

Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 57

Optische Techniken

Jupiter Ace

Abläufe in Datenbanken

Gerät zum Greifen

Interruptmechanismus

**Ein
wöchentliches
Sammelwerk**

computer kurs

Heft 57

Inhalt

Computer Welt



Lichtschalter 1569

Optische Techniken führen zu Lichtcomputern

Spezialanwendungen 1591

Lernverbesserungen für Behinderte

Software



Kreuzverweise 1572

Abläufe in Datenbanken

Aktive Akteure 1586

Wie Darsteller gesteuert werden

Human Defekt 1596

Neue Form der Computerunterhaltung

Hardware



Das „AS“ 1574

Ein Heimcomputer mit Forth

FORTH



Zwei Bedingungen 1576

Jetzt kommen die Strukturen

Bits und Bytes



Die letzten Drei 1579

Interruptmechanismus beim Debugger

Des Fehlers Tod 1588

Der 6809-Kurs ist komplett

Tips für die Praxis



Entwurf 1582

Ein Gerät zum Greifen und Bewegen

BASIC 57



Nützliche Zeiger 1584

Programmverbesserungen

Vorratsbeschaffung 1594

Fachwörter von A—Z

WIE SIE JEDE WOCHEN IHR HEFT BEKOMMEN

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

Zurückliegende Hefte

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

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

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

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

Abonnement

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

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

SAMMELORDNER

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

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

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

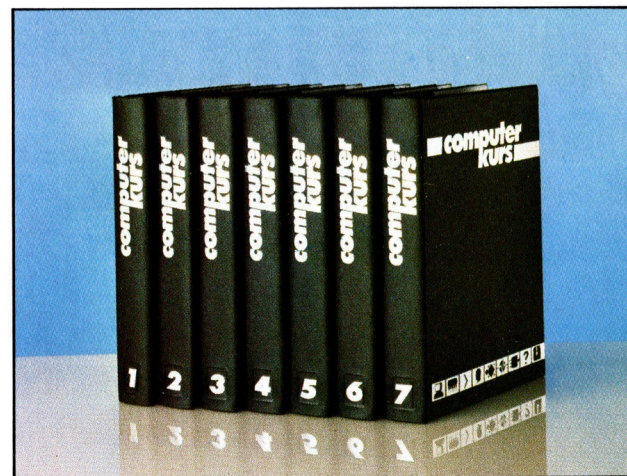
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Peter Aldick, Holger Neuhaus, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

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

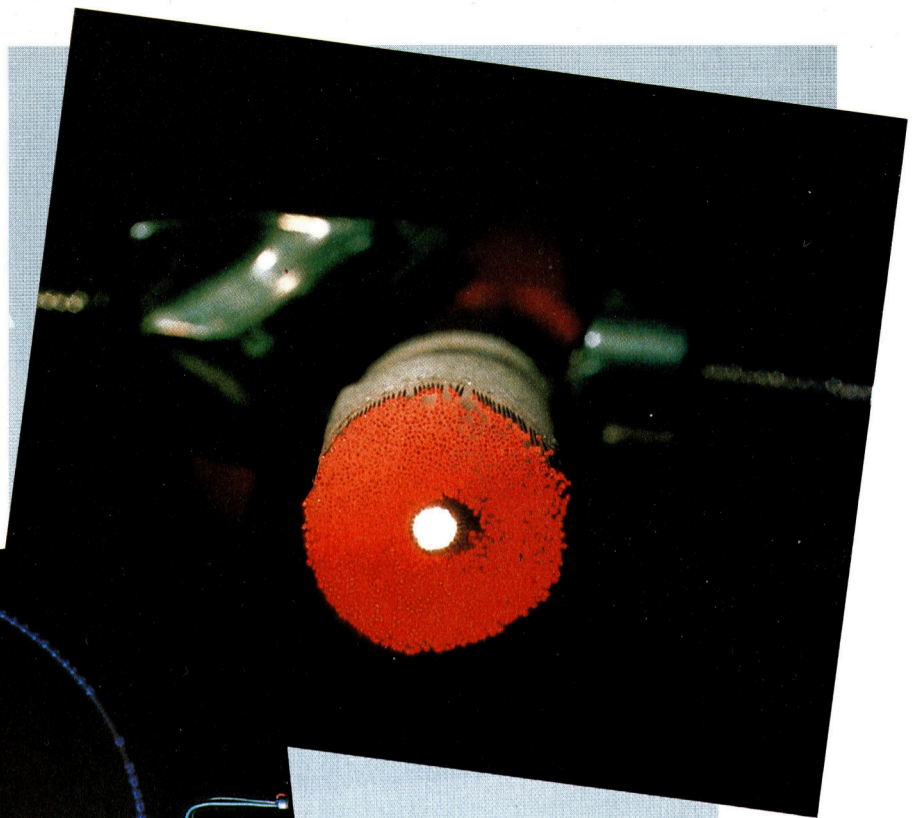
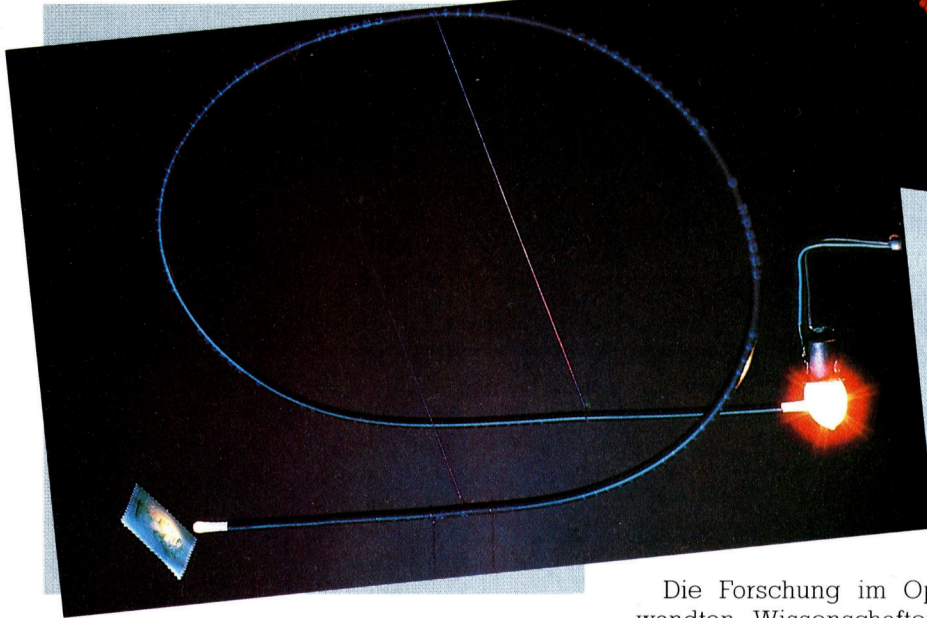


© APSIF, Copenhagen, 1982, 1983; © Orbis Publishing Ltd., 1982, 1983; © Marshall Cavendish Ltd., 1984, 1985, 1986; **Druck:** E. Schwend GmbH, Schmollerstraße 31, 7170 Schwäbisch Hall



Licht-schalter

Die Entwicklung neuer optischer Techniken wird in naher Zukunft auch zum Bau von „Lichtcomputern“ führen. Licht kann Informationen sehr viel schneller übertragen als Elektrizität – der Computer wird also mehr als tausendmal schneller arbeiten.



Unter einem Computer stellt sich fast jeder automatisch ein elektrisches Gerät vor, das Informationen in digitalen Schaltkreisen speichert und verarbeitet. Doch Daten und Informationen können auch ganz anders gespeichert werden: in speziell geformten Zahnradern, auf Lochstreifen, sogar als Perlenmuster – nahezu jede physikalische Anordnung kann Informationen tragen.

Elektronische Halbleiter gelten bisher als beste Informationsspeicher. Sie verbrauchen wenig Energie, sind mikroskopisch klein und in der Datenverarbeitung sehr schnell. Die Arbeitsgeschwindigkeit eines Computers wird maßgeblich von der Zeit bestimmt, die für das Speichern oder Weiterleiten binär codierter Informationen erforderlich ist. Auch die Übertragungszeit zwischen zwei Schaltkreisen spielt eine Rolle. Schon der elektronische Computer, der mit bewegten elektrischen Ladungen arbeitet, ist sehr schnell – trotzdem scheint es möglich, diese Schnelligkeit durch Einsatz von Lichtstrahlen noch zu übertreffen.

Die Forschung im Optikbereich und verwandten Wissenschaften wie der „Photonischen Logik“ ist bereits auf dem Wege zu neuen Erkenntnissen. Viele der für einen optischen Computer notwendigen Teile gibt es bereits heute. Lichtleitfasern werden schon jetzt für die Übertragung von Telefon- und Fernsehsignalen genutzt, optische Leiter, die der Funktion der metallischen Leiterbahnen auf einer Platine entsprechen, sind bereits heute entwickelt, und in Labors und Prüfräumen liegt auch schon das wichtigste Teil: der dem Transistor analoge optische Schalter.

Lichtleitfasern sind für die Optik das, was ein Kabel in der Elektrotechnik darstellt. In ihnen kann Licht auch „um die Ecke“ transportiert und damit an jeden beliebigen Ort geleitet werden. Die einzelnen Fasern bestehen aus einem transparenten Trägermaterial, in dem das Licht in Faserichtung weitergeleitet wird.

Hybrid- und Optoschalter

Die neuen optischen Schalter gehören zwei Kategorien an: Es sind elektro-optische Hybridschalter oder reine Optoschalter. Beide Entwicklungen sind wichtig, denn es ist wahrscheinlich, daß zuerst der Hybridschalter mit elektronischen Komponenten auf den Markt kommen wird, bevor sich sein rein optischer Konkurrent durchsetzt. Beide beruhen auf dem Prinzip der Lichtinterferenz, die durch Veränderungen der Lichtbrechung in optisch nicht li-



Ein wichtiges Bauteil, das für zukünftige optische Computer gebraucht wird, ist der Halbleiter, der mehrere Lichtfrequenzen ohne störendes „Rauschen“ erzeugen kann. Das ist speziell für die Datenübertragung wichtig – hier werden Lichtquellen mit exakt eingehaltenen Wellenlängen gebraucht, damit es keine Überlagerung zwischen einzelnen Signalen gibt.



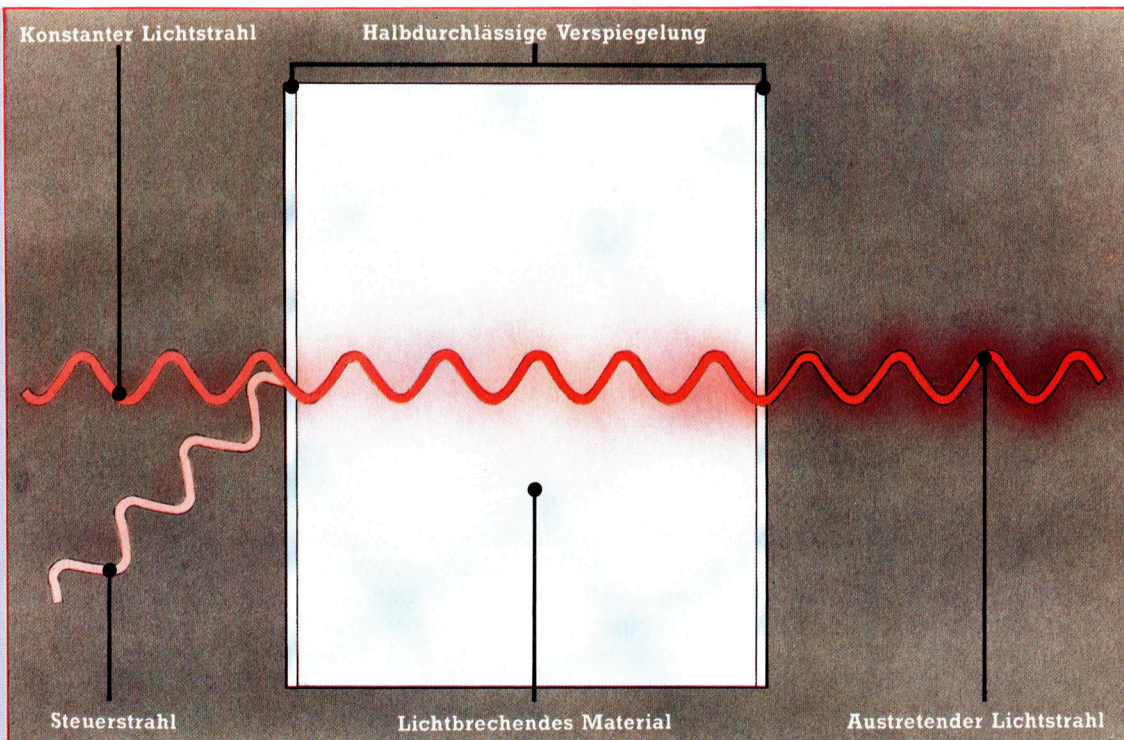
nearen Materialien gesteuert werden soll. Man verwendet meist die Elemente wie Gallium und Lithium für diesen Zweck. Der Brechungsindex drückt das Verhältnis der Geschwindigkeit des Lichts im Vakuum zur Geschwindigkeit in einem Medium aus. Manche Glassorten haben etwa einen Brechungsindex von 1,5. Das bedeutet, daß sich die Lichtgeschwindigkeit im Glas von der Vakuumgeschwindigkeit (300 000 km/s) auf 200 000 km/s reduziert.

Der opto-elektronische Hybridschalter fußt auf der Erfindung des „Mach-Zehnder-Interferometers“. Das Prinzip dieses Geräts ist sehr einfach: Einfallendes Licht wird in zwei Strahlen zerlegt, die durch getrennte Kanäle eines optisch nicht linearen Materials geschickt werden. Liegt keine elektrische Spannung über einem der beiden Kanäle, erreichen die Lichtstrahlen den Ausgang in gleicher Phase und erzeugen bei erneuter Zusammenführung einen starken Lichtstrahl (Status „Ein“). Ist der Brechungsindex eines Kanals durch ein elektrisches Signal verändert, fallen die Lichtstrahlen nicht mehr phasengleich zusammen, sondern schwächen sich gegenseitig ab (Status „Aus“).

Mit Hybridelementen könnte man einfache optische Schaltungen aufbauen, die wegen der größeren Informationsdichte bei Lichtübertragung bis zu 10 000fach schneller als konventionelle Schaltungen wären. Allerdings sind dabei noch elektrische Vorgänge nötig – das Steuern des Lichtflusses erfolgt ja weiterhin elektrisch. Größere Verarbeitungsgeschwindigkeiten wären erreichbar, wenn auch die Ansteuerung des optischen Schalters mit Licht möglich wäre.

Der Prototyp eines rein optischen Schalters ist unter dem Namen „Transphasor“ (analog zum ähnlich arbeitenden Transistor) bereits an der Heriot-Watt-Universität in Edinburgh gebaut worden. Seine Schaltgeschwindigkeit beträgt nur eine Picosekunde (Billionstelsekunde) – tausendmal weniger als der schnellste Transistor, der immerhin eine Nanosekunde (Millionstelsekunde) benötigt. Der bei Heriot Watt entwickelte Transphasor beruht auf dem Prinzip des 1896 von den französi-

Das Grundelement des elektronischen Computers ist der Transistor. Sein Zwilling im optischen Bereich heißt „Transphasor“. In diesem Bauteil findet eine Wechselwirkung zwischen einem starken Lichtstrahl mit einem schwächeren Steuerstrahl statt. Aus kleinen Veränderungen der Lichtintensität im Steuerstrahl ergeben sich große Intensitätsschwankungen beim austretenden Hauptstrahl – der Transphasor kann also wie ein Transistor als logischer Schalter dienen.





schen Physikern Charles Fabry und Alfred Perrot entwickelten Interferometers: Ein Stück optisch nichtlinearen Materials wird zwischen zwei reflektierende Oberflächen gebracht. Licht, das durch eine Oberfläche eindringt, wird innerhalb dieses Materials hin- und herreflektiert, bevor es wieder austritt. Dabei entsteht eine Wechselwirkung im eingeschlossenen Wellenpaket, die vom Brechungsindex des Materials abhängt. Der Brechungsindex kann durch einen Steuer-Lichtstrahl verändert werden. Ist der Brechungsindex derart, daß die Lichtwellen in Phase sind und sich wechselseitig verstärken, tritt ein kräftiger Lichtstrahl aus. Sind die Wellen aber nicht phasengleich, wird das Licht abgeschwächt. Die beiden Transmissions-Formen entsprechen den Zuständen „Ein“ bzw. „Aus“.

Zukunftsprognosen

Ob diese Technik Erfolg haben wird, hängt nicht zuletzt vom Fortschritt der konventionellen Rechner ab, denen sie – jedenfalls in der Theorie – haushoch überlegen ist. Der Arbeitsgeschwindigkeit der elektronischen Computer sind Grenzen gesetzt: In einem Transistor müssen bei jedem Schaltvorgang Elektronen durch die Basis fließen. Ihre Geschwindigkeit ist in einem Halbleiter jedoch beschränkt.

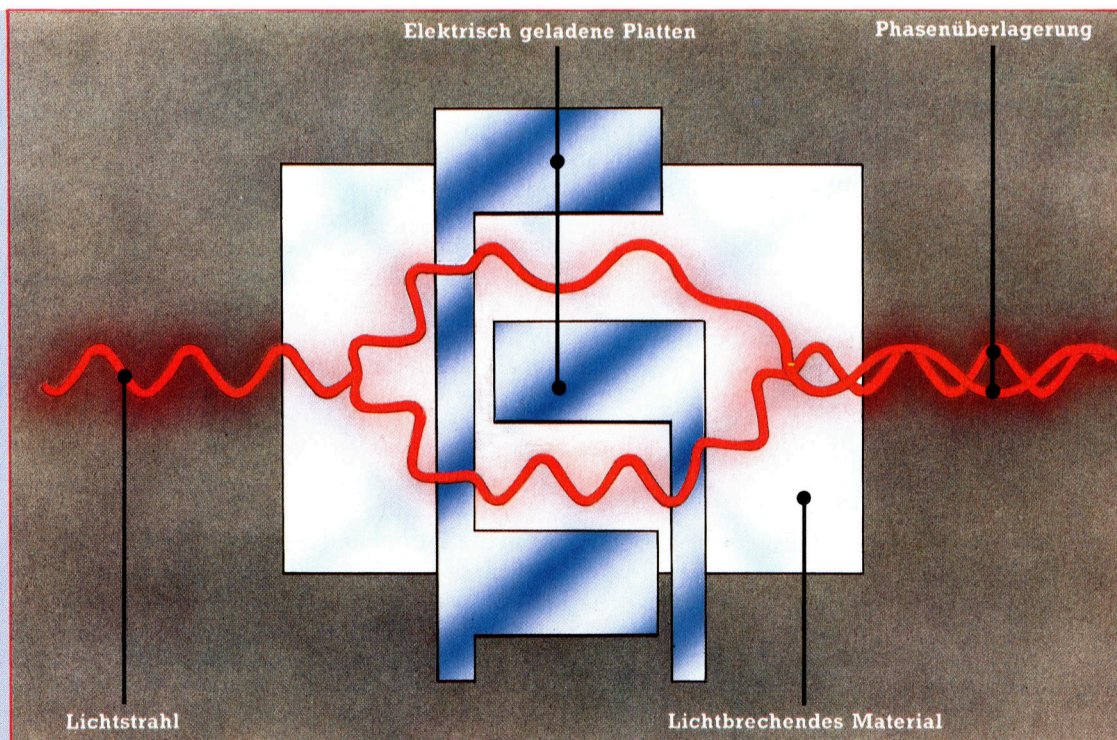
Konventionelle Rechner behindert zudem der „von-Neumann-Bottleneck“ genannte Flaschenhals-Effekt, der alle Rechner mit seriellem Speicherzugriff betrifft. Im optischen Computer können Speicherstellen parallel abgefragt werden, weil Lichtstrahlen nicht wie elektrische Signale gegeneinander isoliert werden

müssen. Ein optischer Schalter kann Lichtstrahlen unterschiedlicher Frequenz gleichzeitig stoppen oder durchlassen, ohne daß diese sich gegenseitig beeinflussen.

Eine weitere Beschränkung der herkömmlichen Computerkonstruktion ist der Platzmangel auf flachen Chips – es gibt nicht genug Raum für die Verdrahtung. Zur Zeit kann man nur eine begrenzte Zahl von Anschlüssen an einem Chip befestigen. Durch dreidimensionale Holografie-Technik kann man aber auf einen zusätzlichen Freiheitsgrad bei der Verdrahtung hoffen, was die gesamte Konfiguration der Silizium-Chips revolutionieren würde.

Bei der Verbindung von Computer und Peripherie zeigt sich ein weiterer Vorteil des optischen Computers. Aus Lichtleitfasern aufgebaute Netze werden bereits heute zur Datenübertragung genutzt. Sie arbeiten extrem schnell und mit hoher Informationsdichte.

Diese Vorteile legen die Frage nahe, wann wohl die ersten optischen Computer auf den Markt kommen werden. Es scheint, daß es noch eine Weile dauern wird. Die fortschreitende Miniaturisierung konventioneller Halbleiter-Chips bedeutet auch, daß optische Computer einen Konkurrenten aus dem Feld schlagen müßten, der momentan mit Riesenschritten davonläuft – noch immer verdoppelt sich die Zahl der auf einem Chip untergebrachten Transistoren im Zeitabstand von 18 Monaten. Für das Jahr 2000 sind bereits Chips mit 10 bis 100 Millionen Elementen angekündigt. Aus „Large Scale Integration“ (LSI) wurde „Very Large Scale Integration“ (VLSI), ja sogar „Ultra Large Scale Integration“, der Parallelprozessor hat sich inzwischen zum „Super-Parallelprozessor“ gemausert.



Im Mach-Zehnder-Interferometer wirken elektronische und optische Komponenten zusammen. Das auf einer Seite eintretende Licht wird in zwei Strahlen zerlegt. Beim Eintreten in einen Festkörper wird die Lichtgeschwindigkeit reduziert. Der Brechungsindex kann für einen Strahl durch den Einfluß eines elektrischen Feldes geändert werden – das Licht wird langsamer und erreicht den Ausgang des Interferometers nicht mehr phasengleich mit dem abgespaltenen zweiten Strahl. Bei der Vereinigung beider Strahlen schwächen sie sich dann gegenseitig stark ab – eine Analogie zum logischen „Aus“-Zustand bei einem Transistor.



Kreuzverweise

Bei der Verwaltung von Datenbanken sind oft für Organisation, Indizierung und Aufruf der einzelnen Dateitypen unterschiedliche Abläufe nötig. Wir zeigen, wie Datenbanken diese Abläufe handhaben.

Die Datensätze „einfacher“ Datenbanken sind im allgemeinen völlig selbständig. Sie enthalten alle benötigten Informationen, ohne sich auf andere Datensätze beziehen zu müssen. Auch die herkömmlichen Dateikästen fallen in diese Kategorie. In der Datenbank einer Bücherei besteht beispielsweise jeder Datensatz in den meisten Fällen nur aus den Feldern TITEL, AUTOR, HERAUSGEBER und ISBN (Internationale-Standard-Buch-Nummer) – ohne jeden Kreuzverweis.

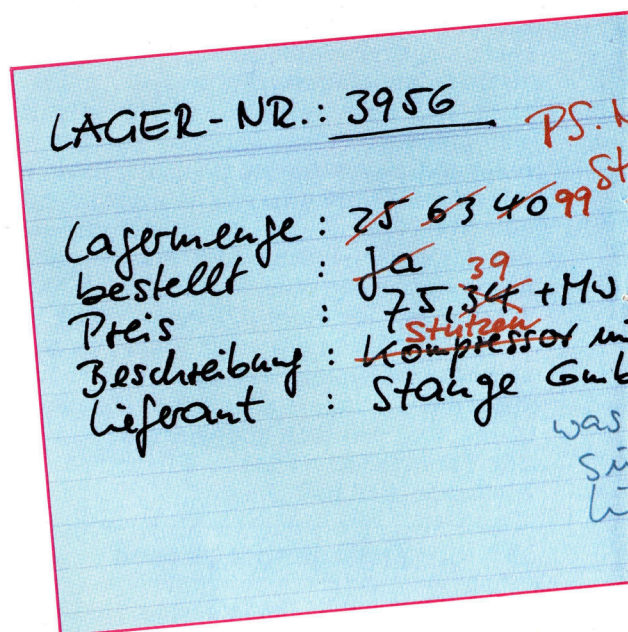
Es gibt aber auch Felder, die auf einen anderen Datensatz verweisen. So kann sich in der Datei CLUBMITGLIEDER das Feld KINDER beispielsweise auf andere Datensätze beziehen, wenn die Kinder des Mitglieds ebenfalls in den Club aufgenommen wurden. Felder dieser Art können sich jedoch auch auf Datensätze beziehen, die nicht in die Struktur der Datei passen: In der Datei LAGER mit den Feldern LAGERNUMMER, PREIS, LAGERMENGE und LIEFERANT verweist das Feld LIEFERANT auf Datensätze, die nicht zu der Struktur der Lagerdatei passen.

Hier das Beispiel einer Mitgliedsdatei:

NAME DES MITGLIEDS	Susanne Gärtner
EINTRITTSJAHR	1979
MITGLIEDSBEITRAG	bezahlt
BERUF	Lehrerin
JAHRESGEHALT	48 000
KINDER	Iris Gärtner, Paul Gärtner

Das Feld KINDER enthält im obigen Beispiel zwei Einträge. Vergleichen Sie dies mit dem Datensatz von LAGER:

LAGERNUMMER	3995
LAGERMENGE	86
PREIS	34,75
BESCHREIBUNG	Stutzen mit Ventilklappe
LIEFERANT	Windy Ltd., Hurrican Corp. of Taiwan



Der Feld LIEFERANT enthält zwei Bezüge auf andere Datensätze, die nicht in die Struktur der Datenbank LAGER passen. Datenbanken dieser Art beziehen sich oft auf eine zweite Datei, die beispielsweise die Lieferantendaten in folgender Form enthalten könnte:

LIEFERANT	Hurrican Corp. of Taiwan
ADRESSE	57 Kau Moo Road, Taipei, Taiwan, R.O.C.
TELEFON	010-886-2-223-4478
TEIL1	Stutzen mit Ventilklappe
PREIS1 (US\$)	34,75
TEIL2	Schmierfett (1/2-Liter-Dose)
PREIS2 (US\$)	6,00
TEIL3	Politur (kleine Dose)
PREIS3(US\$)	2,30
TEIL4	Politur (große Dose)
PREIS4 (US\$)	4,00
TEIL5	—

An diesem Beispiel wird deutlich, daß die Struktur der Lieferantendatei sich grundlegend von der der Lagerdatei unterscheidet. Aus diesem Grund werden zwei unterschiedliche Dateistrukturen eingesetzt – eine für die Lagerdatei und eine andere, getrennte Struktur für die Lieferanten.

Arbeiten mit dBase II

Wenn getrennte, aber miteinander in Beziehung stehende Dateien eingesetzt werden, muß die Datenbankverwaltung (DBV) mehr als eine Datei gleichzeitig bearbeiten können. Dabei läßt sich mit simplen Datenbanken wie „Card Box“ nur eine Datei zur Zeit ansprechen, während beispielsweise dBase II eine Datei als Primärdatei ansehen kann und eine andere als Sekundärdatei. Gute DBVs bieten die Möglichkeit, die Primärdatei zu sortieren, während



Handwritten notes on a blue sticky note:
 nur mit Halbung
 in den Bestellen
 ist
 1 Ventilklappe
 ist passiv
 und die pleile?
 feuerharte!!
 prüfen!!

In einem Karteikartensystem können die Einträge sehr umfangreich (und nicht organisiert) sein. DBVs müssen jedoch systematischer eingerichtet werden. Hochentwickelte Systeme geben dem Anwender die Möglichkeit, mehrere in Beziehung stehende Dateien gleichzeitig anzusprechen und Felder zu überspringen, die durch andere Einträge überflüssig wurden. Im nebenstehenden Beispiel ist das Feld **BESTELT** nur dann wichtig, wenn die vorhandene Lagermenge nahe oder gleich Null ist.

gleichzeitig Datensätze aus anderen Dateien eingelesen werden.

Das Paket dBase II kann auf zwei Dateien gleichzeitig zugreifen und deren Felder zueinander in Beziehung setzen. Die Befehle **SELECT PRIMARY** und **SELECT SECONDARY** stellen dabei den Bezug von einer Datei zur anderen her. Wenn **LAGER** als Primärdatei definiert werden soll und **LIEFERANTEN** als Sekundärdatei, muß zunächst mit dem Befehl **USE LAGER** die Lagerdatei aufgerufen und dann durch die Eingabe **SELECT SECONDARY** und **USE LIEFERANT** der Querverweis auf die Lieferantendatei hergestellt werden. Alle dBase-II-Befehle beziehen sich dabei auf die zuletzt angesprochene Datei.

Bei manchen Anwendungen ist es nicht nötig, voneinander getrennte Dateien anzulegen. Hier genügt es, wenn die Felder eines Datensatzes untereinander einen Bezug haben. Nehmen wir als Beispiel eine Datenbank, in der Presseberichte und Fachliteratur über die neuesten Automobiltechniken gespeichert sind. Die Datenstruktur könnte folgendermaßen aussehen:

THEMA: _____
 ÜBERSICHT1: _____
 ÜBERSICHT2: _____
 ÜBERSICHT3: _____
 QUELLE: _____ ISBN: _____
 TITEL: _____
 DATUM: _____
 SEITENNR: _____

Wenn der Eintrag einer Zeitschrift entnommen wurde, bleibt das Feld **ISBN** leer. Bei einem Buch ist das **ISBN**-Feld jedoch außerordentlich wichtig. In beiden Fällen wird außerdem die Seitenzahl gebraucht. Einige DBVs, beispielsweise „Rescue“ von Microcomputer Business Systems, können das Auftreten eines Feldes innerhalb eines Datensatzes von Berechnun-

gen oder von Einträgen in anderen Feldern abhängig machen.

Wenn dabei das Feld **QUELLE** keinen Eintrag erhält, wird **ISBN** nicht mehr angesprochen, dargestellt oder gedruckt. Ist **QUELLE** weder ein Buch noch eine Zeitschrift, wird dementsprechend auch das Feld **SEITENNR** nicht angezeigt. Durch die Unterdrückung nicht belegter Felder kann eine DBV viel Speicherplatz sparen und die Darstellung der Informationen übersichtlicher gestalten. Diese Methode ist jedoch weit weniger flexibel als eine Datenbankverwaltung, die Datensätze anderer Dateien ansprechen kann.

Datenbanken, deren Datensätze identische Grundinformationen enthalten und einige Zusatzinformationen nur zuweilen gebrauchen, haben oft eine Datenhierarchie mit zwei Ebenen. DBVs mit Vielfachzugriff behandeln hierarchische Strukturen normalerweise als Subset. Dabei kann der Bezug **QUELLE** in einer Literaturdatei auf einen Eintrag der Datei **BUCH** hinweisen, während der Bezug **QUELLE** in einer Programmdatei auf einen Datensatz der Datei **PROGRAMME** zeigt.

Bedenken Sie, daß Sie herkömmliche, handbeschriebene Karteikarten so kompliziert anlegen können wie Sie es wollen, kommerziell einsetzbare DBVs in ihrer Struktur aber oft stark eingeschränkt sind. Bevor Sie sich für Ihren Micro ein Datenbankpaket zulegen, sollten Sie daher zunächst Ihre Anforderungen definieren und sich die Eigenschaften der einzelnen Systeme genau ansehen.

Datenbanken

Name	Hersteller	Gerät	Format	Kommentar
Betabase	Clares	Acorn B	Diskette	Max. Satzlänge 2048. Max. mögl. Felder 200. Max. Dateigröße 99K, 199K.
Database	Gemini Markeking	Spectrum	Cassette	Kartenindexsystem. Anwenderdefinierte Datensätze.
DFM Database	Dialog	C64	Diskette	15 Felder/Datensatz. 36 Zeichen. Alphanumerisches Feld. 9-stelliges Zahlenfeld.
Practifile	Computer Software Associates	C64	Diskette	3800 Datensätze/Datei. Sortiert Daten für Adreßlisten. Batcheingabe.
MicronPen	AMSoft	Schneider	Diskette	Eingebautes Text- und Kalkulationssystem. Max. Satzlänge 1024. Max. mögl. Datensätze in einer Datei: 32750.



Das „As“

Als einziger preisgünstiger Heimcomputer ist der „Jupiter Ace“ mit FORTH anstelle von BASIC als Standardsprache ausgestattet – eine Herausforderung für alle angehenden FORTH-Programmierer.

Der Jupiter Ace ist insofern ein As, als er zu den wenigen Microcomputern gehört, die nicht von Haus aus mit BASIC arbeiten. Deshalb ist unter seinen Fans immer noch ein schwungvoller Handel mit dem Gerät und der dazugehörenden Software im Gange, obwohl der Rechner seit 1983 nicht mehr gebaut wird.

Die Ausrüstung mit FORTH als Standardsprache hebt den Jupiter Ace deutlich von der Konkurrenz ab. Und er ist gebraucht so billig zu haben, daß Sie damit günstiger wegkommen, als daß Sie sich zum Kennenlernen von FORTH die nötigen Ergänzungen zu Ihrem vorhandenen Rechner kaufen.

Programmieren in FORTH

BASIC-Version

```
100 REM A BASIC PROGRAM TO PRINT 'SHAZAM!'
110 FOR X = 1 TO 6
120 PRINT "SHAZAM!"
130 NEXT X
140 END
RUN
```

FORTH-Version

```
( A FORTH PROGRAM TO PRINT 'SHAZAM!' )
: SHOUT ." SHAZAM! " ;
: CHORUS 6 0 DO SHOUT LOOP;
CHORUS
```

Die beiden vorstehenden Programme leisten genau das gleiche, wobei das BASIC-Programm eher an ein Kochrezept erinnert, während die FORTH-Fassung wie ein Zauberspruch aussieht.

In FORTH können auf der Basis eines gegebenen Grundvorrats an Befehlen („Wörter“) jederzeit neue Kommandos definiert werden, um das Vokabular zu ergänzen. Im obigen Beispiel ist SHOUT für das Ausdrucken einer Zeichenkette neu eingeführt worden, und CHORUS ist aus dem zuvor definierten SHOUT und einigen Wörtern zusammengesetzt worden.

Die Zahlenwertspeicher ist in FORTH als Stapel (Stack) organisiert. Die gewohnten arithmetischen und logischen Operationen sind in FORTH – angepaßt an die Stack-Technik – in „Umgekehrter Polnischer Notation“ zu schreiben. Das Programmieren in FORTH erfordert in mehrfacher Hinsicht eine Umstellung, die sich wegen der Leistungsfähigkeit dieser Sprache aber auszahlt. Der Umgang mit FORTH hat einen ähnlichen Reiz wie das Spielen mit dem Rubik-Zauberwürfel, und so etwas macht nun mal dem einen Spaß, während es den anderen langweilt.

Die Vielseitigkeit von FORTH wird dadurch erzeugt, daß neue Befehle definiert werden können. Der Benutzer kann den Wortschatz ganz auf die Bedürfnisse des jeweiligen Anwendungsfalls zuschneiden.

Tastatur

Die Tasten in der obersten Reihe sind alle dreifach belegt. Die Umschaltung erfolgt durch Drücken von SHIFT bzw. SYMBOL SHIFT. Außerdem stehen sieben Grafikzeichen für die Erzeugung einfacher Diagramme und Grafiken zur Verfügung.

Microprozessor

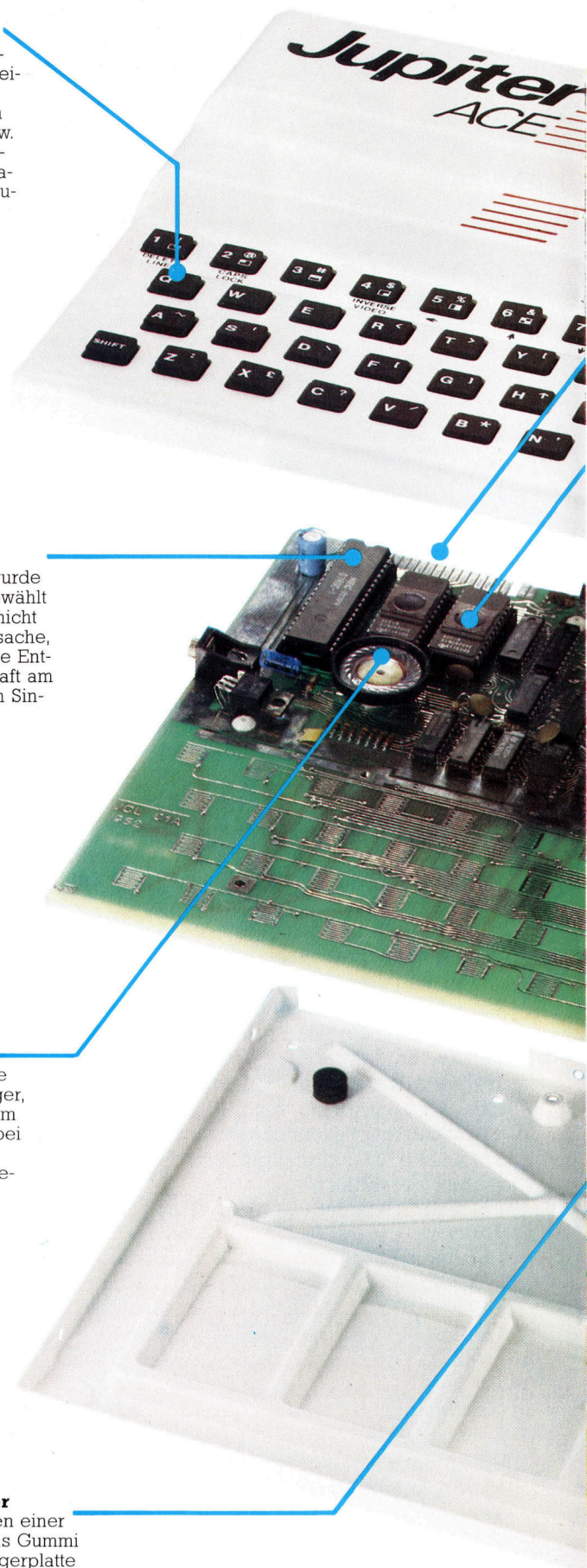
Beim Jupiter Ace wurde als CPU ein Z80 gewählt – das verwundert nicht angesichts der Tatsache, daß hier die gleiche Entwicklungsmannschaft am Werk war wie beim Sinclair Spectrum.

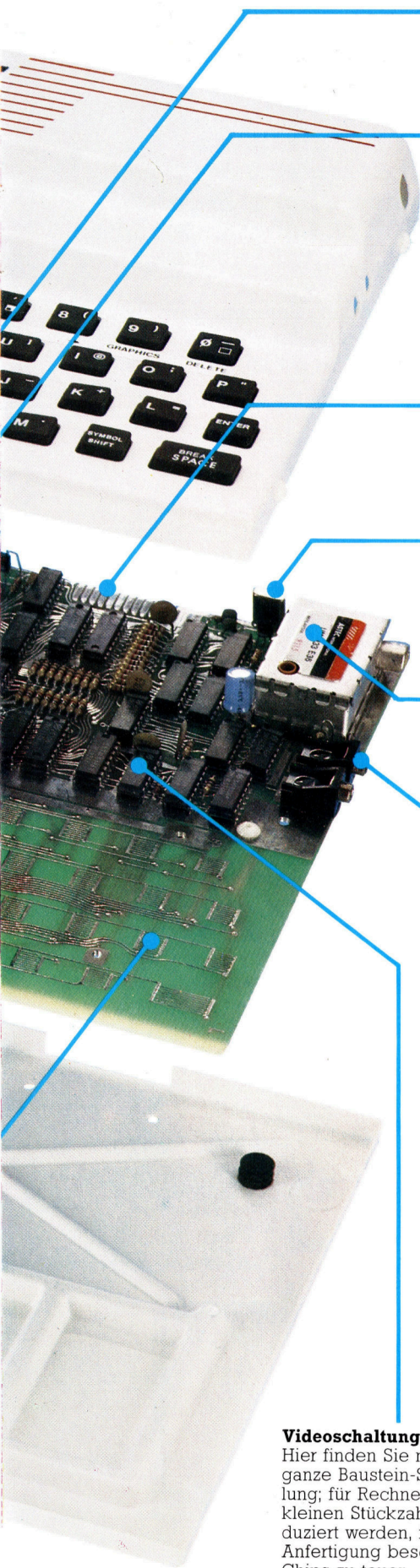
Lautsprecher

Für die Tonwiedergabe sorgt ein Piezoschwinger, wie er in ähnlicher Form auch als Signalgeber bei Digital-Armbanduhrn eingesetzt wird. Das genügt für einfache Sound-Effekte.

Kontaktfelder

Beim Betätigen einer Taste wird das Gummi der Tastenträgerplatte gegen das Kontaktbahnenmuster auf der Platine gedrückt und der Stromkreis geschlossen.



**Buserweiterungsstecker**

Außer für die Speichererweiterung eignet sich diese Steckleiste auch für einen Adapter für das gesamte Sinclair-Zubehör.

FORTH-EPROMs

Das FORTH-System ist in zwei 4-KByte-EPROMs gespeichert. Die Maskenherstellung für ROMs rentiert sich erst bei sehr hohen Stückzahlen, so daß für kleinere Serien nur EPROMs in Frage kommen.

Benutzeranschluß

Dieser Platinenstecker ist vorwiegend für einen Drucker gedacht. Aber auch andere Peripheriegeräte sind möglich.

Taktgeber

Die Z80-CPU hat keinen internen Generator für die Taktfrequenz. Ein separater Quarzkristall erzeugt den 1-MHz-Takt.

HF-Modulator

Mit dem Antennensignal des Jupiter Ace läßt sich ein (Schwarz-weiß-) Fernseher betreiben.

Cassettenrecorderanschluß

Äußerlich ist der Rechner eng mit den Sinclair-Modellen verwandt. Wie der ZX80 hat auch der Jupiter Ace ein leichtgewichtiges, weißes Plastikgehäuse. Zur Klangwiedergabe dient ein kleiner Piezokristall, dem sich nach näherer Beschäftigung mit dem Gerät auch etwas mehr als nur einfache Tonfolgen entlocken lassen. Wie die Sinclair-Rechner hat auch der Jupiter Ace einen Transformatoranschluß, eine Antennenbuchse und zwei große mehrpolige Buchsen für Peripheriegeräte. Ein Cassettenrecorder ist über zwei Steckbuchsen anschließbar.

Die Bildschirmausgabe erfolgt schwarzweiß mit 32 Zeichen in 22 Zeilen, im Grafikmode auch 64x48. Jedes Zeichen kann vom Benutzer neu gestaltet werden, um zum Beispiel mathematische Symbole oder Sonderzeichen zu erzeugen.

Der Jupiter Ace ist serienmäßig mit drei KByte ausgestattet. Für kurze FORTH-Programme reicht diese Speicherkapazität jedoch aus. Für höhere Ansprüche gibt es ein 16K- und ein 32K-Erweiterungspaket. Ferner ist ein Adapter erhältlich, der die Verwendung fast der gesamten Sinclair-Peripherie ermöglicht. Außerdem werden ein Centronics-Interface, eine Tonbox und ein Zusatz zur Erleichterung der Tastaturbedienung vertrieben.

Videoschaltung

Hier finden Sie noch eine ganze Baustein-Sammlung; für Rechner, die in kleinen Stückzahlen produziert werden, ist die Anfertigung besonderer Chips zu teuer.

Jupiter Ace

ABMESSUNGEN

215 x 190 x 30 mm

GEWICHT

246 g

ZENTRALEINHEIT

Z80A

TAKTFREQUENZ

1 MHz

SPEICHER

3 KByte RAM, extern auf 51K erweiterbar; 8 KByte ROM.

BILDSCHIRMDARSTELLUNG

Schwarzweiß mit 22 Zeilen zu 32 Zeichen, bzw. 64 x 48 bei Grafik.

SCHNITTSTELLEN

Fernseher- und Cassettenrecorderanschluß, 9-V-Versorgungsbuchse; zwei Platinenstecker, wovon der eine alle Daten- und Adreßleitungen des Prozessors führt, der andere nur die Daten- und einige Steuerleitungen.

MITGELIEFERTE SPRACHE

FORTH

MITGELIEFERTES ZUBEHÖR

Steckertransformator, Cassettenrecorder- und TV-Anschlußkabel.

TASTATUR

Gummitasten wie beim Sinclair Spectrum, aber weicher und unpräziser, die genau mittig gedrückt werden müssen; alle mit Wiederholfunktion und Zweifachumschaltung, was die Erzeugung sämtlicher ASCII-Codes ermöglicht.

HANDBUCH

Das Handbuch ist sehr ausführlich und enthält alle wichtigen Informationen. Vom gleichen Verfasser stammen auch die Handbücher für den ZX81 und den Sinclair Spectrum. Auf 180 Seiten wird eine Einführung in FORTH und eine ausführliche Beschreibung des Jupiter Ace gebracht, beides durch zahlreiche Beispiele illustriert.

Zwei Bedingungen

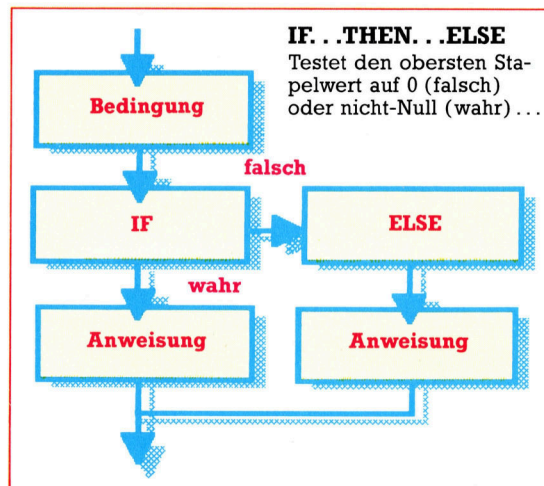
In unserer FORTH-Serie haben wir bisher einfache, gradlinige Routinen behandelt, aber keine Strukturen, mit denen sich der Programmfluß steuern ließe. In dieser Folge sehen wir uns mehrere dieser Strukturen an, darunter IF...THEN...ELSE und die DO-Schleife.

Wie jede andere Programmiersprache besitzt auch FORTH eine ganze Reihe Strukturen, die den Programmfluß steuern. Wenn Sie Sprachen wie PASCAL, C oder die hochentwickelten BASICs bereits kennen, werden Sie die folgenden Abläufe leicht verstehen. Durch die Mechanik des FORTH-Stapels sehen die Strukturen jedoch vielfach „seitenverkehrt“ aus.

Steuerstrukturen können in FORTH nur innerhalb von Colon-Definitionen eingesetzt werden, da die Verzweigungsbefehle erst in bedingte und absolute Sprünge compiliert werden müssen. Dieser Vorgang bräuchte aber viel Zeit, die dann während der Programmausführung nicht zur Verfügung stehen würde. Deshalb optimiert FORTH die Abläufe schon beim Eintrag ins Vokabular.

Hier die Abläufe des Standard-FORTH:

- **Bedingung IF Anweisung wahr ELSE Anweisung falsch THEN**



Wenn die Bedingung „wahr“ ist, wird „Anweisung wahr“ ausgeführt. Ist sie „falsch“, die „Anweisung falsch“. Wie in den meisten Computersprachen kann der ELSE-Teil und die „Anweisung falsch“ auch weggelassen werden. In diesem Fall geht FORTH sofort auf die Anweisung hinter THEN über, wenn die Bedingung „falsch“ ergibt.

Die Bedingung und die beiden Anweisungen (wahr und falsch) können beliebig viele Forth-Wörter sein, sogar andere IF...THEN...

ELSE sind möglich. Die Bedingung wird dabei auf den Stapel geschoben und von IF wieder heruntergezogen. Hier ein Beispiel, wie angezeigt wird, ob eine Zahl des Stapels gerade oder ungerade ist:

```

:PARITAET ( n-- )
    ( zeigt an, ob n "gerade" oder
    "ungerade" ist )
    DUP
    ."ist"
    2 MOD 0=IF
    ."gerade"
    ELSE
    ."ungerade"
    THEN
  
```

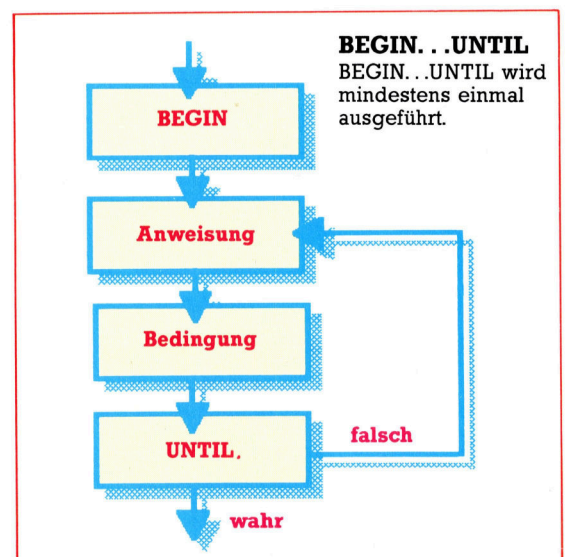
MOD liefert bei der Teilung der ersten Zahl durch die zweite den Rest.

- **BEGIN...UNTIL**

führt eine Schleife aus, solange die Bedingung wahr ist. Die meisten modernen Computer und auch die neueren BASIC-Dialekte verfügen über eine ähnliche Struktur.

BEGIN Anweisung Bedingung UNTIL

Es werden die Schleife und die Bedingung ausgeführt. Die Bedingung ist das Ende des Schleifenteils und beläßt einen Wert auf dem Stapel, mit dem UNTIL weiterarbeiten kann. Der Schleifenteil muß daher mindestens einmal ausgeführt werden:




```

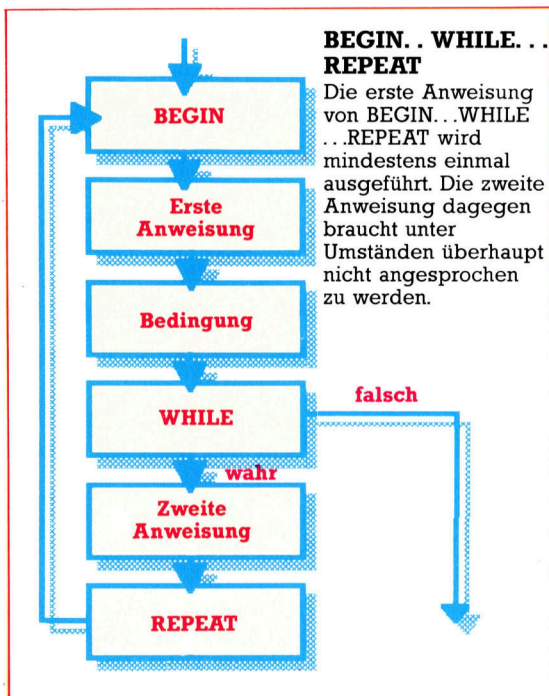
:HOCH2      ( -- )
            ( zeigt alle Quadratzahlen von
              2 an, die unter 10000 liegen )
1
BEGIN
  CR DUP( Quadratzahl auf eine neue
        Zeile setzen )
  2* ( nächste Quadratzahl von 2 holen )
  DUP 1000 > UNTIL ( testen, ob die Zahl
                  schon zu groß ist )
  DROP ( letzte Quadratzahl nicht anzeigen )

```

● BEGIN...WHILE...REPEAT

WHILE ist flexibler als UNTIL, da Sie im Inneren des Schleifenteils entscheiden können, ob die Schleife beendet werden soll, und Sie nicht erst das Ende abwarten müssen.

BEGIN 1. Schleife Bedingung WHILE 2. Schleife REPEAT



Wie in UNTIL wird auch hier die 1. Schleife zumindest einmal ausgeführt und die Bedingung abgefragt. An dieser Stelle gibt es jedoch zwei wesentliche Unterschiede zu UNTIL. Wenn die Bedingung „falsch“ ergibt, wird die 2. Schleife übersprungen und damit die Schleife beendet. Ergibt sie „wahr“, führt FORTH die 2. Schleife aus, bevor sie wieder auf BEGIN springt.

● DO...LOOP

BASIC

```

FOR X=Anfang TO Ende
  Schleifeninhalt
NEXT X
FOR X=Anfang TO Ende
  Step
  Schleifeninhalt
NEXT X

```

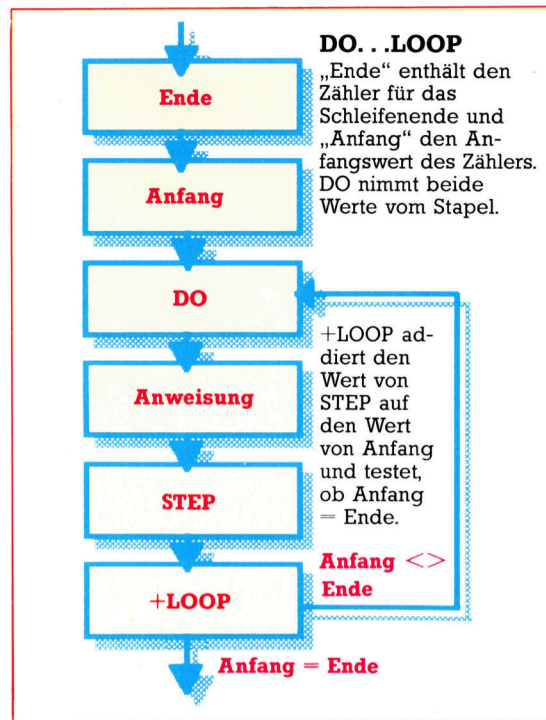
FORTH

```

( Ende+1 ) Anfang DO
  Schleifeninhalt
LOOP
( Ende+1 ) Anfang DO
  Schleifeninhalt
Step +LOOP

```

Die DO-Schleife in FORTH entspricht den FOR-Schleifen in BASIC und vielen anderen Sprachen. Die oben angeführte tabellarische Gegenüberstellung zeigt die auffälligen Unterschiede zwischen FORTH und BASIC.



Zunächst steht der Anfangs- und Endwert vor dem DO, damit das DO sie vom Stapel nehmen kann. Ende wird deshalb auch zuerst ausgeführt. Es ist allerdings weniger deutlich, warum Ende um eins größer sein muß als die Anzahl der Schleifendurchläufe. Die Ursache liegt darin, daß 4 0 DO die Schleife mit den Werten 0, 1, 2 und 3 (also insgesamt viermal) durchläuft, nicht aber mit dem Wert 4.

Wie in BASIC ist STEP möglich. Wird die Angabe weggelassen, nimmt LOOP als STEP automatisch 1 an. Wird STEP jedoch angegeben, muß +LOOP statt LOOP eingesetzt werden,

Bedingungen

Die folgenden Wörter eignen sich für den Einsatz mit IF, UNTIL und WHILE. Ihre Ergebnisse werden auf dem Stapel abgelegt und sind in FORTH-83 entweder -1 (wahr) oder 0 (falsch). Ältere FORTH-Versionen verwenden 1 für wahr und 0 für falsch.

IF, UNTIL und WHILE benötigen nicht die Angaben 0, 1 oder -1. 0 zeigt einfach falsch an, während jede andere Zahl wahr ist.

```

= m,n--wahr oder falsch (wahr IF m = n)
< m,n--wahr oder falsch (wahr IF m < n)
> m,n--wahr oder falsch (wahr IF m > n)
0= m,n--wahr oder falsch (wahr IF m = 0)
0< m,n--wahr oder falsch (wahr IF m < 0)
0> m,n--wahr oder falsch (wahr IF m > 0)
NOT wahr oder falsch--wahr oder falsch
AND m,n--m AND n
OR m,n--m OR n

```


wobei STEP unmittelbar vor +LOOP steht.

In FORTH gibt es keine Steuervariable (wie das X in BASIC). Statt dessen schiebt das Wort I den Schleifenwert auf den Stapel. Wenn mehrere DO-Schleifen ineinander verschachtelt sind, bezieht sich I immer auf die innerste und J auf die nächstäußere. Auf diese Weise ist sichergestellt, daß bei mehrfach verschachtelten Schleifen die entsprechenden Werte intakt sind und korrekt gezählt werden.

Hier ein Programmbeispiel, das eine Zahl mit einer anderen potenziert. Da FORTH im Gegensatz zu BASIC keinen Standardoperator dieser Art besitzt, müssen Sie ihn selbst definieren:

```

: ** ( m,n--m**n)
  DUP 0 < IF ( IF n < 0 ist das Ergebnis 0)
    DROP DROP 0
  ELSE
    DUP 0 = IF ( IF n=0 ist das Ergebnis 1)
      DROP DROP 1
    ELSE
      1 ( multipliziert n mal mit m)
      SWAP 0 DO ( Schleife n mal durch-
        laufen)
        OVER * ( ersten Wert des Stapels
          mit m multiplizieren)
        LOOP
      SWAP DROP ( m löschen)

```

Druckfunktion

Strings im Inneren einer Colon-Definition lassen sich mit folgendem Format ausdrucken: ."String"

Auf ." muß mindestens ein Leerzeichen folgen, da es ein Wort ist.

Weitere Leerzeichen werden als Teil des String angesehen.

Der String steht hinter dem ." (und folgt damit nicht der Umgekehrten Polnischen Notation), da der Stapel keine Strings verarbeiten kann. Für die Verarbeitung von Stringvariablen gibt es im Standard-FORTH keine Abläufe. Sie können die Sprache jedoch leicht um diese Möglichkeit erweitern.

Das Wort CR (Return) spricht auf dem Bildschirm eine neue Zeile an.

Grenzen von DO

Obwohl das Grundkonzept von DO recht einfach ist, gibt es einige Merkwürdigkeiten: Wenn STEP (vor +LOOP) negativ ist, kann der Wert von LOOP bereits das Ende erreichen, ohne davor anzuhalten. 2 0 DO...-1 +LOOP durchläuft daher die Werte 0, -1 und -2.

Wir haben weiterhin gesehen, daß DO-Schleifen mindestens einmal ausgeführt werden, selbst 0 0 DO...LOOP. FORTH-83 hat hier eine Überraschung parat: Es führt diese Schleife 65.536 mal aus, -1 0 DO...LOOP 65.535mal und -2 0 DO...LOOP 65.534mal etc.

Hier wäre es am leichtesten, überhaupt nicht mit diesen Werten zu arbeiten. Wenn Sie jedoch wissen, wie der Computer Ganzzahlen mit und ohne Vorzeichen verarbeitet, wird Ihnen auch der Grund dafür deutlich sein. Sobald der Anfangswert hinter Ende liegt, läuft der Zähler bis 32.768. Diese Zahl wird dann als -32.768 angesehen und aufwärtsgezählt, bis Ende von unten her erreicht ist.

Wenn bei älteren FORTH-Versionen der Anfangswert das Ende überschritten hat, wird die Schleife nur einmal ausgeführt. Diese Versionen behandeln jedoch auch LEAVE geringfügig anders. Hier springt das Programm nicht aus der Schleife heraus, sondern stellt sicher, daß auch beim nächsten LOOP oder +LOOP ein Ausstieg möglich ist.

THEN
THEN

Hier sind die Sonderfälle etwas kompliziert. Ist die Hochzahl negativ, dann ist die korrekte Antwort 1 dividiert durch das potentierte Ergebnis. Da FORTH nur mit Ganzzahlen arbeitet und sich mit diesem Programm die Teilung nicht exakt ausführen läßt, ist das Ergebnis jedoch 0. Der Fall n=0 ist subtiler. Wenn 1 nullmal mit m multipliziert wird (die Schleife also nullmal durchläuft), sollte das Ergebnis 1 lauten. Da FORTH seine Schleifen jedoch mindestens einmal durchläuft, muß dieser Fall besonders berücksichtigt werden. In der Praxis gibt es viele Situationen, in denen die Ausführung einer DO-Schleife nicht gewünscht wird und die abgefangen werden müssen.

Hier ein anschauliches Beispiel mit einer verbesserten Version von HOCH2. Dabei wird außer dem Schleifenwert I auch das Wort LEAVE eingesetzt, das die Schleife unterbricht und sofort auf die Anweisung hinter LOOP oder +LOOP springt:

```

: HOCH2 ( n-- )
  ( zeigt Exponenten und Potenzen
  von 2 an, solange die Potenzen
  kleiner als n sind)
  15 0 DO ( 2 ** 14 ist die größte Zahl, die sich
    anzeigen läßt)
    DUP 2 I * * < IF ( IF 2**I > n)
      LEAVE
    ELSE
      CR I. 2 I * *
    THEN
  LOOP
  DROP ( löscht n)

```

Bei diesen Programmstrukturen kommen Sie ohne Zeilennummern und GOTO aus. Wenn Sie an BASIC gewöhnt sind, kann der Ablauf anfangs etwas ungewohnt sein. Zeilennummern sind jedoch nur eine Eigenschaft von BASIC und haben nur wenig mit der allgemeinen Programmplanung zu tun.

Hätten Sie das DROP am Programmende berücksichtigt, wenn Sie HOCH2 selbst geschrieben hätten? Hier zeigt sich, daß der Einsatz des Stapels genau beobachtet werden muß.

Diese Aufgabe können Sie sich jedoch erleichtern, wenn Sie Ihre Wortdefinition so kurz wie möglich halten und die Abläufe unterteilen – wie in der zweiten Definition von HOCH2. Obwohl die neue Definition flexibler ist – Sie können einen Endwert eingeben und sind nicht an 10000 gebunden – ist sie nicht komplizierter. Wenn Sie dagegen die erste Version verändert hätten, wären Sie schon nach kurzer Zeit mit allen Arten von SWAP und OVER konfrontiert worden. Da Sie ** definiert und damit zwei kurze Wörter haben, ist die zweite Version wesentlich einfacher.



Die letzten Drei

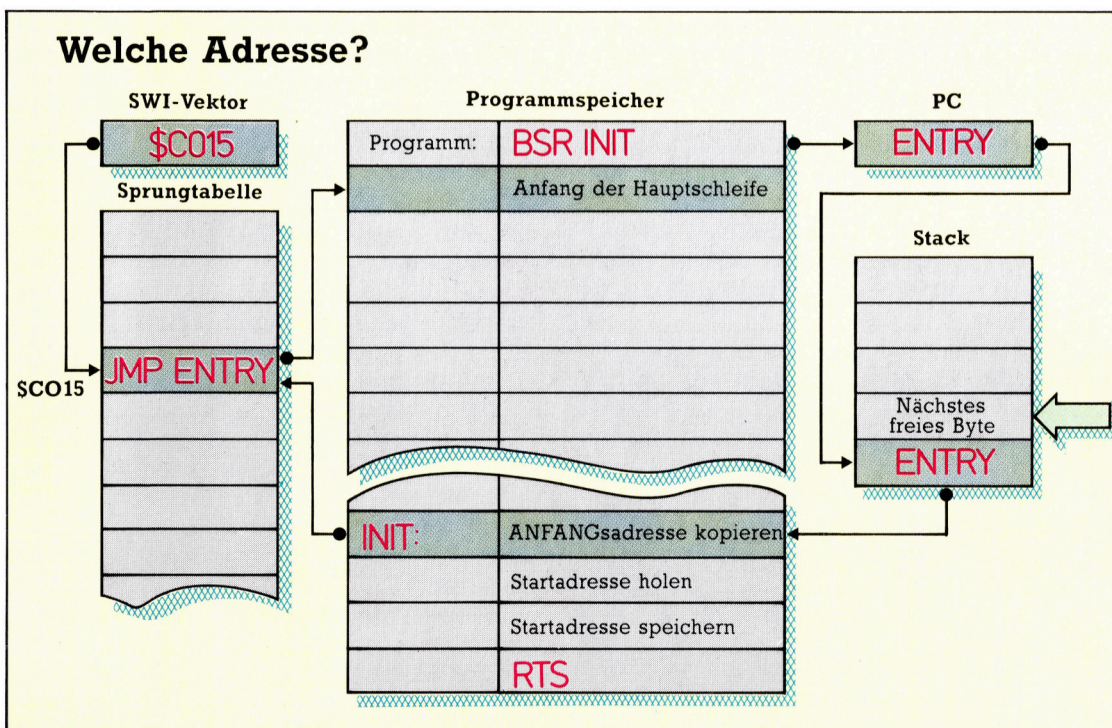
Unserem Debugger fehlen noch drei Befehle. Bevor wir uns jedoch damit beschäftigen, untersuchen wir zunächst den Interruptmechanismus, der an den Unterbrechungspunkten die Steuerung zwischen Debugger und dem analysierten Programm übernimmt.

Der Interruptmechanismus steuert die Unterbrechungspunkte, an denen wir einen Befehl des analysierten Programms durch den SWI-Op-Code (SoftWare Interrupt) ersetzt haben. SWI wird wie alle Interrupts des 6809 durch den Vektor einer bestimmten Speicherstelle geleitet – in diesem Fall \$FFFA. Bei der Ausführung eines SWI werden die Register auf den Stack geschoben, und der Prozessor lädt den Befehlszähler (PC) mit der 16-Bit-Adresse bei \$FFFA und \$FFFB. Die Programmausführung setzt sich dann von dieser Adresse an fort. Dieser Vektor muß nun so verändert werden, daß er auf die Einsprungadresse unseres Debuggerprogramms zeigt. Da Interruptvektoren jedoch fast immer fest im ROM gespeichert sind, bietet das Betriebssystem andere Möglichkeiten der Änderung.

Normalerweise gibt es im Arbeitsspeicher einen „Notzbereich“, in dem eine Sprungtabelle liegt. Dieser Notzbereich steht für Programme nicht zur Verfügung und wird nur vom Betriebssystem angesprochen. Die Adresse, auf die der Vektor zeigt, enthält üblicherweise einen Sprungbefehl auf eine weitere Adresse, die wieder in das Betriebssystem zurückführt.

Wenn wir diese Adresse ändern, kann der erste Befehl, der unmittelbar nach dem Interrupt ausgeführt wird, den Programmablauf auf die Anfangsadresse des Debuggers leiten. Der ursprüngliche Inhalt der Sprungtabelle muß jedoch vor Beendigung des Programms wiederhergestellt werden, da immer die Situation eintreten kann, daß das Betriebssystem ebenfalls einen SWI ausführt. Statt SWI lassen sich ebenso die anderen drei Software Interrupts des 6809 einsetzen: SWI2 (Op-Code 103F mit dem Vektor bei \$FFF4) oder SWI3 (Op-Code 113F mit dem Vektor bei \$FFF2) – wegen des Zwei-Byte-Formats wären dann allerdings einige Änderungen im Code des Debuggers notwendig.

Es gibt noch ein weiteres Problem: Unser Debugger kann nur in den Platz geladen werden, der nach dem Laden des analysierten Programms noch übrig ist. Sie haben sicher bemerkt, daß im Debuggercode alle Speicherstellen relativ angegeben wurden, das heißt mit Bezug auf den Befehlszähler. Wir müssen jedoch auch die absolute Einsprungadresse des Debuggers kennen, um sie in die Sprungtabelle einsetzen zu können.



Unser Debuggerprogramm beginnt mit einem BSR-Aufruf der Initialisierungsroutine, unmittelbar gefolgt von dem Anfang der Hauptprogrammenschleife. Die Initialisierung stellt unter anderem fest, wie die absolute Adresse am Anfang der Schleife lautet, und kopiert diese in die Einsprungtabelle für Interrupts. Bei der Ausführung eines Interrupts wird nun die Steuerung über diese Tabelle an die Startadresse der Hauptprogrammenschleife übergeben.



Da der Assembler diese Aufgabe nicht übernehmen kann, berechnet unser Programm während der Laufzeit als erstes diese Adresse und setzt sie in die Sprungtabelle ein. Beachten Sie, daß die SWI-Einsprungadresse anders lautet als die Anfangsadresse des Debuggers, da die Initialisierungsroutine am Programm-anfang bei einem späteren Aufruf über SWI nicht mehr gebraucht wird. Wenn wir nun die gesamte Initialisierung als Subroutine anlegen, liegt die Einsprungadresse unmittelbar hinter dem BSR-Aufruf der Initialisierungsroutine. Bequemerweise ist dies genau die Adresse, die von BSR auf den Stack geschoben wird. Wir müssen sie nur vom Stack ziehen und in die Sprungtabelle einsetzen.

Der Initialisierungsvorgang hat außerdem die Aufgabe, die Startadresse des analysierten Programms festzustellen. Hier der Aufbau:

B	Unterbrechungspunkt einsetzen
U	Unterbrechungspunkt löschen
D	Aktuelle Unterbrechungspunkte anzeigen
S	Programm starten
G	Nach einer Unterbrechung Programm an der Unterbrechung fortsetzen
R	Registerinhalt anzeigen
M	Speicherstelle untersuchen und ändern
Q	Programm beenden

Daten:

Vektor-Adresse befindet sich bei \$FFA in X

JMP-Op-Code ist der Op-Code für den

JMP-Befehl in A

Einsprungadresse ist die Adresse des Einsprungpunkts in Y

Startadresse ist die Adresse des analysierten Programms in D

Ablauf:

Vektor-Adresse holen

JMP-Op-Code in der Vektor-Adresse speichern

Einsprungadresse holen

Auf (Vektor-Adresse + 1) speichern

Startadresse von der Tastatur holen

Speichern

Wir können nun die drei fehlenden Befehle codieren. Bei R – Anzeigen der Registerinhalte – wollen wir natürlich nicht die Register beim Ablauf des Debuggers sehen, sondern die Werte am Unterbrechungspunkt des analysierten Programms. Diese Werte wurden jedoch von dem SWI-Befehl auf den Stack geschoben und inzwischen von anderen Werten überlagert. Wenn wir nun die Anzahl der darüberliegenden Bytes berechnen würden, hätten wir die gewünschten Werte. Es gibt aber einen einfacheren Weg: Unmittelbar nach Aufruf des SWI speichern wir den Wert des Stack-Pointers und verwenden ihn später als festen Bezugspunkt.

Für die Codierung des Befehls R setzen wir voraus, daß der Wert des Stack-Pointers zur Verfügung steht und daß wir die Registerinhalte abrufen können. Die Struktur der Routine ist einfach – die Werte werden nicht vom Stack gezogen, sondern gelesen und mit den entsprechenden Bezeichnungen dargestellt. Nur das Register S muß errechnet werden, da es die Position vor dem Interrupt anzeigt. Den korrekten Wert von S erhalten wir, wenn wir eine entsprechende Zahl auf die nach dem SWI-Aufruf gespeicherte Bezugsadresse der Stackwerte addieren. Hier der logische Aufbau in der gewohnten Notation.

Daten:

Stack-Pointer ist die Adresse der Stackspitze nach dem Interrupt (in X)

Ein-Byte-Wert ist der Wert des Ein-Byte-Registers in B

Zwei-Byte-Wert ist der Wert des 16-Bit-Registers in D

Namen sind die Namen (Bezeichnungen) der neun Register

Ablauf:

Stack-Pointer holen

CC in Ein-Byte-Wert laden

Namen(1) darstellen, Ein-Byte-Wert

Diesen Vorgang für A, B und DP wiederholen

X in Zwei-Byte-Wert laden

Namen(5) darstellen, Zwei-Byte-Wert

Diesen Vorgang für Y, U und PC wiederholen

12 zum ursprünglichen Wert des Stack-Pointers addieren

Namen(9) darstellen, Stack-Pointer

Von den beiden verbleibenden Befehlen braucht Q – Programm beenden – keine besondere Routine, und G löst die Wiederaufnahme eines Programms nach einem Unterbrechungspunkt aus. Dafür muß der SWI-Code, mit dem wir die Unterbrechung ausgelöst hatten, gegen den ursprünglichen Befehl dieser Position ausgetauscht und die Steuerung wieder an diesen Befehl übergeben werden. Mit RTI (Rücksprung vom Interrupt) lassen sich auch die Register leicht wieder mit ihren ursprünglichen Werten belegen. Bei dem PC müssen wir jedoch darauf achten, daß der vom Stack gezogene Wert auf den nächsten Befehl zeigt. Dieser Wert wurde um eins erhöht, muß daher noch vor dem Rücksprung korrigiert werden.

Daten:

Unterbrechungstabelle ist eine Tabelle mit den 16-Bit-Adressen der Unterbrechungspunkte

Entfernte-Werte ist die Tabelle mit den gegen SWIs ausgetauschten Op-Codes

Nächster-Unterbrechungspunkt ist ein Wert im Bereich von 1 bis 16

Stack-Pointer ist der Wert des Stack-Pointers nach dem SWI

Ablauf:

IF Nächster-Unterbrechungspunkt > 0
AND <= 16 THEN

Den Op-Code von Entfernte-Werte
(Nächster-Unterbrechungspunkt) holen
An der Adresse Unterbrechungstabelle
(Nächster-Unterbrechungspunkt)
speichern

S auf den Stack-Pointer setzen

Den Wert des PC auf dem Stack dekrementieren

Nächster-Unterbrechungspunkt inkrementieren

Rücksprung vom Interrupt

ELSE

Rücksprung von der Subroutine



Initialisierung

START	RMB	2	Startadresse speichern	LDA	,Y+	Zweites Zeichen des Namens
OPJMP	FCB	\$0E	JMP-Op-Code	BSR	OUTCH	Anzeigen
INIT	LDX	\$FFFA	Vektor-Adresse holen	LDA	SPACE,P-CR	Leerzeichen anzeigen
	LDA	OPJMP,PCR	JMP-Op-Code holen und	BSR	OUTCH	
	STA	,X+	in Vektor-Adresse speichern	LDD	,X++	Nächstes Register in
	LDY	1,S	Einsprungsadresse vom Stack holen			Zwei-Byte-Wert laden
	STY	,X	Auf Vektoradresse + 1 ablegen	BSR	DSPADD	Zwei-Byte-Wert anzeigen
	BSR	GETADD	Startadresse von der Tastatur holen	PULS	A,PC	A wiederherst. und Rücksprung
	STD	START,PCR	Speichern			
	RTS		Rücksprung			

Der Befehl R

STACKP	RMB	2	Stack-Pointer
LABELS	FCB	'CC A BDP X Y UPC S'	
SPACE	FCB	32	ASCII-Code für Leerzeichen
CMDR	PSHS	A,B,X,Y	Eingesetzte Register sichern
	LDX	STACKP,PCR	Stack-Pointer holen
	LEAY	LABELS,PCR	Mit Y auf den Namen zeigen
	LDA	#4	Nummer der Ein-Byte-Register
FOR01	BSR	CMDR1	Nächstes Register viermal anzeigen
	DECA		
	BGT	FOR01	
	LDA	#4	Nummer der Zwei-Byte-Register
FOR02	BSR	CMDR2	Nächstes Register viermal anzeigen
	DECA		
	BGT	FOR02	
	LDA	,Y+	Erstes Zeichen des Namens
	BSR	OUTCH	Anzeigen
	LDA	,Y+	Zweites Zeichen des Namens
	BSR	OUTCH	Anzeigen
	LDA	SPACE,PCR	Leerzeichen anzeigen
	BSR	OUTCH	
	TFR	X,D	X enthält nun den gewünschten Wert von S
	BSR	DSPADD	S anzeigen
	PULS	A,B,X,Y,PC	Reg. wiederherst. und Rücksprung

* Subroutine für das Anzeigen eines Ein-Byte-Registers

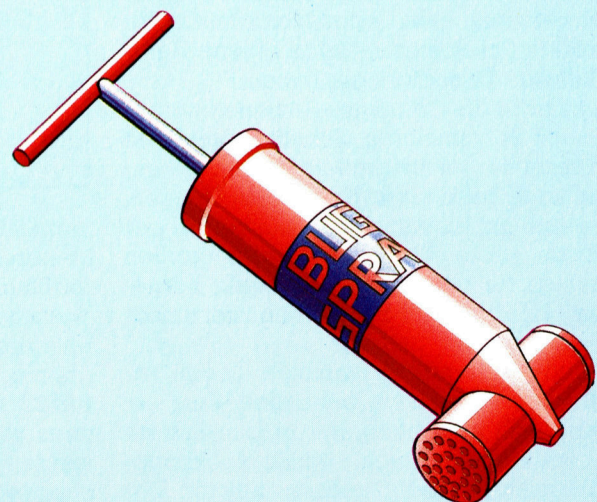
CMDR1	PSHS	A	A sichern
	LDA	,Y+	Erstes Zeichen des Namens
	BSR	OUTCH	Anzeigen
	LDA	,Y+	Zweites Zeichen des Namens
	BSR	OUTCH	Anzeigen
	LDA	SPACE,PCR	Leerzeichen anzeigen
	BSR	OUTCH	
	LDB	,X+	Nächstes Register in Ein-Byte-Wert laden
	BSR	DSPVAL	Ein-Byte-Wert anzeigen
	PULS	A,PC	A wiederherst. und Rücksprung

* Subroutine für das Anzeigen eines Zwei-Byte-Registers

CMDR2	PSHS	A	A sichern
	LDA	,Y+	Erstes Zeichen des Namens
	BSR	OUTCH	Anzeigen

Der Befehl G

BPTAB	RMB	32	Unterbrechungstabelle
REMTAB	RMB	16	Entfernte-Werte
NEXTBP	RMB	1	Nächster-Unterbrechungspunkt
CMDG	PSHS	A	A sichern (für norm. Rücksprung)
	LDA	NEXTBP,PCR	Nächster-Unterbrechungspunkt
IF04	BLE	ENDF04	IF Nächster-Unterbr.punkt > 0
	CMPA	MAXBP,PCR	AND <- 16
	BGT	ENDF04	(Maximale Zahl der Unterbrechungspunkte)
	DECA		In Tabellen-Offset umwandeln
	LEAX	BPTAB,PCR	Adresse der Unterbrechungstabelle
	LEAY	REMTAB,PCR	Adresse von Entfernte-Werte
	LDB	A,Y	Entfernte-Werte holen
	LSLA		A in Offset für 16-Bit-Tab. umwand.
	STB	[A,X]	An der Adresse der Unterbrechungstabelle speichern
	LDS	STACKP,PCR	Stack-Pointer aus S holen
	DEC	10,S	Wert d. PC auf Stack korrigieren
	INC	NEXTBP,PCR	Nächster-Unterbr.punkt inkrem.
ENDF04	RTI		Rücksprung vom Interrupt
	PULS	A,PC	Reg. wiederherst. und Rücksprung





Entwurf

Weiter geht's mit einem neuen Selbstbau-Projekt: Zum Anschluß an den Commodore 64, Spectrum oder Acorn B soll ein kleines Gerät entwickelt werden, das Gegenstände greifen und umherbewegen kann. Im ersten Teil wollen wir verschiedene Konstruktionsmöglichkeiten vorstellen und vergleichen.

Am Anfang jeder Konstruktion steht die Frage nach den gewünschten Eigenschaften der neuen Maschine. In unserem Fall soll das Gerät sich von einem Ausgangspunkt zu einem Zielobjekt (etwa von der Größe einer Garnrolle) hinbewegen, dieses aufheben, es in eine Schachtel legen und mit einer Genauigkeit von 2,5 mm wieder an den Startpunkt zurückkehren. Die einzelnen Positionen sollen im voraus bekannt sein. Eine Maschine dieser Art könnte etwa in einer Fabrik am Fließband Waren verpacken.

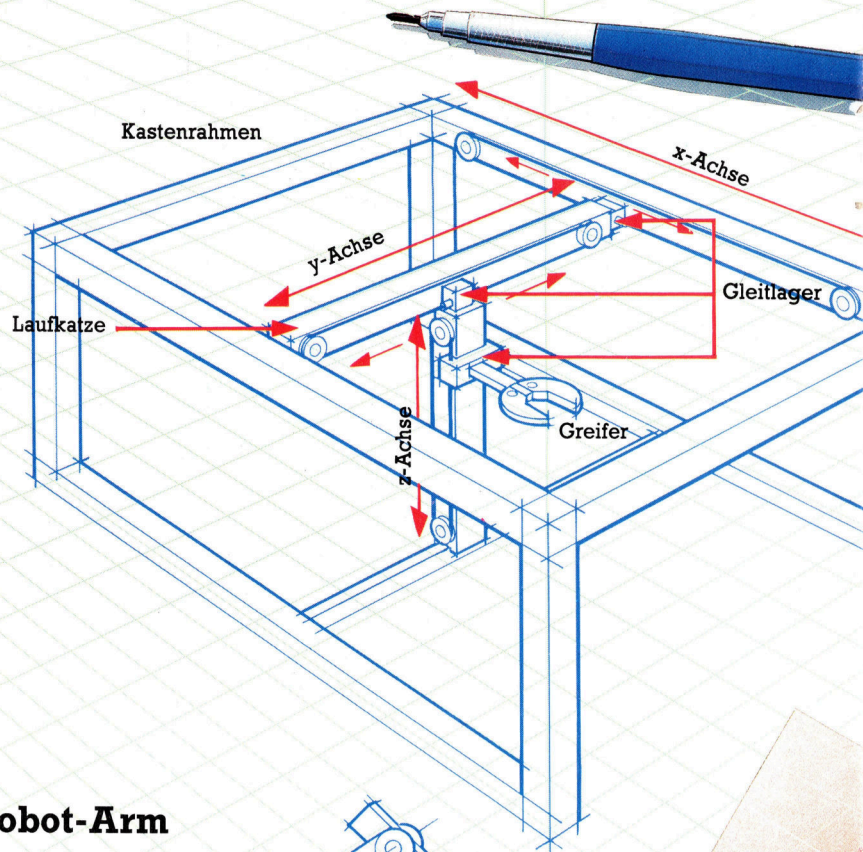
Was wird nun dafür gebraucht? Wir benötigen auf jeden Fall mehrere Motoren. Es könnten zwar auch Hydraulik- oder Druckluftantriebe verwendet werden, wir wollen uns aber auf elektrische Motoren beschränken.

Drei Typen von E-Motoren kommen für uns in Frage. Der einfachste ist ein normaler Gleichstrommotor. Dieser Motor ist preiswert, mit Digitalsignalen aber nur sehr schwer genau zu steuern. Um mit ihm die nötige Präzision zu erzielen, müßte er mit einem Drehzahlmesser versehen sein. Jeder Motor müßte für eine exakte Positionierung eine eigene Hard- oder Software-Decodierlogik haben.

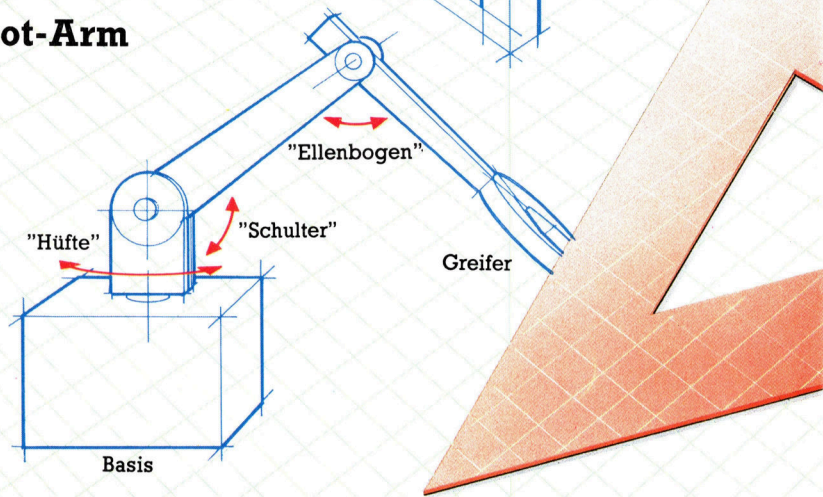
Einfacher ist die Computer-Ansteuerung von Schritt- und Servomotoren. Schrittmotoren sind jedoch nicht nur relativ groß und teuer, sie benötigen auch besondere Treiberschaltungen. Für Systeme mit kleinen Abmessungen ist der Servomotor beste Wahl: Er wird in großen Stückzahlen für den Modellbau-Markt hergestellt, ist fast überall zu bekommen und zudem relativ preiswert.

Die Funktion eines Servomotors haben wir im Selbstbau-Kurs schon behandelt. Kurz zusammengefaßt besteht ein Servomotor-System aus einem kleinen, durch digitale Rückkopplung überwachten 5-V-Gleichstrommotor. Der von einem Miniaturpotentiometer gemessene Drehwinkel der Motorachse wird durch ein eingebautes IC mit den Steuersignalen verglichen. Als Steuersignal dient ein 5-V-Impuls von ein oder zwei Millisekunden, wobei die

Laufkatze



Robot-Arm



Impulsdauer den Drehwinkel festlegt. Die Steuerimpulse müssen alle 20 Millisekunden (50 Hz) wiederholt werden.

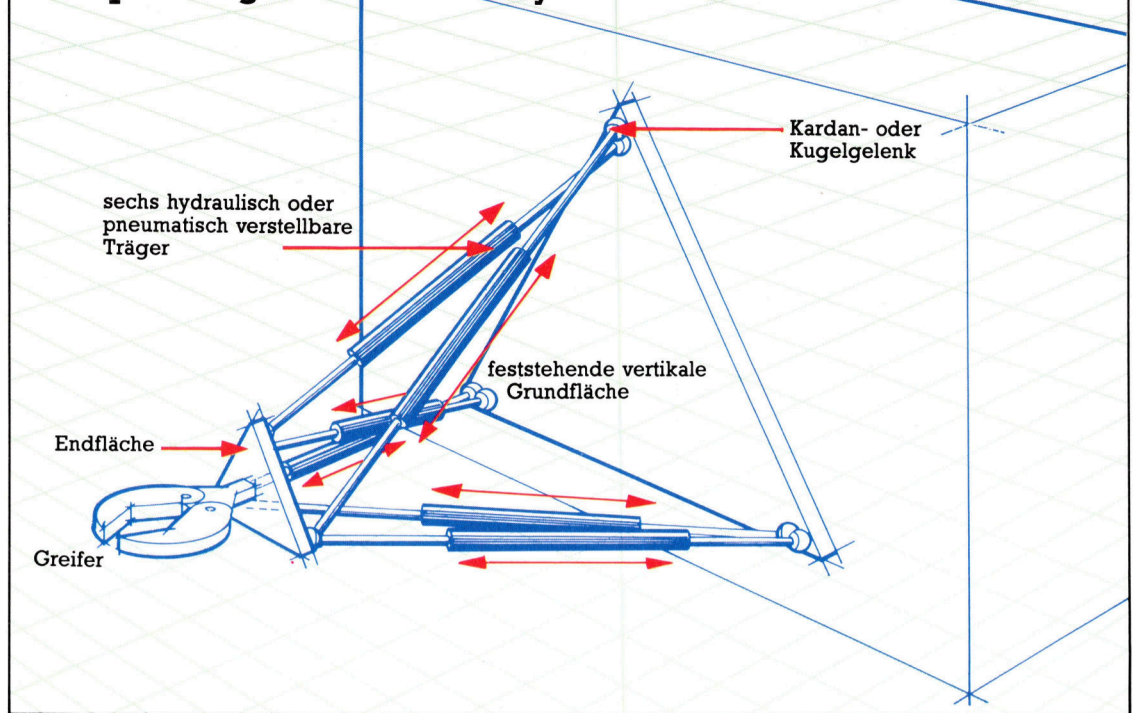
Auch auf die Treiber-Software sind wir bereits eingegangen. Sie nutzt ein Interrupt-Programm, bei dem das im Vordergrund laufende BASIC in regelmäßigen Abständen unterbrochen wird. Während dieser Unterbrechungen werden die Steuerimpulse zu den entsprechenden Motoren geleitet.

Für den mechanischen Aufbau einer Maschine nach unseren Spezifikationen gibt es mehrere Möglichkeiten. Was muß die Maschine können? Zum Aufheben eines Gegenstands (Garnrolle) müssen wir einen Greifme-

Hier sind zwei verschiedene Entwürfe für Maschinen abgebildet, die eine Garnrolle aufheben und in eine Schachtel legen können. Die Laufkatzen-Konstruktion hat eine sehr einfache Geometrie, kann aber ohne Umbau nicht die komplizierteren Aufgaben eines Gelenkarms ausführen. Die Steuerung des Armes ist allerdings für lineare Bewegungen sehr viel schwieriger.



Komplexes geometrisches System



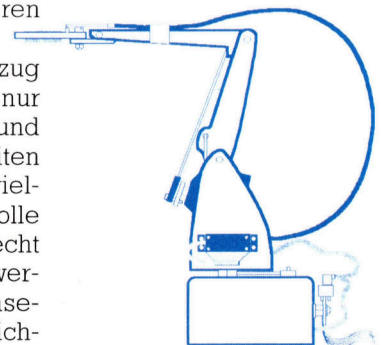
Durch Computer-Steuerung elektrischer Motoren werden sehr vielseitige Maschinen realisierbar. Für das reibungslose Zusammenwirken mehrerer Motoren sorgen geschickt gestaltete Programme, die dem Auge des Anwenders meist verborgen bleiben. Versuchen Sie einmal, sich alle Bewegungsmöglichkeiten dieses Sechsen-Achsen-Systems vorzustellen, bei dem alle Achsen über ein Computer-Programm gesteuert werden müssen.

Eine Möglichkeit wäre die Konstruktion einer kleinen Laufkatze in einem quaderförmigen Rahmen. Laufkatze und Greifer könnten verschiebbar gelagert sein und über Seil- oder Kettenzüge von den Motoren bewegt werden. Dazu bräuchten wir insgesamt vier Motoren – drei für die Positionierung und einen zum Öffnen und Schließen des Greifers.

Ein anderer Weg führt zum Gelenkarm, einer eher menschenähnlichen Lösung: Hier sorgt je ein Motor für die Bewegung in einem der Gelenke. Digitale Servomotoren sind für diesen Zweck gut geeignet, weil sich der Drehwinkel über die Software sehr einfach und exakt steuern läßt. Die genaue Positionierung eines Roboter-Arms ist allerdings problematischer als das Bewegen der Laufkatze: Um den Greifer nämlich in nur einer Ebene – etwa auf der x-Achse – zu bewegen, müssen schon mindestens zwei Motoren synchron angesteuert werden. Bei der Laufkatze genügt ein Motor für diese Bewegung. Da wir den Robot-Arm aber mit dem Computer steuern, gibt es keine Probleme: Maßgeschneiderte Steuerprogramme nehmen uns die für das Positionieren nötigen komplexen Berechnungen ab.

Der Gelenkarm ist einer Laufkatze in bezug auf Flexibilität überlegen. Die Laufkatze ist nur auf bestimmte Bewegungen zugeschnitten und kann ohne Umbau keine anderen Tätigkeiten ausführen. Der Gelenkarm hingegen ist vielseitig: Er kann zum Beispiel unsere Garnrolle auch dann aufheben, wenn sie nicht aufrecht steht, sondern auf dem Boden liegt. Wir werden also den Robot-Arm zur Grundlage unseres Selbstbau-Projektes machen. Die Zeichnungen stellen mögliche Variationen dar.

Während der nächsten Folgen des Projekts werden in dieser Strichzeichnung immer diejenigen Teile farbig herausgehoben, um die es im aktuellen Abschnitt gerade geht.



chanismus vorsehen, für den mindestens ein Motor gebraucht wird. Dieser Greifer soll auf drei Achsen bewegt werden. Sowohl in der x- und y-Achse (rechts-links bzw. vor-zurück) als auch in der z-Achse (oben-unten) soll die Maschine eine Bewegungsfreiheit von 10 cm haben. Jede Achse erfordert einen weiteren Motor. Alle Motoren zusammen sollen dem Greifer mit einer Genauigkeit von mindestens 2,5 mm positionieren können, damit er wieder zu seinem Startpunkt zurückkehrt. Die Klauen des Greifers sollten sich zumindest 3 cm weit öffnen lassen. Der Greifer muß so kräftig sein, daß er ein Gewicht von etwa 20 Gramm leicht tragen kann.

Nützliche Zeiger

Wir untersuchen Möglichkeiten der Programmverbesserung, sowie das Problem des Ladens und Speicherns der undefinierten Zeichensätze in einen speziellen Bereich des C-64-Speichers.

Nun, da unsere Zeichendefinitions-Programme geschrieben sind, lohnt es sich, die drei Versionen miteinander zu vergleichen und zu verbessern. Die C-64-Version wurde zuerst geschrieben, da sie die komplizierteste der drei Versionen ist. Der Bildschirmaufbau ist rudimentär, und es werden keine Farben, Töne oder Grafiken verwendet. In diesen Bereichen könnten also Verbesserungen durchgeführt werden, die wir hier jedoch nicht besprechen wollen.

Wir werden uns auf die Benutzerschnittstelle konzentrieren: Anweisungen, Hilfen, Befehlstasten und sonstige Möglichkeiten.

Im Programm gibt es keine Erklärungen, da das Listing auf eine Seite passen mußte. Beim Programmstart könnte jedoch eine Programmklärung ausgegeben werden. Außerdem ist wahrscheinlich auf dem Hauptbildschirm Platz für abgekürzte Hinweise – beispielsweise für Cursor-Bewegungen oder eine Befehlsliste. Dadurch sollte ein Hilfsbildschirm nicht unbedingt mehr nötig sein.

Die Wahl der Befehlstasten kann auch verbessert werden. Beim Acorn B und Spectrum wird der Cursor mit den normalen Kontrolltasten bewegt, wogegen beim Commodore die ungeshifteten Funktionstasten benutzt werden. Dies gestattet zwar eine bequeme Programmierung auf dem C 64, da die ASCII-Codes der acht Funktionstasten aufeinanderfolgend von 133 bis 140 liegen, doch die Anordnung der Tasten selbst ist nicht sehr logisch. Außerdem haben sie keine Wiederholfunktion. Letzteres kann durch POKE 650,128 geändert werden, die Anordnung der Tasten jedoch nicht, so daß man die Cursor-Kontrolle auf die Cursortasten umlegen sollte.

Eine weitere Verbesserung ist eine Strategie zur Cursor-Bewegung. Zur Zeit wird nur ein Be-

fehl abgefangen, durch den der Cursor das Fenster verlassen würde. Alternativ könnte der Cursor jedoch auch „umgeschlagen“ werden: Wenn er über den unteren Rand des Fensters bewegt wird, erscheint er wieder am oberen Rand und umgekehrt. Ähnliches gilt für horizontale Bewegungen. Dies zu programmieren ist einfach, doch erfordert es mehr Programmcode als die einfachen Abfragen in der Unteroutine ab Zeile 3500.

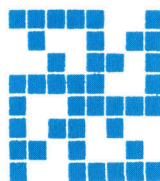
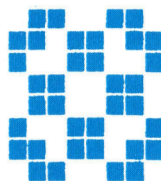
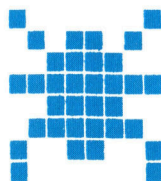
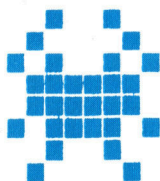
Beim Spectrum und Acorn B könnten das Speichern und Laden von Zeichensätzen als zusätzliche Befehle integriert werden. In allen drei Versionen wäre es sinnvoll, eine Zeichendefinition auf ein anderes Zeichen zu kopieren, so daß CHR\$(N) und CHR\$(N+1) dasselbe Zeichen repräsentieren. Falls Sie einen Ausdruck des neuen Zeichensatzes wünschen, kann eine Druck-Option eingefügt werden. Der modulare Aufbau des Programms macht die Ergänzungen einfach.

Dateien laden

Ein besonderes Problem des C-64-BASIC ist, daß der SAVE-Befehl scheinbar nur für den gesamten BASIC-Speicherbereich verwendet werden kann. Beim BASIC der anderen beiden Computer kann der zu speichernde Bereich angegeben werden. Der LOAD-Befehl des Commodore gestattet das Einladen von Dateien in jeden beliebigen Bereich. Läßt sich also das SAVE-Problem lösen, können die neuen Zeichensätze gespeichert und eingeladen werden.

Der SAVE-Befehl des Commodore bezieht sich über zwei Adreß-Zeiger auf den BASIC-Speicherbereich – TXTTAB (Adressen 43 und 44) und VARTAB (Adressen 45 und 46). TXTTAB zeigt auf den Beginn des BASIC-Speichers (normalerweise ab Adresse 2048), wogegen VARTAB auf den Beginn des BASIC-Variablenbereichs zeigt. Da dieser anfängt, wo das BASIC-Programm endet, zeigt VARTAB direkt auf das Ende des BASIC-Speichers. Ändert man diese beiden Zeiger, so daß sie auf den Start und das Ende des neuen Zeichensatzes zeigen und führt dann den SAVE-Befehl aus, sollte das Problem gelöst sein.

Bevor wir das versuchen, muß die genaue Position des Zeichensatzes ermittelt werden. Die Unteroutine bei Zeile 61000 kopiert den ROM-Zeichensatz in einen 2-KByte-RAM-Block



ab Adresse 14336, und in Zeile 50 wird VARTAB auf einen Wert unterhalb dieser Adresse gesetzt, damit BASIC ihn nicht überschreibt. Auf diese Art beschneiden wir jedoch den Anwenderspeicher um 2/3. Für das Zeichengenerator-Programm spielt das keine Rolle. Will man jedoch ein Programm, das mehr als 12 KByte Speicher benötigt, mit einem neuen Zeichensatz verwenden, so wirft das ernsthafte Probleme auf. Ungünstigerweise ist es durch das Betriebssystem nicht möglich, den Zeichensatz ab einer höheren Adresse abzulegen. Die einzige Lösung ist daher, den Zeichensatz so tief wie möglich und BASIC über ihn zu legen. Dies ist möglich, indem man den TXTTAB-Zeiger ändert. Das kann jedoch nicht vom BASIC-Programm aus gemacht werden, und es muß erledigt werden, bevor das Zeichengenerator-Programm in den Speicher geladen wird.

Die notwendigen Schritte sind:

1) Laden und starten Sie Programm 1. Dies schreibt die notwendigen Befehle auf den Bildschirm, so daß Sie sie durch Drücken von RETURN im Direktmodus ausführen können.

2) Laden Sie das Zeichengenerator-Programm, und ändern Sie es wie folgt:

```
61100 CGEN=53248:NCGEN=2048
```

```
61500 POKE PO,(PEEK(PO)AND240)OR2
```

Löschen Sie außerdem Zeile 50.

3) Speichern Sie die neue Version.

4) Laden und starten Sie das Programm exakt wie bisher.

5) Wenn Sie Ihre Arbeit beendet haben, laden und starten Sie Programm 2. Ähnlich wie bei Programm 1 werden einige Befehle auf den Bildschirm geschrieben.

6) Die Zeiger TXTTAB und VARTAB werden durch Programm 2 so gesetzt, daß SAVE „dateiname“ den gesamten Zeichensatz zwischen 2048 und 4097 speichert. Wollen Sie zukünftig das Zeichengenerator-Programm starten, müssen Sie die eben beschriebene Sequenz (bis auf Punkt 2) wiederholen.

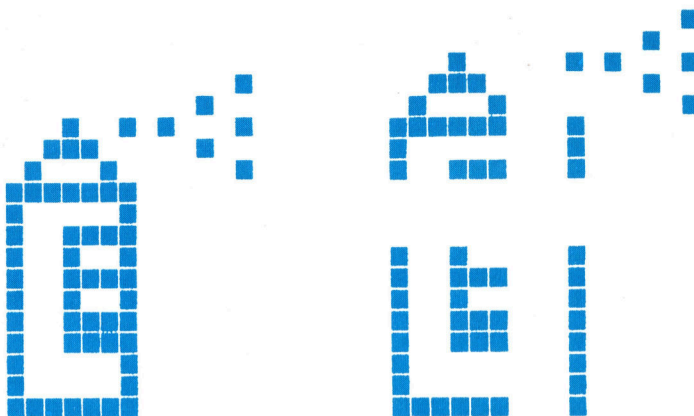
Wenn Sie den Zeichensatz wieder einladen wollen, müssen Sie Programm 1 laden und starten, um BASIC im Speicher hochzuschieben, und dann wie folgt den entsprechenden Zeichensatz laden:

```
LOAD "dateiname",DN,1
```

DN ist entweder 1 für Cassettenrecorder oder 8 für Diskettenstation. Die „1“ am Ende des Befehls wird Sekundäradresse genannt. Sie bezeichnet die Art des Commodore, Befehlsparameter an Peripheriegeräte zu übermitteln. Hier bedeutet sie, daß die Datei an den Platz geladen werden soll, von dem aus sie gespeichert wurde, anstatt durch den TXTTAB-Zeiger in den aktuellen BASIC-Programmspeicher. Dies ist möglich, da das Betriebssystem beim Speichern einer Datei die Startadresse im RAM als erstes Element der Datei ablegt.

Wenn Sie den LOAD-Befehl ohne Angabe verwenden, wird die Startadresse in der Datei zugunsten der Adresse von TXTTAB ignoriert.

Haben Sie BASIC umgelegt und den neuen Zeichensatz geladen, müssen Sie dem Betriebssystem die Position des Zeichensatzes mitteilen. Erklärungen hierzu finden Sie in der Tabelle und zudem auch in der neuen Version von Zeile 61500.



```
199 REM *****
200 REM*          PROGRAM 1          *
201 REM*          RUN THIS PROGRAM    *
202 REM*          THEN HIT RETURN TWICE *
203 REM*          THIS MOVES BASIC TO 4096 *
204 REM *****
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,16:POKE45,3:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE4098,0:CLR:NEW"
600 PRINT CHR$(19)
```

```
199 REM *****
200 REM*          PROGRAM 2          *
201 REM*          RUN THIS PROGRAM    *
202 REM*          THEN HIT RETURN TWICE *
203 REM*          THIS RESETS BASIC PTRS *
204 REM *****
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,8:POKE45,1:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE4098,0:CLR"
600 PRINT CHR$(19)
```

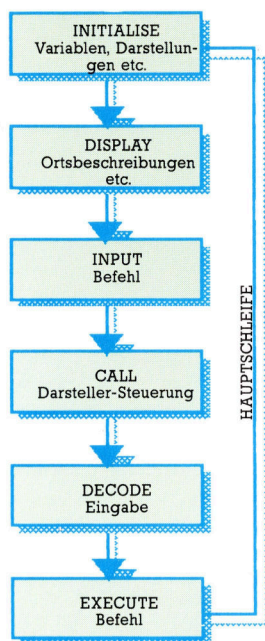
Wert von X=	Bits 3,2,1,0	angegebene Adresse
0	0000	0
2	0010	2048
4	0100	4096
6	0110	6144
8	1000	8192
10	1010	10240
12	1110	12288
14	1110	14336

Werte-Tabelle

Der Befehl „POKE 53272, (PEEK(53272) AND 240) OR X“ zwingt den Bildschirm-Chip, auf den Speicherbereich Zugriff zu nehmen, in dem der Zeichensatz abgelegt ist. Die Tabelle zeigt die Werte von X, die die Startadresse des RAM-Blocks setzen.

Aktive Akteure

Das Programmieren der Spieler/Darsteller-Interaktion in einem Abenteuerprogramm kann sehr kompliziert sein. Nachstehend stellen wir zwei Systeme der Darsteller-Steuerung vor.



Dieses Diagramm eines typischen Abenteuer-spiels zeigt eine mögliche Position unserer Darsteller-Steuerungs-Routine. Wird es an dieser Stelle in den Steuerungsfluß eingebracht, erfolgt die Ausführung der Routine jedesmal dann, wenn der Spieler nach Befehls-eingabe ENTER drückt.

Im ersten Teil dieser Reihe wurde dargelegt, wie sich die computergesteuerten Charaktere in Infocoms „Suspect“ scheinbar „aus eigenem Antrieb“ von Ort zu Ort bewegen. Bewegung ist aber eines der unwichtigsten Merkmale eines computergesteuerten „Daseins“, und das Programm enthält herausragende Beispiele für die wichtigere Spieler/Darsteller-Interaktion in seiner Befehls-Syntax. Beim Spielen von Suspect kann man Befehle benutzen, die die Adressierung von Darstellern auf unterschiedliche Art durchführen. Alle folgenden Angaben werden vom Programm akzeptiert:

Michael, wo ist Veronika?

Beschuldigen Sie Colonel Marston des Mordes?

Rufen Sie meinen Anwalt an

Johnson, erzählen Sie von sich

Offensichtlich gibt es genug Raum, um mit den anderen Gästen im Hause Ashcroft kommunizieren zu können. Was aber das Spiel so besonders eindrucksvoll macht, sind die zahlreichen Botschaften, die bei Ihren Versuchen, höfliche Konversation zu „machen“, als Antwort kommen. Wenngleich die agierenden Darsteller dazu neigen, sich zu wiederholen, sind sie doch imstande, Zusammenhang und Logik zum Ausdruck zu bringen. Dazu folgende Dialog-Beispiele:

Michael, erzähl mir vom Pferd.

„Lurking Grue ist Veronikas Favorit. Es ist ein wirklich schönes Tier.“

Marston, erzählen Sie mir vom Pferd.

„Ich weiß absolut nichts darüber, was Sie interessieren könnte.“

Die Darsteller können nicht nur mit dem Spieler kommunizieren, sie sind auch in der Lage, sich miteinander zu unterhalten. Und letztlich kommt man nicht weit, wenn man nicht einfach im Tanzsaal herumschlendert und Gesprächen wie diesem zuhört:

Michael mischt sich ins Gespräch ein. „Ich erinnere mich an einen schwarzen Hengst, der im vergangenen Jahr hoch gehandelt wurde. Er brachte an die Hunderttausend.“ Colonel Marston starrt ihn an und sagt, „Ich habe ein gutes Zahledegedächtnis. Der Spitzenpreis im vergangenen Jahr lag bei Zweiundneunzigtausend.“ Cochrane sieht Michael an, fühlt sich verraten. „Unfug“, sagt er ärgerlich . . .

Die „Menschen“ in Suspect erfüllen also eine Vielzahl von Anforderungen, die wir im ersten Teil als Voraussetzung für wahrhaft interaktive Charaktere gefordert haben:

1. Sie können sich von einem Ort zum anderen bewegen.
2. Sie sind vom Spieler ansprechbar und antworten sinnvoll.

Sie können auch Gegenstände aufnehmen und ablegen. Einer der Darsteller ist in einer Spielphase sogar in der Lage, einen Gegenstand in einem anderen zu verstecken! Und schließlich können die Darsteller den Spieler unaufgefordert ansprechen. Dafür gibt es im Spielverlauf mehrere Beispiele. Am deutlichsten wird das, wenn der Detektiv den Spieler festnimmt (weil man den wahren Schuldigen nicht gefunden hat).

Ein Nachteil eines solchen Spieles ist, daß es so stark von Verhalten und Aktionen der computergesteuerten Charaktere abhängt. Beispielsweise erfordert die Handlung in Suspect, daß in einem bestimmten Augenblick „jemand“ getötet wird. In bestimmten anderen Spielphasen führen die Darsteller bedeutungsschwere Aktionen aus, die es dem Spieler ermöglichen, das Geheimnis zu lösen, wenn er sie beobachtet. Aus diesem Grunde verhalten sich die Darsteller nicht völlig zufällig. Das bedeutet: Ist das Programm mehrere Male gelaufen, wird dem Spieler bewußt, daß zum Beispiel Charakter A an der Stelle B zum Zeitpunkt C ist und die Handlung D ausführt.

Charakter-Steuerung

Wir werden später noch sehen, daß die Fähigkeit der „Menschen“, bestimmte Handlungen auszuführen, sorgfältiger Überlegungen bedarf, damit eine wirksame Charakter-Steuerungs-Routine daraus wird. Beispielsweise demonstrieren die Charaktere in „The Hobbit“ mit ihrem Verhalten einen bemerkenswerten Grad an Zufälligkeit. Daraus könnte man folgern, daß sie in ihren Verhaltensmöglichkeiten und ihrer Intelligenz irgendwie beschränkt sind. Doch als Grundregel gilt: Ein hoher Zufalls-„Grad“ verbessert die lebensnahe Atmosphäre eines Programms.

Andererseits hat man bei Suspect nach mehrfachem Spielen des Programms das Gefühl des „déjà vu“, wenn beispielsweise die Märchen-Königin den Ballsaal verläßt, um Weinflecke aus ihrem Kleid zu entfernen, nur, um dann als Leichnam zu enden.

Nach der Betrachtung von Suspect wollen wir uns eigenen Routinen zuwenden. Die erste zu treffende Entscheidung: Wie komplex soll



unser Programm sein? Bei der Gestaltung eines Abenteuerprogramms ist es vergleichsweise einfach, die Anzahl der Örtlichkeiten zu bestimmen. Geht es aber um die Darsteller, gibt es fast unzählige Kombinationen. Ein Spiel wie *Suspect* erfordert die Speicherung großer Datenmengen, allein für die von den Charakteren gesandten Botschaften, einmal abgesehen von den für die Steuerung erforderlichen Routinen. Infocom löst dieses Problem durch ausschließliche Verwendung von Disketten. Viele Anwender aber können sich eine Diskettenstation nicht leisten. Deshalb müssen deren Programme völlig auf RAM-Basis lauffähig sein.

Das ist gar nicht so einengend, wie angenommen wird. Wir werden es in der nächsten Folge bei der Betrachtung von Melbourne Houses „*Sherlock*“ aufzeigen. Heute gestalten wir unser Programm so, daß maximal sieben Charaktere damit zu steuern sind.

Die zweite zu beantwortende Frage ist nicht so bedeutsam, muß aber vor Programmierbeginn beantwortet werden. Es gibt zwei Steuerungsmethoden – die synchrone und die asynchrone. Wir haben zu entscheiden, welche benutzt werden soll.

Aus dem Diagramm links ist ersichtlich, daß der Computer jedesmal die Eingabe decodiert, die erforderliche Routine aufruft, die erforderlichen System-Variablen aktualisiert, eine Meldung bringt und zu neuer Eingabe auffordert, wenn der Spieler nach einer Eingabe ENTER drückt. Die Frage ist: An welcher Stelle soll unsere Darsteller-Steuerung eingesetzt werden?

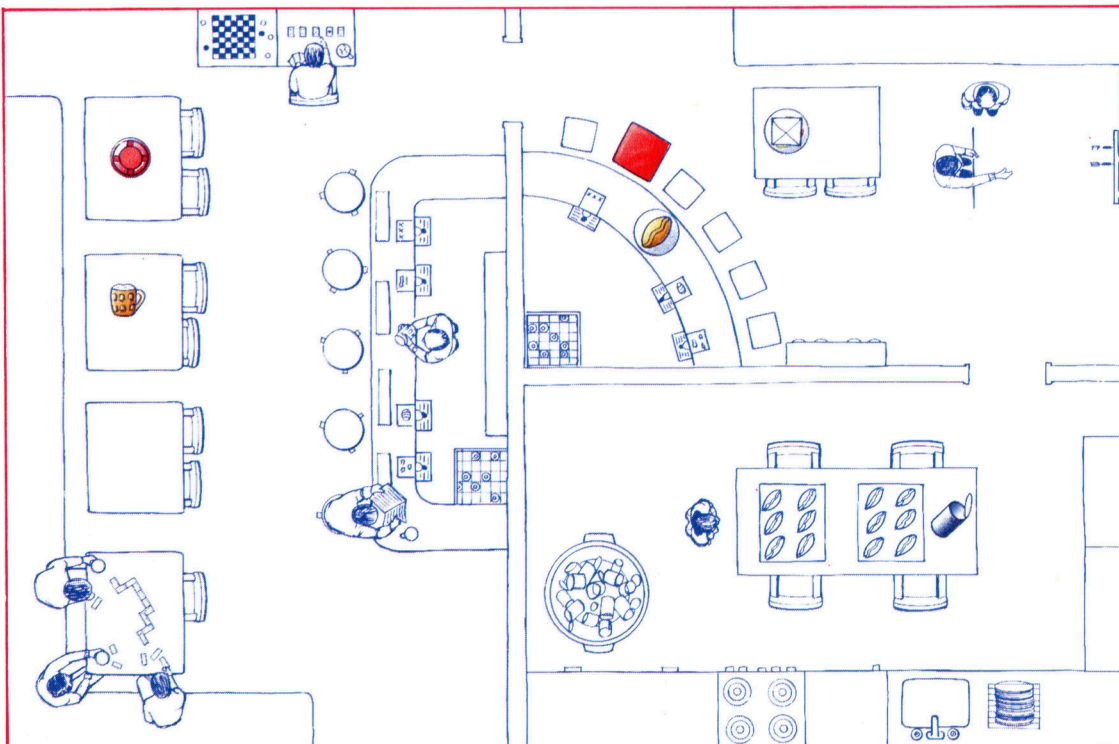
Um bei den erwähnten Begriffen zu bleiben: Ein synchroner Darsteller würde die entspre-

chende Routine in einem bestimmten Programmstadium ausführen. Dieses synchrone System hat bestimmte Vorteile für den Spieler. Geht man zum Beispiel weg, um in der Küche Tee zu trinken, hat man die Gewißheit, daß man in der Zwischenzeit nicht von einem anderen Akteur angegriffen wird. Kehrseite der Medaille ist, daß der Spielablauf zu sehr von den Eingaben des Spielers abhängt.

Bei einem asynchronen System hingegen erfolgt die Steuerung durch „interrupts“, die alles der Charakter-Steuerung überlassen, egal ob man an der Tastatur sitzt oder nicht. Dieses System findet bei „*The Hobbit*“ und „*Valhalla*“ Anwendung. Sitzt man vor dem Bildschirm und sieht einfach zu, kommen und gehen Leute und leben ihr „Leben“ völlig eigenständig. Dieses System hat den Vorteil, daß es Atmosphäre ins Programm bringt. Es ist aber sehr viel schwerer zu programmieren. Dabei ist zu beachten, daß die Kluft zwischen Charakter-Steuerung und dem eigentlichen Programm nicht zu groß werden darf, da es andernfalls ernsthafte Probleme geben kann.

Die Anwendung „interrupt“-gesteuerter BASIC-Routinen ist sehr schwer zu handhaben. Deshalb bedienen wir uns für unsere Routine des synchronen Systems.

Jetzt können wir mit dem Programmieren beginnen. Wir beabsichtigen zwar, ein übertragbares Modul zu schaffen, das in Abenteuerspielen lauffähig ist, benötigen dafür aber noch eine entsprechende Umgebung, um den Character-Steuerer zu überprüfen. Dazu erstellen wir ein einfaches Abenteuer mit drei Räumen, durch die die Darsteller bewegt werden können und in denen sich mehrere Gegenstände befinden, die sie bewegen müssen.



Unsere Darsteller-Steuerungsroute findet in einer Kneipe Anwendung, die für ihr Bier und die ausgezeichneten, selbstgemachten Pasteten bekannt ist. Die Orte, die in der Routine verwendet werden, zeigt nebenstehende Abbildung. Ebenso die Grundpositionen der wichtigsten Gegenstände. In der nächsten Folge zeigen wir, wie diese in die Routine eingesetzt werden, und ergänzen die Illustration in der Reihenfolge ihres Auftretens um die Darsteller.



Des Fehlers Tod

Wir beschließen unsere Serie über die Assemblersprache des 6809. Wir beenden alle unerledigten Aufgaben, geben einen Überblick über den Befehlsfluß und codieren das Hauptmodul.

Unser Hauptmodul aktiviert als erstes den Interruptmechanismus, mit dem wir Unterbrechungspunkte in das analysierte Programm einsetzen. Die Unterbrechungspunkte übergeben die Steuerung an den Debugger und ermöglichen es uns, die Inhalte der Register und Speicherstellen zu untersuchen. Über die Anfangsadresse des getesteten Programms geben wir die Steuerung wieder dorthin zurück (Befehl S). Die Befehle G und S übergeben die Steuerung an das getestete Programm, der Debugger dagegen erhält die Kommandos von den SWI-Codes der Unterbrechungspunkte.

Zwei Teile der Initialisierung hatten wir schon in der letzten Folge codiert. Dabei stand der Einsprungpunkt für Interrupts unmittelbar hinter dem Aufruf der Initialisierungsroutine. Unser erster Befehl sicherte den Stack-Pointer S, der später als Bezugspunkt für die von SWI auf den Stack geschobenen Registerwerte dient. Der nächste Initialisierungsschritt ermöglicht nun die Interpretation der Befehle.

Die Subroutinen für die Ausführung der Befehle sind fertig – ihr Aufruf (je nach Befehlseingabe) muß jedoch noch programmiert werden. Diese Aufgabe ließe sich durchaus mit einer Reihe verschachtelter IF-Befehle lösen, doch setzt die Routine „Befehl-Holen“ einen Offset in die Tabelle der Befehlszeichen, den wir mit einer Sprungtabelle einsetzen können. Diese Methode ist an dieser Stelle zwar nicht optimal, doch lohnt es sich, diese praktische Technik zu zeigen. Wir müssen dazu eine Sprungtabelle mit den Adressen der Subroutinen anlegen.

JMP kann – im Gegensatz zu Verzweigungsbefehlen – alle normalen Adreßarten einsetzen, darunter auch die indizierte und die indirekte Adressierung. Wenn wir X mit der Basisadresse der Tabelle laden und den Offset in B verwenden (in verdoppeltem Format, da die Tabelle 16-Bit-Werte enthält und nicht die Acht-Bit-Werte der Befehle), überträgt der Befehl

JMP [B,X]

die Steuerung auf die entsprechende Subroutine. Der BSR-Aufruf nimmt dabei die Adresse dieses Sprungbefehls an. Wir müssen nun einen zweiten Initialisierungsschritt anlegen.

Daten:

Sprungtabelle – eine Tabelle mit acht 16-Bit-Adressen

CMDB, CMDU etc. die Startadressen der Subroutinen

Ablauf:

```
FOR jede Subroutine
  Startadresse holen
  Startadresse in der Sprungtabelle
  speichern
ENDFOR
```

Wir müssen nun überlegen, was bei Eingabe des Befehls Q (Programm beenden) geschehen soll. Das Programm und der Debugger sollten so zurückgelassen werden, daß sie für weitere Aufrufe ohne irgendwelche Schwierigkeiten zur Verfügung stehen.

Dafür muß der Stack am Programmende aber im gleichen Zustand sein wie am Anfang. Wir könnten nun für unser Programm einen separaten Stack anlegen, indem wir den Wert von S ändern und am Ende einfach den alten Wert wieder einsetzen. Diese Technik ist sehr praktisch – in unserem Fall könnten wir jedoch Schwierigkeiten haben, ungenutzten Speicherplatz zu finden, da der RAM-Bereich bereits mit zwei Programmen belegt ist. Es wäre auch möglich, S einfach entsprechend zu inkrementieren und so alle überflüssigen Werte zu löschen. Doch auch dieser Vorgang bietet Probleme, da wir nicht wissen, ob Interrupts aufgetreten sind und sich dadurch die Stackinhalte verändert haben. Am einfachsten wäre es, den Anfangswert von S zu speichern und mit dem letzten Befehl des Programms wieder einzusetzen.

Ein Zwischenspeicher

Aus diesem Grund speichert der in der Initialisierung eingerichtete Interruptmechanismus drei Byte an der Adresse, die der SWI-Vektor bei \$FFFA angibt. Auch hier müssen wir den ursprünglichen Wert wieder einsetzen, da ein Interrupt des Betriebssystems unvorhergesehene Abläufe auslösen kann. Wir brauchen also einen weiteren Initialisierungsteil, der alle Werte speichert, die von der Endroutine wieder eingesetzt werden sollen.

Daten:

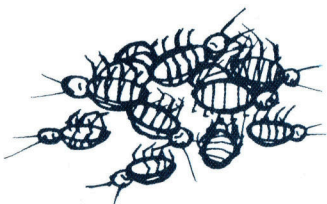
Gesichert – fünf Byte mit den ursprünglichen Werten

Stack-Pointer – der aktuelle Wert von S, plus 2

SWI-Vektor – der Wert von \$FFFA

Ablauf:

Stack-Pointer in Gesichert speichern





SWI-Vektor holen

Die drei Byte beim SWI-Vektor in Geschert speichern

Die Endroutine (der Befehl Q) muß diesen Vorgang nur umkehren und die Steuerung an das Betriebssystem zurückgeben. Für die Ausführung dieser Vorgänge gibt es wieder eine Reihe von Möglichkeiten: Nach der Wiederherstellung des Vektors kann entweder der

SWI-Befehl verwandt oder aber ein Sprung auf einen bekannten Punkt des Betriebssystems ausgeführt werden.

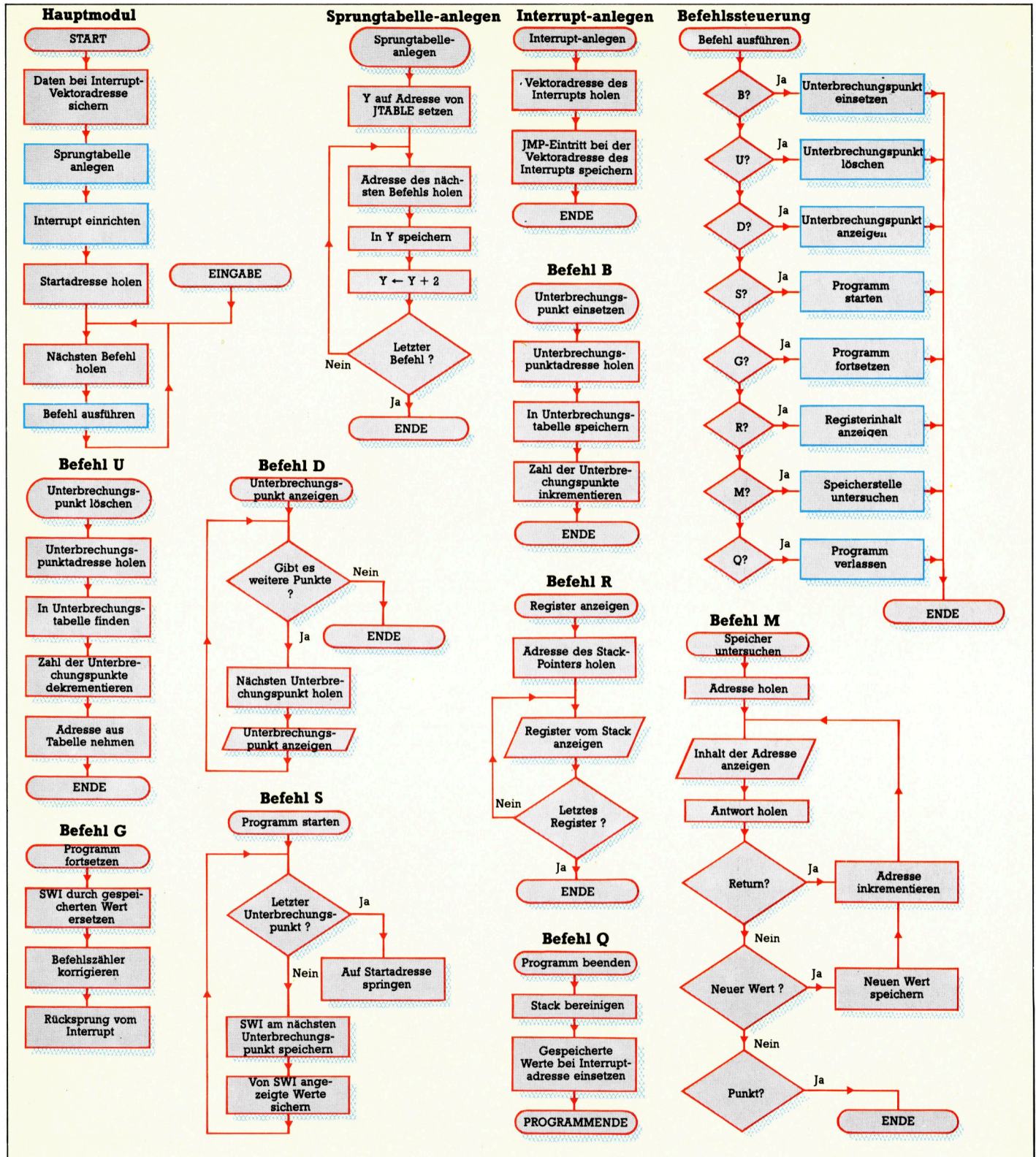
Daten:

Gesichert – fünf Byte mit den ursprünglichen Werten

Stack-Pointer – der aktuelle Wert von S, plus 2

SWI-Vektor – der Wert bei \$FFFA

Die Ablaufdiagramme zeigen den Programmfluß der Debuggermodule. Sie sind in der Reihenfolge aufgeführt, in der sie von anderen Routinen aufgerufen werden. Blau umrahmte Kästen zeigen an, daß separate Routinen aufgerufen werden.



Reset-Vektor – der Wert bei \$FFFE

Ablauf:

- Die drei Byte des SWI-Vektors mit Gesichert wiederherstellen
- Stack-Pointer wiederherstellen
- Zum Betriebssystem verzweigen

Wir wenden uns nun dem Hauptmodul zu.

Daten:

- Prompt** für die Befehlseingabe ist das ASCII-Zeichen '>'
- Befehls-Offset** zeigt in die Befehlstabelle und die Sprungtabelle

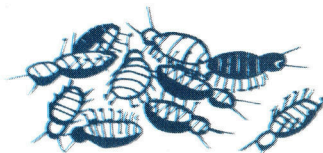
Ablauf:

- Werte-sichern
- Sprungtabelle-anlegen
- Interrupt-einrichten
- Start-Adresse holen
- REPEAT
 - Prompt darstellen
 - Befehl-holen
 - Befehl-ausführen
 - Kein Ende

abkürzen. So müssen beispielsweise viele Werte bewegt werden, damit sie für die Subroutinen in den korrekten Registern stehen. Eine Umstellung der Register könnte die Abläufe vereinfachen, wäre aber nur zu empfehlen, wenn der Speicherplatz sehr begrenzt ist. Weiterhin haben wir die gleichen Daten an mehreren Stellen definiert. Es gibt jedoch zwei Methoden, Daten in einem Programm einzusetzen: Die Daten können entweder innerhalb der eingesetzten Module bleiben – das ist theoretisch der beste Weg – oder vollständig am Anfang des Programms definiert werden. Dies ist besonders dann vorteilhaft, wenn Sie einen Disassembler mit dem Programm einsetzen wollen.

Der Debugger sollte in einen Speicherbereich geladen werden, der nicht von dem zu analysierenden Programm belegt ist. Da das Programm per Sprung auf die DEBUG-Einsprungsadresse aufgerufen wird, muß diese Adresse vor dem Programmstart bekannt sein.

Durch diese Serie sollten Sie umfassendes Verständnis über Assemblerprogrammierung und Entwicklungsmethoden bekommen haben.



Damit ist unser Debuggerprogramm fertig. Sie können den Code jedoch noch optimieren und

Sprungtabelle-anlegen

JTABLE	RMB	16	Platz für acht Zwei-Byte-Adressen
SETUPJ	LEAX	JTABLE,PCR	Basisadresse der Tabelle in Y
	LEAX	CMDB,PCR	Startadresse der CMDB-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDU,PCR	Startadresse der CMDU-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDD,PCR	Startadresse der CMDD-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDS,PCR	Startadresse der CMDS-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDG,PCR	Startadresse der CMDG-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDR,PCR	Startadresse der CMDR-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDM,PCR	Startadresse der CMDM-Subroutine
	STX	,Y++	In der Tabelle speichern
	LEAX	CMDQ,PCR	Startadresse der CMDQ-Subroutine
	STX	,Y++	In der Tabelle speichern

Hier findet der eigentliche Sprung zu den Subroutinen statt. Wir gehen davon aus, daß X die Adresse von JTABLE enthält und B das Offset.

DOCMD JMP [B,X]

Werte-Sichern

SAVED	RMB	5	Die fünf zu sichernden Byte
SAVEIT	LEAX	SAVED,PCR	Speicheradresse
	TFR	S,D	S nach D verlegen

ADDD	#2	Zwei addieren (für Rücksprung-adresse)
STD	,X++	Sichern
LDY	\$FFFA	Interrupt-Vektoradresse holen
LDA	,Y+	Erstes zu sichernde Byte holen
STA	,X+	Sichern
LDD	,Y	Die anderen beiden Byte holen
STD	,X	Sichern
RTS		

Befehl Q

CMDQ	LEAX	SAVED,PCR	Adresse von Gesichert
	LDY	\$FFFA	SWI-Vektor
	LDA	2,X	Erstes von drei Byte
	STA	,Y+	Wiederherstellen
	LDD	3,X	Die anderen beiden Byte
	STD	,Y	Wiederherstellen
	LDS	,X	Stack-Pointer sichern
	JMP	[\$FFFE]	Indirekter Sprung über den Reset-Vektor

Hauptmodul

PROMPT	FCB	'>	
STACKP	RMB	2	Stack-Pointer für Register-anzeigen
DEBUG	BSR	SAVEIT	Werte-sichern
	BSR	SETUPJ	Sprungtabelle-anlegen
	BSR	INIT	Interrupt-einrichten und Start-adresse-holen
ENTRY	STS	STACKP,PCR	Stack-Pointer sichern
REPT02	LEAX	JTABLE,PCR	
	LDA	PROMPT,PCR	Prompt holen und anzeigen
	BSR	OUTCH	
	BSR	GETCOM	Befehl holen
	LSLB		Doppelter Offset für 16-Bit-Tabelle
	BSR	DOCMD	Befehl ausführen
	BRA	REPT02	Nächster Befehl



Spezialanwendungen

Für Behinderte bedeutet der Microcomputer mehr als eine Lernverbesserung: Er hilft, die Barrieren zur „normalen“ Welt zu überwinden.

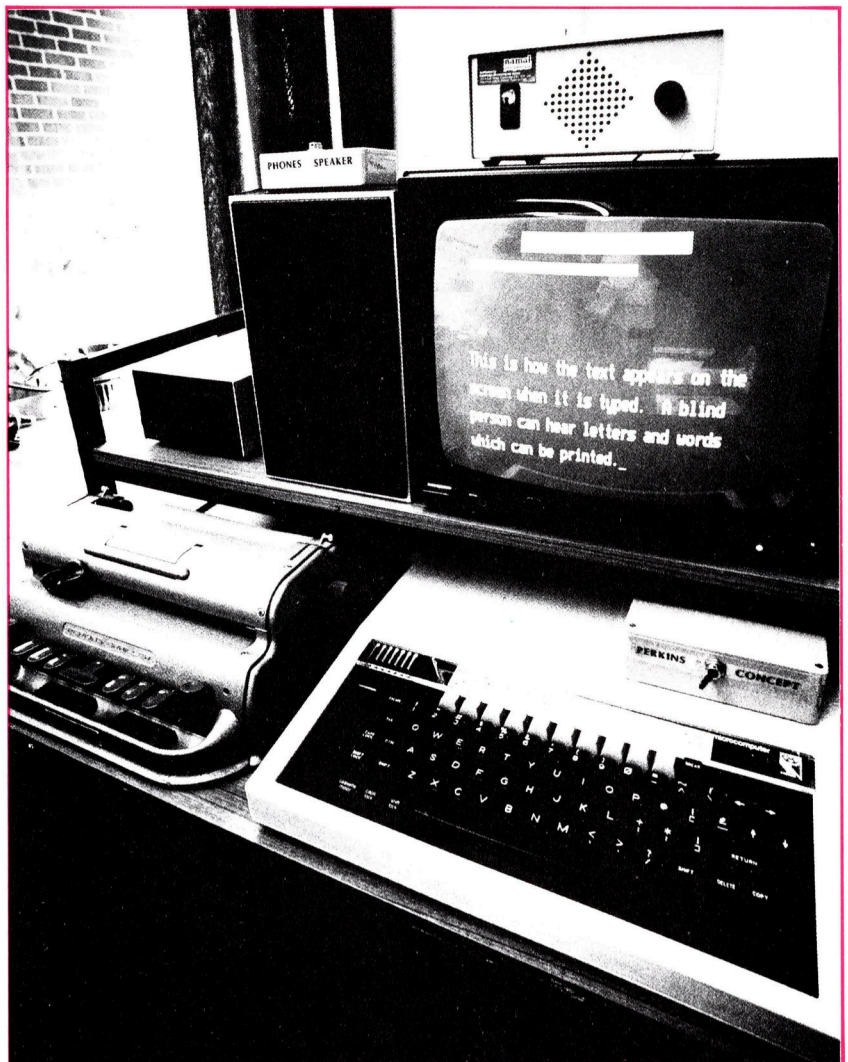
Viele Erwachsene und Kinder brauchen spezielle Ausbildungstechniken. Ursache dafür mag eine körperliche Behinderung wie Taubheit oder Blindheit sein oder eine geistige Behinderung. Einige daraus resultierende Bedürfnisse sind offensichtlich, andere hingegen schwer zu erkennen und zu verstehen. Jede Krankheit ist mit speziellen Problemen verbunden, die häufig mehr durch die Verhaltensweise der Gesellschaft als durch die Behinderung selbst geschaffen werden.

Leider wenden die meisten Regierungen weniger Mittel für behindertengerechte Sonderschulen auf als für normale Schulen. Folglich ist dieser Markt für Hard- und Software-Entwickler nicht lukrativ. Die Energie und Hingabe derjenigen, die dennoch in und für diesen Bereich arbeiten, sollten als beispielhaft für den gesamtpädagogischen Bereich dienen. So sammelten Schüler in South Yorkshire umgerechnet 40 000 Mark in ihrer Freizeit, um die Entwicklung eines Spezialtelefons für Hörbehinderte zu ermöglichen. Diese Arbeit wird üblicherweise „an der Basis“ geleistet, von Einzelpersonen und Initiativ-Gruppen. Lehrer, Programmierer und Ingenieure arbeiten in Eigeninitiative und entwickeln Hard- und Software für bestimmte Ausbildungszwecke.

Lehrerausbildung

In Manchester wurde ein Special Education Microelectronics Resources Centre (SE-MERCs) gegründet, ebenso in Newcastle und Redbridge, um den speziellen Bedürfnissen Behinderter in der Ausbildung zu entsprechen. Eine der Hauptaufgaben besteht darin, Lehrer an Sonderschulen auszubilden und sie über wichtige technische Entwicklungen und Neuerungen zu informieren.

Die Situation wird durch die vielgestaltigen Probleme der speziellen Erziehung noch komplizierter. Bei einigen Krankheiten ist die Kommunikation das größte Problem. So kann ein intelligenter, aufgeweckter Mensch durch eine körperliche Behinderung unfähig sein, mit der Außenwelt zu kommunizieren. Die Microtechnologie hat vielen Menschen neue Kommunikationskanäle eröffnet. So können etwa mit Hilfe der „Photonic Wand“ (einem optischen Sensor, der sich auf einem Plastikhelm befindet und durch Kopfbewegung gesteuert wird) stumme Menschen, die ihre Gliedmaßen nicht bewegen können, einen Acorn B bedienen. Der optische Sensor ist einem Lichtgriffel ähn-



lich und bewegt einen Cursor über den Bildschirm – entsprechend den Kopfbewegungen.

Dieses Hilfsmittel kann auch von Menschen benutzt werden, die ihre Handbewegungen nicht steuern können. Im SEMERC in Manchester hat man das Photonic Wand mit einem Sprachsynthesizer verbunden. Für Telefongespräche entwickelte Antworten wie „Können Sie den Satz wiederholen“ werden auf dem Bildschirm dargestellt und von einem Sprachsynthesizer ausgesprochen. Man hofft, den Wortschatz erweitern zu können und eine einfache Methode zu entwickeln, um den Synthesizer mit dem Telefon zu verbinden.

Computer sind für Sehende entwickelt worden. Monitore, Drucker und Plotter sind visuelle Ausgabeeinheiten, und alle Programme bedienen sich der visuellen Darstellung. Bei

Am „Vincent“-Arbeitsplatz können Menschen mit verschiedenen Behinderungen arbeiten. Mit dem „Perkins Brailier“ können Blinde Informationen in den Computer eingeben. Der Sprachsynthesizer ermöglicht eine hörbare Rückkopplung. Die Textdarstellung auf dem Monitor wurde für Sehbehinderte vergrößert. Der Arbeitsplatz ist außerdem mit einer „Konzept-Tastatur“ verbunden. An die im Bild gezeigte Station kann auch ein spezieller Blindenschrift-Drucker angeschlossen werden.



den Projekten „Open University's Computing“ und „Blind Project“ arbeitet man daran, bestehende preiswerte Software auch Blinden zugänglich zu machen. Das Projekt läuft in Zusammenarbeit mit Blinden an Schulen, am Arbeitsplatz und daheim. Die Arbeitsplätze bestehen aus Computer, Diskettenstation, Monitor, Drucker, Sprach-Synthesizer, einer Spezialtastatur und einem abgewandelten „Perkins Brailleur“ und wurden in Schulen und Arbeitsstätten installiert.

Der Perkins Brailleur wurde vor 40 Jahren zur Erzeugung von Blindenschrift erfunden. Dieses Gerät wird mit dem Computer verbunden und ermöglicht so einem Blinden, den Ausdruck zu lesen. Es gibt spezielle Software, mittels der ein Computer Blindenschrift in normale Schrift umwandeln, speichern, editieren und ausdrucken kann.

Für die Verwendung mit der üblichen Tastatur wurden mehrere sprechende Textverarbeitungsprogramme entwickelt. Eingegebene Zeichen und spezielle Tasten werden durch

den. Das ist einfacher als „FD 1 RETURN“ usw. einzugeben.

Mit der bei Apples „Macintosh“ verwendeten Maus entfällt die „Tastatur-Barriere“ ebenfalls. Eine Benutzer-Gruppe namens AMASE (Apple Macintosh Applications in Special Education) beschäftigt sich ausschließlich mit der Erforschung der Möglichkeiten der Maschine in diesem Bereich.

Man hat auch spezielle Schalter entwickelt, die durch verschiedene Körperteile bedient werden können. So gibt es zum Beispiel für spastische Patienten eine Spezialschaltung. Zwei kleine Metallscheiben werden in Augennähe angebracht und können die horizontale Bewegung der Augen registrieren. Die dabei entstehenden elektrischen Signale werden in Steuersignale umgesetzt.

Im Bereich „Science and Society“ der Bradford-Universität befestigte man auf einer Turtle ein blinkendes, orangefarbenes Licht, wodurch ein seh- und körperlich behinderter 18jähriger Patient instande war, sie zu sehen.

Die „Photonic“-Ausrüstung erlaubt selbst stark behinderten Menschen, die weder sprechen noch ihre Gliedmaßen bewegen können, einen Acorn B zu steuern. Die Kopf-Bewegung wird in ein Analogsignal umgewandelt, das der Computer verarbeitet.



gesprochene Ausgaben bestätigt. Die „Delete“-Taste „spricht“ zugleich die Bezeichnung des gelöschten Zeichens. Text wird mit Hilfe eines Sprech-Cursors editiert. Während er über den Text läuft, werden Wörter oder Sätze gesprochen. Der Cursor kann jederzeit angehalten werden, um Ergänzungen oder Lösungen vorzunehmen.

Manche behinderten Kinder sind nicht in der Lage, eine normale Tastatur zu bedienen. Für solche Menschen wurde ein flaches, berührungsempfindliches Tablett entwickelt. Diese „Konzept-Tastatur“ wird vom Lehrer mit austauschbaren Folien versehen. So kann die Tastatur beispielsweise in vier Segmente geteilt werden, die jeweils ein Symbol tragen. Mit ihnen läßt sich eine auf dem Boden befindliche Turtle steuern. Durch Drücken unterschiedlicher Segmente kann die Turtle vorwärts, rückwärts, nach links oder rechts geführt wer-



Durch Verwendung eines Konzept-Keyboards konnte er die Turtle steuern und die Ergebnisse seiner Aktionen sehen. Für diesen jungen Mann, wie für viele behinderte Kinder, stellen solche Erfahrungen oft den ersten Schritt von der Rolle des passiven Beobachters zur bewußten Anteilnahme und Aktion sowie zur Beeinflussung ihrer Umgebung dar.

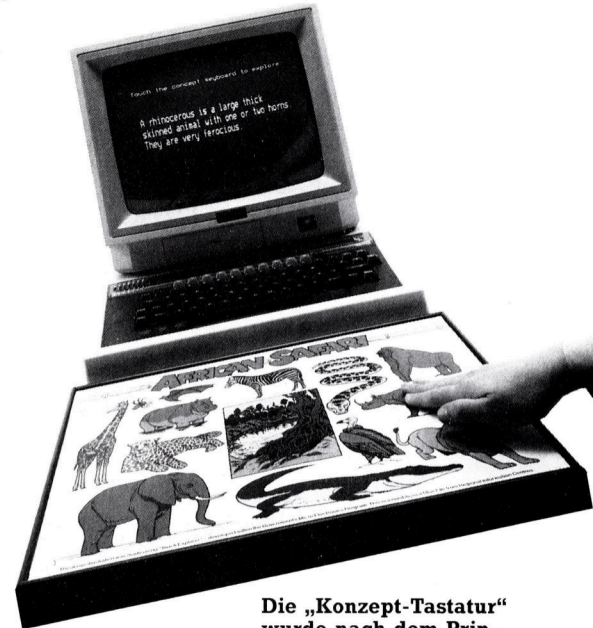
Auf der British Logo User's Group-Konferenz im Jahre 1984 führte Dr. Sylvia Weir bemerkenswerte Beispiele für die Fortschritte an, die geistig und körperlich behinderte Kinder durch den Umgang mit LOGO machten. Der



17jährige Michael, ein Sonderschüler, hatte noch nie ein Wort geschrieben. Seine Lehrer vermuteten, daß er sehr intelligent sei, aber nicht kommunizieren könne. Nachdem er mit dem Computer vertraut gemacht worden war, programmierte er zehn Stunden täglich. Zwei Jahre später schrieb er an der Universität an einer Arbeit zum Thema „Gefangene Intelligenz“. Ein 7½-jähriger sprachbehinderter Junge durfte eine Turtle via Knopfbox steuern, einem der „Konzept-Tastatur“ ähnlichen Gerät. Er wurde durch dieses Erlebnis so aufgeregt, daß er zum ersten Mal sprach.

Dr. Weir wandte LOGO auch an einer Sonderschule an, wo eine Gruppe von Kindern, die

auch die Spezialpädagogik der technischen Entwicklung weit hinterher. Das resultiert daraus, daß zuwenig Mittel zur Forschung zur Verfügung stehen und daß Hard- wie Software-Hersteller ihren Umsatz mit Geschäfts- und Industrieprogrammen erzielen – nicht mit Ausbildungsprogrammen. Der spezialpädagogische Bereich ist noch kleiner. Die SEMERC-Zentren können die Bitten um Hilfe längst nicht alle erfüllen. Der Mangel an finanziellen Mitteln und Erfahrung verlangsamt die Entwick-



Die „Konzept-Tastatur“ wurde nach dem Prinzip des Grafiktablets entwickelt. Selbst spastische Anwender können mit dem Computer kommunizieren. Eine Beherrschung der Tastatur ist nicht erforderlich. Die Gestaltung der Folien hilft, die Hemmschwelle gegenüber dem Computer herabzusetzen.

unter zerebraler Paralyse litten, unterrichtet wurde. Die Steuerung des Computers half ihnen, die durch ihre Behinderung entstandene Passivität zu überwinden. Dr. Weir schrieb: „Die daraus resultierenden Verbesserungen im Sinne der Wertigkeit einer Person und die Konsequenzen für intellektuelle Aktivität haben die Schule veranlaßt, ein eigenes Computertzentrum zu installieren.“

Als Ergebnis gemeinsamer Arbeit des SEMERC in Manchester und des örtlichen Computerclubs entstand „Micromike“, ein Gerät, das sprachgestörten Kindern hilft. Ein modifiziertes Funk-Mikrofon wurde mit einem Acorn B verbunden und erlaubt Kindern, mittels Stimme verschiedenste Aktivitäten auf dem Bildschirm zu steuern. Zu der dafür entwickelten Software gehört „City“, ein Programm, mit dem sie die Skyline einer Stadt zeichnen können. Höhe und Breite der Gebäude werden durch Lautstärke und Dauer der Stimme bestimmt. Programme dieser Art bieten den Kindern die Möglichkeit, ihre Stimme auf individuelle Art zu beherrschen.

Wie die allgemeine Pädagogik, so hinkt





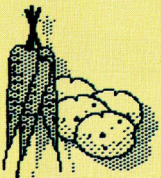
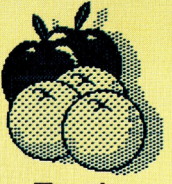
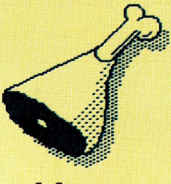
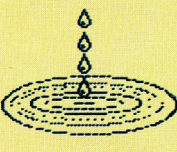

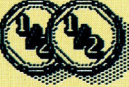
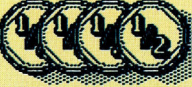

lung von Hard- und Software an diesen Zentren. Und aus dem Geldmangel an Schulen resultiert eine nur geringe Einbeziehung der Computertechnik zu Ausbildungszwecken.

Computer und die Begabten

Begabte und talentierte Kinder haben ebenfalls spezielle pädagogische Bedürfnisse. Der Einsatz von Computern bei der Ausbildung bietet ihnen ebenso viele Möglichkeiten, wie es bei behinderten Kindern der Fall ist.

Behinderte wie begabte Kinder gewinnen mit dem Computer Kontrolle über ihre Umgebung und können so natürliche oder von Menschen geschaffene Barrieren überwinden, um individuell aufzuwachsen und sich selbst zu verwirklichen. Im einzelnen bieten Computer begabten Kindern folgende Möglichkeiten:

- Entwicklung von Fähigkeiten und Interessen;
- kreative neue Wege, Problemlösungen zu finden;
- Kommunikation und Zusammenarbeit mit anderen begabten Schülern;
- die Möglichkeit, ermüdende und langweilige Übungen auf interessante Art durchzuführen;
- ein neues Medium für Experimente.

	(1)	(2)	(3)	(4)	
U\$()					UNITS
P\$()	 Vegetables	 Fruit	 Meat	 Water	DESCRIPTIONS
PC()					COSTS
PN()	0.5	1	1	0.5	MINIMUM PROVISIONS PER MAN PER WEEK
PA()	300	200	100	200	AMOUNT OF PROVISIONS ORDERED

Der Spieler kann aus vier Vorratsarten auswählen: Gemüse, Obst, Fleisch und Wasser. Die Beschreibungen dieser vier Arten werden im Array P\$() und die zugehörigen Einheiten in U\$() gespeichert. Um zu entscheiden, welche Mengen gekauft werden müssen, braucht der Spieler zwei Informationen: die Kosten jeder Vorratsart und den wöchentlichen Minimalbedarf jedes Mitglieds der Mannschaft. Diese Daten werden in den Arrays PC() und PN() gespeichert. Die vier genannten Arrays werden am Programmstart initialisiert.

Vorratsbeschaffung

Wir setzen unsere Simulationsserie mit der Entwicklung des zweiten Moduls für das Handelsspiel fort. Diese Unterroutine befaßt sich mit dem Einkauf von Vorräten für die Fahrt über den Atlantik.

Im ersten Abschnitt des Spieles heuert der Spieler die Mannschaft an. Anschließend erscheint die Meldung "PRESS ANY KEY TO CONTINUE". Durch Drücken einer Taste erscheint der zweite Abschnitt, in dem Vorräte eingekauft werden. Der Spieler weiß, daß die Fahrt acht Wochen dauern wird und daß er für die Besatzung ausreichend Nahrung und Wasser einkaufen muß.

Das Programm berechnet die notwendigen Gesamtbestände für acht Wochen und überprüft, ob die gekauften Mengen ausreichen. Ist dies nicht der Fall, wird der Spieler darauf hingewiesen, daß eventuell jemand hungern oder dursten muß, und man erhält die Gelegenheit nachzukaufen. Ist nicht genug Gold zur Bezahlung da, wird gefragt, ob man es noch einmal mit einer kleineren Bestellung versuchen will (das Kapital wird ständig angezeigt).

Wie in der Anheuerungsphase müssen einige Arrays initialisiert werden. Die Vorräte

werden in unterschiedlichen Einheiten angeboten – Gemüse, Früchte und Fleisch werden in Kilo und Wasser in Barrels gemessen. In Zeile 20 wird für diese Einheiten ein Array U\$() mit vier Elementen initialisiert.

```

19 REM **** PROVISIONING ARRAYS ****
20 DIM U$(4):U$(1)="KILO":U$(2)="KILO":U$(3)="KILO":
  U$(4)="BARREL"
21 DIM P$(4):P$(1)="VEG":P$(2)="FRUIT":P$(3)="MEAT":
  P$(4)="WATER"
22 DIM PC(4)
23 DIM PN(4):PC(1)=.5:PC(2)=1:PC(3)=2:PC(4)=.5
24 DIM PA(4):PN(1)=2:PN(2)=1:PN(3)=1:PN(4)=.5

```

Die Beschreibungen der vier Vorratsarten werden in Zeile 21 durch die vier Elemente des Arrays P\$() repräsentiert. Das Programm installiert ein Array PA() zum Speichern der gekauften Mengen jeder Vorratsart. Ein Kilo Gemüse kostet ein halbes Goldstück, ein Kilo Obst ein Goldstück, ein Kilo Fleisch zwei und ein Barrel Wasser ein halbes Goldstück. Diese Informationen werden im Array in Zeile 23 ab-

gelegt. Dieses Array kann ebenfalls einfach erweitert werden.

Um die Gesundheit der Mannschaft zu erhalten, benötigt jedes Besatzungsmitglied eine bestimmte Menge an Vorräten. Die entsprechenden Angaben befinden sich in Zeile 24 im Array PN(). Ein Mann braucht zum Beispiel pro Woche zwei Kilo Gemüse, so daß das erste Element (PN1) auf 2 gesetzt ist.

In Zeile 500 des ersten Moduls wird die Unteroutine zum Anheuern der Mannschaft (Zeile 1000) aufgerufen. Danach erfolgt der Rücksprung zu Zeile 500. Zum Aufruf der Vorratsroutine muß deshalb nach Zeile 500 ein GOSUB eingefügt werden:

```
550 GOSUB 2000: REM EINKAUF VON
    PROVANT
```

Es folgen einige Erklärungen zum Einkauf der Vorräte. Die Variable CN, die während der Anheuerungsphase gesetzt wurde, enthält die Anzahl der Männer. Sie wird in Zeile 2040 verwendet, um den Spieler an die Anzahl der zu versorgenden Personen zu erinnern.

Der Programmteil zum Einkauf besteht aus einer Schleife, die viermal ausgeführt wird – einmal für jede Vorratsart. Der Wert von T wird in Zeile 2050 jeweils um 1 erhöht. Auf dem Bildschirm erscheint die wöchentliche Menge der benötigten Vorratsart pro Mann.

Die jeweiligen Maßeinheiten für Nahrung und Wasser (Kilo und Barrel) werden durch das entsprechende Element des Arrays U\$() ermittelt. Wird mehr als ein Kilo benötigt, wird in Zeile 2075 ein „s“ an Kilo angefügt. Ansonsten wird in Zeile 2080 das s durch ein Leerzeichen ersetzt. Zeile 2085 schreibt das Wort "OF" und die in P\$() gespeicherte Vorratsart, gefolgt von "FOR EACH WEEK OF THE JOURNEY".

In den Zeilen 2100 und 2110 wird der Spieler gefragt, wieviele Einheiten jeder Vorratsart er kaufen will. Die Eingaben werden in Zeile 2130 in P\$() gespeichert. Die Zahl wird im ersten Element des Mengen-Arrays in Zeile 2140 abgelegt. Ist die Menge 0, gibt das Programm in Zeile 2160 die Meldung "IF YOU DON'T BUY ANY" gefolgt vom Vorrats-Array P\$(T) aus. Danach arbeitet das Programm unter Verwendung der Formel CN*8*PN(T) aus; ob die bestellte und in PA() eingetragene Menge größer ist, als man für eine acht Wochen dauernde Fahrt benötigt. CN ist die Anzahl der Männer und PN(T) die benötigte Menge der Vorratsart, ermittelt durch T. Reichen die Vorräte nicht aus, wird eine Warnung ausgegeben.

Nachdem der Spieler gewarnt wurde, fragt das Programm (bei Zeile 2230) "DO YOU WANT TO TRY AGAIN (Y/N)?" Hat der Spieler nur noch wenig Kapital und antwortet trotzdem mit Y, verzweigt das Programm zu Zeile 2400. Wollen Sie andere Bestellungen machen, wird das entsprechende Element im Array PA() auf 0 gesetzt. In diesem Fall muß das Programm zur gleichen Vorratsart zurückverzweigen, indem in Zeile 2250 mit T=T-1 der Schleife

fenzähler um 1 reduziert wird.

Ist der Spieler mit der Bestellung zufrieden, wird die Menge des verbleibenden Goldes und eine Liste der bisher bestellten Vorräte ausgegeben. Es kann sein, daß der Spieler ausreichende Vorräte bestellt, jedoch nicht mehr genug Kapital zur Bezahlung hat (Zeile 2260). Ist dies der Fall, wird durch die Zeilen 2270 bis 2290 angezeigt, daß das Gold nicht mehr reicht, und der Spieler gefragt, ob er es noch einmal mit einer kleineren Bestellung versuchen will.

In einer anderen Schleife ab Zeile 2410 wird unter Verwendung der Variablen U\$(), PA() und P\$() eine Liste der gekauften Vorräte dargestellt. Die Schleife endet in Zeile 2490. Der Programmlauf wird fortgesetzt, wenn eine Taste gedrückt wurde. An diesem Punkt erfolgt der Rücksprung zum Hauptprogramm.

Zweites Modul

```
2000 REM **** PROVISIONING ****
2005 PRINTCHR$(147):REM CLEAR SCREEN
2010 S$=" STAGE 2 - PROVISIONING*"
2015 GOSUB9100
2020 S$=" -----*
2025 GOSUB9100
2030 GOSUB9200:PRINT
2040 PRINT"YOU'VE HIRED A CREW OF ";CN;" "
2045 GOSUB9200:GOSUB9200
2050 FORT=1:TO4
2055 PRINT
2060 PRINT"EACH CREW MEMBER WILL NEED "
2070 PRINT"AT LEAST ";PN(T);" ";U$(T);
2075 IFPN(T)=1THENPRINT" ";GOTO2085
2080 PRINT"S ";
2085 PRINT" OF ";P$(T)
2090 PRINT"AT";PC(T);"GOLD PIECES PER ";U$(T)
2095 GOSUB9200:PRINT:GOSUB9200
2100 PRINT"FOR EACH WEEK OF THE JOURNEY."
2095 GOSUB9200:PRINT:GOSUB9200
2110 PRINT"HOW MANY ";U$(T);"S OF ";P$(T)
2110 S$="DO YOU WANT TO BUY*":GOSUB9100
2120 PRINT
2130 INPUTP$
2140 PA(T)=VAL(P$):GOSUB9200
2150 IFPA(T)>((CN*8*PN(T))-1) THEN2260
2160 IFPA(T)=0THENPRINT"IF YOU DON'T BUY ANY":GOTO
2180
2170 PRINT"IF YOU ONLY BUY ";PA(T);U$(T);
2175 IFPA(T)=1THENPRINT" OF":GOTO2180
2176 PRINT"S OF "
2180 PRINTP$(T);", ";GOSUB9200
2190 PRINT"SOMEONE MIGHT GET ";
2200 S$="HUNGRY"
2210 IFT=4THENS$="THIRSTY"
2220 PRINTS$;": ";GOSUB9200
2230 S$="DO YOU WANT TO TRY AGAIN (Y/N)":GOSUB9100
2240 INPUTP$:P$=LEFT$(P$,1)
2242 IF P$<"Y" AND P$<"N"THEN 2230
2245 IF P$="N"THEN2400
2250 PA(T)=0:T=T-1:GOTO2410
2260 IFPA(T)*PC(T)>MOTHEN2270
2265 GOTO2400
2270 S$="YOU DON'T HAVE ENOUGH MONEY FOR":GOSUB9100
2280 PRINTPA(T)
2290 PRINTU$(T);"S OF ";P$(T):GOSUB9200
2300 S$="PLEASE TRY AGAIN*":GOSUB9100:PA(T)=0:T=T-1:GOTO2410
2400 MO=MO-(PA(T)*PC(T))
2410 PRINT:S$="PROVISIONS SO FAR*":GOSUB9100
2412 GOSUB9200
2415 FORT=1:TO4
2420 PRINTPA(T);U$(T);
2430 IFPA(T)=1THENPRINT" OF ";GOTO2440
2435 PRINT"S OF ";
2440 PRINTP$(T)
2450 GOSUB9200
2460 NEXT
2480 PRINT"MONEY LEFT = ";MO;" GOLD PIECES"
2485 GOSUB9200:GOSUB9200
2490 NEXTT
2500 GOSUB9200:PRINT:S$="END OF PROVISIONING*":GOSUB9100:GOSUB9200
2510 PRINT: S$=K$:GOSUB9100:PRINT: GOSUB9200
2520 GETP$:IFP$=""THEN2520
2599 RETURN
```

Diese Unteroutine befaßt sich mit dem Einkauf von Vorräten für die Fahrt und sollte zum ersten Modul hinzugefügt werden. Das Programm wird in einer Schleife viermal ausgeführt, so daß der Spieler bei jedem Durchgang jeweils eine Vorratsart bestellen kann.

BASIC-Dialekte

Spectrum
Führen Sie die folgenden Änderungen aus:
2005 CLS
2240 INPUT P\$:LET
P\$=P\$ (1 TO 1)

Acorn B
Ändern Sie folgende Zeile:
2005 CLS



Human-Defekt

Bei „Deus Ex Machina“ übernimmt der Spieler die Hauptrolle in einer „voll animierten Fernseh-Phantasie“, die Elemente bestens bekannter Arcade-Spiele mit einer Audio-Spur vereint.

Computerspiele haben sich zu einem bedeutenden Faktor in der Unterhaltungsindustrie entwickelt. So war es unausweichlich, daß Software-Häuser auf andere Unterhaltungsbereiche übergriffen. Automata Software tat den ersten Schritt in diese Richtung. Man entwickelte ein Produkt, das nicht nur Computer-Software beinhaltet, sondern auch eine Audio-Cassette, die mit dem Computer-Programm synchronisiert werden kann und so einen „Soundtrack“ ins Spiel bringt.

Basisidee von „Deus Ex Machina“, für dessen Entwicklung sechs Monate und für dessen Programmierung drei Monate erforderlich waren: Ein allmächtiger Computer der Zukunft rebelliert und hilft bei der Schaffung eines menschlichen „Defekts“. Der Spieler – der Defekt – durchwandert die verschiedenen Stadien des Spiels, in denen Erfahrungen dargestellt werden, die von der Kindheit bis zum Erwachsenenalter reichen. Der Spieler ist bereits bei der Zeugung einbezogen, da er das Sperma zum Ei steuern muß. Während das Kind wächst, wird es ständig von der „Defekt-Polizei“ angegriffen. Der Spieler hat diese Angriffe abzuwehren, indem er entweder die Tastatur oder den Joystick benutzt. Die Wertung geschieht dergestalt, daß man den „idealen Daseins-Prozentsatz“ zu erhalten versucht, der bei 99% beginnt und mit jedem Angriff der Polizei geringer wird. Mit dem Älterwerden des Defekts verändert sich die Art der Angriffe. Der Spieler muß versuchen, ihnen richtig zu begegnen.

Ist das Spiel geladen, sollte die Audiospur mit dem Programm synchronisiert werden. Da-

bei ist sorgfältig vorzugehen, da die unterschiedlichen Bildschirmdarstellungen präzise auf Text und Musik ausgerichtet sind und so nachhaltig den Unterhaltungswert des Programms beeinflussen.

Das Programm kommt zweigeteilt: Jeweils ein Segment befindet sich auf jeder Cassettenseite – Gesamtkapazität 96 KByte. Am Ende des ersten Programnteils (Seite 1) muß die zweite Seite geladen werden, nachdem der Spieler in einer recht amourösen Szene versuchen mußte, die auf ihn zuströmenden Küsse „einzufangen“. Beim Seitenwechsel darf der Computer nicht ausgeschaltet werden. Wieder ist auf präzise Synchronisation zu achten. Die Aufgabe des Spielers besteht in diesem Teil im wesentlichen darin, Hindernissen auszuweichen, bevor er „alt“ wird.

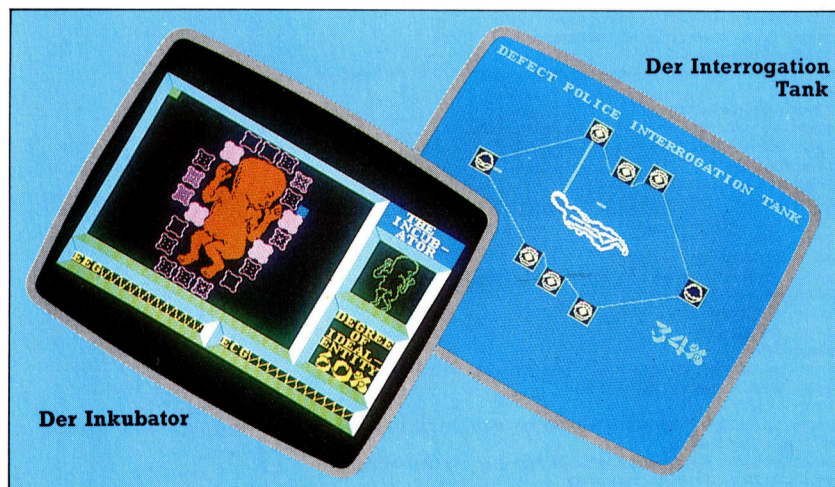
Spielen oder nicht

„Deus Ex Machina“ ist dahingehend ungewöhnlich, als es keine Gewinnpunkte gibt. Und der Spieler muß auch überhaupt nicht am Spiel teilnehmen. Auch ohne Eingriff finden Ereignisse statt. Die Grafiken sind durchweg ausgezeichnet und eindrucksvoll. Wenngleich keine der Bildschirmdarstellungen atemberaubend ist, so spiegeln sie doch die Sorgfalt wider, die man jedem Detail des Programms widmete.

Die Musik des Soundtracks wurde von Automatas Mitbegründer Mel Croucher geschrieben und gespielt. Von ihm stammt auch die Geschichte.

Story wie Soundtrack unterscheiden sich sehr von den meisten anderen Computerspielen. Spiel-Freaks, die Spaß an der Zerstörung sich schnell bewegender Barrieren haben, werden eher enttäuscht sein – und viele Interessenten werden den halbmythischen Inhalt der Texte irritierend finden. Und doch ist das Programm ein bemerkenswertes Experiment. Es wird zweifellos als entscheidender Schritt für die Gestaltung computerisierter Unterhaltung zu bewerten sein.

„Deus Ex Machina“ enthält viele Grafiken, die allerdings in Teilen untereinander ähnlich sind. Die für die Erhaltung des „idealen Daseins“ erforderlichen Taktiken ändern sich ständig. Statt einer Punktzahl wird ein Prozentwert in der unteren rechten Bildschirm-Ecke gezeigt. Er nimmt im Spielverlauf langsam ab.



Deus Ex Machina: Für ZX Spectrum (48 K)
Hersteller: Automata Ltd, 27 Highland Road, Portsmouth, Hants, PO 4 9DA
Autoren: Mel Croucher, Andrew Staggs
Joysticks: Wahlweise
Format: Cassette

Fachwörter von A bis Z

Interrupt = Unterbrechung

Es gibt Software- und Hardware-Interrupts. Software-Unterbrechungen werden innerhalb eines Programms erzeugt, und ihre Behandlung übernimmt das Betriebssystem. Hardwaremäßig erfolgt die Unterbrechung durch einen Impuls am Interrupt-Eingang des Prozessors, was den Programmabbruch und meist ein Reset bewirkt, das das System in den Einschaltzustand versetzt.

Interrupts werden für die verschiedensten Zwecke gebraucht. Wenn ein Programm für eine bestimmte Aufgabe die Hilfe des Betriebssystems benötigt, schickt es eine „freiwillige“ Interrupt-Anforderung an die Systemsteuerung, die daraufhin entsprechend reagiert.

Etwas ähnliches geschieht bei der Auslösung eines „unfreiwilligen“ Interrupts, wenn etwa das ROM-gespeicherte BASIC in einem Programm Fehler entdeckt. Die Unterbrechung führt dann zu einem Programmabsturz und zur Ausgabe der zugehörigen Fehlermeldung. „Timer“-Interrupts treten in festen Zeitintervallen auf, wenn die Systemuhr weitergestellt wird. Sie können als Zeitmarken zur Überwachung von Vorgängen verwendet werden.

Job Control Language = Kommandosprache

Sie umfaßt die Anweisungen, mit denen das Betriebssystem für die selbständige Bearbeitung von Benutzeranträgen programmiert wird. Bei Microcomputern ist so etwas noch selten, obwohl die Stapelverarbeitung unter CP/M schon einen Anfang darstellt. Dagegen bedienen sich mittlere Rechner und Großanlagen verbreitet der JCL und arbeiten

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.

damit kontinuierlich die verschiedenen Benutzerprogramme ab, die als Job-Kette auf Band oder Platte gespeichert werden. Meist benötigt ein Job mehrere Systemprogramme nacheinander, so etwa zum Drucken, Lesen, Sortieren usw. Statt diese Routinen einzeln durch den Operator zu aktivieren, wird ein JCL-Programm zusammengestellt, das Laden und Starten der Systemroutinen übernimmt.

Jump = Sprung

Ein Sprungbefehl bewirkt, daß der Rechner aus der laufenden Anweisungsfolge aussteigt und das Programm an anderer Stelle fortsetzt. In BASIC lautet der Sprungbefehl „GOTO“. Er hat im Unterschied zur GOSUB-Anweisung nicht den Nebeneffekt, daß für den späteren Rücksprung der aktuelle Befehlszählerstand auf den Stack geschoben wird. Die Rückkehr zum Ausgangspunkt kann daher nur über ein neuerliches GOTO erfolgen.

Ein „bedingter“ Sprung wird mit IF...THEN (Wenn...dann) eingeleitet und nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt ist: IF Bedingung erfüllt THEN Sprung.

Junction = Übergang

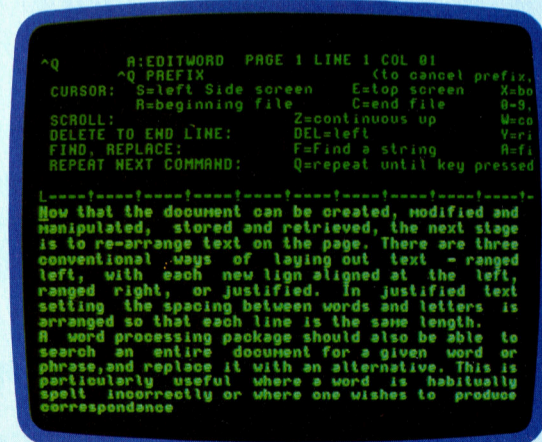
Die Grenzzone zwischen Halbleitern mit unterschiedlichen elektrischen Eigenschaften (auch zwischen Metall und Halbleiter) wird als Übergang bezeichnet, sie ist entscheidend für das Verhalten von Bauteilen. Bei einem p-n-Übergang beispielsweise

stößt ein p-dotierter (d. h. positive Ladungsträger freisetzender) Halbleiter an einen n-dotierten Bereich (mit negativen Ladungsträgern), wobei sich durch Ladungsträgeraustausch und Rekombination eine elektrische „Sperrschicht“ und gleichzeitig eine Spannung zwischen p- und n-Material ausbildet. Durch Anlegen einer äußeren Spannung kann die Sperrschicht mehr oder weniger abgebaut und damit der Stromfluß durch den Übergang gesteuert werden. Darauf beruht die Gleichrichterwirkung von Dioden und im Prinzip auch der Verstärkungsmechanismus bei Transistoren.

Justification = Justierung

Das Justieren von Texten beim Blocksatz beherrscht fast jedes Textverarbeitungsprogramm: Die Buchstaben werden innerhalb der Zeile so ausgerichtet, daß sie am linken und am rechten Rand bündig stehen.

Bei der Maschinencode-Programmierung versteht man unter Justification die Verschiebung von Registerinhalten mit dem Ziel, die erste oder letzte von Null verschiedene Stelle in die höchst- bzw. niedrigstwertige Bitposition zu bringen. Das ist für Abfragezwecke, insbesondere bei der „Normalisierung“ im Gleitkommaformat von Bedeutung.



Das Justieren der Zeilenränder ist eigentlich bei jedem Textverarbeitungsprogramm vorgesehen. Wenn Sie das Bildschirmfoto genau betrachten, sehen Sie, daß der bündige Zeilenabschluß durch Variation der Leerraumlängen zwischen den Wörtern erreicht wird.

Bildnachweise

1569, 1570: Science Photo Library LTD
1570, 1571, 1582, 1583, 1587: Kevin Jones
1572, 1573: Uta Brandl
1574, 1575: Chris Stevens
1576, 1577, 1586: Caroline Clayton
1583, 1584, 1596, U3: Ian McKinnell
1589: Liz Dixon
1591: Tony Sleep
1592: Chris Barker
1593: European Marketing Division

computer kurs

Heft **58**



Auf großer Fahrt

Nachdem die Mannschaft angeheuert wurde, soll nun die Ausrüstung des Schiffes komplettiert werden und Handelswaren für die Expedition an Bord kommen.



Szenenaufbau

Strukturiertes Programmieren ist wichtig für die Gestaltung eines Abenteuer-Programms. Wir erarbeiten schrittweise ein entsprechendes Programm.



Reise in andere Welten

Türen zu sonst nicht erreichbaren Welten werden mit der Entwicklung von Simulation und Abenteuer-Programmen geöffnet.



Muskeln für den Robot

Der Robot-Arm nimmt Form an. Schablonen für das Zuschneiden der einzelnen Teile.



Blick fürs Detail

Zuverlässige Lesegeräte entstanden aus den ersten optischen Geräten zur Zeichenerkennung.

