Einsteigen – Verstehen – Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 81 Ein wöchentliches Sammelwerk

computer kurs

Heft 81

Inhalt

Computer Welt

Tee oder Computer 2266
Der kommerzielle Einsatz des Computers

Computerkunst 2241
Wir stellen einen Computerkünstler vor

Bits und Bytes

Harte Bedingungen 2243
Logigbefehle, Rotation und Verzweigungen

. . . hat man Töne 2256
Programme mit Hintergrundmusik

Sprunghafte Routine 2267
Der Sprung zur Subroutine

BASIC 81

In Zeilen gesperrt 2247
Wie Sie Formeln ins Arbeitsprogramm geben

Überkauft 2262
Wir beschließen die Runde des Spielers

Tips für die Praxis

Die Qual der Wahl 2264
BASIC reicht für die Hobby-Anwender

Rank und schlank 2249
Kompressionsprogramm für den 6502-Prozessor

Software

Das Experten-System HULK 2259
Die schöpferische Begabung der Ingenieure

Jedem das Seine 2253
Die Benutzerschnittstelle nach Wunsch

Hardware

Aquarius 2254
Der kleine Heimcomputer aus Kalifornien

Fachwörter von A—Z

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

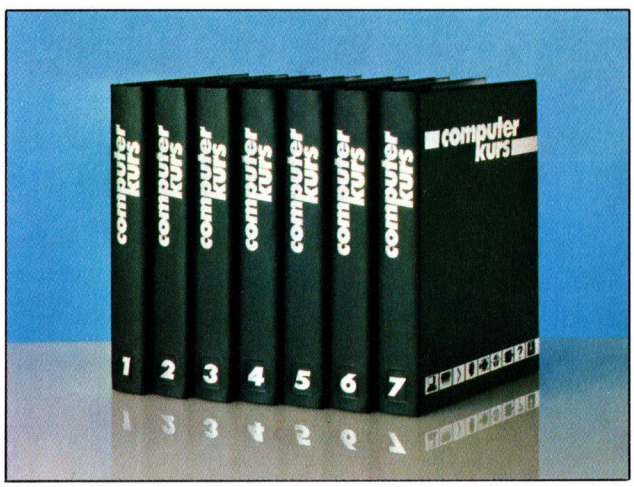
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

Redaktion: Peter Aldick (verantw. f. d. Inhalt), Gudrun Anderson, Joachim Knipp, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

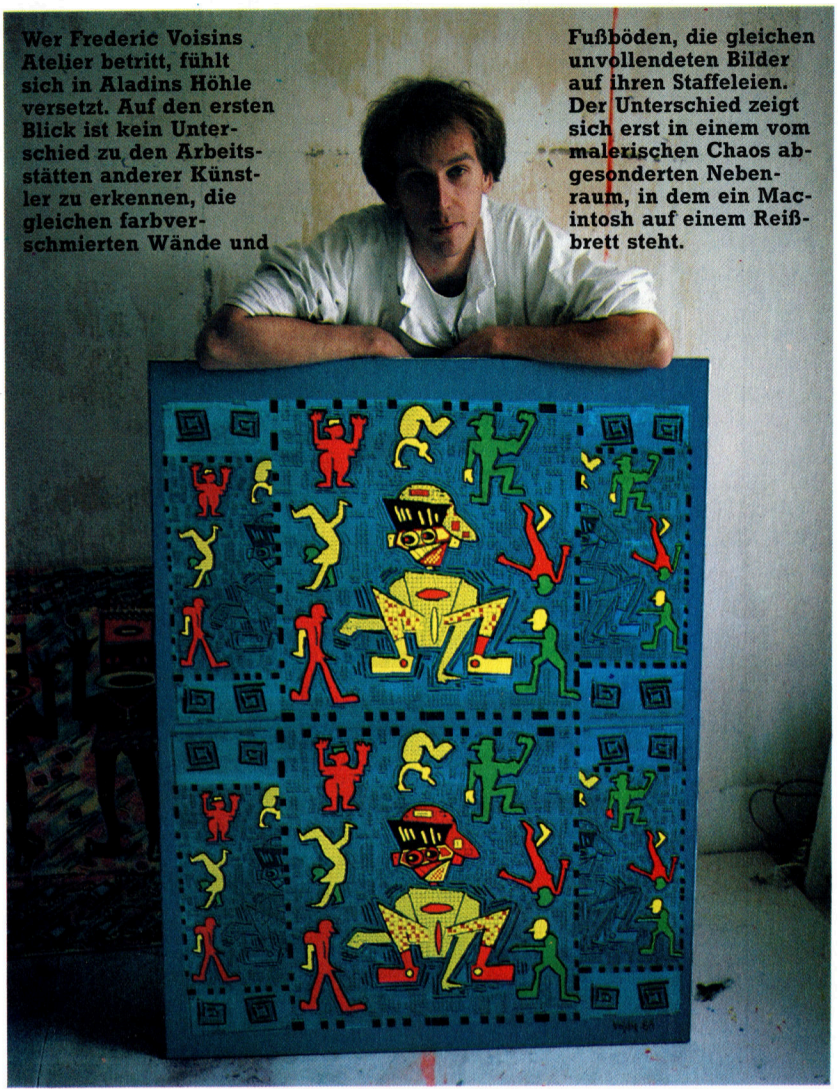
Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



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

Computerkunst

Bisher sind wir hauptsächlich auf Grafikanwendungen wie CAD oder den Entwurf von Zeitungsseiten eingegangen. Heute wollen wir Ihnen einen Künstler vorstellen, dessen Karriere durch die Entwicklung der Apple Mac Paint-Software eine neue Richtung nahm.



Wer Frederic Voisins Atelier betritt, fühlt sich in Aladins Höhle versetzt. Auf den ersten Blick ist kein Unterschied zu den Arbeitsstätten anderer Künstler zu erkennen, die gleichen farbverschmierten Wände und Fußböden, die gleichen unvollendeten Bilder auf ihren Staffeleien. Der Unterschied zeigt sich erst in einem vom malerischen Chaos abgesonderten Nebenraum, in dem ein Macintosh auf einem Reißbrett steht.

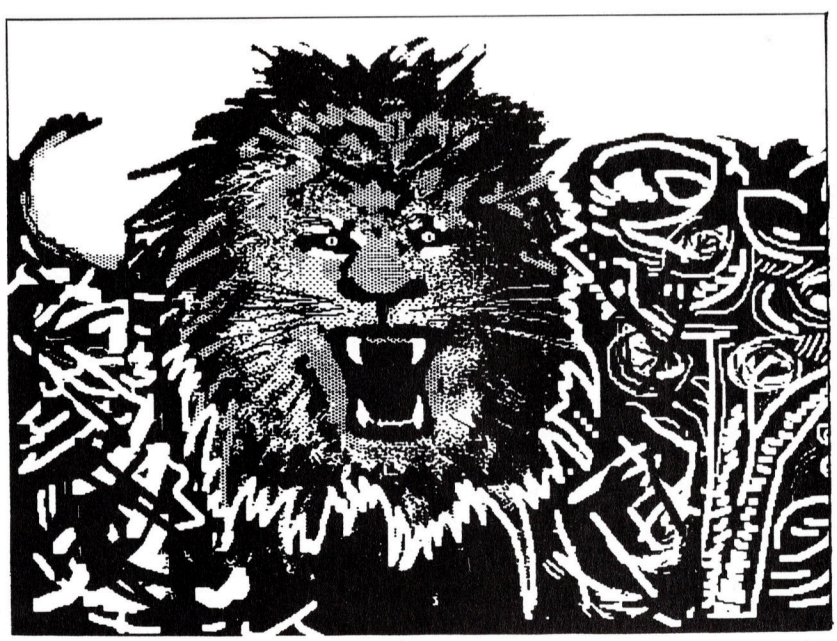
Wie die gesamte Macintosh-Software basiert auch MacPaint auf WIMP: Sobald die Diskette geladen ist, bietet der Rechner verschiedene Mal- und Zeichenmöglichkeiten an. Der Anwender kann zwischen Pinsel und Bleistift wählen. Festgelegte Bereiche werden durch Anwahl von Farbdosen koloriert. Mit der „Spritzpistole" lassen sich auch gepunktete Farbflächen erzeugen. Neben Malen bietet MacPaint auch die Möglichkeit zum Zeichnen geometrischer Figuren und Formen. Alles kann mit einem der 38 Muster oder Farben koloriert werden.

Fehler lassen sich durch Anwahl des Radiergummi-Symbols beseitigen. Außerdem können Bildelemente eingegrenzt und verschoben werden. Das Menü „Goodies" bietet weitere Möglichkeiten: ‚Fat Bits' vergrößert bestimmte Bildteile so weit, daß der Computer-Grafiker einzelne Pixel bearbeiten und kleinste Details löschen oder hinzufügen kann.

Wie die gesamte Macintosh-Software ist auch MacPaint mausgesteuert. Mit Zusatzeinrichtungen wie dem Lichtgriffel-Grafiktablett ‚MacTablet' wird die Arbeit noch angenehmer.

Ein überzeugter MacPaint-Anhänger ist der französische Künstler Frederic Voisin. Vor achtzehn Monaten kam er noch mit dem traditionellen Handwerkszeug aus: Papier, Bleistift, Tinte und Pinsel. Heute benutzt er den Macintosh: Schon nach den ersten auf der Apple Lisa eines Freundes gezeichneten Strichen war er von der Grafik-Fähigkeit des Computers begeistert. Er kam auf die Idee, statt Zeichnungen großformatige Bilder zu machen. Das hatte mit den Besonderheiten der Apple Lisa zu tun: Die Bildschirmdarstellung war schwarzweiß, die Voisin dann nachkolorierte. Zum anderen war das Bild zwar klein, doch die hervorragende Auflösung machte es möglich, die Bilder zu vergrößern.

Der Künstler zögerte nicht lange und beschaffte sich für weitere Experimente einen Apple Macintosh. Interessanterweise führte erst der Computer den Maler an die klassischen Traditionen seiner Kunst heran: die Me-



thode, einen Rohentwurf auf Leinwand nachträglich zu bemalen. Zum Entwurf verwandte Voisin aber keinen Bleistift, sondern einen Ausdruck mit dem Imagewriter, der mit einem Großkopierer auf Formate bis zu drei Meter Höhe vergrößert wurde. Diese Kopie wurde nun mit Kaninchen-Knochenleim auf die Leinwand geklebt; ein Verfahren, das bei Malern seit Jahrhunderten angewendet wird, weil die Elastizität dieses Leims es möglich macht, große Papierbögen glatt aufzuziehen. Voisin koloriert die Kopien mit fluoreszierender Acrylfarbe und schafft so vibrierende Bilder von Robotern, Zulukriegern und Breakdancern. Der Maler beleuchtet seine Bilder gern mit ultraviolettem Licht. Durch die fluoreszierende Farbe entstehen Effekte, die an einen Farbbildschirm erinnern.

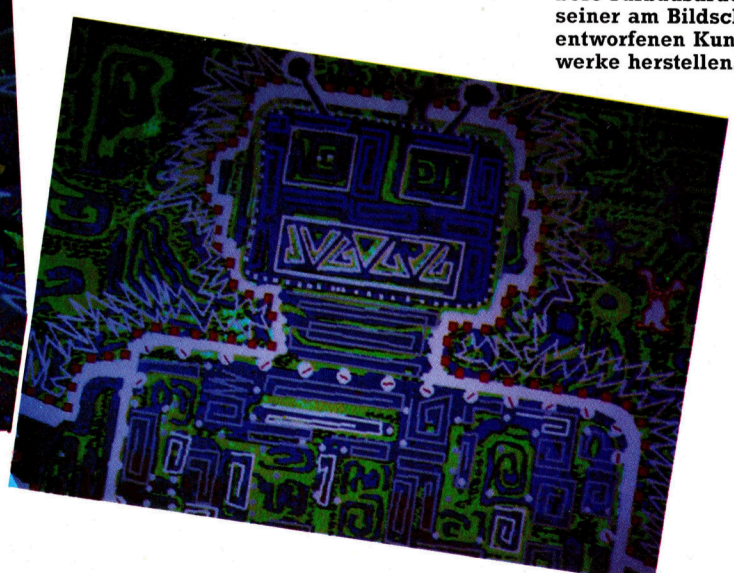
Antiquierte Version

Letztes Jahr „malte" Voisin mit seinem Macintosh 30 Bilder. Er glaubt aber nicht, daß die Leistung von Rechner und Programm damit schon voll ausgenutzt ist. Über die Programmentwickler spricht der Künstler mit Hochachtung, weil sie sich völlig in seine Arbeit hineinversetzt hätten und dadurch seine Bedürfnisse optimal berücksichtigt worden wären. Dabei ist Voisins MacPaint-Version eher antiquiert, viele Verfeinerungen der heutigen Programme sind noch gar nicht implementiert.

Im Gegensatz zu vielen Kollegen schreckt Voisin die Idee der Computer-Kunst nicht ab. Für ihn ist die Maus ebenso wie der Pinsel ein Werkzeug, das mit Kopf und Hand bedient werden will. Der Maler ist der Überzeugung, daß die Computer-Kunst einmal den gleichen Stellenwert einnehmen wird wie beispielsweise der Kupferstich. Der Unterschied liege nur darin, daß das Original auf einer Diskette gespeichert und nicht auf einer Kupferplatte eingraviert sei. Voisin erwartet eine neue Generation von Medienkünstlern: Ihre Produkte würden einem breiten Publikum anstelle einer kleinen Sammlergemeinde zur Verfügung stehen. Das französische Kultusministerium hat ihm inzwischen einen „Radiance" zur Verfügung gestellt, einen Rechner mit mehr Speicher und Farbdrucker. Damit sind bereits Farbdrucke in limitierter Auflage hergestellt worden. Voisin freut sich jetzt auf den Farb-Macintosh, den er für kleinformatige Bilder in seinem Atelier einsetzen möchte.

Der Künstler experimentiert intensiv mit Programmen und Zusatzeinrichtungen. Trotzdem will er sich nicht sein Leben lang nur mit MacPaint beschäftigen. Voisin ist aber sicher, daß er ohne den Macintosh nicht mehr auskommen kann: „Ich brauche ihn, wie ich auch das Telefon oder Pinsel und Farbe brauche. Eine wunderbare Maschine – eine Revolution."

Der Gegensatz zwischen harten Linien, tiefschwarzen Gesichtern und den weicheren Schattierungen des Rauches sind ein gutes Beispiel für den Abwechslungsreichtum, den das Malprogramm MacPaint ermöglicht.



Dies sind Fotografien vom Bildschirm des Radiance-Rechners, den das französische Kultusministerium dem Künstler zur Verfügung gestellt hat. Voisin kann damit sehr saubere Farbausdrucke seiner am Bildschirm entworfenen Kunstwerke herstellen.

Harte Bedingungen

In der letzten Folge hatten wir die Multiplikation und Division des 68000 beschrieben und dabei auch die BCD-Befehle für Addition und Subtraktion behandelt. Wir wenden uns nun den Logikbefehlen zu und untersuchen danach die Anweisungen für Verschiebung, Rotation und Programmverzweigung.

Sehen wir uns zunächst die Logikbefehle des 68000 an. Die Anweisung AND verknüpft den Quelloperanden durch ein logisches AND mit dem Zieloperanden, legt das Ergebnis im Ziel ab und setzt – falls nötig – die Bits N und Z. Es sind viele Adressierungsarten möglich, doch die Operanden müssen Datenregister sein. Weiterhin sind Datenattribute zulässig. Wenn D0 = 1010 1010 und D1 = 1111 0000, dann ist nach dem Befehl

```
AND D1,D0
```

das Datenattribut D0 = 1010 0000.

Unser Beispiel maskiert die niederwertigen vier Bits von D0 aus, da die entsprechenden Bits von D1 auf 0 stehen. Die höherwertigen Bits erscheinen im Ziel, weil die Maskenbits dieser Positionen gesetzt sind. Der Befehl „ANDI" kann den Quelloperand unmittelbar adressieren, dabei sind im Zieloperand datenverändernde Modi möglich. Unser voriges Beispiel sieht mit ANDI so aus:

```
ANDI #$F0,D0
```

Neben OR steht ORI zur Verfügung, für das exklusive OR lassen sich EOR und EORI einsetzen, und auch das logische NOT gibt es. Diese Befehle funktionieren wie AND und sprechen auch die gleichen Bedingungscodes an. Die Adressierung einzelner Bits wird oft für die Steuerung von „Flag"-Bits verwandt.

Für Datenoperanden, die aus einem einzelnen Bit bestehen, bietet der 68000 vier Bitanweisungen, die sich statt der Logikbefehle einsetzen lassen. Die Bitanweisungen testen den Status der angegebenen Bits, die entweder in einem Langwort (Datenregister) von 0 bis 31 durchnumeriert sind oder sich in einem Speicherbyte befinden. Die Anweisungen setzen je nach dem Status des Bits das Z-Bit des SR. Z ist damit ein umgekehrter Ein-Bit-Speicher des angegebenen Bits. Bei D0 = XXXX XXXX XXX1 0000 (Binärformat – X ist 0 oder 1) testet

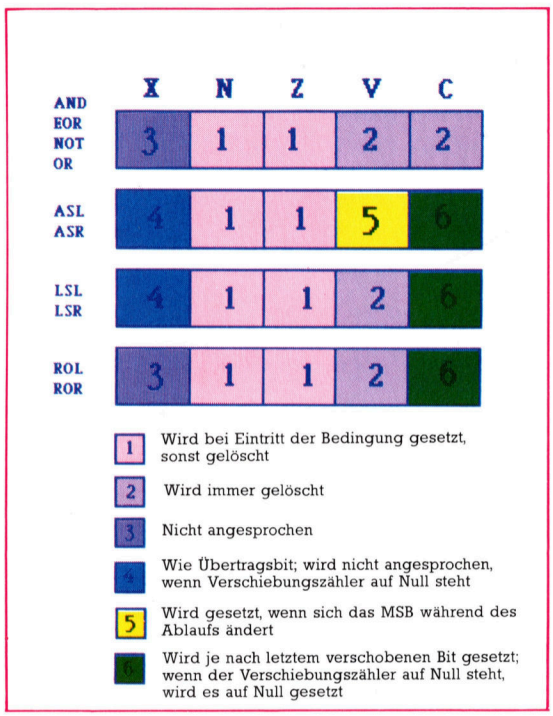
```
BTST #4,D0
```

das Bit vier und setzt 2 auf Null, wenn das entsprechende Bit nicht auf Null steht:

```
BTST #3,DO
```

setzt Z in diesem Fall auf Eins. Die anderen Bedingungscodes werden nicht verändert.

Folgende Befehle ändern das getestete Bit:
BSET setzt das Bit nach dem Test
BCLR löscht das Bit nach dem Test



Die logische Verschiebung stellt je nach Verschiebungsrichtung Nullen in Bit 0 oder 15. Während LSL wird Bit V auf Null gesetzt, während nach LSR Bit N auf Null steht. Beide Abläufe stellen das letzte verschobene Bit in C und X.

BCHG ändert das Bit nach dem Test
Wenn Sie mit den obenstehenden zahlreichen Beispieldaten

```
BCHG #4,D0
```

ausführen und D0 auf 1 gesetzt ist, erhalten Sie D0 = XXXX XXXX XXX0 0000, wobei Z auf Null gesetzt wird und damit den Zustand vor der Änderung anzeigt.

Beachten Sie, daß alle Befehle für das bitweise Testen und Setzen nur aus einer Anweisung bestehen. Dies kann besonders bei Mehrplatzanwendungen wichtig sein, in denen ein Interrupt zwischen Testen und Setzen zu unvorhersehbaren Ergebnissen führen kann. Logikbefehle haben den Vorteil, daß Tests keine Abläufe auslösen und lassen sich daher problemlos für die Bitverarbeitung einsetzen.

Sehen wir uns zunächst einmal an, was passiert, wenn ein Bitmuster nach links oder rechts verschoben wird. Wenn D0 = 0000 0000 0000 1000, dann erhalten wir bei einer dreifachen Rechtsverschiebung 0000 0000 0000 0001. Da der Inhalt von D0 dabei durch acht (2^3) geteilt wurde, entspricht die Rechtsverschiebung um eine Position einer Teilung durch Zwei. Umgekehrt ist die Linksverschiebung dementsprechend eine Multiplikation.

Bei der Rechtsverschiebung haben wir vorausgesetzt, daß auf der linken Seite (bei der Linksverschiebung auf der rechten Seite) automatisch Nullen eingesetzt werden. Wenn das Vorzeichen der Zahl erhalten bleiben soll, muß dies bei einer Rechtsverschiebung berücksichtigt werden. Bei D0 = 1111 1111 1111 0000 (dezimal −16) ergibt eine dreifache Rechtsverschiebung D0 = 1111 1111 1111 1110 (dezimal −2). In diesem Fall wurden links Einsen eingesetzt, um die negative Zahl zu erhalten. Im allgemeinen setzt die Rechtsverschiebung Bits ein, die dem Vorzeichenbit (dem höchstwertigen Bit des Datenoperanden) entsprechen, während die Linksverschiebung Nullen in das niederwertige Bit des Wortes stellt. Verschiebungen, die das Vorzeichen des Operanden beibehalten, heißen „arithmetische Verschiebungen".

Bei der arithmetischen Rechtsverschiebung (ASR) wird Bit 15 in Bit 14 kopiert, der Inhalt von Bit 15 aber nicht verändert. Auf diese Weise bleibt das Vorzeichenbit erhalten. Bit 0 wird in Bit C und X des Statusregisters kopiert. Bei der Linksverschiebung werden Nullen in Bit 0 gestellt, und Bit V wird gesetzt, wenn sich der Inhalt von Bit 15 ändert.

Der 68000 bietet die Befehle ASL (arithmetische Linksverschiebung) und ASR (arithmetische Rechtsverschiebung). Wenn das Ziel ein Datenregister ist, können Sie bis zu acht Bits unmittelbar verschieben, bei einem Langwort wird ein anderes Datenregister als Zähler eingesetzt und so bis zu 31 Bits verschoben. So ist

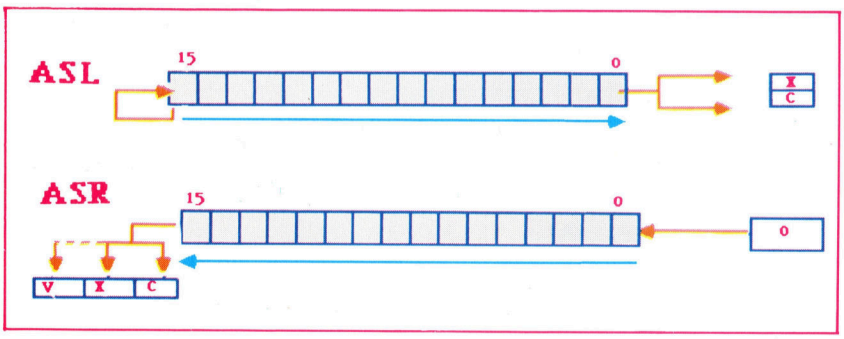
```
ASR #8,D0
```

die größte Verschiebung, während

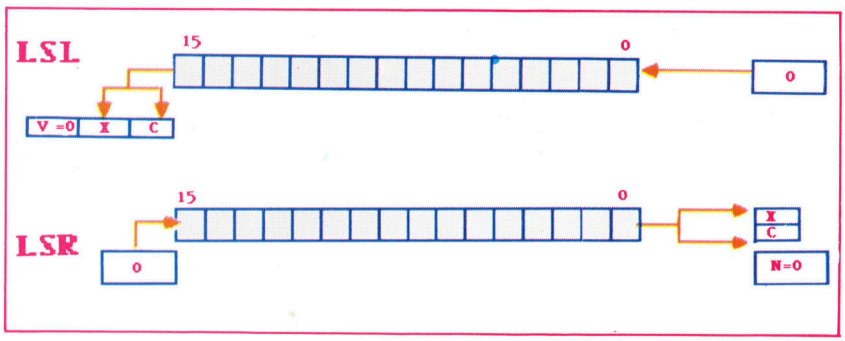
```
ASL D1,D0
```

D0 um soviel Positionen nach links verschieben kann, wie D1 Stellen hat (d. h. bis zu 31). Speicherstellen lassen sich jedoch nur um eine Position verschieben.

Das Bild zeigt, daß alle Bedingungscodes des SR angesprochen sind: C und X werden auf das letzte aus dem Operanden geschobene Bit gesetzt, V zeigt bei Linksverschiebungen die Veränderung des Vorzeichenbits an, während N und Z je nach Ergebnis gesetzt werden.



Die Befehle für Logik, Verschiebung und Rotation beeinflussen den Inhalt des Statusregisters. Die arithmetische Verschiebung unterscheidet sich von der logischen nur durch den Gebrauch des V-Flags, das mögliche Veränderungen im Vorzeichenbit des Operanden anzeigt.

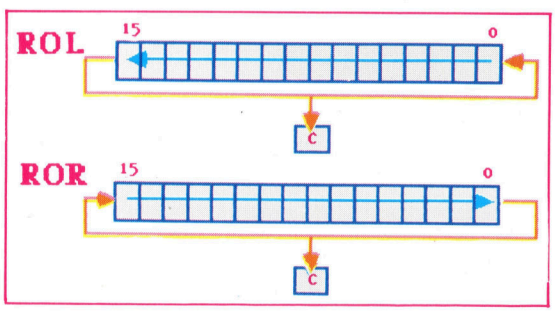


MULT-Vorgänge

Sie werden sich fragen, warum arithmetische Verschiebungen so wichtig sind, da der 68000 doch ausreichend mit Multiplikation- und Divisionsbefehlen ausgerüstet ist. Der Grund ist die Ausführzeit. Ein MULT-Vorgang braucht 70 Taktzyklen, während ASL oder ASR nur 6+2n Zyklen dauert (n ist die Zahl der Verschiebungen). Die Dauer einer arithmetischen Verschiebung reicht von acht Zyklen für die Verschiebung einer Position bis zu 68 Zyklen für alle 31 Positionen: Eine geringe Zahl von Verschiebungen ist daher weit schneller als Multiplikations- oder Divisionsbefehle – im Extremfall ist eine einfache Rechtsverschiebung um das Neunzehnfache schneller als die Teilung durch 2.

Es gibt aber auch die „logischen Verschiebungen" LSR und LSL, die das Vorzeichen des Datenoperanden nicht berücksichtigen und in den freigewordenen Stellen Nullen einsetzen. Auch hier gelten die Adreßbeschränkungen für arithmetische Verschiebungen. Das Bit V wird immer auf Null gesetzt. Logische Verschiebungen werden oft für das Setzen oder Testen von Datenoperanden eingesetzt, die kleinere Untergruppen oder Felder enthalten.

Die ROTATE-Befehle lassen den Operanden kreisen, indem sie bei ROR Bit 0 in Bit 15 stellen und umgekehrt (bei ROL). Das „übergestellte" Bit wird dabei in das C-Bit des Statusregisters kopiert.



Schließlich bietet der 68000 noch eine Gruppe von Rotationsbefehlen, die eigentlich Relikte aus einer Vergangenheit sind, in der die einzelnen Bits des Datenoperanden über die Rotation durch das Bedingungsflag getestet wurden. Im wesentlichen hat das breite Spektrum der Bitbefehle diese Aufgabe übernommen, wobei die neuen Befehle die gleiche Zeit brauchen wie die alten. Wir wollen daher nur ein Beispiel für die Rotation anführen. ROR

#3,D0 rotiert den Inhalt von D0 um drei Positionen nach rechts, setzt das Bit C auf das letzte verschobene Bit und bringt das niederwertige Bit an die höchstwertige Stelle.

Die Befehle zur Programmsteuerung legen die Ablaufreihenfolge der Anweisungen fest. So verändert die Gruppe der „bedingten Verzweigungen" je nach Eintreten bestimmter Bedingungen den normalen sequentiellen Befehlsfluß. Die „unbedingte Verzweigung" löst dagegen in jedem Fall eine Verzweigung im normalen Programmfluß aus.

Eine unbedingte Verzweigung sieht normalerweise so aus:

```
BRA NEULABEL
```

Hier setzt sich der Befehlsfluß nach Ausführung von BRA von der Speicherstelle NEULABEL aus fort. Wenn der Distanzwert in einem vorzeichenbehafteten Acht-Bit-Wort untergebracht werden kann, läßt sich die gesamte Anweisung in einem Computerwort codieren. Die Hälfte des Wortes enthält dann den Op-code für BRA (hex 60), während das andere Byte den Distanzwert mit Vorzeichen angibt. Wenn der Distanzwert zu groß wird oder die Verzweigungsadresse noch nicht bekannt ist (der Assembler kann sie nicht berechnen, da er nur einen Vorwärtsbezug erhalten hat), steht das Distanzbyte auf Null, während das nächste Wort den vollen Distanzwert im 16-Bit-Format mit Vorzeichen angibt. Wir sehen uns diesen Vorgang später genauer an.

Für die meisten Situationen reicht BRA völlig aus. Manche Situationen – besonders berechnete Verzweigungsadressen – brauchen jedoch einen flexibleren Befehl. Nehmen Sie an, Sie möchten auf eine Tabellenadresse verzweigen, deren Index sich in einem Register befindet (die Adressierungsart „indirekt mit Index und Distanzwert"). Da mit BRA diese Art der Adreßberechnung nicht möglich ist, muß hier der Sprungbefehl JMP verwandt werden, der die Verzweigungsadresse gegebenenfalls berechnen kann.

Sehen wir uns die verschiedenen Spielarten der unbedingten Verzweigungen einmal an dem Programmbeispiel (‚Viele Verzweigungen') genauer an.

„PC relativ"

Die Sprungbefehle dieses Listings zeigen die absolute Adressierung (vorwärts und rückwärts) und die indirekte Adressierung mit Index und Distanzwert. Weitere Adressiermethoden sind „einfach indirekt" – zum Beispiel (A2) – und „PC relativ".

Die BRA-Befehle zeigen zunächst einen Distanzwert, der im Befehlswort enthalten ist, und dann zwei Anweisungen mit vollen Worterweiterungen (obwohl für den Distanzwert auch ein Byte genügt hätte). Da dem Assembler die Adresse von FINISH noch nicht bekannt war, mußte er für den Distanzwert eine volle Erweiterung einsetzen. Wenn Sie wissen, daß dieser Wert in ein vorzeichenbehaftetes Byte paßt, können Sie den Assembler mit dem Zusatz .S zwingen, die Kurzform zu nehmen. Unser Beispiel ließe sich daher auch kürzer als BRA.S FINISH schreiben.

Die bedingten Verzweigungen lassen sich in drei Hauptgruppen unterteilen:

- Verzweigungen im Zweierkomplement
- Verzweigungen ohne Vorzeichen
- Schleifensteuerung

Viele Verzweigungen

```
                    WAYOFF  EQU     $3FFF
                            ORG     $1000
100     3200        START   MOVE    D0,D1
1002    4EF8 1000           JMP     START     * Volle Worterweiterung
1006    4EF8 101C           JMP     FINISH    * noch ein Sprung nach vorn
100A    4EE8 0005           JMP     5(A0)     * indirekt mit Adreßdistanzwert
100E    4EF2 000C           JMP     12(A2,D0) * indirekt mit Index und
                                                Adreßdistanzwert

  * Spielarten der unbedingten Verzweigung
  * Wenn der Adreßdistanzwert aus einem Byte besteht, kann
    er im Op-Code-Wort untergebracht werden.

1012    60EC                BRA START         * Negativer Adreßdistanzwert
1014    6000 0004           BRA FINISH        * Vorwärtssprung – genaue
                                                Adresse noch unbekannt
1018    6000 2FE5           BRA WAYOFF        * Computerwort mit einem
                                                Distanzwert im 16-Bit-Format
101C    3200        FINISH  MOVE D0,D1
```

Das Befehlsformat der Gruppen ist identisch:

```
Bcc LABEL
```

cc bezieht sich auf die getesteten Bedingungscodes. Ist die Bedingung wahr, dann findet die Verzweigung auf LABEL statt, falls nicht, wird dann lediglich der nächste sequentielle Befehl angesprochen.

Die Spalte „Wahr, wenn –" zeigt die arithmetische Bedingung an, die als Ergebnis der Vergleichsbefehle CMP oder SUB entsteht. Bei der ersten Bedingungsgruppe der Tabelle (Verzweigungen im Zweierkomplement) wird das Bit V (Überlauf) in das logische Testen mit eingeschlossen. Dieses Merkmal bestimmt die Gruppenzugehörigkeit dieser Befehle und zeigt außerdem an, daß für die korrekte Ausführung auch das Überlaufbit getestet werden muß. Die genauen logischen Bedingungen der Vezweigungen sollten Sie dem Anwenderhandbuch von Motorola entnehmen. Eine BGE-Verzweigung testet beispielsweise, ob N=V. Der Sprung wird ausgelöst, wenn NOT N AND NOT V wahr ist (d. h. es existiert kein Überlauf, und die Zahl ist nicht negativ) oder N und V sind wahr (Überlauf und negative Zahl).

Ein Beispiel: Wir wollen die beiden vorzeichenbehafteten Zahlen D1 und D2 miteinander vergleichen und auf GROESSER verzweigen, wenn D2 größer ist als D1. Die Bedingung wird dabei mit einem Vergleichsbefehl getestet, auf

den die bedingte Verzweigung BGT folgt:

```
CMP D1,D2          * ergibt D2 — D1
BGT D2GROESSER     * verzweige auf
                     D2GROESSER,
                   * wenn nicht Null
                     und nicht
                   * negativ und kein
                     Überlauf
```

Die zweite Gruppe der bedingten Verzweigungen (in der Tabelle) arbeitet mit Zahlen ohne Vorzeichen und testet die Bedingungscodes auf einfachere Weise. So kann beispielsweise der Vergleich des Inhalts der Speicherstelle BETTY mit D1 unabhängig von der Überlaufbedingung ausgeführt werden:

```
CMP BETTY,D1    * ergibt D1 — BETTY
BEQ GLEICH      * verzweige auf GLEICH
                  wenn Z=1
```

Assemblerversion

Weitere Beispiele der bedingten Verzweigung ergeben sich aus der Assemblercodierung von Schleifen. So sieht der Ablauf

```
FOR I := 1 TO 5 D0
  (Programmteil fünfmal ausführen)
NEXT I
```

in der Assemblerversion so aus:

```
   MOVEQ #5,D7   * Schleifenzähler setzen
LOOP (Programmteil fünfmal ausführen)
   SUBQ #1,D7    * Zähler dekrementieren
   BNE LOOP      * Wiederholen bis
                   D7 = 0
```

Mit „Quick"-Befehlen läßt sich die Schleife in nur drei Worten codieren.

Die dritte Gruppe der bedingten Verzweigung bildet der Befehl DBcc, der dekrementiert und bei Eintritt der Bedingung auf cc verzweigt. Dieser Befehl ist eine Erweiterung des oben angeführten Schleifensteuermoduls, enthält in seinem Code aber bereits Dekrementierung und bedingte Verzweigung. DBcc ähnelt dem PASCAL-ähnlichen Pseudo Code für eine REPEAT-Schleife:

```
REPEAT
  (. . . Schleifeninhalt . . .)
UNTIL
  'C' = wahr oder Dn = —1
```

cc ist eine der Bedingungen (im zweiten Teil der Tabelle dargestellt) und Dn das Datenregister mit dem Schleifenzähler. Das vorige Beispiel wird mit DBcc so codiert:

```
   MOVEQ #5,D1   * Schleifenzähler setzen
LOOP (Programmteil fünfmal ausführen)
   DEBQ D1,LOOP  * Ende, wenn das zuletzt
                 * getestete 'cc' 0 ist
                 * oder D1=1
                   (6Durchläufe)
```

Beachten Sie den Unterschied in den Endbedingungen und auch die Unterschiede zu normalen BEQ-Befehlen. Wenn kein bedingtes Testen nötig ist, dann läßt sich DBcc mit einem auf F (falsch) gesetzten cc für einfache FOR-Schleifen verwenden, beispielsweise:

```
   MOVEQ #4,D3
LOOP (. . . Inhalt der FOR-Schleife . . .)
   DBF D3,LOOP
```

Diese Version entspricht der ursprünglichen FOR-Schleife mit fünf Wiederholungen und belegt auch die gleiche Menge Speicher.

Verzweigungsbedingungen

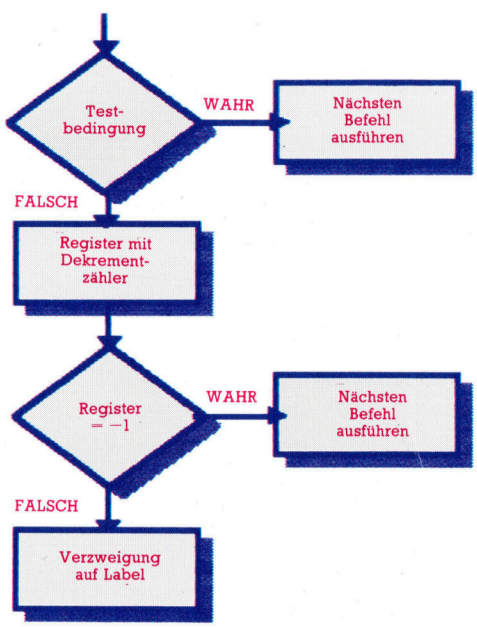
Vor der Verzweigung auf eine Adresse, die mit der Arithmetik im Zweierkomplement berechnet wird, werden die folgenden Bedingungen getestet:

cc Bedingung	Wahr, wenn
GT größer als	Ziel > Quelle
LT kleiner als	Ziel < Quelle
GE größer als oder gleich	Ziel >= Quelle
LE kleiner als oder gleich	Ziel <= Quelle
VS Überlauf	V = 1
VC kein Überlauf	V = 0

Vor der Verzweigung auf eine Adresse, die mit der Ganzzahlenarithmetik ohne Vorzeichen berechnet wird, werden die folgenden Bedingungen getestet:

EQ gleich Null	Z = 1
NE ungleich Null	Z = 0
MI minus	N = 1
PL plus	N = 0
HI höher als	C = Z = 0
LS niedriger als oder gleich	C oder Z = 1
CS Übertrag gesetzt	C = 1
CC Übertrag gelöscht	C = 0

Unter einer Bedingung



Der Befehl DBcc bietet dem Programmierer des 68000 eine Fülle von Möglichkeiten auf hoher Ebene. Das Ablaufdiagramm zeigt die Ausführung des Befehls. Beachten Sie, daß die Anweisung nicht nur ein Zählerregister dekrementiert, sondern auch eine Bedingung testet. Für eine feste Zahl von Wiederholungen (vergleichbar mit der FOR. . .NEXT-Schleife in BASIC) können Sie das Testen der Bedingung mit dem bedingten FALSCH (DBF) abstellen.

In Zellen gesperrt

Nachdem wir die Grafik- und Formelaufbereitungs-Routinen abgeschlossen haben, kommen wir nun zu den Unterprogrammen zur Eingabe von Formeln und Daten in das Arbeitsblatt und zu deren Berechnung.

Der Programmteil, der die Eingabe von Formeln kontrolliert, liegt in den Zeilen 2000 bis 2050. Der INPUT-Befehl übernimmt die Formeln und legt sie in D$ ab. Das Programm kopiert die Eingabe anschließend in das Zeichenfeld F$() und löscht den entsprechenden Eintrag im Feld PS$.

Die Inhalte der Zellen A1 bis A15 sind in F$(1) bis F$(15) gespeichert, die Zellen B1 bis B15 in F$(16) bis F$(30) und so weiter. Ein einzelnes Element in F$() kann daher jederzeit aus den Koordinaten des Cursors mit der Formel (Y-1)*15+X ermittelt werden.

Das Programm übernimmt die Daten zeichenweise in Zeile 2120. Falls eine der Cursor-Tasten gedrückt wurde, wird die Routine abgebrochen und die bisher eingegebenen numerischen Daten in einer Feldvariablen (M(,)) abgelegt. Die Zeilen 2140 bis 2170 überprüfen, ob die gedrückte Taste numerisch war und legen dann das Zeichen in E$ ab.

Die erste Routine arbeitet das Zeichenfeld F$() und die entsprechenden Elemente in PS$() ab und übersetzt jede neu eingegebene Formel in UPN durch Aufruf der Subroutine bei 4000. Wenn sich in der jeweiligen Zelle ein Eintrag befindet, wird die Kalkulationsroutine aufgerufen.

Konditionsbeschreibung

Die Kalkulationsroutine benutzt das UPN-Prüfprogramm ab Zeile 4700, das die Elemente des bearbeitenden UPN-Ausdrucks in das Zeichenfeld G$() legt. Diese Elemente sind entweder Operatoren (wie + und −) oder Operanden (A1, B5,3 usw.). Die Kalkulationsroutine arbeitet dann mit den Elementen des UPN-Ausdrucks unter Berücksichtigung der folgenden Konditionen:

- Ist das Element eine Konstante, wird der Wert in die Feldvariable C() aufgenommen und seine Position in C() auf den Stapel ST() geschoben.
- Ist das Element eine Zelladresse, wird der zugehörige Wert aus der Feldvariablen M(,) in C() kopiert und die Position in C() auf den Stapel geschoben.
- Ist das Element ein Operator, wird die Operation mit den beiden letzten (oder im Falle eines negativen Vorzeichens mit dem letzten) auf den Stapel gebrachten Operanden ausgeführt. Der Stapel enthält nur die Positionen der Operanden in C(), und die Werte dienen uns lediglich zum Auffinden der richtigen Elemente in C(). Das Resultat wird in C() abgelegt, die beiden Operanden werden vom Stapel genommen, und die Position des Resultates wird in C() auf den Stapel gelegt.

BASIC-Dialekte

Schneider CPC:
Beachten Sie folgende Änderungen gegenüber der Commodore-64-Version:

```
2010 LOCATE 1,22
2020 PRINT "NEW FORMULA              ";CHR$(11)
2103 LOCATE 1,22
2120 A$="":WHILE A$="":A$=INKEY$:WEND
2130 IF A$=CHR$(243)OR A$=CHR$(242) OR
     A$=CHR$(241) OR A$=CHR$(240) THEN
     2250
2180 LOCATE H(X+1-H1) -1,V(Y-V1+1)
2190 PRINT CHR$(24) ;SPACE$((5) ;CHR$(11)
2200 LOCATE H(X+1-H1) +4-LEN(E$), V(Y-V1+1):
     PRINT E$;CHR$(24)
2220 LOCATE 1,22
2305 LOCATE 1,22
2315 Q$=CHR$ (J+64) +MID$ (STR$(I), 2,2):
     LOCATE 1,1:PRINT "CELL:";Q$;" "
```

Acorn B:
Beachten Sie folgende Änderungen gegenüber der Commodore-64-Version:

```
2010 PRINT TAB (0,22);
2103 PRINT TAB (0,22);
2120 A$=get$
2130 IF A$=CHR$(136) OR A$=CHR$(137) OR
     A$=CHR(138) OR A$=CHR$(139) THEN 2250
2180 COLOUR 2:COLOUR 129
2190 PRINT TAB(H(X+1-H1) -1,V(Y-V1+1)-1);"    "
2200 PRINT TAB(H(X+1-H1) +4-LEN(E$),
     V(Y-V1+1)-1);E$
2205 COLOUR 1:COLOUR 128
2220 PRINT TAB(0,22);
2230 I$=GET$
2240 PRINT TAB(0,22);
2305 PRINT TAB(0,22);
2315 Q$=CHR$ (J+64) +MID$(STR$(I) ,2,2):PRINT
     TAB (0,0);"CELL:" ;Q$;"  "
```

Eingabe- und Kalkulationsroutinen

Commodore 64:

```
1170 IF A$>="0" AND A$=<"9" THEN GOSUB 2
100:REM ENTER NUMERIC DATA
2000 REM *** INPUT FORMULA ROUTINE ***
2010 GOSUB 1950:REM MOVE CURSOR TO INPUT
 LINE
2020 PRINT "NEW FORMULA:
          ";CU$
2030 INPUT "NEW FORMULA:";D$
2040 LET F$((Y-1)*15+X)=D$:LET PS$((Y-1)
*15+X)=""
2050 GOSUB 1900:RETURN
2100 REM *****  ENTER DATA IN CELL *****
2103 GOSUB 1950:REM MOVE CURSOR TO INPUT
 LINE
2104 PRINT "            ENTERING -DATA
          "
2105 LET P=0:LET E$=""
2110 IF A$<>"" THEN 2150
2120 GET A$:IF A$="" THEN 2120
2130 IF A$=CHR$(29) OR A$=CHR$(157) OR A
$=CHR$(17) OR A$=CHR$(145) THEN 2250
2135 IF A$=" " THEN 2250
2140 IF A$="." THEN 2160
2150 IF A$<"0" OR A$>"9" THEN 2120
2160 LET P=P+1:IF P>5 THEN 2220
2170 LET E$=E$+A$
2180 PRINT CO$;:FOR C=1 TO V(Y+1-H1)-1:P
RINT CD$;:NEXT C
2190 PRINT TAB(H(X+1-H1)-1);CHR$(18);"
   ";CU$
2200 PRINT TAB(H(X+1-H1)+4-LEN(E$));CHR$
(18);E$
2210 GOTO 2120
2220 GOSUB 1950:REM MOVE CURSOR TO INPUT
 LINE
2225 PRINT "       ERROR - INPUT IGNORED"
2230 GET I$:IF I$="" THEN 2230
2240 PRINT CO$;:FOR K=1 TO 22:PRINT CD$;
:NEXT K
2245 PRINT "
":GOTO 2120
2250 LET M(Y,X)=VAL(E$):GOSUB 1900:RETUR
N
2300 REM ****** CALCULATE THE SHEET ****
***
2305 GOSUB 1950:REM MOVE CURSOR TO INPUT
LINE
2306 PRINT "       CALCULATING  -  PLEASE
 WAIT       "
2310 FOR J=1 TO 15:FOR I=1 TO 15
2315 Q$=CHR$(J+64)+MID$(STR$(I),2,2):PRI
NT CO$;CD$;"CELL:";Q$;" "
2320 LET P$=PS$((J-1)*15+I)
2325 IF P$<>"" THEN GOSUB 4700:GOSUB 237
0:GOTO 2350
2330 LET C$=F$((J-1)*15+I)
2335 IF C$="" THEN 2350
2340 GOSUB 4000:GOSUB 2370:REM       DECO
DE FORMULA
2350 NEXT I,J:GOSUB 1700:RETURN
2370 REM *****  EVALUATE FORMULA  ******
2375 SP=0:FOR K=1 TO 20:ST(SP)=0:NEXT K
2380 FOR K=1 TO CP
2390 LET T$=G$(K)
2400 IF T$="+" THEN 2500
2405 IF T$="-" THEN 2510
2410 IF T$="*" THEN 2520
2415 IF T$="/" THEN 2530
2420 IF T$="^" THEN 2540
2425 IF T$="&" THEN 2550
2430 IF T$="" THEN 2450
2432 TE$=MID$(T$,1,1)
2435 IF (TE$>="0" AND TE$<="9") OR TE$="
." THEN C(K)=VAL(T$):GOTO 2445
2440 LET C(K)=M(ASC(MID$(T$,1,1))-64,VAL
(MID$(T$,2,2)))
2445 SP=SP+1:ST(SP)=K
2450 NEXT K
2460 LET M(J,I)=C(K-1):RETURN
2500 C(K)=C(ST(SP-1))+C(ST(SP)):ST(SP)=0
:SP=SP-1:ST(SP)=K:GOTO 2450
2510 C(K)=C(ST(SP-1))-C(ST(SP)):ST(SP)=0
:SP=SP-1:ST(SP)=K:GOTO 2450
2520 C(K)=C(ST(SP-1))*C(ST(SP)):ST(SP)=0
:SP=SP-1:ST(SP)=K:GOTO 2450
2530 C(K)=C(ST(SP-1))/C(ST(SP)):ST(SP)=0
:SP=SP-1:ST(SP)=K:GOTO 2450
2540 C(K)=C(ST(SP-1))^C(ST(SP)):ST(SP)=0
:SP=SP-1:ST(SP)=K:GOTO 2450
2550 C(K)=0-C(ST(SP)):ST(SP)=K:GOTO 2450
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20)
,G$(20),C(20)
```

Sinclair Spectrum:

```
2000>REM ************************
2001 REM * INPUT FORMULAE       *
2002 REM ************************
2010 PRINT AT 18,0;"       ENTER
NEW FORMULA       "
2020 INPUT  LINE D$
2030 LET F$((Y-1)*15+X,1 TO LEN
D$)=D$: LET H$((Y-1)*15+X,1 TO )
=""
2050 GO SUB 1900
2060 RETURN
2100 REM ************************
2101 REM * ENTER DATA IN CELL   *
2102 REM ************************
2105 PRINT AT 18,0;"       ENTERI
NG - DATA          "
2110 LET P=0: LET B$=""
2120 LET A$=INKEY$: IF A$="" THE
N  GO TO 2120
2130  IF A$=CHR$ (13) THEN  GO S
UB 1650: GO TO 2250
2140 IF A$="." THEN  GO TO 2160
2150 IF A$<"0" OR A$>"9" THEN  G
O TO 2120
2160 LET P=P+1: IF P>5 THEN  GO
TO 2220
2170 LET B$=B$+A$
2180 PRINT AT V(Y+1-V1),H(X+1-H1
);"     "
2190 PRINT AT V(Y+1-V1),H(X+1-H1
)+5-LEN B$;B$
2210 GO TO 2120
2220 PRINT AT 18,0;"     ERROR -
INPUT IGNORED       "
2230 LET I$=INKEY$: IF I$="" THE
N  GO TO 2230
2240 PRINT AT 18,0;"
                    ": GO TO 211
2250 LET M(Y,X)=VAL (B$): GO SUB
 1900
2260 RETURN
2300 REM ************************
2301 REM * CALCULATE THE SHEET *
2302 REM ************************
2305 PRINT AT 18,0;"      CALCULA
TING - PLEASE WAIT   "
2310 FOR J=1 TO 15: FOR I=1 TO 15
2315 PRINT AT 0,0;"CELL:";CHR$ (
J+64);STR$ (I);" "
2320 LET P$=H$((J-1)*15+I,1 TO )
2325 IF P$(1)<>" " THEN  GO SUB
2355: GO SUB 4700: GO SUB 2370:
GO TO 2350
2330 LET C$=F$((J-1)*15+I,1 TO )
2335 IF C$(1)=" " THEN  GO TO 2350
2340 GO SUB 4000: GO SUB 2370
2350 NEXT I: NEXT J: GO SUB 1700
: RETURN
2355 FOR Z=1 TO LEN P$: IF P$(Z)
=" " THEN  GO TO 2357
2356 NEXT Z
2357 LET P$=P$(1 TO Z-1): RETURN
2370 REM ************************
2371 REM * DECODE FORMULA      *
2372 REM ************************
2375 LET SP=0: DIM S(20)
2380 FOR K=1 TO CP
2390 LET T$=G$(K)
2400 IF T$(1)="+" THEN  GO TO 2500
2405 IF T$(1)="-" THEN  GO TO 2510
2410 IF T$(1)="*" THEN  GO TO 2520
2415 IF T$(1)="/" THEN  GO TO 2530
2420 IF T$(1)="^" THEN  GO TO 2540
2425 IF T$(1)="&" THEN  GO TO 2550
2430 IF T$(1)="" THEN  GO TO 2450
2432 LET U$=T$(1)
2435 IF (U$)="0" AND U$(="9") OR
 U$="." THEN  LET C(K)=VAL (T$):
 GO TO 2445
2440 LET C(K)=M(CODE (T$( TO 1))
-64,VAL (T$(2 TO )))
2445 LET SP=SP+1: LET S(SP)=K
2450 NEXT K
2460 LET M(J,I)=C(K-1): RETURN
2500 LET C(K)=C(S(SP-1))+C(S(SP)
): LET S(SP)=0: LET SP=SP-1: LET
 S(SP)=K: GO TO 2450
2510 LET C(K)=C(S(SP-1))-C(S(SP)
): LET S(SP)=0: LET SP=SP-1: LET
 S(SP)=K: GO TO 2450
2520 LET C(K)=C(S(SP-1))*C(S(SP)
): LET S(SP)=0: LET SP=SP-1: LET
 S(SP)=K: GO TO 2450
2530 LET C(K)=C(S(SP-1))/C(S(SP)
): LET S(SP)=0: LET SP=SP-1: LET
 S(SP)=K: GO TO 2450
2540 LET C(K)=C(S(SP-1))^C(S(SP)
): LET S(SP)=0: LET SP=SP-1: LET
 S(SP)=K: GO TO 2450
2550 LET C(K)=0-C(S(SP)): LET S(
SP)=K: GO TO 2450
3010>DIM H(4):DIM V(7):DIM S(20)
:DIM S$(20,5):DIM E$(20,5):DIM G
$(20,5):DIM C(20)
```

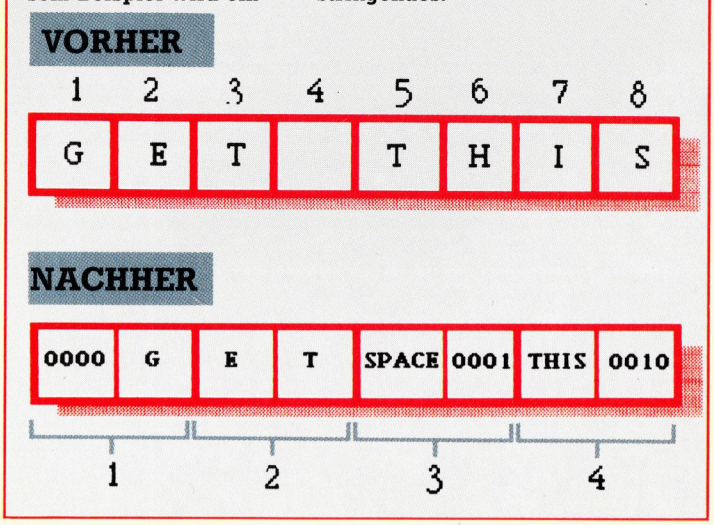
Rank und schlank

Diese Folge aus der Reihe Textkompressionen enthält das Assembler-Kompressionsprogramm für den 6502-Prozessor und ein BASIC-Ladeprogramm für die Z80-Version.

Kompressionsprogramm für den 6502

```
;++++ 6502 TEXT COMPRESSION ++++
;
ZPTR1=$8B               ;0 PAGE INPUT POINTER
ZPTR2=$8D               ;0 APGE OUTPUT POINTER
ZPTR3=$FB               ;UTILITY TABLE POINTER
ZPTR4=$FE               ;UTILITY TABLE POINTER
*=$C000
OUTPUT  *=*+2           ;START OF O/P STRING
INPUT   *=*+2           ;START OF I/P STRING
STATUS  *=*+1           ;STATUS BYTE
MASK    *=*+1           ;NIBBLE MASK
LEN     *=*+1           ;INPUT STRING LEN STORE
OUTOFF  *=*+1           ;OUTPUT OFFSET STORE
TOKCNT  *=*+1           ;TOKEN COUNTER
TABCNT  *=*+1           ;TABLE COUNTER
TEMP    *=*+1           ;SAVE START OF O/P STRING
;
START   LDA INPUT
        STA ZPTR1       ;SET UP I/P POINTER
        LDA INPUT+1
        STA ZPTR1+1
        LDA OUTPUT
        STA ZPTR2       ;SET UP O/P POINTER
        LDA OUTPUT+1
        STA ZPTR2+1
        LDA #1
        STA OUTOFF      ;INIT O/P OFFSET
        LDY #0          ;INIT I/P OFFSET
        LDA (ZPTR1),Y
        STA LEN         ;STORE LENGTH I/P STRING
        LDA #255
        STA MASK        ;SET UP NIBBLE MASK
NCHAR   INY
        LDA (ZPTR1),Y   ;GET NEXT I/P CHAR
        JSR CHECK       ;VALID?
        BNE BADCHR
        JSR TOKEN       ;TOKEN?
        BEQ GOTTOK
        JSR FOURBT      ;FOUR BIT CODE?
        BEQ GOTFOR
        JSR EIGHTB      ;EIGHT BIT CODE?
        PHA             ;SAVE VALUE
        LDA #0
        JSR WRTNIB      ;SEND 0000 CODE
        PLA
GOTFOR  JSR WRTNIB      ;SEND 8 BIT CODE
;
REJOIN  CPY LEN         ;MORE I/P CHARS?
        BCC NCHAR       ;GO GET THEM
        LDA #0
        STA STATUS      ;0 STATUS=OK
        LDA #2
        JSR WRTNIB      ;END OF TEXT MARK
        LDA MASK
        BEQ NODEC
        DEC OUTOFF
NODEC   LDY #0
        LDA OUTOFF
        STA (ZPTR2),Y   ;WRITE O/P LENGTH
        RTS
;
BADCHR  LDA #255
        STA STATUS      ;SEND BAD STATUS
        RTS
GOTTOK  PHA
        LDA #1
        JSR WRTNIB      ;SEND 0001 CODE
        PLA
        JSR WRTNIB      ;AND TOKEN CODE
        JMP REJOIN
;
;++++ SUBROUTINES +++++
TOKEN   PHA             ;SAVE CURRENT CHAR
        TYA
        PHA             ;AND I/P OFFSET
        STY TEMP
        LDA ZPTR1
        CLC
        ADC TEMP        ;CALC ZPTR3 TO
        STA ZPTR3
        LDA ZPTR1+1     ;POINT AT CURRENT
        ADC #0          ;I/P CHAR
        STA ZPTR3+1
        LDA #<TOKTAB
        STA ZPTR4       ;SET ZPTR4
        LDA #>TOKTAB    ;TO POINT TO
        STA ZPTR4+1     ;TOKEN TABLE
        LDA #0
        STA TOKCNT      ;INIT TOKEN COUNTER
TOK1    LDY #0
        LDA (ZPTR4),Y   ;GET LENGTH NEXT TOKEN
        PHA             ;STORE IT
        TAX
        BEQ NOTFND      ;END OF TABLE
        LDA ZPTR4
        CLC
        ADC #1          ;INC ZPTR4
        STA ZPTR4       ;ZPTR3 & ZPTR4
        LDA ZPTR4+1     ;NOW BOTH AT
        ADC #0          ;TEXT STARTS
        STA ZPTR4+1
TOK2    LDA (ZPTR3),Y
        CMP (ZPTR4),Y   ;COMPARE
        BNE NEXTOK      ;NO MATCH GET NEXT TOKEN
        INY
        DEX             ;CONT TILL
        BNE TOK2        ;END TOKEN WORD
;++ TOKEN MATCHED ++
        PLA             ;CLEAR STACK
        PLA
        DEY
        STY TEMP
        CLC
        ADC TEMP        ;CALC NEW I/P OFFSET
        TAY             ;PUT IT IN Y
        PLA
        LDA TOKCNT
        LDX #0          ;SET Z=1
        RTS
;
NEXTOK  PLA             ;GET LENGTH TOKEN
        STA TEMP
        LDA ZPTR4
        CLC
        ADC TEMP
        STA ZPTR4       ;ADJUST ZPTR4
        LDA ZPTR4+1     ;TO POINT AT
```

Der Einsatz von Tokens sowie die Darstellung der häufigsten Buchstaben im Vier-Bit-Code ermöglicht eine Verkürzung des Originaltextes auf die Hälfte. In diesem Beispiel wird ein Acht-Buchstaben-String auf vier Bytes komprimiert. Beachten Sie die Vier-Bit-Codes als Einleitung eines Token-Wortes und zum Bezeichnen des Stringendes.




```
        ADC #0          ;NEXT TOKEN
        STA ZPTR4+1
        INC TOKCNT
        JMP TOK1
NOTFND  PLA             ;DISCARD
        PLA
        TAY             ;RESTORE I/P POINTER
        PLA             ;RESTORE I/P CHAR
        LDX #255        ;Z=0=FAILURE
        RTS
;
FOURBT  PHA             ;SAVE I/P CHAR
        STY TEMP        ;SAVE I/P OFFSET
        LDA #<TAB4BT
        STA ZPTR3
        LDA #>TAB4BT
        STA ZPTR3+1
        PLA             ;GET I/P CHAR BACK
        JMP TBSCAN
EIGHTB  PHA             ;SAVE I/P CHAR
        STY TEMP        ;SAVE I/P OFFSET
        LDA #<TAB8BT
        STA ZPTR3
        LDA #>TAB8BT
        STA ZPTR3+1
        PLA             ;GET I/P CHAR BACK
;
TBSCAN  LDY #0
TBSCN2  CMP (ZPTR3),Y
        BEQ TBSCN1      ;FOUND IT!
        INY
        CPY #16         ;LOOK FOR TAB END
        BNE TBSCN2
        LDY TEMP        ;RESTORE I/P OFFSET
        LDX #255        ;Z=0=FAILURE
        RTS
TBSCN1  TYA             ;PUT TABVAL IN A
        LDY TEMP
        LDX #0          ;Z=1=SUCCESS
        RTS
;
CHECK   CMP #' '
        BEQ RETURN
        CMP #','
        BEQ RETURN
        CMP #'.'
        BEQ RETURN
        CMP #'A'
        BCC NOGOOD
        CMP #$5B
        BCS NOGOOD
RETURN  LDX #0
        RTS
NOGOOD  LDX #255
        RTS
;
WRTNIB  PHA             ;SAVE NIBBLE
        STY TEMP        ;SAVE I/P OFFSET
        LDY OUTOFF
        LDA MASK
        BNE LEFT        ;<>0 MEANS LEFT-HAND
        PLA
        ORA (ZPTR2),Y   ;ADD NEW TO OLD
        STA (ZPTR2),Y   ;REPLACE IT
        INC OUTOFF
        LDY TEMP        ;RESTORE I/P OFFSET
        LDA #255
        STA MASK        ;RESET MASK FOR NEXT
        RTS
LEFT    PLA
        ASL A
        ASL A
        ASL A
        ASL A
        STA (ZPTR2),Y
        LDY TEMP
        LDA #0
        STA MASK
        RTS
;
;++++ MOST COMMON LETTERS ++++
TAB4BT  .BYT 0
        .BYT 0
        .BYT 0
        .BYT 'F'
        .BYT 'L'
        .BYT 'D'
        .BYT 'H'
        .BYT 'S'
        .BYT 'I'
        .BYT 'R'
        .BYT 'N'
        .BYT 'O'
        .BYT 'A'
        .BYT 'T'
        .BYT 'E'
        .BYT ' '
;++++ LESS COMMON LETTERS ++++
TAB8BT  .BYT 'C'
        .BYT 'M'
        .BYT 'U'
        .BYT 'G'
        .BYT 'Y'
        .BYT 'P'
        .BYT 'W'
        .BYT 'B'
        .BYT 'V'
        .BYT 'K'
        .BYT 'X'
        .BYT 'J'
        .BYT 'Q'
        .BYT 'Z'
        .BYT ','
        .BYT '.'
;++++ TOKEN TABLE ++++
TOKTAB  .BYT 3,'THE'

        .BYT 4,'THIS'

        .BYT 4,'THAT'

        .BYT 2,'IF'
```

```
.BYT 3,'YOU'                    .BYT 2,'IT'
.BYT 2,'ME'                     .BYT 2,'IS'
.BYT 3,'WAS'                    .BYT 3,'FOR'
.BYT 2,'HE'                     .BYT 2,'ON'
.BYT 3,'SHE'                    .BYT 2,'TO'
.BYT 4,'THEY'                   .BYT 0
.BYT 2,'OF'                     .END
```

BASIC-Ladeprogramm für Z80-Rechner

```
10 MEMORY 29999
20 FOR n=30000 TO 30640
30 READ p: POKE n,p: c=(c+p)
40 NEXT n
50 IF c<>64282 THEN PRINT "
Checksum error..."
60 STOP
100 DATA 24,7,70,160,102,158,0,255,0,42,50
,117,126,50,56,117
110 DATA 35,237,91,52,117,213,19,237,83,52
,117,62,255,50,55,117
120 DATA 126,205,253,117,32,57,205,162,117
,40,59,205,223,117,40
130 DATA 10,205,232,117,245,62,0,205,22,
118,241,205,22,118,35
140 DATA 58,56,117,61,50,56,117,167,32,216
,50,54,117,62,2,205
150 DATA 22,118,235,209,58,55,117,167,40,1
,43,167,237,82,125,18
160 DATA 201,209,62,255,50,54,117,201,245,
62,1,205,22,118,241
170 DATA 205,22,118,24,203,229,235,33,2,
119,58,56,117,79,6,15
180 DATA 126,167,40,42,213,229,35,197,71,
26,190,32,21,35,19,5
190 DATA 32,13,121,50,56,117,193,225,225,
225,235,43,175,120,201
200 DATA 13,32,231,193,5,225,126,95,22,0,
35,25,209,24,210,225
210 DATA 126,183,201,229,33,226,118,205,
241,117,225,201,229,33
220 DATA 242,118,205,241,117,225,201,6,15,
190,40,5,35,16,250,183
230 DATA 201,120,201,254,32,200,254,44,200
,254,46,200,254,65,56
240 DATA 8,254,91,48,4,79,175,121,201,62,
255,167,201,79,58,55
250 DATA 117,237,91,52,117,167,32,14,26,
177,18,19,237,83,52,117
260 DATA 62,255,50,55,117,201,121,203,39,
203,39,203,39,203,39
270 DATA 18,175,50,55,117,201,42,50,117,35
,34,50,117,237,91,52
280 DATA 117,213,62,255,50,55,117,205,185,
118,254,2,202,129,118
290 DATA 210,153,118,167,202,139,118,205,
185,118,205,110,118,71
300 DATA 126,35,205,174,118,16,249,24,225,
33,2,119,71,62,15,144
310 DATA 71,4,126,35,5,200,95,22,0,25,24,
246,42,52,117,209,167
320 DATA 237,82,125,18,201,205,185,118,33,
242,118,205,164,118,205
330 DATA 174,118,24,182,33,226,118,205,164
,118,205,174,118,24,171
340 DATA 95,62,15,147,95,22,0,25,126,201,
237,91,52,117,19,18
350 DATA 237,83,52,117,201,58,55,117,42,50
,117,167,126,32,14,230
360 DATA 15,35,34,50,117,79,62,255,50,55,
117,121,201,203,47,203
370 DATA 47,203,47,203,47,230,15,79,175,50
,55,117,121,201,32,69
380 DATA 84,65,79,78,82,73,83,72,68,76,70,
0,0,0,67,77,85,71
390 DATA 89,80,87,66,86,75,88,74,81,90,44,
46,3,84,72,69,4,84
400 DATA 72,73,83,4,84,72,65,84,2,73,70,3,
89,79,85,2,77,69
410 DATA 3,87,65,83,2,72,69,3,83,72,69,4,
84,72,69,89,2,79,70
420 DATA 2,73,84,2,73,83,3,70,79,82,2,79,
78,2,84,79,0,205,167
430 DATA 126,32,14,230,15,35,34,50,117,79,
62,255,50,55,117,121
440 DATA 201,203,47,203,47,203,47,203,47,
230,15,79,175,50,55,117
450 DATA 121,201,32,69,84,65,79,78,82,73,
83,72,68,76,70,0,0
460 DATA 0,67,77,85,71,89,80,87,66,86,75,
88,74,81,90,44,46,3
470 DATA 84,72,69,4,84,72,73,83,4,84,72,65
,84,2,73,70,3,89
480 DATA 79,85,2,77,69,3,87,65,83,2,72,69,
3,83,72,69,4,84,72
490 DATA 69,89,2,79,70,2,73,0,0
```

BASIC-Dialekte

Wer mit dem Spectrum arbeitet, muß am Ladeprogramm eine kleine Änderung vornehmen. Das Kompressionsprogramm läuft auch auf anderen Z80-Rechnern, wie beispielsweise Memotech, Schneider und Einstein. Bei diesen Rechnern muß dann Zeile 10 des Ladeprogramms abgewandelt werden (detaillierte Angaben finden Sie in Ihrem Handbuch):

```
10 CLEAR 29999
```

Jedem das Seine

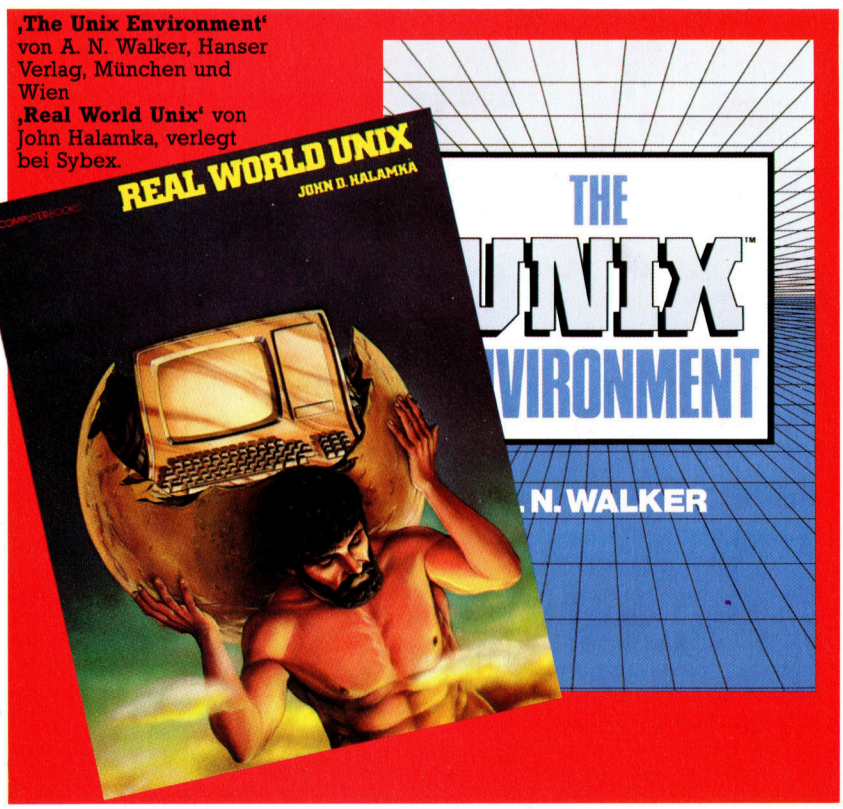
Da sich bei Unix die Benutzerschnittstelle nach Wunsch umdefinieren läßt und die Informationsströme freizügig zu dirigieren sind, kann sich jeder das System auf seine Bedürfnisse zuschneiden – dazu einige Beispiele.

Es gibt eine Menge Unix-Literatur, von dicken Wälzern bis hin zu knappen Einführungen. Der Unix-Einsteiger ist sehr gut mit dem locker geschriebenen Text ‚The Unix Environment' von A. N. Walker bedient (demnächst unter dem Titel ‚Die Unix-Welt' auch in deutscher Übersetzung beim Carl Hanser Verlag erhältlich, München/Wien). Darin werden die wichtigsten Aspekte des Unix-Systems sowie typische Hardware-Konfigurationen und die Systemorganisation behandelt.
Das Buch ‚Real World Unix' von John Halamka ist praktisch orientiert und allen zu empfehlen, die den Umgang mit Unix in der kürzesten Zeit lernen wollen.

Bei den meisten Betriebssystemen ist zwar auch eine begrenzte Anpassung an die spezifischen Anforderungen des Benutzers vorgesehen; so gibt es die Möglichkeit, einen Satz von Kommandos als Block aufzurufen (etwa mit Hilfe der CP/M-SUBMIT- oder der MS-DOS.BAT-Dateien), oder bestimmte Routinen bei Inbetriebnahme des Systems automatisch zu starten. Die Möglichkeiten von Unix gehen aber weit darüber hinaus.

Unix verfügt mit der „Shell Script" über eine eigenständige Programmiersprache für die Gestaltung von Ein/Ausgabe-, Such- und Iterationsprozeduren. In Verbindung mit dem Pipeline-Verfahren und den Umleitungsoperatoren lassen sich damit viele Aufgaben bewältigen, ohne auf eine konventionelle Programmiersprache zurückgreifen zu müssen.

Die ‚Shell' entspricht dem Kommandointerpreter anderer Systeme und kann vom Benutzer vollständig umdefiniert werden. Daraus folgt, daß Systeme, die auf der gleichen Unix-Version beruhen, äußerlich sehr verschieden aussehen können. Dieser Artikelreihe wurde die Berkeley-Fassung 4.2 mit der sog. „C-Shell" zugrundegelegt, die sich an der Sprache C orientiert.



‚The Unix Environment' von A. N. Walker, Hanser Verlag, München und Wien
‚Real World Unix' von John Halamka, verlegt bei Sybex.

Für die Systemanpassung stehen zunächst ein paar Befehle zur Verfügung, die auf die Programmierumgebung Einfluß nehmen. Wie bereits erwähnt, wird der Zugriff auf eine Datei oder Directory für drei Benutzertypen getrennt geregelt: für den Anwender selbst (u=user), für seine Gruppe (g=group) und für alle anderen (o=others). Jede Benutzerkategorie erhält für die einzelnen Dateien oder Directories individuelle Zugriffsrechte, die beim Auflisten der Directory im Langformat (über ls -l) als drei ‚rwx'-Sätze erscheinen – ‚r' bedeutet Freigabe für Lesen (read), ‚w' für Schreiben (write), ‚x' für Ausführen (execute). Wo die entsprechende Zugriffserlaubnis nicht erteilt ist, steht ein Strich.

Die Benutzerkennungen sind mit dem Befehl ‚chmod' änderbar; er hat das Format

```
chmod Kategorie[+,–]Erlaubnis Datei
bzw. Directory
```

Dabei bewirkt ‚+' die Erteilung und ‚–' die Löschung der Erlaubnis. Um für den Benutzer und für seine Gruppe das Schreiben in der Datei „Adressen" freizugeben, tippen Sie ein

```
chmod ug+w adressen
```

Dabei können auch mehrere Änderungen auf einmal (mit Komma statt Leerraum als Trennzeichen) angegeben werden.

Wenn ein neues Terminal eingerichtet werden soll, sind zunächst die Betriebsparameter zu setzen. Die meisten Unix-Systeme enthalten bereits die Treibertabellen für alle gängigen Terminaltypen, und Sie brauchen nur den entsprechenden ‚set environ'-Befehl (Umgebungsdefinition) zu geben:

```
setenv TERM Terminaltyp
```

Es ist im allgemeinen nicht erforderlich, Softwarepakete für eine bestimmte Terminal-Konfiguration auszulegen; zumindest theoretisch arbeitet ein und dieselbe Programmfassung unter Unix auf jeder Anlage.

Für ungebräuchliche Terminals und für spezielle Zwecke lassen sich die Parameter mit ‚stty' aber auch einzeln setzen. Der Befehl

```
stty everything
```

liefert zunächst eine Liste aller aktuellen Einstellungen,

```
stty all
```

dagegen ein verkürztes Verzeichnis mit den wichtigsten Werten. Die Einstellungen für die einzelnen Terminalfunktionen sind dann neu

Maßarbeit

```
Berkeley 4.2 Vax/Unix (infsc3)
Type (Ctrl-D) to disconnect

login: com-mcc
Password:
You are a Normal user (class 3)
Jobs : 15  Superiors : 4  Maximum : 21
Last Login: Wed Nov 13 15:05:36 on ttyn08

Hello nice to see you!          (Dies ist meine User-Meldung)

%cat .login                     (Protokoll der .login-Datei)
setenv TERM tvi910+             (Terminal-Einrichtung)
stty erase                      âH als Löschsymbol erlaubt Löschen mit
                                Rück- statt Delete-Taste)
echo Hello nice to see you!     (Begrüßungstext)

%cat .cshrc
setenv EDITOR /usr/local/emacs
setenv NAME 'Mike Curtis'
set path=(. /usr/local/m2 /usr/local /usr/ucb /usr/bin /bin /usr/games )
(Pfadnamen, anhand derer die zu einem Kommando gehörige Routine gesucht wird; auch auf Programme außerhalb der aktuellen Directory ist auf diese Weise ein unmittelbarer Zugriff möglich)

set history=25                  (Shell merkt sich jeweils die letzten
                                25 Kommandos)
set ignnoreeof                  (Datei-Endzeichen ^D=end of file unter-
alias h history                 bindet versehentliches Ausloggen)
alias ty more
alias dir ls                    (Umbenennung von Unix-Befehlen, z. B.
                                Verwendung von ‚dir' statt ‚ls' zum Auslisten einer Directory)
%dir                            (Ausprobieren der Wirkung von ‚dir')
lsfile mike rec.c receive rx.p transmit

%cat > ! temp                   (cat ohne Nennung einer Eingabedatei,
                                verkettet die Tastatur mit dem neuen File
                                ‚temp', bis ^D=EOF eingegeben wird)
echo this is a list of my files
/bin/ls                         (Erstellen eines neuen ls-Befehls mit
%mv temp ls                     Kopfzeilenausgabe im File ‚temp')
                                (Umbennen von ‚temp' in ‚ls')
%chmod u+x ls                   (Neues ls-File als Kommandodatei aus-
                                führbar, sonst nur Aufruf über sh<ls)
%ls                             (Test des neuen ls-Befehls)
this is a list of my files
ls lsfile mike rec.c receive rx.p transmit

%history                        (Auflistung der bisher erteilten Befehle)
     1  dir
     2  cat > ! temp
     3  mv temp ls
     4  chmod u+x ls
     5  ls
     6  history

% !5                            (Wiederholungsaufruf des 5.Befehls)
ls                              (Bestätigung des Kommandos)
this is a list of my files
ls lsfile mike rec.c rewceive rx.p transmit

%stty everything                (Auflisten aller Terminal-Parameter)
new tty, speed 1200 baud
even odd -raw -nl echo -lcase -tandem -tabs -cbreak ff1
-crtbs -crterase -crtkill -ctlecho -prterase -tostop
-tilde -flusho -mdmbuf -litout -nohang
-pendin -decctlq -nqflsh
erase kill werase rprnt flush lnext susp intr quit stop eof
^H   ^U   ^W   ^R   ^O   ^V   ^Z/^Y ^C   ^\   ^S/^Q ^D

%stty all                       (Wichtigste Terminal-Spezifikationen)
new tty, speed 1200 baud; -tabs ff1

erase kill werase rprnt flush lnext susp intr quit stop eof
^H   ^U   ^W   ^R   ^O   ^V   ^Z/^Y ^C   ^\   ^S/^Q ^D

%logout
```

definierbar, zum Beispiel:

```
stty Funktion Tastenbelegung
```

Ein anderer nützlicher Terminalbefehl ist ‚lock' (Sperren). Er unterbindet eine unbefugte Benutzung. Nach diesem Kommando wird ein neues Paßwort angefordert (nicht die Login-Kennung), und zwar in doppelter Ausführung. Anschließend ignoriert das Betriebssystem jede Eingabe über das betreffende Terminal, es sei denn, das Paßwort wird ein drittes Mal eingetippt.

Die folgenden Kommandos haben nur bei der C-Shell Gültigkeit. Über ‚history' erhalten Sie eine Auflistung aller bisher erteilten Befehle bis zu einer vorgewählten Höchstzahl. Jeden einzelnen können Sie dann durch Angabe seiner Platznummer mit vorgesetztem Rufzeichen wiederholen lassen, z. B. führt ‚!4' zur erneuten Ausführung des vierten Befehls.

Das Kommando „alias' (anders) dient dazu, einen Befehlsnamen oder eine beliebige Zeichenkette umzudefinieren. Wo immer die neue Bezeichnung erscheint, wird sie entsprechend der ‚alias'-Zuordnung uminterpretiert. Wollen z. B. CP/M- oder MS-DOS-Anhänger weiterhin ‚dir' statt ‚ls' verwenden, um eine Directory aufzulisten, brauchen diese nur einzutippen

```
alias dir ls
```

Ab sofort wird ‚dir' als ‚ls' gelesen und ist mit den üblichen Optionen zu benutzen. Wenn die (alte) Zeichenkette ein Semikolon enthält, das in Unix zur Trennung von Kommandos innerhalb einer Zeile dient, ist es in Anführungsstriche zu setzen:

```
alias dir';'date ls -a
```

Durch gezielten Gebrauch von ‚alias' lassen sich bei Unix umständliche Kommandos sehr vereinfachen.

In den meisten Benutzer-Directories tauchen einige spezielle Systemdateien auf, vor deren Namen Punkte erscheinen. Das ist ein Schutz gegen versehentliches Löschen. Zwei von diesen Dateien sind von besonderem Interesse, da sie die Benutzerumgebung definieren, nämlich .login und .cshrc, die in der Bourne-Shell als .profile zusammengefaßt sind und eine Anzahl von Shell-Kommandos enthalten. Die .login-Datei wird beim Einloggen gestartet, während auf .cshrc bei jedem Ansprechen der Shell zugegriffen wird, denn diese Datei umfaßt vor allem die alias-, setenv- und set-Kommandos. Dank der zahlreichen Optionen bei setenv und set, die systemabhängig sind und hier aus Platzgründen nicht aufgeführt werden können, läßt sich das System einschließlich der Peripherietreiber praktisch in jedem Detail so einrichten, wie es den Bedürfnissen des Benutzers entspricht.

Aquarius

Der Aquarius ist ein preisgünstiger und interessanter Heimcomputer, den die als Spielzeughersteller bekannte Firma Mattel auf den Markt gebracht hat.

Auf dem Höhepunkt des Booms der kleinen Heimcomputer versuchten sich viele Firmen an Sinclair-ähnlichen Rechnern. Damals brachte auch der kalifornische Spielzeughersteller Mattel Inc. unter dem Namen ‚Aquarius' einen Computer heraus, der einen Rückblick wert ist, weil er gleichzeitig als Steuerzentrale für die gesamte Hauselektrik vorgesehen war.

Der Aquarius verfügte über eine Bus-Schnittstelle, die im übrigen auch für allerlei Zubehör gedacht war – von kleinen 4-K-RAM-Modulen bis hin zu einem großen Erweiterungschassis. Das Praktischste ist die kleine Erweiterungsbox mit zwei Steckplätzen für Zusatzspeicher oder Softwaremodule sowie mehrere Tonkanäle und Anschlüsse für zwei Handsteuergeräte. Wenn Sie ein 16K-RAM auf dem einen Steckplatz haben und ein Software-ROM wie ‚Finplan' auf dem andern, ergibt sich ein vielseitiges System.

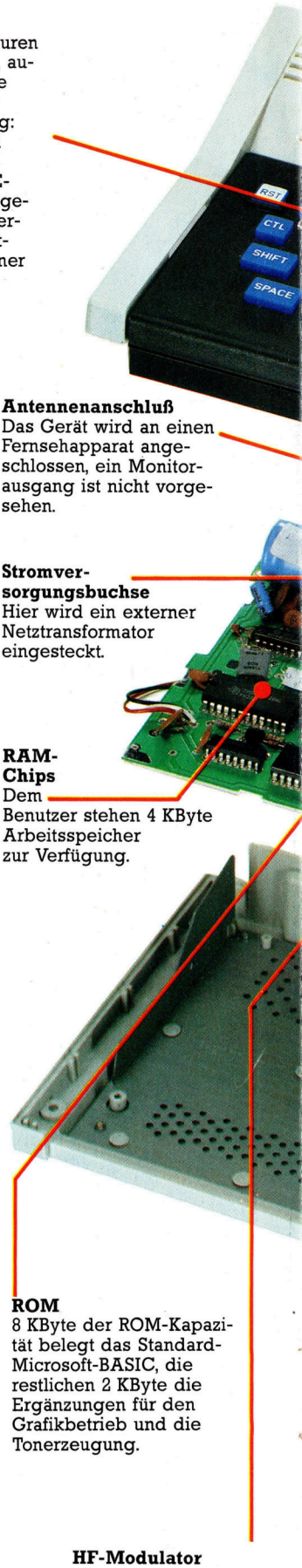
Das serienmäßige 4-KByte-RAM ist mager, aber nach Ausbau der 64 KByte bietet der Rechner vom Speicherplatz her ebensoviel wie jeder andere Heimcomputer. Tastatur und Bildschirmwiedergabe entsprechen dagegen nicht dem Standard anderer Maschinen. Blind schreiben ist wegen des schlechten Anschlags nicht möglich, und es fehlt eine richtige Leertaste. Auch das Bildschirmformat ist mit 24 Zeilen zu 40 Zeichen kaum zu gebrauchen.

Mini-Expander
Mit zwei Steckplätzen ermöglicht diese Erweiterungsbox den gleichzeitigen Betrieb eines RAM-Moduls und eines Software-ROMs. Der Mini-Expander ist für zwei Handsteuergeräte eingerichtet.

Aquarius-Drucker
Dieses Gerät arbeitet nach dem Thermodruckverfahren und benötigt daher Spezialpapier. Es erzeugt maximal 80 Zeichen/s bei 40 Zeichen pro Zeile. Ausreichend für Hobbyanwendungen.

Für Text oder Hintergrund stehen 16 Farben zur Verfügung. Die Zeichen können zwar nicht frei definiert werden, aber es lassen sich 256 Symbole darstellen, einschließlich Klein- und Großbuchstaben sowie einer Auswahl von Grafikzeichen. Die höchste Grafikauflösung beträgt 320 × 192 Punkte. Es ist nur ein Antennen- und kein Monitorausgang vorhanden, so daß die Wiedergabe mittelmäßig, mit leichtem Blaustich und begrenzter Konturenschärfe, ist. Eine Klangerzeugung ist zwar vorgesehen, aber nicht mit Hüllkurvensteuerung und Auswahl der Schwingungsform wie bei anderen Rechnern. Im übrigen ist ein Standard-Microsoft-BASIC eingebaut.

Die interessanteste Variante an Zubehör für den Aquarius ist das BSR-X-10-System. Damit läßt sich nämlich die gesamte Hauselektrik über den Computer steuern; bis zu 255 verschiedene Anschlußgeräte sollen durch Rechnersignale zu beeinflussen sein. Eine zusätzliche Verdrahtung entfällt, weil die Steuersignale der Netzspannung auf der vorhandenen Installation überlagert werden. Davon merkt das Gerät unmittelbar zwar nichts, aber ein X-10-Empfänger, der an irgendeine Steckdose angeschlossen wird, kann die Steuerimpulse decodieren und das angeschlossene Gerät entsprechend schalten. Das System wird für eine Woche im voraus vom Rechner aus programmiert. Während des Programmiervorgangs ist der Computer blockiert.



Aquarius-Tastatur
Wie alle Folientastaturen wenig überzeugend, außerdem hat sie keine standardmäßige QWERTY-Anordnung: Lange Leertaste und zweiter Umschalter rechts fehlen, die RETURN-Taste liegt ungewohnt, und das Rastermaß der Tastatur entspricht nicht dem einer Schreibmaschine.

Antennenanschluß
Das Gerät wird an einen Fernsehapparat angeschlossen, ein Monitorausgang ist nicht vorgesehen.

Stromversorgungsbuchse
Hier wird ein externer Netztransformator eingesteckt.

RAM-Chips
Dem Benutzer stehen 4 KByte Arbeitsspeicher zur Verfügung.

ROM
8 KByte der ROM-Kapazität belegt das Standard-Microsoft-BASIC, die restlichen 2 KByte die Ergänzungen für den Grafikbetrieb und die Tonerzeugung.

HF-Modulator
Das Bildschirmsignal wird einem HF-Träger aufmoduliert.

computer kurs

Leserbefragung

Liebe Leserin, lieber Leser
nun ist Ihr Sammelwerk beinahe komplett.
Da wir gerne wissen möchten, wie Ihnen
„Computer Kurs" gefallen hat, bitten wir
Sie, nachstehende Fragen zu beantworten
und uns den ausgefüllten Fragebogen
zurückzuschicken.

Vielen Dank für Ihre Hilfe.

DER VERLAG

1. Wer hat diese Ausgaben von Computer-Kurs gekauft?

(11)

- ich selbst ☐ 1
- Mutter ☐ 2
- Vater ☐ 3
- Schwester ☐ 4
- Bruder ☐ 5
- Ehepartner ☐ 6
- Freund ☐ 7
- Freundin ☐ 8
- jemand anderer ☐ 9
  wer?
  .....

2. Wie haben Ihnen die Hefte von Computer-Kurs insgesamt gefallen?

(12)

- sehr gut ☐ 1
- gut ☐ 2
- weniger gut ☐ 3
- gar nicht ☐ 4

3. a) Was hat Ihnen besonders gut gefallen?

(13,14)

.....
.....
.....

b) Und was hat Ihnen weniger gut gefallen?

(15,16)

.....
.....
.....

c) Haben Sie in Computer-Kurs spezielle Themen vermißt?

(17,18)

- ja ☐ 1
- nein ☐ 2

wenn ja: was haben Sie vermißt?

.....
.....

4. Wieviele Ausgaben von Computer-Kurs haben Sie gesammelt?

(19,20)

- alle ☐
- nicht alle, sondern ___ Ausgaben

5. Haben Sie sich auch die Sammelordner gekauft?

(21)

- ja ☐ 1
- nein ☐ 2

6. Was war für Sie der Grund, Computer-Kurs zu sammeln?

(22)

- ich wollte Computerwissen erwerben ☐ 1
- ich wollte mein Computerwissen vertiefen ☐ 2
- andere Gründe ☐ 3
  welche?
  .....
  .....
  .....

7. Wie haben Ihnen die einzelnen Rubriken von Computer-Kurs gefallen?

	sehr gut	gut	weniger gut	gar nicht
				(23)
– Hardware	☐ 1	☐ 2	☐ 3	☐ 4
				(24)
– Peripherie	☐ 1	☐ 2	☐ 3	☐ 4
				(25)
– Tips für die Praxis	☐ 1	☐ 2	☐ 3	☐ 4
				(26)
– Software	☐ 1	☐ 2	☐ 3	☐ 4
				(27)
– Bits und Bytes	☐ 1	☐ 2	☐ 3	☐ 4
				(28)
– BASIC-Kurs	☐ 1	☐ 2	☐ 3	☐ 4
				(29)
– LOGO-Kurs	☐ 1	☐ 2	☐ 3	☐ 4
				(30)
– Computer Welt	☐ 1	☐ 2	☐ 3	☐ 4
				(31)
– Fragen und Antworten	☐ 1	☐ 2	☐ 3	☐ 4
				(32)
– Fachwörter von A bis Z	☐ 1	☐ 2	☐ 3	☐ 4

8. a) Besitzen Sie selbst einen Computer?

(33)

- ja ☐ 1
- nein ☐ 2

b) Falls ja: welchen?

(34)

.....
.....

Wie lange besitzen Sie Ihren Computer?

(35)

- weniger als 3 Monate ☐ 1
- 3 bis 6 Monate ☐ 2
- 6 bis 12 Monate ☐ 3
- 1 bis 2 Jahre ☐ 4
- länger als 2 Jahre ☐ 5

9. Was war für Sie der Hauptgrund, einen Computer anzuschaffen?

(36)

- allgemeines technisches Interesse ☐ 1
- schulische oder berufliche Gründe ☐ 2
- Spaß an Computern ☐ 3
- Sonstiges ☐ 4
- was?
  .....
  .....

10. Wofür benutzen Sie Ihren Computer?

	häufig	gelegentlich	selten	nie
				(37)
– Programmieren lernen	☐ 1	☐ 2	☐ 3	☐ 4
				(38)
– Spielen	☐ 1	☐ 2	☐ 3	☐ 4
				(39)
– für berufliche Zwecke	☐ 1	☐ 2	☐ 3	☐ 4
				(40)
– anderes was?	☐ 1	☐ 2	☐ 3	☐ 4

.....
.....

11. Welche Peripheriegeräte besitzen Sie?

(41)

- Cassettenrecorder ☐ 1
- Diskettenstation ☐ 2
- Drucker ☐ 3
- Plotter ☐ 4
- Joystick ☐ 5
- andere ☐ 6
  welche?
  .....
  .....
  .....
  .....
  .....

12. Wieviele Spielprogramme besitzen Sie?

(42,43)

___ Programme

13. Wie lange benutzen Sie Ihren Computer durchschnittlich pro Woche?

(44,45)

durchschnittlich ___ Stunden pro Woche

14. Planen Sie, sich einen Computer zu kaufen?

(46)

- ja ☐ 1
- nein ☐ 2

15. Lesen Sie noch andere Computer-Zeitschriften?

(47,48)

- ja ☐ 1
- nein ☐ 2

wenn ja: welche?

. . . . . .

. . . . . .

. . . . . .

16. a. Für welche Gebiete/Themen interessieren Sie sich noch? (Bitte alle aufführen)

(49,50)

. . . . . .

. . . . . .

. . . . . .

b. Und welche davon würden Sie als Ihr Hobby bezeichnen?

(51,52)

. . . . . .

. . . . . .

17. Wieviel Stunden sehen Sie durchschnittlich fern in der Woche?

	gar nicht	1 Std.	2 Std.	3 Std.	4 Std. u. mehr	
ARD	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	(53)
ZDF	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	(54)
3. Programm	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	(55)

18. An welchen Tagen sehen Sie in der Woche zwischen 17.30 und 20.00 Uhr fern?

	ARD	ZDF	3. Programm	
Montag	☐ 1	☐ 2	☐ 3	(56)
Dienstag	☐ 1	☐ 2	☐ 3	(57)
Mittwoch	☐ 1	☐ 2	☐ 3	(58)
Donnerstag	☐ 1	☐ 2	☐ 3	(59)
Freitag	☐ 1	☐ 2	☐ 3	(60)
Samstag	☐ 1	☐ 2	☐ 3	(61)

19. Hören Sie regelmäßig Radio?

(62)

Nein ☐ 1

Ja ☐ 2

Welche Sender?

. . . . . .

. . . . . .

20. Zu welchen Zeiten schalten Sie am häufigsten das Radio ein?

(63)

- Vor 8.00 Uhr morgens ☐ 1
- Zwischen 8.00 und 11.00 Uhr morgens ☐ 2
- In der Mittagszeit/Pause ☐ 3
- Nachmittags bis 16.00 Uhr ☐ 4
- Zwischen 16.00 und 20.00 Uhr ☐ 5
- Nach 20.00 Uhr ☐ 6

21. Welche Tageszeitungen lesen Sie regelmäßig, d.h. an mindestens 3 Tagen in der Woche?

(64)

- regionale Tageszeitung ☐ 1
- Bild-Zeitung ☐ 2
- Frankfurter Allgemeine Zeitung ☐ 3
- Frankfurter Rundschau ☐ 4
- Die Welt ☐ 5
- Die Süddeutsche Zeitung ☐ 6
- Andere (bitte angeben welche) ☐ 7

. . . . . .

- lese keine Tageszeitung ☐ 8

22. Kaufen Sie regelmäßig Zeitschriften?

(65)

- ja ☐ 1
- nein ☐ 2

(66,67)

wenn ja, welche?

. . . . . .

. . . . . .

. . . . . .

23. Haben Sie oder jemand sonst in Ihrem Haushalt irgendein anderes wöchentlich erscheinendes Sammelwerk bezogen?

(68)

- Nein ☐ 1
- Ja ☐ 2

Wenn ja, welche:

- Wie geht das? ☐ 3
- Fotopraxis ☐ 4
- Kochen Sie mit ☐ 5
- Harmonie ☐ 6
- Aero ☐ 7
- Maler ☐ 8
- Computer Kurs ☐ 9
- Erzähl mir was ☐ 0
- Andere (bitte angeben welche) ☐ X

. . . . . .

. . . . . .

STATISTIK

24. Sind Sie

(69)

- männlich ☐ 1
- weiblich ☐ 2

25. Familienstand

- ledig, alleinstehend ☐ 3
- verheiratet, mit einem Partner zusammenlebend ☐ 4
- geschieden/verwitwet ☐ 5

26. Wie alt sind Sie?

(70,71)

. . . . . Jahre

27. Sind Sie berufstätig?

(72)

- ja, voll berufstätig ☐ 1
- ja, teilweise berufstätig ☐ 2
- nein, bin Student bzw. Schüler ☐ 3
- nein, bin Rentner, also nicht mehr berufstätig ☐ 4
- nein, nicht berufstätig (z.B. Hausfrau) ☐ 5

28. Sind Sie in Ihrem Haushalt

(73)

	ja	nein
a) Der Haushaltsvorstand	☐ 1	☐ 2
b) Die haushaltsführende Person	☐ 3	☐ 4

29. Welcher Berufsgruppe gehören Sie selbst an, bzw. haben Sie früher angehört
Falls Sie selbst nicht der Haushaltsvorstand sind, kreuzen Sie bitte auch das zutreffende Kästchen in der zweiten Spalte für den Haushaltsvorstand an.

(74,75)

	Befragte selbst	Haushaltsvorstand
– Inhaber, Leiter von Unternehmen, selbständig	☐ 1	☐ 1
– Freiberufler	☐ 2	☐ 2
– leitender Angestellter	☐ 3	☐ 3
– sonstiger Angestellter	☐ 4	☐ 4
– leitender Beamter	☐ 5	☐ 5
– sonstiger Beamter	☐ 6	☐ 6
– Facharbeiter	☐ 7	☐ 7
– sonstiger Arbeiter	☐ 8	☐ 8
– zur Zeit arbeitslos	☐ 9	☐ 9
– nie berufstätig gewesen	☐ 0	☐ 0

30. Wieviele Personen leben insgesamt in Ihrem Haushalt?

(76)

..... Personen

31. Und wieviele davon sind Kinder unter 14 Jahren?

(77)

..... Kinder unter 14 Jahre

32. In welchem Land bzw. Bundesland leben Sie?

(78)

– Schleswig-Holstein	☐ 1
– Hamburg	☐ 2
– Niedersachsen	☐ 3
– Bremen	☐ 4
– Nordrhein-Westfalen	☐ 5
– Hessen	☐ 6
– Rheinland-Pfalz	☐ 7
– Baden-Württemberg	☐ 8
– Saarland	☐ 9
– Bayern	☐ 0
– West-Berlin	☐ X

(79)

– Österreich	☐ 1
– Schweiz	☐ 2

33. Wieviele Einwohner hat Ihr Wohnort?

(80)

– bis unter 5.000 Einwohner	☐ 1
– 5.000 bis unter 20.000 Einwohner	☐ 2
– 20.000 bis unter 100.000 Einwohner	☐ 3
– 100.000 bis unter 500.000 Einwohner	☐ 4
– 500.000 Einwohner und mehr	☐ 5

34. Wir möchten gelegentlich eine Untersuchung darüber anstellen, wie unsere Zeitschriften gefallen haben. Wären Sie bereit, uns dabei zu unterstützen? Wenn ja, geben Sie bitte nachstehend Ihre Telefonnummer an, damit einer unserer Mitarbeiter Sie eventuell erreichen kann.

– nein	☐ 1
– ja	☐ 2

wenn ja: Tel. Nr. (mit Vorwahl)

.....
.....
.....
.....
.....
.....
.....

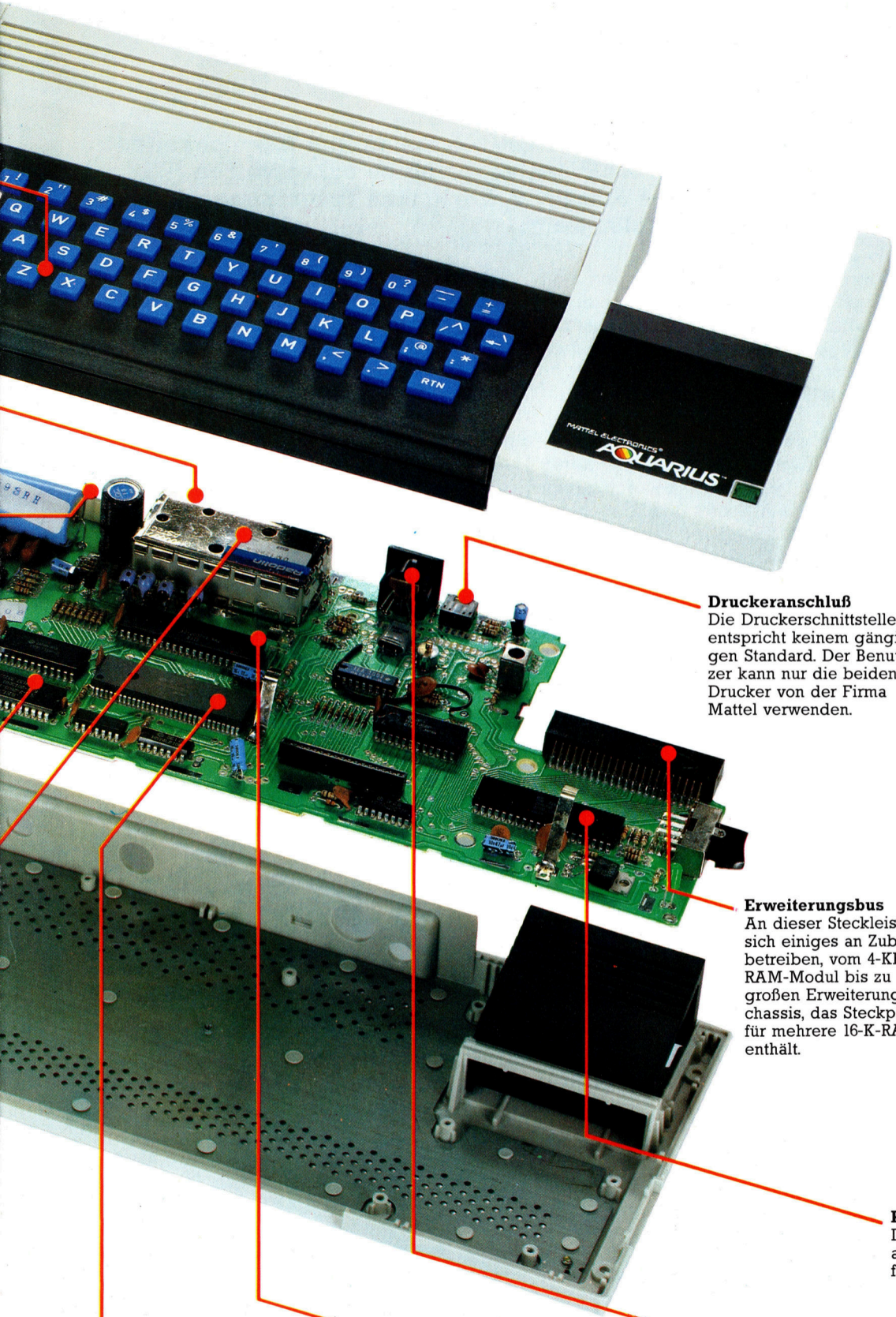
35. Falls Sie noch weitere Anmerkungen, Kommentare, Kritik zu Computer-Kurs haben, schreiben Sie sie bitte hier:

Bitte lösen Sie den Fragebogen heraus und stecken Sie ihn in einen Umschlag. Senden Sie ihre Antwort an:

Computer Kurs
Leserbefragung

computer kurs

Marshall Cavendish Int. Ltd.
Sammelwerk-Service
Postfach 10 57 03
D 2000 Hamburg 1



Aquarius

ABMESSUNGEN:
345 × 55 × 150 mm

ZENTRALEINHEIT:
Z80 mit 3,5 MHz-Takt

SPEICHER:
10 KByte ROM, 4 KByte RAM (auf 64 KByte ausbaubar)

BILDSCHIRMFORMAT:
24 Zeilen zu 40 Zeichen, 16 Farben (Vorder- und Hintergrund unabhängig wählbar); 256 feste Zeichen (keine Möglichkeit der Definition durch den Benutzer)

SCHNITTSTELLEN:
Cassettenrecorder, Drucker, Erweiterungsbus

EINGEBAUTE SPRACHE:
Microsoft BASIC

MITGELIEFERT:
Handbuch für BASIC und Inbetriebnahme, Antennenkabel

TASTATUR:
49 Druckpunkttasten, Reset-Knopf mit Schutz gegen versehentliche Betätigung

DOKUMENTATION:
Die Anleitung ist besonders gut für Anfänger geeignet; dazu gehört ein Satz praktischer Klappkarten, auf denen die wichtigsten Funktionen des Rechners und des BASIC-Systems beschrieben sind.

Druckeranschluß
Die Druckerschnittstelle entspricht keinem gängigen Standard. Der Benutzer kann nur die beiden Drucker von der Firma Mattel verwenden.

Erweiterungsbus
An dieser Steckleiste läßt sich einiges an Zubehör betreiben, vom 4-KByte-RAM-Modul bis zu einem großen Erweiterungschassis, das Steckplätze für mehrere 16-K-RAMs enthält.

Prozessor
Der verwendete Z80 arbeitet mit einer Taktfrequenz von 3,5 MHz.

Video-Chip
Die Bereitstellung des Videosignals erfordert einen hohen Schaltungsaufwand. Beim Aquarius ist der Videochip größer als der Mikroprozessor.

Sicherheitschip
Die Aquarius-ROMs sind durch einen speziellen Chip geschützt.

Cassettenrecorder-Buchse

. . . hat man Töne

Das Betriebssystem der Schneider-CPC-Reihe enthält Möglichkeiten zur Tonsteuerung, die den BASIC-Klangbefehlen nahekommen. Wir zeigen, wie das OS Klänge erzeugt und wie man Programme mit Hintergrundmusik untermalen kann.

Die Schneider-CPC-Reihe arbeitet mit dem Tongenerator AY-3-8912 von General Instruments (kurz „8912" genannt), der nicht nur Klänge erzeugen kann, sondern über einen Acht-Bit-Ausgang auch die Tastatur abfragt.

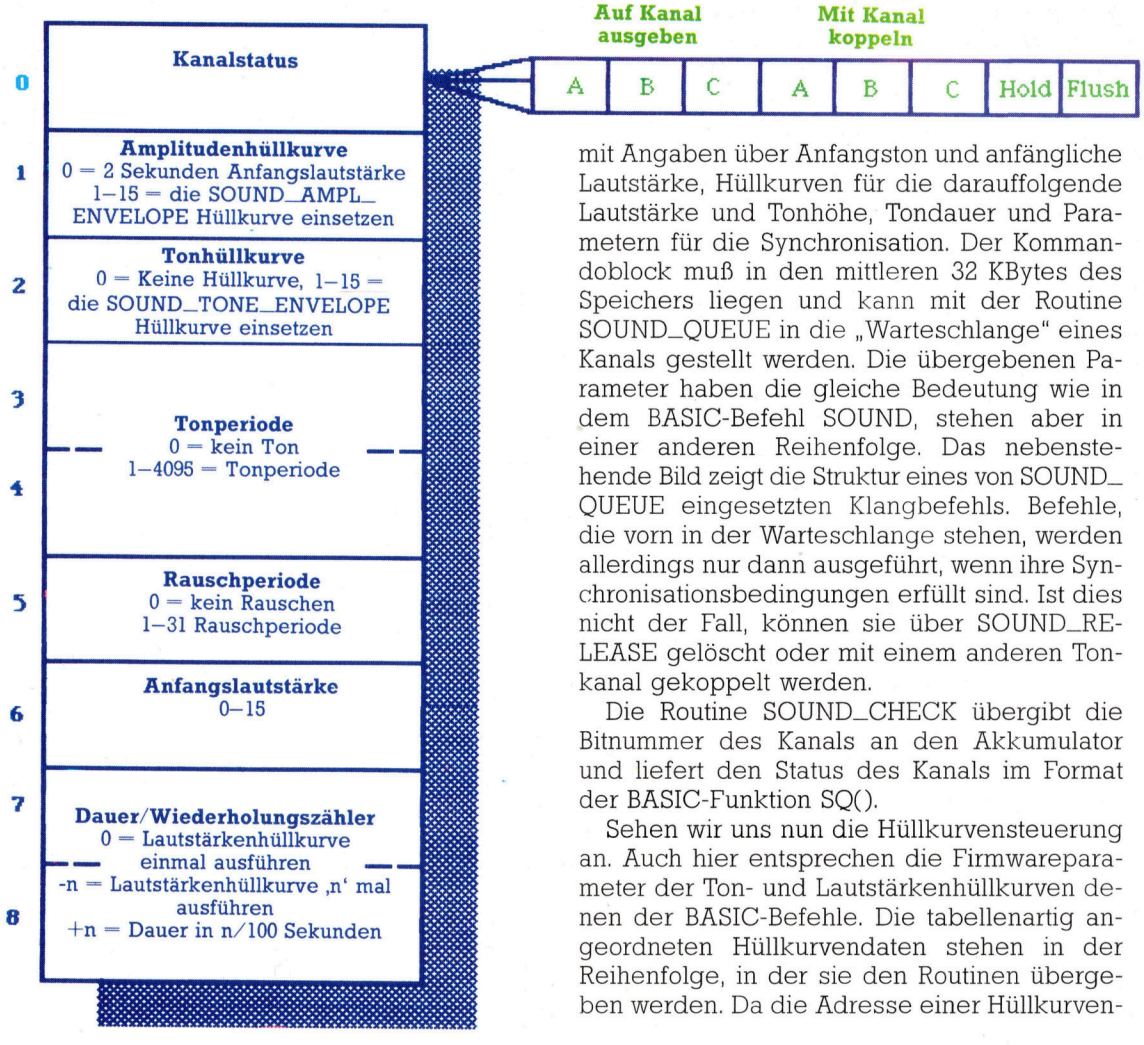
Der 8912 unterstützt drei unabhängige Tonkanäle (A, B und C) und einen Pseudogenerator für Rauschen, der auf jeden Kanal geschaltet werden kann. Der Chip erzeugt Töne, deren Lautstärke er mit vordefinierten Hüllkurven verändern kann. Die Aussage des Tongenerators wird zu Stereoklängen vermischt: Kanal A ist links, Kanal B rechts und Kanal C wird gleichmäßig zwischen beiden aufgeteilt. Diese Grundmöglichkeiten werden noch von dem Betriebssystem verfeinert, das per Software Ton, Lautstärke und Synchronisation steuert.

Bei der Tonerzeugung arbeiten die BASIC-Befehle ENV, ENT, RELEASE, SOUND und SQ direkt mit dem Betriebssystem, so daß Parameter und Konzepte fast völlig übereinstimmen. Das OS steuert Klänge mit einem speziellen Interrupt, der hundertmal pro Sekunde stattfindet und speziell auf die Verarbeitung von Tonbefehlen ausgerichtet ist.

Kommandoblöcke

Dabei werden beispielsweise Kanäle miteinander gekoppelt und Hüllkurven während des Ereignisses aktualisiert. Sehen wir uns zunächst die Klangbefehle und die Arbeitsabläufe genauer an.

Zu jedem Ton gehört ein Kommandoblock



mit Angaben über Anfangston und anfängliche Lautstärke, Hüllkurven für die darauffolgende Lautstärke und Tonhöhe, Tondauer und Parametern für die Synchronisation. Der Kommandoblock muß in den mittleren 32 KBytes des Speichers liegen und kann mit der Routine SOUND_QUEUE in die „Warteschlange" eines Kanals gestellt werden. Die übergebenen Parameter haben die gleiche Bedeutung wie in dem BASIC-Befehl SOUND, stehen aber in einer anderen Reihenfolge. Das nebenstehende Bild zeigt die Struktur eines von SOUND_QUEUE eingesetzten Klangbefehls. Befehle, die vorn in der Warteschlange stehen, werden allerdings nur dann ausgeführt, wenn ihre Synchronisationsbedingungen erfüllt sind. Ist dies nicht der Fall, können sie über SOUND_RELEASE gelöscht oder mit einem anderen Tonkanal gekoppelt werden.

Die Routine SOUND_CHECK übergibt die Bitnummer des Kanals an den Akkumulator und liefert den Status des Kanals im Format der BASIC-Funktion SQ().

Sehen wir uns nun die Hüllkurvensteuerung an. Auch hier entsprechen die Firmwareparameter der Ton- und Lautstärkenhüllkurven denen der BASIC-Befehle. Die tabellenartig angeordneten Hüllkurvendaten stehen in der Reihenfolge, in der sie den Routinen übergeben werden. Da die Adresse einer Hüllkurven-

tabelle jederzeit mit SOUND_T_ADDRESS oder SOUND_A_ADDRESS festgestellt werden kann, lassen sich die Hüllkurven leicht durch Patchen, ohne Aufruf der entsprechenden Firmwareroutinen, ändern.

Das Format dieses Datenblocks haben wir in dem nebenstehenden Diagramm abgebildet. Über den Offset in den Datenblock können Sie jeden beliebigen Bereich patchen. Sie müssen nur darauf achten, daß der Datenbereich während des Patchens nicht auch von den Firmwareroutinen angesprochen wird. Dies stört zwar den Programmablauf nicht, doch können dadurch Töne entstehen, die so nicht geplant waren.

Die Firmwareroutine SOUND_ARM_EVENT hat eine besonders praktische Funktion. Sie legt ein Ereignis an, das ausgelöst wird, wenn die Warteschlange eines angegebenen Kanals leer ist. Die Routine meldet dafür beim Klanginterrupt ein spezielles Ereignis an. Dieses Ereignis prüft zunächst, ob die Warteschlange des entsprechenden Kanals leer ist. Falls ja, wird der von SOUND_ARM_EVENT angelegte Ereignisblock aktiviert und je nach Ereignisklasse verarbeitet.

Der übergebene Ereignisblock wird mit KL_INIT_EVENT intitialisiert und kann jede Klasse oder Priorität annehmen. Da der Klanginterrupt aber alle Hundertstelsekunden stattfindet, genügt normales, asynchrones Ereignis.

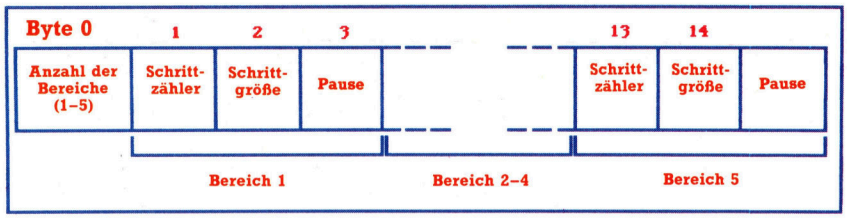
Mit dieser Technik läßt sich beim Ablauf eines Programms eine permanente Hintergrundmusik erzeugen, ohne daß der Tonchip ständig mit neuen Daten versorgt werden muß. Die Ereignisroutine holt sich dabei die nächste Tonprogrammadresse aus dem vordefinierten Datenbereich und ruft dann SOUND_QUEUE auf. Wenn Sie am Ende der Routine wieder an den Anfang verzweigen, erhalten Sie eine ständige Hintergrundbegleitung.

Klangereignis aus

Da SOUND_QUEUE das Klangereignis abschaltet, muß sich die Routine vor ihrem Ende selbst wieder aufrufen, indem sie mit SOUND_ARM_EVENT ihren eigenen Parameterblock übergibt. Das Ablaufdiagramm zeigt, wie die Ereignisroutine mit Hilfe der Warteschlange eine permanente Klangkulisse erzeugt.

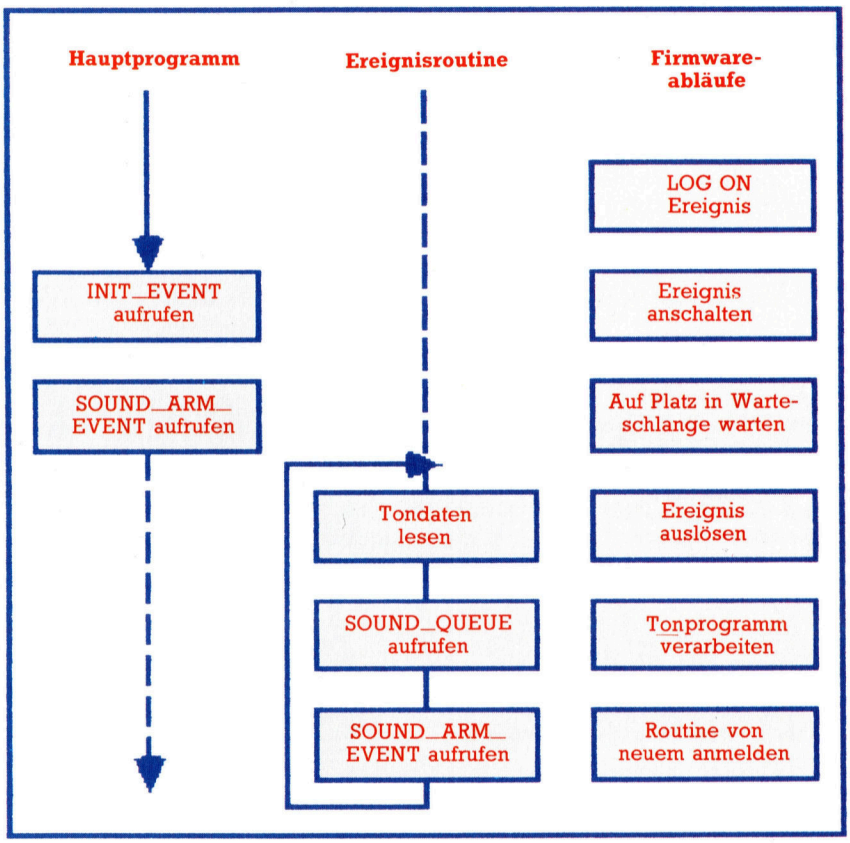
SOUND_HOLD kann die Klangerzeugung jederzeit abstellen. Dabei werden alle aktiven Hüllkurven angehalten und der Ton abgeschaltet. SOUND_CONTINUE, SOUND_QUEUE oder SOUND_RELEASE starten die Musik von neuem. SOUND_RELEASE löst den Start der Klangprogramme aus, die einzeln aufgerufen werden müssen.

Wenn mit der Tastatur eine Unterbrechung veranlaßt wurde, stellt der Aufruf von SOUND_HOLD sicher, daß fehlerhafte Töne nicht ausgeführt werden. So kann kein Mißklang oder eine Disharmonie entstehen.



Jede Hüllkurve kann bis zu fünf Bereiche enthalten. Jeder Bereich besteht aus drei Bytes, die als Schrittzähler dienen und Schrittgröße und Unterbrechungszeiten angeben. Wenn die Lautstärkenhüllkurven der Firmware eingesetzt werden, steht im ersten Byte des Blocks die Hüllkurvennummer (zwischen 128 und 143), und Byte zwei und drei enthalten die Unterbrechungszeit.

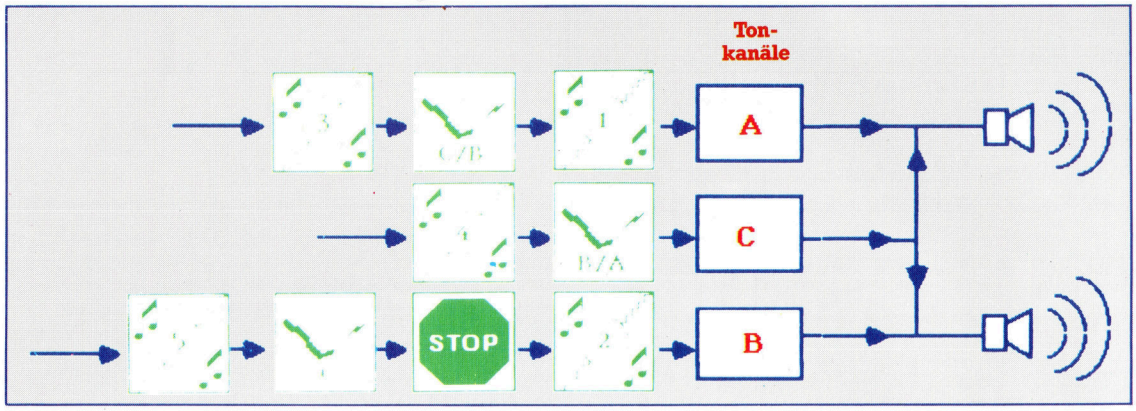
Mit SOUND_ARM_EVENT können Sie ereignisgesteuerte Hintergrundmusik erzeugen, während das Hauptprogramm im Vordergrund weiterläuft. Das Bild zeigt die verschiedenen Abläufe.



Der Routinen der Tonsteuerung

OBCA7H SOUND_RESET	RESET der Tonsteuerung
OBCAAH SOUND_QUEUE	Ton in eine der Warteschlangen stellen
OBCADH SOUND_CHECK	Liefert den Status einer Warteschlange
OBCAOH SOUND_ARM_EVENT	Ermöglicht das Auslösen eines Ereignisses, wenn die Warteschlange leer ist
OBCB3H SOUND_RELEASE	Gibt alle Töne einer Warteschlange frei
OBCB6H SOUND_HOLD	Hält Töne aller Warteschlangen an
OBCB9H SOUND_CONTINUE	Läßt gestoppte Töne weiterlaufen
OBCBCH SOUND_AMPL_ENVELOPE	Initialisiert eine der softwaregesteuerten Hüllkurven für die Lautstärke
OBCBFH SOUND_TONE_ENVELOPE	Initialisiert eine der softwaregesteuerten Hüllkurven für Töne
OBCC2H SOUND_A_ADDRESS	Liefert die Adresse einer Lautstärkenhüllkurve
OBCC5H SOUND_T_ADDRESS	Liefert die Adresse einer Tonhüllkurve

In der hier dargestellten Situation werden gerade die Klänge Eins und Zwei ausgeführt. Nach dem Ende von Klang Zwei wartet der nächste Befehl der Warteschlange auf SOUND_RELEASE. Da der Befehl in Warteschlange C mit den Kanälen A und B gekoppelt werden soll, muß er auf die Befehle dieser Warteschlange warten. Inzwischen ist Klang Eins beendet, so daß A mit C gekoppelt werden kann, doch fehlt noch die Kopplung mit B. Nach Freigabe der Warteschlange B ist die Kopplung möglich, und Klang Drei, Vier und Fünf ertönen gleichzeitig.



Symbole: Tonbefehl (mit Nummer) — Kopplung mit dem angezeigten Kanal — STOP — Warten, bis die Freigabe von SOUND_RELEASE erfolgt

Der Befehl CALL

Der BASIC-Befehl CALL, mit dem Daten als „Parameterblock" an Maschinencoderoutinen übergeben werden, ist in den Handbüchern nicht besonders gut beschrieben. Die vom Anwender gelieferten Parameter werden von der untersten (ersten) Blockadresse an aufwärts abgelegt. Beim Einsprung in die mit CALL aufgerufene Routine zeigt IX auf diese Adresse.

Numerische Werte können direkt übergeben werden, nicht aber Variable. Statt dessen wird die Adresse der Variablen (bei Stringvariablen die Adresse des String-Steuerblocks) mit dem Symbol @ übergeben.

Das folgende Beispiel ruft eine Routine bei &A000 auf, übergibt ihr den Wert 85 und enthält bei der Rückkehr ins BASIC die Resultate in Ergebnis% und Antwort$:

```
CALL &A000,85,@Ergebnis%,@Antwort$
```

Beim Einsprung in die Routine zeigt IX auf einen ROM-Block, der zuerst den Wert 85 als 16-Bit-Ganzzahl ohne Vorzeichen enthält, dann zwei Bytes mit der Adresse von Ergebnis% und schließlich die Adresse des Stringsteuerblocks für Antwort$. Der Stringsteuerblock besteht aus drei Bytes, in denen zunächst die Stringlänge und dann die Adresse stehen. Dieser Block sollte möglichst nicht geändert werden, da sonst Probleme mit der Firmware entstehen könnten.

Das Editieren von Hüllkurven kann recht kompliziert werden. Mit unserem Editor für Hüllkurven können Sie Ihre Parameter eingeben und auf dem Bildschirm gleich die Form der dazugehörigen Hüllkurve sehen.

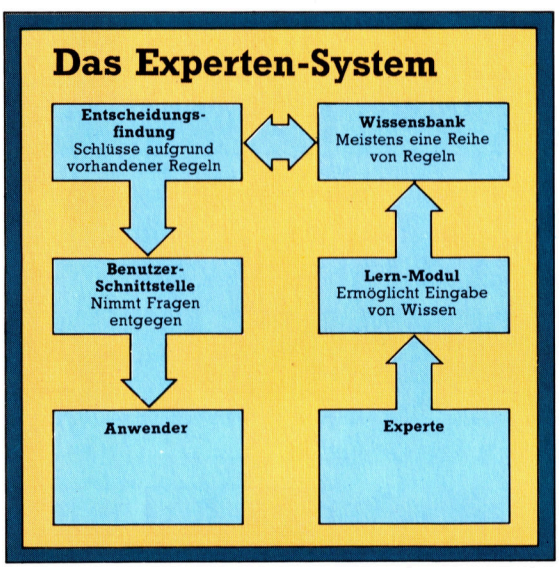
Editor für Hüllkurven

```
50 MEMORY &6FFF: DIM envelope(15)
60 done=0: routine=&7000: buffer=routine+&100: nk=1
70 WHILE -1
80  MODE 2: LOCATE 1,20: INPUT "Start volume (0-15) ";sv%: I
F sv%)15 OR sv%(0 THEN 80
90 LOCATE 1,21: INPUT "Start tone ";tt: IF tt(0 OR tt)4095 T
HEN 90
100 LOCATE 1,22:INPUT "Horizontal scale (1-10) ";sc%: IF sc%
)10 OR sc%(1 THEN 100
110 LOCATE 1,23:INPUT "Volume or tone envelope (V/T) ";senv$
: IF (UPPER$(senv$)()"V") AND (UPPER$(senv$)()"T") THEN GOTO
 110
120 VV=0: IF UPPER$(senv$)="V" THEN pk=&BC: VV=1:  ELSE pk=&
BF
130  length%=0: soundx=0: IF VV=1 THEN soundy=INT(sv%*9.7) E
LSE soundy=68
140  FOR x=1 TO 15
150   envelope(x)=0
160  NEXT
170  GOSUB 270
180  LOCATE 1,20: PRINT CHR$(18)
190  FOR count=1 TO 5
200   GOSUB 360: nk=ABS(nk-1): GOSUB 540: IF VV=1 THEN GOSUB
 880 ELSE GOSUB 890
210  NEXT
220  WHILE NOT done
230   LOCATE 5,20: INPUT "Edit (y/n) ";ed$
240   IF UPPER$(ed$)="N" THEN done=-1 ELSE GOSUB 790
250  WEND
260 WEND
270 REM intialise screen
280 MODE 1: WINDOW #1,1,39,1,2: GOSUB 1000
290 MOVE 20,380: DRAW 20,210,3: DRAW 600,210,3
300 MOVE 21,soundy+211
310 POKE routine, &3E: POKE routine+1,1
320 POKE routine+2, &21: POKE routine+3, buffer-(INT(buffer/
256))*256: POKE routine+4, INT(buffer/256)
330 POKE routine+5,&C3: POKE routine+6,pk: POKE routine+7,&B
C
340 POKE routine+8,&C9
350 RETURN
360 REM get section data
370 LOCATE 1,20: PRINT CHR$(18);: LOCATE 5,20: PRINT "Sectio
n ";count
380 LOCATE 1,22: PRINT CHR$(18);: LOCATE 5,22: INPUT "Step c
ount "; skip%: IF VV=0 THEN GOTO 400 ELSE IF skip%(0 OR skip
%)127 THEN 380
390 GOTO 410
400 IF skip%(0 OR skip%)239 THEN GOTO 380
410 P$=STR$(SKIP%): GOSUB 900
420 LOCATE 1,23: PRINT CHR$(18);: LOCATE 5,23: INPUT "Step s
ize ";size%: IF VV=0 THEN GOTO 440 ELSE IF ABS(size%))15 THE
N 420
430 GOTO 450
440 IF ABS(size%))127 THEN 420
450 P$=STR$(size%): GOSUB 900
460 ss=size%: IF size%(0 THEN size%=256-ABS(size%)
470 LOCATE 1,24: PRINT CHR$(18);: LOCATE 5,24: INPUT "Pause
time "; pause%: IF pause%)255 THEN 470
480 P$=STR$(pause%): GOSUB 900
490 envelope ((count-1)*3+1)=skip%: envelope ((count-1)*3+2)
=size%: envelope ((count-1)*3+3)=pause%
500 FOR x=20 TO 24
510  LOCATE 1,x: PRINT CHR$(18);
520 NEXT
530 RETURN
540 REM envelope intialisation
550 POKE buffer,count
560 CLS #1: GOSUB 1000
570 FOR x=0 TO count-1
580 POKE buffer+(x*3+1),envelope(x*3+1): skip%=envelope(x*3+
1)
590 p$=STR$(envelope(x*3+1)): GOSUB 900
600 POKE buffer+(x*3+2),envelope(x*3+2): size%=envelope(x*3+
2)
610 pp=envelope(x*3+2): IF pp)127 THEN pp=pp-256
620 p$=STR$(pp): GOSUB 900
630 POKE buffer+(x*3+3),envelope(x*3+3): pause%=envelope(x*3
+3)
640 p$=STR$(envelope(x*3+3)): GOSUB 900
650 NEXT x
660 length%=length%+skip%*pause%
670 CALL routine
680 IF VV=0 THEN GOSUB 910: RETURN
690 off%=size%*10: IF size%)127 THEN off%=(size%-256)*10
700 FOR x=1 TO skip%
710  soundy=(soundy+off%) MOD 150: IF soundy(0 THEN soundy=1
50-soundy
720   FOR c=1 TO (10/51)*pause%+2: soundx=soundx+(sc%)
730    DRAW soundx+21,soundy+211,2+nk
740   NEXT
750 NEXT
760 DRAW soundx+21,soundy+211,2+nk
770 RETURN
780 REM redraw graph
790 LOCATE 5,20: PRINT CHR$(18);: INPUT "Section";sec%: IF s
ec%(1 OR sec%)5 THEN 790
800 count=sec%: GOSUB 360
810 CLS: soundx=0: soundy=INT(sv%*9.7): length%=0: GOSUB 270
820 FOR section=1 TO 5
830   count=section: GOSUB 540
840 NEXT
850 GOSUB 870
860 RETURN
870 REM sound
880 SOUND 1,tt,length%,sv%,1: RETURN
890 SOUND 1,tt,length%,sv%,,1: RETURN
900 PRINT #1,MID$(P$,(2-ABS(VAL(p$)(0)));",";: RETURN
910 FOR x=1 TO skip%
920  soundy=soundy-((15/127)*ss): IF soundy(0 THEN soundy=0
930 IF soundy)189 THEN soundy=189
940  FOR c=1 TO (7/51)*pause%+2: soundx=soundx+sc%
950   DRAW soundx+21,soundy+211,2+nk
960  NEXT
970 NEXT
980 DRAW soundx+21,soundy+211,2+nk
990 RETURN
1000 IF VV=1 THEN PRINT #1, "ENV 1, "; ELSE PRINT #1, "ENT 1
, ";
1010 RETURN
```

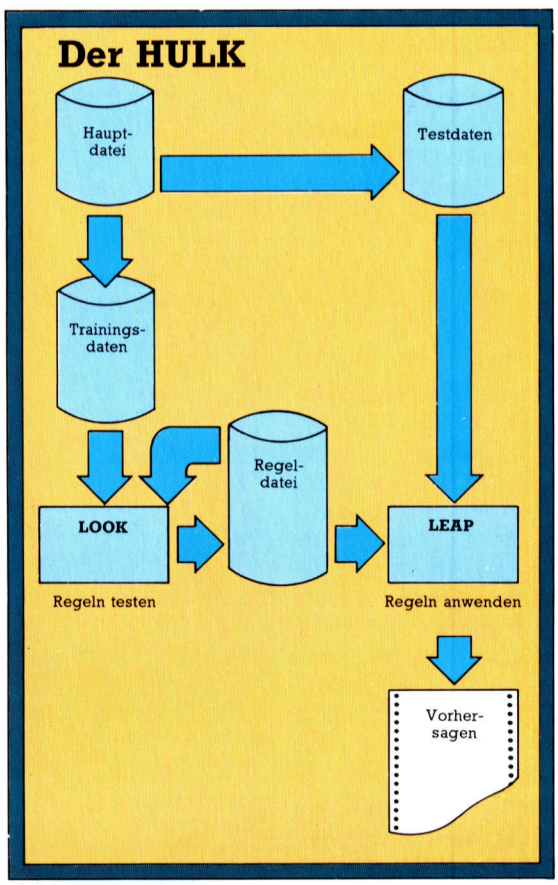
Das Hulk Experten-System

Der Begriff „Ingenieurwesen" unterzieht sich einem Bedeutungswandel. Während Ingenieure des 19. Jahrhunderts Eisen und Stahl umformten und zu Schiffen und Brücken verarbeiteten, bezeichnet das Ingenieurwesen heute eine Vielzahl von Berufen, deren Aufgabe es auch ist, Wissen zu verarbeiten.

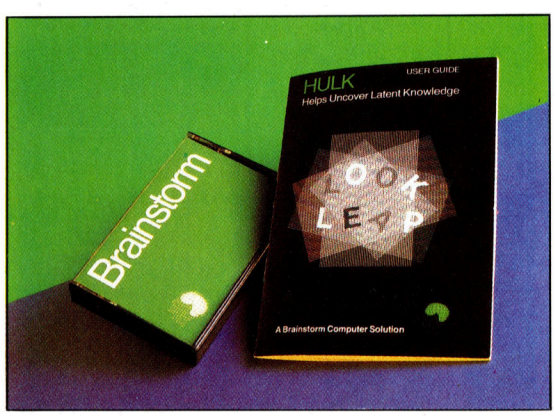
Ein „Wissens-Ingenieur" weiß, wie Experten-Systeme entwickelt werden. Experten-Systeme (ES) beinhalten organisiertes Wissen über ein Fachgebiet menschlicher Sachkenntnis, ein Beispiel ist eine medizinische Diagnose. Der Arzt hat ein umfangreiches, faktisches Wissen über Krankheiten, ihre Anzeichen und Symptome. Seine Sachkenntnis besteht aus der Fähigkeit, die Symptome eines Patienten und die Beschreibung in medizinischen Fachbüchern zueinander in Bezug zu bringen. Dies geschieht auch unter Berücksichtigung zurückliegender Erfahrungen mit der vermuteten Krankheit. Je besser der Arzt im Kombinieren von Fachbuchwissen und aktuellen Beobachtungen ist, desto besser kann seine Diagnosetechnik sein.



Entscheidende Faktoren sind die Fähigkeit, sich an organisierte Daten zu erinnern, daraus Muster zu bilden und beobachtete Fälle mit diesen Mustern zu vergleichen, auch wenn Daten unvollständig sind oder nicht zu vorangegangenen Fällen passen. Die ersten beiden Faktoren, Organisation und Klassifizierung von Daten, sind Arbeitsbereiche des Computers, der letzte Faktor erfordert einen menschlichen Experten. Könnte ein Computersystem statistische Analysen verwenden, um das Gefühl und den Spürsinn des menschlichen Experten zu simulieren, wäre es durch seine Datenorgani-



In einem traditionellen Experten-System verwendet die Entscheidungsfindung eine Reihe von IF... THEN-Regeln aus einer Wissensbank für Antworten auf Fragen, die der Anwender eingibt. Das Wissen, auf dem diese Regeln basieren, wird von Experten über das Lernmodul eingegeben. Mit HULK wird ein Satz von Entscheidungsregeln aus einer Hauptdatei erstellt, gestützt durch Beobachtungen, die der Anwender eingibt. Dafür sind zwei Programme erforderlich: LOOK, das einen Trainings-Datensatz zum Ausprobieren der Regeln benötigt, und einen weiteren Datensatz zum Prüfen auf Anwendbarkeit. LEAP benutzt diese Regeln, um anhand der Daten Wahrscheinlichkeitsberechnungen zu erstellen.



Ursprünglich wurde HULK von Richard Forsyth als praktisches Projekt für seine Studenten am Polytechnikum Nord-London entwickelt. Obwohl die Erstellung des Programms nur zwei Wochen in Anspruch nahm, vergingen weitere sechs Monate, bis es soweit verfeinert war, daß es als anwenderfreundlich gelten konnte.

sationsfähigkeit in der Lage, den Menschen zu übertreffen.

Am wichtigsten sind diese Systeme dort, wo es, wie in der medizinischen Diagnose, keine festgelegten Entscheidungskriterien gibt und menschliches Urteilsvermögen unabdingbar ist. Da Wissen, oft in Form von Regelsammlungen, unverzichtbar für den Betrieb dieser Systeme ist, werden sie auch als „rule-based" (auf Regeln basierend) oder „knowledge-based" (auf Wissen basierend) bezeichnet. Experten-Systeme haben sich auf verschiedenen Gebieten als erfolgreich erwiesen.

Experten-Systeme

Schon jetzt können sie ausgebildete Fachleute in der medizinischen Diagnose, der Suche nach Rohstoffen und vielen anderen Gebieten übertreffen. Aufgrund dieser Erfolge finden wir diese ES nicht mehr nur in Forschungslaboren für Künstliche Intelligenz, sondern auch bei unterschiedlichen Computeranwendungen. Daraus ergeben sich Konsequenzen für die Konzeption und Konstruktion solcher Maschinen. Ihr Einsatz bei der Softwareentwicklung ersetzt den traditionellen Grundsatz
Daten + Algorithmen = Programm

durch die Methodik
Wissen + Folgerungen = System

– ein evolutionärer Fortschritt mit weitreichenden Konsequenzen.

Ein Experten-System basiert auf einem „Wissensgerüst". Daneben stehen aber drei weitere Komponenten, die einem ES volle Arbeitsfähigkeit verleihen: Die Entscheidungsfindung, die neue Regeln zum Interpretieren von Daten auf der Basis existierender Regeln und Daten entwirft; das Lernmodul, das aus „eigenen Erfahrungen" oder denen des menschlichen Experten neues Wissen zusammenstellt, und schließlich die Schnittstelle zum Benutzer, über die auch ein Nicht-Experte das System befragen und benutzen kann. Betrachten wir die beiden Kernmodule, die Entscheidungsfindung und das Lernmodul.

Eine Wissensbasis enthält Fakten (oder Behauptungen) und Regeln. Fakten können sich jedoch sehr schnell ändern (z. B. während einer Konsultation). Regeln sind die Langzeitinformationen darüber, wie man aus Wissen neue Fakten oder Hypothesen entwickelt. Doch was unterscheidet dies von konventionellen Datenbanken? Die Wissensbank ist kreativ. Die Fakten in einer Datenbank sind normalerweise passiv: Sie sind einfach vorhanden und werden, ganz nach Bedarf des Benutzers, archiviert und abgerufen. Dagegen versucht die Wissensbank, unabhängig von den Fragen des Benutzers, fehlende Informationen aufzufüllen, unter Berücksichtigung des vorhandenen Wissens.

Wettervorhersage

Wir ließen **HULK** mit dieser Datei laufen und wurden nach Hypothesen über die Daten befragt. Die Hypothese lautete: **TOMORROW=1** womit gemeint ist, daß wir an solchen Mustern interessiert sind, deren letzte Variable den Wert Eins hat – anders ausgedrückt, Tage, an denen es am darauffolgenden Tage regnete. Wir wollen mit HULK-Regeln arbeiten, die es uns ermöglichen, vom Wetter des heutigen Tages Rückschlüsse auf das Wetter von morgen zu ziehen.
Die Wahrscheinlichkeit, daß Regen zu erwarten ist, wird gemessen und in einer Regel verarbeitet, die später mit anderen Regeln Anwendung findet.
HULK schlägt vor, ob die Regeln in die Regeldatei aufgenommen werden soll oder nicht. Die Entscheidung darüber liegt beim Anwender.
Wir fanden schnell drei Regeln, um für morgen Regen vorherzusagen:
1) Wenn der Niederschlag heute über 2 mm liegt
UND
2) Wenn die Sonne heute weniger als 3,5 Stunden scheint
UND
3) Wenn der Unterschied zwischen der höchsten und der tiefsten Temperatur weniger als sechs Grad beträgt
DANN
können Sie mit 83%iger Sicherheit davon ausgehen, daß es morgen regnet und Sie naß werden.

Hier ein Beispiel zur Datenorganisation:

```
1 WEATHER,30,5
```

Zeile 1 beschreibt die Datei und beinhaltet den Namen (WEATHER), die Anzahl der Datenmuster (30) und die Anzahl der Daten per Muster (5). Die Datei besteht aus den Wetterdaten eines Monats in London. Zu jedem Tag sind aufgeführt: Tiefst- und Höchsttemperatur, Niederschlag (in mm), Sonnenschein-Stunden und eine Boolsche Variable, deren Wert Eins ist, wenn es am folgenden Tag regnete, und Null, wenn es trocken blieb.

```
100 MINIMUMT
200 MAXIMUMT
300 RAINFALL
400 SUNSHINE
500 TOMORROW
```

Die Zeilen 100 bis 500 geben die Namen der Muster-Daten an.

```
1001 D01,54,110,175,32,1
1002 D02,42,125,041,62,1
1003 D03,76,112,077,11,1
1004 D04,27,105,018,43,0
1005 D05,30,120,000,95,0
1006 D06,44,106,000,55,0
1007 D07,48,094,000,51,1
1008 D08,68,092,055,48,1
1009 D09,64,102,048,41,1
     --------"-------
1028 D28,58,154,000,20,1
1029 D29,67,088,064,42,0
1030 D30,45,096,000,68,1
```

Die eigentlichen Daten speichern die Zeilen 1001 bis 1030, jeweils ein Muster per Zeile, beginnend mit einer Identifizierung. Da HULK nur mit Ganzzahlen arbeiten kann, wurden alle Werte mit 10 multipliziert.

Zeile 1001 lautet eigentlich 1001 D01,5.4,11.0,17.5,3.2,1 – was für alle Datenzeilen gilt.

TOMORROW = 1
ASKING FOR RULE [1]

RULE IS: **RAINFALL > 20**

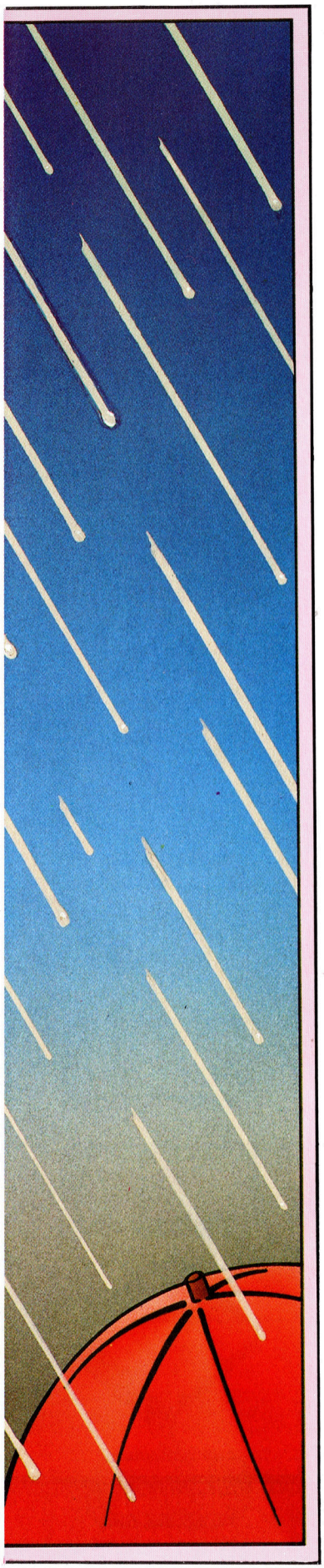
CONTINGENCY TABLE	SUCCESS	FAILURE
RULE TRUE	12	2
RULE FALSE	5	11

SUCCESS RATE = **76.3%**
BITS PER SAMPLE

BEFORE	**RAINFALL > 20**	**1.00**
AFTER	**RAINFALL > 20**	**0.85**

YOU ARE ADVISED TO [KEEP] THE RULE OK
RULE HAS BEEN [ADDED] TO THE RULE SET

Bits pro Muster
Ergebnis, bei wievielen Datenmustern die Regel zutrifft.



Regeln im „IF . . . THEN"-Format sind die bevorzugte Methode, Wissen „über den Daumen zu peilen". Hierzu ein Beispiel:
IF die Heimmannschaft das letzte Heimspiel verlor,
AND die Gesamtmannschaft ihr letztes Heimspiel mit zwei Toren gewann,
THEN wird die Wahrscheinlichkeit eines Unentschieden mit 1088 multipliziert.

Diese Regeln sind nicht im Programm enthalten, sondern sind Daten der Entscheidungsfindung. Es gibt zwei Hauptstrategien, um Schlußfolgerungen zu ziehen: Vor- und Rückwärtsverkettungen. Die Vorwärtsverkettung untersucht Daten, um daraus Hypothesen zu erstellen, hingegen versucht die Rückwärtsverkettung Daten zu finden, um eine erstellte Hypothese zu beweisen oder zu widerlegen. Reine Vorwärtsverkettung führt zu nicht genau definierten „WAS, WENN . . ."-Abfragen des Systems, während die Rückwärtsverkettung zielstrebig vorgeht.

Die erfolgreichsten Systeme benutzen eine Mischung aus beidem. Ob die Entscheidungsfindung vor- oder rückverkettet, sie muß mit „unsicheren" (teilweise noch nicht bewiesenen) Daten arbeiten. Computerexperten versuchten immer wieder, unsere Welt in den starren Abgrenzungen des Computers zu erfassen und gingen damit an der Realität vorbei. Die ES-Forschung ermöglicht eine präzise Arbeit mit unsicheren Daten – eine stärkere Annäherung an die Wirklichkeit, als die idealisierten Abstraktionen unserer Datensysteme es erlaubten.

Tatsächlich gibt es zu viele Wege, mit Unsicherheiten zu arbeiten. Da sind Fuzzy Logic, Bayesian Logic, Multi-Value Logic und Certainty Factors, um nur vier davon zu nennen.

„Wenn X wahr ist, . . ."

Sie ersetzen die Sicherheit von „WENN X WAHR IST, DANN IST Y WAHR" durch die vorsichtige, statistische Schlußfolgerung „FALLS X IN 65% ALLER FÄLLE WAHR IST, DANN LIEGT DIE WAHRSCHEINLICHKEIT FÜR Y BEI 50 BIS 70%". Das Merkwürdige ist, daß viele der ausprobierten Schemata scheinbar funktionieren. Eine mögliche Erklärung hierfür mag darin liegen, daß die Organisation von Wissen mehr Gewicht hat als dessen bloße Anhäufung. Die meisten Wissensbanken verbinden ihre Redundanz, um dem ES verschiedene Möglichkeiten der Entscheidungsfindung zu bieten. Die Wahrscheinlichkeitswerte dienen dabei nur als Kriterium, um zwischen zwei Bewertungen zu wählen.

Es gibt bereits Programmpakete, die mit Wissensbanken arbeiten. Eines der wenigen, auch für den Heimcomputer-Enthusiasten erschwingliche, ist HULK (Helps Uncover Latent Knowledge – wörtlich: Hilft latentes Wissen aufdecken), im Vertrieb von Brainstorm Computer Solutions. Es läuft auf dem Acorn B und den Torch-Computern.

HULK ermöglicht dem Benutzer, eine Reihe von Entscheidungsregeln aufzustellen und zu testen, um sie später für Voraussagen und Klassifikationen zu nutzen. Sie können mit HULK, aus verschiedenen Fällen oder Beispielen unterschiedlicher Bewertung, Muster und Regelmäßigkeiten bilden und diese für vorhersagende Zwecke einsetzen. Nehmen wir an, ein Landwirt speichert detaillierte Messungen über Höhe, Blattfarbe usw. einiger hundert Zuckerrüben in einem Computer und fügt die Anzahl der vor der Ernte erkrankten und gesunden Pflanzen hinzu. Das ES könnte nun Regeln in bezug auf die charakteristischen Merkmale der Pflanzen und ihren Gesundheitszustand entwickeln.

Erntevorhersage

Nach diesen Regeln könnten später gefährdete Pflanzen identifiziert und Vorhersagen über die Ernte des nächsten Jahres verbessert werden. Der Landwirt muß diese Regeln nicht genau kennen, doch das System könnte sie auf die eingegebenen Daten der Pflanzen anwenden. Anwendungsmöglichkeiten für HULK sind reichlich vorhanden.

Das HULK-Paket besteht aus zwei Hauptprogrammen: LOOK (Logical Organiser Of Knowledge – logischer Wissensorganisator) und LEAP (Likelihood Estimator And Predictor – Wahrscheinlichkeitsbewerter und Voraussager). Diese ermöglichen dem Benutzer Regeln aufzustellen, die auf den Daten früherer Beobachtungen beruhen, diese auf andere, unvollständige Datensätze anzuwenden und so beispielsweise Vorhersagen über ein Fußball-Match am nächsten Spieltag zu treffen. Die Regelsammlung ist die Wissensbank, die wiederum aus gesammelten Daten besteht. In HULK sind das Wahrscheinlichkeits-Entscheidungsregeln, die für Klassifikationen und Vorhersagen benutzt werden können.

Zum Arbeiten mit LOOK benötigen Sie einen oder vorzugsweise zwei Datensätze (ein großer Datensatz kann hierfür in zwei kleinere aufgeteilt werden). Der eine ist der ‚Trainingssatz', um Regeln auszuprobieren, der andere ist der ‚Testsatz', um die Anwendbarkeit der Regeln bei unbekannten Daten zu bestätigen. Sobald die Daten erfaßt sind, legt LOOK Ihnen neugefundene Regeln einzeln zur Begutachtung vor. Es prüft jede Regel anhand der Trainingsdaten und teilt Ihnen mit, um wieviel, falls überhaupt, sich der Vorhersageerfolg mit den bereits vorhandenen Regeln verbessert. Daraufhin empfiehlt es, die neue Regel zu übernehmen oder zu verwerfen. Die endgültige Entscheidung liegt beim Anwender.

LOOK ist kein richtiges Lernprogramm, eher eine Art Filter, der nur nützliche Mußmaßungen oder Aussagen durchläßt.

Überkauft

Im vorhergehenden Teil dieser 17+4 Serie beschrieben wir die Routine zur Berechnung eines Blattes. Die Routine dieser Folge beschließt die Runde des Spielers, analysiert die Karten und bereitet die Runde des Computers vor.

Eine universell anwendbare Routine, die das Blatt des Spielers auswertet und eine Variable, EF, entsprechend der ermittelten Blattkategorie setzt, ist vorhanden. Mit diesem Programm wird das Blatt des Spielers ausgespielt.

In diesem Stadium des Spieles sind an den Spieler und die Bank jeweils zwei Karten ausgegeben worden. Der Spieler kann entweder „sticken" oder „twisten", um näher an 21 Punkte heranzukommen, ohne zu überreizen. Um in diesem Spiel der Bank einen kleinen Vorteil zu verschaffen, wird die Regel aufgestellt, daß Sie nicht sticken dürfen, wenn Ihr Blatt weniger als 17 Punkte zählt.

Zeile 120 ruft das Twist/Stick-Unterprogramm von der Hauptschleife auf. Das Unterprogramm ab Zeile 2600 ist nicht selbst für das Twisten oder Sticken zuständig, ruft aber ein anderes Unterprogramm in Zeile 2700 auf, das diese Arbeit durchführt. Bei der Rückkehr von

Auswertung der Karten

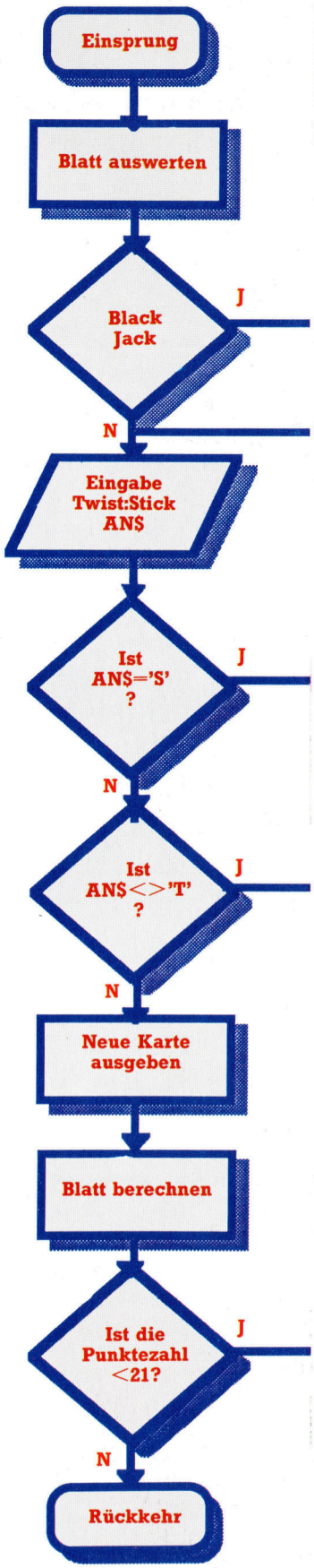
Acorn B

```
 120 GOSUB 2600
 2600 REM
 2610 GOSUB 2700
 2620 ON EF GOSUB 3500,3600,3700,3800,39
00
 2630 RETURN
 2700 REM
 2710 GOSUB 800:IF EF=2 OR EF=3 THEN RET
URN
 2720 GOSUB 700:PRINT"TWIST/STICK/DOUBLE
";
 2725 AN$=GET$
 2727 IF AN$="S" THEN GOSUB 2800:IF CS=0
 THEN RETURN
 2730 IF AN$="S" AND CS=1 THEN 2700
 2750 IF AN$<>"T" THEN 2700
 2760 FL=0:PL=1:GOSUB 1300
 2770 GOSUB 800:IF EF=1 THEN 2700
 2780 RETURN
 2800 REM
 2810 CS=1:GOSUB 800
 2820 IF (TT(PL,2)>17 AND TT(PL,2)<22) O
R TT(PL,1)>=17 THEN CS=0:RETURN
 2825 GOSUB 700:PRINT"YOU CAN'T STICK UN
DER 17"
 2830 FOR DL=1 TO 5000:NEXT DL
 2840 RETURN
 3500 REM **** LESS THAN 21 ****
 3505 PV=1:PS=TT(1,2):IF PS>21 THEN PS=T
T(1,1)
 3510 GOSUB 700:PRINT"LESS THAN 21":FOR
DL=1 TO 500:NEXT DL:RETURN
 3600 REM **** ROYAL PONTOON ****
 3605 PV=4
 3610 GOSUB 700:PRINT"ROYAL PONTOON":RET
URN
 3700 REM
 3705 PV=2
 3710 GOSUB 700:PRINT"PONTOON":RETURN
 3800 REM
 3805 PV=0
 3810 GOSUB 700:PRINT"BUST":RETURN
 3900 REM
 3905 PV=3
 3910 GOSUB 700:PRINT"FIVE CARD TRICK":R
ETURN
```

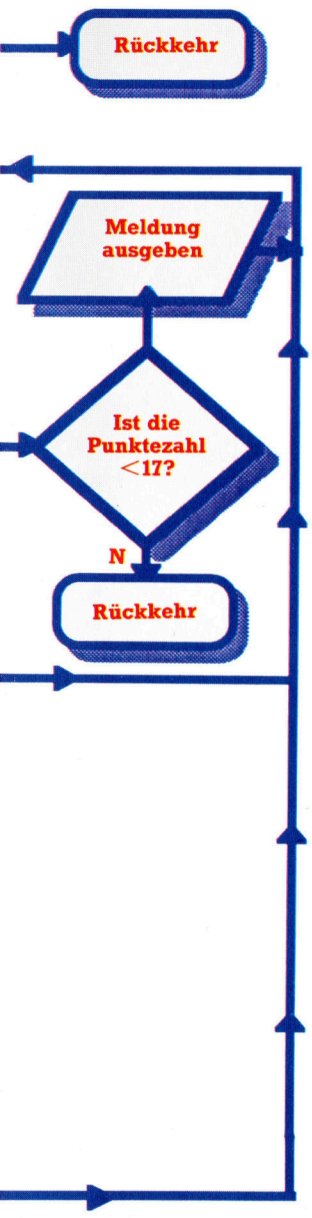
Schneider CPC

```
120 GOSUB 2600:REM tw
ist etc
2600 REM **** punter twist etc ****
2610 GOSUB 2700:REM do it!
2620 ON ef GOSUB 3500,3600,3700,3800,390
0
2630 RETURN
2700 REM **** twist/stick/double ****
2710 GOSUB 800:IF ef=2 OR ef=3 THEN RETU
RN:REM check pontoon/royal pontoon
2720 GOSUB 700:PRINT "Twist/Stick/Double
";
2725 an$="":WHILE an$="":an$=INKEY$:WEND
2727 IF an$<>CHR$(13) THEN PRINT an$
2730 IF an$="s" THEN GOSUB 2800:IF cs=0
THEN RETURN
2733 IF an$="s" AND cs=1 THEN 2700:REM c
an't stick
2750 IF an$<>"t" THEN 2700:REM input err
or
2760 fl=0:pl=1:GOSUB 1300:REM deal
2770 GOSUB 800:IF ef=1 THEN 2700:REM aga
in?
2780 RETURN
2800 REM **** stick ****
2810 cs=1:GOSUB 800:REM evaluate
2820 IF (tt(pl,2)>=17 AND tt(pl,2)<22)OR
 tt(pl,1)>=17 THEN cs=0:RETURN
2825 GOSUB 700:PEN red:PRINT "You can't
stick under 17"
2830 FOR dl=1 TO 1000:NEXT dl
2840 RETURN
3500 REM **** less than 21 ****
3505 pv=1:ps=tt(1,2):IF ps>21 THEN ps=tt
(1,1)
3510 GOSUB 700:PRINT "Less than 21":RETU
RN
3600 REM **** royal pontoon ****
3605 pv=4
3610 GOSUB 700:PRINT "Royal pontoon":RET
URN
3700 REM **** pontoon ****
3705 pv=2
3710 GOSUB 700:PRINT "Pontoon":RETURN
3800 REM **** bust ****
3805 pv=0
3810 GOSUB 700:PRINT "Bust":RETURN
3900 REM **** five card trick ****
3905 pv=3
3910 GOSUB 700:PRINT "Five card trick":R
ETURN
```

Blattschuß



Das Flußdiagramm zeigt, wie der Programmteil des Spielers kontrolliert wird. Ein Verlassen der Kontrollstruktur ist nur unter drei Bedingungen möglich: wenn die Karten des Spielers ein As und eine Zehn oder ein As und eine Bilderkarte zeigen, der Spieler die Option Stick wählt oder das Blatt überreizt wird.



diesem Unterprogramm ist Ihr Blatt komplett und EF entsprechend einer der fünf möglichen Blattkategorien gesetzt. Mit dem ON...GOSUB-Befehl verzweigt das Programm unter Berücksichtigung des Wertes in EF in eine weitere Unterroutine. Diese gibt den Wert des Blattes aus und setzt die Variable PV. Diese Variable wird später, bei der Auswertung des Blattes der Bank, benutzt, um den Wert Ihres Kartenblattes zu speichern.

Das Twist/Stick-Unterprogramm beginnt bei Zeile 2700 mit dem Aufruf der Berechnungsroutine. Die beiden bisher ausgegebenen Karten werden auf den Wert (As und Bilderkarte oder As und Zehn) überprüft. Wenn keine dieser Kombinationen vorliegt, fordert das Programm Sie auf, S oder T zu tippen.

Durch die Verwendung von GET/INKEY$/GET$ anstelle von INPUT brauchen Sie nur die T- oder S-Taste zu drücken, ohne die Eingabe mit RETURN abzuschließen. Der Nachteil dieser Abfrage ist, daß mit GET/INKEY$/GET$ keine Eingabeaufforderung auf dem Bildschirm erscheint. Wir müssen also eine Zeile für die Darstellung hinzufügen.

Wenn Sie an dieser Stelle S für sticken eingeben, prüft das Programm, ob das Blatt weniger als 17 Punkte zählt. Der Programmteil ab Zeile 2800 benutzt dafür das Ergebnis der Auswertungsroutine und gibt eine Meldung aus, wenn Ihre Punktezahl geringer als 17 ist. Die Variable CS wird in diesem Fall auf Eins gesetzt. Bei der Rückkehr in die Twist/Stick-Routine entscheidet der Wert in CS, ob Sie stikken dürfen und die Routine damit beendet werden kann oder nicht. Falls Ihre Eingabe nicht korrekt war, verzweigt das Programm wieder zum Beginn der Routine.

Bei Anwahl der Twist Option wird Ihrem Blatt eine weitere Karte zugeteilt und die neue Punktezahl berechnet. Wenn Ihr Blatt mit der neuen Karte noch unter 21 Punkten liegt (signalisiert durch EF=1), beginnt die Routine von vorn und fragt nach der nächsten Eingabe.

Sinclair Spectrum

```
120 GO SUB 2600: REM TWIST ETC
2600>REM **** PUNTER TWIST ETC EVALUATE
****
2610 GO SUB 2700: REM DO IT!
2620 GO SUB (EF*100)+3400
2630 RETURN
2700 REM **** TWIST/STICK/DOUBLE ****
2710 GO SUB 800: IF EF=2 OR EF=3 THEN  R
ETURN : REM CHECK PONTOON/ROYAL PONTOON
2720 GO SUB 700: PRINT "TWIST/STICK/DOUB
LE ";
2725 LET A$=INKEY$: IF A$="" THEN  GO TO
 2725
2727 IF A$<>CHR$ 13 THEN  PRINT A$
2730 IF A$="S" THEN  GO SUB 2800: IF CS=
0 THEN  RETURN
2733 IF A$="S" AND CS=1 THEN  GO TO 2700
: REM CAN'T STICK
2750 IF A$<>"T" THEN  GO TO 2700: REM IN
PUT ERROR
2760 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL
2770 GO SUB 800: IF EF=1 THEN  GO TO 270
0: REM AGAIN?
2780 RETURN
2800 REM **** STICK ****
2810 LET CS=1: GO SUB 800: REM EVALUATE
2820 IF (T(PL,2)>=17 AND T(PL,2)<22) OR
T(PL,1)>=17 THEN  LET CS=0: RETURN
2825 GO SUB 700: PRINT  FLASH 1;"YOU CAN
'T STICK UNDER 17!"
2830 FOR L=1 TO 300: NEXT L
2840 RETURN
3500 REM LESS THAN 21
3505 LET PV=1: LET PS=T(1,2): IF PS>21 T
HEN  LET PS=T(1,1)
3510 GO SUB 700: PRINT "LESS THAN 21": R
ETURN
3600 REM **** ROYAL PONTOON ****
3605 LET PV=4
3610 GO SUB 700: PRINT "ROYAL PONTOON":
RETURN
3700 REM **** PONTOON ****
3705 LET PV=2
3710 GO SUB 700: PRINT "PONTOON": RETURN

3800 REM **** BUST ****
3805 LET PV=0
3810 GO SUB 700: PRINT "BUST": RETURN
3900 REM **** FIVE CARD TRICK ****
3905 LET PV=3
3910 GO SUB 700: PRINT "FIVE CARD TRICK"
: RETURN
```

Commodore 64

```
120 GOSUB 2600:REM TWIST ETC
2600 REM **** PUNTER TWIST ETC EVALUATE
****
2610 GOSUB2700:REM DO IT!
2620 ON EF GOSUB3500,3600,3700,3800,3900

2630 RETURN
2700 REM **** TWIST/STICK/DOUBLE ****
2710 GOSUB800:IF EF=2 OR EF=3 THEN RETUR
N:REM CHECK PONTOON/ROYALPONTOON
2720 GOSUB700:PRINT"TWIST/STICK/DOUBLE "
;
2725 GET AN$:IF AN$="" THEN 2725
2727 IF AN$<>CHR$(13) THEN PRINT AN$
2730 IF AN$="S" THEN GOSUB 2800:IF CS=0
THEN RETURN
2733 IF AN$="S"AND CS=1 THEN 2700:REM CA
N'T STICK
2750 IF AN$<>"T" THEN 2700:REM INPUT ERR
OR
2760 FL=0:PL=1:GOSUB1300:REM DEAL
2770 GOSUB800:IF EF=1 THEN 2700:REM AGAI
N?
2780 RETURN
2800 REM **** STICK ****
2810 CS=1:GOSUB800:REM EVALUATE
2820 IF (TT(PL,2)>=17 AND TT(PL,2)<22)OR
 TT(PL,1)>=17 THEN CS=0:RETURN
2825 GOSUB700:PRINTCHR$(28);"YOU CAN'T S
TICK UNDER 17"
2830 FOR DL=1 TO 1000:NEXT DL
2840 RETURN
3500 REM **** LESS THAN 21 ****
3505 PV=1:PS=TT(1,2):IF PS>21THEN PS=TT(
1,1)
3510 GOSUB700:PRINT"LESS THAN 21":RETURN

3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB700:PRINT"ROYAL PONTOON":RETUR
N
3700 REM ****  PONTOON ****
3705 PV=2
3710 GOSUB700:PRINT"PONTOON":RETURN
3800 REM ****  BUST ****
3805 PV=0
3810 GOSUB700:PRINT"BUST":RETURN
3900 REM ****  FIVE CARD TRICK ****
3905 PV=3
3910 GOSUB700:PRINT"FIVE CARD TRICK":RET
URN
```

Die Qual der Wahl

Für die meisten Hobby-Anwender reicht BASIC zur Programmentwicklung völlig aus. In Sonderfällen braucht man jedoch besser angepaßte Sprachen. Eine gründliche Analyse des Vorhabens sollte die Qual der Wahl jedoch lindern können.

Bis vor kurzem stand dem Heimcomputer-Freund ausschließlich BASIC oder Maschinensprache zur Verfügung. Heute sind dagegen fast alle verbreiteten Sprachen auch für die kleinsten Rechner erhältlich, die gesamte Palette steht beim IBM-Kompatiblen oder anderen MS-DOS-Rechnern zur Verfügung. Compiler und Interpreter können recht teuer sein, daher arbeiten nur wenige Anwender mit mehr als drei Sprachen.

Effektivität in bezug auf Speichergröße und die Arbeitsgeschwindigkeit sind die wichtigsten Faktoren. Das gilt besonders bei Programmen, die mit der Außenwelt kommunizieren sollen. Manche Programme wären in BASIC so schwierig zu erstellen, daß es leichter ist, eine neue, besser geeignete Sprache zu lernen und das Programm damit zu entwickeln. Ein Pro-

Sprache: BASIC
Vorteile: Leicht erlernbar und einfach in der Anwendung, weit verbreitet. Preiswert. Vielseitige Arithmetik-Funktionen. Komfortable Stringverarbeitung.
Nachteile: In vielen Versionen kaum Möglichkeiten für Modulprogrammierung und Steuerstrukturen. Keine Norm. Programmausführung langsam. Schlechtes File-Handling. Eingeschränkte Zahl von Datentypen und -strukturen. Programme undurchsichtig.
Anwendung: Kurze, einfache Programme mit Berechnungen oder Stringverarbeitung.

Sprache: LOGO
Vorteile: Solide mathematische Grundlage, leicht zu lernen. Turtle-Grafik. Gute Möglichkeiten zur Listenverarbeitung und für modulares Programmieren, viele Datentypen und Strukturen. Preiswert.
Nachteile: Auf höherer Ebene ist die Programmierung schwierig. Viele nicht-kompatible Versionen. Ausführung langsam.
Anwendung: Grafik, Listenverarbeitung, Lernen von Mathematik und Einführung in höhere Programmkonzepte.

Sprache: PASCAL
Vorteile: Gut strukturiert. Viele Datentypen. Weitgehend standardisiert. Verbreitet. Einfache Programmierung.
Nachteile: Ein/Ausgabe nicht klar definiert, File-Handling unbefriedigend.
Anwendung: Einüben guter Programmiertechnik. Allgemeine Aufgaben von beschränktem Umfang.

Sprache: FORTH
Vorteile: Schnelle Verarbeitung, vom Anwen-

gramm zur on-line-Verarbeitung von Lagerdateien wäre in COBOL durch den Einsatz indexierter Dateien sehr einfach zu erstellen. In BASIC müßte dagegen mit sequentiellen Dateien oder einem Hashing-Algorithmus und Random Access Files gearbeitet werden.

Bedienerfreundliche, leicht erlernbare Sprachen bereiten bei größerem Programmumfang oft Probleme. Hier kommt es darauf an, daß sich das Programm in selbständige Module zerlegen läßt, die unabhängig voneinander programmiert werden können.

Einige Sprachen, etwa COBOL und FORTRAN, sind gemäß allgemeingültiger Richtlinien fest definiert; die Anpassung an neue Hardware- und Verarbeitungstechniken ist dann recht mühsam und langwierig. Dafür erfordert das auf einem Rechner erstellte Programm nur eine erneute Compilierung und nur wenige Änderungen, bis es auf dem anderen Computer läuft.

Sprachen wie PASCAL und C haben nur eine Art de facto-Standard, der von den Erfindern festgelegt wurde. Die meisten Versionen dieser Sprachen entsprechen diesem Standard zwar, bei PASCAL sind jedoch Ein- und Ausgabeprozeduren nicht fest definiert; der Anwender hat also freie Bahn für Zusätze und Änderungen. Die entstandenen Programme sind natürlich nicht problemlos übertragbar.

Ein weiterer Faktor ist das Übersetzungsprogramm. Interpreter sind im allgemeinen einfacher zu bedienen und eignen sich besser für die Programmentwicklung, dafür sind sie aber auch langsamer beim Programmablauf. Compiler dagegen sind schwierig anzuwenden, das Endprodukt arbeitet dafür effektiver. Die Unterschiede nehmen allerdings durch die Entwicklung von Debugger-Programmen und komfortablen Compilern zunehmend ab.

Einige Sprachen arbeiten durch spezielle Hardware-Neuerungen sehr viel schneller. So wurde viel Aufwand in die Entwicklung eines Prozessors gesteckt, der direkt in FORTH arbeiten kann.

Der Hauptfaktor bleibt aber immer die Anwendung, für die eine Computersprache eingesetzt werden soll. Wir möchten Ihnen hier die wichtigsten Sprachen mit ihren Vorzügen und Schwachpunkten vorstellen.

der auszubauen. Gute Strukturierung.
Nachteile: Das Niveau ist oft zu niedrig. Schwer verständlich. Viele verschiedene Versionen.
Anwendung: Programmierung auf maschinennahem Niveau, speziell bei der Hardwaresteuerung.

Sprache: FORTRAN
Vorteile: Standardisiert. Umfangreiche Software-Bibliothek. Sehr viele Funktionen.
Nachteile: Altmodisch. Unbefriedigende Strukturierung. Programme oft unverständlich. Festgelegte Programmgestaltung.
Anwendung: Allgemeine wissenschaftliche, mathematische Programme und Anwendungen aus dem Produktionsbereich.

Sprache: COBOL
Vorteile: Standardisiert, für viele Rechner. Leicht verständlich. Gutes File-Handling.
Nachteile: Schwer zu erlernen. Compiler sind umfangreich und teuer. Schlecht strukturiert, geringe Zahl von Datentypen.
Anwendung: Kommerzielle Datenverarbeitung.

Sprache: C
Vorteile: Viele Datentypen und Strukturen. Gute modulare Struktur. Einfacher Zugang zur Hardware. Programme sind schnell.
Nachteile: Programme unübersichtlich. Für ernsthafte Anwendungen ist das Niveau zu niedrig. Nicht völlig standardisiert.
Anwendung: Systemsoftware als Ersatz für Assembler. Für besonders schnelle Anwendungen.
Sprache: LISP
Vorteile: Gute Listenverarbeitung, vielseitige mathematische Funktionen. Umfangreiche Software und Hilfsprogramme. Bei größeren Rechnern weit verbreitet.
Nachteile: Schwer erlernbar, schwer anzuwenden.
Anwendung: Künstliche Intelligenz und allgemeine Listenverarbeitung.

Sprache: PROLOG
Vorteile: Gute mathematische Basis. Läßt sich einfach einsetzen. Steht dem menschlichen Denken nahe.
Nachteile: Keine rein relationale Sprache, es gibt verfahrensorientierte Ansätze.
Anwendung: Künstliche Intelligenz, Datenbanksysteme.

Sprache: Assembler
Vorteile: Volle Kontrolle über alle Rechnerkomponenten. Effektivste Programmiersprache.
Nachteile: Keinerlei Norm. Schwer zu erlernen und anzuwenden.
Anwendung: Höchste Effizienz, aber nur, wenn sonst „nichts geht".

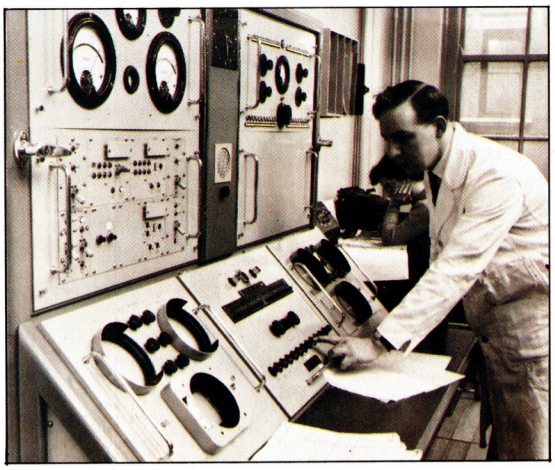
Sprachgefühl

In der Tabelle ist eine Reihe von Anwendungen aufgeführt. Die passenden Sprachen stehen nach Eignung geordnet rechts daneben.

Anwendung	Sprachen
Statistische Analysen	FORTRAN, BASIC, PASCAL
Lagerverwaltung	COBOL, PASCAL, BASIC
Robotersteuerung	FORTH, C, Assembler
Adventure-Spiele	PROLOG, C, BASIC
Expertensysteme	LISP, PROLOG, LOGO
Ausbildung	LOGO, PROLOG, PASCAL

Tee oder Computer

Der kommerzielle Einsatz des Computers begann auf der britischen Insel ausgerechnet in einem Teegeschäft.



Anders als seine Vorläufer, die für wissenschaftliche oder militärische Zwecke konstruiert wurden, war der LEO 1 nur für einfachste Arithmetik gedacht, konnte aber viele tausend Geschäftsvorgänge pro Tag bearbeiten.

An einem ungewöhnlichen Ort, nämlich in einer englischen Teehandlung, fiel 1947 die bahnbrechende Entscheidung, einen Rechner für die Automatisierung von Büroarbeiten zu entwickeln. Diesen Entschluß wagte das Unternehmen ‚J. Lyons Ltd.', dessen Ladenkette internationalen Ruf genoß. Kennzeichnend für ein derartiges Geschäft sind zahllose kleine Vorgänge, und um die Abwicklung überhaupt rentabel zu gestalten, muß der zugehörige Papierkram möglichst effizient bearbeitet werden.

Die Firma hatte schon eine gewisse Tradition in der Innovation der Bürotechnik: Bereits 1896 bekamen die Läden Rechenmaschinen, und in den dreißiger Jahren experimentierte Lyons mit Mikrofilm-Dokumentationsverfahren. Damals wurde auch erstmals ein eigenes Büro-Forschungszentrum zur Untersuchung von Arbeitsmethoden gegründet.

Das Unternehmen schickte des öfteren Mitarbeiter aus, die sich nach interessanten neuen Entwicklungen umsahen. 1947 kamen zwei Lyons-Mitarbeiter in die USA, um in den geheimnisvollen „Elektronengehirnen" herumzuschnüffeln. Sie kehrten mit der Erkenntnis zurück, daß sie sich die Reise hätten sparen können, da auch vor ihrer Haustür, an der Universität Cambridge, Computer entstanden.

Der Lyons-Vorstand ließ daraufhin durch eine Untersuchung klären, ob sich die Entwicklung eines eigenen Rechners für die Firma lohnen würde – den Entwicklungskosten von £ 100 000 stand eine jährliche Einsparung von £ 50 000 gegenüber. Folgerichtig gab Lyons im Oktober 1947 den Startschuß.

Die traditionsreichen Teehandlungen der englischen Firma Lyons & Co. gaben äußerlich eigentlich nicht das passende Milieu für den ersten kommerziellen Computereinsatz ab. Genau dieses Vertriebsgeschäft mit sehr vielen kleinen Umsätzen war aber prädestiniert für die Einführung der elektronischen Datenverarbeitung.

Primzahlen-Tabelle

1949 war es dann soweit, der Computer bewältigte eine akademische Aufgabe: die Erstellung einer Primzahlen-Tabelle.

Lyons analysierte derweil die Aufgaben, für die der neue Rechner gedacht war, und skizzierte die erforderlichen Programmabläufe. Diese Studien lieferten die Basis für spätere Software und für die Hardware: Es zeigte sich bald, daß ein Computer für den Büroeinsatz anders aussehen mußte als einer für die Wissenschaft. Während der Computer nach Eingabe von wenigen Zahlen einen komplizierten Formelsalat langwierig numerisch auszuwerten hatte, sollte der Bürorechner praktisch das Gegenteil tun – bei minimalem mathematischem Aufwand, beschränkt auf die Grundrechenarten, waren viele Daten zu bewältigen.

Am 9. Februar 1954 trat LEO (Lyons Electronic Office) erstmals in Aktion und erledigte die wöchentliche Lohnabrechnung für die 1700 Lyons-Mitarbeiter. Einen Vorgang, für den ein Angestellter zuvor acht Minuten gebraucht hatte, schaffte LEO in eineinhalb Sekunden.

LEO erwies sich als großer Erfolg, so daß nicht nur Lyons, sondern auch andere Unternehmen bald an diesem Gerät interessiert waren. Lyons gründete eine Tochtergesellschaft namens ‚LEO Computers', um das vorhandene Know-how auszuschlachten. Sie erzielte hervorragende Geschäftsergebnisse, indem sie eine Reihe verbesserter LEO-Versionen auf den Markt brachte.

Sprunghafte Routine

Programme, die mit Subroutinen arbeiten, sind flexibel, lassen sich leicht ändern und machen die Fehlersuche einfach. In dieser Folge untersuchen wir die JSR-Anweisung des 68000 und zeigen, wie sie eingesetzt wird.

Wir wollen den Befehl JSR (Sprung zum Unterprogramm) zunächst an einem einfachen Beispiel erläutern. Nehmen wir ein Programm, das ein Array von zehn Wörtern nach der Formel

```
TARRAY[I] := ARRAY[I] ^2 + 20*ARRAY[I]
```

umwandelt und die Summe aller Elemente von TARRAY in die Speicherstelle SUM stellt. Die Formel sieht zwar kompliziert aus, besagt aber nur: „Quadriere jedes Arrayelement einzeln, addiere darauf das Zwanzigfache des Elements und speichere das Ergebnis in dem entsprechenden Element von TARRAY."

Das erste Programmlisting zeigt, wie der 68000 die Umwandlung erledigt. Um feststellen zu können, ob das Programm richtig arbeitet, laden wir die Testdaten (1 bis 10) in ARRAY und prüfen TARRAY. Nach Ablauf des Programms müssen die Werte von TARRAY 21, 44,69,96 etc. betragen. Beachten Sie, daß nicht festgelegt wurde, für welchen Zahlenbereich ARRAY gültig ist. Bei einem fehlerlosen Programm sollten die Umwandlungsergebnisse und die Summe in eine Wortlänge (16 Bits) passen.

Unser Beispiel ist natürlich nicht der einzige Weg, diese Aufgabe zu lösen. Hier kommen Programmstruktur und die Bedeutung der Subroutine, speziell in der Assemblersprache, ins Spiel.

Zwar gibt es viele Möglichkeiten, Programme zu strukturieren, doch arbeiten fast alle „sauberen Programme" (d. h. Programme, die Sie auch später noch verstehen) mit der „funktionellen Aufsplittung". Dabei wird ein Programm in Bereiche aufgeteilt, die bestimmte Funktionen ausführen. Die Funktionen lassen sich mit Begriffen wie „Berechnen", „Umstellen" oder „Prüfen" beschreiben. In Hochsprachen heißen diese Codebereiche „Prozeduren" und die dorthin übergebenen Daten „Parameter".

Der Assemblercodierung steht nur die modulare Struktur der Subroutine zur Verfügung. Auf dem 68000 wird sie so aufgerufen:

```
JSR SUBR
```

Dabei ändert sich der Programmfluß durch einen Sprung auf den Codebereich mit dem Label SUBR. Dieser Code wird ausgeführt, bis er auf den Befehl RTS trifft. An dieser Stelle verzweigt der Programmfluß zu der Anweisung, die dem JSR-Befehl folgt (RTS bedeutet „Rückkehr aus dem Unterprogramm").

Doch zurück zu unserem Programmbeispiel. Statt „in einem Stück" zu quadrieren, multiplizieren und addieren, läßt sich dieser Vorgang auch abkoppeln und in einer Subroutine unterbringen. Das Programmlisting zwei zeigt den neuen Aufbau. Dabei steht in Zeile 6 der Aufruf der Subroutine CALC, und die früheren Zeilen 6 bis 9 befinden sich in Zeile 12 bis 15 – mit einem RTS am Ende.

Beliebig oft CALC

Das bringt den Vorteil, daß ein klar abgegrenzter Codebereich (Zeile 12 bis 16) einen exakt definierten Ablauf mit den Daten ausführt, die über den Parameter D1 übergeben wurden. Sie haben nicht nur einen übersichtlicheren Programmaufbau, sondern können die Subroutine CALC auch sooft und von wo aus Sie wollen aufrufen. Die Ergebnisse der in D1 übergebenen Daten werden in D2 zurückgegeben.

Beim Anlegen von Subroutinen sollten Sie, besonders bei Mehrplatzanwendungen, darauf achten, daß nur ein Einsprungspunkt und ein Rücksprung existiert. Auch sollten Sie vermeiden, von der Subroutine aus das Hauptmodul aufzurufen und weiterhin nicht zu viele bedingte Rücksprünge anlegen, da dies die Fehlersuche sehr erschwert.

Subroutinen fördern nicht nur einen guten Programmaufbau, sondern bieten auch die Möglichkeit, Speicherplatz zu sparen. (So muß der Code von CALC nicht für jeden Einsatz wiederholt werden.)

Wenn Sie CALC an mehreren Punkten des Programms aufrufen, brauchen Sie dann bei einer Änderung (beispielsweise der Umwandlung) lediglich einen Programmbereich zu ändern.

Nehmen wir an, die Umformung sollte

```
D2 := D1^2-20*D1
```

Programmbeispiel Eins

```
* Umformung von ARRAY in TARRAY mit der Formel
* TARRAY[I] := ARRAY[I] ^2 + 20*ARRAY[I]
* Beide Arrays enthalten zehn Elemente
* Folgende Register werden eingesetzt:
*    AO zeigt auf ARRAY
*    A1 zeigt auf TARRAY
*    DO ist ein Schleifenzähler
*    D1 ist ein Zwischenspeicher
*    D2 ist eine Kopie von ARRAY[I]

* Schleifenzähler initialisieren
* den ersten Pointer setzen
* den zweiten Pointer setzen
* SUM auf Null stellen
* ARRAY[I] holen
* und kopieren
* Quadrat von ARRAY[I] bilden
* D1 wird das 20fache von
  ARRAY[I]
* Komponenten addieren
* in TARRAY[I] speichern
* Gesamtsumme aktualisieren
* Schleifenzähler dekrementieren
* wiederholen, falls nicht fertig
* Ende der Routine, zurück zum
  Monitor
```

Anmerkungen zum Programm:
Die Zeilen 1 bis 4 initialisieren die Programmvariablen. In Zeile 5 werden die Elemente von ARRAY mit indirekter Adressierung und Nach-Inkrementierung in D1 geladen. Zeile 6 legt eine Kopie in D2 an, damit ARRAY nicht nochmals angesprochen werden muß. Die Zeilen 7 und 8 bilden mit dem vorzeichenbehafteten Multiplikationsbefehl MULS die beiden Umformungskomponenten D1 und D2, die dann von Zeile 9 addiert werden.

Zeile 10 legt das Ergebnis in dem entsprechenden Element von TARRAY ab, und Zeile 11 addiert das Ergebnis zu der Gesamtsumme SUM. Zeile 12 und 13 steuern die Schleife, damit nur zehn Elemente umgewandelt werden. Das Programm ist eigentlich eine FOR... NEXT-Schleife zwischen den Zeilen 5 und 12/13, deren Schleifenzähler in Zeile 1 gesetzt wird.

Programmbeispiel Zwei

```
* Subroutine zur Berechnung
  aufrufen
* neuer Arraywert

* Subroutine zur Berechnung
  von D1 ^2+20*D1
* und Rücksprung mit dem
  Ergebnis in D2
* Rücksprung zum aufrufenden
  Programm
```

lauten, dann genügt in der neuen CALC-Routine nur die Änderung der Zeile 15 in

```
SUB D1,D2
```

Die Programme werden dadurch nicht nur flexibler, sondern lassen sich auch leichter von Fehlern befreien. So könnten Sie CALC beispielsweise mit jeder Art von Werten (in D1 übergeben) testen, die (bei Rückgabe in D2) keine Probleme im Hauptprogramm verursachen. Die Subroutine hat nur einen Einsprungspunkt (bei dem Adreßlabel CALC) und eine Rücksprunganweisung (RTS).

Sie sehen, daß die Kriterien Flexibilität, Anpassungsfähigkeit und Zuverlässigkeit erfüllt sind. Subroutinen haben jedoch auch Nachteile. Wir müssen uns dazu (und auch für den Mechanismus der Parameterübergabe) die Stackvorgänge des 68000 genauer ansehen.

„Nebenwirkungen"

Bei der Ausführung von JSR muß die Adresse des Befehls direkt hinter JSR (die Rücksprungverbindung) gespeichert werden. JSR schiebt diese Adresse daher (als die vier Bytes einer Langwortadresse) auf den Stack, während der Programmzähler (PC) mit der Adresse der Subroutine geladen wird. Wenn der Ablauf der Subroutine nun auf ein RTS trifft, wird die Rücksprungadresse wieder vom Stack gezogen und der PC damit geladen. Auf diese Weise geht die Programmausführung nach Beendigung der Subroutine weiter. Von all diesen Vorgängen merkt der Anwender nichts, da der 68000 sie automatisch ausführt. Achten Sie auf folgende „Nebenwirkungen".

Zunächst müssen Sie sicherstellen, daß der Stackpointer, SP (auf dem 68000 das Register A7), richtig gesetzt ist, da man sonst Programmcode oder Daten überschreibt oder gar Hardwareadressen ändert. Dies geschieht normalerweise mit:

```
STACK EQU $1000
* Stack auf 1000 hex setzen
BEGIN LEA STACK,SP
* Stackpointer initialisieren
```

Der Stack „wächst" nun von 1000 aus in Richtung Null (jeder ‚PUSH' ist demnach eine Vor-Dekrementierung).

Sie müssen aber auch darauf achten, daß der Stack nicht zu groß wird. Hier können Sie nur schätzen, wieviel Stackkapazität nötig ist – und dies ist manchmal sehr schwierig. Bei kurzen Programmen mit einfachen Subroutinenaufrufen ergibt eine Untersuchung des Codes und ein zusätzlicher Sicherheitsfaktor oft schon die richtigen Werte. Bei verschachtelten Subroutinen oder Rekursionen kann die Berechnung der Stackgröße so schwierig werden, daß Sie sich nur durch Ausprobieren an die richtigen Werte herantasten können.

In der nächsten Folge geben wir weitere Beispiele für Subroutinenaufrufe und konzentrieren uns besonders auf die Datenübergabe an und von Subroutinen. Hier bietet der 68000 einige interessante Befehle.

Fachwörter von A bis Z

Thermal Printer = Thermodrucker
Der Druckknopf eines Thermodrukkers enthält eine Matrixanordnung von kleinen Heizwiderständen, die das Schriftbild durch Wärmeeinwirkung auf ‚Thermopapier' erzeugen. Dessen wachsartige Beschichtung wird beim Schmelzen transparent und läßt die darunter befindliche Farbe sichtbar werden.

Thermodrucker finden im Heimcomputerbereich Einsatz, da sie preiswert, sehr leise und ziemlich schnell sind.

Threaded List = Verknüpfte Liste
Zusatzinformationen an Datenfeldern sind nützliche Maßnahmen bei Datenbanksystemen oder Sprachkonzepten für die ‚Künstliche Intelligenz', so daß der Rechner über diese ‚Fäden' (threads) dann Querverbindungen folgen kann. Aus einer derart präparierten Liste läßt sich leicht eine bestimmte Auswahl von Datensätzen zusammenstellen.

Time Sharing = Timesharing
Dieses Verfahren für die Zuteilung von CPU-Zeit bei Mehrplatzsystemen arbeitet mit festen Zeitanteilen (Time Slices = ‚Zeitscheiben') im Millisekundenbereich, die den einzelnen Teilnehmern im Wechsel zugewiesen werden. Bei anderen Multiprogramming-Techniken werden die Zeitscheiben dagegen unter Berücksichtigung diverser Gesichtspunkte individuell bemessen.

Der Timesharing-Betrieb einer größeren Anlage ist kostengünstiger als das Arbeiten mit vielen kleinen Einzelsystemen, da die Auslastung von CPU und Peripherie besser ist. Ein Nachteil besteht darin, daß gleich eine sehr aufwendige Maschine angeschafft werden muß, die eine Vielzahl von Terminals bedienen kann und dabei so schnell ist, daß für die Benutzer keine störenden Wartezeiten in Erscheinung treten.

Top-Down Development = Top-Down-Programmierung
Bei diesem Vorgehen wird ein stufenweise immer feiner gegliederter Entwurf erstellt – es entsteht eine

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.

Thermodrucker erfreuen sich trotz einiger systembedingter Nachteile im Heimcomputerbereich großer Beliebtheit. Sie sind billig in der Anschaffung, leise und zuverlässig, brauchen aber teures Spezialpapier und bieten keine hohe Druckqualität.

‚Baumstruktur'. Erst wenn die Verästelung zu einfachen Teilaufgaben geführt hat, beginnt die Codierung.

Wenn Sie ein Programm schreiben wollen, das einen Roboter eine Tüte Milch aus dem Laden um die Ecke holen läßt, besteht die Gesamtaufgabe aus ‚Hingehen, Einkaufen, Zurückkommen'. Jeder dieser drei Aufträge wäre nun weiter zu unterteilen, das ‚Hingehen' z. B. in ‚Haus verlassen, Straße entlang, links, Geschäft betreten'. Als nächstes müßten Sie den Punkt ‚Haus verlassen' in ‚Tür ansteuern, Klinke fassen, ...' zerlegen usw. Das Unterteilen der Aufgabenstellung in ihre Details führt schließlich zur Formulierung der Anweisungen für die Antriebsmotoren.

Trace Programm = Tracer
Ein Tracer oder ‚Ablaufverfolger' ist eine Hilfsroutine zum schrittweisen Protokollieren der Ausführung eines Programms, um die Fehlersuche zu unterstützen. Je nach Betriebssystem sind mehr oder weniger komfortable Tracer verfügbar, auch für Heimcomputer, wobei als Ein-/Ausbefehle häufig TRON (TRace ON) bzw. TROFF (TRace OFF) verwendet werden.

Die Ablaufprotokollierung erfolgt unterschiedlich detailliert, sie kann beispielsweise alle angesteuerten Zeilennummern enthalten oder nur die Prozedurnamen in der Reihenfolge ihres Aufrufs.

Bildnachweise
2241: Claudia Zeff
2242: Frederic Voisin
2244, 2257, 2259, 2260: Caroline Clayton
2254, 2255, U3: Chris Stevens
2261: Liz Dixon
2261: Liz Heany
2263: Mike Brownlow

In unserem Serviceteil geht es diesmal um ein Spielprogramm bei dem Sie ohne jedes Risiko in das Goldgeschäft einsteigen können. Es geht um hohe Einsätze bei denen Sie Ihre Mitstreiter ausstechen können. Es geht um hohe Einsätze bei denen Sie Ihre Mitstreiter ausstechen können. Im Goldrausch werden Sie über Sich hinauswachsen und endlich einmal ohne jedes Risiko viel Geld verdienen.

+ Vorschau +++ Vorschau +++ Vorschau +

computer kurs

Die Bank zieht

Bei unserem 17+4 Programmierspiel geht es um die Routinen, es der Bank zu ermöglichen, auf Aktionen des Spielers zu reagieren. Trotz aller Cleverneß, die Bank gewinnt immer.

Elektronische Post

Unser letzter Artikel der UNIX-Reihe befaßt sich mit einer weiteren Annehmlichkeit, die das System bietet: die elektronische Post. Jeder hat seine eigene Briefkastendatei.

Bit für Bit

Bei 68000-Systemen benutzt die Peripherie den gleichen Kommunikationsbus, an den auch Speicher und CPU angeschlossen sind.