Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Strategiespiel

Adressverwaltung

Pluspunkt: der Acorn B+

Wissensformen

Heft 55

computer kurs Heft 55

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

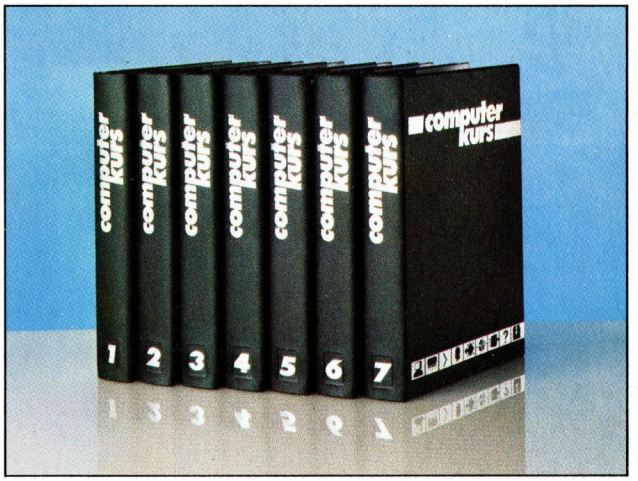
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

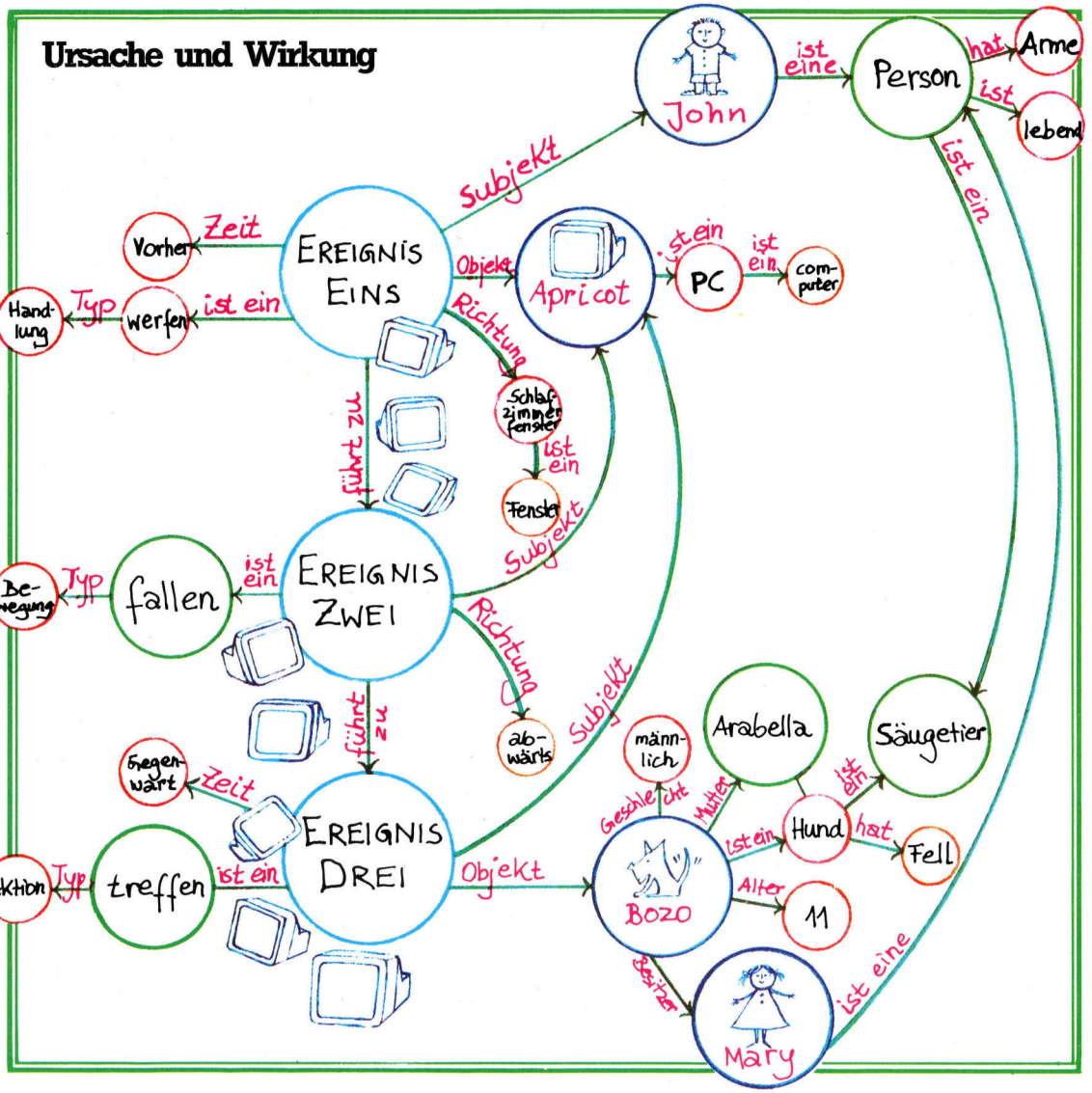
Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

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



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

Wissensformen



Semantische Netze werden benutzt, um Aktionen und Objekt miteinander zu verbinden und so ihre Beziehung darzustellen. Das hier gezeigte Netz beinhaltet: „John warf den PC aus dem Fenster. Er traf Marys Hund." Die Kreise sind die Objektknoten, die Pfeile die Verwandtschaftsbögen. Aus den semantischen Verbindungen könnte man beispielsweise ableiten: „Ein Computer traf ein pelziges Säugetier" oder „Etwas fiel auf Arabellas Kind". In einem echten System wäre das Netzwerk weit ausführlicher.

Ein kritischer Punkt der Künstlichen Intelligenz ist die Informationsdarstellung. Nur wenn die reale Welt richtig umgesetzt wird, verfügt man über ein arbeitsfähiges System. Wir beschäftigen uns mit den verschiedenen zur Verfügung stehenden Strukturen.

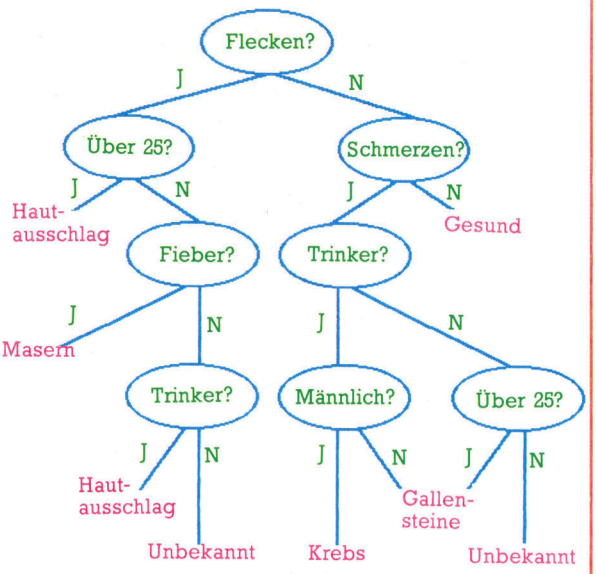
Wohl jeder Programmierer hat eine Vorstellung dessen, was mit Daten gemeint ist. Viele aber sind skeptisch, was die Bezeichnung „Wissen" anbelangt, wenn es um im Computer enthaltene Informationen geht. Der Unterschied zwischen Daten und Wissen kommt letztlich durch die Probleme zum Ausdruck, die sich den KI-Programmierern stellen, sowie den zu findenden Lösungen. Wichtig ist deshalb die Betrachtung der vier Hauptkomponenten, durch die sich Wissen von Daten unterscheidet.

Zunächst muß die Darstellung von Wissen flexibel sein. Wissen muß in Form von schrumpfenden oder wachsenden Strukturen umgesetzt werden (etwa durch Verwendung dynamischer Speicheradressierung). Statische Strukturen, deren Größe und Umfang sich während des Programmablaufs nicht ändert, sind für diese Zwecke unbrauchbar. Dynamische Daten-Strukturen sind mit den meisten BASIC-Dialekten leider nicht erstellbar.

Für die Wissensdarstellung sind zudem geschichtete Strukturen oder Strukturen mit mehreren Ebenen statt mit nur einer Ebene erforderlich. So sollten Listen zum Beispiel sogenannte Sublisten enthalten, Bäume mit „Sub"-

Lösungswege

Entscheidungsbäume sind leicht zu erstellen und anzuwenden. Sie sind allerdings von der absoluten Genauigkeit der Information abhängig. Sind Daten unsicher (was im „richtigen" Leben häufig der Fall ist), kann eine einzige falsche Antwort in eine völlig falsche Richtung führen.



Bäumen verknüpft sein, Regeln auf Subregeln verweisen etc.

Drittens ist Wissensdarstellung vom Wesen her heterogen: Sie enthält eine Mischung verschiedener Daten. Nehmen wir als Beispiel die Beschreibung einer Figur in einem Abenteuer-Programm. Dazu wären mehrere Strings erforderlich (für den Namen), ferner Zahlen (Alter und Größe), Verweise auf andere Daten (Freunde und Gegner des Akteurs), Gesetze, die sein Verhalten in typischen Situationen beschreiben, und vieles andere mehr. Auch das ist in BASIC durchaus machbar, allerdings nicht ganz unproblematisch. Das liegt daran, daß ein BASIC-Array entweder Strings oder Zahlen enthält, nicht aber beides.

Schließlich, und das ist das Entscheidende, ist Wissensdarstellung aktiv, wogegen Datenstrukturen passiv sind. Auf Wissen basierende Systeme haben sich letztlich gewandelt: von der Zweischichtigkeit (Programm und Daten) zur Dreischichtigkeit von Fakten, Gesetzen und einem Inferenz-Mechanismus.

Fakten entsprechen eindeutig den Daten. Bei der Inferenz-Maschine handelt es sich dagegen um ein Programm. Regeln sind eine Mischung aus beidem und legen aktive Daten fest. Nachdem wir nun grob dargelegt haben, wo die Unterschiede zwischen Daten und Wissen liegen, befassen wir uns mit den verschiedenen Möglichkeiten der Darstellung und Speicherung von Wissen.

Arrays bieten eine einfache Möglichkeit der Wissensdarstellung. Das Problem dabei ist, daß sie sich für viele KI-Anwendungen als zu einfach erweisen. Stellen wir uns etwa innerhalb des Expertensystems ein vereinfachtes medizinisches Diagnoseprogramm vor, das lediglich vier Diagnosen (Hypothesen) und sieben Symptome (Beweise) enthält. Man könnte das Systemwissen wie dargestellt als zweidimensionales Array codieren.

Dieses Format entspräche einer einfachen Inferenz-Strategie: Jeweils eine Beweisführung würde auf einer Skala erfolgen, die von −1(Nein) über 0 (Vielleicht) bis +1(Ja) reicht. Würden alle Beweise ausgeführt, könnten sie mit den Zahlen in der entsprechenden Reihe jeder Kolumne multipliziert werden. Die Spalte mit der höchsten Gesamtsumme wäre dann die am besten begründete Hypothese.

Probleme ergäben sich, wenn man versuchte, eine solche Darstellung zu erweitern. Sie ist unflexibel: Bei 200 Beweisen und 100 Krankheits-Kategorien enthielte ein Array 20 000 Zeilen, von denen die meisten „leer" wären. Überdies wäre es zu „flach", so daß die hierarchische Struktur medizinischen Wissens ignoriert würde. Denn Krankheiten können nach Typen kategorisiert werden. Weiß ein Arzt, daß ein Patient eine bestimmte Krankheit hat, werden bestimmte Fragen überflüssig. Geeigneter für diesen Zweck ist deshalb die Baumstruktur.

Zwei wichtige Applikationen für Baumstrukturen sind: (1) Entscheidungs-Bäume und (2) und Vererbungs-Hierarchien. Als anschauliches Beispiel für die erste Baumstruktur haben wir die tabellarischen Daten des 1. Beispiels in einen Entscheidungsbaum (siehe Diagramm links) umgewandelt.

Auf falscher Fährte

Hauptproblem bei Entscheidungsbäumen, die für die Wissensdarstellung eingesetzt werden, ist, daß sie Unsicherheiten sehr schlecht darstellen. Sie sind effizient, wenn die Ergebnisse jedes Knotens klar sind. Bleiben aber Antworten offen oder unsicher, kann das System leicht auf den falschen Weg geraten. Im wirklichen Leben ist die definitive Beantwortung von Fragen mit Ja oder Nein sehr selten.

Vererbungs-Hierarchie-Bäume (siehe Beispiel rechts) verbinden Begriffe und Konzepte derart, daß die Baumstruktur die Verwandtschaften methodisch wiedergibt. Diese Baumart erlaubt allgemeinverständliche Schlußfolgerungen.

Eine Netzwerk- oder Kurven-Struktur ist komplexer als ein Baum. Ein Baum hat einen speziellen „Wurzelknoten", und die Zweige weisen alle in eine Richtung. Beim Netz dagegen können die Verbindungen in jede Richtung weisen.

Semantische Netzwerke sind eine besondere Art von Kurven-Strukturen, die in der Künstlichen Intelligenz bevorzugt verwendet werden, um Wissen darzustellen. In einem semantischen Netz repräsentieren Knoten Gegenstände, und die Bögen oder Bindeglieder dazwischen verdeutlichen Verwandtschaften.

Das semantische Netz ist eine Generalisierung einer LISP-Konstruktion, die wir in der letzten Folge behandelten, nämlich eine „Property-

Liste". Sowohl Property-Listen als auch semantische Netze können unter dem Begriff „Tuples" zusammengefaßt werden. Bei einem „n-Tuple" handelt es sich um die Gruppierung von „n" Objekten. Darstellungen von Objekt-Attribut-Wert bestehen aus Dreiergruppen, zum Beispiel:

Objekt	Attribut	Wert
BOZO	IST-EIN	HUND

Von der Idee der Wissensdarstellung in Dreifachform zum Rahmenbegriff ist es nur ein kurzer Weg. Der Rahmen ist eine Struktur, die in der KI häufig Verwendung findet. Er enthält eine Reihe von „Schlitzen", die den üblichen Dateiabschnitten entsprechen.

Eine Rahmendarstellung, die die zuvor behandelten semantischen Netze einschließt, könnte so aussehen:

Name:	Ereignis 3
Typ:	Trifft
Zeit:	Jetzt
Subjekt:	PC 49
Objekt:	Bozo
Vorangegangen:	Ereignis 2

Name:	Bozo
Ist ein:	Prototyp Hund
Besitzer:	Mary
Geschlecht:	Männlich
Alter:	11
Eltern:	(Ferdinand, Arabella)
Einbezogen in:	(Ereignis 3, Ereignis 7, Ereignis 20 . . .)

Name:	Prototyp Hund
Hat:	Fell
Kann:	(Bellen, Beißen, Jagen . . .)
Beine:	Grenzwert = 4
Beispiel:	(Bozo, Fido, Arabella, Spot)
Gefährlich:	IF klein AND schlafend OR schwanzwedelnd THEN nein ELSE ja.
Fazit:	Wenn schlafend THEN liegenlassen

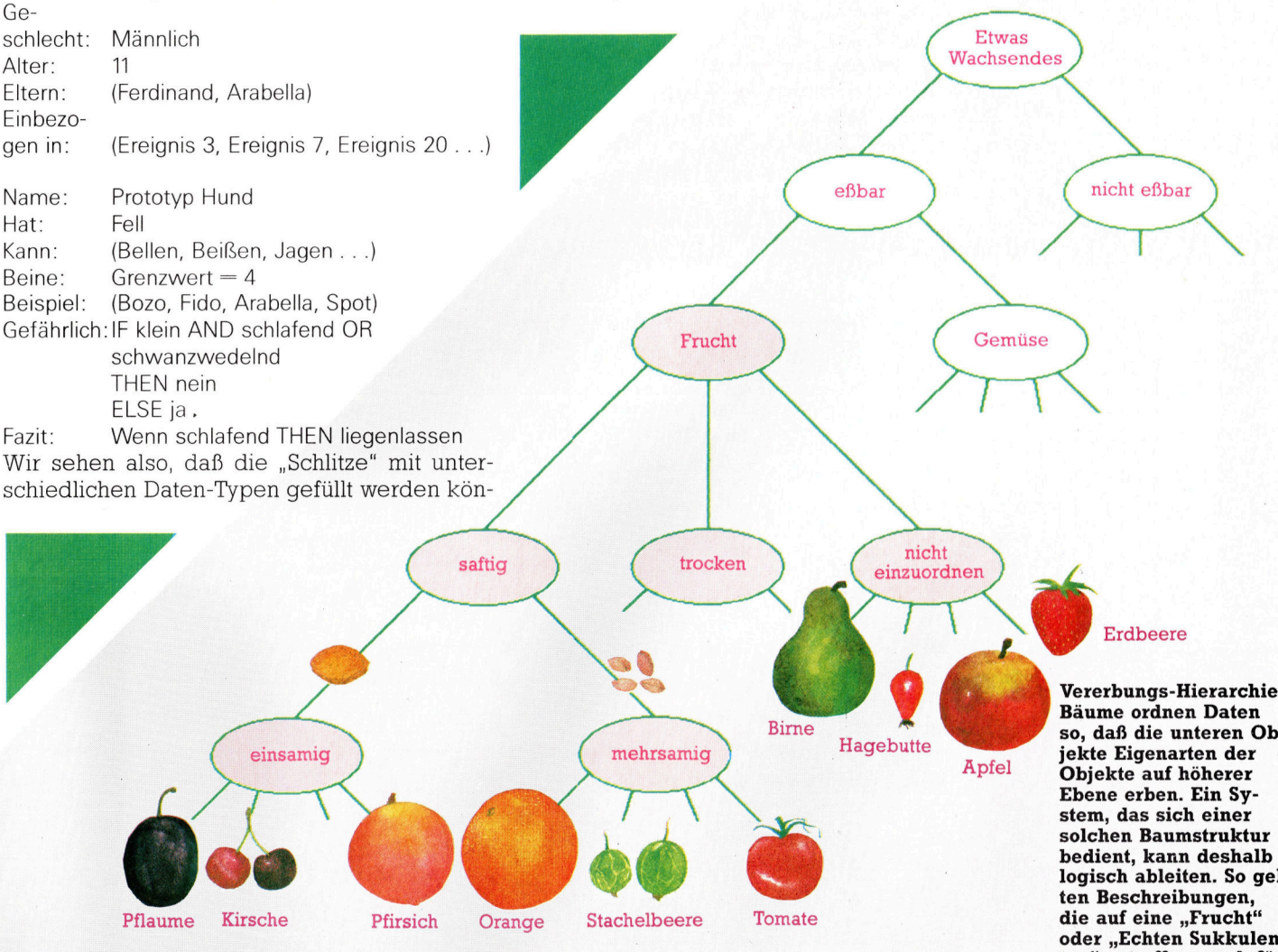
Wir sehen also, daß die „Schlitze" mit unterschiedlichen Daten-Typen gefüllt werden können: „Alter" ist eine Zahl zugeordnet, „Eltern" entspricht einer Liste, „Gefährlich" beinhaltet eine Entscheidungsregel, „Ist ein" enthält einen Verweis auf einen anderen Rahmen und dergleichen.

Da jede flexible Struktur die Information anderer Strukturen darstellen kann, scheint es, als sei die Entscheidung für eine Darstellung vorrangig eine Frage der Effizienz. Einige KI-Forscher aber lehnen diesen Relativismus ab und versuchen einen umfassenden logischen Formalismus zu finden. Logik als Darstellungssprache ist die Basis für PROLOG. Zum Bespiel lautet die PROLOG-Version der vorgenannten Rahmendarstellung wie folgt:

```
gefährlich (X):—
                Hund (x), nicht (sicher(X))
sicher (X):— klein (X), schlafend (X)
sicher (X):— schwanzwedelnd (x)
```

Rein theoretisch ist die Logik tatsächlich fundamentaler als die anderen diskutierten Darstellungsformen. In der Praxis aber sind für die Wahl der Darstellung andere Überlegungen entscheidend als theoretische „Eleganz". Die richtige Darstellung dem Wissen entsprechend anzuwenden ist gerade die „Kunst" bei der Künstlichen Intelligenz.

Früchte sortieren



Vererbungs-Hierarchie-Bäume ordnen Daten so, daß die unteren Objekte Eigenarten der Objekte auf höherer Ebene erben. Ein System, das sich einer solchen Baumstruktur bedient, kann deshalb logisch ableiten. So gelten Beschreibungen, die auf eine „Frucht" oder „Echten Sukkulenten" zutreffen, auch für Tomaten und Pflaumen.

Handelsplätze

Strategiespiele, die reale Situationen simulieren, sind unterhaltsam und fordern logisches Denken. Im folgenden beginnen wir mit einem neuen Programmprojekt: ein Handelssimulationsspiel, das auf einer Schiffahrt nach Amerika im 15. Jahrhundert basiert.

Computer eignen sich bekanntermaßen in hervorragender Weise für Reaktionsspiele aller Art. Darüber hinaus erfreut sich eine Spielvariante immer größerer Beliebtheit: Sogenannte Simulationsspiele ermöglichen es dem Anwender, reale Geschehnisse via Rechner nachzuvollziehen. Dies geschieht zum teil sogar in Echtzeit-Simulationen. Am bekanntesten sind Flugsimulations-Programme oder auch anspruchsvolle Strategiespiele, bei denen der Benutzer das Geschehen bestimmt.

In einem Strategiespiel muß der Spieler bzw. die Gruppe von Spielern Aufgaben lösen und erhält Hinweise, wenn Entscheidungen getroffen werden müssen. Ein typischer Vertreter dieser Sparte ist ein Handelsspiel, in dem den Spielern bestimmte Geldmittel zur Verfügung stehen, mit denen sie mit Waren handeln können, um Gewinne zu erzielen. Im Gegensatz zu Abenteuerspielen, in denen der Spieler mit plötzlich auftretenden Gefahren fertig werden muß, kann man bei Simulationsspielen oftmals vorausplanen und so vorhersehbare (oder auch unvorhersehbare) Probleme meistern.

Dieser Spieltyp erfordert also in hohem Maß organisatorische Fähigkeiten statt Reaktionsgeschwindigkeit. Da die Realität widergespiegelt wird, ist nicht alles vorhersehbar. – In einer guten Simulation treten immer Zufallselemente auf, um die Bank-Schemas des Spielers durcheinanderzubringen. Dieser muß daher seine Spielweise auch unvorhersehbaren Ereignissen anpassen können.

Simulationsspiele haben den Vorteil, daß mehrere Personen gemeinsam als Team spielen können und dabei ihre Ideen und Gedanken vereinen, um das Spielziel zu erreichen.

Kontroverse Entscheidungen

Zusätzlich zu all diesen Punkten berühren Simulationsspiele ein Gebiet, das von den meisten Computeranwendungen nur selten angesprochen wird: Es können moralische Fragen auftauchen, wenn der Spieler größeren Profit gegen die Zufriedenheit der Angestellten abwägen muß. Müssen solche Entscheidungen im Team getroffen werden, löst dies nicht selten hitzige Diskussionen aus.

Als Thema für eine Simulation bietet sich fast alles an. Ein gemeinsamer Faktor ist jedoch immer, eine bestimmte Aufgabe zu erfüllen oder ein Ziel zu erreichen. So ist ein mögliches Ziel, für eine bestimmte Zeit im Geschäft zu bleiben oder alternativ eine Million Mark zu verdienen. Gerade finanzielle Strategien sind die Basis vieler Spiele, wogegen einige auch allgemeinere Ziele haben. Ein Simulationsspiel, das gegenwärtig in amerikanischen Hochschulen verwendet wird, behandelt die Suche und Bergung des Wracks der „Mary Rose", einer elisabethanischen Galleone. In diesem Spiel werden dem Spieler auch Dokumente zur Verfügung gestellt, um das Programm zu ergänzen und den Entscheidungsprozeß zu unterstützen.

In der folgenden Artikelserie entwickeln wir ein Handelssimulationsspiel, das den Spieler in das 15. Jahrhundert zurückversetzt, wo er ein Schiff ausstatten und in die „Neue Welt" segeln muß. Das Programm ist in eine Anzahl Module aufgeteilt. Außerdem können die Scenarios einfach geändert werden. Dadurch kann die Simulation jederzeit aktualisiert werden, so daß man als Hintergrundgeschichte zum Beispiel einen Handel zwischen mehreren Planeten entwerfen könnte.

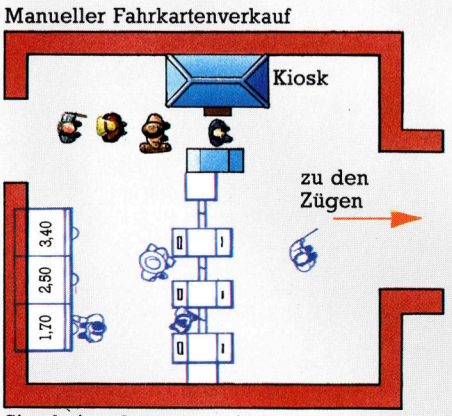
Hier sehen Sie Beispiele der drei Haupttypen von Simulationen. „Space Shuttle" ist eine realistische Aktionssimulation eines Raumschiffs. „Air Traffic Control" ist eine Echtzeitsimulation, in der verschiedenen Flugzeugen Anweisungen zum sicheren Landen und Starten gegeben werden müssen. The „Great Nordic War" ist eine Kriegsspielsimulation des zweiten Nordischen Krieges, in der ein Spieler die Rolle Karls XII. von Schweden innehat.

Im Untergrund

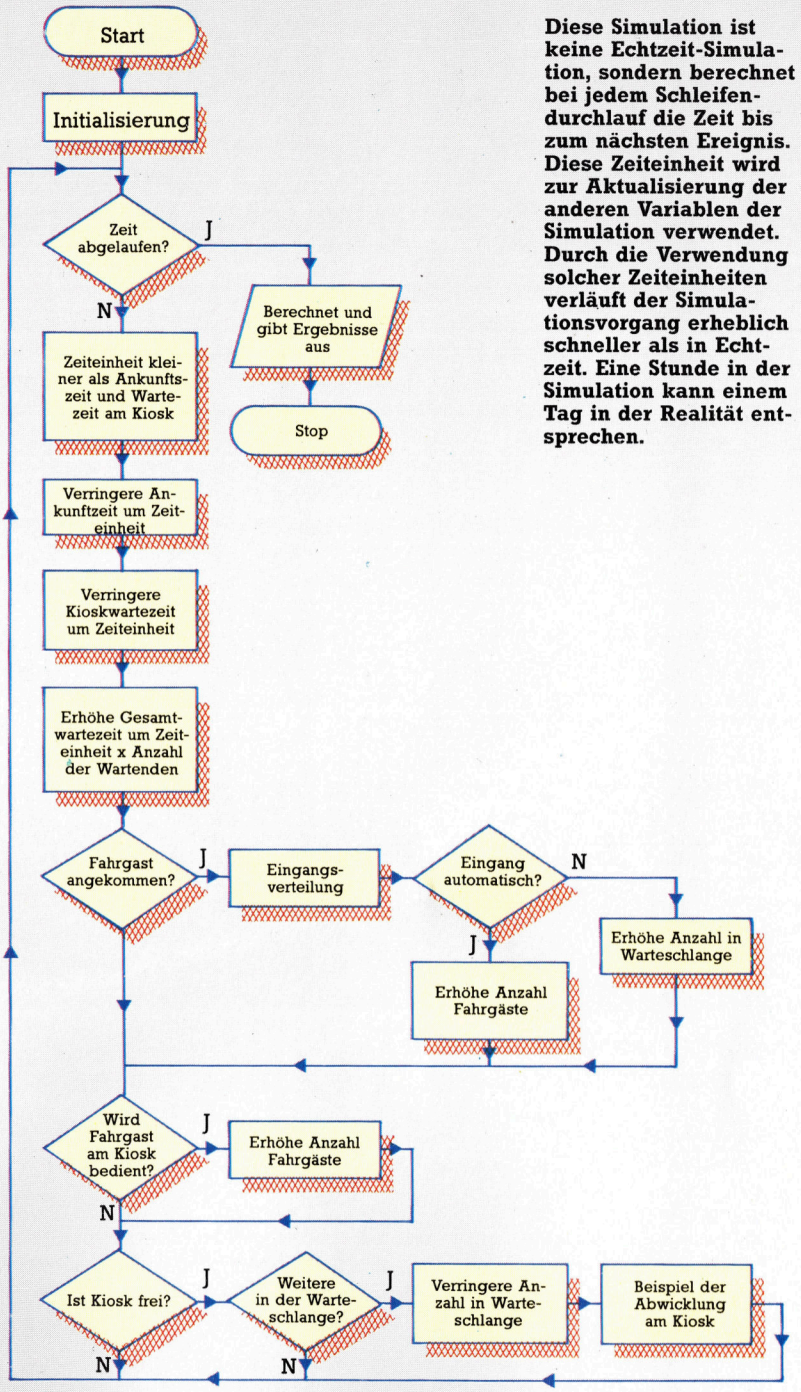
Computersimulationen zum Spielen und Lernen sind heute weit verbreitet, wurden jedoch ursprünglich als Management-Werkzeuge entwikkelt. Solche Simulationen sind bei der Planung von Verkehrs- oder ähnlichen Systemen sehr nützlich. Im allgemeinen basieren sie auf statistischen und mathematischen Prinzipien, mit denen Vorhersagen getroffen werden können.

Wir können die Grundlagen einer Simulation durch einen Blick auf ein einfaches Servicesystem demonstrieren. Das bestehende System gestattet den Eintritt in eine U-Bahn-Station nach dem Kauf einer Fahrkarte am Kiosk. Nun wird vorgeschlagen, eine bestimmte Anzahl automatische Entwerter zu installieren, die mit Automatentickets arbeiten. Das nebenstehende Flußdiagramm illustriert die notwendigen Vorgänge zur Entwicklung einer entsprechenden Simulation. Drei Orte werden verwendet: ein Haupteingang, ein Kiosk für Fahrkarten und eine (manuelle oder automatische) Fahrkartenentwertung.

Die Verteilungssysteme der ersten beiden Orte werden maßgeblich von statistischen und mathematischen Berechnungen abgeleitet. Das dritte System jedoch variiert bei jedem Programmlauf, um die verschiedenen Verhältnisse von manueller und automatisierter Abfertigung zu ermitteln. Mit dieser einfachen Simulation läßt sich die durchschnittliche Wartezeit für einen Fahrgast auf einfache Weise berechnen.



Simulation des automatischen Fahrkartenverkaufs



Diese Simulation ist keine Echtzeit-Simulation, sondern berechnet bei jedem Schleifendurchlauf die Zeit bis zum nächsten Ereignis. Diese Zeiteinheit wird zur Aktualisierung der anderen Variablen der Simulation verwendet. Durch die Verwendung solcher Zeiteinheiten verläuft der Simulationsvorgang erheblich schneller als in Echtzeit. Eine Stunde in der Simulation kann einem Tag in der Realität entsprechen.

Das Ziel des Spieles (für einen oder mehrere Spieler) ist, durch Handel das Kapital zu vermehren, dieses Ziel jedoch im richtigen Verhältnis zur Sicherheit und dem Befinden der Crew zu verfolgen. Das Spiel wird in drei Hauptteile eingeteilt: die Vorbereitung, die Fahrt und den Handel.

Planung und Zufallselemente

Der erste Teil dient zum Planen der Fahrt und zum Erwerb von Vorräten. Im zweiten Teil werden Zufallselemente in das Spiel eingebracht. Die Fähigkeit, mit diesen Problemen fertig zu werden, hängt wesentlich von Ihrer Planung im ersten Abschnitt ab. Im dritten Teil schließlich kommt es auf gute Beziehungen zu den Eingeborenen und entsprechenden Handel an.

Vom Standpunkt eines Programmierers aus besteht die Konstruktion des Spieles aus einer Serie unabhängiger Module, wobei für Acorn B, Spectrum und Commodore 64 entsprechende Hinweise gegeben werden. Die Hauptaufgabe des Programms ist, die Entscheidungen des Spielers zu kontrollieren, Variationen zu generieren sowie den Spielstatus während des Ablaufs zu analysieren und zu verfolgen. Im nächsten Artikel werden wir uns mit dem ersten Programmteil für das Spiel befassen.

An der Leine

Diesmal wird die Verbindungsleitung zwischen Rechner und Roboter hergestellt. Außerdem finden Sie in diesem Abschnitt die Spectrum-Versionen einiger Steuerprogramme.

Was zum „Robotern" mit dem Spectrum noch fehlt, ist das passende 12polige Verbindungskabel von der Schnittstellenplatine zum D-Anschluß. Neben den durch das Interface gepufferten acht Ein/Ausgabeleitungen vom Datenbus stellt die neue Schnittstelle auch die 5-V- und 12-V-Stromversorgung des Spectrums bereit, so daß wir auf eine zusätzliche externe Spannungsversorgung verzichten können. Das Netzteil des Spectrums kann die zusätzliche Belastung verkraften.

Von dem im Bauteilverzeichnis des letzten Artikels dieser Serie beschriebenen Flachkabel werden vier Drähte abgetrennt. Die verbleibenden zwölf Leitungen isoliert man auf beiden Seiten ab, verzinnt die blanken Litzen und verlötet sie genau der Zeichnung entsprechend mit der Platine und dem D-Stecker. Achten Sie bei der Verdrahtung besonders darauf, die 5-V- und 12-V-Leitung nicht miteinander oder gar mit einer Datenleitung zu verwechseln.

Mit der Verbindungsleitung ist das neue Interface komplett und kann probeweise am Spectrum angeschlossen werden. Stecken Sie das Interface bei ausgeschaltetem Computer mit der Bauteilseite der Platine nach oben an den Erweiterungsanschluß des Spectrums. Wenn der Blinkkontakt auf Position 5 korrekt eingebaut ist, kann dabei eigentlich nichts schiefgehen. Ist der Stecker sehr schwergängig, sollten Sie die Platine mit leichtem seitlichen Rütteln einsetzen. Als nächstes den D-Stecker am Roboter anschließen und den Rechner einschalten. Stimmt alles, erscheinen die üblichen Copyright-Zeilen auf dem Bildschirm. Dann können Sie mit einem kleinen Testprogramm fortfahren.

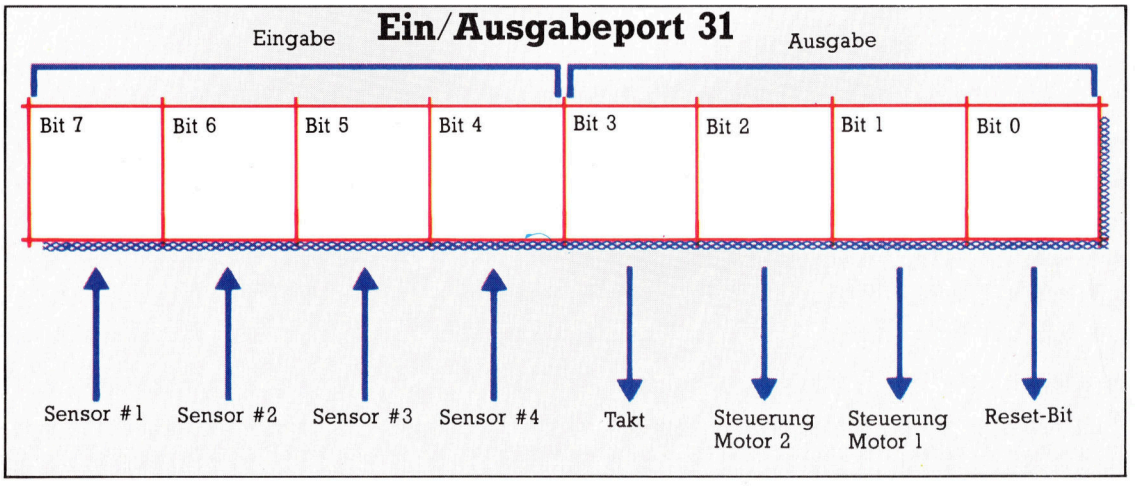
Der Roboter ist mit dem Ein/Ausgabeport 31 verbunden, dessen acht Bits zur Motorsteuerung und zur Verarbeitung der Sensormeldungen dienen. Die vier niederwertigen Bits regeln den Roboterantrieb, wobei Bit 1 und 2 die Drehrichtung der Schrittmotoren steuern. Durch das Setzen dieser beiden Bits in verschiedenen Kombinationen läßt sich jede Fahrtrichtung anwählen. Bit 3 ist das Taktbit – beim Umschalten von 0 auf 1 werden beide Motoren um je einen Schritt in die mit Bit 1 und 3 vorgewählte Richtung gedreht. Reset (Nullstellen) erfolgt mit Bit 0, das normalerweise auf 1 liegt.

Den Ausgangsteil des Interface können wir durch Eingeben und Starten eines Programms prüfen, mit dem sich der Roboter über die Tastatur lenken läßt. Die Buchstaben T, B, F und H entsprechen den Befehlen vorwärts, rückwärts, links und rechts.

Bit – Kombinationen

Die Anfangsroutine des Programms erzeugt entsprechend den vier möglichen Richtungen vier Variablen, deren Werte den Kombinationen von Bit 1 und 2 am Ein/Ausgabeport 31 entsprechen. Bei Betrachtung der Variablen und ihrer binären Vier-Bit-Werte (siehe Tabelle) wird auch klar, warum.

Die Zielrichtung des Roboters steht in der Variablen dr. Damit er sich in diese Richtung bewegt, müssen die Motoren von Bit 3 durch ein High mit nachfolgendem Low getaktet werden.



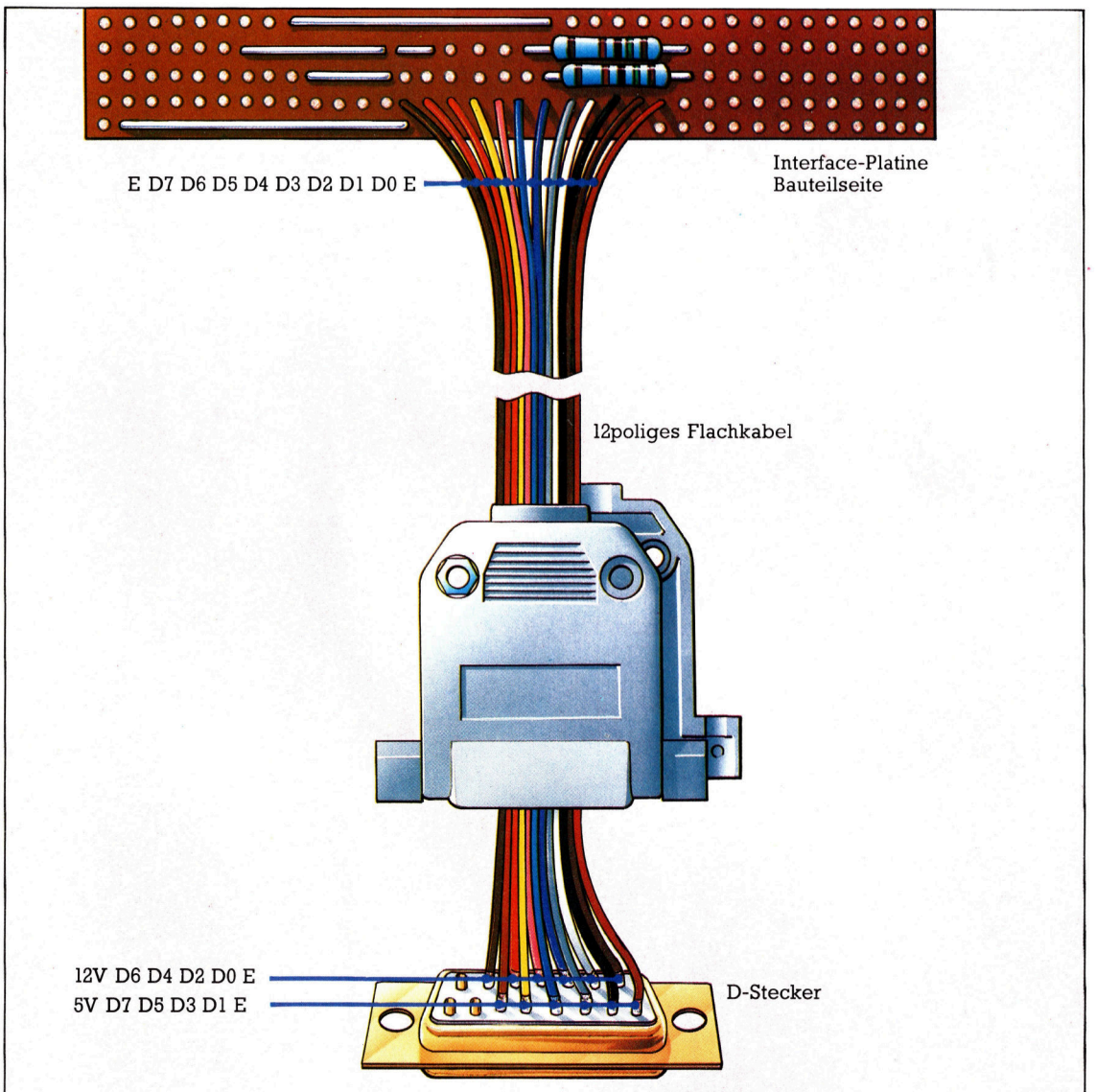
Das Spectrum-Interface nutzt die niederwertigen vier Bits des Ein/Ausgabeports 31 für die Ausgabe und die oberen vier für die Eingabe. Die Ausgabebits steuern die Funktion der Schrittmotoren, während die Eingabebits zur Sensorabfrage dienen.

Diesen Vorgang erledigt ein Unterprogramm (ab Zeile 2000), das die Werte mit OUT zum Port 31 schickt. Berechnung der Werte: dr+8 setzt Bit 3 auf 1, plus 1 setzt Bit 0 auf 1 – also dr+9. Zum Ausschalten von Bit 3 wieder 8 abziehen, ergibt dr+1. Das High auf Bit 3 mit nachfolgendem Low läßt die Motoren um einen 7,5-Grad-Schritt weiterdrehen. Die entsprechende Bewegung beträgt an den Rädern weniger als einen Millimeter, der Taktvorgang wird daher in eine Schleife geschrieben. Die Anzahl der Schleifendurchläufe steht in der Variablen m.

Der Wert der Variablen dr – und damit die Fahrtrichtung – wird über die Tastatur verändert. Beim nächsten Motortakt werden dann die Richtungsbits geändert – unser Roboter schlägt somit einen neuen Weg ein.

Richtung	Variable	Dezimal-wert	Binär-wert
Vorwärts	fw	4	0100
Rückwärts	bw	2	0010
Links	lf	6	0110
Rechts	rt	0	0000

Die höherwertigen vier Bits sollen Meldungen der Roboter-Sensoren überwachen, wobei jedes Bit einer bestimmten Buchse am Steckerfeld des Roboters entspricht. Bei offenen Microschaltern liegen alle Bits auf „High", also auf 1. Das Schließen der Schalter (Verbindung mit Masse) setzt sie dagegen auf „Low", also 0. Um die Eingabe für alle vier Bits voneinander zu trennen, brauchen wir eine Methode für die Abfrage der Bit-Einzelwerte. In den meisten BASIC-Dialekten kann dazu der logische AND-Befehl verwendet werden, um nicht abgefragte Bitwerte „auszuschalten". Auch der Spectrum kennt das logische AND, leider eignet sich der Befehl im Sinclair-BASIC nicht zu dem von uns gewünschten Zweck. Der Befehlssatz des Z80-Prozessors verfügt jedoch über ein AND, mit



Verbindung Interface-Platine/Roboterstecker

Die Verbindung zwischen Interface-Platine und D-Stecker ist unkompliziert – folgen Sie ganz einfach unserer Zeichnung. Beim Anlöten am D-Stecker kommt immer abwechselnd ein Drähtchen nach oben, eins nach unten.

dem sich diese Aufgabe bewältigen läßt. Also brauchen wir ein einfaches Maschinenprogramm, das ein logisches AND auf zwei Zahlen anwendet und das Ergebnis ausgibt. Die Zahlen, die mit AND verknüpft werden sollen, müssen dabei in zwei Speicherzellen gePOKEt werden. Das Ergebnis erhalten wir mit dem USR-Befehl. Durch diesen etwas umständlichen Vorgang kann auch der Spectrum die Eingabeleitungen unabhängig voneinander auswerten.

Das Spectrum-Sensorprogramm prüft die Funktion des Interface-Eingabeteils: Der Roboter fährt bis zum Auftreffen auf ein Hindernis vorwärts und kehrt danach auf seine Ausgangsposition zurück.

Spectrum-Besitzer müssen in den Commodore-Versionen der Steuerprogramme die unten beschriebenen Änderungen vornehmen.

BASIC-Dialekte

Im linearen Kalibrierprogramm (Seite 1322) müssen Zeilen 20, 30 und 70 gelöscht und folgende Zeilen verändert werden:

```
 80 LET dr=fw
100 LET a$=INKEY$: IF a$="" THEN GO TO 100
280 OUT 31,dr+9
290 OUT 31,dr+1
```

Im Kurven-Kalibrierprogramm (Seite 1323) die Zeilen 20, 30 und 50 löschen, die folgenden Zeilen ändern:

```
 60 LET dr=rt
 80 LET dr=fw
100 OUT 31,dr+9
110 OUT 31,dr+1
```

Im Roboter-Meßprogramm (Seite 1423) die Programmzeilen 1010, 1020, 1030, 7010 und 7510 löschen, folgende Zeilen ändern:

```
2050 GO SUB 8100: IF USR st<>lb THEN GO TO
     2040
2090 GO SUB 8100: IF USR st<>lb THEN GO TO
     2080
2150 GO SUB 8100: IF USR st<>rb THEN GO TO
     2140
2190 GO SUB 8100: IF USR st<>rb THEN GO TO
     2180
2200 OUT 31,0
3010 CLS
3520 GO SUB 8100: IF USR st=nb THEN GO TO 3510
4010 GO SUB 8100: IF USR st=rb THEN LET ss=rt:
     GO SUB 5000: RETURN
4020 GO SUB 8100: IF USR st=lb THEN LET ss=lf:
     GO SUB 5000: RETURN
5020 GO SUB 8100: IF USR st=nb THEN GO TO 5010
6080 GO SUB 8100: IF USR st=nb THEN GO TO 6070
8100 REM **** perform AND ****
8110 POKE st+1, 192: POKE st+3, IN 31: RETURN
8200 REM **** machine code loader ****
8210 FOR i=st TO st+8
8220 READ a: POKE i, a
8230 NEXT i
8240 DATA 62,0,14,0,161,6,0,79,201
8250 RETURN
```

Roboter-Steuerprogramm

```
10 REM **** spectrum robot controller ****
20 GO SUB 1000: REM  initialise
30 LET a$=INKEY$: IF a$<>"x" THEN
GO SUB 3000: REM read key
40 LET m=10: GO SUB 2000: REM pulse
50 IF a$<>"x" THEN  GO TO 30
60 OUT 31,0
70 STOP
80:
1000 REM **** initialise ****
1010 LET fw=4: LET bw=2: LET lf=6: LET rt=0
1020 LET dr=fw
1030 RETURN
1040:
2000 REM *** pulse ****
2010 FOR c=1  TO m
2020 OUT 31,dr+9
2030 OUT 31,dr+1
2040 NEXT c
2050 RETURN
2060:
3000 REM **** read keys ****
3010 IF a$="t" THEN  LET dr=fw: RETURN
3020 IF a$="b" THEN  LET dr=bw: RETURN
3030 IF a$="f" THEN  LET dr=lf: RETURN
3040 IF a$="h" THEN  LET dr=rt: RETURN
3050 RETURN
```

Spectrum-Sensorprogramm

```
10 REM **** spectrum bumpers ****
15 CLEAR 32499: LET st=32500
20 GO SUB 500: REM initialise
25 GO SUB 3000: REM load machine code
30 LET dr=fw
40 REM **** pulse forwards ****
50 LET cc=0
60 GO SUB 1000: LET cc=cc+1: REM pulse
70 REM ** test bumpers **
80 LET nm=192: GO SUB 2000: REM perform AND
90 IF USR st=192 THEN  GO TO 60
100 REM **** back to start ****
110 LET dr=bw
120 FOR i=1 TO cc
130 GO SUB 1000: REM pulse
140 NEXT i
150 OUT 31,0: STOP
500 REM **** initialise ****
510 LET fw=4: LET bw=2: LET lf=6: LET rt=0
520 RETURN
1000 REM **** pulse ****
1010 OUT 31,dr+9
1020 OUT 31,dr+1
1030 RETURN
2000 REM **** perform AND ****
2010 POKE st+1,IN 31
2020 POKE st+3,nm: RETURN
3000 REM **** machine code loader ****
3010 FOR i=st TO st+8
3020 READ a: POKE i,a
3030 NEXT i
3040 DATA 62,0,14,0,161,6,0,79,201
3050 RETURN
```

Alarmstufe ROT

Bei dem Programm „Flyerfox" können die Besitzer eines Commodore 64 ihre Geschicklichkeit in Luftkämpfen beweisen. Zwar gibt es schon viele Flugsimulatoren, doch bietet der Flyerfox von Tymac Sprachroutinen, die sich ohne zusätzliche Hardware einsetzen lassen.

Seit den ersten Versionen von „Space Invaders" wurden die Computerspiele – besonders im grafischen Bereich – immer mehr verfeinert. Die Programmierer „zwängten" dabei die Daten für mehr und mehr Grafikschirme in begrenzte RAM-Speicher, ließen aber die ausgezeichneten Soundmöglichkeiten vieler Heimcomputer unberücksichtigt.

Die amerikanische Firma Tymac bietet Spiele mit Sprachsynthese an, die ohne ein spezielles Sprachinterface auskommen. Eines der ersten dieser Art war „Flyerfox" für den Commodore 64. In diesem Flugsimulationsprogramm „fliegt" der Spieler ein Kampfflugzeug, das einen Jumbo Jet mit einem hochrangigen Regierungsbeamten durch einen gefährlichen Luftraum begleitet. Feindliche MiGs versuchen, den Jumbo abzuschießen. Und es ist die Aufgabe des Spielers, die gegnerischen Flugzeuge zu zerstören. Über die Sprachsynthese werden wichtige Meldungen aus dem Jumbo an den Spieler/Piloten im Begleitflugzeug weitergegeben.

Die Sprachsynthese ist Teil der Software und belegt etwa elf KByte. Bei Flyerfox werden die Wörter in digitale Signale umgewandelt und im RAM gespeichert. Wenn ein bestimmtes Wort gebraucht wird, ruft der Computer die entsprechenden digitalen Daten ab und erzeugt die Laute über den SID-Chip des Commodore.

Luftkämpfe

Die gesamte Grafik des Spiels wurde in hoher Auflösung programmiert. Auf dem Bildschirm erscheinen die Instrumententafel und der Himmel vor dem Cockpit. Es gibt verschiedene Navigationshilfen. Ein Radarschirm zeigt attackierende MiGs an, so daß der Spieler sich auf den Angriff vorbereiten kann. Und zwei blinkende Lichter zu beiden Seiten des künstlichen Horizonts informieren darüber, ob die MiG-Jäger sich ober- oder unterhalb der Cockpitebene befinden.

Die Luftkampfsequenzen sind schnell und werden sehr realistisch dargestellt. Wenn MiGs auf dem Schirm erscheinen, erzeugt das Programm einen Warnton, und der Spieler muß versuchen, den Angreifer in das Fadenkreuz seiner Zieleinrichtung zu bekommen. Diese Aufgabe ist nicht einfach, da die Flugzeuge mit hoher Geschwindigkeit ausweichen und wegtauchen. Erscheint das Ziel schließlich im Visier, kann der Spieler eine Rakete mit hitzeempfindlicher Zielvorrichtung abfeuern. Die Raketen treffen jedoch nicht immer, und die Feindflugzeuge entkommen recht häufig.

Über seine Instrumententafel erhält der Pilot des Flyerfox ebensoviele Informationen über die Positionen der feindlichen Flieger wie aus der Beobachtung des Himmels. Die Punkte auf dem Radar stellen Flugzeuge dar. Zwei weiße Quadrate zu beiden Seiten des Höhenmessers zeigen die relative Höhe der MiGs an. Der Kompaß hilft dabei, den Jumbo wiederzufinden.

Die Qualität der Grafik ist zwar hoch, bietet aber nicht viel Abwechslung. Die Bewegungsillusion wird durch wechselnde Wolkenbilder hervorgerufen und der Boden durch ein Raster dargestellt. Der Jumbo an sich trägt nur wenig zum Spiel bei, und es ist kaum zu erkennen, warum er sich überhaupt im Spiel befindet. Der Jumbo ist nur von hinten zu sehen und läßt sich beim Kampf mit feindlichen Flugzeugen nicht überholen. Im Gegensatz zu einer realistischen Situation unternimmt der Jumbo auch keinen Versuch, den angreifenden MiGs durch entsprechende Manöver zu entkommen.

Man wird den Verdacht nicht los, daß er vor allem deshalb in die Handlung eingeflochten wurde, um dem Sprachsynthesizer zu einer wichtigen Rolle zu verhelfen. Doch was soll's? Flyerfox-Fans dürften eher an Action als an Logik Interesse haben.

Flyerfox: Für den Commodore 64.
Hersteller: Tymac Corporation
Autoren: Gregory Carbonaro, Charles Teufert, Ronald Pintus, Arthur Aspromatis
Joysticks: Erforderlich
Format: Diskette oder Cassette

Geflügelte Worte

Programme in FORTH werden aus „Wörtern" aufgebaut, die der Programmierer selbst definieren kann. Die Wörter sind als Symbolgruppen definiert, die durch Leerzeichen voneinander getrennt werden. Wir untersuchen das „Vokabular" der Sprache, sehen uns an, wie FORTH Befehle, Konstanten und Variablen definiert.

Die Programmiersprache FORTH legt für ihre Befehle ein „Vokabular" an, das wie ein Wörterbuch eine Anzahl Wörter mit den zugehörigen Definitionen enthält. Die Wörter sind nicht alphabetisch gespeichert, sondern in der Reihenfolge, in der sie definiet wurden. Beim Aufruf eines Wortes teilt die Definition dem Computer mit, welche Abläufe er ausführen soll. Wörter können dabei entweder direkt eingegeben werden oder Bestandteil der Definition eines anderen Wortes sein.

FORTH interpretiert alles – Programme wie Daten – als Wörter. Die einzige Ausnahme sind Zahlen. Wörter lassen sich als Routinen (mit den Parametern Doppelpunkt und Semikolon), Variablen oder Konstanten definieren. Die Definitionen müssen aber in jedem Fall im Vokabular eingetragen sein.

Auch die ins System eingebauten Befehle sind Wörter. So stellt beispielsweise der Befehl WORDS (manche Systeme verwenden VLIST – die Kurzform für VokabularLISTe) eine Wörterliste des Vokabulars dar, in der der Befehl WORDS selbst als Eintrag auftaucht. Befehle von erweiterten FORTH-Versionen sind ebenfalls Wörter – Beispiele dafür sind GRAD und ELEVATION des Telescope-FORTH und FORWARD und RIGHT des Turtle-FORTH.

Vom Anwender definierte Befehle beginnen immer mit einem Doppelpunkt und enden mit dem Semikolon:

```
:PARK 0 GRAD AZIMUT 90 GRAD ELEVA-
TION;
```

Dabei kommt zuerst der (:), dann das definierte Wort, danach die Definition und schließlich ein (;). Die einzelnen Komponenten der Definitionen müssen durch Leerzeichen getrennt werden. Definitionen lassen sich aber auch auf mehrere Zeilen aufteilen. Sie werden dadurch übersichtlicher:

```
:PARK
0 GRAD AZIMUT
90 GRAD ELEVATION
;
```

Selbst die Programme werden als Wörter bezeichnet. Wenn Definitionen aus einer einzigen langen Befehlskette bestehen – beispielsweise mit dem Namen RUN – dann können Sie RUN als Programm ansehen, das andere Wörter als Subroutinen verwendet. Dabei können durchaus weitere Befehle/Programme im Vokabular eingetragen sein. Subroutinen sind Wörter, die in anderen Definitionen eingesetzt werden.

FORTH unterscheidet nicht zwischen Befehlen, Programmen und Subroutinen. Alle sind mit (:) und (;) definiert und können direkt (über die Tastatur) oder indirekt (von einer anderen Definition) aufgerufen werden. Da diesen Definitionen ein Doppelpunkt (englisch: Colon) voransteht, werden sie „Colon-Definitionen" genannt.

Variablen sind ebenfalls Wörter. Die Variable LAENGE wird zum Beispiel so definiert:

```
VARIABLE LAENGE
```

Bei dieser Definition wird dem Wort LAENGE im Vokabular zusätzlicher Speicherplatz für einen Wert zur Verfügung gestellt. Variablen müssen deklariert werden, bevor sie eingesetzt werden können, da sie ohne Vokabulareintrag nicht als Variablen erkannt werden. Die Zeichen @ („holen") und ! („speichern") werden in Verbindung mit Variablen eingesetzt.

```
LAENGE @
```

ergibt den Wert von LAENGE und

```
26 LAENGE !
```

setzt den Wert von LAENGE auf 26 (in BASIC identisch mit: LET LAENGE = 26).

Die Zahlen

Vor der Definition

```
0 CONSTANT 0
```

war 0 eine Zahl, die durch Regel 2 erkannt wurde. Nach der Definition ist sie ein Vokabulareintrag, der von Regel 1 erkannt wird, bevor Regel 2 überhaupt zur Anwendung kommt. Da die Zahlen 0, 1 und 2 oft vorkommen, werden sie von figFORTH im Vokabular definiert.

Es ist aber auch die Definition

```
12 CONSTANT 13
```

möglich, bei der Sie jedesmal, wenn Sie 13 eingeben, die Zahl 12 erhalten. (Durch die Eingabe von 013 wird jedoch der korrekte Wert [13] ausgegeben.)

Das @ ist unbedingt notwendig, wenn Sie den Wert einer Variablen abfragen möchten. Ohne das @ erhalten Sie lediglich die Adresse der Variablen.

Konstante sind ebenfalls Wörter. Wenn ein Wert häufig in einem bestimmten Zusammenhang verwendet wird, sollten Sie ihn als Konstante definieren, zum Beispiel:

```
66 CONSTANT KLICKKLICK
```

Das Wort KLICKKLICK hat nun die Bedeutung 66. Sie können natürlich auch eine Variable verwenden, doch lassen sich Konstante ohne @ einsetzen. Allerdings funktioniert der Speicherbefehl (!) nicht mit Konstanten.

Deklarationen mit VARIABLE und CONSTANT dürfen nicht innerhalb einer Colon-Definition eingesetzt werden. Der Grund dafür liegt in der Art, wie Definitionen im Vokabular gespeichert sind. Zwar wäre es praktisch,

```
:INITIALISIERE
VARIABLE PUNKTE 0 PUNKTE !
VARIABLE SPIELE 0 SPIELE !
```

definieren zu können, doch es funktioniet leider nicht. Wie in BASIC sind alle Variablen „global". Lokale Variable wie in PASCAL gibt es in FORTH nicht.

FORTH arbeitet mit +, −, *, / und vielen anderen arithmetischen Operatoren, die wie alle Wörter mit Leerzeichen von anderen getrennt sein müssen. Die Argumente werden jeweils vorangestellt. Statt 2 + 3 wird

```
2 3 +
```

geschrieben. In der nächsten Folge werden wir darauf noch ausführlicher eingehen.

Der Doppelpunkt, VARIABLE und CONSTANT werden „Definitionswörter" genannt, da sie Definitionen im Vokabular anlegen und somit eine Sonderfunktion ausführen. Sie bleiben aber Wörter des Vokabulars. Es ist möglich, neue Definitionswörter anzulegen.

Unter den Symbolen für die Programmstruktur gibt es DO und LOOP (entspricht etwa dem FOR. . .NEXT in BASIC), das (;) als Abschluß einer Colon-Definition und andere Strukturen wie IF. . .THEN und BEGIN. . .UNTIL, die für Verzweigungen und Schleifen eingesetzt werden. Auch hier ist es möglich, eigene Programmstrukturwörter zu definieren.

Wie behandelt FORTH nun Eingaben? Symbolgruppen ohne Leerzeichen werden als Wörter interpretiert und nach den folgenden drei Regeln bearbeitet:

1) Wenn FORTH das Wort im Vokabular findet, führt es die entsprechende Definition aus. Kommt das Wort mehrfach vor, wird die zuletzt eingegebene Definition eingesetzt.

2) Steht das Wort nicht im Vokabular, dann prüft FORTH, ob das Symbol (oder die Symbolgruppe) eine Zahl ist. Ist dies der Fall, legt es die Zahlen auf dem „Stapel" ab.

Die Leerzeichen

Da Wörter aus Zeichenfolgen bestehen, sieht FORTH die folgenden Angaben als Wörter an (die jedoch nicht im Vokabular definiert sein müssen):

```
KARTOFFEL
ersehnterLottogewinn
239
pH
25DM
+
"achtung"
,!XXX)hallo-dortQWERTZ
```

Außer beim Leerzeichen versucht FORTH nicht, den Zeichen eine bestimmte Bedeutung zuzuordnen. Sie werden vermutlich Wörter wie LAENGE und PUNKTE für Ihre Variablen einsetzen, doch FORTH würde auch 23PI/2 oder 1BIS10 akzeptieren.

In vielen Sprachen müssen Variablennamen mit einem Buchstaben anfangen und dürfen danach nur aus Buchstaben und Zahlen bestehen. Ursache dafür ist die Interpretation des Leerzeichens. Die meisten Sprachen sehen Leerzeichen lediglich als „Dekoration" an, durch die Programme übersichtlicher werden. Da sie problemlos weggelassen werden können, enthalten mathematische Ausdrücke wie 2+3 in BASIC selten zusätzliche Leerzeichen.

Die Sprachen müssen daher andere Möglichkeiten haben, zwischen den verschiedenen Komponenten eines Programms unterscheiden zu können. Wenn Sie in BASIC eine Variable 2+3 nennen würden, weiß der Interpreter nicht, ob Sie die Variable oder einen mathematischen Ausdruck meinen. Um diese Mehrdeutigkeit auszuschließen, werden Variablennamen Einschränkungen unterworfen.

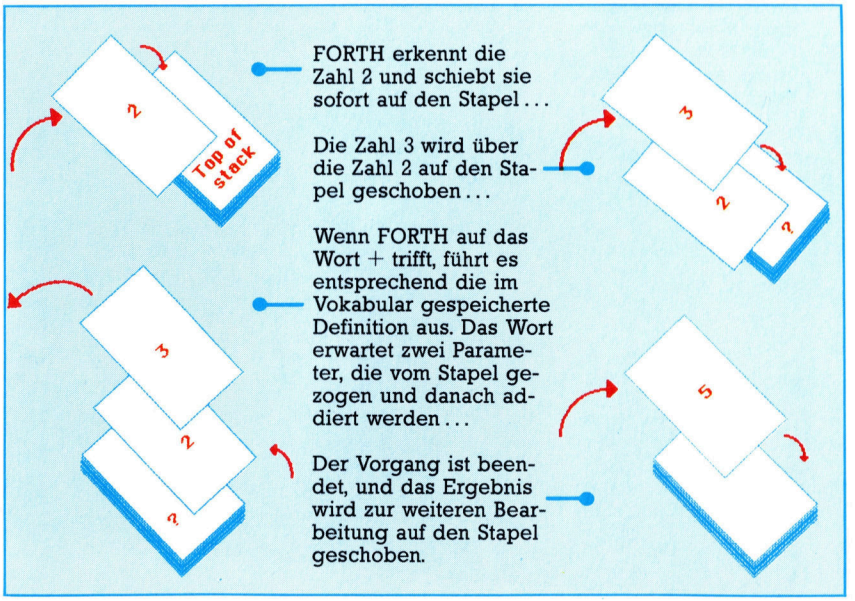
FORTH löst dieses Problem auf andere Weise. Da das Leerzeichen kompromißlos als Trennelement zwischen Wörtern eingesetzt wird, sind keine weiteren Regeln nötig.

FORTH-Dialekte
In figFORTH muß der Wert einer Variablen jeweils vor der Variablen stehen.

```
10 VARIABLE
BLUMENVASEN
```

In diesem Beispiel ist der Wert Zehn.

Wenn FORTH auf ein Wort trifft, führt es den Vorgang aus, der unter diesem Wort im Vokabular gespeichert ist. Trifft es jedoch auf eine Zahl, dann speichert FORTH diese in einem Bereich, der „Stapel" genannt wird. Dort bleibt die Zahl, bis FORTH sich wieder daran „erinnern" muß – beispielsweise bei der Ausführung eines weiteren Befehls. Unser Bild zeigt, wie FORTH die Eingabe 2 3 + interpretiert.



3) Ist die Symbolgruppe weder eine Zahl, noch ein im Vokabular eingetragenes Wort, gibt FORTH eine Fehlermeldung aus.

Bei Colon-Definitionen sind diese Regeln etwas geändert, da die entsprechenden Vorgänge nicht direkt ausgeführt werden können. Wörter wie (:), VARIABLE und CONSTANT sind daher Ausnahmen. Die Schreibweise

```
VARIABLE PUNKTE
```

würde gemäß Regel 3 einen Fehler erzeugen. VARIABLE behandelt PUNKTE jedoch auf eigene Weise und umgeht so die drei Regeln. Wenn dies nicht der Fall wäre, ließen sich keine Wörter in das Vokabular eintragen.

Durch das Vokabular erhält FORTH eine breit angelegte, interaktive Flexibilität. Die drei Regeln stellen sicher, daß neue Definitionen mit den alten Definitionen gleichgestellt sind.

Zurück zum Ursprung

Mit ein wenig Erfahrung in der Maschinencodeprogrammierung ist FORTH leicht zu verstehen. Für Anfänger ohne diese Kenntnisse können die ersten Programmierversuche oft sehr frustrierend sein.

Bei der Definition des Wortes QUADRAT, das als Subroutine auf dem Bildschirm ein Quadrat mit der Seitenlänge SEITE zeichnet, gehen wir folgendermaßen vor. Zunächst definieren wir die Variable SEITE:

```
VARIABLE SEITE
```

FORTH wird dabei angewiesen, Platz zu reservieren, an dem der numerische Wert gespeichert werden kann. Mit @ (holen) können wir von dieser Adresse Werte abrufen und mit ! (speichern) dort ablegen.

```
20 SEITE !
```

ordnet der Variablen SEITE den Wert 20 zu.

Wir können das neue Wort QUADRAT nun mit den „Wörtern" : und ; (ähnlich wie TO und END in LOGO) definieren:

```
:QUADRAT
  SEITE !
    4 0 DO
      SEITE @ FORWARD
    90 RIGHT
  LOOP
```

Diese Prozedur setzt voraus, daß die Wörter RIGHT und FORWARD bereits definiert wurden, bei denen beispielsweise die Funktion

```
50 FORWARD
```

die Turtle 50 Bildschirmpunkte vorwärts bewegt.

Weiterhin muß der Anwender den Wert für SEITE eingeben, so daß

```
50 QUADRAT
```

ein Quadrat mit der Seitenlänge von 50 Bildschirmpunkten zeichnet.

Sehen wir uns einmal an, wie die Subroutine arbeitet, wenn wir 50 QUADRAT eingeben. FORTH untersucht zunächst die Eingabe und stellt fest, daß die erste Symbolgruppe (50) durch ein Leerzeichen von der nächsten (QUADRAT) getrennt ist. FORTH überprüft nun, ob diese Gruppe im Vokabular eingetragen ist. Wenn Sie die Zeichengruppe 50 zuvor nicht als Wort definiert haben, prüft FORTH, ob es sich um eine Zahl handelt. Da dies der Fall ist, legt FORTH die Zahl im Speicher ab und untersucht die nächste Symbolgruppe – QUADRAT

QUADRAT ist natürlich im Vokabular vorhanden (wir haben sie gerade definiert). FORTH ruft nun die Definition auf. Dabei trifft es als erstes auf den Ausdruck SEITE !. Da SEITE ! an dieser Stelle kein Wert zugewiesen wurde, prüft FORTH seinen Speicher und findet dort den von uns eingegebenen Wert (50), den es der Variablen SEITE nun zuordnet.

Im Gegensatz zu Logo überprüft FORTH an dieser Stelle nicht, ob dies der richtige Wert ist, sondern nimmt an, daß wir die Zahl an der korrekten Stelle eingegeben haben. Wenn wir statt 50 QUADRAT nur QUADRAT angeben, erhalten wir keine Fehlermeldung.

4 0 DO...LOOP erklärt sich eigentlich von selbst. Interessant ist hierbei, daß die Variable SEITE im Inneren der Schleife durch die Wörter SEITE @ FORWARD an die Subroutine FORWARD übergeben wird. Die Subroutine ruft dabei zunächst die Adresse von SEITE auf und holt sich dann (mit @) den Wert dieser Adresse. 90 RIGHT ist notwendig, um die Turtle nach Abschluß einer Seite um 90 Grad zu drehen und das Quadrat zu vervollständigen.

Die Definition des Wortes QUADRAT zeigt nur einen der vielen Aspekte, die die Flexibilität von FORTH bietet. Es ist nicht nur möglich, „Vokabulare" anzulegen, die mechanische Geräte wie das im Bild gezeigte Teleskop steuern, es lassen sich auch Systeme für abstrakte Anwendungen damit erstellen. So setzt beispielsweise das britische Softwarehaus *Mastertronics* FORTH für die Entwicklung von Abenteuerspielen ein.

Pluspunkte

Der Acorn B wurde 1981 auf den Markt gebracht. Nach anfänglichen Verkaufserfolgen geriet der Rechner mit den Jahren gegenüber der Konkurrenz ins Hintertreffen. Bei der erweiterten Version Acorn B+ ist zwar durch Vergrößerung des Arbeitsspeichers der Hauptmangel behoben, aber wegen des hohen Verkaufspreises dürften nur schwer neue Käuferkreise zu erschließen sein.

Für die Wahl des Acorn B seitens der englischen Fernsehgesellschaft BBC war ausschlaggebend, daß dieser Rechner mit seiner Geschwindigkeit und Vielseitigkeit gegenüber den anderen damaligen Heimcomputern einen gewaltigen Fortschritt darstellte. Aber seit 1981 hat sich sehr viel getan. Zwar liegt der Acorn hinsichtlich Schnittstellen und Peripherieangebot immer noch vorn, doch sind auch einige Schwächen zutage getreten.

Der größte Nachteil dieses Computers ist der Mangel an freier RAM-Kapazität. Wenn mit hoher Grafikauflösung gearbeitet wird, bleibt im Speicher nur wenig Platz für BASIC-Programme. Ursache dafür sind die einschränkenden Bedingungen, die die Adressierungsmöglichkeiten eines 8-Bit-Prozessors bei der Implementierung einer durchaus leistungsfähigen Grafik auferlegen.

Als im Lauf der letzten Jahre die Konkurrenz angesichts des allgemeinen Preisverfalls bei Prozessor- und Speicherchips immer schärfer kalkulierte, während Acorn am ursprünglichen Verkaufspreis festhielt, wurde die Lage für diesen Computer bedrohlich. Acorn vertraute bei seiner Preispolitik wohl auf den prestigeträchtigen BBC-Vertrag und auf die Zuschüsse, mit denen die britische Regierung die Rechneranschaffung für Ausbildungszwecke im Rahmen eines Förderprogramms unterstützt.

16 Bit contra 8 Bit

Hinzu kam die Gefahr, mit dem 8-Bit-Prozessor den Anschluß an die Entwicklung zu verlieren. Die Firma Sinclair Research – ein aussichtsreicher Mitbewerber um den BBC-Vertrag – brachte ihren QL mit 16 Bits zum gleichen Preis auf den Markt, der in England für den Acorn B verlangt wurde.

Die Befürchtung, daß die 16-Bit-Rechner die 8-Bit-Maschinen völlig an die Wand drücken würden, hat sich allerdings zumindest kurzfristig nicht bestätigt: Da das Computergeschäft hinter den Erwartungen zurückblieb, was unter anderem Acorn und Sinclair in arge finanzielle Bedrängnis brachte, verzichteten die meisten Hersteller lieber auf risikoreiche Neuentwicklungen und beschränkten sich auf verbesserte Versionen ihrer bewährten Modelle. Fairerweise muß man sagen, daß dieses Vorgehen den Wünschen der meisten Käufer entgegenkommt. Der Heimcomputermarkt wird immer unübersichtlicher, und daher legen die Kunden mehr Wert auf eine breite und solide Software-Basis als auf den allerneuesten Computer. Das rechtfertigt die Entwicklung leistungsfähigerer Rechner, die mit dem Vorgängermodell software-kompatibel sind.

Der Acorn-B+ erlaubt dem Benutzer, vorhandene Software für den Acorn B weiter einzusetzen, bietet aber gleichzeitig den nötigen Speicherplatz für aufwendigere Programme. Weil der neue Rechner mit mehr als 64KByte RAM ausgestattet wurde, ist eine direkte Adressierung des gesamten Speicherplatzes durch den 8-Bit-Prozessor nicht mehr möglich. Acorn macht wie Atari und Commodore vom „Bank Switching"-Verfahren Gebrauch, um den zusätzlichen Speicherraum erreichen zu können.

Der Arbeitsspeicher wird dabei in mehrere „Speicherbänke" unterteilt, die den gleichen Adreßraum belegen. Je nachdem, welcher Teil des Speichers angesprochen wird, arbeitet der Prozessor mit der einen oder der anderen RAM-Bank, das heißt, es werden unterschiedliche Speicherinhalte über die gleiche Adresse angesprochen.

Die zusätzlichen 32 K RAM werden über die Adressen 12288-45055 (hexadezimal 3000-AFFF) angesprochen. Dieser Bereich umfaßt den größten Teil des BASIC-Programmspeichers und außerdem das Feld für die ROM-Seiten, auf denen das BASIC und zusätzlich eingebaute Systeme wie View oder LOGO stehen. Die zusätzliche RAM-Bank überdeckt also zwei verschiedene Abschnitte des Speichers.

Wie erwähnt beruht die Speicherknappheit des Acorn B hauptsächlich darauf, daß die hochauflösende Grafik viel Platz im RAM beansprucht. Deshalb wird der größte Teil der Zusatzkapazität als Video-RAM reserviert, nämlich die 20 KByte, die über dem BASIC-Programmspeicher liegen. Das reicht auch für die speicherplatzintensiven Bildschirm-Betriebsarten. Bei der Neugestaltung der Rechnerplatine für die Speichererweiterung hat Acorn auch sonst noch eine Reihe von Änderungen vorge-

Acorn B+

SPEICHER

76 KByte RAM, davon 64 KByte frei für BASIC-Programme; 32 KByte ROM, erweiterungsfähig auf 192 KByte.

ZENTRALEINHEIT

6512-Prozessor mit 2 MHz Taktfrequenz.

DISKETTENBETRIEB

Das zugehörige Betriebssystem (Acorn DFS) ist im Grundpreis enthalten.

DOKUMENTATION

Das ausführliche Benutzerhandbuch ist überarbeitet worden und enthält jetzt noch mehr Informationen (nicht nur für den Gebrauch der zusätzlichen RAM-Bank).

STÄRKEN

Die höhere Arbeitsspeicher-Kapazität ermöglicht die Entwicklung umfangreicher BASIC-Programme in Verbindung mit hoher Grafik-Auflösung.

SCHWÄCHEN

Auch bei Berücksichtigung der Verbesserungen ist der Acorn B+ für die meisten Heimcomputer-Interessenten noch sehr teuer, und im Ausbildungsbereich geht der Trend jetzt zu den MS-DOS-Rechnern. Daher erscheinen die Marktchancen für das neue Modell nicht gerade günstig.

nommen und beispielsweise statt des bisherigen 6502-Prozessors einen 6512 verwendet, der besser mit dem Schnittstellenbaustein 6522 harmoniert. Der Datenverkehr verlief zuvor nicht immer reibungslos. Die Bank-Switching-Adresse selbst (FE34) steht in einem der weiterentwickelten Betriebssystem-ROMs (IC36). Wegen der weitgehenden Interrupt-Steuerung des Acorn B erforderte das Bank Switching keine tiefgreifende Systemänderung.

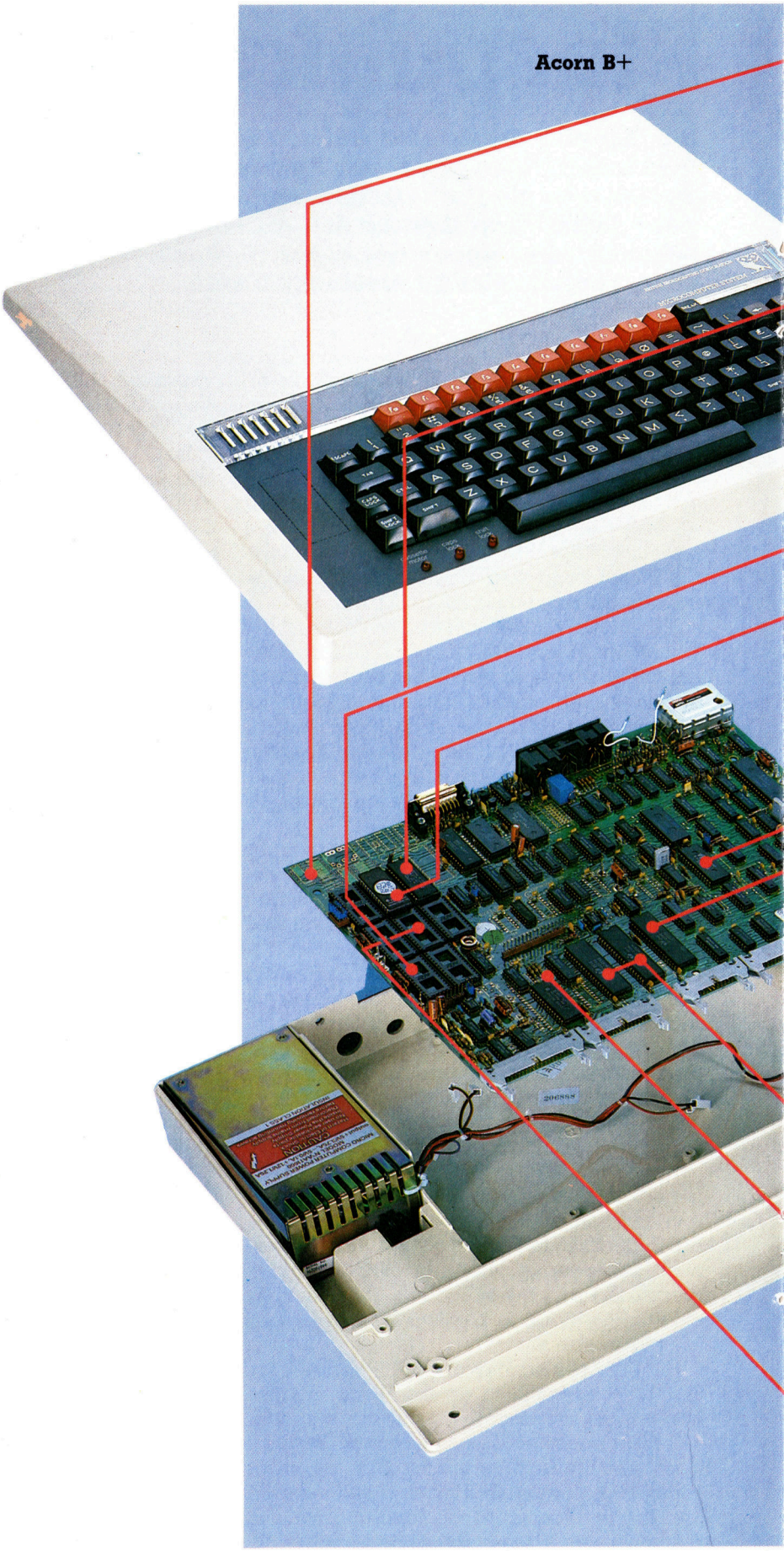
Die übrigen neuen 12 K RAM liegen auf den Adressen 32768-45055 (8000-AFFF hexadezimal) – das ist der Bereich, der für die ROM-Einblendung vorgesehen ist. Das Bank Switching erlaubt auch hier im zeitlichen Nacheinander wahlweise das Ansprechen der ROM-Seiten oder der zusätzlichen RAM-Bytes.

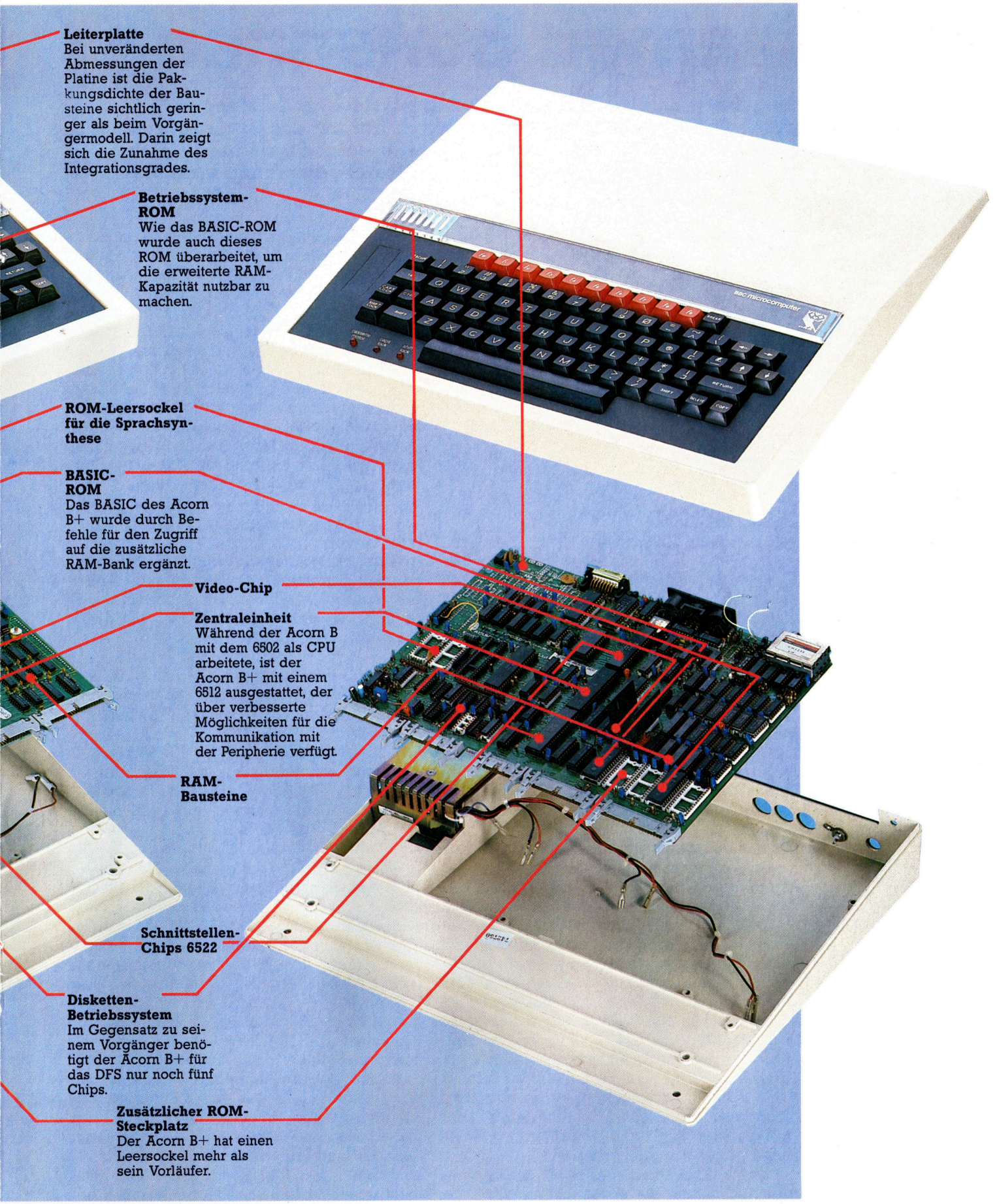
Shadow-RAM

Der Acorn B+ bietet gegenüber den Atari- und Commodore-Rechnern den Vorteil, daß der Benutzer auf die zweite RAM-Bank, die als „Shadow-RAM" bezeichnet wird, ohne umständliche POKE-Befehle zugreifen kann. Acorn hat auch das BASIC und das Betriebssystem (MOS = Machine Operating System) überarbeitet, um das Bank Switching softwaremäßig deutlich zu unterstützen.

Der Bildschirmspeicher im Shadow-RAM ist auf verschiedene Weise zugänglich. Ein MODE-Befehl mit Kennzahl zwischen 128 und 135 (entsprechend den Bildschirm-Betriebsarten 0 bis 7) spricht automatisch das Shadow-RAM an. Das Kommando *SHADOW legt den Bildschirmspeicher dauerhaft in das Shadow-RAM, so daß ein versehentliches Überschreiben der BASIC-Programme verhindert wird. Diese Einstellung läßt sich mit *SHADOW 1 rückgängig machen. Wie für alle * -Kommandos des Acorn gibt es auch für die SHADOW-Befehle analoge *FX-Kommandos, nämlich *FX 114 bzw. *FX 114,1. Entsprechend können Sie statt der MODE-Befehle VDU 22, (128+n) verwenden, wobei n den gewünschten Modus entsprechend angibt.

Im Zuge der Aufbesserung gehören jetzt beim Acorn B+ die ROMs mit dem Disketten-Betriebssystem (DFS-Disk Filing System) zur Grundausstattung. Die ist sicherlich nützlich für Käufer, die gleich von Anfang an mit Disketten arbeiten wollen und sonst ein paar hundert Mark extra ausgeben müßten. Aber die meisten Benutzer möchten bloß ein Kassettenlaufwerk anschließen, und das vorhandene DFS bringt ihnen nichts als einen höheren Einstiegspreis. Das bedeutet Wasser auf die Mühlen jener Kritiker, die eine Preissenkung, wie sie andere Hersteller schon längst praktiziert haben, auch beim BBC-Acorn für überfällig halten. Mittlerweile ist für den Acorn B+ eine externe Speichererweiterung erhältlich.




Leiterplatte
Bei unveränderten Abmessungen der Platine ist die Packungsdichte der Bausteine sichtlich geringer als beim Vorgängermodell. Darin zeigt sich die Zunahme des Integrationsgrades.
Betriebssystem-ROM
Wie das BASIC-ROM wurde auch dieses ROM überarbeitet, um die erweiterte RAM-Kapazität nutzbar zu machen.
ROM-Leersockel für die Sprachsynthese
BASIC-ROM
Das BASIC des Acorn B+ wurde durch Befehle für den Zugriff auf die zusätzliche RAM-Bank ergänzt.
Video-Chip
Zentraleinheit
Während der Acorn B mit dem 6502 als CPU arbeitete, ist der Acorn B+ mit einem 6512 ausgestattet, der über verbesserte Möglichkeiten für die Kommunikation mit der Peripherie verfügt.
RAM-Bausteine
Schnittstellen-Chips 6522
Disketten-Betriebssystem
Im Gegensatz zu seinem Vorgänger benötigt der Acorn B+ für das DFS nur noch fünf Chips.
Zusätzlicher ROM-Steckplatz
Der Acorn B+ hat einen Leersockel mehr als sein Vorläufer.

Namen benennen

Sie wissen bereits, daß ein Datenbanksystem große Mengen miteinander verknüpfter Informationen nur auf der Grundlage genauer Strukturierung verwalten kann. Wie eine solche Struktur aussieht, möchten wir Ihnen hier anhand einer Adreßverwaltung vorführen.

Der große Vorteil des altmodischen Adreßbuchs ist, daß es neben einer alphabetischen Ordnung nicht zusätzlich strukturiert werden muß. Ein Eintrag kann zum Beispiel so aussehen:

PETER GERBER, Buchenring 16
Barmstedt (04123-601227)
– seine Freundin heißt Anke – Büro:
(04123-680545, Zimmer 160)
– nach 16.00 anrufen – zieht im Januar um

als nächster Eintrag
GEORG – 040-9013546

als nächster Eintrag
GÖTZ – siehe Müller-John

In einem Adreßbuch können Sie Bekannte beliebig unter dem Vornamen, Nachnamen oder auch mit „indirekter Adressierung" durch „Zeiger" zu anderen Einträgen auflisten. Bei der Eingabe von Informationen in eine Computer-Datenbank muß man dagegen sehr viel systematischer vorgehen.

Die Flexibilität eines herkömmlichen Adreßbuchs kann ein Computer-Adreßprogramm zwar nicht bieten, man kann diesem Vorbild aber zumindest nacheifern. Wir behandeln die Felder nacheinander, zuerst das Namensfeld:

Namen sind zweiteilig, sie setzen sich aus dem Vor- und dem Familiennamen zusammen. In den meisten Ländern steht der Familienname hinter dem Vornamen. Japan, Ungarn und einige andere Länder bilden eine Ausnahme – dort steht der Familienname an erster Stelle.

Obwohl wir erst beim Namensfeld sind, gibt es bereits mehrere Stolpersteine. Es bleibt uns nichts übrig, als die Reihenfolge von Familien- und Vornamen sowie auch deren maximale Länge fest zu definieren. Namen können so kurz sein wie Jo, aber auch so lang wie Mueller-Luedenscheidt oder noch umfangreicher sein – wir müssen also für Extremfälle genug Platz vorsehen. Wenn die Datenbankverwaltung (DBV) mit Feldern fester Länge arbeitet, sollten diese also ausreichend dimensioniert sein. (Auf die Nachteile von Datenfeldern mit fester Länge gehen wir später noch ein.)

Für das Gesamtformat ist es am günstigsten, den Nachnamen als erstes Schlüsselfeld zu wählen, auf das ein weiteres Feld für einen oder mehrere Vornamen folgt. Auf das nächste Problem stoßen wir beim Format der Adresse. In amerikanischen Zeitschriften haben Sie sicher schon einmal die Angaben zu einer USA-Anschrift gesehen: Stadt, Staat und „Zip Code", eine Abfolge, die für deutsche Verhältnisse völlig ungeeignet ist. Auch die bei uns übliche Folge aus Name, Straße, Hausnummer, Postleitzahl, Wohnort und Land kann nicht auf andere Länder übertragen werden: In England beispielsweise steht die Hausnummer vor dem Straßennamen, in Japan hat der Wohnort die oberste Position, während der Name des Empfängers als letztes geschrieben wird.

Kompromisse eingehen

In einfachen wie auch in komplexen Datenbanken läßt sich natürlich nicht jedes Schreibformat berücksichtigen. Ohne Kompromisse geht es nicht. Im Normalfall sollte man mit dieser Aufteilung zurechtkommen:

- Feld für Familiennamen – bis zu 40 Zeichen
- Feld für Vornamen – bis zu 60 Zeichen
- 1. Zeile Adreßfeld – bis zu 80 Zeichen
- 2. Zeile Adreßfeld – bis zu 80 Zeichen
- 3. Zeile Adreßfeld – bis zu 80 Zeichen
- 4. Zeile Adreßfeld – bis zu 80 Zeichen
- 5. Zeile Adreßfeld – bis zu 80 Zeichen
- Feld für Telefonnummer – bis zu 20 Zeichen
- Notizfeld – bis zu 80 Zeichen

In diesem Schema lassen sich die meisten Adressen unterbringen. Allerdings können auch hier noch Schwierigkeiten auftauchen, etwa ein Platzmangel im Notizfeld. Ein weiteres Problem ist, daß für das Land kein spezieller Platz definiert ist – es kann im dritten, vierten, fünften oder gar keinem Adreßfeld auftauchen. Das macht so lange nichts, wie Sie Ihre Datenbank nicht mit dem Suchbegriff „Land" bearbeiten wollen. Enthält eine Datenbank viele ausländische Einträge, sollte ein spezielles Feld für das Land eingerichtet werden. Die Entscheidung dafür muß aber in jedem Fall bereits im Planungsstadium fallen.

Die Datenbankverwaltung, die Sie auf Ihrem Rechner laufen lassen, entscheidet über die Leichtigkeit, mit der eine spezielle Datenbank eingerichtet werden kann. Eine der einfachsten Verwaltungen ist „Caxton's Card Box". Ihre

Möglichkeiten der Datensuche und -veränderung sind zwar beschränkt, dafür kann man jedoch bei einfachen Anwendungen mit einem Minimum an Aufwand zu guten Resultaten kommen. Auf eine komplizierte Programmiersprache zur Veränderung von Feldern und Datensätzen wurde bei Card Box verzichtet. Auf die gewünschten Records kann mit einfachen Befehlen zugegriffen werden. Mit Card Box kann die oben beschriebene Datenbank nach dem Laden des Programms durch eine Folge einfacher Eingaben erzeugt werden.

Als erstes muß man sich für ein bestimmtes Datensatz-Format entscheiden. Davon hängt ab, welche Informationen gespeichert werden und welchen Index sie tragen (etwa, welches Feld als Suchfeld definiert ist). Dabei wird auch die visuelle Darstellung der Datensätze auf dem Bildschirm bzw. beim Ausdruck festgelegt. Beim Aufruf des Datenbank-Files ADBOOK erzeugt Card Box das dazugehörige Format-File ADBOOK.FMT. Durch Modifikation von ADBOOK.FMT kann die Darstellung der Informationen jederzeit geändert werden. Sie können auch mehrere Format-Files einrichten, um den Datenbankinhalt in unterschiedlichen Formen darzustellen.

Wie bei den meisten DBVs kann auch mit Card Box jedem Feld ein „Dauertext" zugeordnet werden. In einer Datenbank mit Adressen sind Namen, Wohnorte und Telefonnummern leicht zu unterscheiden, in anderen Dateien brauchen Sie dagegen einen Hinweis auf die Bedeutung der einzelnen Felder. In diesem Fall bewährt sich der Dauertext.
Vergleichen Sie einmal diese beiden Datensätze:

```
06116
3995
86
34,75
Flansch mit Ventilklappe
```

und

```
BESTELLNUMMER
DER LIEFERFIRMA                    06116
UNSERE KATALOGNUMMER               3995
LAGERBESTAND (STÜCK)               86
PREIS                              34,75
BEZEICHNUNG      Flansch mit Ventilklappe
```

Kaum Fehler

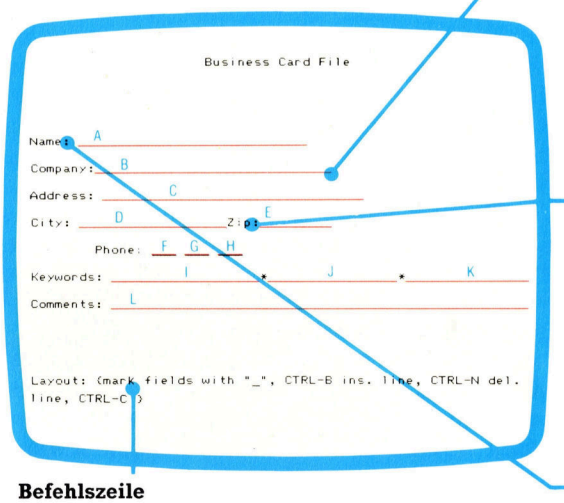
In beiden Fällen ist die Information dieselbe, die Erläuterungen im zweiten Datensatz machen aber Fehler sehr viel unwahrscheinlicher.

Mit Card Box kann jedes Feld mit einem von den vier Indizes NONE, MAN(ual), AUTO oder ALL versehen werden. Für eine Lagerverwaltung würde man etwa das PREIS-Feld nicht als Suchfeld definieren – wer fragt schon: „Haben Sie Artikel, die weniger als 60 Mark, aber mehr als 30 Mark kosten?" Man möchte eher wissen, wieviel Stück noch von Artikel #06116 am Lager sind – die „Bestellnummer der Lieferfirma" müßte also als Suchfeld definiert sein.

In unserer Adreßkartei wollen wir die Datenbank sicherlich nach Familiennamen, vielleicht aber auch nach Vornamen durchsuchen – wir müssen also beide Felder als Suchfelder ausführen. In einem Geschäftsbetrieb ist auch das Sortieren nach Städten oder gar nach den Vorwahlnummern üblich. Datenbanken unterscheiden sich vom althergebrachten Karteikasten: Sie sind nur dann effektiv, wenn man sich bei ihrer Einrichtung über die Art und Weise der späteren Nutzung klar ist.

Form mit Format

Mit „Megafinder" kann man eigene Formulare erzeugen. Dieses hier haben wir aus der mit dem Programm gelieferten Formularversion (unteres Bild) entwickelt.

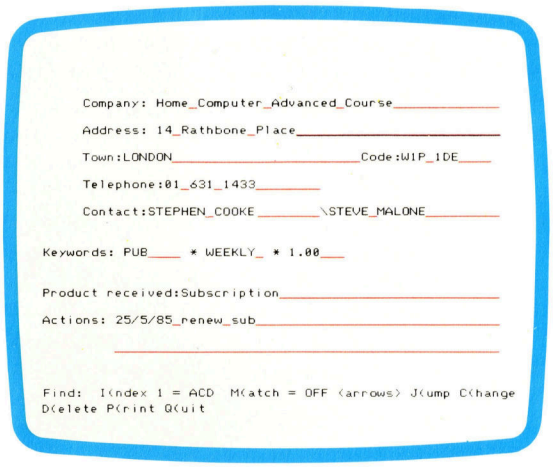


Feldlänge
Wird durch die Unterstreichung dargestellt(_). Die Länge aller Felder eines Datensatzes zusammen entscheidet darüber, wieviele Karteikarten in einer Datei gespeichert werden können.

Feldbezeichnungen (Labels)
Der Anwender kann die Feldbezeichnungen beim Entwurf des Formulars frei festlegen. Sie werden nur bei der Darstellung von Informationen auf dem Bildschirm verwendet.

Feldmarkierungen (Identifiers)
Die Markierungen bestimmen bei der Dateneingabe die Reihenfolge des Zugangs zu den Feldern. Sie ermöglichen auch das Sortieren nach einzelnen Feldern bzw. Feldgruppen. Mit dem INDEX-Befehl kann bei Megafinder das Feld D etwa den Index 1 erhalten. Anschließend kann der Anwender schnell alle Records nach der alphabetischen Reihenfolge ihrer Städtenamen sortieren lassen.

Befehlszeile
Hier stehen einige der Befehle für das Editieren eines Datensatzes. Mit CTRL-C wird die Eingabe auf der Diskette abgelegt.

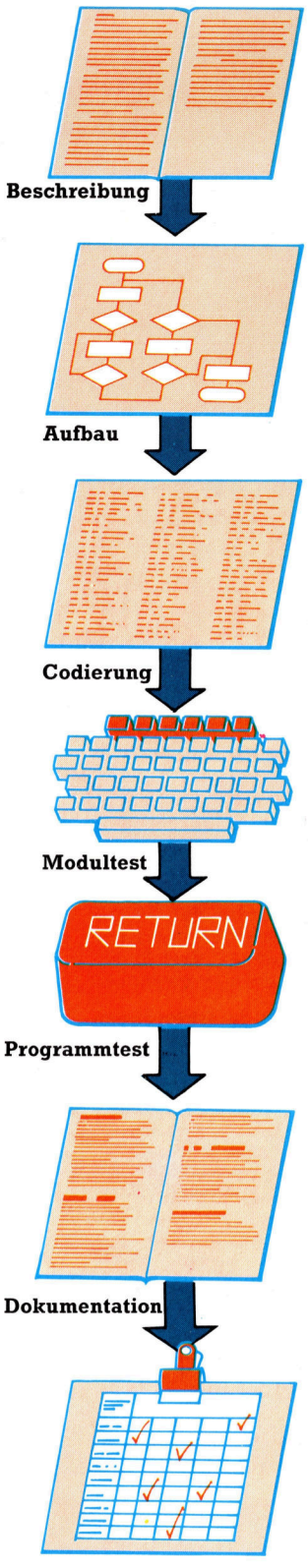


Datenbank-Systeme bieten eine Vielzahl von Möglichkeiten zum Bearbeiten von Informationen. Die Bildschirmdarstellung des Datenbankprogramms Megafinder zeigt einen einzelnen Datensatz aus der Datei „Business Card". Zum schnellen Auffinden von Daten können bis zu vier unterschiedliche Indizes eingesetzt werden.

Die DBV erkennt die Übereinstimmung neu eingegebener und bereits vorhandener Daten (MATCH). JUMP springt zu einzelnen Abteilungen (etwa dem Verzeichnis aller Firmen mit dem Anfangsbuchstaben M), CHANGE verändert und DELETE löscht Informationen. Mit PRINT wird ein Eintrag gezeigt oder ausgedruckt.

Mit Struktur

In dieser Serie wurden zunächst der Befehlssatz des 6809 untersucht und dann einige einfache Routinen angelegt. Nun stellen wir Techniken vor, mit denen sich umfangreiche Assemblerprogramme strukturieren lassen.



Es ist zwar nicht immer einfach, Maschinencode strukturiert zu programmieren, doch macht sich der Aufwand durch kürzere Testzeiten und übersichtliche Programme bezahlt.

Wenn korrekt angelegte Programme, modularer Aufbau und strukturierte Programmierung schon bei höheren Programmiersprachen große Vorteile bringen, so kommt man bei dem komplizierten Code der Maschinensprache ohne Systematik überhaupt nicht mehr zurecht. Nun besitzt die Assemblersprache keine vorgefertigten Strukturen wie WHILE . . .WEND oder IF . . .THEN . . .ELSE, die von sich aus schon eine Systematik vorgeben. Es gibt keine einheitliche Schreibweise, keine Datentypen für Variablen, und im Vergleich mit Hochsprachen haben Assemblerprogramme die sechs- bis zehnfache Menge von Befehlen. Auch Fehler mit weitreichenden Konsequenzen können leicht vorkommen: Ein einziges fehlerhaftes Byte kann durchaus alle Daten einer Diskette löschen. In diesem Artikel stellen wir daher eine Methode vor, die den Umgang mit der Assemblersprache erleichtert.

Schon früh haben Programmierer erkannt, daß Vorrausplanung und Klarheit die Grundlage jedes erfolgreichen Programmierstils ist. Doch obwohl das Konzept der strukturierten Programmierung in der Welt der Microcomputer noch neu und ungewohnt zu sein scheint, arbeiten mehr und mehr Amateure mit den professionellen Programmiertechniken. Wenn Sie an Ihre ersten unstrukturierten, nicht dokumentierten und hand-assemblierten Maschinencodeprogramme zurückdenken, werden Sie diese Entwicklung auch bei sich selbst feststellen können. Durch klaren Programmaufbau und transparente Arbeitsmethoden entstehen auch gute Programme.

- Aufgabenbeschreibung: In diesem Stadium sollten Sie den Ein- und Ausgaben spezielle Aufmerksamkeit widmen. Hier müssen besonders die Signale der direkt angesprochenen Peripheriegeräte (zum Beispiel Tastatur und Bildschirm) berücksichtigt werden. Dabei können Zeitprobleme auftreten oder auch Routinen fehlen (beispielsweise die Umwandlung der vom Programm eingelesenen ASCII-Zeichen in binär codierte Dezimalzahlen). Achten Sie darauf, daß nicht nur die ankommenden Daten definiert werden, sondern auch das Format, in dem das Programm sie weiterverarbeitet.
- Programmaufbau: Im zweiten Stadium entwirft man die Prozesse, die die Eingaben in Ausgaben umwandeln. Routinen und Daten sollten dabei möglichst in logischen Blöcken gruppiert und als selbständige Module angelegt werden. Es gibt zwei Techniken, Programme zu zerlegen: „vom Teil zum Ganzen" – das Programm wird Stück für Stück aus Einzelmodulen aufgebaut, und „vom Ganzen zum Teil" – hier wird das Programm in immer kleinere Einheiten zerlegt. Das Hauptaugenmerk liegt dabei auf der Funktion und nicht auf der Umsetzung in den Programmcode. Erst nach Definition dieser Funktionsmodule wird mit der Assemblierung begonnen.

Vom Teil zum Ganzen

Bei der Methode „vom Teil zum Ganzen" werden fertige Standardmodule verwendet, die sich leicht zusammensetzen lassen und den Speicher besser ausnutzen. Leider werden die Programme dadurch fehleranfällig, schwer zu testen und unübersichtlich. Die zweite Methode führt zu strukturierten Programmen, die sich leicht modulweise testen lassen. Dabei werden die Programmschritte durch Kurzroutinen voneinander getrennt und einzeln überprüft. Die Kurzroutinen ersetzen fehlende Module. Sie nehmen die Eingaben entgegen und liefern ohne Verarbeitung die korrekten Ausgaben. Nach dieser Methode entwickelte Programme belegen meistens jedoch mehr Speicherplatz. Auch lassen sich ihre Module nur selten in andere Programme einsetzen.

Nun werden für jedes einzelne Modul der Datenbedarf, die Datenstrukturen und die Algorithmen definiert. Hier kann ein Ablaufdiagramm die Aufgabe sehr vereinfachen. Viele Programmierer ziehen allerdings eine Art Hochsprache – den „Pseudo-Code" – vor oder arbeiten mit BASIC. Nachdem Algorithmen und Daten in einer vertrauten Sprache entworfen worden sind, brauchen sie nur noch vom Pseudo-Code in die Assemblersprache übersetzt zu werden – die einzige Aufgabe, bei der Sie es direkt mit dem Maschinencode zu tun haben. Diese Methode ist einfacher als eine gleichzeitige Entwicklung und Codierung in der Assemblersprache.

- Codierung: Bei gut strukturierten Routinen ist dieser Schritt der einfachste und schnellste. Die Codierung überträgt die Steuerstrukturen der Hochsprache auf die Maschinenebene. BRA und JMP sollten nicht wahllos eingesetzt werden, da ein unübersichtlicher Code später die

Fehlersuche sehr erschwert. Unser Diagramm zeigt, wie die bekanntesten Steuermechanismen programmiert werden. Der Einfachheit halber setzen wir voraus, daß alle Daten ein Acht-Bit-Format haben.

Der Code für Steuerstrukturen wird oft länger als notwendig. Wenn ausreichend Platz vorhanden ist, sollten Sie sich jedoch nicht weiter darum kümmern: Ein kürzerer Code führt nicht unbedingt zu kürzeren Ablaufzeiten, verursacht aber oft zusätzlichen Aufwand bei der Programmentwicklung und beim Testen. Bei beschränktem Speicherplatz lohnt es sich allerdings, zunächst den ausführlich strukturierten Code zu schreiben und ihn später zu optimieren. Erst zu diesem Zeitpunkt müssen Sie die Speicherplatzgrenzen berücksichtigen.

- Modultest: In diesem Stadium wird jedes Modul einzeln (wenn nötig mit Kurzroutinen) getestet. Damit ist sichergestellt, daß gültige Eingaben auch gültige Ergebnisse erzeugen. Die Fehlersuche der Assemblersprache unterscheidet sich grundlegend von der Fehlersuche in BASIC. Um die Abläufe überhaupt verfolgen zu können, müssen die vom Programm eingesetzten Speicherstellen und Registerinhalte untersucht und (unter Umständen) verändert werden. Es ist kaum möglich, Assemblerprogramme ohne Hilfsroutinen für das Setzen und Löschen von Unterbrechungspunkten fehlerfrei zu bekommen. Das Programm läuft dabei bis zu einem Haltepunkt, an dem die Hilfsroutine den Registerinhalt anzeigt und die Möglichkeit zur Änderung gibt.
- Test des vollständigen Programms: Sobald die Einzelmodule fehlerfrei sind, wird das Programm zusammengesetzt und mit Testdaten geprüft. Wenn alle Einzelmodule fehlerfrei laufen, ist diese Aufgabe recht einfach.
- Dokumentation: Da Assemblerprogramme komplizierter sind als Programme in Hochsprachen, ist eine ausführliche Dokumentation sehr wichtig. Besondere Aufmerksamkeit sollte dabei dem Speichereinsatz, den Stacks und den von Subroutinen eingesetzten Registern gewidmet werden.

Sorgfalt zahlt sich aus

- Wartung: Programme, die längere Zeit gelaufen sind, müssen oft überarbeitet werden. Dabei werden Fehler beseitigt, die während der Programmentwicklung nicht erkannt wurden, oder Verbesserungen eingebaut. Hier zahlt sich eine sorgfältige Programmierung und ausführliche Dokumentation besonders aus: Wenn keine Struktur vorhanden ist und/oder die Dokumentation fehlt, lassen sich Änderungen unter Umständen nicht mehr einfügen, und das Programm muß völlig neu geschrieben werden. Auf die Weise kann man aber kaum sein Talent in Assembler entfalten.

All diese Entwicklungsstufen zeigen wir am Beispiel eines Monitor/Debugger (Programm für Maschinencodeanzeige und Fehlersuche). Wenn Sie zuvor mit einem Assembler gearbeitet haben, sind Sie mit den Funktionen eines Monitor/Debuggers sicher schon vertraut. Das Programm unterstützt den Assemblerprogrammierer mit Funktionen, die BASIC-Programmierer als selbstverständlich voraussetzen – der Möglichkeit, Speicherstellen untersuchen und verändern zu können.

Im nächsten Artikel führen wir das Projekt durch die ersten in diesem Artikel beschriebenen Entwicklungsstadien.

Steuerstrukturen

Die IF . . . THEN . . . ELSE-Struktur

Pseudo-Code	Assemblersprache

IF NUM1 = 3
THEN
Routine 1
 	
ELSE
Routine 2
ENDIF
	DREI FCB 3
	.....
	IF LDA NUM 1
	CMPA DREI
	BNE ELSE
	THEN .....
	* Routine 1
	BRA ENDIF
	ELSE .....
	* Routine 2
	ENDIF .....

Die WHILE. . . WEND-Struktur

Pseudo-Code	Assemblersprache
WHILE NUM1 < = 3 wiederhole Routine WEND	WHILE LDA NUM1 CMPA DREI BGT WEND * wiederhole Routine BRA WHILE WEND .....

Die REPEAT . . . UNTIL-Struktur

Pseudo-Code	Assemblersprache
REPEAT WIEDERHOLE Routine UNTIL NUM1 < = 3	REPEAT ..... * wiederhole Routine LDA NUM1 CMPA DREI BGE REPEAT UNTIL .....

Die FOR . . . NEXT-Struktur

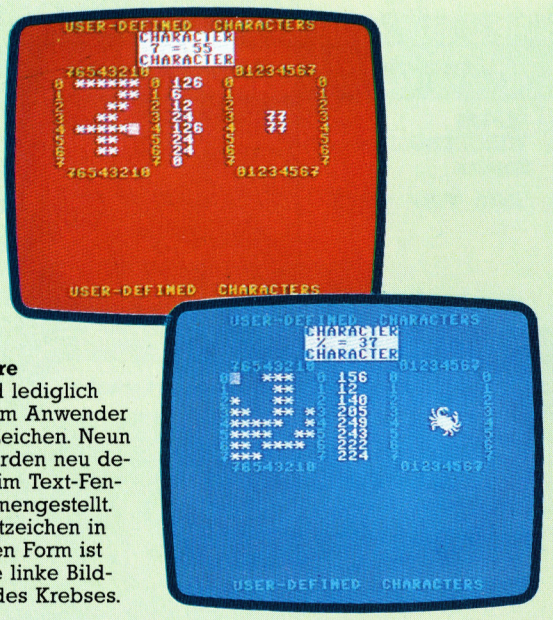
Pseudo-Code	Assemblersprache
FOR NUM1 = 1 TO NUM2 wiederhole Routine NEXT NUM1	LDA NUM2 FOR ..... * wiederhole Routine DECA BGT FOR NEXT .....

Schriftarten

Mit dem nachfolgenden Programm für den Commodore 64 können Sie neue Zeichensätze erzeugen, festdefinierte Zeichen beliebig modifizieren und sogar Sprites übersichtlich gestalten.

Der C 64 bietet hervorragende Sound- und Grafikmöglichkeiten – wie aus der kommerziellen Software zu ersehen ist. Das Commodore BASIC unterstützt jedoch keinen einzigen „zweckgebundenen" Farb- oder Sound-Befehl. Für die Spectrum-Befehle BEEP, DRAW, INK und PAPER zum Beispiel gibt es keine Äquivalente für C-64-Programmierer. Das rechts aufgelistete Programm ermöglicht eine unkomplizierte Generierung neuer Zeichen, die Sie auf dem Bildschirm entwerfen können (anstatt sie direkt ins RAM zu POKEn). Diese Definitionen werden erst anschließend in den Speicher gePOKEt.

Wir haben die Prozedur zur Generierung eigener Zeichen auf dem C 64 schon einmal kurz angesprochen. Die wichtigsten Vorbereitungen werden von der Unterroutine in Zeile 61000 ausgeführt. Die Obergrenze des Anwenderspeichers liegt zwischen Speicherstelle 40959 und 14335. Die gesamten zwei KByte für den Großbuchstaben-Zeichensatz, die beim C 64 im ROM abgelegt sind (ab Adresse 53248 aufwärts), werden dann ins RAM kopiert (ab 14336 aufwärts). Von hier aus kann über PEEK- und POKE-Anweisungen darauf zugegriffen werden. Zuletzt wird der Video Interface Chip (VIC) umgeschaltet, um den neu positionierten Zeichensatz zu adressieren.



Ohne Serifen
Im Edit-Fenster (links) sehen Sie die neu definierte Version des Zeichens 55 – die Zahl 7. Die Serife wurde gelöscht und ein horizontaler Balken eingefügt. Im Text-Fenster (rechts) befinden sich vier Kopien der Sieben. Die Veränderungen sind anhand dieser vier Zahlen sowie anhand der Zahl in der Einrahmung zu erkennen.

Krebsschere
Sprites sind lediglich mehrere vom Anwender definierte Zeichen. Neun Zeichen wurden neu definiert und im Text-Fenster zusammengestellt. Das Prozentzeichen in seiner neuen Form ist der mittlere linke Bildausschnitt des Krebses.

Danach werden durch die Initialisierungsroutine zwei „Fenster" auf dem Schirm dargestellt. Die Programmkontrolle wird an Zeile 2500 (Eingaberoutine) gegeben. Diese Routine fragt die Tastatur ab und zeigt einen blinkenden Cursor im linken (Edit-)Fenster. Das neu definierte Zeichen wird zusammen mit den Werten der acht Definitions-Byte in diesem Fenster vergrößert dargestellt.

Die Funktionstasten f1, f3, f5 und f7 dienen zur Cursorsteuerung im Edit-Fenster. Die Zeile unter dem Cursor kann mit f2 ein- oder abgeschaltet werden. In diesem Fall werden die acht Definitionswerte aktualisiert.

Bequeme Handhabung

Durch Drücken von f4 können Sie das Zeichen im Edit-Fenster durch ein anderes ersetzen. Zeichen werden über ihre POKE- oder Bildschirmcode-Werte angegeben. Diese Werte entsprechen nicht den CHR$-Codes (obwohl es Übereinstimmungen gibt), sind jedoch einfacher zu verwenden, da die Zeichendefinitionen im Speicher in der Reihenfolge dieser Codes abgelegt sind.

Mit f6 können Sie ein editiertes Zeichen in „Zeichengröße" im Text-Fenster (rechts neben dem Edit-Fenster) an der entsprechenden Cursorposition darstellen. Befindet sich der Cursor links oben im Edit-Fenster, während Sie ein „A" editieren, wird durch f6 das A links oben im Textfenster dargestellt.

Mit dem Ausrufezeichen (STOP) wird das Programm angehalten. Durch CONT fahren Sie mit dem Programmlauf fort. Wenn Sie das Programm verlassen (quit oder exit), können Sie nach Eingabe von NEW ein neues Programm laden, ohne den generierten Zeichensatz zu zerstören. Dabei sind allerdings einige Punkte zu beachten. Zuerst wurde die Obergrenze des Anwenderspeichers gesenkt, so daß nur noch zwölf KByte für ein neues Programm zur Verfügung stehen. Außerdem wird der Zeichensatz durch Abschalten des Computers gelöscht. Beide Probleme werden in einem späteren Artikel besprochen.

Schon jetzt aber können Sie unser Programm einsetzen, sei es, um sich einen kyrillischen Zeichensatz für ein Russisch-Vokabelprogramm zu erstellen, oder auch nur, um die Umlaute und das „ß" ansehnlich auf Ihren Bildschirm zu bringen.

Zeichengenerierung

```
19 REM**********C64***************
20 REM*   USER-DEFINED CHAR-GEN   *
21 REM**********C64***************
50 POKE 52,56:POKE 56,56:CLR
60 PRINT CHR$(147)"PLEASE WAIT 22
   SEC'S"
70 GOSUB 61000:  REM COPY CHAR.SET.
100 GOSUB 1000:  REM INITIALISE
120 FOR CT=0 TO 1 STEP 0
140 GOSUB 2500:  REM INPUT
160 GOSUB 3000:  REM VALIDATE
180 ON PT GOSUB 3500,4000,4500,7000
200 NEXT CT
900 END
999 REMM*********************
1000 REM*  INITIALISE         *
1001 REM*********************
1020 DIM BD(8,8),C$(2,2),OF(2,7),
   CX(4)
1040 SH$=CHR$(19):SC$=CHR$(147):R$=
CHR$(18):N$=CHR$(146):CD$=CHR$(17)
1060 P$=CD$+CD$+CD$:P$=P$+P$+P$+P$:
   P$=P$+P$:P$=SH$+P$
1080 C1$=CHR$(144):C2$=CHR$(5)
1200 REM-----INITIALISE SCREEN-----
1210 SO=1024:PRINT SC$C1$
1220 RO=4:CO=3:RL=8:CL=8:OF=16
1230 Z$="USER-DEFINED  CHARACTERS"
1240 C=4:GOSUB2100:R=24:GOSUB2100
1250 L$=" 76543210 ":S$="          "
1260 Z$=L$:R=RO:C=CO:GOSUB 2100
1270 C=CO-1:FOR R=RO+1 TO RO+8
1280 Z$=STR$(R-RO-1):Z$=Z$+S$+Z$
1290 GOSUB2100:C=C+OF:GOSUB2100
1300 C=C-OF:NEXT R
1310 C=CO:Z$=L$:GOSUB2100
1320 L$=" 01234567 "
1330 Z$=L$:R=RO:C=CO+OF:GOSUB 2100
1350 C=CO+OF:R=RO+CL+1 :GOSUB2100
1370 PRINT C2$
1400 C$(1,1)=R$+" "+N$:C$(1,2)=" "
1410 C$(2,1)=R$+"*"+N$:C$(2,2)="*"
1420 REM-----CURSOR    OFFSETS-----
1440 DATA 0,-1,+1,0,-1,0,0,+1
1460 FOR K=1 TO 2:FOR L=1 TO 4:
1480 READ OF(K,L):NEXT L,K
1500 REM-----CHAR  CONVERSIONS-----
1520 DATA 64,0,32,64
1540 FOR K=0 TO 3:READ CX(K):NEXT K
1620 RP=1:CP=1:CN=1:GOSUB 6000
1990 RETURN
1999 REM*********************
2000 REM*  PUT CRSR @ R,C     *
2001 REM*********************
2050 PRINT LEFT$(P$,R+1)TAB(C);
2070 RETURN
2099 REM*********************
2100 REM*  PRINT Z$ @ R,C     *
2101 REM*********************
2150 PRINT LEFT$(P$,R+1)TAB(C)Z$;
2170 RETURN
2499 REM*********************
2500 REM* FLASH CRSR @ RP,CP*
2501 REM*********************
2520 CF=1+BD(RP,CP):R=RP+RO:C=CP+CO
2540 FOR LP=0 TO 1 STEP 0
2560 FOR CS=1 TO 2:DE=10:GOSUB 2800
2580 GET GT$
2600 IF GT$<>"" THEN LP=1:CS=2
2620 Z$=C$(CF,CS):GOSUB 2100
2640 DE=10:GOSUB 2800
2660 NEXT CS,LP:RETURN
2799 REM*********************
2800 REM*   DELAY FOR DE      *
2801 REM*********************
2820 FOR NN=1 TO DE:NEXT:RETURN
2999 REM*********************
3000 REM*  VALIDATE  INPUT    *
3001 REM*********************
3020 IF GT$="!" THEN R=18:C=0:
   GOSUB2000:STOP
3040 GT=ASC(GT$)-132:F=2*INT(GT/2)
3060 IF (GT<1)OR(GT>8) THEN PT=0:
   RETURN
3080 IF GT<5 THEN PT=1:RETURN
3100 PT=GT-3
3490 RETURN
3499 REMM********************
3500 REM*  MOVE THE CURSOR    *
3501 REM*********************
3520 NY=RP+OF(2,GT):NX=CP+OF(1,GT)
3540 IF (NY<1)OR(NY>RL) THEN RETURN
3560 IF (NX<1)OR(NX>CL) THEN RETURN
3580 RP=NY:CP=NX
3620 RETURN
3999 REMM********************
4000 REM*  TOGGLE A CELL      *
4001 REM*********************
4020 TG=1-BD(RP,CP):Z$=C$(1+TG,2)
4040 R=RO+RP:C=CO+CP:GOSUB2100
4060 BD(RP,CP)=TG:MP=NCGEN+CN*8-1
4120 PE=PEEK(MP+RP)
4140 PE=PE+(TG*2-1)*(2^(8-CP))
4160 POKE(MP+RP),PE:GOSUB6500:
   RETURN
4499 REMM********************
4500 REM*  DEFINE NEW CHAR    *
4501 REM*********************
4520 FOR K=1 TO 1
4540 Z$=R$+"   CHANGE  CHAR    "
4550 R=14:C=7:GOSUB2100
4560 Z$=" NEW NUMBER          "
4570 R=15:GOSUB2100
4580 C=19:GOSUB2000:INPUT A$
4600 CN=VAL(A$):PRINT C2$
4620 IF (CN<0)OR(CN>127) THEN K=0
4640 NEXT K:GOSUB 6000
4660 Z$="                     ":C=7
4670 R=14:GOSUB2100:R=15:GOSUB2100
4680 RETURN
5999 REM*********************
6000 REM*    DISPLAY CHAR     *
6001 REM*********************
6020 MP=NCGEN+CN*8-1
6040 FOR RP=1 TO 8:PE=PEEK(MP+RP):
   Z$=""
6060 FOR CP=8 TO 1 STEP-1:
   N=INT(PE/2)
6080 Q=PE-2*N:BD(RP,CP)=Q:PE=N
6100 Z$=C$(Q+1,2)+Z$:NEXT CP
6120 R=RO+RP:C=CO+1:GOSUB2100
6130 NEXT RP
6140 X$=CHR$(CN+CX(INT(CN/32)))
6150 Z$=R$+"CHARACTER":R=1:C=11
6160 GOSUB2100:R=R+2:GOSUB2100
6170 Z$="  "+X$+"  =          "
6180 R=R-1:GOSUB2100
6190 C=C+4:Z$=STR$(CN)+N$:GOSUB2100
6220 RP=1:CP=1
6490 GOSUB6500:RETURN
6499 REM*********************
6500 REM*    DISPLAY BYTES    *
6501 REM*********************
6540 C=CO+CL+2:FOR R=RO+1 TO RO+8
6560 Z$=STR$(PEEK(MP+R-RO))+"   "
6580 GOSUB2100:NEXT:RETURN
6999 REM*********************
7000 REM*   PLACE A CHAR      *
7001 REM*********************
7020 Z$=X$:C=C+OF:GOSUB2100:RETURN
60999 REM********************
61000 REM* RELOCATE CHARGEN   *
61001 REM********************
61100 CGEN=53248:NCGEN=14336
61120 ITRPT=56334:IOPT=1:PO=53272
61125 RETURN
61150 POKE IT,PEEK(IT)AND254
61200 POKE IO,PEEK(IO)AND251
61250 FOR J=0 TO 2047
61300 POKE (NCGEN+J),PEEK(CGEN+J)
61350 NEXT J
61400 POKEIO,PEEK(IO)OR4
61450 POKE IT,PEEK(IT)OR1
61500 POKE PO,(PEEK(PO)AND240)OR14
61990 RETURN
```

Schlüsselwörter

Mit Datenbanksystemen, die Informationen in komplexen Datensätzen speichern, haben wir uns bereits befaßt. Hier geht es um „Schlüssel" erster und zweiter Ordnung zum Abruf von Informationen.

Ein Suchwort ist der Teil eines Datenbanksystems, mit dem sich ein bestimmter Datensatz finden läßt. Stellen sie sich einmal das Reparaturbuch für ein Auto vor. Die Anleitung zum Einstellen des Vergasers kann an einer beliebigen Stelle des Buches stehen. Wäre es beispielsweise auf Seite 36, würde diese Seitennummer der „Schlüssel" zur gewünschten Information. Wenn man nicht weiß, auf welcher Seite die Vergasereinstellung behandelt wird, benutzt man das Inhaltsverzeichnis – es führt einen schnell an die richtige Stelle, ohne daß man das gesamte Buch erst durchlesen muß.

Suchbegriffe

Im Grunde arbeiten alle DBVs ähnlich: Jeder Datensatz in einer Datenbank-Datei hat seine eigene Nummer – den Suchbegriff erster Ordnung. Bei der Suche nach einer bestimmten Information kann man sich nun entweder alle Datensätze nacheinander ansehen oder durch Angeben der Nummer direkt auf die Information zugreifen. Außerdem kann man auch eines der Felder als Schlüssel zweiter Ordnung definieren. Eine Datenbank einer Kfz-Werkstatt hätte etwa das Feld ERSATZTEILBEZEICHNUNG als Schüsselwort.

Die meisten DBVs definieren bestimmte Felder als „Suchfeld". Ist ein Feld – in unserem Fall das Feld „ERSATZTEILBEZEICHNUNG" – als Suchfeld definiert, erstellt die Datenbankverwaltung eine interne Tabelle aus den im Feld vorkommenden Wörtern zusammen mit ihrer jeweiligen Datensatznummer (Schlüssel erster Ordnung). Bei der Suche nach Informationen über Vergaser geht die Datenbankverwaltung die Tabelle ERSATZTEILBEZEICHNUNG durch, bis sie auf den String „Vergaser" stößt, ermittelt die dazugehörige Datensatznummer und gibt den Datensatz aus. In einer einfachen BASIC-Datenbankverwaltung würde der Ablauf so aussehen:

```
INPUT ,EINGABE SUCHFELD';KEYF$
INPUT ,GESUCHTES WORT
   EINGEBEN';WRD$
GOSUB 20000:REM SUCH-UNTER-
   PROGRAMM
PRINT ERGEBNIS
```

Hier wird das mit KEYF$ bezeichnete Array mit einem Unterprogramm nach dem String (WRD$) durchsucht. Ein leistungsfähiges Suchprogramm erlaubt sogar Tippfehler und findet den gewünschten Eintrag auch bei geringen Abweichungen.

In der Vergangenheit waren DBVs für Heimcomputer recht einfach aufgebaut und wirkten eher „handgestrickt". Der Sinclair QL mit seinem leistungsstarken 68000-Prozessor war in diesem Bereich ein wirklicher Sprung nach vorn. Die Firma Psion entwickelte mit VAX-Computern das Datenbanksystem Archive – ein für den QL maßgeschneidertes Programm, das Datenverwaltung auch auf einem Heimcomputer möglich macht. Die Funktion von Archive soll am einfachen Beispiel einer Datenbank für die Kfz-Werkstatt mit nur vier Datensätzen verdeutlicht werden. Jeder Eintrag enthält zwei Felder, TEILBEZEICHNUNG und WARTUNG:

```
Vergaser
Deckel und Flügelmutter entfernen
Öltank
Deckel abnehmen und Öl nachfüllen
Batterie
Stopfen abnehmen und destilliertes Wasser
einfüllen
Scheibenwaschanlage
Verschluß öffnen und Wasser nachgießen
```

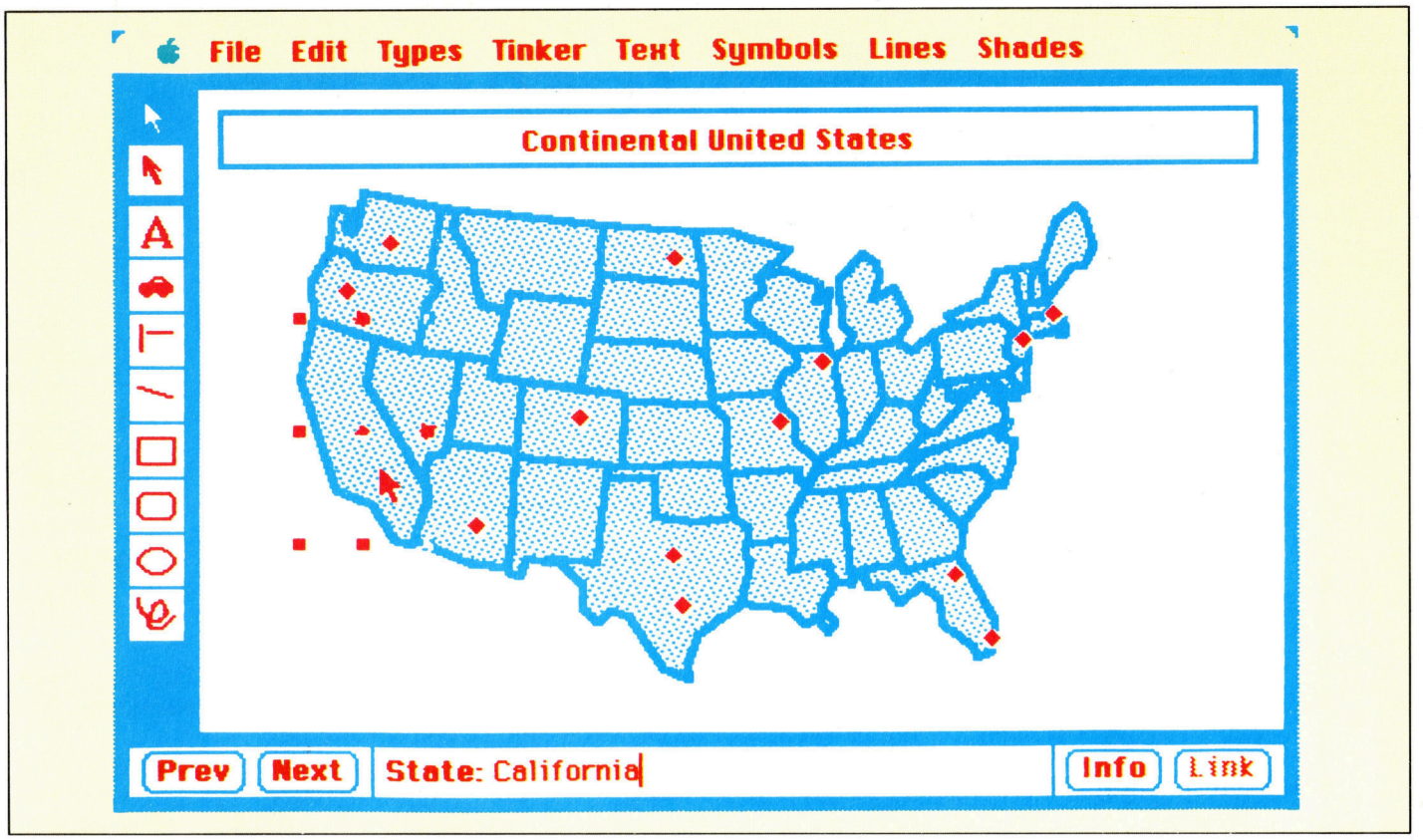
Archive teilt den Bildschirm in drei Bereiche: Der oberste Teil des Bildschirms dient zur Befehlsanzeige, und die Bildmitte ist der Arbeitsbereich. Darstellungen erscheinen unten im Bild. Eine neue Datenbank-Datei wird mit CREATE erzeugt. Unsere Wartungs-Datenbank entsteht durch Eingabe von CREATE ‚AUTO'<CR> (AUTO wird zum Dateinamen). Als nächstes werden die Bezeichnungen der Felder eingegeben:

```
TEILBEZEICHNUNG$<CR>
WARTUNG$
<CR>
```

Das angehängte $-Zeichen bedeutet, daß das dazugehörige Feld einen String beinhalten soll. <CR> beendet den Vorgang.

Mit INSERT<CR> werden Einträge in die Datenbank vorgenommen. Im Arbeitsbereich erscheinen dann die Feldnamen. Die in ein Feld eingegebenen Daten werden mit F5 von der Datenbankdatei übernommen, danach kann mit der Escape-Taste der INSERT-Modus wieder verlassen werden. CLOSE schließt die Datei.

Um eine Datei mit Archive nach einem Datensatz zu durchsuchen, muß zuerst die Datei geöffnet werden. Das geht entweder mit dem Befehl LOOK, der nur das Überprüfen des Daten-



satzes erlaubt, oder mit OPEN – dann können die Einträge nicht nur betrachtet, sondern auch sortiert und verändert werden. Im OPEN- oder LOOK-Modus gibt es folgende Befehle: FIRST zeigt den ersten Eintrag der Datei, LAST den letzten, mit NEXT wird zum nächsten Eintrag übergegangen, und BACK stellt den vorhergehenden Datensatz dar.

Für die Suche nach einem bestimmten Eintrag muß der Befehl mit einem Argument versehen werden, etwa FIND ‚VERGASER'. Daraufhin werden alle Dateien nach dem Begriff „VERGASER" durchsucht. Dieser Vorgang kann durch logische Operatoren optimiert werden : SEARCH TEILBEZEICHNUNG$= ‚VERGASER' AND WARTUNG$= ‚FLÜGELMUTTER' sucht nach einem Eintrag, der beide Begriffe in den angegebenen Feldern aufweist. Auch die OR-Funktion ist zulässig, sie führt aber dann zu einem Datensatz, der entweder den einen oder den anderen String im entsprechenden Feld enthält.

Katalog erstellen

In den meisten Fällen werden die Datensätze in eher zufälliger Reihenfolge eingegeben. Häufig ist es jedoch erforderlich, die Einträge nach bestimmten Kriterien zu sortieren. Angenommen, Sie sollten alle Bücher einer Bibliothek mit Titel, Verfasser, Verlag und ISBN in einer Datenbank erfassen. Das ist am einfachsten, wenn Sie an den Bücherregalen vorbeigehen und die Daten so eingeben, wie die Bücher gerade stehen. Das alphabetische Sortieren der Einträge nach Verfasser, Titel usw. besorgt dann später die DBV. Sogar eine numerische Anordnung der Bücher nach ihrer ISBN kann damit erfolgen.

Sie können auch „zweifach" sortieren: Angenommen, eine Datei soll in der alphabetischen Ordnung der Autoren ausgedruckt werden, wobei die Werke der einzelnen Verfasser ebenfalls alphabetisch geordnet auftauchen sollen. Der Ausdruck müßte etwa so aussehen:

ADELMANN, G.H.
Aufzucht von Jagdhunden als Beruf und Hobby
Geier Verlagsgesellschaft
0-85527-435-2
ADELMANN, G.H.
Hunde — Freunde des Menschen
St. Hubertus Verlag
0-7195-1332-4
ADELMANN, G.H.
Schutz- und Lawinenhunde
Geier Verlagsgesellschaft
0—85527-438-6
ADELMANN, M
Schrift und Sprache Altgriechenlands
Verlag der Universität Wien
226-52719-0

Das Sortieren der Einträge nach Autoren und dann nach Titeln erfordert beim Archive-Programm nur die Eingabe von:

ORDER AUTOR$;A,TITEL$;A

Das A in diesem Beispiel steht für „in alphabetischer Reihenfolge ordnen".

Schon diese einfachen Beispiele zeigen, daß Datenbanksysteme zeitraubende Aufgaben wie das Suchen spezieller Einträge oder Sortiervorgänge abkürzen können.

Das Programm Filevision wurde ursprünglich für die Computer der Apple-II-Serie entwickelt. Erst der leistungsfähigere Apple Macintosh brachte an den Tag, was wirklich in diesem Programm steckt. Das Programm ist mausgesteuert – der Anwender muß bei der Abfrage der Datenbank kaum noch Tastatureingaben machen. Ein ausgefeiltes Zeichenprogramm erleichtert die Konstruktion des grafischen Rahmens für die Darstellung der Informationen.

Die Abbildung oben zeigt den Bildschirm einer Datenbank mit Informationen über die USA. Wenn man die Maus auf Kalifornien setzt, reicht ein simpler Tastendruck zum Aufrufen einer kurzen Zusammenfassung von Daten über diesen Bundesstaat. Ein doppelter Tastendruck erweitert die Menge der dargestellten Informationen. Auch das Zusammenfügen und Sortieren der Daten läßt sich schnell und einfach bewerkstelligen.

Auf Fehlersuche

Als Programmbeispiel für die Technik „vom Ganzen zum Teil" konstruieren wir in Assembler einen „Debugger". Dabei entwickeln wir zunächst das Hauptmodul, das die einzelnen Aufgaben der Routinen in den niedrigeren Ebenen steuert.

Zunächst ein kurzer Blick auf die Aufgaben des Debuggers und seine Entwicklungsstufen. Das Programm soll folgende Eingaben empfangen können:

1) **Das zu bearbeitende Programm:** Wir gehen davon aus, daß das Programm sich beim Laden des Debuggers schon im Speicher befindet.

2) **Befehle:** Wir müssen entscheiden, welche Befehle direkt eingegeben und welche über ein Menü angeboten werden sollen. Wir nehmen dafür die Ein-Zeichen-Befehle des nebenstehenden Diagramms.

3) **Adressen:** Da die Eingabe in Hex erfolgt, muß der Hexadezimalstring im ASCII-Format in eine 16-Bit-Binärzahl umgewandelt werden.

Folgende Ausgaben werden erzeugt:

1) **Die „Echos" der eingegebenen Zeichen:** Bedenken Sie, daß das Drücken einer Taste nicht automatisch Zeichen auf den Schirm bringt – dafür ist ein Programm nötig.

2) **Zahlen im Acht- und 16-Bit-Format,** die als Hexadezimalstrings ausgegeben werden.

3) **Strings,** die als Labels für die unter Punkt 2 aufgeführten Zahlen dienen.

Ein Programm kann auf verschiedene Arten in Module und dann in Subroutinen zerlegt werden. Es muß jedoch ein Rahmenmodul vorhanden sein, das alle anderen Routinen aufruft.

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

Daten:
  Anfangsadresse des Programms (16 Bit)
  Prompt – für Befehlseingaben (das ASCII-Zeichen '>')
  Befehlszeichen – ein einzelnes ASCII-Zeichen (sollen hier auch Kleinbuchstaben möglich sein?)
  Unterbrechungsadresse – die Adresse der Routine, die den SWI-Interrupt handhabt.
Ablauf:
  Interrupt anlegen
  Anfangsandresse holen
  REPEAT
    Prompt anzeigen
    REPEAT
      Befehl holen
    UNTIL Befehl gültig ist
    Befehl darstellen (Echo)
    IF Befehl = 'B' THEN
      Unterbrechungspunkt einfügen
    ELSE IF Befehl = 'U' THEN
      Unterbrechungspunkt löschen
    ELSE IF . . .
  UNTIL Befehl = 'Q'
Ende des Hauptmoduls

Mehrere Subroutinen, die eine logische Einheit bilden, werden als Modul bezeichnet. So kann beispielsweise ein Modul das Einsetzen der Unterbrechungspunkte ausführen. Das nächste Stadium zeigt den Aufbau eines Moduls:

Daten:
Unterbrechungstabelle – ein Array von 16-Bit-Adressen, an denen die Unterbrechungsadressen gespeichert werden.
Entfernte-Werte – ein Array von Acht-Bit-Werten, die mit der obigen Tabelle zusammenhängt. Hier werden die durch SWI-Befehle ersetzten Unterbrechungspunkte gespeichert.
Zahl-der-Unterbrechungspunkte – ein Acht-Bit-Wert, in dem die Zahl der aktiven Unterbrechungspunkte gespeichert ist.
Nächster-Unterbrechungspunkt – ein Acht-Bit-Wert, der den nächsten Unterbrechungspunkt des Programms angibt.
SWI-Op-Code – ein Acht-Bit-Op-Code für den SWI-Befehl
Ablauf1: Unterbrechungspunkt-einfügen
IF Zahl-der-Unterbrechungspunkte < MAX
  THEN Nächster-Unterbrechungspkt. holen
  1 auf die Zahl-der-Unterbrechungspunkte addieren
  Adresse in der Unterbrechungstabelle speichern (Zahl-der-Unterbrechungspunkte)
ENDIF
Ende von Ablauf1
Ablauf 2: Unterbrechungspunkt(N)-anlegen
(N enthält die Information, welcher der Unterbrechungspunkte der Tabelle angelegt wird)
Aus der Unterbrechungstabelle(N)
Nächster-Unterbrechungspunkt holen
Den Op-Code dieser Adresse holen
In Entfernte-Werte(N) speichern
Für den Op-Code SWI-Adresse einsetzen
Ende von Ablauf 2

Ablauf 2 könnte jetzt schon codiert werden. Er hat folgende vier Datenwerte: Den Parameter N – eine Acht-Bit-Zahl von (1 bis Anzahl-der-Unterbrechungspunkte-1), die angibt, welcher Unterbrechungspunkt eingesetzt werden soll und als Offset für beide Tabellen dient. Beachten Sie, daß die eine Tabelle 16-Bit-Werte enthält und die andere Acht-Bit-Werte. Wir gehen

davon aus, daß N in A übergeben wird und die Adresse des Unterbrechungspunktes in X. Der ersetzte Op-Code wird zunächst in B zwischengespeichert und dann in die Tabelle Entfernte-Werte eingetragen.

Im nebenstehenden Kasten haben wir den Code für Ablauf2 aufgeführt (Modul „Unterbrechungspunkt-anlegen"). Wir müssen nun ein Modul für die Ein- und Ausgaben entwickeln. Wir gehen erst einmal davon aus, daß dafür zwei Subroutinen zur Verfügung stehen: INCH setzt ein einzelnes Zeichen von der Tastatur in das A-Register und OUTCH sendet das in A gespeicherte Zeichen an die aktuelle Cursorposition. Das Modul benötigt folgende Subroutinen:

1) **BefehlHolen:** Den nächsten Befehl von der Tastatur einlesen.
2) **AdresseHolen:** Eine Hex-Adresse von der Tastatur einlesen (ein bis vier Zeichen).
3) **WertHolen:** Einen Hex-Wert holen, der den Wert der Speicherstelle verändert (ein oder zwei Zeichen).
4) **WertAnzeigen:** Einen Hex-Wert aus zwei Zeichen auf dem Bildschirm anzeigen.
5) **AdresseAnzeigen:** Eine Hex-Adresse aus vier Zeichen auf dem Bildschirm anzeigen.

Hilfreiche Routinen

Bei unserem Verfahren sind die Unterschiede zwischen den Methoden „vom Ganzen zum Teil" und „vom Teil zum Ganzen" zu erkennen. Bei „Vom Ganzen zum Teil" definieren und codieren wir Vorgänge unabhängig voneinander und erhalten mehrere Routinen, die ähnliche Aufgaben ausführen. Die Methode „vom Teil zum Ganzen" spart jedoch Zeit, Aufwand und Platz, da wir nun einige wenige Routinen schreiben, die sich für mehrere Aufgaben einsetzen lassen. Hier die Definition dieser Routinen:

GETCH: Bei Eingabe eines einzelnen Zeichens in A das Zeichen mit der Liste der gültigen Zeichen vergleichen, gültige Zeichen (Befehl- oder Hexzeichen) darstellen und den Rest ignorieren.
GETHX2: Mit GETCH zwei Hex-Ziffern holen und in eine Acht-Bit-Zahl umwandeln.
GETHX4: Vier Hex-Ziffern für eine 16-Bit-Zahl holen.
PUTHEX: Eine Acht-Bit-Zahl als zwei Hex-Ziffern darstellen. (Kann für die Darstellung einer 16-Bit-Zahl zweimal aufgerufen werden.)
PUTCR: Ein Return (CR) ausgeben (oder – falls nötig – ein Return mit Zeilenvorschub).
Wir werden diese fünf Routinen nacheinander entwickeln. Hier der Aufbau von GETCH:

Daten:
Zeichen-ein ist ein per Tastatur eingegebenes ASCII-Zeichen (in A gespeichert)
Zeichen-OK enthält die 16-Bit-Adresse der Tabelle für gültige Zeichen
Zahl-der-Zeichen-OK ist ein Acht-Bit-Wert
Zeichen-gesucht ist ein Acht-Bit-Zähler

Ablauf:

```
REPEAT
   Nächstes Zeichen-ein holen
   Zeichen-gesucht auf (Zahl-der-Zeichen-
      OK -1) setzen
   WHILE Zeichen-OK(Zeichen-gesucht)
      <> Zeichen-ein
      AND Zeichen-gesucht >= 0
      Zeichen-gesucht dekrementieren
UNTIL Zeichen-gesucht >= 0
Zeichen-ein anzeigen
```

Für die Codierung dieses Ablaufs speichern wir Zeichen-Ein in A und die 16-Bit-Zahl Zeichen-OK in X. Zahl-der-Zeichen-OK kann zunächst an B übergeben werden, muß aber auf den Stack geschoben werden, damit sie nicht gelöscht wird. B läßt sich dann für Zeichen-gesucht einsetzen. Beachten Sie, daß B den Offset an die Tabelle übergibt und sich damit auch für die Identifizierung der Befehle und die Hexadezimalumwandlungen eignet.

Die GETCH-Routine

```
GETCH   PSHS  B       B speichern
REPTOO  BSR   INCH    Nächstes Zeichen-ein holen
        LDB   1,S     Zeichen-gesucht setzen
        DECB          Eins vom max. Offset subtrahieren
WHILOO  BLT   ENDWOO  WHILE Zeichen-gesucht >=0
        CMPA  B,X     AND
        BEQ   ENDWOO  Zeichen-ein <> Zeichen-OK
                      (Zeichen-gesucht)
        DECB          Zeichen-gesucht dekrementieren
        BRA   WHILOO
ENDWOO  TSTB
UNTLOO  BLT   REPTOO  UNTIL Zeichen-gesucht >=0
        BSR   OUTCH   Zeichen-ein anzeigen
        LEAS  1,S     S inkrementieren, um den
                      ursprünglichen Wert von B zu
                      löschen
        RTS
```

Das Modul „Unterbrechungspunkt-anlegen"

```
                      Definition der Daten
BPTAB   RMB   32      Tabelle der Unterbrechungspunkte
REMTAB  RMB   16      Entfernte Werte
NUMBP   FCB   0       Zahl-der-Unterbrechungspunkte
NEXTBP  FCB   0       Nächster-Unterbrechungspunkt
SWIOP   FCB   $3F     SWI-Op-Code
MAXBP   FCB   16      Max. Unterbrechungspunkte
                      Ablauf2 – Unterbrechungspunkt anlegen
BP02    PSHS  B,X     Zu ändernde Register sichern,
        LSLA          Den Offset mit 2 multiplizieren
        LEAX  BPTAB,PCR  Basisadresse der Tabelle
        LDX   A,X     Adresse (N) holen
        LDB   ,X      Op-Code dieser Adresse holen
        LSRA          A auf Urwert zurückstellen
        STB   A,X     Op-Code speichern
        LDB   SWIOP,PCR  SWI-Op-Code holen
        STB   ,X      In dieser Adresse speichern
        PULS  B,X,PC  Werte wiederherstellen/Return
                      Ende von Ablauf2
```

Computer-Lehrer

Diese Artikelserie beschäftigt sich mit den Erfahrungen, die speziell an britischen Lehranstalten bei Einsatz von Computern im Unterricht gemacht wurden. In der BRD dagegen gibt es noch wenig Anwendungen.

Im Klassenzimmer von heute kann ein Computer (mit entsprechender Software) Tafel, Overhead-Projektor, Fernseher, Cassettengerät und Bibliothek leicht ersetzen. Wo direkter Ersatz nicht möglich ist – zum Bespiel bei malenden Erstklässlern – können Computer herkömmliche Lehrmethoden sinnvoll unterstützen. Wichtiger noch: Der Lehrer wird von zeitaufwendigen Aufgaben befreit (zum Beispiel dem Korrigieren von Arbeiten) und hat Zeit, sich interaktiv und individuell mit den Schülern zu befassen.

Langfristig gesehen rechnen wir damit, daß Computer-Informationssysteme einen mindestens ebensogroßen Einfluß auf die Ausbildung haben werden wie die Entwicklung des gedruckten Buchs." Dieser Satz aus einem Bericht der englischen Reierung im Jahre 1978 wurde unlängst von Professor Tom Stonier, einer Autorität der Informationstechnik, wieder aufgegriffen. Er und andere, darunter auch der amerikanische Pädagoge Seymour Papert, glauben, daß Schulsysteme in der bisher bekannten Form noch in dieser Generation verschwinden werden.

Diese radikal anmutende Ansicht ist eine logische Konsequenz aus der engen Verbindung der Wissenschaftler mit dem computergestützten Lernen. Tatsächlich aber vollzieht sich die Veränderung nur schleppend. Dennoch gibt es viele Fachleute, sowohl Pädagogen als auch Außenstehende, die der Überzeugung sind, daß – richtiger Einsatz vorausgesetzt – Computer einen entscheidenden Einfluß auf den Lernprozeß haben werden.

Der Einsatz von Computern im Klassenzimmer läßt sich anhand verschiedener Ausbildungsprojekte etwa 20 Jahre zurückverfolgen. Die zugrunde gelegten Prinzipien bestehen allerdings schon länger – zumindest an englischen Schulen. Schon 1920 wurden einfache „Lernmaschinen" gebaut. Bekannt wurden sie in den fünfziger Jahren, als B. F. Skinner eine einfache Lernmaschine mit Rückkopplungseffekt entwickelte. Skinner, einer der einflußreichsten Lehrer und Psychologen in den 50er und 60er Jahren, erarbeitete seine Ideen in einem 20-Jahres-Zeitraum, basierend auf Expe-

rimenten mit Ratten und Tauben. Er war Fürsprecher des Systems der „Linear-Programmierung" (was nichts mit dem Programmieren eines Computers zu tun hatte). Dabei wurden den Schülern Informationen immer nur in kleinen Einheiten gegeben, und richtige Antworten wurden gelobt.

Aus der Linear-Programmierung entwickelten sich zwei wichtige Konzepte: „Rückkopplung" und „Individualisierung". Rückkopplung bedeutet in diesem Fall, daß dem Schüler gesagt wird, ob er richtig oder falsch geantwortet hat. Unglückseligerweise konnte Skinners mit Hebeln und Knöpfen ausgestattete Lernmaschine nur bestimmte Antworten „lesen". Individualisierung bezeichnet die direkte Art der Methodik. Dem Schüler wird ermöglicht, entsprechend seinen eigenen Fähigkeiten vorzugehen, und seine individuelle Lerngeschwindigkeit wird nicht vom Lehrer oder von Mitschülern beeinflußt.

Lineare Programmierung

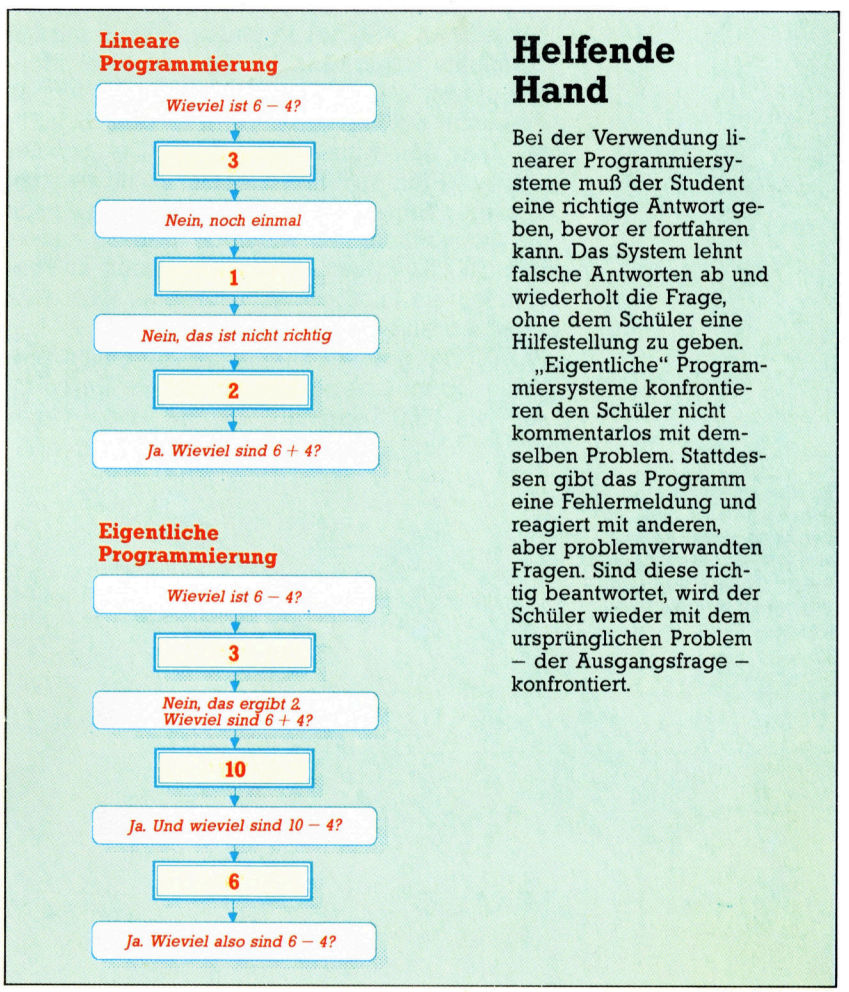
Als Skinners Maschinen eine sehr große Popularität erreicht hatten, gelangten Computer auf den Markt. Lineare Programmiersysteme wurden bald schon auf die Rechner übertragen. Trotz der Einschränkungen dieser Methode haben wir es recht häufig mit Programmen zu tun, die sich ihrer bedienen. „Tut mir leid, versuch's noch einmal" oder „Gut, fein gemacht" erscheint auf dem Bildschirm als ermunternde Antwort, damit der Schüler trotz eines ermüdenden Stroms von Fragen weiterarbeitet.

Eine Entwicklung des Linear-Programmierens war das „Verzweigungs-Programm". Statt der Möglichkeit, bei einer falschen Antwort noch eine Antwort zu geben, erhielt der Schüler zunächst eine Hilfestellung und wurde erst dann neuerlich gefragt. Diese Vorgehensweise scheint logisch. Dieses sogenannte „eigentliche Programmieren" beunruhigte Skinner und seine Anhänger. Für den Schüler aber erwies sich die intensivere Rückkopplung als besser. Man entwickelte „Autoren-Sprachen", die dem Lehrer dabei helfen sollten, diese Art Programme zu schreiben. Diese ersten Versuche von Anwenderfreundlichkeit sollten den Lehrer befähigen, Ausbildungsprogramme zu schreiben, ohne daß er zuvor hätte Computer-Programmierung lernen müssen.

Lineare wie „eigentliche" Programmierung weisen entscheidende Nachteile auf. Der Schüler wird wie ein „leeres Gefäß" behandelt, das mit Wissen zu füllen ist. Lernen wird mehr als Faktenaufnahme denn als Sammeln von Erfahrungen betrachtet. Für Selbstverwirklichung, Vorstellungsvermögen und Kreativität bleibt kein Raum. Lediglich eine einfache „richtige" Antwort wird verlangt. Diese Systeme, denen wir den Begriff des „programmierten Lernens" zu verdanken haben, hatten zur Folge, daß computergestütztes Lernen generell einen schlechten Ruf bekam.

Trotzdem erkannte man bald die neuen Möglichkeiten, die Computer bei der Ausbildung boten. Die seinerzeit entwickelten Arbeiten erwiesen sich als nachhaltig und wichtig – nicht nur für Lehrer. 1965 regte das amerikanische Dartmouth College an, eine Lehrsprache zu entwickeln, die „dem Englischen so nahe wie möglich" sein sollte. Sie sollte Wissenschaftler, Ingenieure und Studenten befähigen, ihre eigenen Programme zu schreiben. Aufgrund der damaligen Einschränkungen, als eine 16-KByte-Maschine noch 160 000 Mark kostete und die Ausmaße eines Wohnzimmers hatte, mußte die Sprache so entwickelt werden, daß sie den kleinstmöglichen Speicherplatz belegte. Das Ergebnis war der „Beginner's All-purpose Symbolic Instruction Code" – oder kurz BASIC – mit dem wir bis heute arbeiten.

Einige Jahre nach der Entwicklung von BASIC erarbeitete ein Team von Wissenschaftlern und Pädagogen am MIT eine Programmiersprache, die „einfacher für den Programmierer als für den Computer" war. Das Ziel: Sehr kleine Kinder sollten einen Computer steuern können. Diese Idee war revolutionär, da alle vorhergehenden Projekte auf Studenten abgestellt waren. Leiter des Projekts war Seymour Papert, der mit dem Kinder-Psychologen Jean Piaget studiert hatte. Piagets Vorstellungen über Erziehung unterschieden sich völlig von denen Skin-



Helfende Hand

Bei der Verwendung linearer Programmiersysteme muß der Student eine richtige Antwort geben, bevor er fortfahren kann. Das System lehnt falsche Antworten ab und wiederholt die Frage, ohne dem Schüler eine Hilfestellung zu geben.

„Eigentliche" Programmiersysteme konfrontieren den Schüler nicht kommentarlos mit demselben Problem. Stattdessen gibt das Programm eine Fehlermeldung und reagiert mit anderen, aber problemverwandten Fragen. Sind diese richtig beantwortet, wird der Schüler wieder mit dem ursprünglichen Problem – der Ausgangsfrage – konfrontiert.

ners. Im Gegensatz zur mechanischen Vorgehensweise Skinners glaubte Papert an die Schaffung einer Umgebung, in der Kinder durch Entdecken lernen könnten. Sein Vorschlag: „Nicht der Computer soll das Kind programmieren, sondern das Kind den Computer." Ergebnis dieser Arbeit war LOGO, jene Sprache, die auch für Microcomputer zur Verfügung steht und Einzug in Schulen gehalten hat.

Trotz der Energie, die auf die Entwicklung computergestützter Lerntechniken verwendet wurde, gab es viele Leute, die die Vorteile dieser Techniken, namentlich die Verwendung von Computern bei der Ausbildung, infrage stellten. Um diese Unsicherheit zu klären, investierte die amerikanische „National Science Foundation" zehn Millionen Dollar für zwei fünfjährige Projekte – TICCIT und PLATO.

TICCIT und PLATO

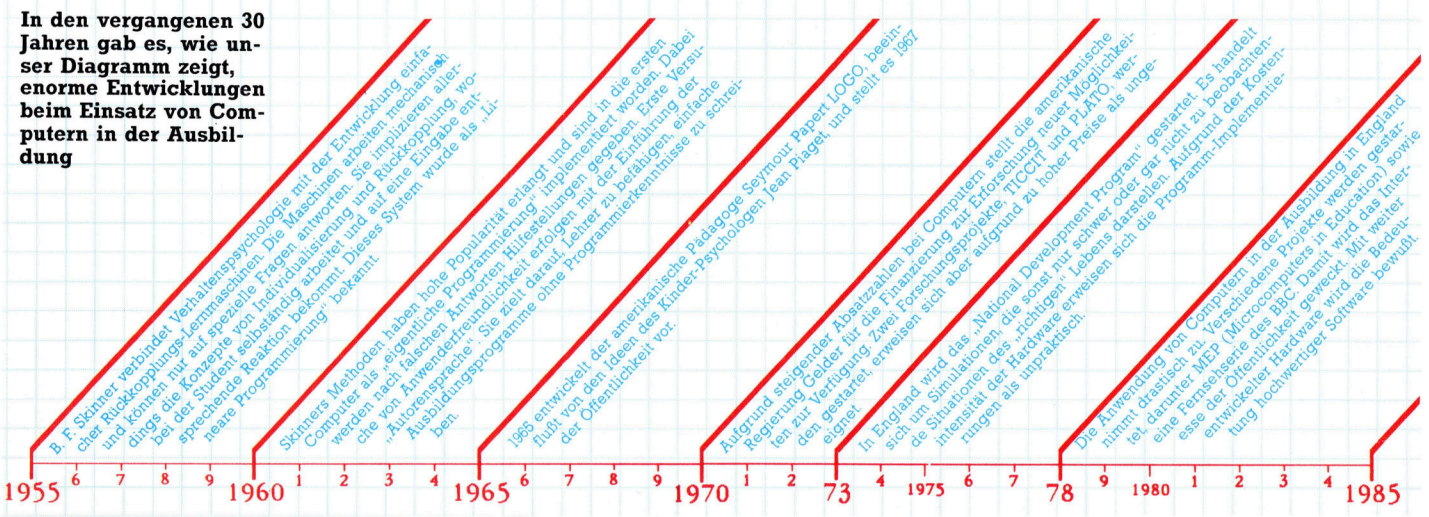
Ziel von TICCIT (Time-shared Interactive Computer Controlled Information Television) war es, „bessere Ausbildung zu niedrigeren Kosten als bei der traditionellen Ausbildung an Grundschulen" zu bekommen. Ein Unternehmen wurde mit der Herstellung von Hard- und Software beauftragt und entwickelte eine spezielle Tastatur für Schüler. Ferner gehörten dazu ein Bildschirm für die Wiedergabe von Text, Grafik und Videos sowie ein Lautsprecher für Anweisungen. All dies war in einem System mit 128 Terminals untergebracht, die von zwei Minicomputern gesteuert wurden. Esgab jedoch Probleme mit der Software, und viele Schüler brachen den Kurs vorzeitig ab. Die Schüler aber, die ihn zu Ende führten, schlossen mit besseren Ergebnissen ab, als Schüler, die nach der konventionellen Methode gelernt hatten. Das TICCIT-System wird heute noch in zwei Schulen als Pilot-Projekt betrieben, kam aber sonst nirgendwo zur Anwendung.

PLATO (Programmed Logic for Automatic Teaching Operation) besteht aus einem Netzwerk von fast 1000 Terminals, die mit einem Großrechner in Illinois verbunden sind. Die Terminals können untereinander kommunizieren und haben Zugriff zu einer im Großrechner vorhandenen Bibliothek. Die PLATO-Darstellungs-Einheiten sind mit transparenten Feldern ausgestattet, mit denen Dias über Computergrafiken projiziert werden können. Eine „Maltafel" erlaubt den Schülern, durch Berührung des Bildschirms mit dem Programm zu kommunizieren. Lehrer können mit einer TUTOR genannten Sprache ihre eigenen Programme schreiben. Wie bei TICCIT erforderte die Software-Vorbereitung mehr Zeit und Aufwand als vorhergesehen. Die Ergebnisse der Schüler, die mit PLATO arbeiteten, waren weder besser noch schlechter als die von Schülern, die nach herkömmlicher Methode unterrichtet wurden. Allerdings empfanden die mit PLATO arbeitenden Schüler das System als benutzerfreundlicher im Vergleich zu TICCIT.

1973 stellte die britische Regierung zehn Millionen Mark für ein Fünf-Jahres-Programm (benannt National Development Program) für computergestütztes Lernen zur Verfügung. Der Löwenanteil der Summe wurde für die Entwicklung von Software ausgegeben, um verschiedene Simulationen zu erstellen. Ein Programm beispielsweise simulierte die Konzentration eines Farbstoffs im Blutkreislauf in unterschiedlichen Zeitintervallen. Andere verdeutlichten chemische Reaktionen. Aufgrund der hohen Kosten für Hardware in den siebziger Jahren erwiesen sich die Programme jedoch als nicht einsetzbar.

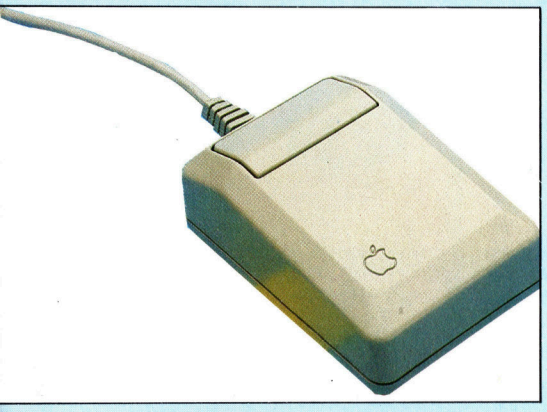
Jedes Lernsystem wird durch Politiker, die Wirtschaft, sowie durch technische Veränderungen gesellschaftlicher Gegebenheiten beeinflußt. Daß die Ausbildung mit dem Computer dennoch heute auf dem Vormarsch ist, beruht auf den ständig fallenden Preisen für Speicherchips. In einer einzigen Londoner Grundschule steht heute mehr Computerkapazität zur Verfügung als an sämtlichen britischen Universitäten vor zwanzig Jahren. Von „Computernutzung" an deutschen Schulen kann keine Rede sein.

In den vergangenen 30 Jahren gab es, wie unser Diagramm zeigt, enorme Entwicklungen beim Einsatz von Computern in der Ausbildung



Fachwörter von A bis Z

Input Device = Eingabegerät
Über Eingabegeräte wird der Rechner in Form digitaler elektrischer Signale von außen mit Information versorgt. Meist dient dazu eine Tastatur, obwohl andere Verfahren wie Spracheingabe oder Maussteuerung, auch Lichtgriffel, Grafiktabletts und natürlich Joysticks eingesetzt werden.



Die Maus ist einer von vielen Versuchen, die Eingabe benutzerfreundlicher zu gestalten. Damit soll vor allem Rechnerneulingen, die mit der Schreibmaschinentastatur und der EDV nicht vertraut sind, der Umgang mit dem Rechner erleichtert werden.

Input/Output = Ein/Ausgabe
Die Ein/Ausgabe (E/A-System) eines Computers ist für den Datenaustausch zwischen der CPU und den anderen Rechnerkomponenten zuständig. Ohne die Ein/Ausgabe hätte die CPU keine Verbindung mit der „Außenwelt". Erst über das E/t-System kann der Benutzer mit dem Prozessor kommunizieren.

Die vorrangige Aufgabe des E/A-Systems besteht darin, die von außen eintreffenden Signale in eine für die CPU verständliche Form umzusetzen. Außerdem kontrolliert das System, welches der Peripheriegeräte gerade aktiv ist und ob Daten ein- oder ausgegeben werden. Dies alles bindet eine erhebliche Prozessorkapazität, insbesondere bei der Datenausgabe über Bildschirmgeräte, deren Anzeige ja ständig aktualisiert werden muß. Deshalb werden diese Aufgaben bei den meisten Rechnern speziellen Ein/Ausgabe-Chips übertragen.

Ein vorrangiges Betätigungsfeld für das E/A-System ist auch die Kommunikation mit peripheren Massenspeichern wie Cassetten und Diskettenlaufwerken.

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.

Instruction = Befehl
Ein Befehl ist eine Handlungsanweisung für den Rechner in Form einer spezifischen Folge von Zeichen bzw. Bits. Der Befehl besteht aus dem Operationsteil (Op-Code), der die Art der Aktivität beschreibt, und dem Operanden- oder Adreßteil, der angibt, womit die Operation durchzuführen ist. In dem Befehl „LDY 8" zum Beispiel ist LDY der Op-Code und die Ziffer 8 der Operand.

Drei Gruppen von Befehlen sind zu unterscheiden: die arithmetischen Befehle wie Addition, Subtraktion, Multiplikation und Division, die logischen Befehle wie die Verknüpfungen AND/OR und die Ein/Ausgabebefehle für den Datentransfer, etwa das Laden von Registern.

Instruction Counter = Befehlszähler
Der Befehlszähler (manchmal auch „Programmzähler" oder „Befehlsadreßregister") ist ein CPU-Register, das die Speicheradresse des nächsten auszuführenden Befehls enthält. Beim Programmablauf wird der Befehlszähler schrittweise weitergeschaltet, so daß er stets den aktuellen Stand der Bearbeitung widerspiegelt.

Stößt der Rechner auf eine Verzweigung oder einen Sprungbefehl, wird der Inhalt des Befehlszählers der Anweisung entsprechend um einen festgelegten Wert erhöht oder erniedrigt. Ein Verzweigungsbefehl bewirkt, daß der aktuelle Befehlszählerstand in den Stack geschoben und die entsprechende Unterprogrammadresse geladen wird. Nach Abarbeiten der Routine setzt die CPU die Rückkehradresse vom Stack wieder in den Befehlszähler und fährt mit dem Hauptprogramm fort. Diese Vorgänge werden automatisch durchgeführt, ohne daß sich der Programmierer mit dem unteren Abläufen befassen muß.

Instruction Set = Befehlssatz
Den Vorrat an Befehlen, die ein bestimmter Prozessortyp verstehen und ausführen kann, bezeichnet man als Befehlssatz. Daraus werden alle Kommandos, zum Beispiel die BASIC-Befehle, aufgebaut. Der Befehlssatz ist prozessorintern in einem ROM festgeschrieben, Umfang und Leistungsfähigkeit sind eine Frage der CPU-Architektur. Zu dem Befehlssatz gehören unter anderem die Handhabung der Register- und Akkumulator-Inhalte, die Realisierung bedingter und unbedingter Sprünge und Verzweigungen sowie die Möglichkeit zum Abfragen und Rücksetzen der Statusregister-Flags.

Integer = Ganze Zahl
Eine Integer-Konstante oder -Variable ist immer ganzzahlig. Sie darf positiv, negativ oder Null sein. Die Zahlen 56, −43 und 0 sind Integers, 3,8 oder −2,001 oder 3 1/2 nicht.

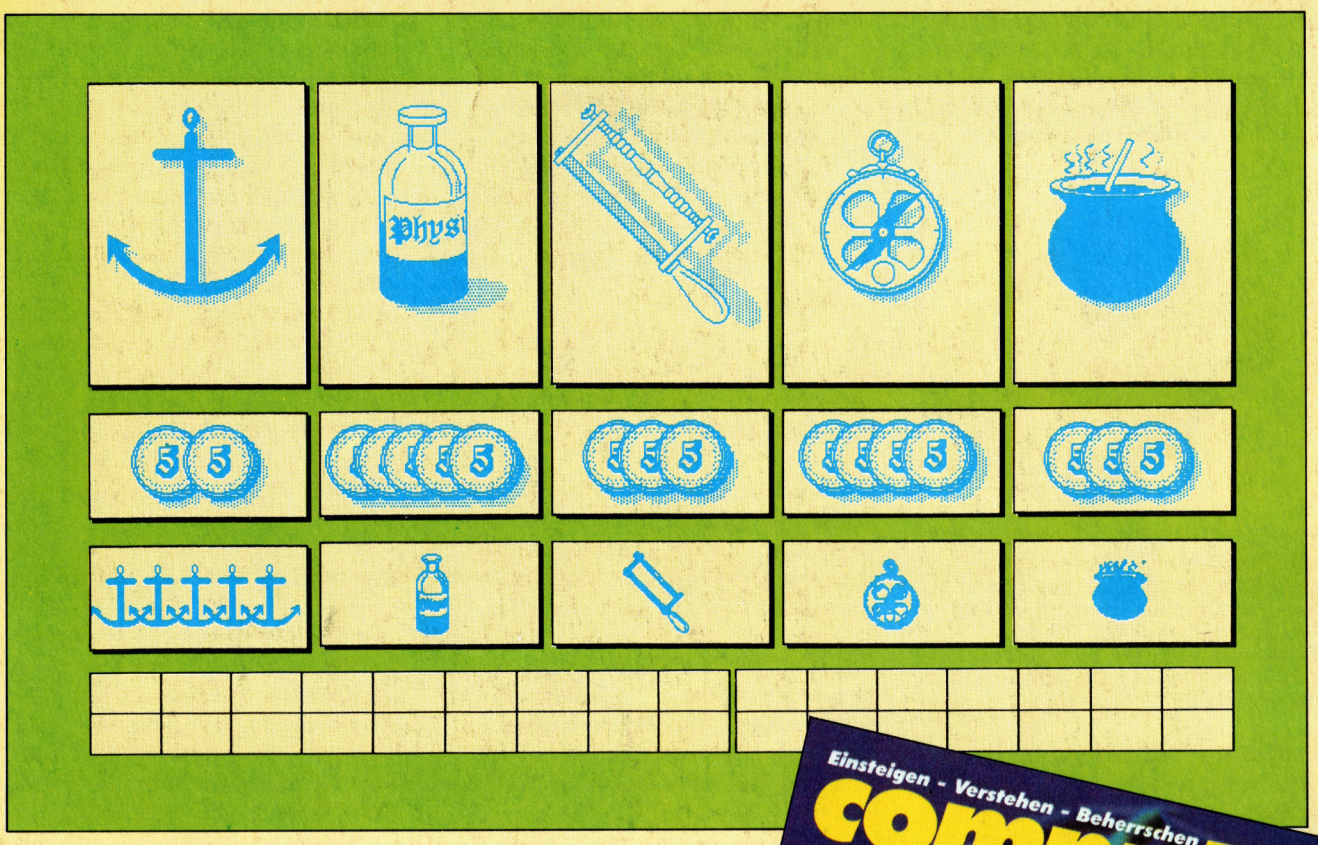
Das Byte 1000 0001 beispielsweise kann folgende Zahlen darstellen:

Bitmuster	Interpretation	Dezimalwert
1000 0001	Integer o. Vorzeichenbit	129
1000 0001	Integer m. Vorzeichenbit	−1
1000 0001	Zweierkomplement-Integer	−127

Bildnachweis
1513: Gabrielle Izen, Holger Neuhaus
1514: Caroline Clayton
1515: David Davies
1516, 1517: Liz Heaney
1521, 1532, 1540: Ian McKinnell
1530: Kevin Jones
1538: Tony Lodge
U3: Chris Stevens

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

computer kurs Heft 56



Große Fahrt

Stellen Sie Ihre Mannschaft zusammen, um in See zu stechen. Vergessen Sie aber nicht, einen Koch mitzunehmen, wenn Sie sich mit unserem Simulationsprogramm aufs hohe Meer wagen ...

Komfortabel

Datenbankprogramme mit eingebauter Programmiersprache sind besonders anwenderfreundlich. Wir zeigen, wie's geht.

Unterbrechung

Unser Debugger ist fast fertig. Nun das Löschen von Unterbrechungspunkten.

Schöne neue Welt

Zum Schluß unserer KI-Serie betrachten wir die Auswirkungen von Maschinen mit „kreativer Intelligenz". Wird die Evolution des Menschen verändert?