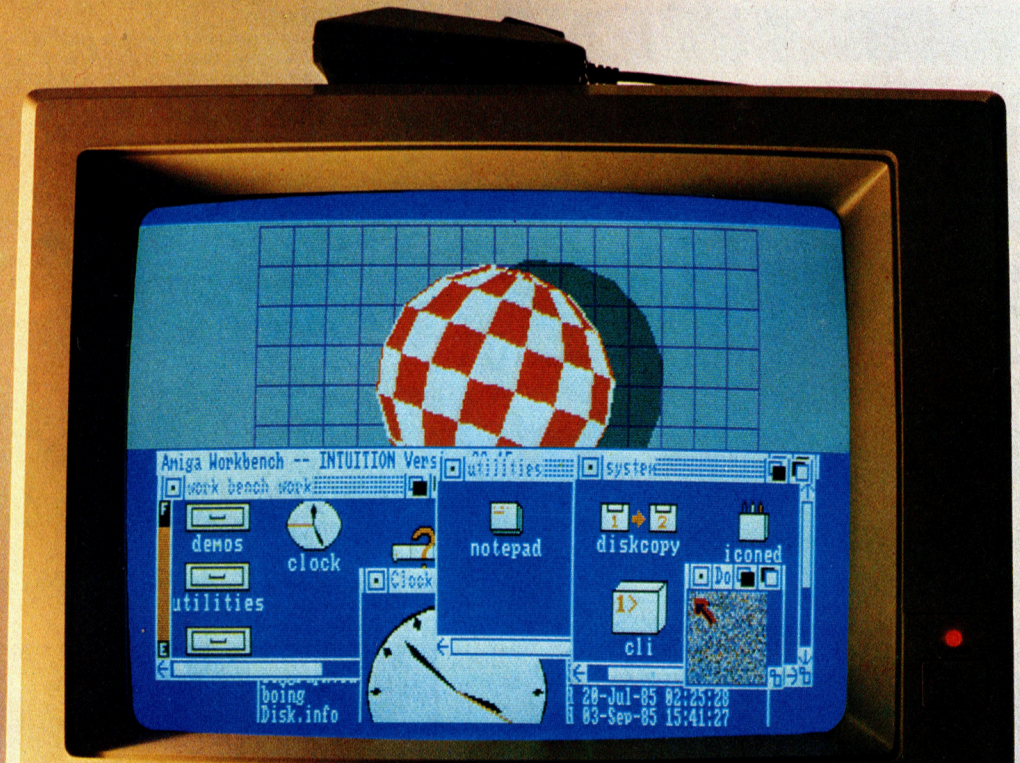
Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Heft 63



Amiga von Commodore

Sieg oder Platz

Bewegungssteuerung

CP/M von Spur zu Spur

computer kurs

Heft 63

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.

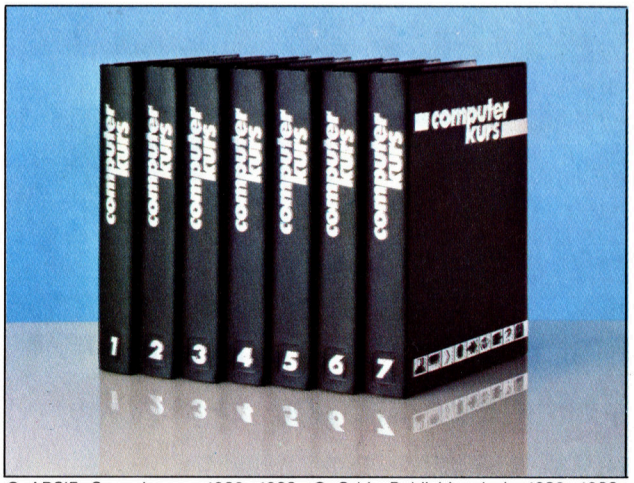
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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), Peter Aldick, Holger Neuhaus, 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

Das Top-Modell

Bereits vor der Präsentation in der Öffentlichkeit sorgte Commodores Amiga für beträchtliche Spannung. Seine innovativen Designs und seine Leistungsfähigkeit ließen aufhorchen.

Commodores Amiga steht für eine dramatische Entwicklung in der Evolution der Microcomputer. Die Verwendung von Spezial-Chips bei gleichzeitigem Einsatz der „Blitter"-Technologie stützt die Klang- und Grafikfähigkeiten des Rechners und entlastet zugleich die CPU.

Ursprünglich wurde der Amiga von der Amiga Corporation in den USA entwickelt. Commodore erwarb lediglich die Rechte an diesem Gerät. Da der Amiga für den Geschäftsbereich und den ernsthaften Anwender entwickelt wurde, Commodore aber in den USA vornehmlich als Hersteller kleinerer „Spielecomputer" betrachtet wird, entschied sich das Unternehmen für ein völlig neues Marketing.

Der Amiga in seinem Komplettgehäuse mit integriertem Laufwerk macht einen sehr professionellen Eindruck. Die Tastatur ist an der Unterseite mit Klappfüßen versehen, mit denen sich der Bedienungswinkel den Gegebenheiten des Arbeitsplatzes gemäß einstellen läßt. Wenngleich die Tastatur-Ergonomie nicht optimal ist, liegt sie doch über dem gewohnten Standard. Rechts neben der Tastatur ist ein Cursor-Steuerungsfeld, mit dem Maus-Funktionen nachgeahmt werden können.

Der Amiga wird mit integriertem 3 1/2-Zoll-Laufwerk geliefert, das eine Speicherkapazität von 880 KBytes bietet. Damit hat der Amiga auf einem Laufwerk Zugriffsmöglichkeit zu mehr Daten als mancher Rechner mit Doppel-Diskettenstation.

Entfernt man die vordere Abdeckung links vom Laufwerk, entdeckt man den Erweiterungsschacht. Standardmäßig ist der Amiga mit 256 KByte ausgestattet, er kann aber mit erweiterndem Speichermodul auf 512 KByte hochgerüstet werden. Seitlich rechts am Computer befindet sich ein Interface, über die der Rechner mit der Maximalkonfiguration verbunden werden kann, womit die Gesamtkapazität bei rund acht MegaByte liegt. An dieser Seite befinden sich ebenfalls zwei D-Stecker (9polig), an die Steuerknüppel bzw. die Maus angeschlossen werden.

Auf der Rückseite des Gehäuses sind die weiteren peripheren Schnittstellen angebracht, von links der Tastatur-Port, der Centronics-Drucker-Port sowie ein Interface für eine zweite Diskettenstation. Ferner verfügt der Amiga über eine RS232-Schnittstelle für Modem und andere serielle Peripheriegeräte sowie zwei Audiobuchsen. Es empfiehlt sich, sie direkt an den speziellen Amiga-Monitor oder – noch besser – an ein HiFi-System anzuschließen, um die herausragenden Klangmöglichkeiten des Computers optimal ausschöpfen zu können.

Die verbleibenden drei Ports sind für Video-Funktionen reserviert; darunter ist ein 23poliger Stecker für RGB-Monitore sowie einer für Composite-Video. Auch steht ein „Video In"-Anschluß zur Verfügung, mit dem Videobilder vom Videorecorder oder einem anderen Computer in den Amiga gespeist werden können. Künftige Hardware-Erweiterungen für den Amiga werden einen „Rahmentaster" umfassen (mit dem sich Bilder aus einer Video-Quelle digitalisieren lassen), die man dann seitenverkehrt, gedreht oder auf andere Art unter Software-Steuerung manipulieren kann.

Agnus, Daphne und Portia

Herz des Amiga ist ein Motorola 68000-Microprozessor, wie er auch bei Apples Macintosh und dem Atari 520ST Anwendung findet. Was den Amiga aber dabei auszeichnet, ist das Zusammenspiel der drei Custom-Chips untereinander und in Verbindung mit dem 68000. Die Agnus, Daphne und Portia genannten Chips steuern die Bildschirmdarstellung, die Sprites,

Alles unter Taste
Die Tastatur des Amiga gehört zum besten, was derzeit auf dem Markt ist. Dem neuen Trend folgend sind in die Tastatur numerische Tasten integriert, ferner zehn programmierbare Funktionstasten sowie andere Tasten, die für Kontrollfunktionen verwendet werden können.

Metacomco im Amiga

Das in Bristol ansässige Software-Haus Metacomco wurde 1981 gegründet und entwickelte System-Software für 16- und 32-Bit-Computer. Sein Personal-BASIC, für Digital Research entwickelt, wurde zum Standard für CP/M-86-Computer. Unlängst konzentrierte Metacomco seine Aktivitäten auf das Schreiben von 68000-Software. Dabei entstanden Versionen von PASCAL, LISP und C für den Sinclair QL, den Atari 520ST sowie 68000 Assembler und Software-Entwicklungsprogramme.

Metacomco wurde aufgefordert, das AmigaDOS zu schreiben, nachdem ein anderes Systemhaus die Aufgabe nicht lösen konnte. AmigaDOS basiert auf einem Multi-Tasking-Netzsystem, das als Tripos bekannt ist. Es wurde ursprünglich vor einigen Jahren als PhD-Projekt an der Universität Cambridge

und den Disketten-Eingang/Ausgang. Damit kann der 68000 alle weiteren Aufgaben mit hoher Geschwindigkeit ausführen. Agnus etwa ist mit einem eigenen Co-Prozessor versehen, ferner einem „bit image manipulator" (blitter), mit dem pro Sekunde eine Million Punkte bewegt werden. Das Ziehen von Linien und das Füllen von Konturen sind ebenfalls in die Schaltung des Amiga integriert. Das bedeutet: Amiga besitzt die Fähigkeit, Figuren so schnell zu bewegen, zu verändern oder zu modifizieren, daß die Illusion von fließender Bewegung erzeugt wird.

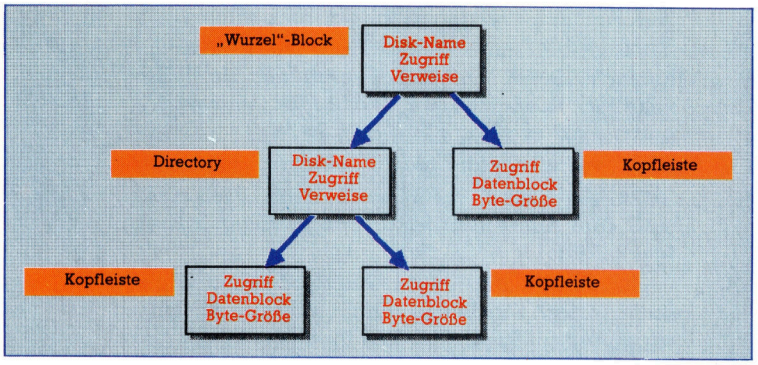
„Blitter Objekte"

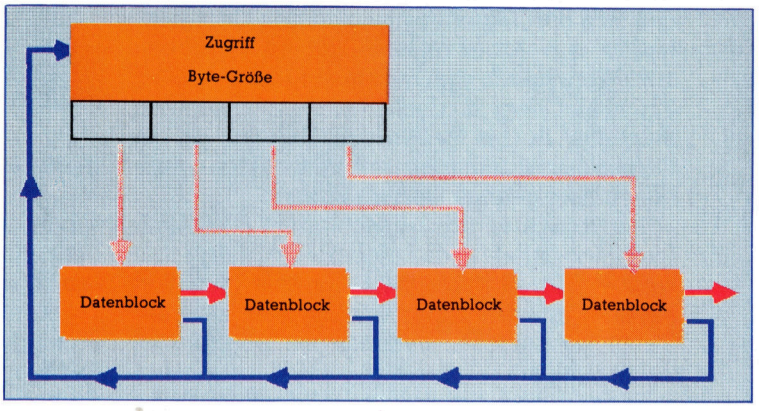
Zwei Sprite-Arten stehen zur Verfügung. Die sogenannten „Vsprites" sind hardwaregesteuert, wodurch sie mit hoher Geschwindigkeit auf dem Bildschirm bewegt werden können. Unter Verwendung des Blitters wird die zweite, auf Software-Basis gesteuerte Spriteart eingesetzt. Diese „Blitter Objekte" (Bobs) erzeugen noch komplexere Konturen und Farbgebungen, als es mit VSprites möglich ist.

Die Klangmöglichkeiten sind ebenfalls ungeheuer beeindruckend und in Teilen vielen kommerziellen Synthesizern zumindest ebenbürtig. Der Amiga kann digital aufgezeichnete Klänge reproduzieren, die sich daraufhin in der Tonhöhe manipulieren lassen, nach Belieben auch in Stereo. Standardmäßig ist das System mit Sprachsynthese ausgestattet.

Gesteuert wird der Amiga mit einem anwenderfreundlichen WIMP-System, Workbench benannt, das in der Art dem Macintosh und den Rechnern auf GEM-Basis entspricht. Files werden durch Ansteuerung der entsprechenden Piktogramme und anschließendem Druck auf den Knopf der Maus geladen. Darunter liegt das als AmigaDOS bekannte Betriebssystem, mit dem der Rechner wirklich „Multi-Tasking"-fähig ist. AmigaDOS wiederum ruft „Intuition" auf, jenen Firmware-Teil, der die Fenster und die Maus überwacht.

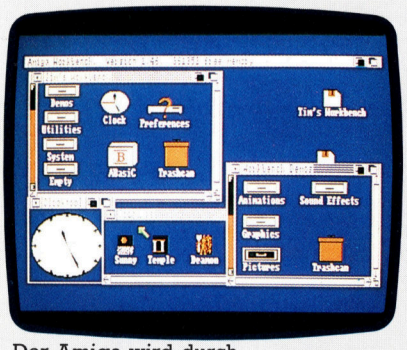
In der Praxis erlauben die Multi-Tasking-Möglichkeiten des Amiga den Betrieb mehrerer Anwendungen gleichzeitig und doch unab-





Bilder einer Ausstellung

Die Grafikmöglichkeiten des Amiga liegen weit über denen anderer Rechner dieser Preisklasse. Komplexe Animation und hochauflösende Grafiken machen den Amiga zu einer exzellenten Spielmaschine. Sein Macintosh-ähnliches WIMP-Betriebssystem wird ihn auch für Grafiker und Künstler attraktiv machen.



Der Amiga wird durch eine Piktogramm-Schnittstelle mit Namen Workbench gesteuert. Die vertrauten Fenster und Piktogramme sind hier dargestellt.

entwickelt. Dr. Tim King, Leiter der Metacomco-Entwicklungs-Abteilung, bekam den Auftrag, Tripos in nur einem Monat in ein Betriebssystem für den Amiga umzuwandeln.

Anschließend wurde Metacomco stärker in das Amiga-Projekt integriert. Man schrieb das Amiga BASIC, den mit dem Amiga gekoppelten BASIC-Interpreter als auch PASCAL und LISP. Das Haus erstellte auch einen Makro-Assembler und Entwicklungssysteme, die unter UNIX und MS-DOS laufen.

Wenngleich AmigaDOS derzeit die Netzmöglichkeiten noch nicht nutzt, handelt es sich bei Tripos um ein Netzwerk-Betriebssystem. Für die Zukunft ist die Vernetzung von bis zu 255 Amigas vorgesehen. Tim King sieht das System als Alternative zu Großrechensystemen. Das Netzwerk ist flexibel genug, um nötigenfalls weitere Rechner anschließen zu können.

Zurück und Vorwärts

Das AmigaDOS ist ungewöhnlich. Herkömmliche Systeme bedienen sich einer Directory, die die Namen aller Files auf der Diskette enthält und Hinweise auf die ersten Datenblocks gibt. Das AmigaDOS hingegen ist in Baumstruktur aufgebaut und verfügt über einen Satz von vor- und rückwärts zeigenden Verweisen, die die einzelnen Segmente der Diskette miteinander verbinden. Eine Directory in diesem Sinne gibt es nicht, sondern lediglich einen sogenannten „Wurzel"-Block mit Verzweigungspfeilen.

Die Kopfleiste des Files enthält mehrere Hinweise auf die Datenblocks im File sowie andere Informationen, etwa Byte-Größe und Datum des letzten Zugriffs. Die Datenblocks im File sind ferner durch Pfeile miteinander verknüpft. Jeder Datenblock enthält zudem einen Rückwärts-Verweis auf die Kopfleiste des Files, zu dem er gehört.

Dieses Pfeil- bzw. Verweis-System bietet einen entscheidenden Vorteil. Wird eine Diskette beschädigt und verfügt nur noch über einen einzigen „guten" Block, kann die komplette Diskettenstruktur aufgrund der Verweise neu geschaffen werden.

hängig voneinander. Das ist ein gewichtiges Argument für den Einsatz im Bürobereich, da es sich um den ersten Microcomputer dieser Preisklasse handelt, der solche Möglichkeiten bietet. Da die Verarbeitungsmöglichkeiten des Rechners durch sogenanntes „Time-Slice" entsprechend beeinträchtigt werden (jede Anwendung belegt eine gewisse Zugriffszeit), kommt bei mehreren „Multi-Tasking"-Anwendungen ein erheblicher Geschwindigkeitsverlust zustande.

Eines der Hauptprobleme für alle Computer-Hersteller in den USA, die in den Business-Bereich vordringen wollen, ist die Dominanz von IBM und die Abneigung der Business-Computer-Benutzer, neue Rechner auszuprobieren, eine Folge der Überfülle an IBM-Nachbauten. Wenngleich der Amiga dem IBM PC in Sachen Preis (er kostet nur die Hälfte) überlegen ist, versucht Commodore den Erfolg des Systems dadurch sicherzustellen, daß es mit einem 5¼" Laufwerk ausgestattet und entsprechende Übersetzungs-Software verfügbar ist, die den Rechner IBM-kompatibel macht. Commodore sagt dazu, daß damit auf dem Amiga Programme wie Lotus 1-2-3 lauffähig sind. Die Emulations-Technik übersetzt die Op-codes des 8088 in entsprechende für den 68000. Damit wird die Verarbeitungsgeschwindigkeit des Amiga verlangsamt.

In bezug auf Verarbeitungsgeschwindigkeit, Grafik- und Klangmöglichkeiten setzt der Amiga neue Standards. Aufgrund dieser Innovationen wird das Unternehmen mit seiner Philosophie, einen völlig neuen Markt zu eröffnen, richtig liegen. Ein neues Computersystem steht und fällt aber unverändert mit der dafür erhältlichen Software. Bei der Einführung in den USA standen für den Amiga lediglich acht Programme zur Verfügung. Manches indes weist darauf hin, daß weitere Softwarehäuser Programme für den Rechner schreiben; darunter auch Adaptionen von populären Macintosh-Programmen.

Commodore Amiga

ABMESSUNGEN

444 × 300 × 120 mm

ZENTRALEINHEIT

Motorola 68000, 8 MHz Taktgeschwindigkeit

SPEICHER

256 KByte, auf 8 MegaByte erweiterbar.

BILDSCHIRMDARSTELLUNG

Textauflösung maximal 80 × 25 Zeichen. Die vier Grafik-Modi bieten Auflösungen von 320 × 200 Punkten im 32-Farben-Modus und bis zu 640 × 400 Zeichen im 16-Farben-Modus.

SCHNITTSTELLEN

Zwei Maus-Anschlüsse, Erweiterungsbus, RS232-Schnittstelle, Centronics-Schnittstelle, zweiter Anschluß für Diskettenlaufwerk, RGB und Composite-Video-Anschlüsse.

TASTATUR

82 Tasten, einschließlich 10 Funktionstasten und numerischer Tastatur.

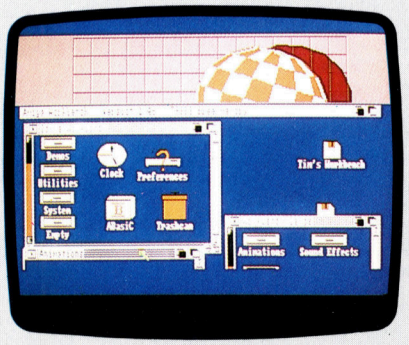
STÄRKEN

Der Amiga macht Multi-Tasking möglich. Die Grafik- und Klangfähigkeiten sind atemberaubend.

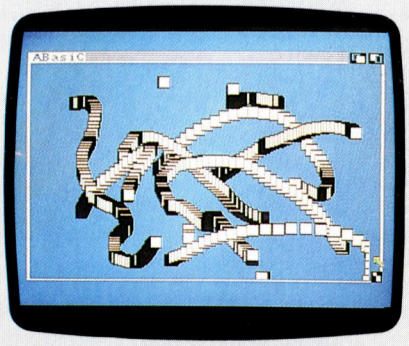
SCHWÄCHEN

Die versprochene Software existiert in wesentlichen Teilen noch nicht. Zudem hat Commodore noch keinen Zielmarkt für den Rechner definiert.

Die sich bewegenden Figuren, die hier gezeigt werden, wurden mit Hilfe der „Blitter Objekte" erstellt. Das sind Software-„Sprites", mit denen exakte Kontur- und Farbdefinitionen möglich sind. Hier nimmt der Hydrant vor einem Hund Reißaus.



Wie hier gezeigt, ist der Amiga ein Multi-Tasking-Rechner. Die Demonstration des springenden Balls und Workbench laufen gleichzeitig.



Wenngleich das Amiga-BASIC keine herausragende Implementierung der Sprache ist, unterstützt sie doch die meisten Besonderheiten des Rechners. Um dieses Malprogramm für die Maus zu erstellen, genügen fünf Programmzeilen.

Globaler Austausch

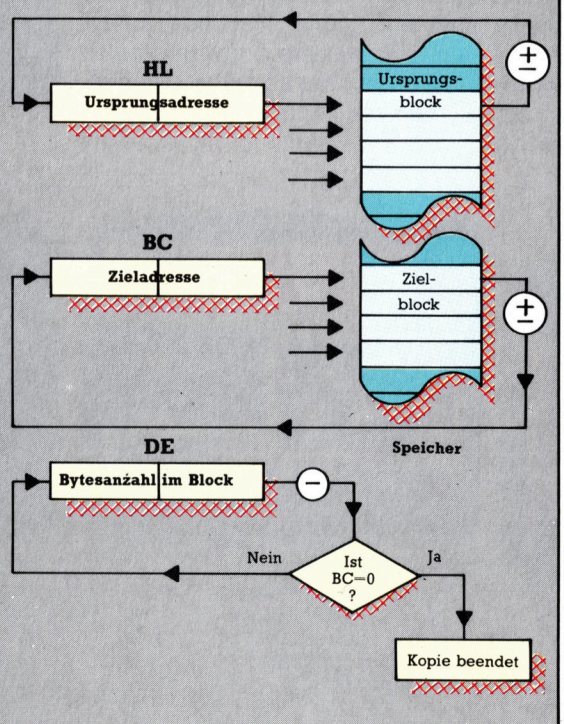
Für die Implementierung des Variablen-Austauschprogramms auf den Spectrum muß eine andere Methode als beim Acorn B oder C64 verwendet werden. Anstatt das Hilfsprogramm in einem unterschiedlichen Speicherbereich abzulegen, wird hier das Variablen-Austauschprogramm ans Ende angefügt.

Während das Variablen-Austauschprogramm das Programm durchläuft, legt es eine Kopie der geänderten Version in einem Bereich oberhalb RAMTOP ab.

Der erste Teil des BASIC-Programms ähnelt dem Variablen-Suchprogramm. Hier gibt es einige Zusatz-Variablen wie etwa ALTPROG, die auf den Anfang des für die Programmkopie reservierten Bereichs zeigt, oder ALTPOINTER, die die Position des nächsten Bytes im geän-

Automatisches Schreiben

LDIR und LDDR beim Z80 sind Block-Transferanweisungen, die mit automatischer Inkrementierung oder Dekrementierung arbeiten: Das HL-Register ist so initialisiert, daß es auf den Anfang des Ursprungsblockes zeigt, und DE muß auf den Anfang des bestimmungsblockes zeigen. BT enthält die Byteanzahl des Blockes. LDIR und LDDR kopieren dann das Ursprungsbyte in das Bestimmungsbyte, erhöhen oder verringern dabei automatisch HL und DE und verringern BC bis hin zum Wert 0, wenn die Kopie fertiggestellt ist.



Variablenaustauschprogramm

```
9000 INPUT "Name to search for? "; LINE
t$
9005 INPUT "Replace by?"; LINE r$
9010 FOR i=1 TO LEN (t$)
9020 IF t$(i)>="a" AND t$(i)<="z" THEN
LET t$(i)=CHR$ (CODE (t$(i))-32)
9030 NEXT i
9040 LET TokenforREM=234
9050 LET Quote=34
9060 LET Newline=13
9070 LET Underscore=95
9080 LET Number=14
9090 LET PROG=23635
9100 LET Textpointer=PEEK (PROG)+256*PEE
K (PROG+1)
9102 LET Altprog=46000
9105 LET Altpointer=Altprog
9110 LET Lineno=256*PEEK (Textpointer)+P
EEK (Textpointer+1)
9111 PRINT lineno
9120 IF Lineno>=9000 THEN  GO TO 9600
9130 LET q=2: GO SUB 9800
9135 LET Lengthaddr=Altpointer
9140 LET Nextline=Textpointer+2+PEEK (Te
xtpointer)+256*PEEK (Textpointer+1)
9150 LET q=2: GO SUB 9800
9160 LET Byte=PEEK (Textpointer): LET q=
1: GO SUB 9800
9170 IF Byte=Newline THEN  GO TO 9110
9180 IF Byte<>TokenforREM THEN  GO TO 92
20
9190 REM Copy REM  unaltered
9200 LET q=Nextline-Textpointer: GO SUB
9800
9210 GO TO 9110
9220 IF Byte<>Quote THEN  GO TO 9280
9230 REM Copy anything between quotes, b
ut stop at end of line in case of unmatc
hed quote
9235 LET q=1
9240 IF PEEK (Textpointer+q-1)=Newline T
HEN  GO SUB 9800: GO TO 9110
9250 IF PEEK (Textpointer+q-1)=Quote THE
N  GO SUB 9800: GO TO 9160
9260 LET q=q+1
9270 GO TO 9240
9280 REM Copy 5-byte binary number
9290 IF Byte=Number THEN  LET q=5: GO SU
B 9800: GO TO 9160
9310 REM First character of name must be
 upper or lower case letter
9320 IF Byte>=CODE ("A") AND Byte<=CODE
("Z") THEN  LET c$=CHR$ (Byte): GO TO 93
70
9330 REM Use upper case instead of lower
 case
9340 IF Byte>=CODE ("a") AND Byte<=CODE
("z") THEN  LET c$=CHR$ (Byte-32): GO TO
 9370
9360 GO TO 9160
9370 LET n$=""
9380 LET n$=n$+c$
9400 REM Letter, digit or underscore aft
er first character of name
9410 IF PEEK (Textpointer)>=CODE ("A") A
ND PEEK (Textpointer)<=CODE ("Z") THEN
LET c$=CHR$ (PEEK (Textpointer)): LET Te
xtpointer=Textpointer+1: GO TO 9380
9420 REM Use upper case instead of lower
 case
9430 IF PEEK (Textpointer)>=CODE ("a") A
ND PEEK (Textpointer)<=CODE ("z") THEN
LET c$=CHR$ (PEEK (Textpointer)-32): LET
 Textpointer=Textpointer+1: GO TO 9380
9440 IF PEEK (Textpointer)>=CODE ("0") A
ND PEEK (Textpointer)<=CODE ("9") THEN
LET c$=CHR$ (PEEK (Textpointer)): LET Te
xtpointer=Textpointer+1: GO TO 9380
9450 IF PEEK (Textpointer)=Underscore TH
EN  LET c$=CHR$ (PEEK (Textpointer)): LE
T Textpointer=Textpointer+1: GO TO 9380
9460 REM End with $ for string variable
9470 IF PEEK (Textpointer)=CODE ("$") TH
EN  LET n$=n$+"$": LET Textpointer=Textp
ointer+1: GO TO 9500
9480 REM ( if array or function
9490 IF PEEK (Textpointer)=CODE ("(") TH
EN  LET n$=n$+CHR$ (PEEK (Textpointer)):
 LET Textpointer=Textpointer+1
```

```
9500 IF n$=t$ THEN  LET n$=r$
9505 LET Altpointer=Altpointer-1
9510 FOR p=1 TO LEN (n$)
9520 POKE Altpointer,CODE (n$(p))
9530 LET Altpointer=Altpointer+1
9540 NEXT p
9550 IF n$<>r$ THEN  GO TO 9160
9560 LET LengthLow=PEEK (Lengthaddr)+LEN
 (r$)-LEN (t$)
9570 IF LengthLow>255 THEN  LET Lengthlo
w=LengthLow-256: POKE Lengthaddr+1,1+PEE
K (Lengthaddr+1)
9580 POKE Lengthaddr,LengthLow
9590 GO TO 9160
9599 REM Prepare to move altered program
 back to main program area
9600 LET Oldlength=Textpointer-(PEEK (PR
OG)+256*PEEK (PROG+1))
9610 LET Newlength=Altpointer-Altprog
9620 POKE 45060,Newlength-256*INT (Newle
ngth/256)
9630 POKE 45061,INT (Newlength/256)
9660 POKE 45056,Textpointer-256*INT (Tex
tpointer/256)
9670 POKE 45057,INT (Textpointer/256)
9680 IF Oldlength=Newlength THEN  RANDOM
IZE USR (45084)
9690 IF Oldlength<Newlength THEN  LET X=
Newlength-Oldlength
9700 IF Oldlength>Newlength THEN  LET X=
Textpointer-(Oldlength-Newlength)
9710 POKE 45058,X-256*INT (X/256)
9720 POKE 45059,INT (X/256)
9730 IF Oldlength<Newlength THEN  RANDOM
IZE USR 45062
9740 IF Oldlength>Newlength THEN  RANDOM
IZE USR 45074
9800 FOR p=Textpointer TO Textpointer+q-
1
9810 POKE Altpointer,PEEK (p)
9820 LET Altpointer=Altpointer+1
9830 NEXT p
9840 LET Textpointer=p
9850 RETURN
9900 SAVE "REPLACE" LINE 9910: SAVE "REP
MC"CODE 45064,37: STOP
9910 CLEAR 45055: LOAD "REPMC"CODE
```

Ladeprogramm

```
  10 CLEAR 45055
  20 LET a=45062
  30 FOR l=1000 TO 1040 STEP 10
  40 LET s=0
  50 FOR a=a TO a+7
  60 READ b
  70 POKE a,b
  80 LET s=s+b
  90 NEXT a
 100 READ c
 110 IF s<>c THEN  PRINT "DATA ERROR IN
LINE ";l: STOP
 120 NEXT l
1000 DATA 42,0,176,237,75,2,176,205,913
1010 DATA 85,22,24,10,42,0,176,237,596
1020 DATA 91,2,176,205,229,25,33,176,937
1030 DATA 179,237,91,83,92,237,75,4,998
1040 DATA 176,237,176,207,255,0,0,0,1051
```

Austauschprogramm

```
0000              MKROOM  EQU     $1655
0000              RCLAM1  EQU     $19E5
0000              T0      EQU     $B000
0000              T1      EQU     $B002
0000              T2      EQU     $B004
0000              ALTPRG  EQU     $B3B0
0000              PROG    EQU     $5C53
B006                      ORG     $B006
B006  2A00B0      UP      LD      HL, (T0)
B009  ED4B02B0            LD      BC, (T1)
B00D  CD5516              CALL    MKROOM
B010  180A                JR      COPY
B012  2A00B0      DOWN    LD      HL, (T0)
B015  ED5B02B0            LD      DE, (T1)
B019  CDE519              CALL    RCLAM1
B01C  21B0B3      COPY    LD      HL, ALTPRG
B01F  ED5B535C            LD      DE, (PROG)
B023  ED4B04B0            LD      BC, (T2)
B027  EDB0                LDIR
B029  CF                  RST     8
B02A  FF                  DB      $FF
```

derten Programm kontrolliert. Aufgrund dieser Änderungen muß das Programm kopiert werden. Das erfolgt durch die Unterroutine in Zeile 9800, die die durch q spezifizierte Anzahl von Bytes kopiert und die Zeiger für das alte und neue Programm aktualisiert.

Die Variablennamen werden ab Zeile 9500 kopiert. Hat sich der Name inzwischen geändert, so werden die beiden Längen-Bytes (am Anfang der Zeile) mit den neuen Werten aktualisiert.

Nach Verändern und Kopieren des gesamten Programms errechnet das BASIC-Programm die vom Maschinencode für ein Rückkopieren des geänderten Programms erforderlichen Werte und POKEt diese in die Speicherplätze für den Maschinencode.

ROM-Routine MAKEROOM

Sind das alte und das neue Programm gleich lang, kann das neue Programm an die Position des alten kopiert werden. Hier ist die einzige für das Maschinencode-Programm erforderliche Information die Programmlänge.

Andernfalls muß durch Verschieben der Variablen-Austauschroutine nach oben Platz im Programmbereich geschaffen werden. Hierzu wird die ROM-Routine MAKEROOM auf Adresse 1655 hex aufgerufen. Vor dem Aufruf muß das HL-Registerpaar die Adresse hinter der Stelle enthalten, an der zusätzlicher Platz geschaffen werden soll. Das BC-Registerpaar muß die Länge des nötigen Platzes ausweisen.

Ist das neue Programm kürzer als das alte, muß das Variablen-Austauschprogramm nach unten geschoben werden. Dazu wird die ROM-Unterroutine RECLAIM-1 auf Adresse 19E5 hex benutzt. Vor dem Aufruf muß das HL-Registerpaar die Adresse des ersten alleinstehenden Bytes und das DE-Registerpaar die Adresse des ersten wiederzugewinnenden Bytes enthalten. Der Wert für HL ist wieder der Endwert von TEXTPOINTER. Der Wert für DE ist die Differenz zwischen alter und neuer Länge, subtrahiert von TEXTPOINTER.

Das geänderte Programm wird über den Befehl LDIR ins Hauptprogramm zurückkopiert. Die Startadresse des geänderten Programmbereichs wird in HL, die Startadresse des Hauptprogrammbereichs in DE und die Länge des geänderten Programms in BC geladen. Mit LDIR wird das gesamte geänderte Programm Byte für Byte verschoben.

Die letzten beiden Zeilen im Assembler-Programm benutzen eine andere ROM-Routine (bei Adresse 8). Das ist die „Bericht"-Routine, die Fehlermeldungen und ähnliches druckt. Sie wird über RST 8 aufgerufen. Der erstellte Bericht wird durch das Byte nach RST 8 spezifiziert. Der Wert des Bytes ist um 1 kleiner als die Bericht-Zahl, das heißt, FF hex oder −1 ergibt OK oder PROGRAMM FINISHED, 0 ergibt NEXT without FOR usw.

Systemkopie

Mit dem Diskettenbefehl SYSGEN kann CP/M die eigenen Routinen auf andere Disketten kopieren. Wir zeigen den internen Informationsfluß und stellen CCP, BIOS und BDOS vor.

In dieser Serie wurden bisher nur CP/M-Befehle untersucht, die den normalen Betrieb steuern. Es wurde bereits erklärt, wie die verschiedenen Dateitypen von CP/M geladen, editiert, gesichert und kopiert werden. CP/M selbst jedoch blieb „unsichtbar". Es wird im Directory nicht aufgeführt und läßt sich auch nicht mit Standardbefehlen ansprechen.

Die Übertragung von CP/M auf andere Disketten ist jedoch außerordentlich wichtig, da damit nicht nur Sicherheitskopien der Systemdiskette hergestellt werden, sondern sich auch Spezialversionen des CP/M auf andere Disketten übertragen lassen. Ein gutes Beispiel dafür ist Dr. LOGO des Schneider CPC. Dieses Programm arbeitet mit einer CP/M-Version, die zusätzliche, vielerorts nützliche Editier- und Farbbefehle enthält.

CP/M ist leicht zu kopieren. Bei Eingabe des Diskettenbefehls SYSGEN erscheinen je nach Systemkonfiguration Fragen auf dem Bildschirm, mit deren Hilfe Sie den Kopiervorgang steuern können.

Das hört sich so einfach an, daß man sich fragt, warum dafür ein Spezialbefehl nötig ist, warum CP/M und die integrierten Befehle nicht im DIRectory erscheinen und wie das CP/M Directory überhaupt aussieht. Wir gehen daher ausführlicher auf die internen Abläufe des Betriebssystems ein.

Das ins RAM geladene System besteht aus drei Teilen, die unterschiedliche Funktionsbereiche steuern. Der Anwender hat hauptsächlich mit dem „Console Command Processor" (CCP – Befehlsinterpreter) zu tun, der Informationen zum Monitor sendet und die per Tastatur eingegebenen Befehle interpretiert.

Hinter den Kulissen sieht es jedoch anders aus. Der CCP interpretiert zunächst alle eingegebenen Zeichen als CP/M-Befehle und reicht sie über das BIOS (Basic Input/Output System – Ein/Ausgabesystem) an den CCP-Buffer weiter. Der CCP prüft dann, ob die Eingabe mit einem der integrierten CP/M-Befehle übereinstimmt und führt ihn in diesem Fall sofort aus. Handelt es sich nicht um eine der „eingebauten" Befehle, nimmt der CCP an, daß es sich um einen Diskettenbefehl handelt und sucht nun auf der Systemdiskette.

CP/M führt Regie

Wird der Befehl gefunden, lädt das System das zugehörige Modul in den Arbeitsspeicher und führt es aus – falls nicht, wird der eingegebene Name in Großbuchstaben dargestellt, gefolgt von einem Fragezeichen: die CP/M-Meldung für „Datei nicht gefunden". Die Umwandlung in Großbuchstaben wird vom CCP veranlaßt, damit alle Befehlsnamen dem CP/M-Standardformat entsprechen.

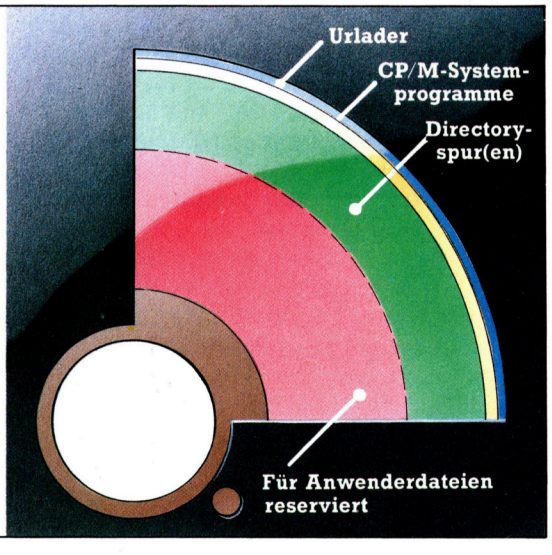
Das BIOS besteht aus einer Reihe von Routinen (Peripherietreiber genannt), die die Ein- und Ausgabefunktionen von CP/M steuern. Ihre Hauptaufgabe ist die Verwaltung der angeschlossenen Peripheriegeräte (darunter auch Tastatur und Bildschirm – das CCP sendet und interpretiert alle Informationen über das BIOS). Das System setzt dafür die vom CCP gelieferten Parameter ein. Bei einer Anpassung von CP/M auf andere Maschinen muß daher das BIOS geändert werden, da die Methoden der Peripherieverwaltung von System zu System große, keineswegs zu vernachlässigende Unterschiede aufweisen.

Das BDOS (Basic Disk Operating System – Diskettentreiber) ist der dritte Bereich von CP/M. Das BDOS verwaltet die auf der Diskette gespeicherten Dateien, teilt den Diskettenplatz zu und aktualisiert die Directory. Bevor wir uns jedoch dem BDOS zuwenden, untersuchen wir zunächst, wie der Disketteninhalt organisiert ist.

Vom Standpunkt der Hardware aus enthält eine Diskette 40 Spuren (Tracks). Neuere CP/M-Versionen teilen die Spuren per Software weiterhin in sogenannte „weiche" Sektoren

Spur halten

Auf einer CP/M-Datendiskette steht der meiste Platz zwar dem Anwender zur Verfügung, doch reserviert das System sich die äußersten Spuren für eigene Zwecke. Spur Null enthält den Urlader, der das CP/M beim Einschalten des Geräts oder bei einem Reset automatisch in den Arbeitsspeicher lädt. Auf Spur eins und zwei befindet sich das CP/M-System selbst. Darauf folgen Directoryspuren, deren Dateikontrollblöcke die Sektoren mit Dateien angeben.



(Soft Sectors) auf. Ältere Diskettensysteme arbeiten noch mit „harten" Sektoren, die von der Hardware mechanisch gelesen werden. Auf den äußeren Spuren stehen natürlich mehr Sektoren zur Verfügung als auf den inneren. Die meisten Spuren werden zur Speicherung von Informationen eingesetzt.

Auf Disketten im 5 1/4-Zoll-Format sind drei Spuren (0 bis 2) für CP/M reserviert. Sie liegen am äußeren Rand der Diskette und enthalten den „Urlader" für den Systemstart („Bootstrap Loader" oder auch „Resident Monitor" genannt), mit dem sich CP/M selbst in den Arbeitsspeicher laden kann. Auch CP/M liegt in diesem Bereich. Mit dem Befehl SYSGEN läßt sich die Information dieser drei Spuren auf andere Disketten übertragen.

Die Sektoren (auch „Records" genannt) speichern je 128 Datenbytes. Bis zu 128 „Records" lassen sich zu einem „Unit" (Einheit) zusammenstellen. Eine CP/M-Datei kann bis zu 16 Units enthalten und eine Größe von maximal 128 * 128 * 16 Bytes (=256 KBytes) erreichen.

Da sich nur extrem kleine Dateien in einem einzigen Record unterbringen lassen und es außerdem nicht möglich ist, alle Records in sequentieller Reihenfolge auf der Diskette zu speichern, muß das Verwaltungssystem wissen, auf welche Sektoren der Diskette eine Datei aufgeteilt wurde.

Diese Information ist im sogenannten Dateikontrollblock (FCB – „File Control Block") gespeichert, der sich auf der vierten (Directory) Spur einer 5 1/4-Zoll-Diskette befindet. Die Verwaltung dieses Blocks erledigt das BDOS. Ein FCB besteht aus 33 Datenbytes, über die das BDOS alle auf der Diskette gespeicherten Dateien identifizieren und finden kann.

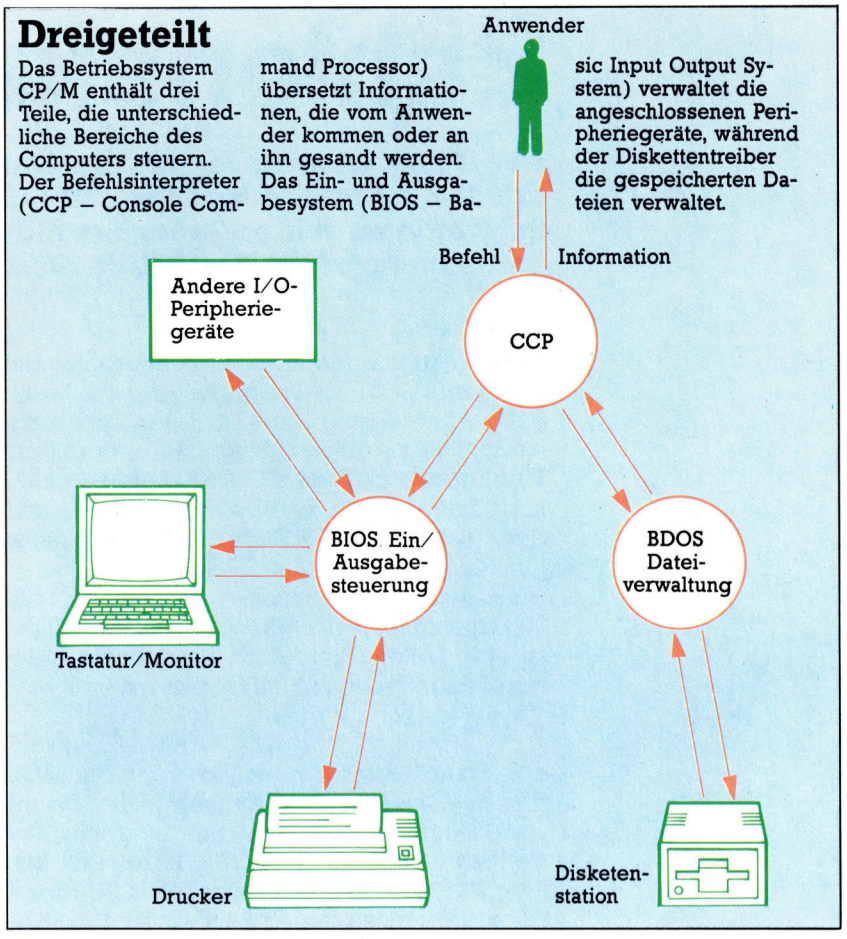
Spurensicherung

Das erste Byte des FCB gibt die aktuelle Laufwerksnummer an. Die nächsten elf Bytes enthalten den Dateinamen und den Dateityp (die Namenserweiterung). Byte 12 gibt die augenblickliche „logische Extension" an (Abschnitte zu je 16 KBytes, in die die Dateien intern aufgeteilt werden), während Byte 13 und 14 für das System reserviert sind. Byte 15 zeigt, wieviel Datenblöcke die aktuelle Extension einer Datei enthält – daraus errechnet sich die exakte Anzahl der Records einer Datei. Die nächsten 16 Bytes enthalten die Belegungstabelle der Datei, gefolgt von einem Byte mit der Nummer des zuletzt angesprochenen Blocks. Den Abschluß bilden drei Bytes, die die Nummer des nächsten Records angeben, falls die Datei größer als 16 KByte ist.

Wenn das CCP einen Befehl zur Eröffnung einer Datei erhält, legt es in dem für Diskettenbefehle reservierten Zwischenspeicher einen neuen FCB an. Findet das BDOS nun eine Datei, deren Namen mit dem vom CCP angegebenen übereinstimmt, werden auch die weiteren Informationen darin eingetragen. Dieser neue FCB im RAM läßt sich schnell vom BDOS aktualisieren, wenn die angesprochene Datei durch CP/M-Befehle verändert wird. Nach Abschluß aller Vorgänge wird die endgültige Version des FCB mit den geänderten Daten über Recordzahl und Sektorenzuteilung vom BDOS auf die Diskette geschrieben.

Wir können nun den Ablauf beim Aufruf einer Datei zusammenfassen: Das CCP interpretiert das erste Wort, das es vom Terminal erhält, als CP/M-Befehl. Befindet sich dieser Befehl nicht im Arbeitsspeicher, sendet das CCP den Namen per BIOS an das BDOS. Das BDOS durchsucht nun die Directoryspur der Diskette, bis es den Namen in einem der FCBs findet. Wenn es sich dabei um eine Befehlsdatei (.COM) handelt, stellt das BDOS über die Belegungstabelle fest, in welchen Diskettensektoren die Datei gespeichert ist, spricht diese Sektoren nacheinander an, kopiert sie in den Arbeitsspeicher und führt den Befehl dann automatisch aus.

Dabei kann es vorkommen, daß das BDOS eine weitere Datei suchen und laden muß. Wenn sich die Datei im Arbeitsspeicher befindet, ergänzt das BDOS die Daten des FCB, der vom CCP im RAM angelegt wurde. Bei einer Bearbeitung der Datei wird dieser FCB automatisch auf den neuesten Stand gebracht und auf die Diskette geschrieben.



Zahlenakrobatik

Das Array ist zwar die einzige Datenstruktur von FORTRAN, doch erhält die Sprache viel Flexibilität durch den komplexen Einsatz von Subroutinen. Wir untersuchen die speziellen Techniken für die Zahlenverarbeitung mit Hilfe dieser Sprache.

Das Array ist die einzige Datenstruktur, die dem FORTRAN-Programmierer zur Verfügung steht. Wie in BASIC müssen Arrays vor ihrem Einsatz deklariert werden – direkt am Programmanfang, da FORTRAN eine compilierte Sprache ist. Diese Deklaration erfolgt mit dem Befehl DIMENSION, der die gleiche Funktion hat wie DIM in BASIC.

Bei systemvorgegebenen Datentypen (alle Variablennamen, die mit I, J, K, L, M oder N anfangen, stellen Ganzzahlen dar und alle anderen Namen reelle Zahlen) reserviert

```
DIMENSION A(20), I(50)
```

den Speicherplatz für ein Array mit bis zu 20 reellen Zahlen und ein weiteres Array für bis zu 50 Ganzzahlen. Befehle, die die Systemeinstellung verändern (INTEGER, REAL, DOUBLE PRECISION, COMPLEX oder LOGICAL) müssen zusätzlich zu der Anweisung DIMENSION gegeben werden. Arrays lassen sich aber auch gleich in der Typendeklaration definieren:

```
INTEGER ARR1(20)
REAL NUMS(30)
```

Der erste Befehl reserviert Platz für ein Array mit 20 Ganzzahlen und der zweite für 30 reelle Zahlen.

Zweidimensionale Arrays sind möglich, einige Sprachversionen erlauben sogar drei oder mehr Dimensionen. Zweidimensionale Arrays werden auf übliche Weise deklariert:

```
DIMENSION ARR(20,30)
INTEGER ARR(20,30)
```

Jede Ganzzahlkonstante oder Variable läßt sich als Arrayindex einsetzen, beispielsweise ARR(3,4) oder INTARR(I).

FORTRAN IV verarbeitet zwar keine Zeichenstrings, doch stehen bei FORTRAN 77 dafür andere Möglichkeiten zur Verfügung (mit FORTRAN 77 werden wir uns in der nächsten Folge genauer beschäftigen). Zeichen werden wie Ganzzahlen in normalen Ganzzahlvariablen gespeichert. In der letzten Folge haben wir gesehen, wie sich ein oder mehr Zeichen in Ganzzahlvariablen unterbringen lassen. Zwar kommt das FORTRAN-Ganzzahlenarray einem Zeichenstring sehr nahe, doch ist es immer noch recht unbefriedigend, wenn man die Stringverarbeitungsmöglichkeiten von BASIC gewöhnt ist.

Außer den Befehlen IF und GOTO ist die DO-Schleife die wichtigste Steuerstruktur von FORTRAN. Sie funktioniert wie die FOR... NEXT-Schleife in BASIC und hat folgendes Format:

```
DO Sn (intvar) = Startwert, Endwert,
Schrittwert
.....
Sn (letzter Befehl des zu wiederholenden
Blocks)
```

Sn ist die Nummer des letzten Schleifenbefehls. Schleifenzähler (intvar) kann jede Ganzzahlvariable sein. Die Anfangs-, End- und Schrittwerte sind angegeben. In BASIC würde die DO-Schleife so aussehen:

```
FOR intvar = Startwert TO Endwert STEP
Schrittwert
......
NEXT intvar
```

Folgende Befehle setzen ein Array auf Null:

```
    DO 100 I = 1,20,1
100 ARR(I)=0
```

Der Schrittwert Eins darf weggelassen werden:

```
    DO 100 I = 1,20
100 ARR(I)=0
```

Die numerierte Anweisung, die das Schleifenende kennzeichnet, kann jeder ausführbare Befehl außer IF, GOTO oder DO sein.

Manchmal ist es praktisch, keine feste Schleifenendanweisung zu haben, da sich der Code damit komprimieren läßt. Diese Möglichkeit kann aber auch zu schweren Fehlern führen, und besonders bei verschachtelten Schleifen werden Programme dann völlig unverständlich. Die folgende Initialisierung eines zweidimensionalen Arrays ist jedoch legal:

```
    DO 100 I = 1,20
    DO 100 J = 1,30
100 ARR(I,J)=0
```

(Beachten Sie, daß bei beiden Schleifen die gleiche Endanweisung verwendet wird.)

Oft wird der „Leerbefehl" CONTINUE eingesetzt, um das Schleifenende zu markieren und den Code einfacher zu gestalten. CONTINUE ist ein FORTRAN-Befehl, der keine Wirkung hat und an jeder beliebigen Position eingesetzt werden kann. Jede DO-Schleife muß normalerweise mit einem eigenen CONTINUE abge-

schlossen sein. Das vorige Beispiel sieht dann in ausführlicher Schreibweise so aus:

```
      DO 101 I = 1,20
      DO 100 J = 1,30
      ARR(I,J)=0
  100 CONTINUE
  101 CONTINUE
```

Damit FORTRAN-Programme generell übersichtlicher werden, verlangen einige Programmierer sogar, daß alle DOs, IFs und GOTOs mit einem CONTINUE abgeschlossen werden.

Das speziell für Mathematik ausgelegte FORTRAN besitzt Anweisungen, die besonders gut für den Umgang mit Zahlenarrays geeignet sind. So kann mit einem einzigen READ- oder WRITE-Befehl ein ganzes ein- oder zweidimensionales Array gelesen oder geschrieben werden, wobei FORMAT den Aufbau jeder Zeile oder jedes Datensatzes angibt. Zum Beispiel:

```
      DIMENSION IARR1(20)
      .....
      WRITE(1,10)IARR1
```

veranlaßt die Ausgabe des gesamten Inhalts von IARR1 (20 Ganzzahlen).

10 FORMAT (20I4)

erzeugt 20 vierstellige Ganzzahlen in einer Zeile;

```
      10 FORMAT (I4)
```

erzeugt eine Ganzzahl per Zeile;

```
      10 FORMAT (5I4)
```

Wurzelziehen

```
C     EIN FORTRAN PROGRAMM, DAS SETS VON
C     DREI WERTEN A,B,C, LIEST, DIE DIE
C     KOEFFIZIENTEN DER QUADRATISCHEN
C     GLEICHUNG A*X**2+B*X+C=0 ÜBER DIE
C     ÜBLICHE FORMEL DARSTELLEN
      FUNCTION DISCR(A,B,C)
      DISCR=B*B-4.0*A*C
      RETURN
      END
      SUBROUTINE SOLVE(A,B,C,X1,X2,NROOT)
      D=DISCR(A,B,C)
C     ZWAR LIESSE SICH HIER EIN
C     ARITHMETISCHES IF
C     EINSETZEN, DOCH IST ES NICHT
C     EMPFEHLENSWERT
      IF (D.GE.0.0)GOTO 100
C     DA D UNTER NULL LIEGT, IST DIE WURZEL
C     KOMPLEX
      NROOT=0
      X1=0.0-B/(2.0*A)
      X2=SQR(ABS(D))
      GOTO 300
  100 IF (D.GT.0.0)GOTO 200
C     D IST NULL, DAHER GLEICHE WURZELN
      NROOT=1
      X1=0.0-B/(2.0*A)
      X2=X1
      GOTO 300
C     D IST POSITIV, DAHER ZWEI WURZELN
      NROOT=2
  200 X1=(0.0-B+SQRT(D))/2.0*A
      X2=(0.0-B-SQRT(D))/2.0*A
  300 RETURN
      END
C     ERST DIE ANZAHL DER SETS LESEN
C     READ(2,10)NVAL
C     ÜBERSCHRIFTEN ANZEIGEN
      WRITE(3,11)
      DO 500 L=1, NVAL
C     DREI WERTE LESEN UND PRÜFEN, OB SIE
C     GEEIGNET SIND
  100 READ (1,12) A,B,C
      IF (A.EQ.0.0) GOTO 500
C     DIE WERTE SIND GEEIGNET, DAHER
C     SUBROUTINE AUFRUFEN
      CALL SOLVE(A,B,C,X1,X2, NROOT)
      IF (NROOT.GT.0)GOTO 200
C     KOMPLEXE WURZELN
      WRITE (1,13)A,B,X1,X2
      GOTO 500
  200 IF (NROOT.GT.1)GOTO 300
C     GLEICHE WURZELN
      WRITE(1,14)A,B,C,X1,X2
      GOTO 500
C     ZWEI WURZELN
  300 WRITE(1,15)A,B,C,X1,X2
      GOTO 500
C     UNGÜLTIGE WERTE FÜR A, B UND C
  400 WRITE(1,16)A,B,C
  500 CONTINUE
      STOP
C     FORMAT BEFEHLE
   10 FORMAT (I4)
   11 FORMAT(1H ,8X,1HA,8X,1HB,8X,1HC,8X,
     4HTYPE,8X,2HX1,8X,2HX2)
   12 FORMAT(1H ,3F7,2)
   13 FORMAT(1H ,4X,3(F7.2,2X),4X,7HCOMPLEX,
     X,F7.2,3X,F7.2)
   14 FORMAT(1H ,4X,3(F7.2,2X),4X,5HEQUAL3X,
     F7.2,3XF7.2)
   15 FORMAT(1H ,4X,3,(F.2.2X),4X,
     7HUNEQUAL,X,
     F7.2.3X,F7.2)
   16 FORMAT(1H ,4X,3,(F7.2,2X),4X,7HINVALID)
      END
```

Durchschnittsberechnung

```
C     EINE FORTRAN-FUNKTION, DIE
C     DEN DURCHSCHNITT
C     (IN REELLEN ZAHLEN) EINES STETS MIT N
C     GANZZAHLEN
C     DES GANZZAHLENARRAYS IARR BERECHNET
      FUNCTION AVGE (IARR,N)
      DIMENSION IARR (N)
      SUM=0
      DO 100 I=1,N
      SUM=SUM+FLOAT (IARR(I))
  100 CONTINUE
      AVGE=SUM/FLOAT (N)
      RETURN
      END
```

vier Zeilen mit je fünf Zahlen und

```
10 FORMAT (20A1)
```

einen String von 20 Zeichen in einer Zeile.

Beim Einlesen oder Ausgeben von zweidimensionalen Arrays entsteht oft ein Problem, da FORTRAN, im Gegensatz zu anderen Sprachen, zuerst die Spalten und dann erst die Zeilen speichert. Wenn Sie hier nicht aufpassen, entsteht eine versetzte Version des Arrays. Es gibt jedoch eine spezielle Form der DO-Schleife, die „implizierte Schleifenanweisung", die sich gut für die Arbeit mit einem Teilarray eignet. Implizierte Schleifenanweisungen werden mit dem Variablennamen in die Befehle READ oder WRITE eingefügt. Die folgenden Zeilen füllen beispielsweise das Stringarray IARR2(10,20) mit zehn Strings zu je 20 Zeichen:

```
    DO 100 I=1,10
    READ (1,10) (IARR2(I,J), J=1,20)
10 FORMAT (20A1)
100 CONTINUE
```

FORTRAN bietet die Möglichkeit, mit Subroutinen zu arbeiten – wenn auch mit einigen Einschränkungen. Die Subroutinen sind völlig unabhängig und arbeiten hauptsächlich mit lokalen Variablen. Sie werden entweder an das Ende des Hauptprogramms angehängt oder separat geschrieben und compiliert. Subroutinen können durchaus die gleichen Zeilennummern einsetzen, die auch das Hauptprogramm verwendet (in den meisten Fällen sogar die gleichen Variablennamen). Einfache Parameter werden durch Wertaufruf übergeben, wobei Werte, die von der Subroutine geändert wurden, auch die entsprechenden Variablen des aufrufenden Programms beeinflussen. Globale Variablen, auf die sich Subprogramme und Hauptprogramm beziehen, müssen in beiden Programmodulen mit COMMON deklariert werden. Es lassen sich eine ganze Reihe von COMMON-Blöcken anlegen und mit Namen versehen, doch meistens genügt ein einziger Block dafür.

Die Variablen der COMMON-Anweisungen müssen einander in Typ und Gesamtlänge entsprechen. So könnte das Hauptprogramm beispielsweise den Befehl

```
COMMON I1,I2,I3
```

enthalten und die Subroutine

```
COMMON I(3)
```

In beiden Fällen wird der Speicherplatz auf die gleiche Weise eingesetzt, wobei die Parameter sich im ersten Fall auf Ganzzahlenvariablen beziehen und im zweiten Fall auf drei Arrayelemente des gleichen Typs.

Ein typischer Subroutinenaufruf sieht folgendermaßen aus:

```
CALL SUBNAME(X,Y,I,J)
```

wobei die Subroutine mit dem Befehl

```
SUBROUTINE SUBNAME(A,B,L,M)
```

anfängt. (Beachten Sie, daß die Parameter im Inneren der Klammern in Bezug auf Anzahl und Typ übereinstimmen.)

Die Anweisung RETURN gibt die Steuerung an das aufrufende Programm zurück (in einer Subroutine können sich mehrere RETURNs befinden). Eines der RETURNs sollte jedoch in jedem Fall ausgeführt werden – unabhängig von dem Weg, der durch die Subroutine führt. In Subroutinen wie im Hauptprogramm muß der letzte Befehl immer ein END sein.

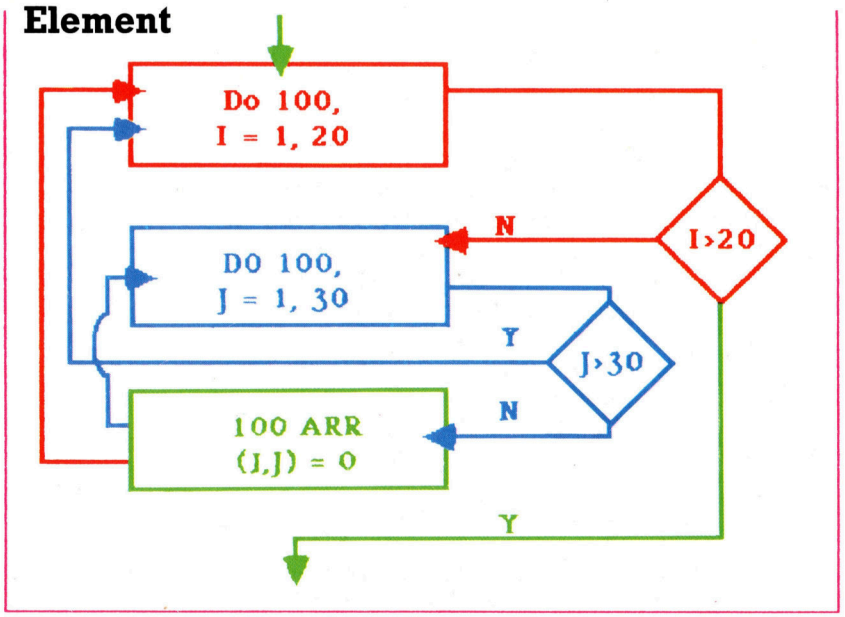
Bei der Parameterübergabe ist es in bestimmter Hinsicht möglich, die Größe eines Arrays dynamisch zu verändern. Obwohl das in einer interpretierten Sprache wie BASIC leicht auszuführen ist, gibt es diese Möglichkeit in compilierten Sprachen nur selten, da der Speicherplatz bei der Compilierung fest zugeordnet wird und nicht erst mit Geschwindigkeitsverlust beim Ablauf des Programms.

Da Subroutinen nur einen einzigen Wert zurückliefern, können sie auch (wie in PASCAL) als FUNCTION angelegt werden. Derartige Subprogramme werden, wie normale Subroutinen, separat geschrieben, beginnen aber mit der Anweisung FUNCTION. Vor dem RETURN sollte eine Zuweisung auf den Funktionsnamen liegen. Der Funktionsname selbst bestimmt den von der Funktion zurückgelieferten Datentyp, wobei die Vorgaben der vom System vorgegebenen Variablennamen angewandt werden. Diese Anforderung läßt sich zwar umgehen, sollte aber wenn möglich beibehalten werden. Das Hauptprogramm setzt die definierten Funktionen auf die gleiche Weise ein wie die normalen Systemfunktionen. Der Funktionsname (mit zugehöriger Parameterliste) kann dabei selbstverständlich in jedem Ausdruck erscheinen, in dem eine Variable dieses Typs auftreten darf.

Da FORTRAN keinen fest definierten Befehl für das Schleifenende besitzt, können mehrere Schleifen mit der gleichen Endanweisung abgeschlossen werden. Damit läßt sich sehr kompakter Programmcode schreiben (siehe Bild). In BASIC ist der gleiche Ablauf zwar weniger komprimiert, aber dafür auch weit leichter zu lesen.

```
100 FOR I=1 TO 20
110 FOR J=1 TO 30
120 LET A(I,J)=0
130 NEXT J
140 NEXT I
```

Das letzte Element



Sieg oder Platz?

Die meisten Laien am Rennplatz schließen ihre Wetten auf gut Glück ab. Professionelle Spieler und Buchmacher nutzen dagegen Methoden der Statistik und der Wahrscheinlichkeitsrechnung.

Die großen Buchmacher-Ketten in England nutzen die neuen Technologien zur Darstellung vielfältiger Informationen über das Renngeschehen. Der Computer dient aber auch dazu, die Gewohnheiten und Vorlieben des Wett-Publikums zu ermitteln. Sogar die Ermittlung der Gewinnquoten und die Erfassung der Mitarbeiterleistung erfolgt bereits rechnergesteuert.

Ein Buchmacher ist der Mittelsmann zwischen einzelnen Wettern, bei dem sich die Gewinnvorhersage der Wetter in barer Münze niederschlagen. An der Quote eines Pferdes kann man sehen, wieviel Geld darauf gesetzt wurde. Wenn viele Leute ihren Einsatz auf denselben Renner wetten, ist die Quote niedrig. Natürlich werden die Quoten so festgesetzt, daß für den Buchmacher stets ein Gewinn übrigbleibt, unabhängig vom Rennergebnis. Mit diesem Gewinn wird die Dienstleistung des Buchmachers von den Wettern gerechterweise bezahlt.

In den letzten zehn Minuten vor einem Pferderennen geht es am Wettmarkt besonders hoch her. Und das nicht nur auf der Rennbahn selbst; in den lokalen Büros der Buchmacher kann ein Spieler seine Wette auch aus der Ferne plazieren.

Am Rennplatz selbst gibt es Schwierigkeiten bei der Einführung neuer Technologien. Anders ist das bei den lokalen Buchmachern im traditionellen Land des Pferderennens. Speziell die englischen Buchmacher-Ketten erhoffen vom Computereinsatz Effektivität und mehr Profit. Ladbrokes, eine der größten Buchmacherfirmen Englands, nutzt den Computer speziell für das Wetten auf Kredit. Die Wetten aller Kreditkunden werden über dezentrale Cromemco-Microcomputer erfaßt, die mit einem zentralen Harddisk-Speicher verbunden sind. Zum Tagesabschluß werden die Daten auf einen IBM-Großrechner übertragen, der auch die Rennergebnisse des Tages kennt. Die Gewinnermittlung für Sieg-, Platz- oder Kombinationswetten erfolgt dann vollautomatisch. Einmal in der Woche werden die Kundenkonten auf den neuesten Stand gebracht und Schecks bzw. Rechnungen erstellt.

Wetten hat Stil

Große Buchmacherfirmen sind längst aus den muffigen Hinterzimmern in großzügige Büroetagen umgezogen. Das Filmklischee stimmt schon lange nicht mehr. Dazu paßt auch der verstärkte Computereinsatz für die unterschiedlichsten Zwecke. Bei der Firma Mecca Bookmakers ist man überzeugt, daß die sinkenden Computerkosten das Geschäft um mannigfaltige Möglichkeiten bereichern werden. Eingabeterminals an den wichtigsten Wettplätzen könnten etwa Informationen über die Vorlieben der Wetter oder die beliebtesten Wett-Kombinationen und die Wünsche der Spieler erfassen, aber auch für die interne Leistungsstatistik nutzbar gemacht werden.

Vor dem Hintergrund einer gesetzlichen Liberalisierung des Buchmachergewerbes in England erhofft man sich bei Mecca durch

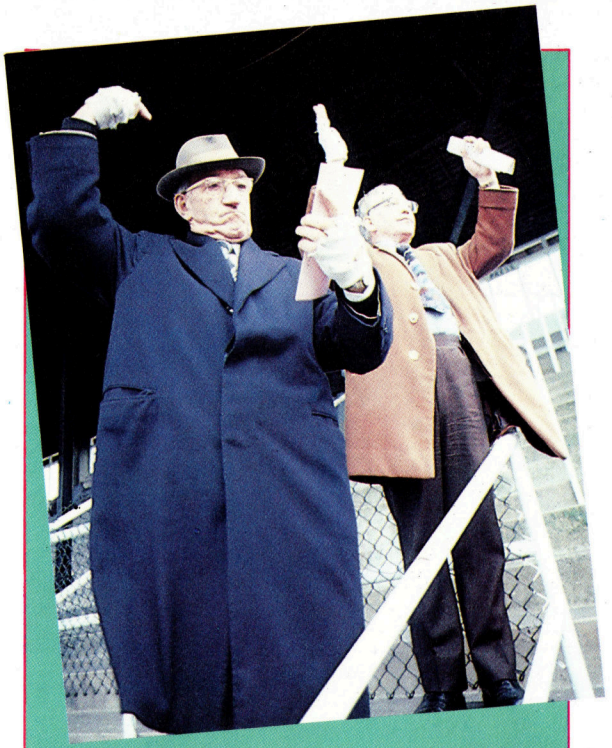
computergestützte Marktforschung eine erhebliche Ausdehnung des Geschäfts und damit einen weiteren Aufstieg im expandierenden Bereich der Freizeitindustrie. Schon heute wird der Großrechner der Firma für die grafische Darstellung von Informationen über Renn- und Sportereignisse genutzt, die in Farbe zu mehreren lokalen Wettbüros übertragen werden. Aber auch die finanziellen Ergebnisse der einzelnen Büros und der gesamten Firma werden vom Rechner überwacht. Schnelle Erfolgskontrolle wird zum Kinderspiel, und Entscheidungen können schneller mit guter Begründung getroffen werden.

Die Formel für Erfolg

Bei Buchmachern, die nicht direkt auf der Rennbahn arbeiten, kann auf die Reihenfolge der zwei oder drei ersten Pferde beim Einlauf gewettet werden. Der Gewinn aus einer solchen Wette wird mit einer komplexen Formel berechnet. Früher wurden zur Abschätzung der Gewinnchancen vom Buchmacher einfach nur die Einzelquoten der Pferde miteinander multipliziert. Unwägbarkeiten entstanden dabei durch die noch in letzter Minute schwankenden Quoten – oft ging die Rechnung nicht auf, die Buchmacher mußten Verluste hinnehmen. Speziell beim Sieg durch Favoriten gab es Probleme.

Man brauchte also eine Formel, mit der die Gewinnquoten so abzuschätzen wären, daß für die Buchmacher ein Mindestgewinn von 20 Prozent übrigblieb. 1977 war man in der Lage, diese Formel zu programmieren. CSF, die Wette auf die Reihenfolge der beiden ersten Pferde, wurde eingeführt, und 1981 folgte „Tricast" (für drei Pferde). Die mit der Formel ermittelten Ergebnisse werden von den beiden wichtigsten Buchmacher-Vereinigungen überwacht. Das in BASIC geschriebene Originalprogramm wird ständig an aktuelle Veränderungen der Wett-Gewohnheiten und andere Randbedingungen angepaßt.

Die spektakulärste Änderung der Formel wurde 1982 durch den „Little Owl"-Skandal initiiert: Bei einem Hindernisrennen gab es den Favoriten Little Owl, der eine Quote von 11:4 hatte. Ein weiterer Favorit wurde mit 5:2 gehandelt, ein krasser Außenseiter hatte die Quote 66:1. Aus irgendwelchen Gründen wurde der Favorit zu Anfang des Rennens gebremst, so daß das Pferd mit der Quote 5:2 und auch der Außenseiter vor dem Favoriten das Ziel erreichten. Einige Buchmacher gingen nach diesem ungewöhnlichen Rennausgang in Konkurs. Manche vermuteten professionell vorbereiteten Betrug. Um das Risiko von Überraschungsergebnissen bei einer kleinen Zahl startender Pferde zu verringern, wurde daraufhin der „Harmonische Faktor" in die Gewinnformel eingeführt. Dieser Faktor senkt die Quoten krasser Außenseiter, die Quote der Fa-



Zeichensprache

Unter den Buchmachern auf dem Rennplatz hat sich über Jahre hinweg ein komplexes System des Zusammenwirkens herausgebildet, das sie vor finanziellen Verlusten schützt. In England, dem klassischen Land des Pferdesports, gibt es zwei deutlich unterscheidbare Gruppen von Buchmachern: Nah bei den Pferden findet man die „großen", die Wetten nur von ihnen gut bekannten Klienten entgegennehmen und auf Barzahlung des Einsatzes verzichten können. Die zweite Gruppe steht mitten im Publikum und schließt Wetten über kleinere Beträge ab. Dabei geht das Geld bar von Hand zu Hand.

In den letzten Minuten vor dem Rennen wird es hektisch – der Markt explodiert, in kürzester Zeit fließen Zehntausende von Pfund Sterling hin und her. Innerhalb dieser Zeit verständigen sich die Buchmacher mit ihren Helfern durch eine spezielle Zeichensprache.

Eine wichtige Funktion dieser Helfer ist auch das Absichern von Wetten. Wenn ein Buchmacher merkt, daß er sich beim Abschließen von Wetten übernommen hat und bei einem bestimmten Pferd große Verluste machen kann, schließt er auf eben dieses Pferd bei einem anderen Buchmacher eine Wette ab. Gewinnt das Pferd später tatsächlich, kann er einen Teil seiner Verluste auf den Buchmacherkollegen abwälzen.

Im hektischen Geschehen des Rennplatzes wird der Computer wohl kaum verwendbar sein, es geschieht einfach alles viel zu schnell. Natürlich ließe sich die Tätigkeit eines Buchmachers durch ein Programm simulieren. Nur bei der Dateneingabe könnte der Computer seinem menschlichen Vorbild nicht das Wasser reichen: Beim Tempo-Wettbewerb liegen die Buchmacher immer noch um Längen vor dem Computer.

voriten wird dagegen erhöht. Mit der alten Formel hätte es bei der richtigen Vorhersage des oben beschriebenen Rennens eine Gewinnausschüttung von 141:1 gegeben, mit der neuen Formel beträgt sie nur noch 14:1.

Computer bieten lokalen Buchmachern umfangreiche Möglichkeiten zur Analyse auf Schwachpunkte im Geschäft – richtige Anwendung dieses Werkzeuges steigert also den Profit, indem unnötig hohe Risiken gemieden werden können.

Sicher ist, daß es für den Wetter kein Computerprogramm geben kann, mit dem er über Nacht reich wird oder den Buchmacher über die Löffel barbieren kann. Eine gewisse Unterstützung kann der Rechner aber auch dem Wetter bieten.

Eigentlich ist das Verlieren des Spielers auf lange Sicht unvermeidlich – dafür sorgt neben dem Buchmacher-Gewinn auch die Steuer. Trotzdem gibt es Profis, die vom Wetten leben. Durch genaue Kenntnis ihres Metiers streichen sie als Gewinn ein, was weniger kundige Wett-Freunde einsetzen und verlieren. Das ist jedoch kein müheloses Geschäft. Die sorgfältige und ausdauernde Analyse der Rennergebnisse ist Voraussetzung des Erfolgs.

Chancen bieten sich, wenn das Feld nur mit zwei oder drei wirklich erfolgversprechenden Pferden besetzt ist. Ein Pferd mit guten Gewinnaussichten ist schwierig zu finden, weil ausgesprochen viele, unter Umständen widersprüchliche Faktoren wie die Kondition des Pferdes, der Rennplatz, der Jockey und natürlich auch die Wetten anderer Spieler berücksichtigt werden müssen. Und oft bleiben auch anderen die Qualitäten eines Pferdes nicht verborgen, was zu sinkenden Quoten führt.

Mit Pferdeverstand

Erfolgreiche Wetter suchen Pferde mit guten Chancen, auf die aber wenig gesetzt wird. Solche Gelegenheiten sind schwer zu erkennen und nicht gerade häufig. Profis müssen also strenge Auswahlkriterien anlegen.

Bei einem Minimum von sechs Läufen während eines Renntages ist einfach nicht genug Zeit, ein festes Berechnungsschema für diesen Zweck anzuwenden. In einem Buchmacherbüro können Spieler auf Läufe an bis zu sechs verschiedenen Orten gleichzeitig wetten. Viel mehr als eine grobe Abschätzung ist bei dieser Flut von Möglichkeiten und Kombinationen selten durchführbar. Ganz ohne Methode und nur mit dem Vertrauen auf das große Glück ist Verlust allerdings ziemlich sicher vorprogrammiert – wie bei jedem Geschäft.

Mit einem Wett-Programm kann das anders werden. Das Programm zwingt den Spieler zu einer gewissen Auswahl und verhindert dadurch unkontrollierte Einsätze, die so oft Schaden anrichten. Was vielfach als Mangel angesehen wird – die relativ lange für die Dateneingabe benötigte Zeit, die für ein einzelnes Rennen bis zu einer halben Stunde betragen kann –, dient hier als wirkungsvoller Schutz vor Unbedachtsamkeit. Das Wetter muß entscheiden, welches Rennen die besten Möglichkeiten bietet. Sehr günstig ist es, nur in Rennen mit zehn oder weniger Pferden zu setzen. Dadurch wird die Dauer der Dateneingabe reduziert, und die Übersichtlichkeit wie die Gewinnchancen steigen.

Trotz allen technischen Fortschritts ist der Computer-Pferdewetter für den Buchmacher keine Bedrohung. Wettprogramme werden sowieso nur von eher bedächtigen und vorsichtigen Kunden eingesetzt. Eine wirklich durchgreifende Veränderung in den Wettgewohnheiten des breiten Publikums würde den Buchmachern nicht lange verborgen bleiben und sie zu geeigneten Gegenmaßnahmen veranlassen. Daß der Buchmacher seine Prozente am Wett-Kuchen nicht einbüßt – zumindest darauf kann man sicher wetten.

Solche Tabellen werden regelmäßig in speziellen Sportzeitschriften veröffentlicht, die zur Pflichtlektüre jedes Wettbegeisterten gehören. In diesem Beispiel werden die Erfolge eines bestimmten Gestüts über ein Jahr hinweg analysiert.

Deutsche Statistik 1985

Besitzer – Hindernisrennen

Name	Starter	Starts	Sieger	Siege	Gewinnsumme	Erfolgreichstes Pferd	mit DM
Stall Steintor	28	133	12	16	183 350	Constantin	28 500
W.Kaul	1	6	1	4	142 500	Ottilie	142 500
Gestüt Dachsborn	9	69	4	7	127 413	Parallele	41 770
Gestüt Sybille	5	17	2	5	86 100	Black Bottom	47 500
L.Swärd/Schweden	1	3	1	3	70 000	Ormus	70 000
SPFZ Baden-Baden	6	23	3	4	56 900	Park Rainbow	27 000
Stall Annelie	1	5	1	2	56 500	Sorbonne	56 500
Stall Windhof	6	26	3	7	56 450	Ditano	22 100
Stall Margarethe	1	4	1	3	55 500	Stragon	55 500
Gestüt Pfauenhof	4	19	1	3	48 700	Juliano	44 300
Gestüt Idee	1	5	1	2	39 800	Hakimi	39 800
Stall Steigenberger	1	5	1	4	38 600	Toscalon	38 600
E.Stoltefuß	1	14	1	4	37 500	Ledru-Rollin	37 500
Stall Barbarossa	2	15	1	2	36 600	Manet	21 500
Dr.A.Szemes	4	14	3	5	36 050	Feuerschuh	20 700
K.A.Staudt	1	12	1	4	35 600	Sonnenbalz	35 600
Gestüt Falkenstein	1	4	1	4	28 600	Pikante	28 600
H.Bölter	3	12	2	2	27 000	Santella King	13 600
Frau M.Bollow	2	13	1	2	25 750	Tanjanka	21 900
Frau L.Neumayer-Kieffer	2	7	1	2	25 100	Winterkönig	25 100
Stall Berbuk	3	14	2	3	24 450	Narew	15 300
Gestüt Birkenhof	2	9	1	2	24 000	Golden Berry	23 000
R.Klostermann	2	23	2	3	23 000	Colonius	12 300
Gestüt Harzburg	6	20	2	3	22 250	Ahnfrau	14 400
Frau K.Degott	2	20	2	3	22 250	Quattro Cento	14 250
M.Renner	7	26	2	4	21 400	Dolly Dollar	9 600
Stall Bacchant	2	9	1	1	20 900	April love	12 500
Frau H.Rotering	2	7	1	1	20 900	Pontal	20 900
Stall Caligula	1	12	1	1	20 600	Around	20 600
K.Witte	3	9	1	2	20 300	Virginia Lake	12 500
B.Mock	9	40	2	2	19 550	Oranje Manege	8 200
K.Lau	1	7	1	3	19 200	Leones (H)	19 200
Stall Marienhof	5	25	1	1	19 200	Massliebchen	6 900
A.Heyer u.F.Steinebach	1	7	1	3	18 100	Fernandel	18 100
Stall Moulin Rouge	1	9	1	3	17 800	Wunsch	17 800
J.Lißmann	1	6	1	2	17 600	Amicus	17 600
Gestüt Rodau	1	4	1	1	17 500	Rican	17 500
G.Spina	2	8	1	2	17 100	Ensor	17 100
Stall Marwin	2	14	1	2	16 700	New Commander	16 300
G.Deuerling u.F.Rühl	1	3	1	1	16 400	Wittelsbacher	16 400
Stall AGAU	2	6	-	-	16 200	Erbprinz	8 500
H.Hasler	1	9	1	2	15 300	Glückspfennig	15 300
R.Jösch	1	3	1	1	14 850	Assam	14 850

Große Sorgen

Bisher haben wir gesehen, wie auch kleinere Ereignisse die Handelsreise in die Neue Welt beeinflussen können. Jetzt gilt es, Schiffbrüchige aus einem Rettungsboot an Bord zu holen und mit der Pest fertigzuwerden.

Es gibt insgesamt sechs schwerwiegendere Ereignisse, von denen jedes einmal pro Reise eintreten kann. Eines dieser Ereignisse wird am Anfang jeder Woche per Zufall ausgewählt. Ist das gewählte Ereignis im Verlauf der Reise bereits eingetreten, wird für diese Woche keine weitere Wahl getroffen.

In Zeile 49 wird ein Array M() DIMensioniert, in dem vermerkt wird, welches der Ereignisse bereits eingetreten ist. Danach verzweigt das Programm durch GOSUB in Zeile 870 zur Unterroutine bei Zeile 6500, in der ein Ereignis bestimmt wird. In Zeile 6508 wird eine Zufallszahl zwischen 1 und 10 ermittelt. Da es nur sechs Zwischenfälle gibt, besteht somit eine 60prozentige Chance für ein unvorhergesehenes Ereignis: Bei den Zahlen 7, 8, 9 und 10 geschieht nichts. Sie können die Wahrscheinlichkeit variieren, indem Sie den Zufallsfaktor in dieser Zeile ändern.

Ändern Sie beispielsweise Zeile 6508 in

```
X=INT(RND(1)*8)+1
```

wird die Wahrscheinlichkeit auf 75 Prozent erhöht. Beachten Sie jedoch, daß sich, nachdem ein Ereignis gewählt und dies in M() verzeichnet wurde, die Wahrscheinlichkeit in der nächsten Woche auf 62,5 Prozent reduziert. Beim dritten Ereignis sind es sogar nur noch 50 Prozent. Die Wahrscheinlichkeit sinkt also, je mehr Zwischenfälle bereits eingetreten sind.

Die dem Ereignis entsprechende Unterroutine wird mittels ON X GOSUB gewählt, genau wie bei den geringfügigeren Ereignissen. Ist X (die Zufallszahl) gleich 1, wird das Programm zur ersten Zeilennummer der hinter der Anweisung befindlichen Liste verzweigt (Zeile 6530=das Sichten eines Rettungsbootes). Ist X=2, wird zur zweiten Nummer verzweigt (Zeile 6700=die Pest). In diesem Artikel befassen wir uns nur mit den ersten beiden größeren Ereignissen, doch müssen wir die Zeilennummern der restlichen vier Unterroutinen später noch in Zeile 6510 einfügen.

Durch die erste Unterroutine wird mit dem Fernrohr ein Rettungsboot gesichtet. Im Boot befinden sich vier Personen und eine große Kiste. Der Spieler muß nun entscheiden, ob er sie an Bord nehmen will oder nicht. Wenn ja, verlängert sich die Reisedauer um zwei Tage und es müssen vier Männer zusätzlich ernährt werden. Andererseits brauchen Sie vielleicht die Stärke der vier zusätzlichen Männer. Zuerst überprüft das Programm über das erste Element von M() in Zeile 6535, ob das Boot zuvor bereits gesichtet wurde. Ist es auf 1 gesetzt, ist das Ereignis schon einmal eingetreten, und es erfolgt Rücksprung zum Hauptprogramm. Ist der Wert 0, setzt Zeile 6540 M(1) auf 1. Wie bereits erwähnt, wird dieser Test vor jedem größeren Ereignis durchgeführt.

In Zeile 6576 wird der Spieler gefragt, ob er die Schiffbrüchigen aufnehmen will. Zeile 6580 überprüft, ob der erste Buchstabe der Antwort Y oder N ist. Ist die Antwort Nein, nimmt ein anderes Schiff die Schiffbrüchigen an Bord, und es erfolgt Rücksprung zum Hauptprogramm. Entscheidet der Spieler, die Schiffbrüchigen an Bord zu nehmen, überprüft das Programm in Zeile 6625 zuerst, ob die Mannschaft nicht schon komplett ist. Ist dies der Fall (CN=16), ist kein Platz mehr, und die Routine verzweigt zum Hauptprogramm. Um festzustellen, ob für alle Schiffbrüchigen Platz ist, wird in Zeile 6630 der Wert von X auf 16 minus CN gesetzt. Danach wird entschieden, ob alle vier Personen aufgenommen werden können (X>3). Trifft dies zu, werden alle an Bord genommen. Ist X kleiner als 4, wird dem Spieler mitgeteilt, daß nur X Personen aufgenommen werden können.

Überlebende eingegliedert

Die Überlebenden aus dem Rettungsboot werden dann mittels der Schleife von Zeile 6638 bis 6697 in das Stärke-/Typ-Array eingegliedert. Der Wert von X wird auf 0 gesetzt, um die zusätzlichen Mannschaftsmitglieder zu kontrollieren. Mit X wird TS(,) überprüft, ob ein Eintrag 0 ist und somit dort ein neues Mitglied eingetragen werden kann. Wird ein Platz gefunden, wird X um 1 erhöht. Erreicht X den Wert 4, sind alle Überlebenden eingegliedert, und in Zeile 6655 wird T auf 16 gesetzt, so daß beim nächsten NEXT die Schleife verlassen wird. Reicht der Platz nur für weniger als vier Personen, verläßt das Programm die Schleife, wenn sie 16mal durchlaufen wurde. Die zusätzlichen Mitglieder werden auch zum Wert von CN addiert. Jedem neuen Mitglied wird ein Zu-

fallswert als Mannschaftstyp und Stärke in TS(,) zugeordnet. Der Bereich für den Typ liegt zwischen 1 und 5, für die Stärke zwischen 50 und 99 (gesund oder sehr gesund).

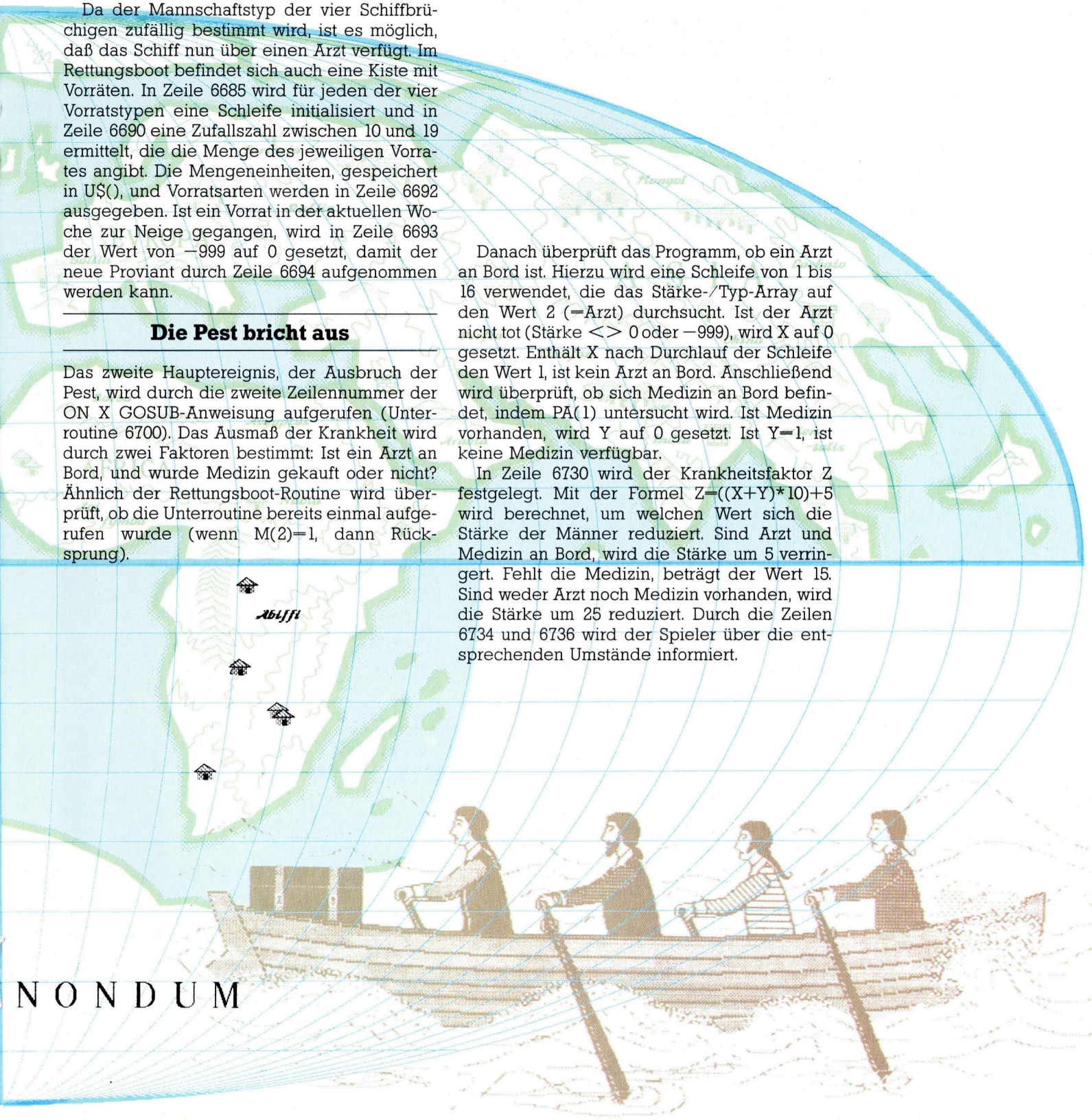
Da der Mannschaftstyp der vier Schiffbrüchigen zufällig bestimmt wird, ist es möglich, daß das Schiff nun über einen Arzt verfügt. Im Rettungsboot befindet sich auch eine Kiste mit Vorräten. In Zeile 6685 wird für jeden der vier Vorratstypen eine Schleife initialisiert und in Zeile 6690 eine Zufallszahl zwischen 10 und 19 ermittelt, die die Menge des jeweiligen Vorrates angibt. Die Mengeneinheiten, gespeichert in U$(), und Vorratsarten werden in Zeile 6692 ausgegeben. Ist ein Vorrat in der aktuellen Woche zur Neige gegangen, wird in Zeile 6693 der Wert von −999 auf 0 gesetzt, damit der neue Proviant durch Zeile 6694 aufgenommen werden kann.

Die Pest bricht aus

Das zweite Hauptereignis, der Ausbruch der Pest, wird durch die zweite Zeilennummer der ON X GOSUB-Anweisung aufgerufen (Unterroutine 6700). Das Ausmaß der Krankheit wird durch zwei Faktoren bestimmt: Ist ein Arzt an Bord, und wurde Medizin gekauft oder nicht? Ähnlich der Rettungsboot-Routine wird überprüft, ob die Unterroutine bereits einmal aufgerufen wurde (wenn M(2)=1, dann Rücksprung).

Danach überprüft das Programm, ob ein Arzt an Bord ist. Hierzu wird eine Schleife von 1 bis 16 verwendet, die das Stärke-/Typ-Array auf den Wert 2 (=Arzt) durchsucht. Ist der Arzt nicht tot (Stärke <> 0 oder −999), wird X auf 0 gesetzt. Enthält X nach Durchlauf der Schleife den Wert 1, ist kein Arzt an Bord. Anschließend wird überprüft, ob sich Medizin an Bord befindet, indem PA(1) untersucht wird. Ist Medizin vorhanden, wird Y auf 0 gesetzt. Ist Y=1, ist keine Medizin verfügbar.

In Zeile 6730 wird der Krankheitsfaktor Z festgelegt. Mit der Formel Z=((X+Y)*10)+5 wird berechnet, um welchen Wert sich die Stärke der Männer reduziert. Sind Arzt und Medizin an Bord, wird die Stärke um 5 verringert. Fehlt die Medizin, beträgt der Wert 15. Sind weder Arzt noch Medizin vorhanden, wird die Stärke um 25 reduziert. Durch die Zeilen 6734 und 6736 wird der Spieler über die entsprechenden Umstände informiert.



Sind einige Männer bereits sehr geschwächt, kann sie die Krankheit töten. X wird auf 0 gesetzt, um die Verstorbenen zu zählen. Eine Schleife von Zeile 6755 bis 6775 durchläuft das Stärke-/Typ-Array und reduziert die Stärkewerte der Mannschaft. Doch die Pest befällt nicht alle Männer. Zeile 6756 generiert eine Zufallszahl. Ist sie kleiner als 0.3, wird der entsprechende Mann nicht angesteckt. Damit hat jeder eine 30prozentige Chance, nicht zu erkranken. Die nächste Zeile überprüft, ob eine Person bereits tot ist und darum nicht mehr erkranken kann. Lebt der Mann, wird die Stärke um den Wert von Z reduziert, und es wird überprüft, ob er nach dem Abzug gestorben ist (Stärkewert kleiner als 0). In diesem Fall wird der Wert auf 999 gesetzt und der Wert von X um 1 erhöht. War Medizin an Bord (Y=0), wird der Vorrat halbiert und auf die nächste volle Flasche gerundet. Ist durch die Epidemie niemand gestorben (X=0), erfolgt Rücksprung zum Hauptprogramm. Ansonsten wird der Spieler über die Verluste informiert.

Im nächsten Artikel befassen wir uns mit weiteren Ereignissen auf der Reise, und es gibt ein Problem mit der Ruderanlage.

Modul acht: Zwei Hauptereignisse:

Dimensionierung des Flag-Arrays

```
40 DIMM(6):REM INDS TO SHOW IF MAJOR EVENTS HAVE B
EEN DONE
```

Zusatz zum Hauptprogramm

```
870 GOSUB6500:REM GOTO MAJOR CONTINGENCY
```

Auswahlroutine der Hauptereignisse

```
6500 REM MAJOR CONTINGENCIES
6508 X=INT(RND(1)*10)+1
6510 ON X GOSUB6530,6700
6520 RETURN
```

Routine der Hauptereignisse

```
6530 REM LIFEBOAT
6535 IFM(1)=1THENRETURN
6536 PRINTCHR$(147)
6540 M(1)=1
6550 S$="A LIFEBOAT IS SIGHTED*":GOSUB9100
6552 S$="DRIFTING IN THE DISTANCE*":GOSUB9100
6554 PRINT:GOSUB9200
6556 S$="THROUGH THE TELESCOPE YOU SEE*":GOSUB9100
6558 S$="THAT IT CONTAINS:*":GOSUB9100
6560 PRINT:GOSUB9200
6562 S$="4 PEOPLE*":GOSUB9100
6563 GOSUB9200
6564 S$="AND A LARGE CHEST!*":GOSUB9100
6566 PRINT:GOSUB9200
6568 S$="IF YOU CHANGE COURSE*":GOSUB9100
6570 S$="TO PICK THEM UP*":GOSUB9100
6572 S$="IT WILL TAKE YOU TWO EXTRA DAYS*":GOSUB91
00
6574 PRINT:GOSUB9200
6576 S$="DO YOU WANT TO PICK THEM UP (Y/N)*":GOSUB
9100
6578 INPUTI$:I$=LEFT$(I$,1)
6580 IFI$<>"Y"ANDI$<>"N"THEN6578
6585 IFI$="Y"THEN6600
6588 PRINT:GOSUB9200
6590 S$="THE LIFEBOAT DISAPPEARS....*":GOSUB9100
6592 PRINT:GOSUB9200
6594 S$=K$:GOSUB9100
6596 GETI$:IFI$=""THEN6596
6599 RETURN
6600 PRINT:GOSUB9200
6610 EW=EW+2/7
6625 IFCN<>16 THEN6630
6627 S$="YOU CAN'T PICK THEM UP*":GOSUB9100
6628 S$="YOU HAVE NO ROOM ON THE SHIP*":GOSUB9100
6629 GOTO6592
6630 X=16-CN:IFX>3THEN6635
6632 S$="YOU ONLY HAVE ROOM FOR*":GOSUB9100
6633 PRINTX:"MORE PEOPLE ON THE SHIP"
6634 PRINT:GOSUB9200
6635 S$="YOU PICK UP:*":GOSUB9100
6638 X=0
6640 FORT=1TO16
6645 IFTS(T,1)<>0THEN6679
6650 X=X+1
6655 IFX>4THENT=16:GOTO6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ";C$(TS(T,1))
6679 NEXT
6680 PRINT:GOSUB9200
6682 S$="THE CHEST CONTAINS:*":GOSUB9100
6685 FORT=1TO4
6690 X=INT(RND(1)*10)+10
6692 PRINTX;U$(T);"S OF ";P$(T)
6693 IFPA(T)=-999THENPA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO6592
6700 REM PLAGUE STRIKES
6705 IFM(2)=1THENRETURN
6706 PRINTCHR$(147)
6710 M(2)=1
6712 S$="A PLAGUE STRIKES*":GOSUB9100
6714 PRINT:GOSUB9200
6716 X=1
6718 FORT=1TO16
6720 IFTS(T,1)=2ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=0:T=16
6722 NEXT
6724 Y=1
6726 IFOA(1)<>0ANDOA(1)<>-999THENY=0
6730 Z=((X+Y)*10)+5
6732 I$="YOU HAVE ":IFX=0ANDY=0THEN6740
6734 IFX=1THENS$="WITH NO DOCTOR*":GOSUB9100:I$="A
ND "
6736 IFY=1THEN S$=I$+"NO MEDICINE*":GOSUB9100
6740 S$="MANY OF THE CREW ARE AFFECTED*":GOSUB9100
6745 PRINT:GOSUB9200
6750 X=0
6755 FORT=1TO16
6756 IFRND(1)<.3THEN6775
6760 IFTS(T,2)=0ORTS(T,2)=-999THEN6775
6765 TS(T,2)=TS(T,2)-Z
6770 IFTS(T,2)<1THENTS(T,2)=-999:X=X+1
6775 NEXT
6776 IFY=1THEN6780
6777 S$="1/2 YOUR MEDICINE IS USED*":GOSUB9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT:GOSUB9200
6785 IFX=0THEN6797
6790 PRINT"AND";X;
6792 S$="CREW MEMBERS DIE*"
6794 IFX=1THENS$="CREW MEMBER DIES*"
6795 GOSUB9100
6796 PRINT:GOSUB9200
6797 S$=K$:GOSUB9100
6798 GETI$:IFI$=""THEN6798
6799 RETURN
```

BASIC-Dialekte

Spectrum:
Führen Sie die folgenden Änderungen aus:

```
6510 IF X = 1 THEN GO SUB 6530
6511 IF X = 2 THEN GO SUB 6700
6536 CLS
6578 INPUT I$ : LET I$ = I$ (1 TO 1)
6596 LET I$ = INKEY$ : IF I$ = "" THEN GO TO 6596
6706 CLS
6798 LET I$ = INKEY$ : IF I$ = "" THEN GO TO 6798
```

Acorn B:
Ändern Sie das Programm wie folgt:

```
6536 CLS
6596 I$ = GET$
6706 CLS
6798 I$ = GET$
```

Byte für Byte

Dieser Artikel beschließt unsere Untersuchung der ROM-Routinen des Dateisystems für den Acorn B. Wir untersuchen OSFIND und verwandte Routinen, mit denen wir Zugriff auf einzelne Bytes haben.

OSFIND wird über die Adresse &FFCE aufgerufen und eröffnet eine Datei, um auf einzelne Bytes zugreifen zu können. Dabei geschieht das gleiche wie bei der Eröffnung einer Datei mit OPENOUT, OPENIN oder OPENUP von BASIC aus. Weiterhin lassen sich damit Dateien auf die gleiche Weise schließen wie mit CLOSE# in BASIC. Das A-Register gibt an, welche Funktion ausgeführt wird, während X und Y auf den Dateinamen im Speicher zeigen.

●**A=0:** Zeigt an, daß die Datei geschlossen werden soll. Das Y-Register enthält dabei die „Nummer", die der Datei bei ihrer Eröffnung zugeteilt wurde. Die Angabe der Dateinummer 0 schließt alle geöffneten Dateien.

●**A=&40:** Entspricht dem BASIC-Befehl OPENIN. Eine Datei wird nur für die Eingabe von Daten(bytes) geöffnet.

●**A=&80:** Entspricht dem BASIC-Befehl OPENOUT. Eine Datei wird nur für die benötigte Ausgabe geöffnet.

●**A=&C0:** Entspricht dem BASIC-Befehl OPENUP. Eine Datei wird für Ein- und Ausgabe eröffnet. Wird für Dateisysteme eingesetzt, die Random-Access-Dateien unterstützen.

A wird auf Null gesetzt

Falls OSFIND eine Datei nicht eröffnen kann (beispielsweise bei einer schreibgeschützten Diskette), wird A bei der Rückkehr von OSFIND auf Null gesetzt. Ist die Eröffnung jedoch erfolgreich, erhält A eine Dateinummer, die für einen späteren Einsatz zur Verfügung steht.

Alle folgenden Aufrufe von ROM-Routinen des Dateisystems haben eine Eigenschaft gemeinsam: Sie verwenden die mit OSFIND erzeugte Dateinummer.

Die OSARGS-Routine wird über &FFDA aufgerufen und läßt sich für eine ganze Reihe von Aufgaben einsetzen. Sie hat eine Eigenheit: Ihr Parameterblock von nur vier Byte Länge ist auf der Zero Page untergebracht. Das X-Register enthält dabei die Adresse des ersten Parameterblockbytes, Y die Dateinummer und A die Nummer der Funktion, die ausgeführt werden soll. Ein spezieller Fall von OSARGS tritt ein, wenn das Y-Register auf Null gesetzt ist. In diesem Fall liefert die Routine Informationen über das Dateisystem statt über eine bestimmte Datei oder veranlaßt, daß alle eröffneten Dateien des Dateisystems mit den Daten aktualisiert werden, die sich noch in den Dateibuffern befinden. Hier zunächst diese beiden Routinen:

●**Y=0:A=0:** Wenn OSARGS mit diesen Parametern aufgerufen wird und das X-Register auf einen freien Bereich der Zero Page zeigt, enthält das Register A nach dem Rücksprung Informationen über das Dateisystem. Dies ist der einzige Aufruf von OSARGS, der sich für Cassettensysteme eignet. Unsere Tabelle am Rand zeigt, wie die Daten, die nach der Rückkehr in A stehen, interpretiert werden.

●**Y=0:A=255:** Bei dem zweiten Aufruf mit Y auf Null enthält A den Wert 255. Dieser Aufruf von OSARGS stellt sicher, daß alle Daten, die sich noch in den Buffern der Maschine befinden, in die entsprechenden Dateien geschrieben werden. Sind keine Dateien eröffnet, kann auch keine Aktualisierung stattfinden.

●**Y=Dateinummer:** Wenn Y die Dateinummer enthält, bestimmt der Inhalt des Registers A, welche Funktion ausgeführt wird (siehe untenstehende Tabelle). Der „sequentielle Zeiger" auf eine offene Datei informiert das Betriebssystem darüber, wohin das nächste Byte geschrieben oder von wo es abgerufen werden soll. Der Zeiger läßt sich daher von der BASIC-Funktion PTR# und auch von diesen Routinen ändern. Die Daten werden im Parameterblock auf der Zero Page gespeichert, wobei das LSB in der Adresse (X+0) untergebracht ist. Daten, die OSARGS aus dem Dateisystem liest, werden entsprechend in diesem Parameterblock zwischengespeichert.

Inhalt des A-Registers	Beschreibung
0	Die aktuelle Position des sequentiellen Pointers der im Y-Register angegebenen Datei lesen
1	Den Parameterblockwert in den sequentiellen Pointer schreiben. Ähnlich dem Befehl PTR#n=Wert
2	Dateilänge in den Parameterblock einsetzen
255	Datei (deren Nummer im Y-Register steht) auf dem Speichermedium aktualisieren

Aus der Beschreibung wird deutlich, warum der Parameterblock von OSARGS sich auf der Zero Page befinden muß: Für die Adresse steht nur das X-Register zur Verfügung.

Mit den Routinen OSBGET und OSBPUT wird

Inhalt des A-Registers	Beschreibung
0	Kein Dateisystem
1	Band mit 1200 Baud
2	Band mit 300 Baud
3	ROM
4	Diskette
5	Netzwerk
6	Telesoft

ein Byte aus einer offenen Datei gelesen oder dorthin geschrieben. Der Aufruf von OSBGET erfolgt über &FFD7. Dabei wird das Y-Register mit der von OSFIND angelegten Dateinummer geladen und über den Aufruf von &FFD7 ein Byte von der aktuellen Position des sequentiellen Pointer gelesen und in A gespeichert.

OSBPUT ist die Umkehrung dieses Vorgangs. Die Routine schreibt ein Byte an die Position der Datei, auf die der sequentielle Pointer zeigt. Das Byte, das geschrieben werden soll, wird dafür in A gespeichert und die Dateinummer in Y. Der Aufruf von &FFD4 führt die Routine aus. Das folgende Modul zeigt, wie OSBPUT ein einzelnes Byte speichert.

```
LDA   #&80 / Dateieröffnung vorbereiten
LDX   #name MOD 256
LDY   #name DIV 256
JSR   OSFIND /Datei öffnen
STA   &71 / Dateinummer speichern
TAY   / und im Y-Register ablegen
LDA   #65 / Das Byte, das geschrieben werden soll
JSR   OSBPUT / schreiben
LDA   #0
LDY   &71 / Dateischließung vorbereiten
JSR   OSFIND / Datei schließen
RTS   / Ende
```

Die OSGBPB-Routine wird über &FFD1 aufgerufen. Sie transferiert Bytegruppen zwischen dem Speicher und offenen Dateien und kann auch Informationen über das eingesetzte Dateisystem liefern. Wir werden hier jedoch nur auf ihre Funktionen bei der Datenübertragung eingehen. Die Routine läßt sich nicht mit Cassettensystemen einsetzen, da sie sich eigentlich nur für den Zugriff auf Random-Access-Dateien vom Maschinencode aus eignet.

Parameterblock von OSGBPB	
Adresse	**Beschreibung**
0	Dateinummer, wie von OSFIND geliefert
1	LSB der Adr. der Daten (READ/WRITE)
2	Nicht festgelegt
3	Nicht festgelegt
4	MSB der Datenadresse
5	LSB der Anzahl Bytes, für oder aus Datei
6	Nicht festgelegt
7	Nicht festgelegt
8	MSB der Anzahl der zu bewegenden Bytes
9	LSB der Position des sequentiellen Pointers (welche Daten gelesen werden sollen)
10	Nicht festgelegt
11	Nicht festgelegt
12	MSB der Position des seq Pointers

Die vorstehende Tabelle stellt den Parameterblock des Aufrufs dar, auf dessen Adresse die Register X und Y zeigen. A gibt an, welche Funktion eingesetzt werden soll.

Bei einem standardmäßigen Acorn B wird die Datenadresse vom Inhalt des Bytes 1 und 4 des Parameterblocks angegeben.

```
10REM disk system only
20DIM  block 20
30DIM data 100
40$data="This is a test program!"
50 block!1=data:REM address of data to be
 written
60block!5=100:REM amount of data to be written
70block!9=0:REM sequential pointer=0
80 PRINT "Opening the file for output"
90Y%=OPENOUT("FILE"):REM open the file
100block?0=Y%          :REM file number
110X%=block MOD 256
120Y%=block DIV 256
130A%=1: REM prepare to write data
140 CALL &FFD1 :REM  do it
150CLOSE £(block?0)
160 PRINT "Close the file"
170 TIME =0: REPEAT: UNTIL TIME=400
180 PRINT "Opening the file for input"
190PRINT "Now read the data in!"
200$data=STRING$(100," "):REM clear data area
210Y%=OPENIN("FILE"):REM open the file up
220block?0=Y%          :REM file number
230block!1=data :REM data address for read
operation
240block!5=100 : REM number of bytes
250block?9=0 : REM sequential pointer to 0
260X%=block MOD 256
270Y%=block DIV 256
280A%=3: REM prepare to read
290CALL &FFD1 :REM do it
300CLOSE£(block?0)
310 PRINT "Close the file again"
320PRINT $data
330END
```

Es stehen vier verschiedene Übertragungsarten zur Verfügung: zwei Lese- und zwei Schreibvorgänge. Je ein Lese- und Schreibvorgang spricht die Daten der Position an, die der sequentielle Pointer des Parameterblocks angibt. Die anderen beiden Routinen ignorieren diesen Eintrag und greifen statt dessen auf die Position zu, auf die der Pointer gerade zeigt. A=1 schreibt daher Bytes an den im Block angegebenen Pointer, während A=2 den Pointer ignoriert und Bytes an die aktuelle Pointerposition schreibt. A=3 liest Daten von der im Block angegebenen Position, während A=4 Bytes liest, aber die Pointeradresse des Blocks ignoriert. Das folgende Programm setzt OSGBPB ein, um eine einfache Datei zu schreiben.

Der Status des Übertragsflags zeigt nach dem Ablauf von &FFD1, ob die Übertragung erfolgreich war oder nicht. Wenn C Null enthält, verlief sie reibungslos, steht C auf Eins, fand keine Übertragung statt.

Die letzte Betriebssystemroutine für Dateivorgänge ist OSFSC. Sie hat für eine Programmierung jedoch nur wenig Nutzen. Ihr Aufruf unterscheidet sich von den zuvor erläuterten, da er nicht über eine direkte Adresse stattfindet, sondern über die Adresse seines Vektors. In einem Maschinencodeprogramm sieht das so aus:

```
JMP (&21E)
```

Weitere OS-Routinen führen Spezialfunktionen für bestimmte Dateisysteme aus.

In Bewegung

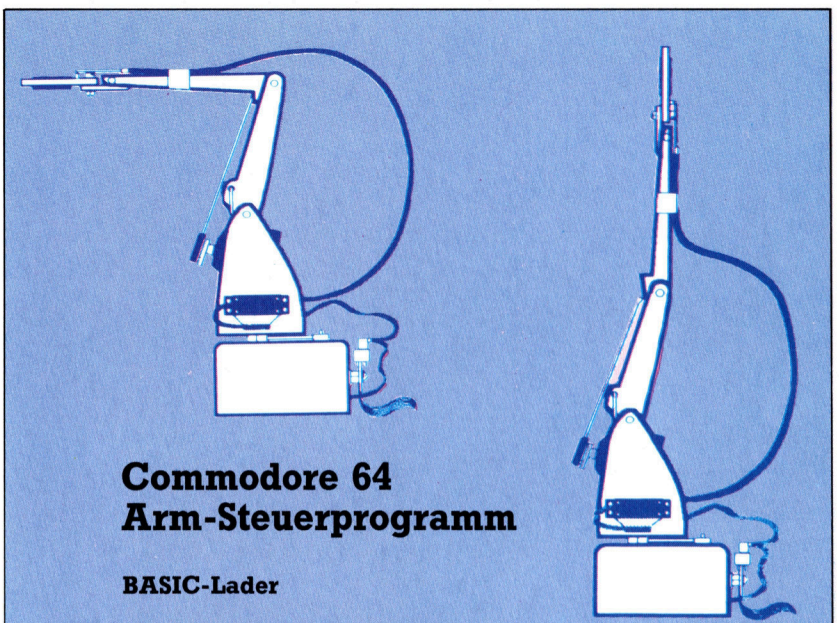
Abweichungen der BASIC-Dialekte und Verschiedenheiten im Betriebssystem von Acorn B und Commodore 64 machen den Entwurf maßgeschneiderter Steuerprogramme für den Robot-Arm erforderlich. Wir wollen nun auf den Commodore 64 eingehen.

Das Betriebssystem des Commodore 64 erzeugt regelmäßige Interrupts zur Steuerung interner Vorgänge wie etwa der Tastaturabfrage. Diese Interrupts erfolgen in festen Zeitabständen von einer sechzigstel Sekunde. Das Steuerprogramm für den Commodore wurde in diese Interrupt-Routine eingebaut. Der Rechner kehrt erst nach Ausführung beider Abläufe wieder zur normalen Funktion zurück. Der erste Teil des Maschinenprogramms dient dem Austausch des normalen IRQ-Vektors durch einen neuen Vektor für unser Programm.

Wenn der normale Vektor durch unseren neudefinierten ersetzt ist, wird bei jedem Interrupt die Steuerroutine für den Arm durchlaufen. Das Programm schickt Steuerimpulse über den User Port an die Motoren. Die Länge jedes Impulses teilt dem betreffenden Motor mit, auf welchen Winkel er sich einstellen soll. Die richtige Impulslänge ruft das Maschinenprogramm aus einer Acht-Bit-Tabelle mit dem Namen ANGLE ab. Wie beim Acorn B werden die Speicherplätze der ANGLE-Tabelle nach und nach den Werten einer weiteren Tabelle (NEWPOS) angeglichen. NEWPOS wird durch den BASIC-Programmteil definiert.

Das Arm-Steuerprogramm nutzt das eben beschriebene Maschinenprogramm zum Antrieb der Servomotoren. Ähnlich wie beim Acorn B dient auch hier die Tastatur zur Steuerung der Bewegungen. Das Speichern der Positionen funktioniert wie beim Acorn. Allerdings gibt es auch Unterschiede zwischen den beiden Versionen: Bei Bewegungen neigen die Motoren dazu, auf der angewählten Position hin- und herzupendeln. Das liegt am Video-Chip des Commodore, der die Interrupt-Steuerung beeinflußt. Am einfachsten läßt sich das Problem umgehen, indem der Bildschirm während der Laufzeit des Wedge-Programms gelöscht wird. Diese Funktion führen Zeile 7000 und 8000 aus.

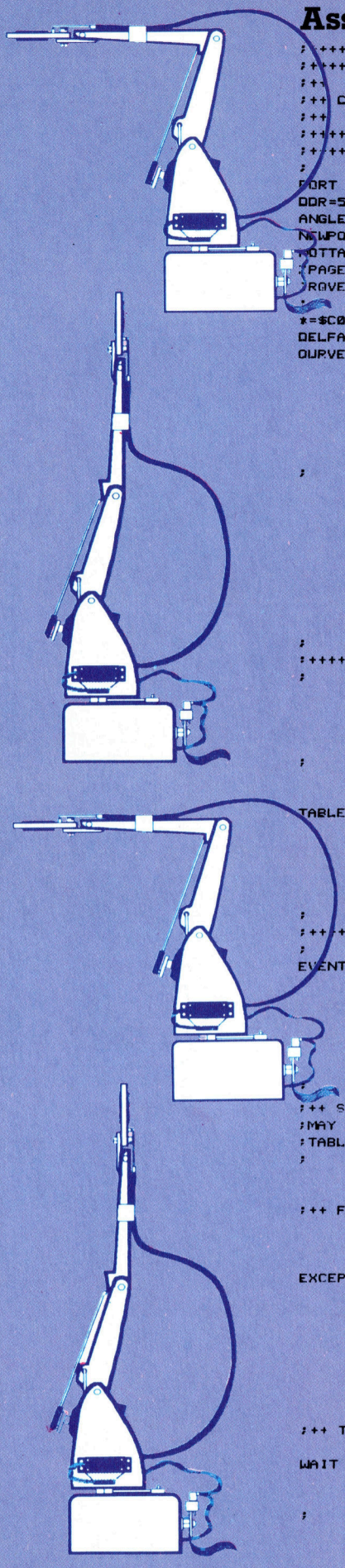
Die zweite Schwierigkeit ist die Langsamkeit des Commodore-BASIC bei der Tastaturabfrage. Das fertige Maschinenprogramm vergleicht den ASCII-Code eines Tastendrucks mit den durch eine Funktion belegten Tasten aus der Tabelle. Bei einer Übereinstimmung wird die entsprechende Zahl in Verbindung mit dem ON. . .GOSUB-Befehl zum Aufruf des entsprechenden Unterprogramms eingesetzt.



Commodore 64 Arm-Steuerprogramm

BASIC-Lader

```
10 REM ************************
20 REM **                      **
30 REM **  BASIC LOADER FOR    **
40 REM ** CBM ARM CONTROLLER   **
50 REM **                      **
60 REM ************************
65 :
70 FOR I=49155 TO 49394
75 READ A:POKE I,A:CC=CC+A
80 NEXT I
90 READ CS:IF CS<>CC THEN PRINT"CHECKSUM ERROR":STOP
95 :
100 DATA169,255,141,3,221,169,62,141,1
110 DATA192,169,192,141,2,192,120,173
120 DATA20,3,174,1,192,141,1,192,142
130 DATA20,3,173,21,3,174,2,192,141,2
140 DATA192,142,21,3,169,16,133,251
150 DATA169,193,133,252,169,255,160,0
160 DATA145,251,136,208,251,88,96,8,72
170 DATA152,72,138,72,169,255,141,1
180 DATA221,162,7,169,255,24,106,72
190 DATA188,0,193,49,251,145,251,104
200 DATA202,16,243,160,48,136,208,253
210 DATA169,255,160,0,49,251,141,1,221
220 DATA200,208,248,162,7,169,255,188
230 DATA0,193,145,251,202,16,248,104
240 DATA170,104,168,104,40,108,1,192
250 DATA162,0,160,0,189,8,193,221,0
260 DATA193,240,14,176,6,222,0,193,76
270 DATA156,192,254,0,193,76,156,192
280 DATA200,232,224,4,208,228,32,169
290 DATA192,192,4,208,217,96,138,72
300 DATA152,72,160,255,174,0,192,136
310 DATA234,234,208,251,202,208,248
320 DATA104,168,104,170,96,0,255,255,0
330 DATA0,255,255,0,0,255,255,0,0,255
340 DATA255,0,0,72,138,72,152,72,32
350 DATA228,255,201,0,240,249,162,0
360 DATA221,191,192,240,7,232,224,17
370 DATA208,246,162,255,142,207,192
380 DATA104,168,104,170,104,96
390 DATA33097:REM*CHECKSUM*
```



Assembler-Listing

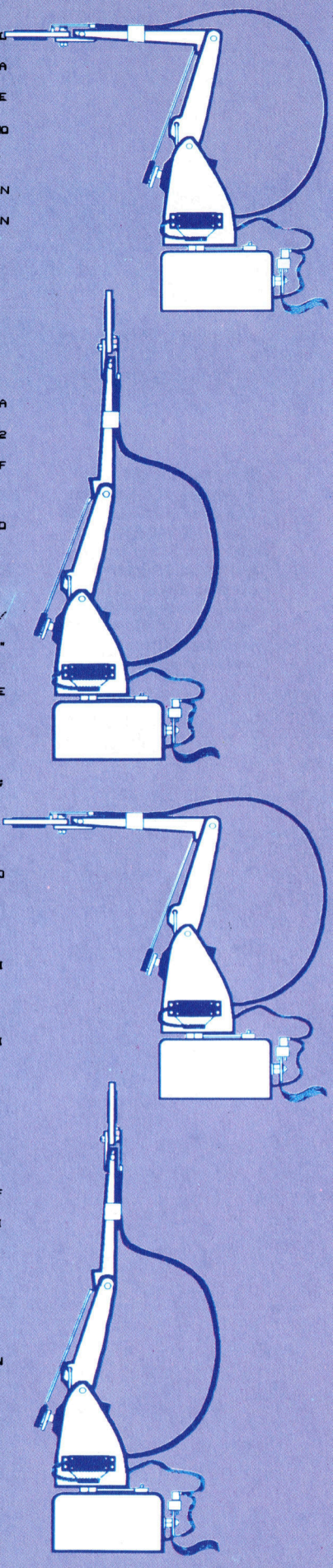
```
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;++                    ++
;++ CBM ARM CONTROLLER ++
;++                    ++
;++++++++++++++++++++++++
;++++++++++++++++++++++++
;
PORT = 56577 ;USER PORT DATA REGISTER
DDR=56579    ;USER PORT DATA DIRN REG
ANGLE=$C100  ;ANGLE VALUE LOCATION
NEWPOS=$C108 ;POSITION TABLE
MOTTAB=$C110 ;MOTOR LOOK UP TABLE
ZPAGE=$FB    ;0 PAGE POINTER TO TABLE
IRQVEC=$0314 ;IRQ VECTOR LOBYTE
;
*=$C000
DELFAC *=*+1 ;STORAGE FOR DELAY FACTOR
OURVEC *=*+2 ;STORAGE FOR OUR VECTOR
      LDA #$FF
      STA DDR      ;SET DDR TO OUTPUT
      LDA #<EVENT
      STA OURVEC   ;POINT TO EVENT
      LDA #>EVENT  ;HANDLER
      STA OURVEC+1
;
      SEI          ;INTERRUPTS OFF
      LDA IRQVEC   ;SWAP EXISTING
      LDX OURVEC   ;IRQ VECTOR FOR
      STA OURVEC   ;OUR VECTOR
      STX IRQVEC
      LDA IRQVEC+1
      LDX OURVEC+1
      STA OURVEC+1
      STX IRQVEC+1
;
;++++ INITIALISE TABLE ++++
;
      LDA #<MOTTAB
      STA ZPAGE
      LDA #>MOTTAB
      STA ZPAGE+1
;
      LDA #$FF
      LDY #$00
TABLE
      STA (ZPAGE),Y
      DEY
      BNE TABLE
      CLI          ;INTERRUPTS ON
      RTS
;
;++++ EVENT HANDLER ++++
;
EVENT
      PHP
      PHA          ;SAVE REGISTERS
      TYA          ;ON STACK
      PHA
      TXA
      PHA
;
;++ START PULSE, FOR SOME MOTORS IT
;MAY BE POSSIBLE TO START BEFORE FILLING
;TABLE AND SO REDUCE WAIT LOOP BELOW ++
;
      LDA #$FF
      STA PORT
;++ FILL TABLE WITH EXCEPTIONS ++
      LDX #$07
      LDA #$FF
      CLC
EXCEPT
      ROR A
      PHA          ;BIT PATTERN
      LDY ANGLE,X  ;GET MOTOR X OFFSET
      AND (ZPAGE),Y ;KEEP EXISTING PATTERN
      STA (ZPAGE),Y ;BUT MODIFIED FOR MOTOR X
      PLA
      DEX
      BPL EXCEPT
;++ TABLE IS NOW LOADED ++
      LDY #$30
WAIT
      DEY          ;FILL IN SOME
      BNE WAIT     ;TIME
;
      LDA #$FF     ;ALL PULSES ON
      LDY #$00
LOOP
      AND (ZPAGE),Y ;BUT MASK OFF WITH EACH
      STA PORT      ;TABLE ELEMENT IN TURN
      INY
      BNE LOOP
;
      LDX #$07
      LDA #$FF
CLEAR
      LDY ANGLE,X   ;CLEAR ALL EXCEPTIONS
      STA (ZPAGE),Y
      DEX
      BPL CLEAR
;++ ALL PULSES SHOULD NOW BE FINISHED ++
      PLA
      TAX
      PLA           ;RESTORE REGISTERS
      TAY
      PLA
      PLP
      JMP (OURVEC)  ;JUMP TO NORMAL
                    ;IRQ ROUTINE
;
;++++ SMOOTH SERVO MOVER ++++
;
LOOP2
      LDX #$00
      LDY #$00
LOOP1
      LDA NEWPOS,X ;NEW POSN FOR SERVO X
      CMP ANGLE,X  ;IS OLD POSN SAME?
      BEQ MOVED    ;IF = THEN DO NOTHING
      BCS ADD      ;IF > THEN ADD
      DEC ANGLE,X  ;ELSE SUBTRACT
      JMP INCRX
ADD
      INC ANGLE,X
      JMP INCRX
MOVED
      INY
INCRX
      INX
      CPX #$04     ;ALL 4 SERVOS DONE
      BNE LOOP1    ;IF NOT NEXT SERVO
      JSR WAITER   ;SHORT PAUSE
      CPY #$04     ;IF Y=4 THEN ALL MOVED
      BNE LOOP2
      RTS
;
WAITER
      TXA
      PHA          ;SAVE X & Y REGS
      TYA
      PHA
      LDY #$FF
      LDX DELFAC   ;GET DELAY FACTOR
ENCORE
      DEY
      NOP          ;256*8 CLOCK CYCLES
      NOP          ;WAIT
      BNE ENCORE
      DEX          ;IF MORE DELAY THEN
      BNE ENCORE   ;REPEAT
      PLA
      TAY
      PLA
      TAX
      RTS
;
;++++ CHECK KEYBOARD ++++
;
KEYTAB *=*+16
RESULT *=*+1
GETIN=$FFE4
;
CHECK
      PHA
      TXA
      PHA
      TYA
      PHA
NOKEY
      JSR GETIN   ;GET A CHAR
      CMP #$00    ;NO CHAR?
      BEQ NOKEY
      LDX #$00
COMPAR
      CMP KEYTAB,X
      BEQ EXIT    ;IS CHAR IN TABLE?
      INX
      CPX #17
```

```
      BNE COMPAR
      LDX #$FF      ;SIGNAL NOT FOUND
EXIT
      STX RESULT    ;STORE KEYTAB OFFSET
      PLA
      TAY
      PLA
      TAX
      PLA
      RTS
```

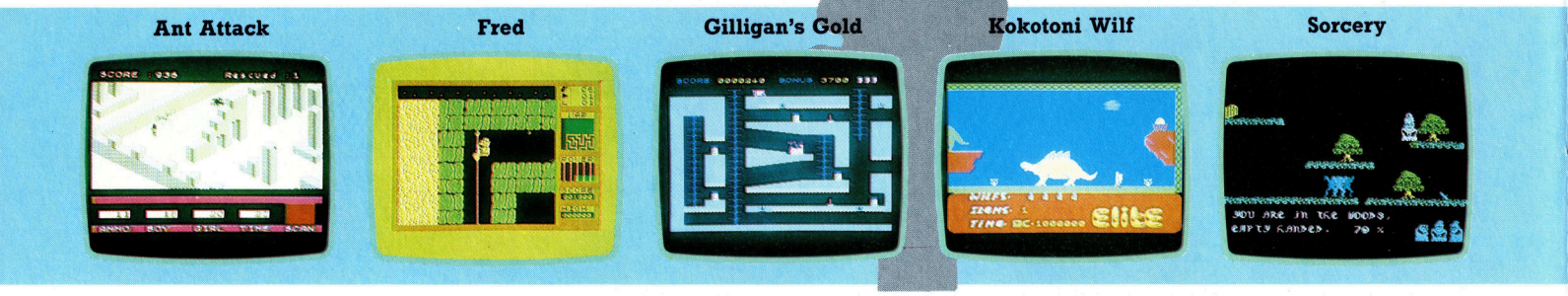
Arm-Steuerprogramm

```
10 REM ************************
20 REM ************************
30 REM **                    **
40 REM ** CBM ARM SEQUENCE  **
50 REM **     PROGRAMMER     **
70 REM **                    **
80 REM ************************
90 REM ************************
95 :
96 DN=8:REM FOR CASS DN=1
97 IF A=0 THEN A=1:LOAD"ROBARM.HEX",DN,1
100 GOSUB 1000:REM SET UP
110 PRINT CL$
120 X=5:Y=6:GOSUB10000:PRINT"WOULD YOU LIKE TO :"
130 X=4:Y=10:GOSUB10000:PRINT"1....PROGRAM NEW ARM
 SEQUENCE"
140 X=4:Y=12:GOSUB10000:PRINT"2....ADD MOVES TO A
PROGRAM"
150 X=4:Y=14:GOSUB10000:PRINT"3....REPLAY A FILE"
160 X=4:Y=16:GOSUB10000:PRINT"4....LEAVE PROGRAM"
170 GET G$:IF G$="" THEN 170
180 IF G$="1" THEN C=0:LM=0:REM REST ARRAY PTRS
190 IF G$="1" OR G$="2" THEN GOSUB2000:GOSUB3000:G
OSUB4000:GOSUB5000
200 IF G$="3" THEN GOSUB6000:GOSUB4000
210 IF G$="4" THEN PRINTCL$:END
220 GOTO 110
500 REM **** KEY FUNCTIONS ****
540 DX=DX+1:RETURN
550 DX=DX-1:IF DX<0 THEN DX=0
555 RETURN
560 P=PEEK(NP)+DX:IF P<256THEN POKENP,P
565 RETURN
570 P=PEEK(NP)-DX:IF P>0THEN POKENP,P
575 RETURN
580 P=PEEK(NP+1)+DX:IF P<256THEN POKENP+1,P
585 RETURN
590 P=PEEK(NP+1)-DX:IF P>0THEN POKENP+1,P
595 RETURN
600 P=PEEK(NP+2)-DX:IF P>0THEN POKENP+2,P
605 RETURN
610 P=PEEK(NP+2)+DX:IF P<256THEN POKENP+2,P
615 RETURN
620 P=PEEK(NP+3)+3*DX:IF P<256THEN POKENP+3,P
625 RETURN
630 P=PEEK(NP+3)-3*DX:IF P<256THEN POKENP+3,P
635 RETURN
640 FOR I=0 TO 3:POKE NP+I,R%(C,I):NEXT I:RETURN
650 C=C+1:IF C>MC THEN C=0
655 RETURN
660 C=C-1:IF C<0 THEN C=MC
665 RETURN
670 FOR I=0 TO 3:R%(C,I)=PEEK(NP+I):NEXT:C=C+1:RET
URN
680 FOR I=0 TO 3:R%(C,I)=PEEK(NP+I):NEXT:C=C+1:RET
URN
999 :
1000 REM **** SET UP ****
1010 C=0:NS=3:REM NO. OF SERVOS-1
1020 LM=0:OC=0:DF=10:REM DELAY FACTOR
1030 MC=100:REM MAX COUNT
1040 DIM R%(MC,NS):REM KEY POSITIONS ARRAY
1050 CL$=CHR$(147):REM CLEAR SCREEN
1060 DW$="":FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT I:
REM CURSOR DOWN
1065 POKE 650,128:REM SET KEYS TO REPEAT
1070 DL=49152:NP=49416:REM DELAY AND POSITION ADDR
ESSES
1075 GS=49155:OFF=49170:REM WEDGE ON/OFF SYS ADDRE
SSES
1077 MOVE=49281:REM MOVE SERVOS SYS ADDRESSES
1078 KEYTAB=49343:RESULT=49359:REM KEYPRESS ADDRES
SES
1079 CHECK=49360:REM KEY CHECK SYS ADDRESS
1080 REM ** READ KEYPRESS ASC DATA **
1090 FOR I=0 TO 14:READ A:POKE KEYTAB+I,A:NEXT I
1100 DATA 73,68,157,29,145,17,65,90,88,67
1110 DATA 82,78,66,83,81
1200 REM ** ZERO NEWPOS LOCATIONS **
1210 FOR I=0 TO 3:POKENP+I,0:NEXT I
1990 RETURN
1999 :
2000 REM **** INFORM ****
2010 PRINTCL$
2020 X=5:Y=5:GOSUB10000:PRINT"PLEASE USE :"
2030 X=1:Y=7:GOSUB10000:PRINT"CURSOR KEYS...FOR L/
R AND 1ST ARM U/D"
2040 X=1:Y=9:GOSUB10000:PRINT"A & Z.........FOR 2N
D ARM U/D"
2050 X=1:Y=11:GOSUB10000:PRINT"X & C.........FOR G
RAB OPEN/CLOSE"
2060 X=1:Y=13:GOSUB10000:PRINT"S.............TO SA
VE A POSITION"
2070 X=1:Y=15:GOSUB10000:PRINT"Q.............TO RE
TURN TO MENU"
2080 X=1:Y=17:GOSUB10000:PRINT"R.............TO MO
VE TO SAVED POSITION"
2090 X=1:Y=19:GOSUB10000:PRINT"N & B.........NEXT
AND BACK COUNT"
2100 X=1:Y=21:GOSUB10000:PRINT"E.............SET N
EW COUNT"
2110 X=1:Y=23:GOSUB10000:PRINT"I & D.........TO IN
C AND DEC SPEED"
2120 X=2:Y=2:GOSUB10000:PRINT"COUNT=";C;" ";
2130 FOR I=0 TO 3:PRINTR%(C,I);" ";:NEXT:PRINT
2135 GET A$:IF A$="" THEN 2135
2140 RETURN
2999 :
3000 REM **** PROGRAM ARM ****
3005 GOSUB7000:REM BLANK SCREEN AND START WEDGE
3010 POKE DL,1:REM SET DELAY FACTOR TO 1
3020 DX=8:REM CHANGE RATE
3030 SYS CHECK:REM CHECK KEYPRESS
3035 R=PEEK(RESULT)+1:IF R>15 THEN 3030:REM NON-VA
LID KEY
3037 ON R GOSUB 540,550,560,570,580,590,600,610,62
0,630,640,650,660,670
3190 IF C>LM THEN LM=C-1:REM RECORD MAX COUNT SO F
AR
3220 OC=C
3260 SYS MOVE:REM M/C MOVE SERVOS
3270 IF P=15 OR C>MC THEN GOSUB8000:RETURN:REM WED
GE OFF AND QUIT
3280 GOTO 3030:REM REPEAT
3999 :
4000 REM **** REPLAY MOVEMENT SEQUENCE ****
4010 PRINTCL$
4020 X=5:Y=23:GOSUB10000:PRINT"REPLAY SEQUENCE (Y/
N), R REPEATS"
4030 GET AN$:IF AN$<>"Y" AND AN$<>"N" AND AN$<>"R"
 THEN 4030
4040 IF AN$="N" THEN RETURN
4050 IF AN$<>"R" THEN X=5:Y=22:GOSUB10000:INPUT"DE
LAY FACTOR 1-255";DF
4060 IF DF<0 OR DF>255 THEN 4050:REM CHECK RANGE
4070 POKE DL,DF:REM SET DELAY FACTOR REGISTER
4075 GOSUB7000:REM BLANK SCREEN AND START WEDGE
4080 FOR I=0 TO LM
4090 X=5:Y=2:GOSUB10000:PRINT"NO. IN SEQUENCE = ";
I;" ";
4100 FOR S=0 TO 3
4110 POKE NP+S,R%(I,S)
4120 NEXT S:SYS MOVE:REM M/C MOVE SERVOS
4130 NEXT I
4135 GOSUB8000:REM RESTORE SCREEN AND TURN OFF WED
GE
4140 GOTO 4010:REM REPEAT
4999 :
5000 REM **** SAVE A SEQUENCE ****
5010 PRINTCL$
5020 X=5:Y=10:GOSUB10000:PRINT"SAVE SEQUENCE ON FI
LE (Y/N)"
5030 GET AN$:IF AN$<>"Y" AND AN$<>"N" THEN 5030
5040 IF AN$="N" THEN RETURN
5050 X=5:Y=12:GOSUB10000:INPUT"FILENAME";FL$
5060 OPEN 3,DN,3,"@0:"+FL$+".DATA,S,W":REM OPEN FI
LE
5070 PRINT#3,LM
5080 FOR A=0 TO LM:FOR B=0 TO 3
5090 PRINT#3,R%(A,B):NEXT B,A
5100 PRINT#3:CLOSE3
5120 RETURN
5999 :
6000 REM **** LOAD A FILE ****
6010 PRINTCL$
6020 X=5:Y=10:GOSUB10000:INPUT"FILENAME TO LOAD";F
L$
6040 OPEN 3,DN,3,"@0:"+FL$+".DATA,S,R":REM OPEN FI
LE
6050 INPUT#3,C:LM=C
6060 FOR A=0 TO C:FOR B=0 TO 3
6070 INPUT#3,R%(A,B):NEXT B,A
6080 PRINT#3:CLOSE3
6100 RETURN
6999 :
7000 REM **** START WEDGE ****
7010 POKE53265,PEEK(53265)AND 239:REM BLANK SCREEN

7015 SYS GS
7020 RETURN
7999 :
8000 REM **** STOP WEDGE ****
8010 POKE53265,PEEK(53265)OR 16:REM BLANK SCREEN
8015 SYS OFF
8020 RETURN
8999 :
10000 REM **** POSITION CURSOR AT X,Y ****
10010 PRINTCHR$(19);:PRINTTAB(X);:LEFT$(DW$,Y);
10020 RETURN
```



Brot und Spiele



Hier zeigen wir eine Auswahl von Screens aus der Spectrum-Version von SoftAid. Die auf dem Sampler enthaltenen Programme sind bewährte Top-Titel. Alle Erlöse aus dem Verkauf der Cassette wurden an Band-Aid weitergeleitet.

SoftAid ist ein Zusammenschluß von mehreren Software-Häusern, die gemeinsam Geld sammeln, um Bedürftigen in Afrika zu helfen. Mit eigenen Mitteln wurde eine Cassette produziert, die mehrere tausend Mark an Spenden für die Aktion „Band-Aid" brachte.

Die „Band-Aid"-Aktivitäten Bob Geldofs rüttelten 1985 weltweit Millionen von Menschen auf. Neben dem Erfolg der Single „Do They Know It's Christmas?" Ende 1984 und den „Live Aid"-Konzerten in London und Philadelphia im Juli 1985 unterstützten auch Organisationen aus anderen Unterhaltungsbranchen die Arbeit von Live Aid und sammelten Geld für Afrika. Eine der erfolgreichsten Initiativen ging von englischen Software-Verlegern aus. Das Ergebnis war „SoftAid", ein Programm-Sampler für die populärsten Heimcomputer mit interessanten Top-Titeln.

Treibende Kraft hinter „SoftAid" war die Guild of Software Houses (GOSH), eine Branchenorganisation der Computerspiel-Industrie, deren eigentlicher Aufgabenbereich die Verfolgung von Software-Piraten ist. Sie bat ihre Mitglieder, mit Spielen an dem Hilfsprojekt teilzunehmen. Ursprünglich wollte man für diesen Sampler neue Programme schreiben. Es erwies sich aber als einfacher, bereits vorhandene, erfolgreiche Spiele zu einem Sampler zusammenzustellen.

Die Reaktionen der Software-Häuser waren überwältigend: GOSH bekam mehr Programme zur Verfügung gestellt, als auf einer Cassette für den ZX Spectrum und den C 64 unterzubringen waren. So mußte man auf Programme renommierter Software-Häuser wie LlamaSoft oder Software Projects verzichten. Trotzdem gehörten die ausgewählten Programme zu den besten Games der letzten Jahre: darunter befand sich auch „Ant Attack".

Die vollständige Programmliste für den Spectrum umfaßt „Spellbound" (Beyond), „Starbike" (The Edge), „Kokotoni Wilf" (Elite), „The Pyramid" (Fantasy), „Horace Goes Skiing" (Melbourne House), „Gilligan's Gold" (Ocean), „Ant Attack" (Quicksilva), „3D Tank Duel" (Real Time), „Jack and the Beanstalk" (Thor) und „Sorcery" von Virgin. Die Cassette für den C 64 enthält „Gumshoe" (A'n'F), „Beam Rider" (Activision), „Star Trader" (Buy-Byte), „Gyropod" (Taskset), „China Miner" (Interceptor), „Kokotoni Wilf" (Elite), „Gilligan's Gold" (Ocean), „Fred" (Quicksilva), „Falcon Patrol" (Virgin) und „Flak" von US-Gold. Während der Projektentwicklung arbeitete GOSH eng mit Band-Aid zusammen. Der Single-Hit von Band-Aid „Do They Know It's Christmas?", ebenfalls auf den Systemcassetten enthalten, war ein zusätzlicher Anreiz, den Sampler zu erwerben.

Ein echter Hit

Auch bei Herstellung und Vertrieb ging GOSH nach der generellen Band-Aid-Philosophie vor: Man bat die gesamte Branche um Unterstützung und bekam so nicht nur die Programme kostenlos, sondern auch Druck, Werbung, Cassettenvervielfältigung und Vertrieb; lediglich die reinen Materialkosten mußten bezahlt werden. Im Frühjahr 1985 wurden die Programmsammlungen veröffentlicht und kamen sofort an die Spitze der Bestseller-Listen. Kein Wunder, denn der Wunsch zu helfen wird durch die Sammlung durchaus sehenswerter Spiele kräftig gefördert.

SoftAid: Für ZX Spectrum und C 64. Versionen für Acorn B und Schneider befinden sich in Vorbereitung.
Vertrieb: Rushware GmbH, An der Guempgesbrücke, 4044 Kaarst
Autoren: Verschiedene
Steuerknüppel: Optional
Programm: Cassette

Kostenprobleme

Der Einzug von Computern in den englischen Schulunterricht hat eine Reihe von Problemen mit sich gebracht – nicht zuletzt aufgrund der mangelnden Unterstützung seitens der Regierung.

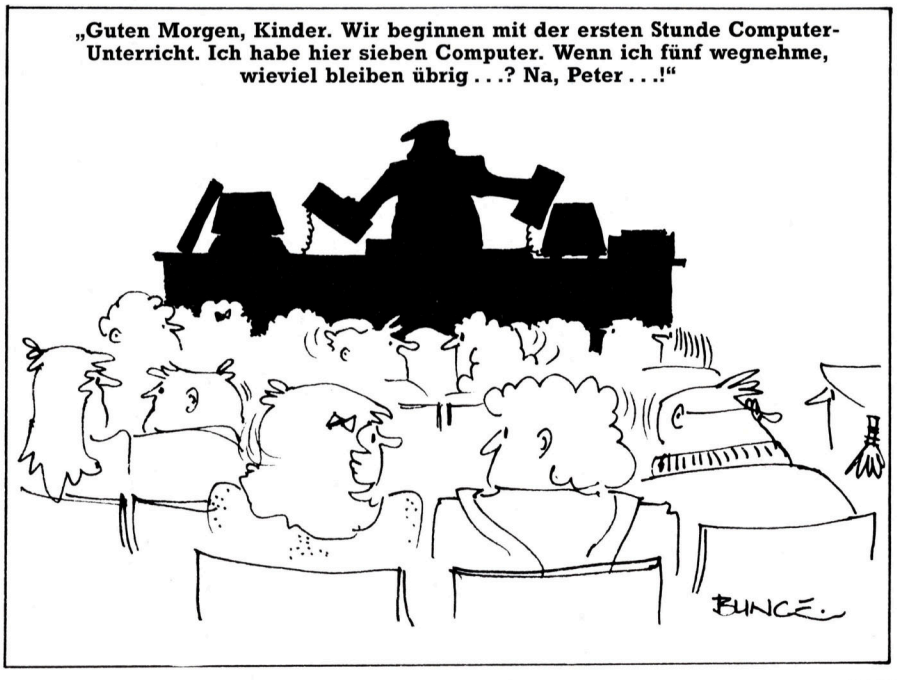
Probleme hat es seit Einführung der Computer in den Unterricht stets gegeben. Sparmaßnahmen, ausgeprägte Ignoranz gegenüber dem Thema an sich und den Schwierigkeiten bei der Ausbildung der Lehrkräfte sind Ursachen für den geringen Einsatz von Computern im Schulunterricht. Die neue Technologie ist von Regierungsbeschlüssen abhängig, und die beträchtlichen Sparmaßnahmen verhindern die angemessene Ausstattung der Schulen. Nach dem Erwerb kostspieliger Hardware ärgern sich viele britische Lehrer darüber, daß keine Mittel für den Kauf geeigneter Software zur Verfügung stehen.

Diese fehlende Unterstützung ist auch Grund dafür, daß mehrere Projekte, an denen Kinder und Computer beteiligt sein sollten, nicht begonnen werden konnten. Man hat kaum darüber nachgedacht, inwieweit Computer in bestehende Lehrpläne einbezogen werden könnten. Folglich gibt es auch keine klare Strategie für ihre Einführung an Schulen. TIC-CIT und PLATO in den USA sowie das National Development Program in England wurden in der „Vor-Micro-Zeit" Anfang der 70er Jahre durchgeführt, als noch Gelder flüssig gemacht werden konnten. In den 80er Jahren aber sehen sich die Regierungen beider Länder außerstande, ähnliche Projekte zu finanzieren, obgleich ein dringender Bedarf dafür vorhanden ist. An den weiterführenden Schulen wird über BASIC und Computer-Studien generell heiß diskutiert. Aber: Man stößt auf Ablehnung. Grundschulen werden mit einem Computer ausgestattet – der Lehrer muß alleine sehen, wie er damit klarkommt.

LOGO zum Spitzenpreis

Es gibt ernsthafte Kommunikationsstörungen in der pädagogischen Computerwelt. In London investieren viele Schulen in RML-Rechner. Folglich gab das ILECC – das „Inner London Educational Computing Centre" – viel Geld aus und wandte viel Zeit auf, um eine LOGO-Version für RML-Rechner zu schreiben. Inzwischen aber stellte Research Machines Limited selbst eine volle Implementierung der Sprache her. Unabhängig davon warf die ILECC weiter gutes Geld für eine schlechte Sache aus dem Fenster, um das eigene, schlechtere Turtle-Grafik-Programm herzustellen und an den Schulen zu verbreiten.

Weitere Probleme ergeben sich aus den fehlenden Ausbildungsmöglichkeiten für Lehrer. Man hat Computer im Klassenzimmer aufgestellt und erwartete von den Lehrern, daß sie alles aus den Handbüchern lernen würden.



Unterstützt durch die BBC und von der Regierung für die Benutzung in Schulen empfohlen, spielen Acorns-B-Rechner eine entscheidende Rolle im britischen Schulsystem. Die auf diesen Seiten gezeigten Diagramme sind einer Untersuchung entnommen, die BBCs Educational Broadcasting Services Research Unit im Januar 1985 veröffentlichte. Überraschend dabei ist die noch immer währende Präsenz von Commodores Pet und dem Sinclair ZX 81.

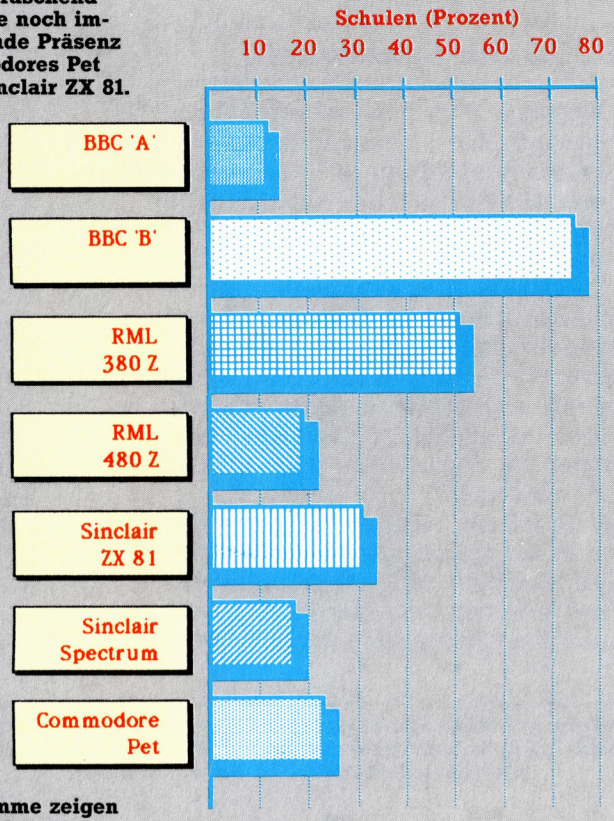
Wegen der Sparmaßnahmen sind Einführungskurse mittlerweile unbezahlbar. Zu jener Zeit, als noch Geld dafür zur Verfügung gestellt wurde, weigerten sich die Schulen, die Lehrer wegen des Unterrichtsausfalls auf Kurse zu schicken.

Diese Situation an Lehrerseminaren ist noch schlimmer. 1983 stellte das Erziehungsministerium besorgt fest, daß es kaum Computerausbildungskurse für Lehrer gäbe. Unterstützt vom Wirtschaftsministerium empfahl man Kurse von mindestens 20 Stunden Dauer. Chris Gregory, Dozent für Mathematik am Bradford College und Mitglied des Projekts MEP (Microcomputers in Education Project), sagte: „20 Stunden sind ohnehin lächerlich, aber viele Kollegen schaffen nicht einmal das." Ein Hauptproblem ist das Fachpersonal. Das Erziehungsministerium beläßt das Personal beim derzeitigen Stand oder streicht Planstellen. Unter diesen Voraussetzungen kann man kaum Computerfachleute an den Schulen finden.

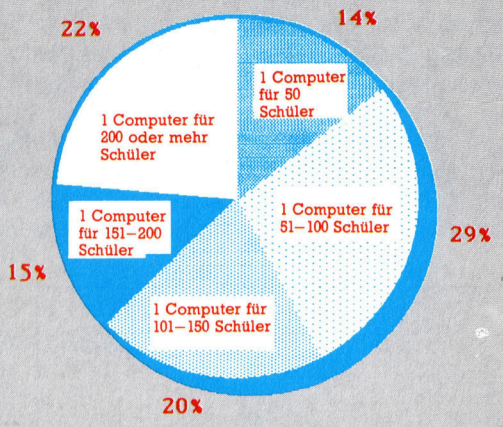
Gute Lernprogramme gelangen nur selten auf den Markt. Die Regierung hat in Hardware, nicht aber in Software investiert. Da die Schulen kein Geld ausgeben können, liegt die Entwicklung von Software im argen. Anläßlich der Educational Publishers Software-Ausstellung 1983 ergab sich, daß die meisten in diesem Bereich tätigen Unternehmen Verluste hinnehmen mußten. Was den Markt noch komplizierter macht, ist der Umstand, daß viele Schulbehörden Geld in eigene Software-Entwicklungen gesteckt haben. Damit sind die teureren kommerziellen Programme noch weniger attraktiv, der Marktanreiz erstirbt.

Sloane Software beispielsweise veröffentlichte kürzlich ein „Hopslide" betiteltes Mathematik-Programm für fünf- bis siebenjährige Kinder. Es ist ein einfaches Programm, das zum Verständnis von Zahlenwert und Zahlenfolge beiträgt. „Hopslide" kostet weniger als so manches Schulbuch und könnte gut verkäuflich sein. Die Inner London Education Authority produziert aber eigene Software. So gibt es ebenfalls ein Hopslide-Programm, das im SMILE-Mathematik-Programmpaket enthalten ist. Es ist identisch mit der Sloane Software-Version, abgesehen davon, daß der Zahlenbereich größer und die Grafik etwas besser ist. Das SMILE-Programm enthält 19 weitere, ähnliche Programme und kostet nur unbedeutend mehr als die Sloane-Version.

Geldmangel und schlechte Organisation sind aber nicht die alleinigen Ursachen für die Misere. Es gibt andere Probleme, die bei der Einführung der neuen Technologie in Schulen hinderlich sind. So hat allein die physische Präsenz von Computern Probleme verursacht. Elektronische Ausrüstung wird in vielen Schulen nachts in einem gesicherten Raum deponiert. Bevor es Computer gab, waren dort zwei oder drei Cassetten- und Videorecorder untergebracht. In einer Grundschule muß nun ein Lehrer zwei- oder dreimal mit Computer, Monitor, Kabeln, Diskettenstation und Software über einen langen Gang gehen, ein paar Treppen steigen, und dies jeden Morgen und Abend. Schließlich weigerten sich Lehrer, die Ausrüstung weiterhin zu schleppen. Die Schulbehörde bestand jedoch darauf, daß die Computer gesichert untergebracht sein müßten. Ergebnis: Die Computer blieben vier Monate un-



Rechnertypen an Schulen



Schüler je Computer

Die Diagramme zeigen deutlich, daß trotz der Versuche der Regierung, das Verhältnis Schüler/Microcomputer an englischen Schulen zu verbessern, eine von fünf Schulen noch immer mit einem Computer für durchschnittlich 200 Kinder auskommen muß.

genutzt im Aufbewahrungsraum, während Lehrergewerkschaft und Behörden versuchten, einen Kompromiß zu finden. Man einigte sich schließlich darauf, Stahlschränke im Klassenzimmer aufzustellen.

Die Situation an vielen Schulen ist beklagenswert. Einfacher Grund dafür: Man hat kein Geld, um Sicherheitssysteme zu installieren. Die Schulen in Englands Ballungszentren werden jährlich mehrfach von Dieben und Einbrechern heimgesucht. Die Versicherungsprämie ist so hoch, daß die Behörden es für preiswerter erachten, gestohlene Gegenstände neu zu erwerben, als sie zu versichern.

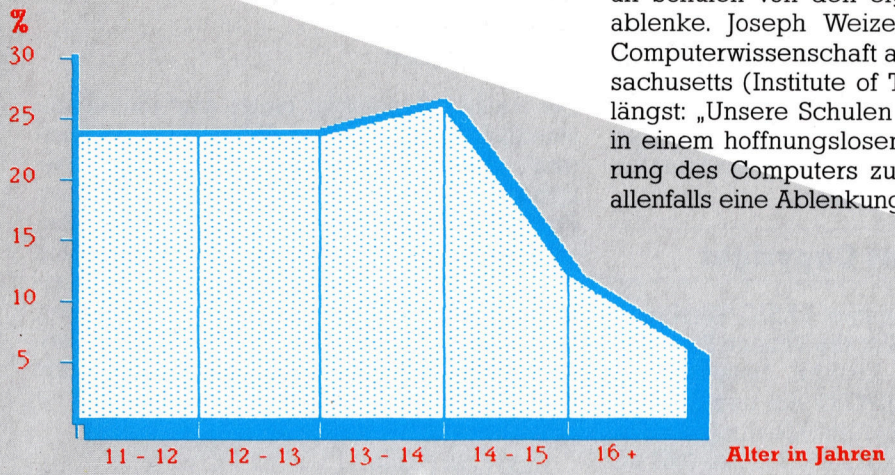
Männersache!

Doch auch, wenn die Computer tatsächlich eingesetzt werden können, gibt es Probleme. Die viel zu wenigen Computer an Schulen scheinen überdies im Unterricht Jungen vorbehalten zu sein. Jungen sind auch meist Besitzer von Heimcomputern, die sie hauptsächlich zum Spielen benutzen. Ihre Games sind oft militant und aggressiv. Im Klassenzimmer setzt sich der gewohnte Umgang mit dem Rechner fort. Der Computer wird als männliche Domäne betrachtet. Die Lernprogramme sind meist für Jungen geschrieben, und diese monopolisieren die Computer zu Lasten der Mädchen. Ein Sprecher der Equal Opportunities Unit der Inner London Education Authority sagte: „Wir sind uns dieser Unausgewogenheit beim Computerunterricht bewußt. In reinen Mädchen-Schulen gibt es das Problem nicht. Es existiert nur in gemischten Schulen, wo Jungen dominieren. Dort müssen wir die Mädchen dazu ermutigen, Interesse am Computer zu finden, um so das Gleichgewicht herzustellen." Interessant ist das Forschungsergebnis der Universität Edinburgh, die den Einsatz von LOGO an Schulen untersuchte: Mädchen wie Jungen waren gleichermaßen interessiert.

Das größte Problem ist jedoch, wie die Regierung Ausbildung bewertet, nämlich nach dem Produktivitäts- und Kostenschlüssel. Betrachtet man den Computer als pädagogisches Lernwerkzeug, ist die Zukunft erschreckend: Kinder werden zum Bestandteil einer computerisierten Produktionsfolge, Lehrer werden durch Rechner ersetzt und Lehrpläne von Regierungskomitees festgelegt. Wie Seymour Papert anmerkte: „Es gibt starke Kräfte in der Welt, die auf eine starrere Gesellschaft drängen und den Computer auf eine Weise nutzen wollen, die das Leben weniger aufregend macht, starrer und unsozialer. Was können wir tun, um ihnen zu begegnen . . .? Wir können sagen: ‚Verbrennt die Computer!'. Aber das wäre Zeitvergeudung . . . Oder wir können versuchen, das Ruder herumzureißen und den Computer auf eine sozial wünschenswerte Weise zu nutzen."

Die Ausbildung auf beiden Seiten des Atlantik steckt in der Krise. Mangel an Investitionen, niedrige Löhne sowie eine generelle Abkehr von sozialen Errungenschaften verlangen ihren Tribut. Es gibt Menschen, die damit argumentieren, daß die Einführung von Computern an Schulen von den eigentlichen Problemen ablenke. Joseph Weizenbaum, Professor für Computerwissenschaft an der Universität Massachusetts (Institute of Technology) sagte unlängst: „Unsere Schulen befinden sich bereits in einem hoffnungslosen Zustand. Die Einführung des Computers zu diesem Zeitpunkt ist allenfalls eine Ablenkung . . . Er wird benutzt . . . um einen technologischen Fixpunkt zu setzen . . . und dabei fundamentale Probleme zu kaschieren, indem die Illusion erzeugt wird, man setze sich mit ihnen auseinander."

Jener Teil der Gesellschaft, der letztendlich über Einsatz oder Nicht-Einsatz von Computern entscheidet, sind die Kinder. Sie sagen immer ihre Meinung und wissen oft genau, ob etwas gut für sie ist oder langweilig und unwichtig. Sie reagieren auf gute Software. In einer vom Computer bestimmen Umwelt schaffen Kinder ihre eigene Computer-Kultur und entwickeln ihre eigene Sprache.



Die Grafik zeigt die „Computererfahrung", die Schulkinder in England haben. Dabei werden die Auswirkungen der Spezialisierung im oberen Altersbereich deutlich. Ältere Schüler finden in ihren Leistungsfächern oft keine Anwendungsmöglichkeit für Computer. Lediglich 30 % der befragten Altersgruppen geben an, den Computer „im Griff" zu haben.

Computer „im Griff"

Zwei Versionen

In dieser und der nächsten Folge untersuchen wir, wie BASIC mit dem Betriebssystem des Acorn B kommuniziert und nehmen die Fehlerbearbeitungsroutinen des Rechners aufs Korn.

Das BASIC-ROM des Acorn B belegt die Speicheradressen zwischen &8000 und &BFFF. Dieser Bereich wird auch als „Paged ROM" bezeichnet. Dabei können sich mehrere verschiedene ROM-Chips in der Maschine befinden, die mit dem gleichen Speicherbereich arbeiten, von denen jedoch nur jeweils einer aktiv sein kann. Beispiele sind der DFS-Chip, der Econet-Chip, Programm-Chips wie Wordwise, View oder Disk Doctor und Sprachen wie FORTH oder BCPL. Die Technik der Paged ROMs spart viel Speicherplatz.

Der Wert des Paging-Registers bestimmt, welches Paged ROM aktiv ist. Dieses Register ist Teil der Hardware, die vom OS gesteuert wird. Es kann bis zu 16 Paged ROMs aufrufen. Wenn das OS einen Befehl nicht identifizieren kann, spricht es zunächst die Paged ROMs an. Bei einem unbekannten *-Befehl werden daher erst die Paged ROMs untersucht, bevor die Fehlermeldung „BAD COMMAND" erscheint. Beispielsweise wird *INFO vom OS-ROM nicht erkannt und an die Paged ROMs weitergereicht. Ist das DFS-ROM installiert, wird der Befehl akzeptiert und ausgeführt.

Informatives Copyright

Für den Acorn B gibt es zwei BASIC-Versionen: BASIC I und BASIC II. Um die Version Ihrer Maschine feststellen zu können, betätigen Sie CTRL-Break, geben REPORT ein und drücken Return. Enthält die Copyright-Meldung das Datum 1982, arbeiten Sie mit BASIC II, bei 1981 mit der älteren Version BASIC I. Im neueren Dialekt wurden einige Fehler des BASIC I korrigiert. Es hat außerdem zusätzliche Fähigkeiten zur Ausführung bereit.

Neu sind unter anderem der Dateibefehl OPENUP und die modifizierte Funktion von OPENIN. In BASIC II eröffnet OPENIN eine Datei nur für die Eingabe, während in BASIC I damit die Ein- und Ausgabe möglich war – eine Fähigkeit, die sich für Random-Access-Dateien eignet. BASIC II hat für diesen Zweck den Befehl OPENUP, der Dateien für die Ein- und Ausgabe eröffnet und dem OPENIN des BASIC I entspricht. Diese Änderung verursacht Probleme bei der Übertragung von Programmen zwischen Maschinen mit unterschiedlichen BASIC-Versionen, da der Token, der OPENIN in BASIC II darstellt, von BASIC I nicht erkannt wird. Der BASIC-II-Befehl:

```
Y%=OPENIN("FRED")
```

erscheint in BASIC I als

```
Y%=("FRED")
```

und ruft dort eine Fehlermeldung hervor.

Einige neue Eigenschaften von BASIC II sind besonders für den Maschinencodeprogrammierer interessant. Eine Verbesserung sollte besonders erwähnt werden. Um unter BASIC I Daten in Maschinencodeprogramme einsetzen zu können, mußte erst der Assembler verlassen und die Operationen ?, ! und $ eingesetzt werden. Hier ein Beispiel:

```
1000 LDA data
1010 .data
1020 ]
1030 ?P%=00
1040 P%=P%+1
1050 OPT pass
```

Die meisten Assembler bieten statt dieser umständlichen Methode „Assembleranweisungen", die den Assembler veranlassen, bestimmte Aufgaben auszuführen. In unserem Programmbeispiel ist OPT eine derartige Anweisung. BASIC II bietet vier weitere Befehle dieser Art: EQUB reserviert ein einzelnes Speicherbyte, EQUD zwei Bytes, EQUW vier Bytes, und mit EQUS kann ein String im Speicher angelegt werden. Das BASIC-I-Programm sieht daher in BASIC II so aus:

```
1000 LDA Daten
1010 .Daten EQUB 00
1020 . . . weitere Programmteile
```

Dabei wird das Register A mit dem Byte der Adresse „Daten" geladen (hier eine Null). Ein weiteres Beispiel:

```
1000 .Meldung EQUS "Hallo!"
```

Hier werden die Bytes, die die Meldung „Hallo!" darstellen, im Speicher von der Adresse mit dem Label .Meldung an gespeichert. Bei all diesen Assembleranweisungen wird P% automatisch hochgezählt.

Auch der BASIC-II-Befehl OSCLI ist sehr praktisch. Wir hatten bereits erwähnt, daß *-Befehle per OSCLI (bei &FFF7) an das Betriebssystem übergeben werden. In BASIC I mußten dafür erst die Variablen X% und Y% (die Register X und Y) mit der Adresse des Befehlsstrings geladen werden. Der neue OSCLI-

Befehl ist weit einfacher. Vergleichen Sie einmal die beiden Versionen:

BASIC I	BASIC II
\$&A00="*TAPE" X%=&A00 MOD 256 Y%=&A00 DIV 256 CALL &FFF7	OSCLI("*TAPE")

Außer den zwei erwähnten BASIC-Versionen gibt es noch ein BASIC II für den amerikanischen Markt, das die andersartige Bildschirmdarstellung berücksichtigt. Zudem besitzt der zweite 6502-Prozessor eine eigene BASIC-Version namens „HI BASIC", die eine Mischung der BASIC II US-Version und BASIC II ist.

Speichertopographie

Zwischen &400 und &7FF ist der Speicher für die aktuelle Sprache reserviert („Language Workspace"). Auch auf der Zero Page stehen die Adressen zwischen &00 und &6F der aktuellen Sprache zur Verfügung, wenn auch der Platz von &50 bis &6F nicht oft eingesetzt wird. Auf Speicherseite 4 liegen die BASIC-Variablen: zwischen &400 und &46B die Werte für die eingebauten Ganzzahlvariablen wie A% oder X%. @% wird in den vier Bytes von &400 bis &403 gespeichert (mit dem niederwertigen Byte in &400), A% von &404 bis &407 – und so weiter bis Z%.

Zwischen &480 und &4F9 befindet sich der Index des BASIC-Interpreters. Er gibt die Bereiche an, in denen Arrays und die anderen Fließkomma- oder Stringvariablen gespeichert sind. So zeigen beispielsweise die Bytes in &482 und &483 auf einen Speicherbereich zwischen TOP und HIMEM, in dem sich die Daten aller zur Zeit eingesetzten Variablen befinden, deren Namen mit A beginnen. Die in &4F6 und &4F7 gespeicherte Adresse zeigt auf den Speicher der BASIC-Prozeduren und die Adresse in &4F8 und &4F9 auf den Bereich für Funktionen. Auf diese Weise sind die Übersichtlichkeit und der geregelte Zugriff jederzeit sicher gewährleistet.

Schleifeninformationen

Das RAM zwischen &500 und &5FF dient dem BASIC-Interpreter als Stack (darf nicht mit dem Maschinenstack des 6502 verwechselt werden, der auf Speicherseite 1 liegt). Die Seite 5 verwendet BASIC für die RETURN-Daten der GOSUBs. Dort liegen auch alle Informationen, die für die Ausführung von FOR . . .NEXT- und REPEAT . . .UNTIL-Schleifen nötig sind.

Ihre Ausführung verlangt vom BASIC-Interpreter recht komplexe „Denkleistungen". Sobald etwa FOR im Programm auftritt, muß sich der Rechner merken, wohin er beim zu erwartenden NEXT zurückspringen muß.

Die Seite 6 setzt BASIC entweder für Stringvariablen ein oder für die Ausführung von Maschinencoderoutinen, die mit der erweiterten Version von CALL aufgerufen werden. Seite 7 dient als Eingabebuffer für BASIC-Befehle oder Programmzeilen. Die Eingaben werden dort bis zu ihrer Weiterverarbeitung gespeichert. Direkte Befehle werden interpretiert und ausgeführt und Programmzeilen an der entsprechenden Position in das Programm eingesetzt. Wenn dieser Speicherbereich nicht für BASIC benötigt wird, steht er für andere Zwecke zur Verfügung.

Sicher haben Sie schon gesehen, wie Ihr Computer einen fehlerhaften Befehl mit einer Fehlermeldung quittierte. Auf dem Acorn B können Sie eigene Fehleranweisungen mit Fehlernummer und Meldung anlegen. Diese Möglichkeit ist besonders für Maschinencodeprogramme praktisch, die zusammen mit BASIC eingesetzt werden. Der Inhalt des Vektors BRKV (bei &202 und &203) zeigt auf die Routine, die die von BASIC-Programmen erzeugten Fehler bearbeitet.

Kommentierbare Fehler

Sobald BASIC einen Fehler entdeckt – beispielsweise bei einer Teilung durch Null –, wird der Maschinencodebefehl BRK ausgeführt. Dadurch spricht der 6502 eine Routine an, deren Adresse in BRKV gespeichert ist. Auf den BRK-Befehl müssen eine Reihe Bytes folgen, die folgendes Format haben:

BRK
Fehlernummer (ein Byte)
Fehlermeldung (eine Bytefolge)
Ende der Meldung (ein Null-Byte)

Fehler Nummer 4 (der die Meldung „Mistake" ausgibt) wird mit folgender Routine bearbeitet:

Byte	Beschreibung
BRK	Befehl
4	Fehlernummer
M	
i	
s	ASCII-Code der
t	Fehlermeldung
a	
k	
e	
00	Ende der Meldung

Das BASIC-Fehlermodul setzt nun die Systemvariable ERL auf die Nummer der Zeile, in der der Fehler auftrat, und ERR auf die Fehlernummer – in diesem Fall 4. Das Fehlermodul stellt sicher, daß bei der nächsten Bewertung von REPORT die Fehlermeldung „Mistake" ausgegeben wird.

Der Befehl BRK läßt sich aber auch in selbsterstellten Maschinencodeprogrammen einsetzen, und somit können eigene Fehlermeldungen definiert werden. Der BASIC-Befehl ON ERROR GOTO behandelt diese neu generierten Meldungen wie die normalen, vom BASIC-Interpreter gemeldeten Fehler. So verursacht der folgende Maschinencode die Fehlermeldung „Zahl zu groß":

```
1010 BRK
1020 EQUB 254
1030 EQUS "Zahl zu groß"
1040 EQUB 00
```

Nach der Ausführung erhält ERR die Zahl 254 und REPORT die Meldung "Zahl zu groß".

Da der Inhalt dieses Vektors verändert werden kann, läßt sich die gesamte Fehlerbehandlung des Acorn B umstellen. Es besteht sogar die Möglichkeit, neue Befehle an das BASIC des Acorn B anzufügen – die Programmierung dafür ist jedoch recht kompliziert.

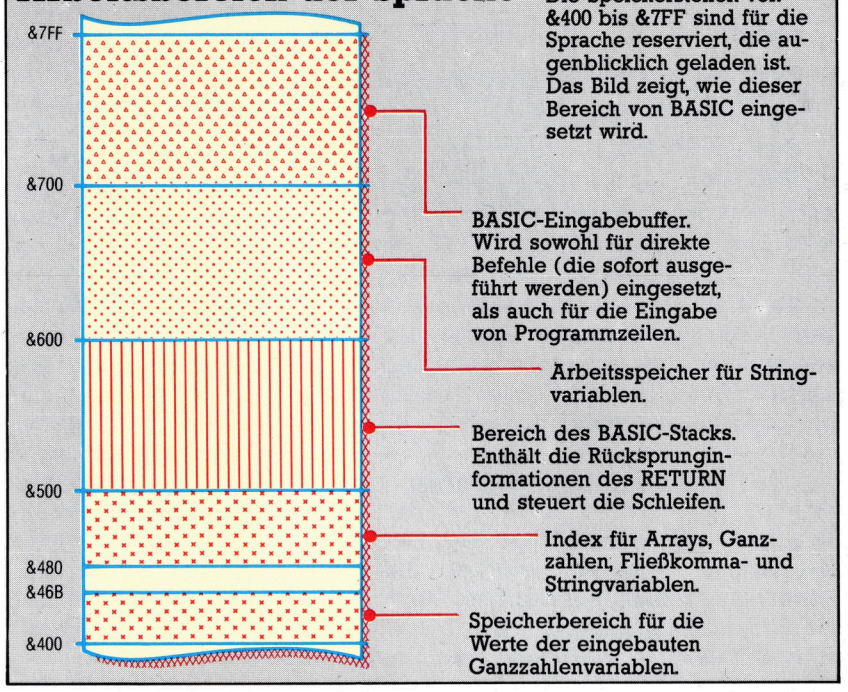
In der nächsten Folge sehen wir uns an, wie BASIC II mit dem Betriebssystem des Acorn B arbeitet und welche Vorteile es bei seiner Kooperation mit dem OS gegenüber seinem Vorgänger tatsächlich bietet.

Routineuntersuchung

Die Liste zeigt die ROM-Adressen der BASIC-Routinen des Acorn B, die wir in dieser Serie behandelt hatten. Wenn Sie sich die Routinen genauer ansehen, erhalten Sie einen guten Einblick in die internen Arbeitsabläufe des BASIC. Da nicht alle hier aufgeführten Routinen mit dem Befehl RTS enden, empfehlen wir, sie nacheinander von einem eigenen Programm aus aufzurufen. Die Analyse dieser Techniken kann Ihnen wertvolle Hinweise für Ihre eigenen Programme liefern.

ROM-Einstiegsadresse	**&8000**
BGET Anweisung	&BF6F
BPUT Anweisung	&BF58
CHAIN Anweisung	&BF2A
CLOSE Anweisung	&BF99
EOF Funktion	&ACB8
INKEY Funktion	&ACAD
INKEY$ Funktion	&B026
INPUT Anweisung	&BA44
INPUT# Anweisung	&B9CF
LOAD Befehl	&BF24
OPENIN Funktion	&BF78
OPENOUT Funktion	&BF7C
OPENUP Funktion	&BF80
PRINT# Anweisung	&8D2B
SAVE Befehl	&BEF3
TIME Anweisung	&92C9
TIME Funktion	&AEB4
VDU Anweisung	&942F

Arbeitsbereich der Sprache



Unterschiede zwischen BASIC I und BASIC II

Befehl oder Fehler	BASIC I	BASIC II
REPORT	Liefert die Meldung 1981	Liefert die Meldung 1982
OPENUP	Nicht vorhanden	Eröffnet eine Datei für Lese- und Schreibvorgänge (Wie OPENIN in BASIC I)
OPENIN	Eröffnet eine Datei zum Lesen und Schreiben	Eröffnet eine Datei nur zum Lesen
OPT n	Es gibt nur OPT 1 bis 3	Es gibt OPT 1 bis 7. Die vier neuen Befehle (OPT 4 bis 7) haben die gleichen Funktionen wie die des BASIC I, werden jedoch auf der Adresse abgelegt, die in der Variablen 0% gespeichert ist
0%	Keine spezielle Bedeutung	Hat mit OPT (4 bis 7) die dort beschriebene Funktion
EQUB EQUD EQUS EQUW	Nicht vorhanden	Erlaubt dem Programmierer, Daten in Maschinencodeprogramme einzusetzen, ohne den Assembler verlassen zu müssen
INSTR (a$,b$)	Ein schwerer Fehler. Wenn a$ kürzer war als b$, bestand die Möglichkeit eines Programmabsturzes	Der Fehler wurde korrigiert
reelle Zahlen	Genauigkeit: neun Stellen	Genauigkeit: zehn Stellen
OSCLI	Nicht vorhanden	Eine einfache Möglichkeit, per BASIC String-Variablen-Befehle an das OS zu übergeben

Fachwörter von A bis Z

Loop = Schleife
Unter einer Schleife ist ein Programmabschnitt zu verstehen, der durch wiederholten Rücksprung mehrfach ausgeführt werden kann. Es gibt „bedingte“ und „unbedingte“ Schleifen.

Bei einer bedingten Schleife wird der zyklische Programmteil nur solange durchlaufen, bis eine bestimmte Bedingung erfüllt ist. In BASIC kann dieser Schleifentyp unterschiedlich realisiert werden, etwa durch die Anweisung DO LOOP...UNTIL, WHILE...WEND, REPEAT...UNTIL und FOR...NEXT.

Bei einer unbedingten Schleife erfolgt der Rücksprung zum Eingang ohne Bedingungsabfrage; ein Ausstieg ist nur durch äußeres Eingreifen möglich. Für solche Schleifen wird in BASIC das einfache GOTO verwendet – mit der Anweisung 10 GOTO 10 beispielsweise kann man aufs bequemste eine Endlosschleife ausführen lassen.

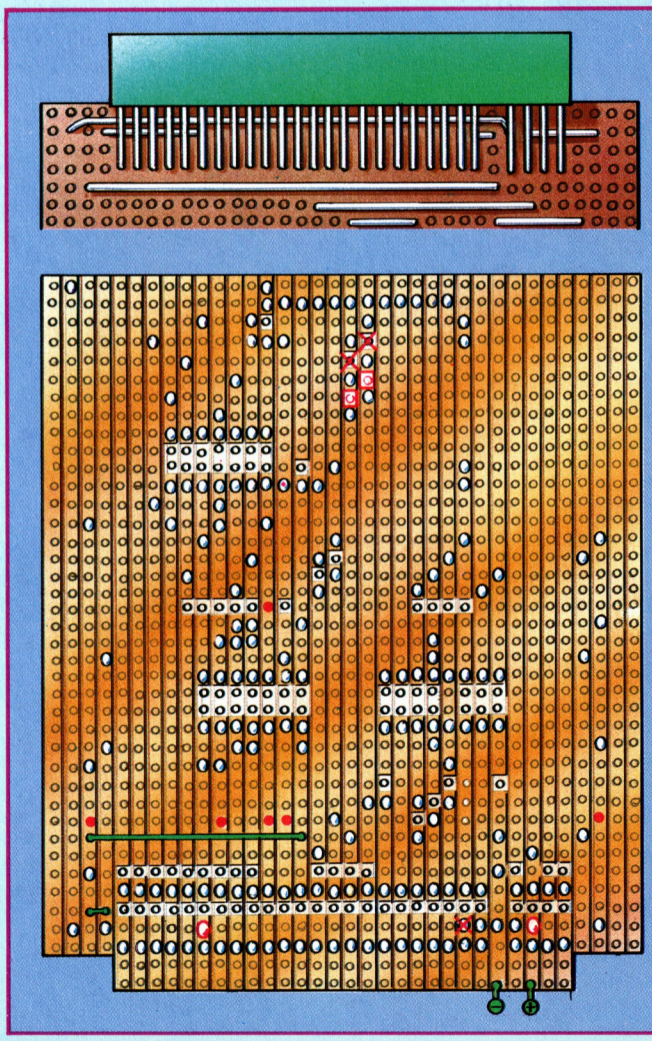
Low-Level Language = Maschinenorientierte Sprache
Eine Programmiersprache stellt zumeist einen Kompromiß zwischen zwei gegensätzlichen Forderungen dar: Einerseits soll sie aus Gründen der Effizienz an der Funktionsweise des Prozessors ausgerichtet, andererseits aber so benutzerfreundlich wie möglich sein. Wenn die Anweisungen eng an den Befehlssatz der CPU angelehnt sind, ist die Bezeichnung „maschinenorientierte“ oder auch „niedere” (Low-Level) Sprache üblich. Dabei ist wegen der Verwandtschaft mit der Kommandostruktur des Prozessors wenig Übersetzungsaufwand erforderlich, wodurch die Arbeit des Rechners entsprechend beschleunigt wird.

Typische maschinenorientierte Sprachen sind neben dem hexadezimalen Maschinencode vor allem die Assemblersprachen. Die Grenze zu den höheren Programmiersprachen ist fließend – beispielsweise lassen FORTH-Anweisungen eigentlich keinen Bezug zum CPU-Befehlssatz erkennen, aber strukturell orientiert sich diese Sprache eng an dem Befehlsablauf des Prozessors.

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.

Bildnachweise
1737–1739: Marcus Wilson-Smith
U2, 1740, 1742, 1743, 1755: Kevin Jones
1744: Caroline Clayton
1747: Mecca Bookmakers
1750, 1751, 1756, 1757, 1760, 1761: Ian McKinnell
1758: Spectrum 1764, U3: Liz Dixon

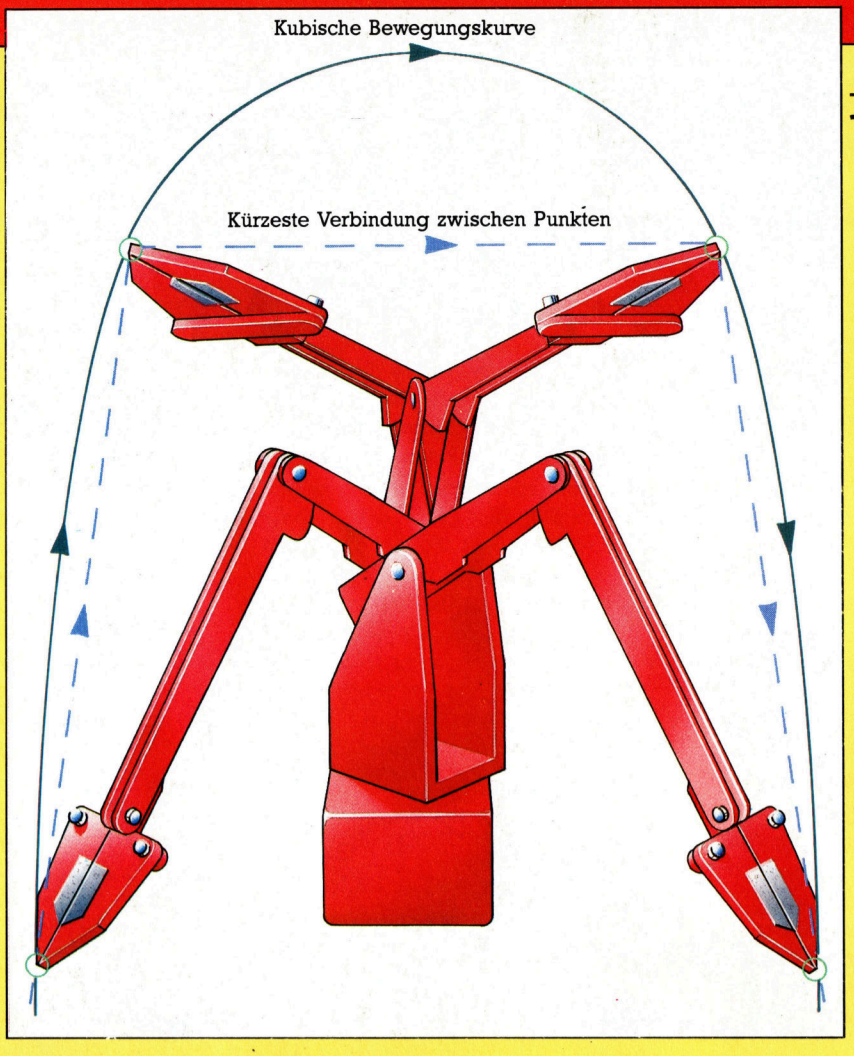
Machine Independent = Maschinenunabhängig
Software wird meist für ein ganz bestimmtes Rechnermodell mit charakteristischen Eigenschaften geschrieben. Soll ein Programm etwa die Grafikfähigkeiten des Apple voll ausschöpfen, dann muß es wegen der Inkompatibilität der Mikrocomputer für den Commodore 64 vollständig neu gestaltet werden. Die Programmierung kann aber (unter Verzicht auf die Nutzung der Eigenheiten eines speziellen Rechners) auch so erfolgen, daß die Software maschinenunabhängig auf einer Vielzahl von Computern lauffähig ist. Solche Programme werden als „portabel“ bezeichnet; die Portabilität ist im allgemeinen nur zwischen Rechnern mit gleichem Betriebssystem (etwa CP/M) gegeben.



Fehlerberichtigung

Beim Platinenlayout für das Spectrum-Interface von Seite 1491 sind uns leider Fehler unterlaufen. Folgendes muß geändert werden: Der lange Draht muß auf seiner rechten Seite um ein Loch weiter nach rechts versetzt werden. Das linke Ende des nächsten Drahtes muß ebenfalls eine Position weiterrücken. Die einzelne Drahtbrücke in der zweiten Reihe muß um ein Loch nach rechts wandern. Das rechte Ende der kurzen Drahtbrücke in der zweiten Reihe links sollte um zwei Löcher nach rechts verlegt werden. Beachten Sie auch die Modifikationen auf der Leiterbahnseite der Platine.

+ Vorschau +



+ Vorschau +

computer kurs

Sinclairs neue Kraft

Der Robot-Arm verlangt schnelle Programme für die Servomotoren.

Aus alt mach neu

Einige Techniken, die fehlende Möglichkeiten und Strukturen von Fortran beheben. Die Sprache wurde in den fünfziger Jahren entwickelt.

Piraten voraus!

Action beim Piratenangriff, Sturm auf hoher See oder einem Ruderbruch.

Wetten, daß . . .?

. . . Spieler sich kaum mit der Wahrscheinlichkeitstheorie bei Wettspielen befassen?

Reden ist Silber

Sprachsynthesizer gibt es für fast alle Heimcomputer. Wir betrachten, was sie uns zu sagen haben.