Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Künstliche Intelligenz

Die modulare Programmierung

Ausmalen mit Maschinencode

Annäherungswerte

Ein wöchentliches Sammelwerk

Heft 43

computer kurs Heft 43

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

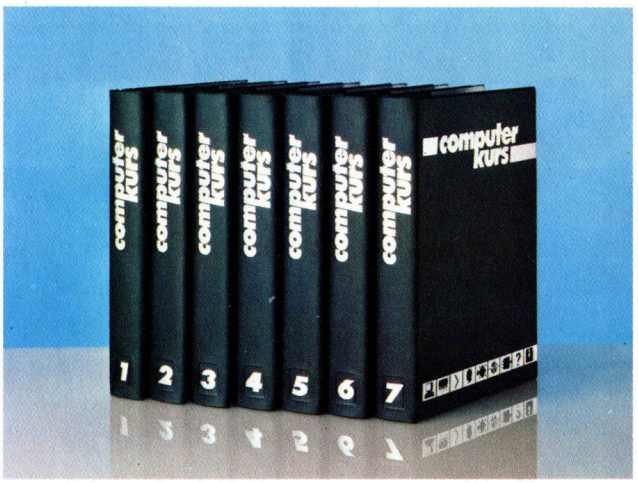
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

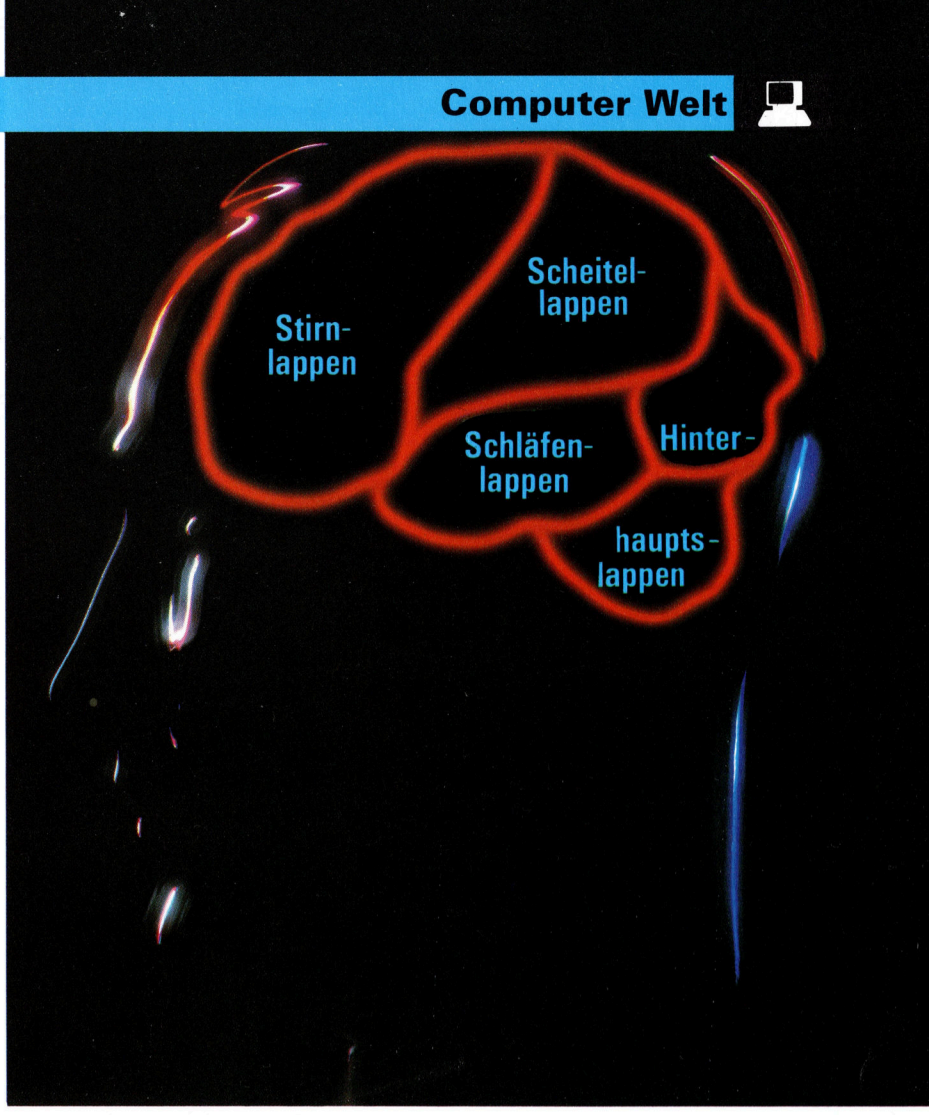
Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



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

Ran ans Hirn

Künstliche Intelligenz ist zweifelsfrei einer der aufregendsten Teilbereiche der gegenwärtigen Computerarbeit. In dieser Artikelserie gehen wir auf das faszinierende Thema detailliert ein und fügen Beispielprogramme in BASIC hinzu, die zeigen, wie „Künstliche Intelligenz" auf dem Heimcomputer Anwendung findet.



Der Bereich der Künstlichen Intelligenz befaßt sich mit der Entwicklung von Computersystemen, die Aufgaben durchführen, die – sofern ein Mensch sie ausübt – Intelligenz erfordern. Die Palette ist inzwischen dahingehend erweitert, daß man mit Wahrnehmungen verbundene Aufgaben (Sehen, Hören) einbezieht. Hauptaufgabe und Ziel der KI-Forschung ist das Programmieren von Maschinen, mit denen Aspekte menschlichen Verhaltens und Verstehens nachgeahmt werden.

Anfang der siebziger Jahre war es um die Artificial Intelligence (AI) oder auch „Künstliche Intelligenz" (KI) sehr still geworden. Gemeinhin betrachtete man den Bereich als einen Anflug von Irrsinn in der Computerwissenschaft. Heute dagegen ist KI sehr gefragt. Die damit betrauten Leute haben einen so guten Ruf, daß Risiko-Kapital-Anleger sie mit Angeboten förmlich überschütten. Regierungsstellen unterstützen kostspielige R & D-(Research & Development-) Programme, aus Angst, im „Rennen" um die KI ins Hintertreffen zu geraten. Software-Häuser verbreiten Pressemitteilungen, in denen ihre bekannten Produkte nun als KI-Systeme bezeichnet werden.

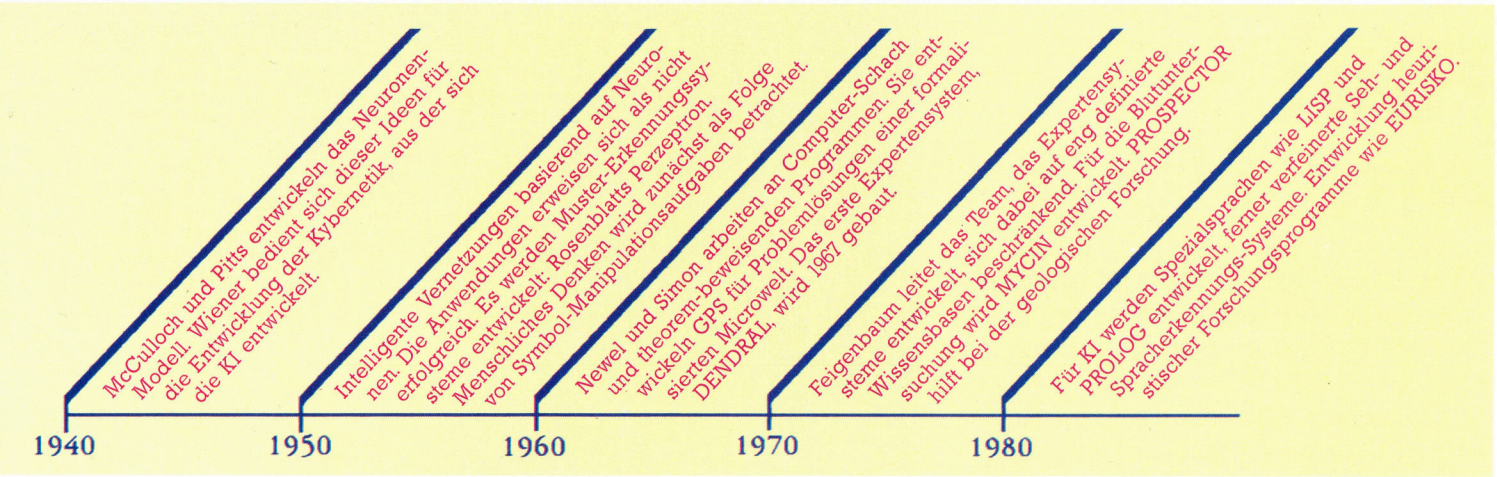
Um zu verstehen, in welchem Stadium sich die KI heute befindet und wohin sich dieser Bereich künftig entwickeln wird, ist es sinnvoll – wie bei so vielen Technologien –, einen Blick in die Vergangenheit zu werfen. Wir können unsere Kurzgeschichte der Künstlichen Intelligenz in vier dekadische Abschnitte gliedern, die jeweils durch ein beherrschendes Thema charakterisiert sind. Damit wird die Thematik zwangsläufig stark vereinfacht, hebt so aber die wesentlichen Punkte heraus. Jedes dieser Themen kann als Antwort auf die Frage gewertet werden, die einem KI-Forscher zum jeweiligen Zeitpunkt gestellt wurde, nämlich: „Was ist eigentlich Künstliche Intelligenz?"

1943 erstellten Warren McCulloch und Walter Pitts ein Modell des Neurons im menschlichen wie tierischen Gehirn. Diese abstrakte Nervenzelle diente als Basis für die mathematische, symbolische Darstellung der Gehirnaktivität. Andere Forscher, vor allem Norbert Wiener, übertrugen diese und ähnliche Ideen in jenen Bereich, der „Kybernetik" genannt wird (ein System, das auf der Voraussetzung basiert, daß eine fühlende Maschine entwickelt werden kann, indem man sich modellhaft biologischer Rückkopplungsmethoden und der Analyse bedient). In den fünfziger Jahren entwickelte sich aus der Kybernetik die „Künstliche Intelligenz".

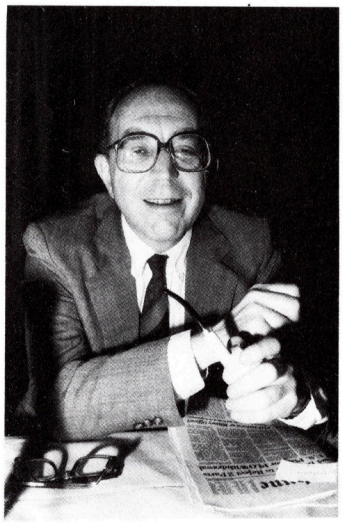
Die ersten KI-Forscher verwandten McCullochs formalisiertes Neuron als Baustein. In Anbetracht der ungeheuren Komplexität des Gehirns war es nicht überraschend, daß es ihnen auf Basis dieses Modells nicht gelang, intelligente Systeme zu schaffen. Sie gingen von folgendem aus: „Das Gehirn ist ein intelligenter Problemlöser. Simulieren wir also das Gehirn." Damit aber wurden Hard- und Software jener Zeit nicht fertig.

Zuerst war das Perzeptron

Eines der wenigen erfolgreichen Systeme, die damals entwickelt wurden, war Rosenblatts „Perzeptron". Es handelte sich um ein elementares, visuelles System, das man lehren konnte, Muster zu erkennen. Wie in der Abbildung auf der übernächsten Seite gezeigt, besteht das „Perzeptron" aus einem endlichen Raster lichtempfindlicher Zellen, mit denen eine miniaturisierte Augennetzhaut als Modell dargestellt wird. Ergänzend gibt es eine Reihe von Elementen, die Zustände erkennen – im übertragenen Sinne als „Dämonen" bezeichnet – und den Zustand der Zellengruppen im Raster darstellen. Sie reagieren, wenn charakteristische Untermuster vorhanden sind, indem sie einem „Entscheidungsfäller" ein Signal über-



KI wird ein immer größer werdendes Gebiet praktischer Forschung und greift in vielen Anwendungsbereichen ein. Die Geschichte der KI ist kurz, gemessen an anderen wissenschaftlichen Disziplinen, wie unsere Übersicht zeigt . . .



Edward Feigenbaum ist der Leiter des KI-Teams an der Stanford Universität, Kalifornien. Er entwickelte die ersten Expertensysteme. Das ist besonders bemerkenswert, da er sich von der Idee umfassend intelligenter Programme entfernte und auf Systeme beschränkte, die Teilaufgaben in eng definierten Wissens- und Betrachtungs-Bereichen zu lösen haben.

senden. Dieser vervielfacht jedes von Einzel-„Dämonen" empfangene Signal mit einem positiven oder negativen Wertungsfaktor und addiert die daraus resultierenden Werte. Wird eine vorgegebene Schwelle überschritten, zeigt das Perzeptron „Ja" an. Andernfalls lautet die Entscheidung „Nein". So kann das Perzeptron zwischen zwei Bildarten unterscheiden. Dieselben Regeln lassen sich zwecks Unterscheidung mehrerer Bildarten anwenden.

Man hoffte anfangs darauf, daß Perzeptrone viele Probleme unterschiedlicher Art lösen würden. Die Hoffnungen aber erfüllten sich nicht. KI-Forscher fanden dann eine neue Vorgehensweise: Sie betrachteten menschliches Denken als Koordination wichtiger Symbol-Manipulations-Aufgaben. Dies war eine grundlegende Veränderung des Forschungsansatzes. Die Gestalter befanden sich zumindest aber auf festerem Boden, da Computer ja Dinge wie Symbolvergleiche und ähnliches durchführen können. Dies eben wurde als Grundlage intelligenter Problemlösung angesehen. Der schwierige Teil bestand darin, diese einfachen Aktivitäten miteinander zu verknüpfen.

In den sechziger Jahren schafften sich Allen Newell und Herbert Simon von der Carnegie-Mellon-Universität auf dem Gebiet der KI einen Namen. Neben anderen Erfindungen arbeiteten sie an Theorem-Beweisen und Computerschach. Ihre beeindruckendste Entwicklung war ein Programm, GPS genannt oder „General Problem Solver" (Allgemeiner Problemlöser). Es war insofern „generell", als der Benutzer ein „Aufgaben-Umfeld" definieren konnte, indem er die Gegenstände eines bestimmten Bereichs benannte sowie die mit den Gegenständen verknüpften Operatoren. Dieses System war aber auf Aufgaben mit relativ wenigen möglichen Zuständen und fest definierten Regeln beschränkt.

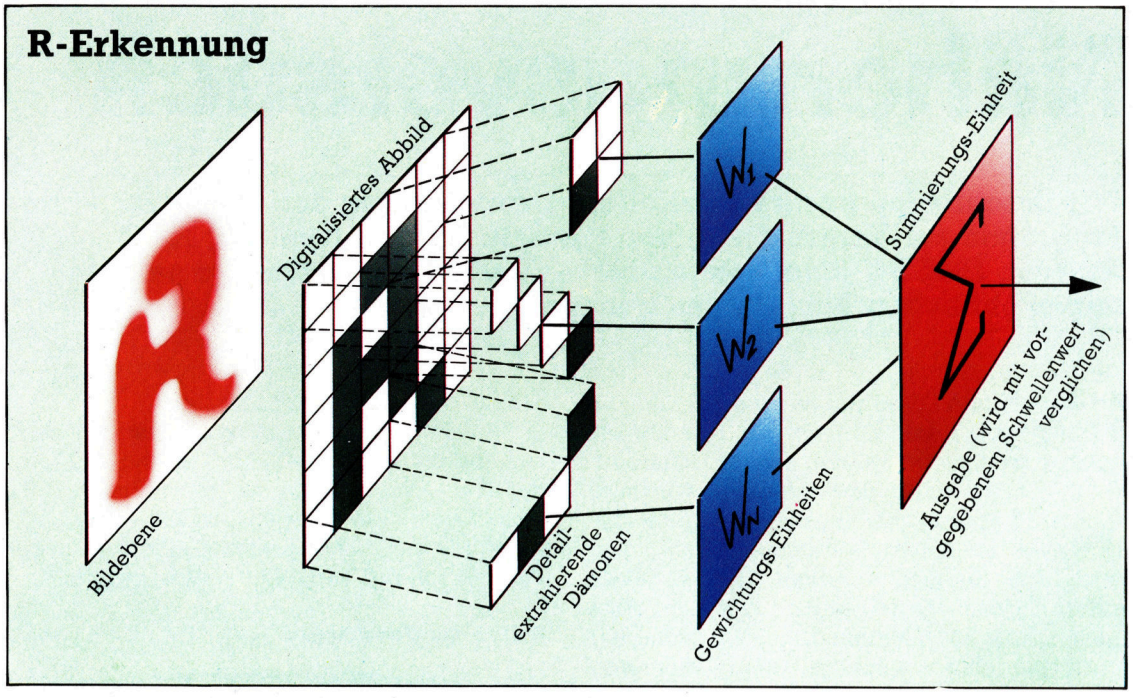
Die hinter GPS steckende Grundidee war, daß Problemlösung nichts weiter sei als das Durchsuchen eines Bereichs möglicher Lösungen. Um die Suche erfolgreich werden zu lassen, mußte sie nach heuristischen (Lernen durch eigene Erfahrungen) Regeln stattfinden, die sie zum gewünschten Ziel führte. Das heißt, sie mußte einen Weg nehmen, der durch Versuch und Irrtum gekennzeichnet war. Ein Automat, der ein Labyrinth zu durchdringen versucht, würde folglich endlos lange suchen, wenn er über die Labyrinthstruktur nichts weiß. Würde er aber einen Hinweis bekommen, wenn es „warm" würde, hätte er die Chance, sein Ziel (schneller) zu erreichen (wenn auch nicht immer, da Heuristik keine Funktionsgarantie beinhaltet und zuweilen leider in Sackgassen führt).

Wie gesagt, entpuppte sich der GPS als ungeeignet für die Lösung von Problemen der Realwelt. In den siebziger Jahren begann ein Team an der Stanford-Universität unter Leitung von Edward Feigenbaum, diesen Nachteil zu beseitigen. Statt zu versuchen, umfassende Intelligenz auf dem Computer zu erzeugen, konzentrierte man sich auf den kleinen Bereich des Fachwissens, die „Expertise". So wurde das Expertensystem geboren.

Das erste dieser Art war DENDRAL, ein Interpreter für die Verarbeitung massenspektrometrischer Daten. Er wurde bereits 1967 gebaut und beeinflußte den 1974 entwickelten MYCIN am meisten. MYCIN diagnostizierte bakterielle Blutinfektionen und verordnete medikamentöse Behandlungen. Daraus leitete sich eine ganze Familie von „Clones" ab, die inzwischen an Kliniken Verwendung finden.

Mit MYCIN wurden zahlreiche neue Elemente entwickelt, die zu den Grundlagen des Expertensystems gehören. Sein „Wissen" besteht aus Hunderten von Regeln folgender Art:
Regel 85
WENN:
1. Der Sitz der Kultur im Blut ist und
2. die Gramfärbung des Organismus gramnegativ ist und
3. der Bau des Organismus stäbchenförmig ist und
4. der Patient in Verdacht steht, ein Träger dieser Krankheit zu sein

DANN:
Besteht die Vermutung, daß es sich beim Organismus um Pseudomonas-aeruginosa handelt.



Das vom Perzeptron zu erfassende Bild (in diesem Fall der Buchstabe „R") wird auf eine Ebene projiziert und digitalisiert. „Dämonen" sammeln kleine Gruppen von Punkten und reagieren, wenn das Muster, auf dessen Erkennung sie programmiert sind, präsent ist. Jede Reaktion eines „Dämonen" (entweder 0 oder 1) wird mit einem Gewichtungsfaktor multipliziert, abhängig von seiner Wichtigkeit im Gesamtmuster. Diese Reaktionen werden addiert. Das Ergebnis wird dann mit einem Schwellenwert verglichen. Ist es größer als der Schwellenwert, hat das System den Umriß erkannt, andernfalls nicht. Die Schwellen und „Dämonen" können bei der „Ausbildung" des Perzeptron zur Mustererkennung justiert werden.

Diese Regeln beruhen auf Wahrscheinlichkeiten. Shortliffe, der Erfinder von MYCIN, war Mediziner. Er entwickelte ein Schema, das auf sicheren Faktoren beruhte, um dem System die Möglichkeit zu geben, folgerichtige Schlüsse aus unsicheren Daten abzuleiten. Folglich steht die „Vermutung" nicht mit einer bestimmten Wahrscheinlichkeit fest, sondern entspricht vielmehr einem „Vortäuschungs-Faktor", das heißt, dieser simuliert etwas, um zu einem Ergebnis zu kommen. Der entscheidende Punkt jedoch ist, daß MYCIN und ähnliche Systeme selbst dann zu richtigen Folgerungen kommen, wenn die Informationen unvollständig und teilweise irreführend sind. Dabei bedienen sie sich der Methoden der Näherungsrechnung.

Entscheidung mit Begründung

Zudem kann MYCIN seine Entscheidung begründen. Der Arzt kann das System auf verschiedene Art befragen. Entweder derart, daß er fragt, wie es zu einer bestimmten Folgerung gekommen ist oder warum es nach einer bestimmten Information fragt. Das System antwortet, indem es die Schlußfolgerung zurückverfolgt und den Prozeß, der zur derzeitigen Entscheidung führt, beschreibt.

Schließlich – und das ist entscheidend – funktioniert MYCIN. Es führt aus, wozu Menschen eine jahrelange Ausbildung brauchen. In der Praxis wird MYCIN mehr zur Ausbildung als zur Diagnose verwendet. Große Unternehmen, Regierungsstellen und die Medien sind an diesem System interessiert.

Und damit kommen wir zu den achtziger Jahren. Expertensysteme sind „in", und ihre „aktive Zutat" ist Wissen – der Umfang und die Qualität der Wissensbasis bestimmen den Erfolg eines Expertensystems. Nun kann man aber Wissen nicht in ein Computerprogramm wie Zahnpasta in eine Tube pressen. Die Codierung der Fähigkeiten eines menschlichen Experten ist ein langer und arbeitsintensiver Prozeß. Während die Welt noch über Expertensysteme staunt, konzentriert man sich bei der Künstlichen Intelligenz bereits auf das Problem des Lernens der Maschine. Dies nämlich ist der Weg, wie man Wissen automatisch erhalten kann. Im Mittelpunkt der Betrachtungen steht jetzt ein Programm mit Namen Eurisko.

Eurisko ist ein Forschungsprogramm, das seine eigene Struktur heuristischer Gesetze durch Induktion erweitert und verbessert. Davon abgesehen, daß Eurisko das „Trillion Credit Squadron"-Flottenmanöver gleich drei Jahre hintereinander gewonnen hat, kann Eurisko auch praktische Probleme lösen. Ein Ergebnis war die Erfindung eines neuartigen dreidimensionalen Logik-Gatters beim IC-Design. Unzweifelhaft ist Eurisko das Beste, was KI-Forschung bisher hervorgebracht hat. Und da die KI selbst der progressivste Bereich in der Computerwissenschaft ist, muß von hier aus die künftige Entwicklung dieses Feldes betrachtet werden.

Ironischerweise ist KI wieder zu den Anfängen zurückgekehrt, da schon in den frühen Tagen der Kybernetik Lernen als das Schlüsselproblem gesehen wurde.

In dieser Serie werden wir auf die unterschiedlichen Aspekte Künstlicher Intelligenz eingehen, die verschiedenen Problembereiche aufzeigen (so das Sehen und die Verarbeitung natürlicher Sprachen) und ebenso die Techniken darstellen, die dafür entwickelt wurden. Alle Musterprogramme sind in BASIC für den Acorn B, den Commodore 64 und den Sinclair ZX Spectrum geschrieben.

Näherungsversuch

Wir beschließen unsere zweiteilige Untersuchung des TK!Solver – ein Programm zur Verarbeitung von Formeln für den Apple II, ACT Apricot, IBM PC (und Kompatible) –, indem wir uns einige seiner ungewöhnlichen Fähigkeiten genauer ansehen.

In der vorigen Folge wurde deutlich, daß der TK!Solver mit dem Konzept der Tabellenkalkulation weit in den Bereich höherer Mathematik und technischer Berechnungen vordringt. Hier wird die ungewöhnliche Fähigkeit des TK!Solver untersucht, Berechnungen zu iterieren. Eine Iteration versucht, das Ergebnis durch „Raten" festzustellen. Normalerweise lassen sich in Gleichungen alle Variablenwerte berechnen, wenn zu Anfang genügend Daten vorhanden sind. Das Programm teilt ein Problem dann einfach in mehrere Berechnungen auf. So kann

$A^2 + B^2 = 2COS\ Y$

leicht für jede der drei Variablen gelöst werden, wenn die anderen beiden Werte bekannt sind. TK!Solver löst diese Gleichung – vorausgesetzt, das Programm hat die Werte A und B – mit dem „Direct Solver" problemlos.

Es gibt jedoch Berechnungen, deren Werte sich nicht direkt bestimmen lassen. Einer dieser Fälle ist die redundante Gleichung, die eine Variable mit sich selbst definiert. So ist beispielsweise in der Gleichung:

D = (A + B) / (2 * D)

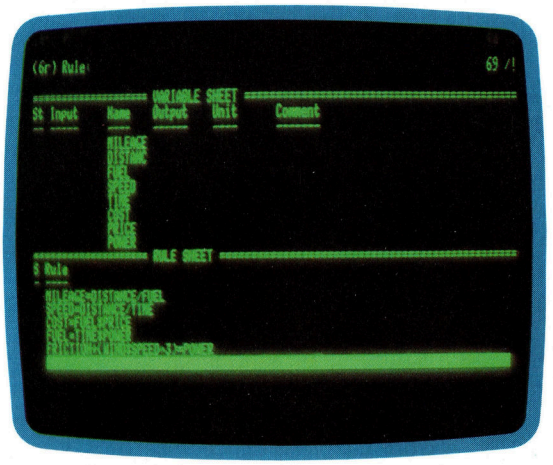
A der einzige bekannte Wert. Andere Probleme können bei unvollständigen Modellen auftreten oder bei Modellen mit vielen unabhängigen Variablen und wenigen Daten. Da das Konzept der Iteration recht kompliziert ist, sehen wir uns ein praktisches Beispiel an.

Wir nehmen dazu das Modell der Autofahrt (siehe letzte Folge) und fügen weitere Komponenten ein. Das Modell enthielt fünf Werte: distance (Entfernung), time (Zeit), speed (Geschwindigkeit), fuel (Treibstoff) und mileage (Fahrstrecke in Meilen). Die Fahrstrecke errechnet sich aus Geschwindigkeit und Treibstoffverbrauch, die Entfernung aus Geschwindigkeit und Zeit etc. Wir wollen nun feststellen, wie schnell wir fahren müssen bzw. dürfen, wenn die Reisekosten innerhalb einer vorgegebenen Summe bleiben sollen.

Zunächst müssen in das Modell weitere Faktoren eingebaut werden. Wir brauchen die Fahrleistung, den Beschleunigungswiderstand, den Luftwiderstand während der Fahrt, kurz alle Komponenten, die Auswirkungen auf Geschwindigkeit und Fahrstrecke haben. Außer den Fahrzeugeigenschaften braucht man nun noch die Obergrenze der vorgegebenen Summe und die Treibstoffkosten.

Mit diesen Daten kann das eigentliche Modell aufgebaut werden, indem die entsprechenden Gleichungen in die Formeltabelle (Rule Sheet) eingetragen werden (eine Gleichung pro Zeile). Die Gleichungen werden automatisch in die Variablentabelle übertragen:

Aufbau der Gleichungen



Da der Bildschirm nicht alle Informationen gleichzeitig darstellen kann, dehnen wir die Variablentabelle auf den gesamten Monitor aus, indem wir den Cursor mit der Taste (;) in die Variablentabelle setzen und W1 eingeben. Jetzt zeigt der Bildschirm alle Variablen, und wir können die Werte eintragen.

Wenn genügend Daten vorhanden sind, laßt sich das Modell auf direktem Wege lösen. Geben Sie in Spalte „INPUT" folgende Werte ein:

INPUT-Werte für „Direct Solver"

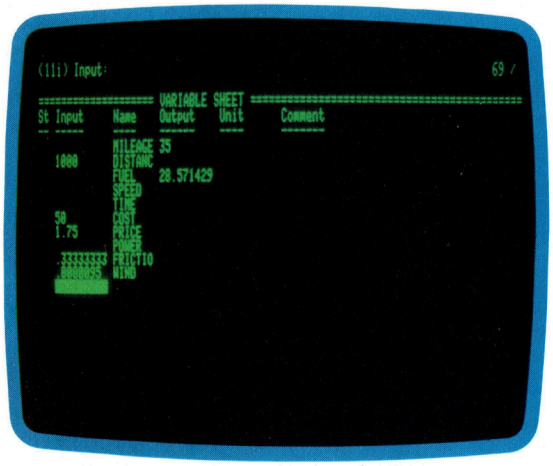
Drücken Sie nun !, um die Berechnung auszuführen. TK! gibt die Meldung „Direct Solver" aus und kurz darauf die gesuchten Werte in der Spalte „OUTPUT":

Ergebnisse des „Direct Solver"

Jetzt haben wir alle gewünschten Werte. Was aber passiert, wenn anfangs nicht genügend Daten zur Verfügung stehen? Nehmen wir eine Berechnung, bei der wir für eine Reise von 1000 Meilen maximal 50 Pfund zur Verfügung haben. Da wir den Treibstoffpreis (1,75 Pfund per Gallon) kennen, läßt sich leicht herausfinden, wieviel wir pro Meile ausgeben dürfen. Wie aber können wir berechnen, mit welcher Geschwindigkeit wir fahren müssen, um innerhalb des Budgets zu bleiben?

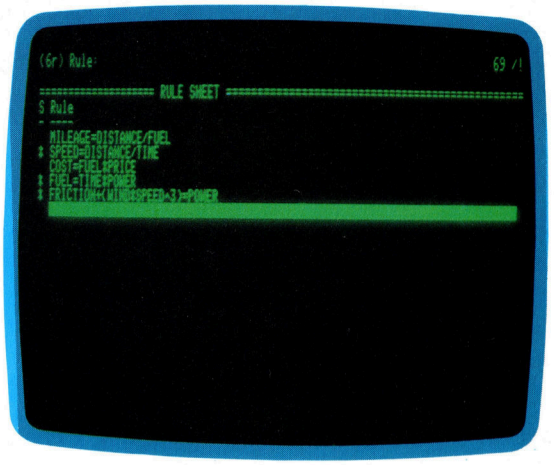
Für diese Berechnung löschen wir zunächst alle eingegebenen Werte mit RVY (Reset Variables Yes) und geben dann die bekannten Daten ein: 1000 für „distance", 50 für „cost" und 1.75 für „price". Wir nehmen den Wert von 1/3 für den Beschleunigungswiderstand und 0,0000095 für den Luftwiderstand. Nach der Eingabe des ! erscheinen folgende Werte:

Unvollständiges Modell



Sie sehen, daß für speed, time und power keine Werte errechnet wurden. Speed ist aber gerade der Wert, den wir benötigen. Die Formeltabelle zeigt nun, daß drei Gleichungen mit einem Stern (*) in der Statusspalte gekennzeichnet sind.

Unvollständige Gleichungen



Da sich das Modell nicht mit dem „Direct Solver" lösen läßt, müssen wir den „Iterative Solver" nehmen, der den Ergebniswert mit einer Reihe von Annäherungen feststellt.

Zunächst müssen wir dazu den zuvor berechneten Wert für mileage als Anfangswert in die Input-Spalte setzen, indem wir in der Statusspalte der Variablentabelle neben mileage ein I eingeben. Dann schätzen wir einen Wert für speed – beispielsweise 50 –, tragen die Zahl ebenfalls in die INPUT-Spalte ein, setzen ein G (für Guess – raten) in die Statusspalte und drücken ! für die Berechnung. TK! gibt nun am oberen Bildschirmrand die Meldung „Iterative Solver" aus und zählt die Anzahl der Annäherungen. Beim vierten Versuch erreicht TK! die richtigen Werte für speed, time und power:

Ergebniswerte der Iteration

Das Ergebnis zeigt, daß wir mit einer Durchschnittsgeschwindigkeit von nur 47 Meilen pro Stunde ans Ziel kommen, ohne mehr als die vorgegebene Summe zu verbrauchen. Je näher der geratene Wert der Lösung ist, desto weniger Zeit braucht TK! für die Berechnung.

Algorithmen

Es wird untersucht, wie sich die Programmerstellung mit Hilfe von Algorithmen optimieren läßt.

Algorithmen können Abläufe entweder durch „einfache" Vorgänge beschreiben, die keiner Erklärung bedürfen, oder mit anderen, zuvor definierten Vorgängen. So genügt im Rezept eines Kochbuches die Anweisung „bereiten Sie eine Bechamelsoße", wenn das Rezept für Bechamelsoße an einer anderen Stelle des Buches ausführlich erklärt ist. Programmalgorithmen bestehen daher aus Befehlen, die wiederum Algorithmen (Prozeduren, Routinen, Funktionen) einsetzen. Diese (Unter-)Algorithmen sind entweder an anderen Stellen des Programms bereits angelegt oder in der Sprache selbst enthalten (beispielsweise Befehle wie PRINT und DIM oder Funktionen wie LOG und TAN).

In diesem Artikel untersuchen wir, wie sich Algorithmen aus anderen Algorithmen und „einfachen" Abläufen (den Befehlen und Funktionen einer Programmiersprache) aufbauen lassen. Zunächst werden die Algorithmen für grundlegende Aufgaben entwickelt (beispielsweise die Bewegung eines Sprites oder eine Zahleneingabe). Die Grundalgorithmen werden dann zu umfassenderen zusammengesetzt, die etwa den Bildschirm aufbauen oder ein Menüsystem steuern. Diese Algorithmen sind wiederum Teile größerer Funktionszusammenhänge, so daß sich am Ende das gesamte Programm als ein Algorithmus mit vielen untergeordneten Algorithmen ansehen läßt. Dieses Konzept bildet die Grundlage der „strukturierten" oder „modularen" Programmierung, auf die wir später genauer eingehen.

Jeder Algorithmus besitzt eine Eingabe und eine Ausgabe. Mit anderen Worten, Algorithmen sind Vorgänge, die Eingangsdaten in Ergebnisdaten umwandeln. Die Eingangsdaten werden dem Algorithmus von außen in Form von „Parametern" übergeben. Sie erscheinen dem Algorithmus als Konstanten, können sich aber zwischen zwei Aufrufen eines Algorithmus verändern. Parameterübergaben sind schon dem Anfänger vertraut. Das einfache Programm:

```
10 PRINT "HALLO!"
```

übergibt den Parameter „HALLO!" an den Algorithmus, den der Befehl „PRINT" aufruft. Ähnliche Beispiele sind TAN(P), LEFT$ (P$,5) oder POKE P,5, wobei P, P$ und 5 die Parameter sind. Die Ergebniswerte des Algorithmus sind ebenfalls Parameter. Programmiersprachen mit lokalen Variablen (wie PASCAL und C) übergeben die Parameter im allgemeinen gleich beim Aufruf einer Prozedur:

```
procedure ( parameter1, parameter2 etc. );
```

Beim Aufbau eines Algorithmus besteht der erste Schritt immer darin, den Inhalt, die Typen (Ganzzahl, reale Zahl, String etc.), die Größe und den Wirkungsbereich der Ein- und Ausgabeparameter festzulegen.

Nach den Definitionen der Ein- und Ausgabe wird im nächsten Schritt skizziert, wie die Eingabeparameter in Ausgabeparameter verwandelt werden können. Da es für diese Umformung kein allgemeingültiges Rezept gibt, ist man hier auf seine eigene Kreativität angewiesen. Es gibt jedoch einige systematische Ansätze, die dabei helfen können.

Die einfachste Methode besteht darin, fertige Algorithmen zu übernehmen. Viele Programmiersprachen bieten brauchbare Algorithmen wie String-Verarbeitung, trigonometrische Funktionen, Ein- und Ausgabemechanismen und Module für Sortierung und Matrixbearbeitung. Doch auch Ihre eigenen Programme können einsatzbereite Algorithmen enthalten. Sie brauchen den bestehenden Code dann nur in neue Programme zu übernehmen.

Es lohnt sich weiterhin, die in Computerzeitschriften veröffentlichten Programme auf brauchbare Routinen durchzusehen. Und schließlich gibt es Bereiche, deren Algorithmen zwar nicht für Computer gedacht waren, die sich aber vielseitig einsetzen lassen. Es gibt beispielsweise zahlreiche Bücher über Buchhaltung, die Formeln für die Berechnung von Salden, Abschreibungen etc. enthalten. Mit ein wenig Nachlesen lassen sich Buchhaltungsprogramme so viel einfacher und zuverlässiger entwickeln. Das Prinzip läßt sich auch auf andere Bereiche wie Elektronik oder Mathematik anwenden.

Klar und wirksam

Ob Sie einen Algorithmus nun übernehmen oder von Grund auf entwickeln, jeder Ablauf muß zwei grundlegende Kriterien erfüllen: Er muß klar und er muß wirksam sein. Klarheit schließt mehrdeutige Anweisungen aus, die sich bei der Entwicklung eines Algorithmus leicht einschleichen können. Eine Routine, die aus einer Liste alle mit „A" und „B" anfangenden Namen heraussuchen soll, könnte folgendermaßen aussehen:

```
IF ANFANG='A' AND ANFANG='B' THEN . . .
```

Da sich die Wörter „und" und „oder" der Um-

gangssprache grundlegend von dem AND und OR der booleschen Logik unterscheiden, ist diese Programmzeile falsch: Hier wird ein logisches OR gebraucht!

Wirksamkeit bedeutet, daß Programme keine Befehle enthalten, deren Ausführung unmöglich ist. Befehle sind wirksam, wenn sie mit Papier und Bleistift in einem endlichen Zeitraum ausgeführt werden können. Anweisungen wie „setze X auf die höchste Primzahl" sind nicht wirksam, da es keine „höchste Primzahl" gibt.

Bestimmte Kriterien bewerten Algorithmen auch als Ganzes. So müssen Algorithmen beendet werden können. Die folgende Routine erfüllt dieses Kriterium nicht (obwohl ihre Befehle klar und wirksam sind), sondern erzeugt eine Endlosschleife:

Schritt 1 Setze X gleich 1
Schritt 2 if X > 3 then Ende
Schritt 3 gehe zu Schritt 1

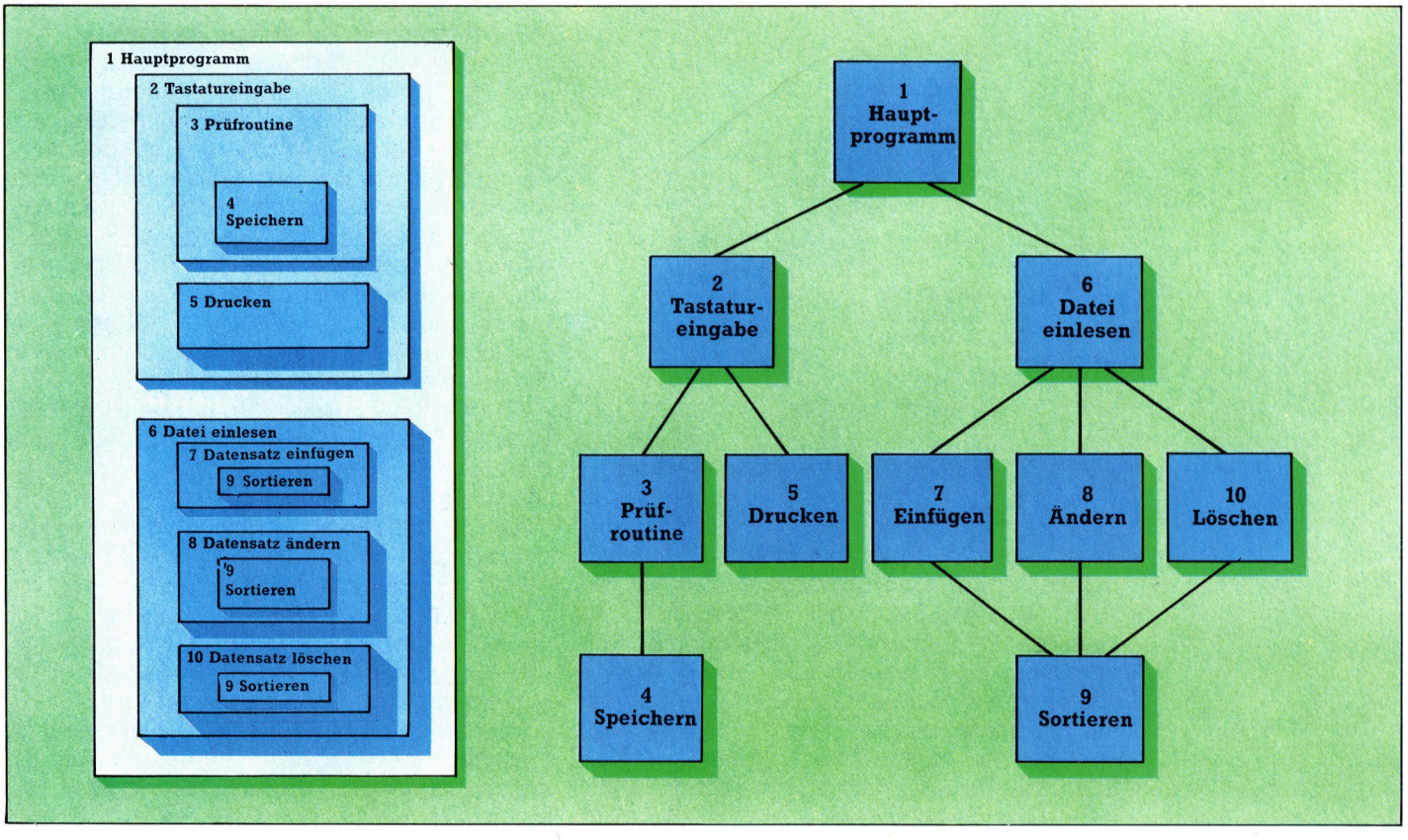
Es läßt sich nicht immer auf eine einfache Weise feststellen, ob ein Algorithmus beendet werden kann. Im allgemeinen enthalten Schleifenalgorithmen Bedingungen, die einen Ausstieg aus der Schleife ermöglichen. Es muß jedoch immer überprüft werden, ob die Endbedingung überhaupt eintreten kann.

Über die Kriterien Wirksamkeit, Allgemeingültigkeit und Eleganz lassen sich Algorithmen miteinander vergleichen. Wirksamkeit wird dabei aufgrund von Schnelligkeit und Speicherbedarf beurteilt. Oft – aber nicht immer – arbeiten diese beiden Komponenten Hand in Hand (ein schneller Code braucht selten viel Speicherplatz). Ist ein Algorithmus einmal gefunden, sollte seine Wirksamkeit durch geeignete Anordnung der Befehle optimiert werden. Beispielsweise lassen sich Berechnungen mit Ganzzahlen weitaus schneller durchführen als mit Fließkommaarithmetik. Zur Optimierung einer Routine müssen daher oft völlig neue Algorithmen gefunden werden, die die gleiche Aufgabe erfüllen.

Allgemeingültig

Allgemeingültigkeit ist ein weiteres wichtiges Merkmal: Ein Algorithmus sollte nicht nur für eine Aufgabe Gültigkeit haben, sondern ein breites Spektrum von Aufgaben ausführen können. Auf lange Sicht gesehen lohnt es sich, Algorithmen so allgemeingültig wie möglich anzulegen. Wenn beispielsweise ein Programm vom Anwender mehrmals eine JA/NEIN-Entscheidung verlangt, sollte die entsprechende Routine die Meldung „Eingabe J(a) oder N(ein)" darstellen, die Eingabe annehmen, überprüfen, ob auch „J" oder „N" eingegeben wurde, und den Anwender entweder erneut zur Eingabe veranlassen oder das Ergebnis als Parameter zurückgeben. Elegante Algorithmen sind zugleich einfach und raffiniert. Wirksame und allgemeingültige Algorithmen lassen sich jedoch leichter aufbauen.

Für Algorithmen sind weiterhin die Steuerung und der Datenfluß wichtig. Wie diese beiden Komponenten in einem Flußdiagramm dargestellt werden, behandeln wir in der nächsten Folge dieser Serie.



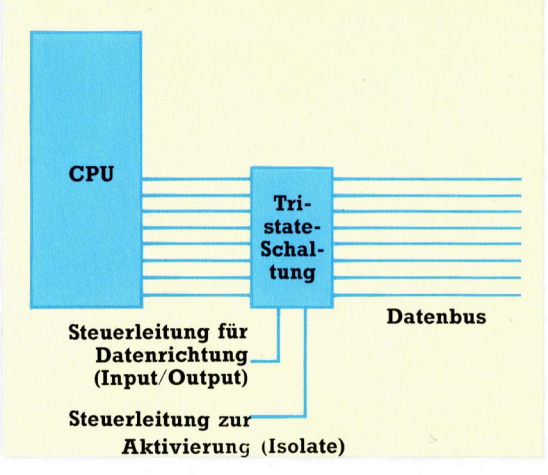
Das Blockdiagramm (links) zeigt, wie die Algorithmen eines Programms ineinander verschachtelt sind, während das Flußdiagramm (rechts) die Funktionsebenen der gleichen Vorgänge darstellt. Die einfachsten Algorithmen befinden sich dabei in der größten Schachtelungstiefe beziehungsweise auf der untersten Ebene der Hierarchie.

Bits im Bus

Diese Folge des Logik-Kurses befaßt sich mit dem System der Datenübertragung zwischen Prozessor und Speicher. Sie erfahren, was es mit Adressen, Datenbussen, Adreßregistern und Tri-State-Schaltungen auf sich hat.

Jeder Speicherplatz in einem Heimcomputer besteht normalerweise aus acht Bits. Diese acht Bits können gleichzeitig über acht Leitungen zur CPU gelangen, wo sie dann entsprechend den Anweisungen des Programms weiterverarbeitet werden. Aber auch der „Gegenverkehr" – von der CPU zum Speicher – ist möglich. Der Maschinencodebefehl LDA $1234 bewirkt etwa, daß der Wert von Speicherplatz $1234 an die CPU geschickt wird. STA $1234 hingegen sendet einen Wert von der CPU über den Datenbus zum Speicherplatz $1234.

Damit beides funktioniert, muß der Datenbus die Übertragung in beiden Richtungen erlauben. Manchmal ist es aber auch nötig, die CPU vom Datenbus zu trennen. Jede Busleitung kann deshalb in drei Zustände versetzt werden: INPUT, OUTPUT oder ISOLATE. Zum Wechseln zwischen diesen Zuständen verfügt jede Busleitung über eine sogenannte „Tri-State"-Schaltung.



Acht Tri-State-Schaltungen sind in einem IC vereinigt. Auf der kleinen Zeichnung sehen Sie, wie die Schaltung den Datenbus mit der CPU verbindet. Außerdem sind die Steuerleitungen für die Datenrichtung und die Aktivierung eingezeichnet, mit denen der gewünschte Betriebszustand angewählt wird. Solche Schaltungen werden auch beim Anschluß von Peripheriegeräten zur Ein- und Ausgabe auf den Datenbus verwendet.

Bestimmte Speicherplätze müssen immer über ihre Adresse abgefragt werden. Dazu trägt jeder Speicherplatz im ROM oder RAM als Namen eine eindeutige Ziffernkombination. Wie geht nun der Datentransfer vor sich?

Die meisten Heimcomputer haben neben dem Datenbus eine zweite Verbindung zwischen CPU und Speicher, den „Adreßbus". Oft hat der Adreßbus nicht nur acht, sondern 16 Leitungen, mit denen bis zu 65 536 einzelne Adressen – also 64 KByte – angesteuert werden können (2^{16}=65 536). Den gesamten Speicher muß man sich aufgeteilt in Sektionen mit jeweils 256 Speicherplätzen denken. Die acht niederwertigen Bits bestimmen den Speicherplatz innerhalb einer der Sektionen, die mit den acht höherwertigen Bits gewählt werden.

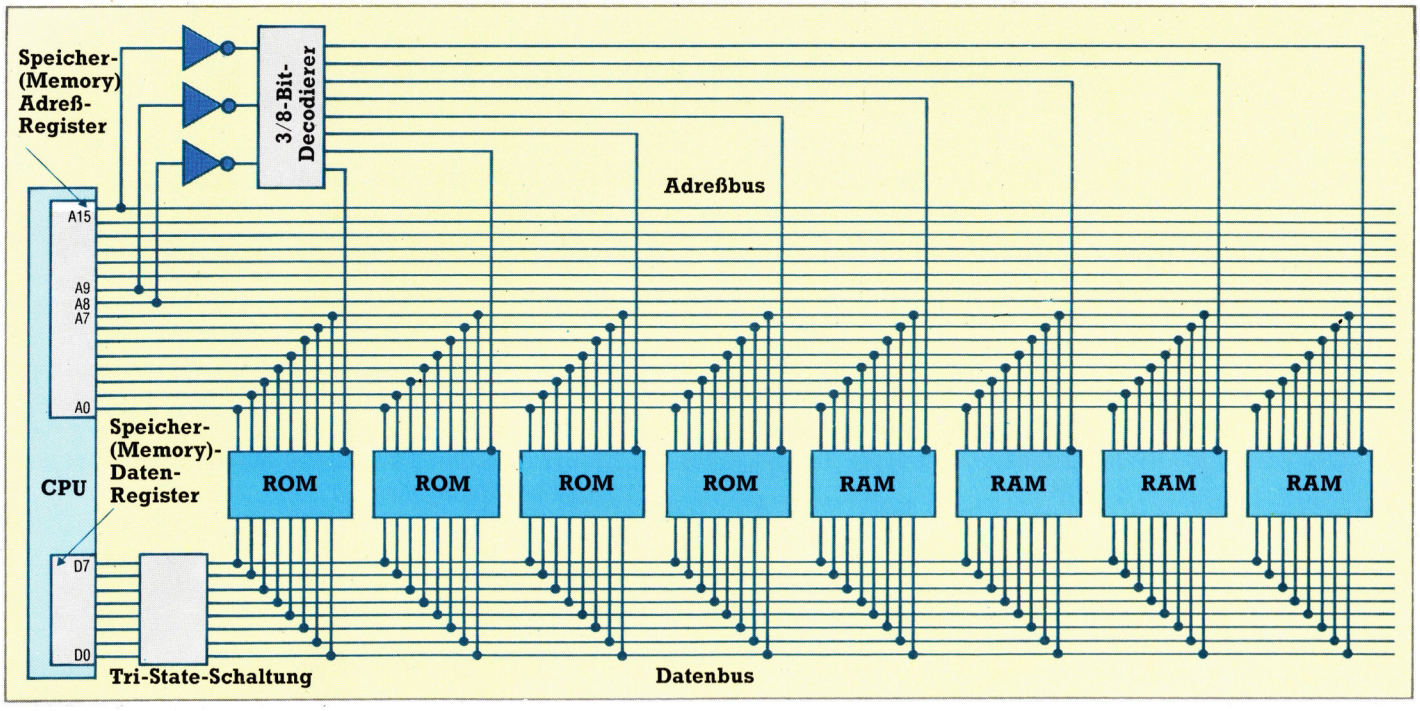
Am vereinfachten Beispiel eines Computers mit zwei KByte Speicher können wir sehen, wie das Ansprechen einer bestimmten Adresse vor sich geht: 256 Speicherplätze pro Sektion bedeutet, daß der Computer über insgesamt acht Sektionen verfügt. Für unser Beispiel gehen wir davon aus, daß der Speicherbereich zur Hälfte aus RAM-Bytes und zur Hälfte aus ROM besteht.

Memory Address Register

Die Adresse des benötigten Speicherplatzes findet sich in einem speziellen 16-Bit-Register der CPU, dem „Memory Address Register" oder „MAR". Da die acht niederwertigen Bits nur den Speicherplatz innerhalb einer Sektion definieren, können sie gleichzeitig mit allen Sektionen verbunden bleiben. Zur Auswahl eines bestimmten Moduls brauchen wir dann nur noch drei weitere Bits (2^3=8). Dieser Drei-Bit-Code muß auf acht Ausgangsleitungen – für jede Sektion eine – verteilt werden.

Die Zeichnung rechts oben zeigt, wie die Speicher-Sektionen über den Daten- und Adreßbus mit der CPU verbunden sind. Jede Sektion hat eine Zuleitung vom Drei(Bit)-auf-Acht(Leitungen)-Decodierer. Mit den drei höherwertigen Adreßbits wird die entsprechende Sektion angewählt.

Sie wissen jetzt, wie ein bestimmter Speicherplatz ausgewählt und die Daten daraus übertragen werden. Aber wie führt die CPU nun einen Befehl aus? Üblicherweise sind Maschinenprogramme in aufeinanderfolgenden Speicherplätzen abgelegt. Die Befehle können dabei auch zwei oder drei Speicherplätze belegen. Die Anweisung „ADD $13FF" bedeutet

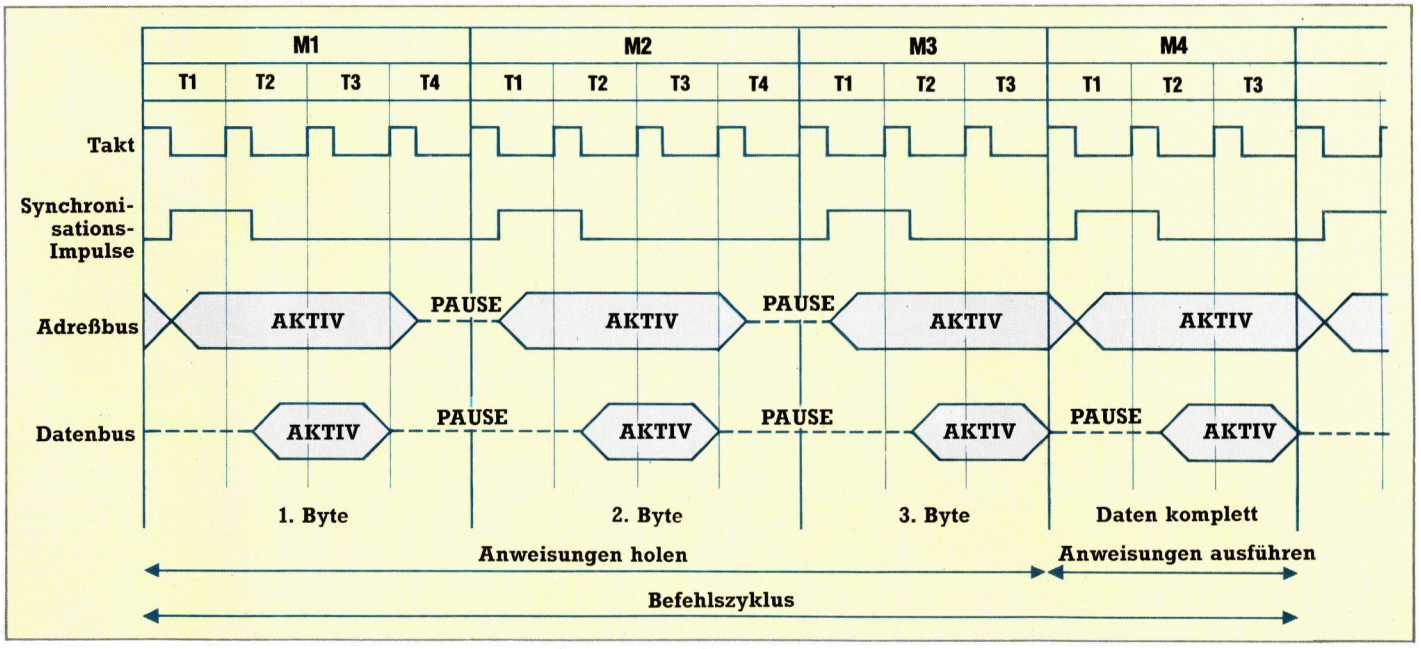


etwa „addiere den Inhalt des Speicherplatzes mit der hexadezimalen Adresse $13FF zum Akkumulator". Dafür wären drei Byte nötig: eines für den Befehl „ADD" und zwei für die 16-Bit-Adresse $13FF.

Bevor die Anweisung ausgeführt werden kann, muß sie erst einmal aus dem Speicher geholt werden. Drei Adressen müssen abgefragt werden, um die Bytes über den Datenbus zur CPU zu schaffen. Danach steht der komplette Befehl in einem Register der CPU, muß aber noch entschlüsselt und natürlich ausgeführt werden.

Computerhersteller veröffentlichen die Charakteristiken der verwendeten Prozessoren in Form von Zeitdiagrammen, welche die Abfolge der unterschiedlichen Aktionen des Computers darstellen. Die Überwachung des Ablaufs besorgt ein Taktgenerator. In unserem Beispiel wird der Adreßbus mit der positiven Flanke, der Datenbus mit der negativen Flanke eines Synchronisations-Impulses aktiviert. Der Synchronisations-Impuls selbst wird von der positiven Flanke des jeweils ersten Zeittaktes in einem Maschinenzyklus erzeugt. Die einzelnen Zyklen haben unterschiedliche Dauer, weil der Prozessor länger zur Entschlüsselung eines Maschinenbefehls als zur Verarbeitung eines numerischen Byte braucht. Die Entschlüsselung geht jedoch vor, weil erst der Befehl die genaue Anzahl der zu verarbeitenden Bytes angibt.

Die Abarbeitung eines Maschinenbefehls geht in zwei Phasen vor sich: Während der „Hol-Phase" wird über den Adreßbus zuerst der Befehl selbst von seinen Speicherplätzen zur CPU transportiert. Wenn sich der Befehl auf die Verarbeitung anderer Speicherinhalte bezieht, bleiben Daten- und Adreßbus in der darauf folgenden „Ausführungsphase" weiter in Aktion, denn auch die zu verarbeitenden Datenbits müssen nacheinander zum Prozessor geschafft werden.



Auf- und Abbau

Wir entwickeln einen allgemeinen Algorithmus für den Aufbau unserer Datenbank und stellen mehrere Programme vor, die die strenge Programmstruktur von PASCAL beim Aufbau und Abruf von Dateien verdeutlichen.

In unserer letzten Folge begannen wir mit dem Aufbau einer Datenbank, für die wir nun einige Variablen und folgenden allgemeinen Algorithmus definieren:

```
VAR
   Liste   : RecordListe;
   Anzahl : Cardinal;
BEGIN
   Anzahl : = 0; (* aktuelle Laenge der Liste *)
   (* Daten in die Liste einlesen und
      Anzahl aktualisieren *)
   (* Datensaetze sortieren *)
   (* Datensaetze anzeigen *)
END.
```

Für die Umsetzung dieses Algorithmus in ein Programm legen wir zunächst die Prozeduren für die drei Hauptschritte an. Nachdem wir für jede Prozedur die Parameterlisten aufgestellt haben, können wir das Gerüst für die einzelnen Prozedurblöcke entwerfen. Die letzte Anweisung des Programms soll folgendermaßen lauten:

```
Print (Liste, Anzahl)
```

Für die Übergabe der Datenliste und der Datensatzzahl brauchen wir in den Prozedurkopf von „Print" nur folgende Variablen einzusetzen:

```
PROCEDURE Print (Objekte : RecordListe,
                 MaxAnzahl : Grenze);
```

Für den Augenblick werden wir dieses Modul jedoch nicht weiterentwickeln, sondern erst die anderen Grundprozeduren entwerfen. Wir müssen dabei jede einzelne mit einem geeigneten Namen versehen, entscheiden, welche Daten als Parameter übergeben werden sollen und ob ein Teil davon das Format von „VAR"-(Adreß-)Parametern haben soll.

Bisher haben wir nur Datenstrukturen mit festen Grenzen kennengelernt. Die Grenzen wurden mit TYPE-Definitionen festgelegt und so dem Compiler mitgeteilt. PASCAL bietet aber auch Datenstrukturen, deren Größe sich während der Programmausführung verändern kann (mit bestimmten Einschränkungen sogar der Typ). Diese Strukturen werden nur durch die Kapazität des Arbeitsspeichers oder des Speichermediums begrenzt.

Wie Arrays können die Elemente einer Datei jeden strukturierten oder unstrukturierten Typ annehmen – nur eine Datei von Dateien ist nicht möglich. Um eine Datei mit einer unendlichen Anzahl Datensätze aufbauen und verarbeiten zu können, müssen wir an unsere Definition des Recordtyps nur folgende Deklaration anhängen:

```
TYPE
   DateiTyp = FILE OF Daten;
VAR
   Datei      : DateiTyp;
```

Die einzelnen Datensätze können dabei Namen, Summen oder jede andere Information enthalten. Ebenso wie wir mit „read (Zeichen)" ein Zeichen aus einer Datei lesen, so können wir mit „read (Datei, Feld)" oder „write (Datei, Feld)" ganze Datensatzstrukturen als ein Datenobjekt ansprechen. Wir brauchen diese Dateien jetzt nur noch zu eröffnen, um mit den vordefinierten Prozeduren „reset" und „rewrite" darauf zugreifen zu können.

Wenn Dateien auf ein Speichermedium geschrieben werden sollen, müssen ihre Namen in der Parameterliste des Programmkopfes aufgeführt sein.

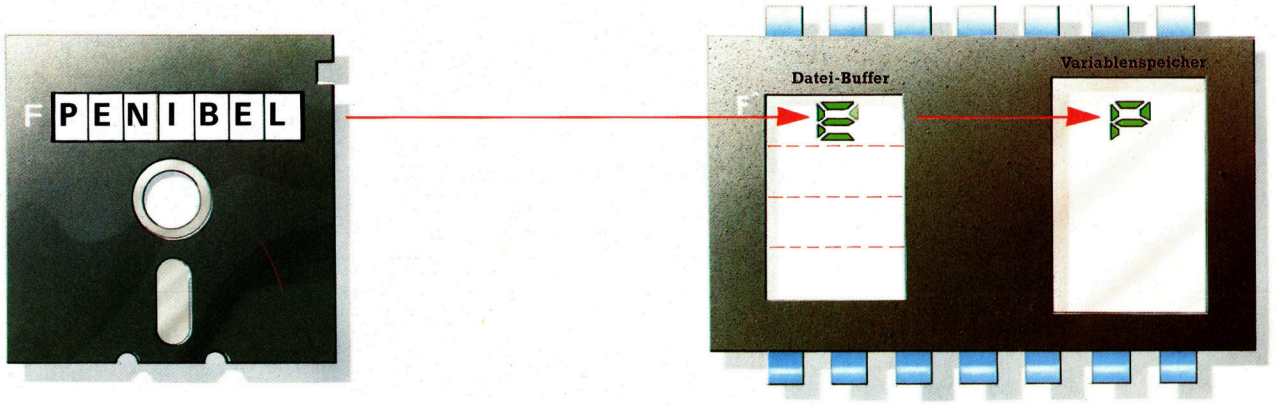
```
PROGRAM DateiVerwaltung
   (input,output,Datei);
```

Jeder Einsatz der Anweisungen „read" und „write" ruft die vordefinierten PASCAL-Prozeduren für Ein- und Ausgabe auf. Bisher kannten wir nur die Ein- und Ausgabedateien „input" und „output", die auf Microcomputersystemen normalerweise die Tastatur und den Bildschirm bezeichnen. Da PASCAL alle Systemteile als Dateien ansieht, haben alle Ein- und Ausgabevorgänge das gleiche Format. So schreibt „write (N)" ohne Angabe eines Dateinamens den Wert von N in die Datei „output", während „read" und „ReadLn" ohne Dateinamen automatisch die input-Datei ansprechen. Beides sind Textdateien – das heißt, jeder Datensatz besteht aus einem einzigen char-Wert.

Im Gegensatz zu anderen Dateiarten können Textdateien außer einer Datei-Endmarkierung noch beliebig viele Zeilen-Endmarkierungen enthalten. Diese Markierungen sind je nach Betriebssystem unterschiedlich. Sie können beispielsweise aus einzelnen Steuerzeichen oder zwei Zeichen (wie CR und LF) bestehen oder statt eines Zeichens die Zeilenlänge enthalten.

Für den universellen Einsatz bietet PASCAL

Das Einlesen einer Datei



ausschließlich für Textdateien die Funktionen „EoLn (F)" und die beiden vordefinierten Prozeduren „ReadLn (F)" und „WriteLn (F)". Die EoF-(F)-Funktion kann natürlich bei allen Dateitypen verwandt werden. Da Zeilen-Endmarkierungen jede Form annehmen können, ergibt das Lesen von „char" bei EoLn=true ein Leerzeichen. Normalerweise wird daher vor dem Lesen auf EoLn überprüft.

Dateien eröffnen

Da „input" und „output" vor und nach jeder Programmausführung existieren, sind sie permanent eröffnet und brauchen nicht erst lokalisiert oder aufgebaut zu werden. Alle anderen Dateien müssen jedoch mit einem externen Systemnamen angesprochen werden und eröffnet sein, wenn mit der Prozedur „reset" daraus gelesen werden soll. Für das Schreiben einer Datei mit der Prozedur „rewrite (Datei)" gelten die gleichen Regeln. Der generelle Ablauf für die Bearbeitung einer Textdatei sieht folgendermaßen aus:

```
(* Datei eröffnen *)
WHILE (* nicht Ende der Datei *) DO
  WHILE (* nicht Zeilenende *) DO
    (* Zeichen lesen *)
    (* Zeichen verarbeiten *)
    (* Zeilenende überspringen *)
```

Die Prozeduren „read" und „write" arbeiten mit Dateibuffern und den einfachen Ein- und Ausgabebefehlen „put" (Ausgabe) und „get" (Eingabe von einer Datei). Beim Aufruf der Prozedur „rewrite" befindet sich so lange keine Information im Datenbuffer, bis eine Schreiboperation ausgeführt wird:

```
F^ := Daten;
put ( F )
```

Ähnlich funktioniert die Anweisung „read (F, Daten)":

```
Daten := F^;
get ( F )
```

Die Ein- und Ausgabeanweisungen unserer Copy-Prozedur:

```
read ( Quelle, Zeichen );
write ( Ziel, Zeichen )
```

könnten daher auch ganz anders, nämlich folgendermaßen aussehen:

```
Ziel^ := Quelle^;
put ( Ziel );
get ( Quelle )
```

In diesem Fall benötigen wir die lokale char-Variable „Zeichen" nicht. Mit folgendem Ablauf läßt sich der Vorgang von ReadLn (F) besser verstehen:

```
WHILE NOT EoLn ( F ) DO
  get ( F ); (* Rest der Zeile ignorieren *)
get ( F ) (* und EoLn-Zeichen überspringen *)
```

Nach einem „ReadLn" enthält der Dateibuffer immer das erste Datenelement der nächsten Zeile. Wenn EoLn (F) „true" ist, kann dies ein Leerzeichen sein, bei EoF (F) = true ist es jedoch undefiniert. In diesem Fall ist nicht nur der Versuch, aus dieser Datei zu lesen, illegal, sondern auch jede andere Dateioperation, wie das Testen des Dateibuffers. Andere Dateien als input und output müssen daher sehr sorgfältig programmiert werden.

Wir wissen nun, wie die verschiedenen Endbedingungen begonnen werden und können daher eine Prozedur schreiben, die Leerzeichen überspringt.

```
PROCEDURE LeerzeichenUeberspringen ( VAR
  F : text );
  CONST
    Leerzeichen = ' ';
  VAR
    fertig : boolean;
  BEGIN
    fertig := EoF ( F );
    IF NOT fertig THEN
      fertig := input^ > Leerzeichen;
    WHILE NOT fertig DO
      BEGIN
```

PASCAL legt bei der Eröffnung der Datei F automatisch den Bufferbereich F^ an und liest dort das erste Zeichen der Datei ein. Ein read-Befehl ordnet dieses Zeichen einer Variablen zu und überträgt das nächste Zeichen der Datei in den Bufferbereich. Sind die ersten Zeichen von F beispielsweise „PENIBEL", dann wird bei Eröffnung der Datei das P in F^ übertragen. Der erste read-Befehl ordnet das Zeichen P einer Variablen zu und ersetzt es dann im Buffer F^ durch das E.

```
            get ( F );
            fertig := EoF ( F );
            IF NOT fertig THEN
              fertig := F^ > Leerzeichen
        END
    END; (* LeerzeichenUeberspringen *)
```

Beachten Sie, daß diese Routine in einer Textdatei nicht nur alle Leerzeichen überspringt, sondern auch die Zeichen der Zeilen-Endmarkierung. Sollen nur die Leerzeichen einer Zeile übersprungen werden, muß die Bedingung folgendermaßen geändert werden:

```
fertig := EoLn ( F ) OR ( F^ > Leerzeichen )
```

Der Ablauf für die Beseitigung aller „leeren" Zeichen sieht daher so aus:

```
REPEAT
  LeerzeichenUeberspringen ( F );
  IF NOT EoF ( F ) THEN
    TextGefunden := NOT EoLn ( F )
UNTIL TextGefunden OR EoF ( F )
```

Mit dieser letzten Änderung lassen sich leicht interaktive Programme schreiben, die Leereingaben ausschließen:

```
REPEAT
  write ( 'Daten eingeben :' );
  LeerzeichenUeberspringen ( input )
UNTIL NOT EoLn ( input )
```

Zwar kann diese Routine nicht die Eingabe von Steuerzeichen abfangen, die das Dateiende markieren, doch können wir dafür immer auf die vorherige Prozedur zurückgreifen.

Die Prozedur „assign" wird inzwischen fast als „Standarderweiterung" angesehen und findet – wie „open" und „seek" für Random-Access-Dateien – vermutlich schon bald Eingang in alle PASCAL-Versionen. Zwei oft verwandte Erweiterungen sind auch:

```
FUNCTION Fstat ( DateiName )
```

die den booleschen Wert „true" liefert, wenn eine Datei bereits besteht, und

```
PROCEDURE Rename ( AlterName,
  NeuerName )
```

ermöglicht die Umbenennung von Dateien.

PASCAL-Dialekte

PASCAL kennt keine Beschränkungen in der Zahl der Dateien, die gleichzeitig eröffnet sein können. Da Dateien jedoch über das Betriebssystem des Computers aktiviert werden, gibt es hier Abweichungen von der strikten Einheitlichkeit der Sprache. So existieren PASCAL-Versionen, die Dateien nicht unterstützen.

Auch die von Betriebssystemen vorgegebenen Namensstrukturen für Dateien können Probleme verursachen. PASCAL-Versionen von Großrechnern nehmen beispielsweise die ersten zehn Zeichen eines Dateinamens und durchsuchen damit den Speicher nach der zugehörigen Datei. Viele Systeme verfügen jedoch nicht über diese Möglichkeit. Beispiele dafür sind: A:CP/M-FIL.DAT, #4:APPLEFOR.MAT und O:P.BBCFILE. Die „Standarderweiterung" für den korrekten Aufbau eines PASCAL-Dateinamens ist die Prozedur „assign". Sie wird vor „reset" oder „rewrite" eingesetzt:

```
assign ( Dateizuordnung, DateiName )
```

Für unser Programm genügt daher: assign (Datei, D:DateiNam.dat).

Das Programm „CopyText"

Die Zuordnung „text" kennzeichnet einen in PASCAL häufig eingesetzten Datei-Typ. Die Deklaration einer Textdatei-Variablen geschieht in der Parameterliste des Programmkopfes. Das folgende Programm kopiert Textdateien. Damit es sich für eine Vielzahl von Anwendungen einsetzen läßt, haben wir den Kopiervorgang als Prozedur angelegt.

Bedenken sie, daß Compiler, die nicht standardmäßig ausgelegt sind, die Anweisung „reset (F1,'Quelle')" im Hauptprogramm benötigen. Beachten Sie weiterhin, daß beide Dateien als VAR-Parameter an die Prozedur Copy übergeben werden und damit nicht nur die Zieldatei aktualisiert, sondern auch die Quelldatei gelesen und vom Status her verändert wird. Dateivariablen werden immer als Adreß-Parameter und nie als Wertparameter übergeben, da die letzteren immer lokale Variablen sind. Bei Dateien, deren Größe den verfügbaren Arbeitsspeicher überschreitet, muß ein Einlesen mit Wertvariablen daher immer zum Absturz des Programms führen. Aus Gründen der Platzersparnis wird hier jedoch für umfangreiche Strukturen vom Typ Array und Record oft eine Ausnahme gemacht.

```
PROGRAM CopyText ( F1, F2 );

VAR
  F1,
  F2 : text;

  (* ------------------------- *)
PROCEDURE Copy ( VAR Quelle,
                     Ziel   : text );
VAR
    Zeichen : char;
BEGIN
  WHILE NOT EoF ( Quelle ) DO
    BEGIN (* eine Zeile kopieren *)
      WHILE NOT EoLn ( Quelle ) DO
        BEGIN
          read ( Quelle, Zeichen );
          write ( Ziel, Zeichen )
        END;
      (* jetzt das Zeilenende kopieren: *)
      ReadLn ( Quelle );
      WriteLn ( Ziel )
    END
END; (* Copy *)
  (* ------------------------- *)

BEGIN (* CopyText — Hauptprogramm *)

  assign ( F1, 'Quelle' );
  reset ( F1 ); (* Datei finden und zum Lesen eroeffnen *)
  assign ( F2, 'Ziel' );
  rewrite ( F2 ); (* Datei anlegen *)
  Copy ( F1, F2 )

END.
```

. . . praktisch, gut

Der PX-8 „Lap-Held" wird von Epson hergestellt, dem Unternehmen, das wegen seines Matrix-Druckers sowie seines HX-20-Portables und seines QX-10-Büro-Computers bekannt geworden ist. Der PX-8 wird mit 64 K RAM geliefert, hat einen LCD-Schirm, ist CP/M-lauffähig und wird mit mehreren Programmen ausgeliefert.

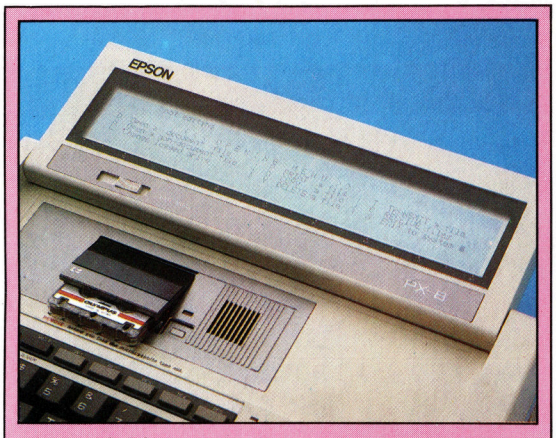
Tragbare Computer können vergleichsweise große „Koffermaschinen" sein oder so klein, daß sie in eine Tasche passen. Zwischen beiden Extremen liegen die „Lap-Helds". Eine der neueren Entwicklungen in diesem Bereich ist Epsons PX-8, ein tragbarer Rechner mit 64 K RAM, CP/M und einem umfangreichen Software-Paket.

Der PX-8 wird in einem Karton von der Größe eines Telefonbuchs geliefert und wiegt etwa 2,3 kg. Das Gehäuse ist in zwei Beige-Tönen gehalten und mit einem Metalltraggriff versehen. Auf den ersten Blick scheint das Ganze wenig mit einem Computer zu tun zu haben. Nimmt man aber einen Teil des Gehäuses ab, treten eine komplette Computertastatur und ein klappbarer Bildschirm zutage. Durch Betätigung eines Gleitriegels mit der Aufschrift „UNLOCK" wird der Bildschirm aus der Transporthalterung gelöst. Zugleich wird ein Microcassetten-Recorder freigegeben. Der Bildschirm läßt sich in elf verschiedene Positionen stellen, obwohl nur fünf oder sechs einen guten Blickwinkel gewährleisten.

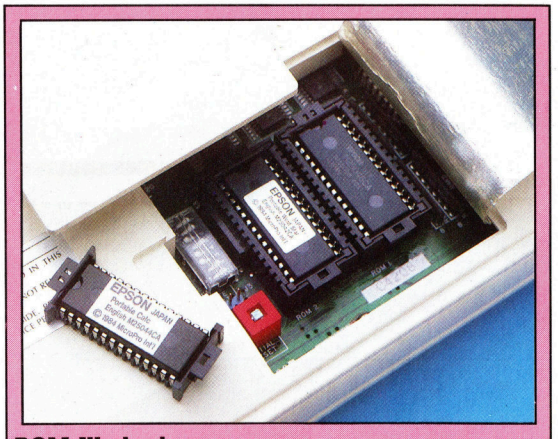
Die Tastatur verfügt über 72 Schreibmaschinentasten, die – entsprechend ihrer Verwendung – farblich codiert sind. Die dunkelbraunen alphanumerischen Tasten sind in der üblichen QWERTZ-Form angeordnet. Durch Änderung der DIP-Schalter des PX-8 hat man Zugriff zum „internationalen" Zeichensatz. Wie das funktioniert, wird im Benutzerhandbuch genau erläutert. Ferner gibt es vier orangefarbene Cursorsteuerungstasten, Insert-, Delete- und Home-Tasten, drei System-Funktions-Tasten (Escape, Pause und Help) sowie fünf programmierbare Funktionstasten.

Die Tastatur ist hochwertig und besonders einfach in der Handhabung, wenn man den Rechner auf dem Schoß hat. Arbeitet man aber mit dem PX-8 auf dem Schreibtisch, wird die Sache komplizierter, da die Tastatur viel Druck erfordert. Zwei ausklappbare Beine sind am System befestigt, die das Gerät kippen, aber das genannte Problem nicht lösen können. Die Caps-Lock-, Number- und Insert-Tasten sind Kippschalter, mit denen sich der Darstellungsmodus des PX-8 verändern läßt. Drei kleine rote LEDs zeigen an, in welchem Modus man sich befindet.

Die Epson-Dokumentation ist umfangreich und sehr gut geschrieben. Zwei dicke Handbü-



Großer Schirm
Der LCD-Schirm des PX-8 stellt acht Zeilen mit jeweils 80 Zeichen dar. Der Betrieb erfolgt über Batterie. Die maximale Auflösung des Bildschirms beträgt 480 mal 64 Punkte.



ROM-Wechsel
Nach Entfernen eines kleinen Deckels auf der Unterseite des Rechners hat man Zugriff zu den Steckschächten, die die ROM-Software aufnehmen. Das CP/M-Betriebssystem ist integriert.

cher werden mitgeliefert. Beim ersten handelt es sich um ein Benutzer-Handbuch von mehreren hundert Seiten, in dem die Inbetriebnahme des Rechners erläutert wird, ferner die Benutzung von Hard- und Software sowie CP/M-Operationen. Das Handbuch beinhaltet ebenfalls Speicherverzeichnisse, vollständige Listen der verfügbaren Zeichen und ihrer entsprechenden Codes und ein relativ langes Maschinenprogramm zum Speichern und Laden des Grafik-Bildschirms von Diskette.

Beim PX-8 wird ein Z80-kompatibler CMOS als Zentraleinheit verwendet. CMOS-(Complementary Metal Oxide Semiconductor) Chips benötigen erheblich weniger Spannung als übliche Zentraleinheiten. Da zudem ein LCD-Schirm (ebenfalls energiesparend) benutzt wird, kann das System vollständig im Batteriebetrieb verwendet werden. Zwei Batterieeinheiten werden mitgeliefert – eine für die Hauptstromversorgung, die andere als Reserve. Die Batterien müssen vor Inbetriebnahme geladen werden, was eine Wartezeit von acht Stunden bedeutet, bevor man erstmals an den Rechner gehen kann. Die Haupteinheit ist wiederaufladbar und ermöglicht eine Betriebszeit von insgesamt 15 Stunden. Laut Herstellerangabe hat die Stromversorgung eine „Lebenszeit" von drei bis vier Jahren.

Nach Einschalten des PX-8 muß das Betriebssystem initialisiert werden. Die dazu erforderlichen Schritte sind ausführlich im Handbuch dargelegt. Unter anderem sind Tag, Datum und Zeit einzugeben. Der PX-8 kann einen Teil seines RAM – vom Benutzer wählbar zwischen neun und 24 KByte – als „Disketten"-Speicher abzweigen. Das Betriebssystem behandelt diesen Speicherbereich tatsächlich so, als handle es sich um eine externe Diskettenstation. Vor Benutzung muß die RAM-Diskette formatiert und die RAM-Kapazität, die man für das betreffende Programm benötigt, spezifiziert werden. Epson bietet ferner eine ergänzende RAM-Disketten-Einheit mit 120 KByte an.

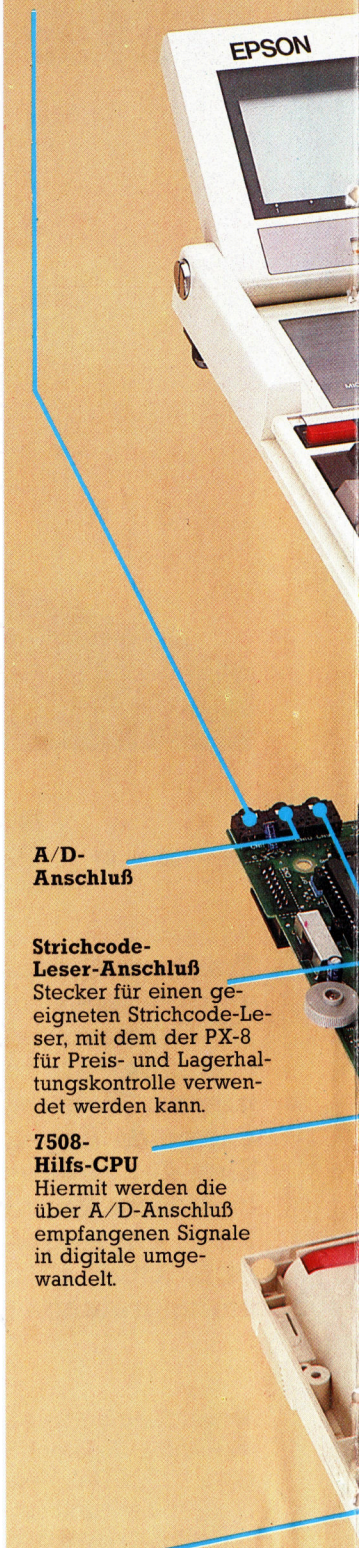
Nach Berücksichtigung dieser Details lädt der PX-8 das CP/M-Betriebssystem aus dem ROM und stellt die CP/M-Ultilities sowie das ROM-Inhaltsverzeichnis dar. Software kann in allen drei Formaten benutzt werden: Cassette, Diskette oder ROM. ROM-Software ist auf bzw. in EPROM-Chips erhältlich, die in eine Fassung auf der Unterseite des Geräts gesteckt werden. Die mit dem PX-8 gelieferte Software – Handy-Text und Handy-Calc – wird wie der BASIC-Interpreter als ROM angeboten.

Langsamer Bildschirm

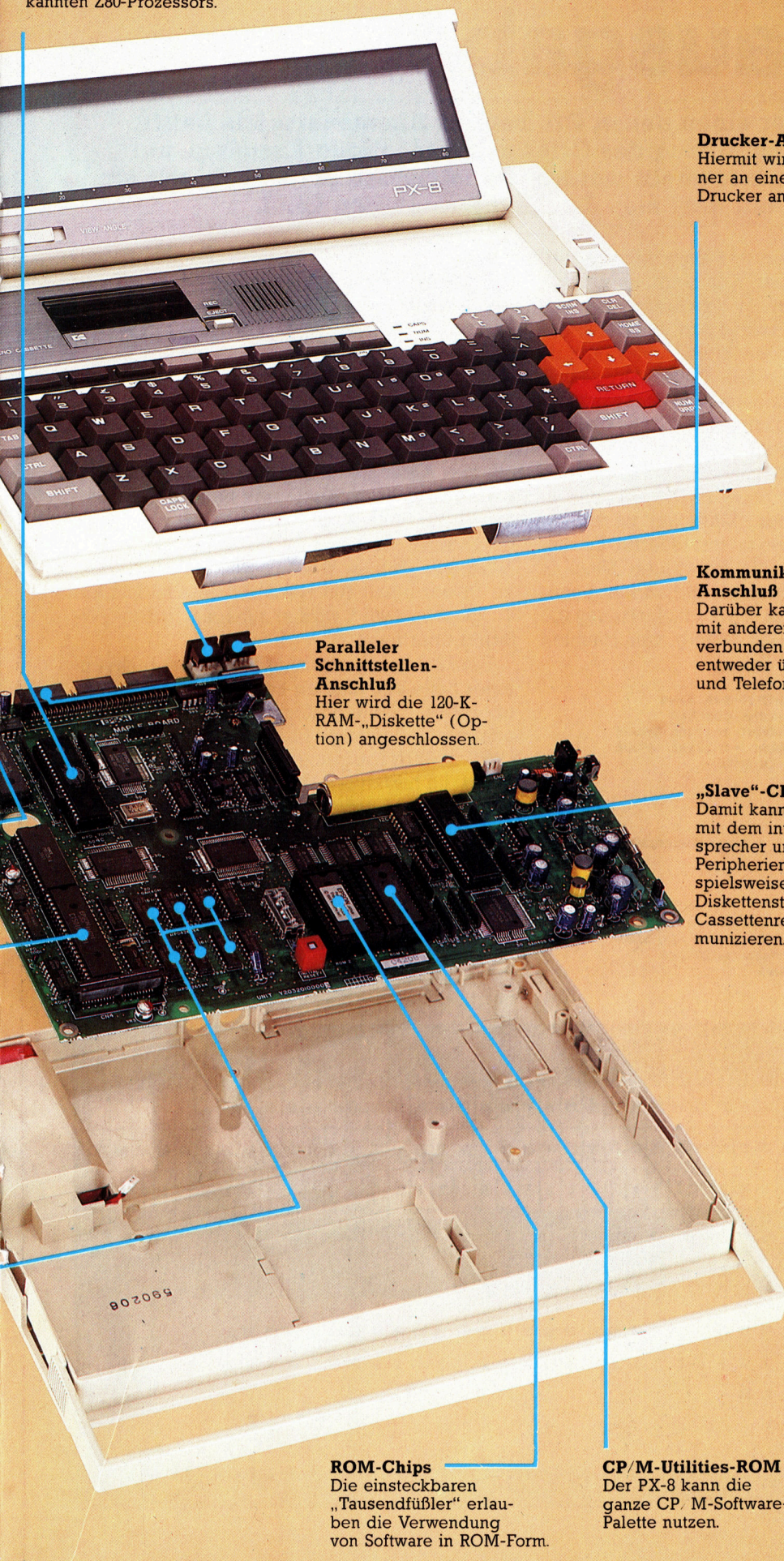
Der LCD-Schirm stellt acht Zeilen mit 80 Zeichen bei einer Grafikauflösung von 480 mal 64 Punkten dar. Schwachpunkt dieses Systems ist der Bildschirm; denn er arbeitet extrem langsam. Die Zeichen erscheinen nach der Eingabe rasch, die Korrekturen dagegen – vor allem, wenn es sich um ganze Wörter oder Sätze handelt – kommen langsam.

Die mit dem PX-8 gelieferte Software ist umfangreich. Ergänzend zu den Programmen Wordprocessor, Spreadsheet und Database, bietet Epson auch ein Telekommunikations-Programm für den Modem-Betrieb an sowie ein Programm, das die Übertragung von Dateien vom PX-8 auf größere Systeme wie etwa den Epson QX 10 ermöglicht. Da es sich beim PX-8 um einen CP/M-Rechner handelt, ist das Gros der verfügbaren CP/M-Software darauf lauffähig – aber nicht jedes Programm.

Beim PX-8-BASIC handelt es sich um ein von Epson erweitertes Microsoft, das unter anderem eine automatische Zeilen-Numerierung und Neu-Numerierung beinhaltet, ferner einen Bildschirm-Editor, Grafik- und Sound-Befehle, Statements, die die Kommunikation über die eingebaute RS232-Schnittstelle unterstützen, und Befehle, die den Betrieb des Micro-Cassettenrecorders als Diskettenstation behandelt. So kann direkt auf den Speicher zugegriffen werden.



CPU
CMOS-Version des bekannten Z80-Prozessors.



Drucker-Anschluß
Hiermit wird der Rechner an einen externen Drucker angeschlossen.

Kommunikations-Anschluß
Darüber kann der PX-8 mit anderen Computern verbunden werden – entweder über Modem und Telefon oder direkt.

Paralleler Schnittstellen-Anschluß
Hier wird die 120-K-RAM-„Diskette" (Option) angeschlossen.

„Slave"-CPU
Damit kann der PX-8 mit dem internen Lautsprecher und externen Peripherien wie beispielsweise Drucker, Diskettenstation oder Cassettenrecorder kommunizieren.

ROM-Chips
Die einsteckbaren „Tausendfüßler" erlauben die Verwendung von Software in ROM-Form.

CP/M-Utilities-ROM
Der PX-8 kann die ganze CP/M-Software-Palette nutzen.

Epson PX-8

ABMESSUNGEN
297 × 216 × 48 mm

ZENTRALEINHEIT
Z80-kompatible CMOS-CPU, 2,4 MHz

SPEICHER-KAPAZITÄT
64 K RAM, 32 K ROM plus 6 K Video-RAM

BILDSCHIRM-DARSTELLUNG
Text: 80 Zeichen x acht Zeilen; Grafik: 480 × 64 Punkte

SCHNITTSTELLEN
RS232C, seriell, Strichcode-Leser, Analog-Eingabe

PROGRAMMIER-SPRACHEN
Erweitertes Microsoft-BASIC, betrieben mit CP/M.

TASTATUR
72 Schreibmaschinentasten, QWERTZ-Format, einschließlich Cursorsteuerungstasten und fünf programmierbaren Funktionstasten. Zwölf Tasten können als numerische Tastatur umdefiniert werden.

HANDBÜCHER
Zwei umfangreiche Bände, ein Betriebs-Handbuch sowie ein BASIC-Referenz-Buch. Ausführlich und hervorragend geschrieben.

STÄRKEN
Breiter LCD-Schirm (80 Zeichen). Die ROM-Software vereinfacht das Laden von Programmen. Leicht erweiterbar.

SCHWÄCHEN
Der LCD-Schirm stellt nur acht Zeilen dar und ist im Betrieb langsam. Trotz der guten Dokumentation ist CP/M nicht sehr anwenderfreundlich, vor allem nicht für Erstbenutzer.

Textausgabe

Sowie der Spieler einen neuen Ort unserer Abenteuerspiele betritt, müssen die Beschreibung des Ortes sowie mögliche Ausgänge auf dem Bildschirm dargestellt werden. Im folgenden entwickeln wir ein Hilfsprogramm, mit dem man den Text formatieren kann.

Digitaya" und „Haunted Forest" sind textorientierte Abenteuer, das heißt, es werden Wörter zur Beschreibung der Ereignisse verwendet. Benutzt man hierzu nun PRINT-Anweisungen, so kann das manchmal unschön aussehen. Ist der Text einer PRINT-Anweisung länger als eine Bildschirmzeile, so wird der restliche Text rücksichtslos in die nächste Zeile geschrieben, egal ob dabei Wörter getrennt werden. Eine umständliche Lösung des Problems wäre nun, jede PRINT-Anweisung manuell zu formatieren. Einfacher ist es, die Texte mit einer Routine zu formatieren. Zu diesem Zweck müssen die jeweiligen Sätze in Form einer String-Variablen gehandhabt werden, damit die Routine die Bildschirmausgabe kontrollieren kann. Das Listing von Haunted Forest sieht wie folgt aus:

Die Formatier-Routine bei Haunted Forest und Digitaya ermöglicht es, Texte so darzustellen, daß an den Zeilenenden Wörter nicht mehr getrennt werden. Unter Verwendung der Variablen OW$ und NW$ „betrachtet" die Routine jeweils das nach dem auszugebenden Text befindliche Wort. Reicht das nächste Wort über das Zeilenende hinaus, so erfolgt die Ausgabe ohne Semikolon, und es wird eine neue Zeile begonnen.

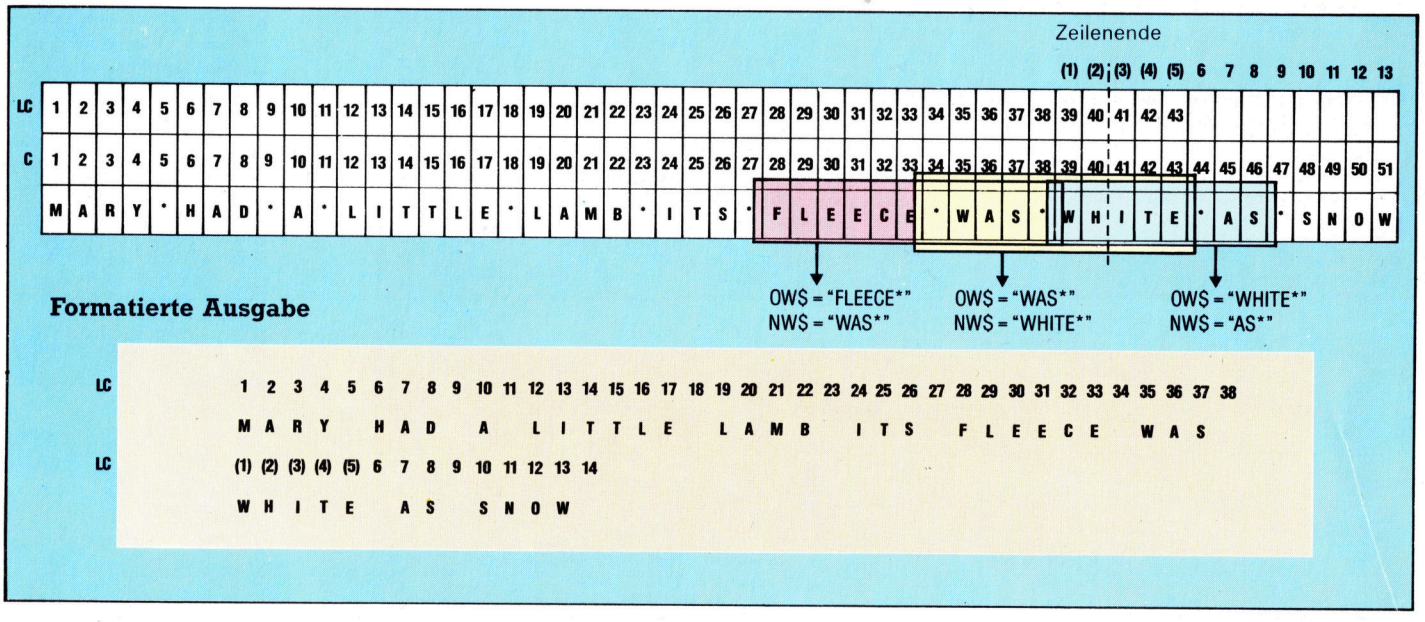
```
5500 REM **** FORMAT OUTPUT S/R ****
5510 LC=0:     REM CHAR/LINE COUNTER
5520 OC=1:     REM OLD COUNT INITIAL VALUE
5530 OW$="":   REM OLD WORD  INITIAL VALUE
5540 LL=40:    REM LINE LENGTH
5550 SN$=SN$+" DUMMY "
5560 PRINT
5570 FOR C=1 TO LEN(SN$)
5580 LC=LC+1
5590 IF MID$(SN$,C,1)=" " THEN GOSUB5800
5600 NEXT C
5605 PRINT
5610 RETURN
5620 :
5800 REM ** END OF LINE CHECK S/R **
5810 NW$=MID$(SN$,OC,C-OC+1):REM NEW WORD
5820 IF LC<LL THENPRINTOW$;:GOTO5840
5830 PRINTOW$:LC=LEN(NW$)
5840 OC=C+1:OW$=NW$
5850 RETURN
```

Sowohl Digitaya als auch Haunted Forest verwenden spezielle Routinen zur Formatierung der Bildschirmtexte, die nachfolgend erläutert werden.

Als erstes durchsucht die Routine den Satz in der Variablen SN$ nach einer Leerstelle. Wird eine solche gefunden, so wird die Unterroutine in Zeile 5800 aufgerufen. Diese Routine führt einige wichtige Aufgaben aus. Sie verwendet OC um den Anfang eines Wortes anzuzeigen, und C zum Speichern des gerade bearbeiteten Zeichens. Dadurch kann das vor einer Leerstelle gefundene Wort mittels MID$ isoliert und in NW$ (für „Neues Wort") gespeichert werden. Bevor der Inhalt von NW$ auf dem Bildschirm ausgegeben wird, wird er zunächst in den OW-String übertragen.

Ein Zähler, LC, wird zum Zählen der bisher in einer Zeile enthaltenen Zeichen verwendet. Er überprüft in Zeile 5820, ob er kleiner ist als die erlaubte Zeilenlänge LL. Ist dies der Fall, wird OW$ gePRINTet, gefolgt von einem Semikolon, damit die folgenden Wörter in derselben Zeile ausgegeben werden. Ist LC größer als LL, dann wird ebenfalls OW$ gePRINTet, jedoch ohne Semikolon (wodurch die folgenden Wörter in der nächsten Bildschirmzeile dargestellt werden). Außerdem wird der Zähler LC auf die Länge des neuen Wortes gesetzt.

Betrachten wir nun, wie die Routine in der Praxis arbeitet. Zuerst durchsucht sie einen Satz nach Leerstellen. Wird eine gefunden, so wer-



den die Zeichen zwischen ihr und der zuvor gefundenen Leerstelle als neues Wort zusammengefaßt. Die Routine überprüft also jeweils das Wort, das nach dem auszudruckenden Begriff steht. Dabei wird die maximale Wortlänge überpruft. Ist das Wort zu lang, so wird eine neue Zeile angefangen. Eine Teilung von Wörtern wird somit vermieden. An das Ende jeden Satzes wird „ DUMMY " angefügt. Dies ist notwendig, da so ein letztes Wort in NW$ gespeichert werden kann. Die Leerstelle vor „ DUMMY " kennzeichnet es als separates Wort. Die zweite dient dazu, daß die Routine noch eine letzte Leerstelle finden kann.

Die Formatierungsroutine

Nehmen wir als Beispiel den Satz „Mary had a little lamb its fleece was white as snow". Als Zeilenbreite nehmen wir 40 Zeichen an. Bei unformatierter Ausgabe würde das Wort „white" gespalten, so daß die Buchstaben „ite" in der nächsten Zeile erscheinen würden. Die Formatierroutine untersucht nun jeweils zwei Wörter parallel. Betrachtet man die beiden Wörter vor „white", würde „fleece" in OW$ und „was" in NW$ gespeichert. Nach Überprüfung, daß der Wert von LC nicht größer als 40 ist, wird OW$ gefolgt von einem Semikolon gePRINTet. Danach wird „was" von NW$ in OW$ übertragen, und die Routine findet das Wort „white". Jetzt ist der Wert von LC größer als 40, so daß das Wort „white" über das Zeilenende reichen würde. In diesem Fall wird OW$ (der nun „was" enthält) trotzdem gePRINTet, jedoch ohne Semikolon. LC wird nun auf den Wert der Anzahl der Zeichen dieses Wortes gesetzt. Das Wort „white" wird in OW$ übertragen und in der nächsten Zeile ausgegeben.

Um die Routine zu testen, verwenden wir sie zur Ausgabe der Spieleinführung. Dabei können unter Verwendung der Variablen SN$ ein Satz von bis zu 248 Zeichen zusammengestellt und die besprochene Unterroutine aufgerufen werden. Geben Sie die folgenden Zeilen ein:

```
1000 REM **** STORY SO FAR S/R ****
1010 SN$="WELCOME TO THE HAUNTED FOREST"
1020 GOSUB5500:REM FORMAT
1030 PRINT
1040 SN$="AS YOU AWAKE FROM A DEEP SLEEP, THE "
1050 SN$=SN$+"FOREST FLOOR FEELS SOFT AND DRY. "
1060 SN$="YOU DO NOT KNOW HOW YOU CAME TO BE HERE "
1070 SN$=SN$+"BUT KNOW THAT YOU MUST FIND THE "
1080 SN$=SN$+"VILLAGE ON THE EDGE OF THE WOOD TO "
1090 SN$=SN$+"REACH SAFETY."
1100 GOSUB5500:REM FORMAT
1110 PRINT
1120 SN$="YOU LOOK AROUND, TRYING TO GET YOUR BEAR
INGS."
1130 GOSUB5500:REM FORMAT
1140 PRINT:PRINT"PRESS ANY KEY TO START"
1150 GET A$:IF A$="" THEN 1150
1160 PRINTCHR$(147):REM CLEAR SCREEN
1170 RETURN
```

Außerdem braucht man die folgenden Zeilen, um diese Unterroutine aufzurufen:

```
205 GOSUB 1000: REM STORY SO FAR
990 END
```

Digitaya Listings

```
1110 GOSUB1250:REM STORY SO FAR
1270 END

1290 REM **** STORY SO FAR ****
1300 SN$="WELCOME TO 'DIGITAYA'"
1310 GOSUB5880:REM FORMAT
1320 PRINT
1330 SN$="AS THE MACHINE HUMS QUIETLY. YOU LOOK
AROUND."
1340 SN$=SN$+" TO THE NORTH AND SOUTH STRETCHES
A WIDE HIGHWAY."
1350 SN$=SN$+" YOUR MISSION IS TO FIND THE
MYSTERIOUS DIGITAYA"
1360 SN$=SN$+" AND CARRY IT TO SAFETY THROUGH
ONE OF THE OUTPUT PORTS."
1370 SN$=SN$+".. BUT WHICH ONE ?"
1380 GOSUB5880
1390 PRINT:PRINT"PRESS A KEY TO START"
1400 GETA$:IFA$=""THEN1400
1410 PRINTCHR$(147):REM CLEAR SCREEN
1420 RETURN

5880 REM **** FORMAT PRINTING S/R ****
5890 LC=0:  REM CHAR/LINE COUNTER
5900 OC=1:  REM OLD COUNT
5910 OW$="":REM OLD WORD
5920 LL=40:REM SCREEN LINE LENGTH
5930 SN$=SN$+" DUMMY "
5940 PRINT
5950 FOR C=1 TO LEN(SN$)
5960 LC=LC+1
5970 IF MID$(SN$,C,1)=" " THENGOSUB6020
5980 NEXTC
5990 PRINT
6000 RETURN
6010 :
6020 REM **** END OF LINE CHECK S/R ****
6030 NW$=MID$(SN$,OC,C-OC+1)
6040 IF LC<LL THENPRINTOW$;:GOTO6060
6050 PRINTOW$:LC=LEN(NW$)
6060 OC=C+1:OW$=NW$
6070 RETURN
```

BASIC-Dialekte

Spectrum:
Für das Digitaya-Listing andern Sie die Formatier-Routine wie folgt ab.

Ersetze SN$ durch S$, OW$ durch O$, NW$ durch N$.

5920 LET LL=32: REM SCREEN LENGTH LINE
5970 IF S$(C TO C)=" " THEN GOSUB 6020
6030 LET N$=S$ (OC TO C)

In der Einführungsunterroutine ersetzen Sie SN$ durch S$.

1400 IF INKEY$="" THEN 1400
1410 CLS

Im Haunted-Forest-Listing ersetzen Sie ebenfalls die eben angegebenen String-Variablennamen sowie die folgenden Programmzeilen.

5540 LET LL=32: REM SCREEN LINE LENGTH
5590 IF S$ (C TO C)=" " THEN GOSUB 5800
5810 LET N$=S$ (OC TO C)

und

1150 IF INKEY$="" THEN 1150
1160 CLS

Acorn B:
In der Einführungsunterroutine müssen beim Digitaya-Listing folgende Änderungen durchgeführt werden:

1095 MODE 1
1400 A$=GET$
1410 CLS

sowie bei Haunted-Forest:

1160 CLS

Die Alternative

Bisher mußten C 64-Besitzer meist mit Commodore-eigenen Speichergeräten vorliebnehmen, die nicht komfortabel sind und deren Speicherkapazität limitiert ist. Die Situation hat sich mit Einführung von Phonemarks 8500 Quick Data Drive entscheidend verbessert.

Es gibt nur wenige Drittanbieter, die Speicheralternativen zu Commodores Datasette und Diskettenstation herstellen, trotz der bekannten Kritikpunkte. Der Quick Data Drive ist ein „Stringy Floppy System", das mit einer Endlos-Schleife (identisch mit der beim Rotronics Wafadrive) arbeitet.

Da viele Heimcomputer-Besitzer nach schnelleren und effizienteren Zugriffsmöglichkeiten suchen, produzieren viele Hersteller verschiedene Massenspeicher für die derzeit populärsten Computersysteme. Darunter ist auch der Phonemark 8500 Quick Data Drive von Dean Electronics, für den C 64 und den VC 20 entwickelt.

Das Gerät ist dem Wafadrive von Rotronics für den ZX Spectrum sehr ähnlich. Die Laufwerke beider Systeme wurden von BSR Electronics entwickelt und benutzen identische „Wafer".

Die normale Commodore-1541-Diskettenstation ist – wie der Recorder – unglaublich langsam. Das liegt nicht an der Diskettenstation selbst, sondern an der Methode, wie Daten in den Computer geladen werden. Das Betriebssystem des C 64 ist im wesentlichen von dem des PET abgeleitet (der Mitte der siebziger Jahre auf den Markt kam), als Massenspeicher – besonders Cassettenrecorder – noch recht unzuverlässig waren. Als Commodore sich mit Massenspeichern für den PET befaßte, beschloß man, einen eigenen Cassettenrecorder (Datacorder) zu entwickeln, der beim Laden zugleich die Datenübertragung auf ihre Richtigkeit überprüfte. Das verbesserte die Zuverlässigkeit des Ladens – allerdings zu Lasten der Geschwindigkeit. Das Verfahren wurde für den VC 20 und den C 64 übernommen.

Heute hat sich die Qualität von Cassetten entscheidend verbessert, und die Notwendigkeit langer und komplizierter Datenkontrolle auf Commodore-Rechnern ist nicht mehr gegeben. Viele im Handel befindlichen Programme sind mit eigenen Laderoutinen versehen, die – bei gleicher Zuverlässigkeit der Übertragung – die Überprüfungen überflüssig machen und den Ladevorgang erheblich beschleunigen.

Beim Laden eigener Programme haben die meisten Anwender allerdings keinen Zugriff zu diesen Hochgeschwindigkeitstechniken und müssen sich mit den Verzögerungen des Commodore-Betriebssystems abfinden. Der Quick Data Drive, für den der Hersteller eine 15fach schnellere Ladegeschwindigkeit als bei Cassette und eine größere Ladegeschwindigkeit als bei der 1541-Diskettenstation angibt, kann als Alternative zur Commodore-eigenen Peripherie gesehen werden.

Der Quick Data Drive ist recht klein – nicht einmal halb so groß wie die Datasette von Commodore. Anders als der Rotronic Wafadrive verfügt der Quick Data Drive nur über ein einzelnes Laufwerk.

In Reihe schalten

Obwohl das Laufwerk an den Cassettenport angeschlossen wird, kann gleichzeitig die Datasette betrieben werden. Es gibt einen Anschluß auf der Rückseite des Data Drive, mit dem der gemeinsame Betrieb einer Datassette oder eines zweiten Data Drive (in Reihe geschaltet) möglich ist.

Das Quick Data Drive Operating System (QOS) befindet sich auf einem Wafer. Um dieses in den Computer zu laden, drückt man Shift/Run (wie beim Laden einer normalen Cassette). Sobald auf dem Schirm die Meldung „PRESS PLAY ON TAPE" erscheint, drückt man einen kleinen Knopf auf der Rückseite des Gerätes, womit der Autoboot erfolgt. Danach ist das QOS arbeitsbereit.

Die Programme, aus denen das QOS besteht, werden in zwei verschiedene Speicherbereiche geladen. Zunächst werden die Ma-

schinencode-Routinen für LOADen, SAVEn und Suchprogramme zwischen den Adressen C000 und CFFF in den oberen Speicherbereich geladen (normalerweise für Programme in Maschinencode reserviert).

Obwohl das QOS über keine eigenen Befehle verfügt (es benutzt die im Commodore-Betriebssystem vorhandenen), schaltet es normale Laderoutinen einfach ab und fügt seine eigenen ein.

Der andere Teil des Quick Operating System ist die sogenannte File Management Utility (FMU), die eine Reihe nützlicher Routinen enthält. Diese wird in der oberen Hälfte der acht KByte des „Shadow-Speichers" unter dem BASIC-ROM abgelegt – zwischen den Adressen B000 und AFFF. Die unteren vier KByte zwischen A000 und AFFF werden vom FMU als sequentieller Datenbuffer benutzt.

Entweder, oder

Da das FMU im Speicherbereich unter dem BASIC-ROM abgelegt ist, kann man nicht beide gleichzeitig benutzen. Um es aus dem Speicher aufzurufen, muß der Befehl LOAD "FMU" gegeben werden. Damit werden das BASIC „abgeschaltet" und das FMU aktiviert.

Das Betriebssystem des Quick Data Drive ist zweifellos erheblich schneller, als es bei den üblichen Cassettenrecordern der Fall ist. Für den Benchtest haben wir ein Simulationsspiel benutzt. Das komplette 25-KByte-Programm, für dessen Laden von Cassette üblicherweise über neun Minuten und anderthalb Minuten für das Laden von Diskette erforderlich sind, brauchte nur 30 Sekunden, um vom Quick Data Drive geladen zu werden. Doch wie bei allen Systemen, die mit Bandschleifen arbeiten, hängt diese Zeitspanne davon ab, wo sich der Schreib-Leseknopf im Verhältnis zum Programmbeginn befindet.

Das Quick Operating System findet die gewünschten Datenfiles durch Überprüfung der Kopfblöcke auf dem Band. Beim Formatieren eines Wafers teilt das Betriebssystem das Band in Blöcke, die mit eigenen File-Namen versehen werden. Lädt man nun ein File in den Computer, sucht es so lange, bis es den Block findet, der den ersten Teil des Files enthält, lädt ihn und sucht dann den zweiten Block.

Ähnlich ist es mit dem Inhaltsverzeichnis des Wafer. Während der Lesekopf über das Band geht, liest das QOS jeden File-Namen und registriert die leeren Blocks und die, die Files enthalten. Sind alle File-Namen gelesen, zeigt das System die Liste der Files sowie die Gesamtmenge des verfügbaren Platzes.

Die File Management Utility ist eine menügeführte Reihe von Routinen, die Applikationen wie das Formatieren und Lesen des Wafer-Inhaltsverzeichnisses enthält. Sie beinhaltet ferner Kopierroutinen, die das Übertragen der Daten von Diskette, Cassette oder Wafer auf eine Back-Up-Wafer ermöglichen. Dies ist ein weiterer Vorteil des Quick-Data-Drive-Systems, da wohl nur wenige Programmierer ein Speichersystem kaufen würden, das bestehende Programme nicht auf den neuen Datenträger übertragen kann.

Natürlich hat das System auch Nachteile. Während die Kopierroutinen für BASIC-Programme und sequentielle Dateien ausgezeichnet arbeiten, gibt es bei bestimmten anderen Routinen (speziell im Maschinencode), die in einen bestimmten Speicherbereich geladen werden müssen, Probleme. Das liegt daran, daß die beiden Bereiche, die vom QOS und vom FMU belegt werden, auch für den Maschinencode benutzt werden. Lädt man Programme im Maschinencode, wird das Quick-Data-Drive-Betriebssystem überschrieben, und das Programm bricht zusammen.

Sollte der Quick Data Drive in ausreichenden Mengen verkauft werden, wird es sicherlich ein Programm geben, das diesen Mißstand behebt. Bis dahin hängt viel davon ab, ob Dean Electronics die Softwarehäuser davon überzeugen kann, Programme und Applikationen auch auf Wafer zu produzieren. Unter diesem Gesichtspunkt erweist sich der Wafadrive von Rotronics als wertvoller Verbündeter; denn Softwarehauser konnen Wafers nur dann zu günstigen Preisen herstellen, wenn größere Mengen benötigt werden. Dies könnte erreicht werden, wenn auch die Programme für den C 64 und den ZX Spectrum auf dasselbe Speichermedium gebracht werden.

Es ist aber noch zu früh, den Erfolg des Unternehmens einzuschätzen. Einige Programme, wie etwa der Epyx-Hit „Impossible Mission", sind bereits auf Wafer erhältlich.

Quick Data Drive

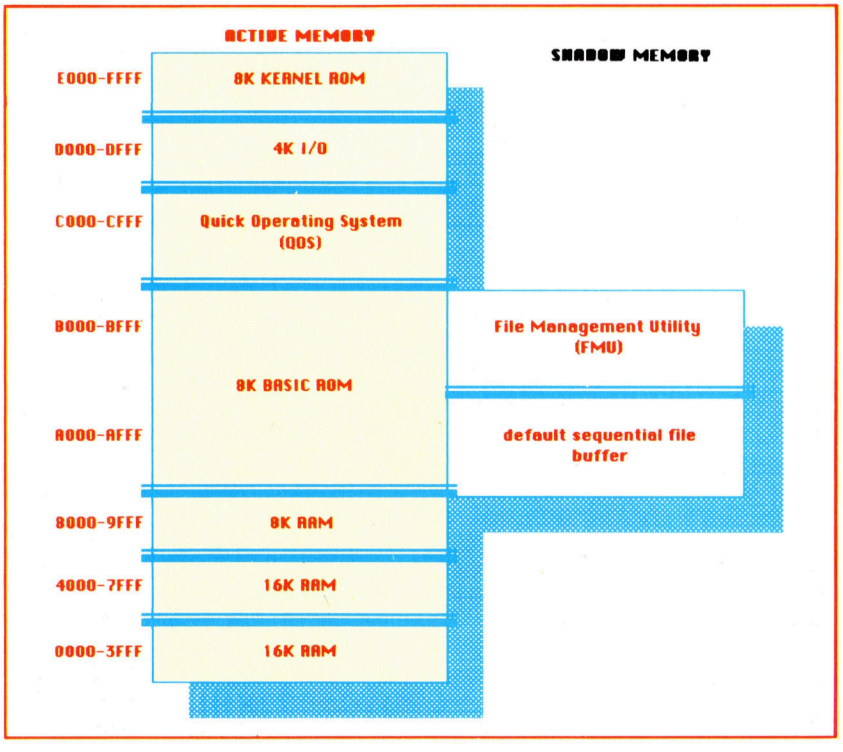
ABMESSUNGEN
147 × 118 × 49 mm

SCHNITTSTELLEN
Buchsen für den C 64/VC 20, Cassettenrecorderanschluß.

FORMAT
Endlosband, Floppy Wafer.

KAPAZITÄT
16, 64 und 128 KByte Wafer verfügbar.

Das Betriebssystem des Quick Data Drive ist nicht im ROM gespeichert, sondern muß von einem System-Wafer geladen werden. Die verschiedenen Komponenten werden in zwei verschiedenen Speicherbereichen abgelegt. Das erste, das Quick Operating System (QOS), wird in den Bereich geladen, der üblicherweise Maschinencode-Programmen vorbehalten ist. Der andere Teil des Systems, die File Management Utility (FMU), wird im „Shadow RAM" unter dem BASIC-ROM abgelegt. Hier befindet sich auch der Dateienbuffer, in den Programme vor dem Speichern geladen werden.



Das System

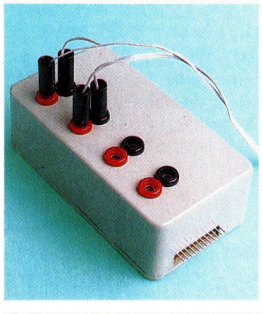
Wir haben ein User-Port-System zur Steuerung externer Geräte mit dem Commodore 64 und dem Acorn B entwickelt. Hier stellen wir Ihnen die Komponenten noch einmal in Kurzform vor.

Der Commodore 64 und der Acorn B kommunizieren über einen User Port mit der „Außenwelt", der bei beiden Computern über acht Datenleitungen und einen Masseanschluß verfügt. Die einzelnen Pins sind mit jeweils einem Bit im sogenannten „Datenregister" – einem speziellen Speicherplatz im Rechner – gekoppelt. Ein zweiter Speicherplatz, das Datenrichtungsregister (DDR), steuert die Richtung des Datenflusses: Ein auf Ausgabe geschalteter User-Port-Anschluß (DDR-Bit = 1) führt eine Spannung von +5 Volt, wenn das entsprechende Bit auf High (1) liegt. Liegt das Bit auf Low (0), geht die Spannung auf 0 Volt zurück. Diese Spannungen werden nicht direkt zum Betrieb externer Geräte verwendet, sondern dienen zur Ansteuerung von Relaisschaltungen, mit denen größere Spannungen sicher ein- und ausgeschaltet werden können.

Die Funktion einer auf Eingabe geschalteten User-Port-Leitung (DDR-Bit = 0) ist anders: In diesem Fall wird das entsprechende Bit im Datenregister auf High gehalten und geht nur auf Low, wenn die Leitung mit der Masse verbunden wird. Mit einem einfachen Schalter zwischen einer Datenleitung des User Ports und Masse können Vorgänge außerhalb des Computers erkannt werden: Bei geschlossenem Schalter geht das Bit im Datenregister auf Low.

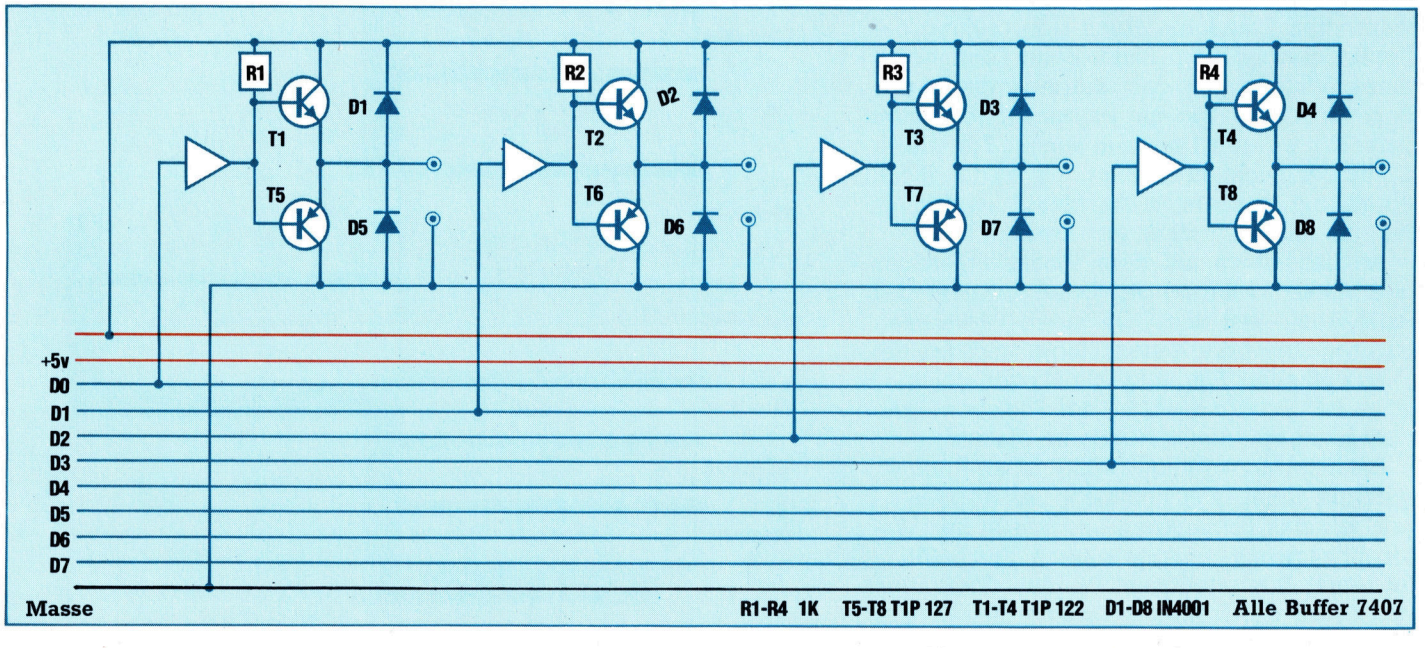
Die acht Datenleitungen und der Masseanschluß müssen auf unterschiedliche Weise mit den Schaltungen des User-Port-Systems verbunden werden. Wir haben das System daher mit einem gemeinsamen Datenbus (neun Leitungen) versehen, der bei allen Modulen über einen 12-poligen Minicon-Anschluß angeschlossen wird. Jedes Modul trägt auf der einen Seite einen Minicon-Stecker, auf der anderen eine Minicon-Buchse.

Neben den Schaltplänen finden Sie hier eine Kurzbeschreibung der Funktion aller Module. Einzelheiten des Aufbaus und die Bauteil-Listen finden Sie in den vorhergehenden Artikeln.



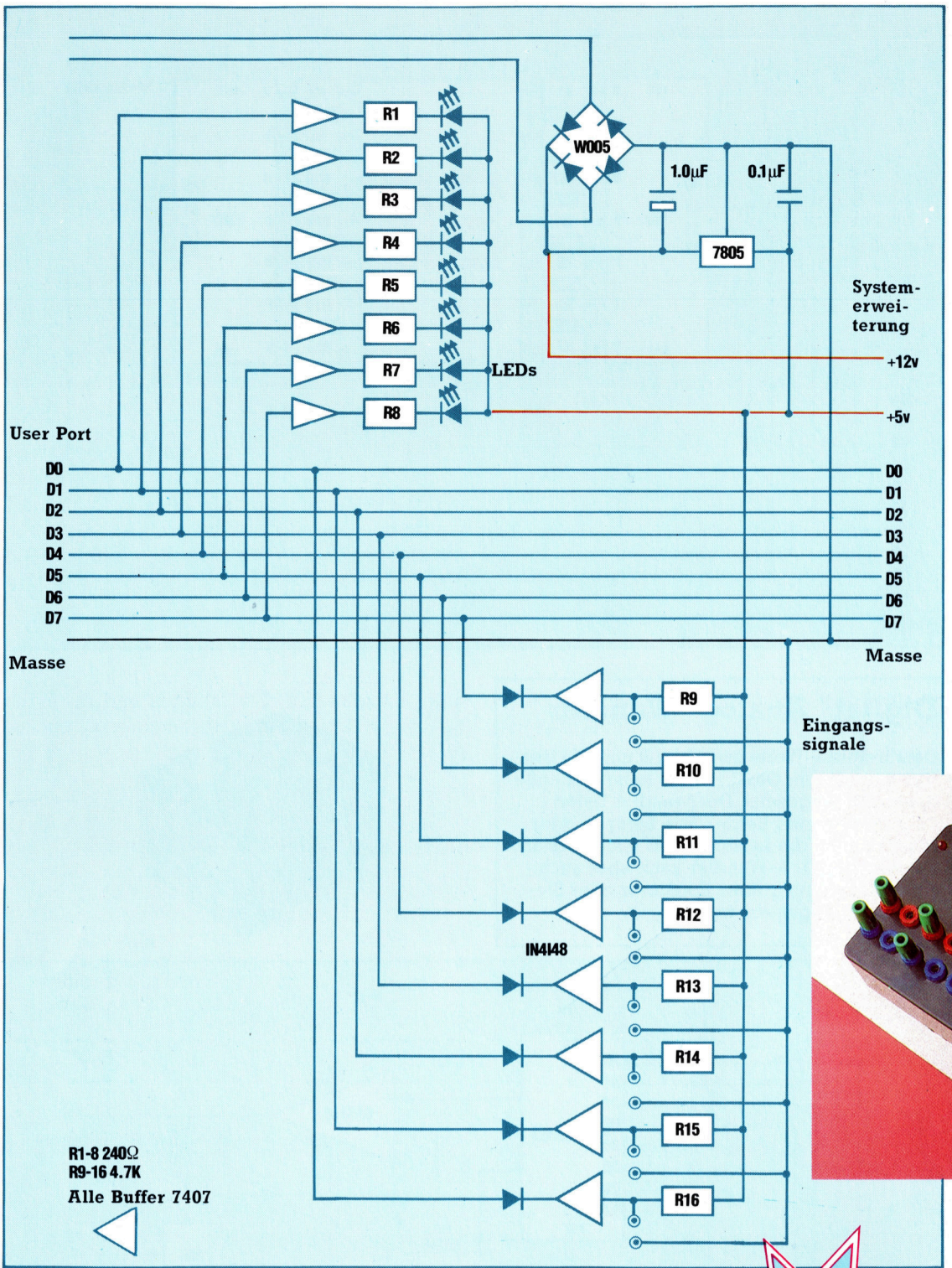
Ausgangs-Buffer

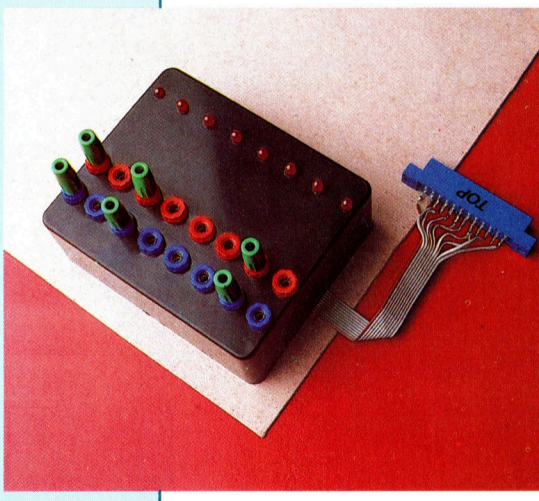
Der Niedervolt-Ausgang des Ausgangs-Buffers wird mit einem 12poligen Minicon-Stekker an der Minicon-Buchse der Buffer-Box angeschlossen. Ein Pin am User Port gibt nur einen Strom von wenigen Milliampere ab, was zur Versorgung eines Elektromotors nicht ausreicht. Ein Transistor dagegen kann damit durchgesteuert werden. Der Niedervolt-Ausgang nutzt die Datenleitungen 0 bis 3. Wird eine dieser Leitungen auf High gesetzt, schaltet der dazugehörige Transistor die Versorgungsspannung zur roten Buchse des Moduls durch. Je nach verwendetem Trafo können bis zu vier Geräte gleichzeitig mit Strömen bis 1 Ampere versorgt werden.



Die Buffer-Box

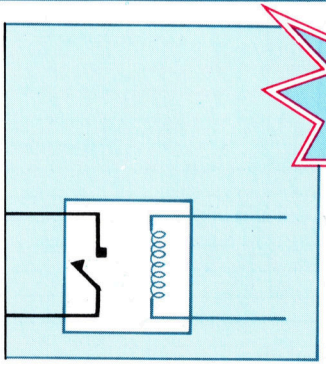
Die Buffer-Box ist das erste – und wichtigste – Teil unseres User-Port-Systems. Die Schaltung schützt die Ein- und Ausgabechips des Computers vor Überlastung. Zusätzlich dient sie zur Gleichrichtung und Regelung einer Trafospannung zwischen +5 und +21 Volt, mit der über den Systembus auch die anderen Module versorgt werden. Über die Buffer-Box können Eingaben zum User Port gemacht werden, wobei die acht roten Buchsen den acht Datenleitungen entsprechen. Die schwarzen Buchsen stellen die separate Masse für jede Datenleitung bereit. Zur Überwachung des Zustands der einzelnen Datenleitungen sind acht LEDs vorgesehen: Wenn eine Datenleitung auf „Low" geht, leuchtet die dazugehörige LED auf.





Netzrelais-Modul

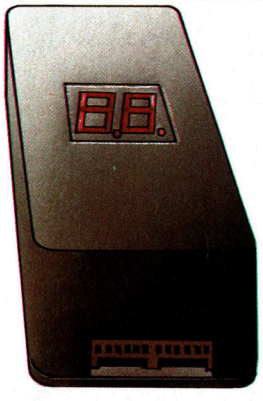
Die Versorgungsspannung kann auch dazu genutzt werden, über ein Relais die Netzspannung zu schalten. Das Modul wird an einer der vier Leitungen des Ausgangs-Buffers angeschlossen. Wenn das entsprechende Bit im Datenregister auf High gesetzt wird, läßt der mit Transistoren verstärkte Strom das Relais anziehen. Dadurch wird die Steckdose mit der Netzspannung versorgt, so daß auch Geräte für 220 Volt steuerbar sind.

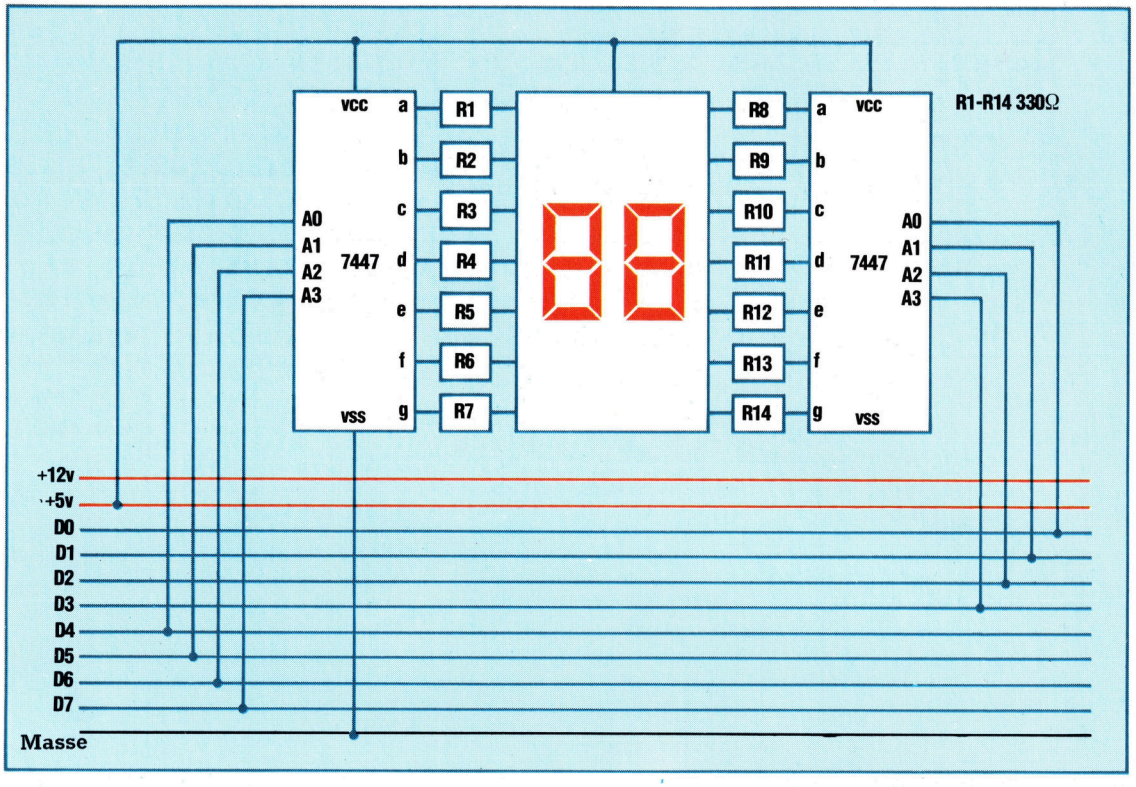


Achtung!
Das Netzrelais ist zwar nicht besonders schwer zu bauen, man sollte es aber wirklich nur dann selbst machen, wenn man sich gut auskennt. Netzspannung ist lebensgefährlich! Wenden Sie sich an einen Fachmann.

Sieben-Segment-Anzeige

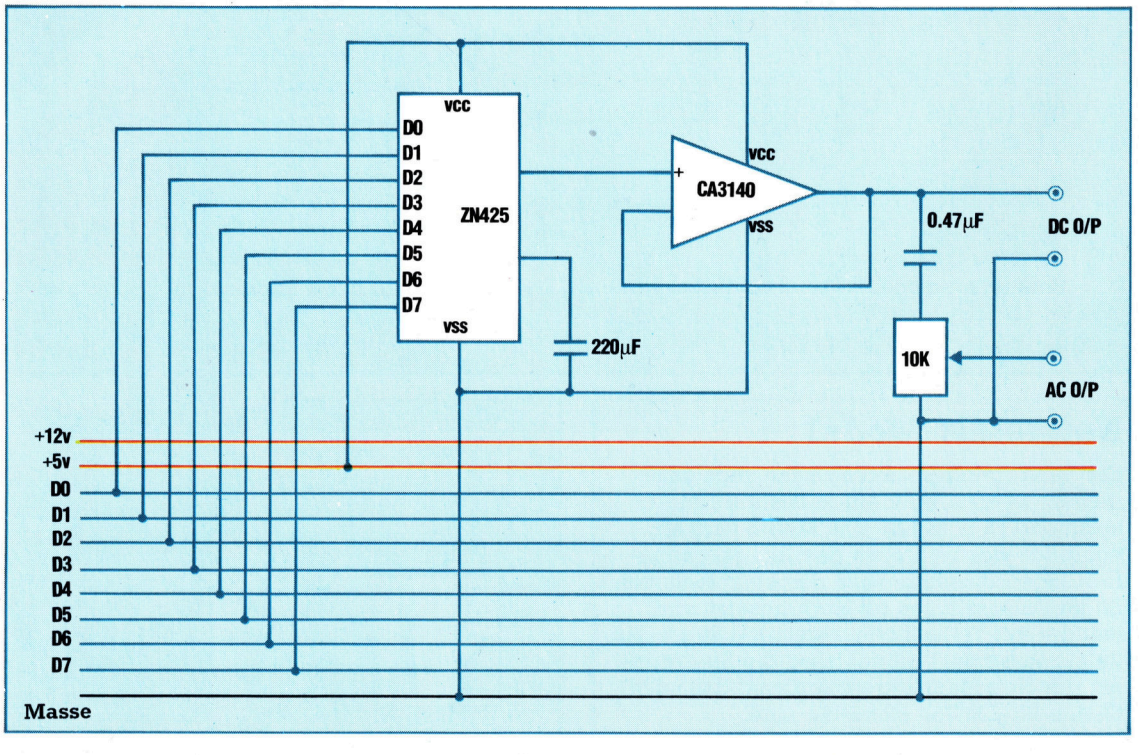
Mit nur vier Anschlüssen kann diese Sieben-Segment-Anzeige alle 16 Hexadezimalziffern darstellen. Mit acht Datenleitungen können wir also zwei Anzeigen betreiben, die den Inhalt des User-Port-Datenregisters in Form zweier Hex-Ziffern sichtbar machen. Der Anschluß kann über die Buffer-Box oder die Minicon-Buchse erfolgen.





Digital/ Analog- Wandler

Datenregister-Werte zwischen 0 und 255 werden von diesem Gerät in eine entsprechende Spannung umgesetzt. Der Wandler liefert circa 50 mW, zum Betrieb von Lampen oder Motoren muß die Leistung also noch verstärkt werden. Der D/A-Wandler kann aber auch zur Tonerzeugung über Kopfhörer oder Stereoanlage eingesetzt werden.



Interstellare Interaktion

In diesem Artikel betrachten wir die Unterschiede zwischen der englischen und amerikanischen Programmiertechnik und stellen ein neues Produkt von Infocom vor, das sehr populär werden dürfte.

Abenteuerspiele belegen viel Speicherplatz und eignen sich vornehmlich für Systeme auf Diskettenbasis. So können neue Örtlichkeiten beschrieben, Nachrichten und Befehle bei Bedarf von Diskette nachgeladen werden. Diese Situation war für englische Abenteuer-Programmierer immer ein Problem, da sie für einen Markt Programme zu schreiben hatten, der traditionell mit der weniger leistungsfähigen Cassette arbeitet.

Aus diesem Grunde ist fast die gesamte britische Abenteuer-Software auf RAM-Basis entwickelt worden und entsprechend limitiert, obwohl viele Programmierer die Grenzen extrem gut ausgenutzt haben. Speziell die Entwicklung „interaktiver Charaktere" (die besonders viel Daten benötigen) sowie von Programmen mit umfangreicherem Vokabular wurde dadurch verhindert.

In den Vereinigten Staaten ist die Situation anders. Alle populären Heimcomputer-Systeme sind mit Diskettenstationen ausgestattet. Deshalb ist auch der Markt für die Entwicklung komplexer Abenteuer-Software besonders geeignet. Das amerikanische Software-Haus „Infocom" ist einer der Marktführer in diesem Bereich.

Das neueste Produkt aus diesem Haus ist typisch für den Standard, den man mittlerweile von Infocom erwartet. „The Hitch Hiker's Guide to the Galaxy", entstanden in Kooperation zwischen Infocom-Programmierern und Douglas Adams, gibt es inzwischen für viele Systeme, darunter den Apple IIe und die Atari-Computer. (Eine Version für den C 64 wird in Kürze lieferbar sein.) Für das Programm sind eine Diskettenstation und viel Geduld erforderlich.

Das Spiel basiert im wesentlichen auf der gleichnamigen, von Douglas Adams für den Rundfunk geschriebenen Serie und enthält Charaktere wie Ford Perfect, Zaphod Beeblebrox, zahlreiche Außerirdische und, natürlich, Arthur Dent, den Antihelden der Geschichte, der sich eines Tages unvermittelt aus seiner normalen Umgebung gerissen findet und auf einem Vogon-Sternenkreuzer irgendwo in den Tiefen des Alls herumkurvt.

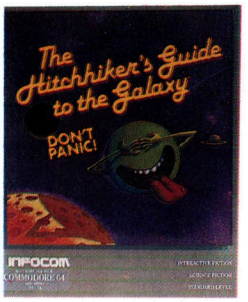
Die Stärke des Programms liegt vor allem im sogenannten „Parser". Das ist der Programmteil, der die Eingabe des Benutzers akzeptiert und interpretiert. Infocom-Parser können zwischen Adjektiven, Adverben und Präpositionen ebenso unterscheiden wie zwischen traditionellen Verben und Substantiven, auf die die meisten bekannten Abenteuerprogramme reduziert sind. Zudem ist der Wortschatz extrem umfangreich – zwischen ein- bis zweitausend Wörter –, und die Eingabe kann in unterschiedlichen Formaten erfolgen. So ist etwa neben einem direkten Befehl (wie „Trink das Bier.") ein Vielfachbefehl wie („Nimm den Sägefisch und stecke ihn in die Tasche.") ebenso möglich wie eine direkte Frage („Wo bin ich?").

Charaktere können durch einfaches Nennen des Namens angesprochen werden, wie „Ford, wo sind wir?" oder „Marvin, verzieh' dich". Selbst wenn der Computer nicht genau versteht, was damit gemeint ist, wird das Programm eine entsprechende Antwort bereithalten – eine erhebliche Verbesserung gegenüber den in britischen Programmen üblichen, jedoch wenig hilfreichen Kommentaren wie „Das kann man nicht" oder „Ich verstehe nicht".

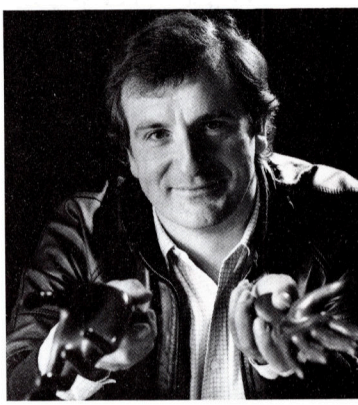
Weitere in diesem Programm verwendete Programmiertechniken sind sogenannte „Container" (Gegenstände, die andere Gegenstände enthalten können), Elemente, die in englischen Programmen häufig fehlen. Gleiches gilt für „allgemeine Objekte" wie zum Beispiel „Boden", „Wand" usw., die an vielen unterschiedlichen Orten sein können, dazu „Vehikel", – Behälter, die den Spieler aufnehmen und von einem Ort zum anderen transportieren können.

„Hitch Hiker's Guide to the Galaxy" bringt zweifelsfrei viele Stunden herrlichen Spielvergnügens und birgt all jene Überraschungen und Witzigkeit, die auch die Original-Rundfunkserie auszeichnet.

Hitch Hiker's Guide To The Galaxy:
Für Apple, Apricot, IBM und Atari-Computer. Eine Commodore-Version steht bald zur Verfügung.
Vertrieb: Rushware
Joystick: Nein
Format: Diskette



„The Hich Hiker's Guide To The Galaxy" ist eines der neuesten Programme der erfolgreichen Infocom-Abenteuer-Serie, die mit der Zork-Trilogie begann.



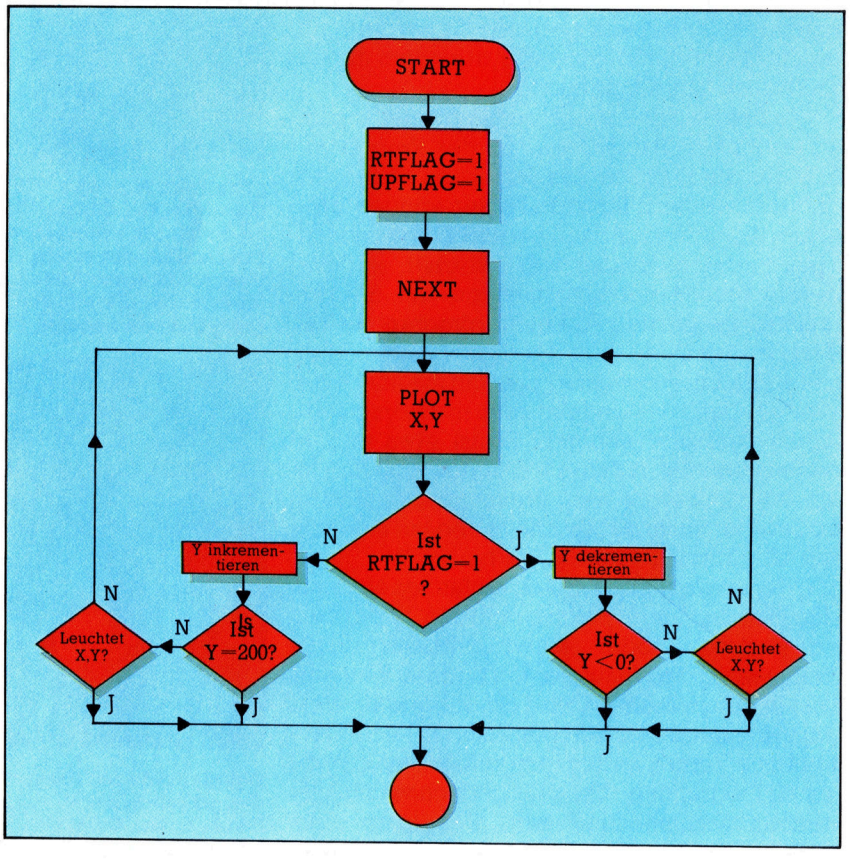
Douglas Adams, Autor des „Hitch Hiker's Guide to the Galaxy" (ursprünglich für den Rundfunk geschrieben), verdient allein an den Buchauflagen, die weltweit verkauft werden, siebenstellige Beträge.

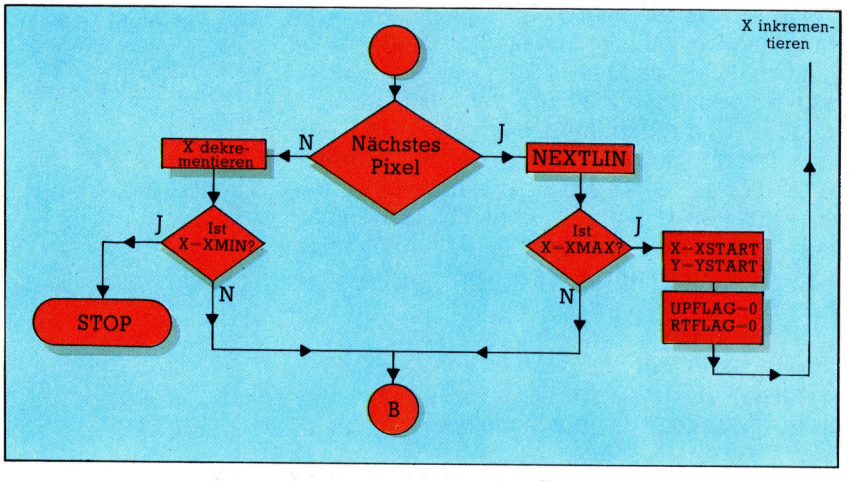
Formschön

Wir entwickelten bisher Maschinencoderoutinen, die die hochauflösenden Fähigkeiten des C 64 einsetzen. Wir beschließen dieses Thema mit einer Routine, die Umrisse ausfüllt.

Es gibt viele Algorithmen zum Ausfüllen von Umrissen. Was jedoch auf den ersten Blick einfach aussieht, ist in der Praxis nicht so leicht zu verwirklichen. So werfen innere Winkel über 180 Grad und Umrisse, die zuerst kleiner und dann größer werden, spezielle Probleme auf. Eine einzelne Routine kann zwar einige dieser Probleme lösen, aber nicht alle gleichzeitig. Außerdem muß das Programm zunächst so „intelligent" gemacht werden, daß es einen geschlossenen Umriß erkennen kann.

In unserem Programm gibt der Anwender einen beliebigen Punkt innerhalb des Umrisses an. Die Routine füllt dann den Bildschirm nach oben hin so lange aus, bis sie auf eine Grenze stößt. Hier bewegt sie sich ein Pixel nach rechts und dann so lange nach unten, bis sie wiederum auf die Umrißlinie trifft. Nun geht sie ein weiteres Pixel nach rechts und bewegt sich wieder aufwärts. Dieser Vorgang setzt sich fort, bis die rechte Seite des Umrisses gefüllt ist. Für die linke Seite geht die Routine zum Anfangspunkt zurück und wiederholt den Vorgang nach links, bis der gesamte Umriß ausgefüllt ist. So können beliebige Flächen ausgemalt werden.





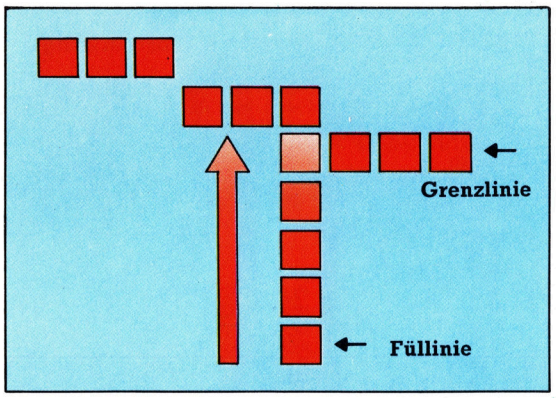
Plot-Inkrementierung

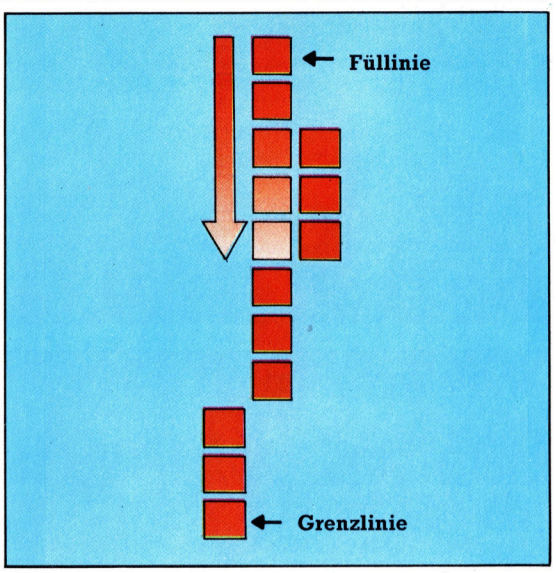
Der erste Teil der Routine ist einfach. Die zwei Flags „UPFLAG" und „RTFLAG" geben die Bewegungsrichtung der Routine an. Das erste Flußdiagramm zeigt den Test für die Plot-Inkrementierung. Die Hauptschleife dieses Programmteils inkrementiert oder dekrementiert (abhängig von UPFLAG) den Wert der Y-Koordinate. Sie testet, ob der Bildschirmrand erreicht ist und ob das nächste Pixel bereits leuchtet. Dann kehrt sie an den Schleifenanfang zurück, um den gleichen Vorgang mit dem darauffolgenden Pixel durchzuführen. Trifft die Routine auf ein bereits aktiviertes Pixel oder den Bildschirmrand, wird die nächste Programmstufe angesprochen.

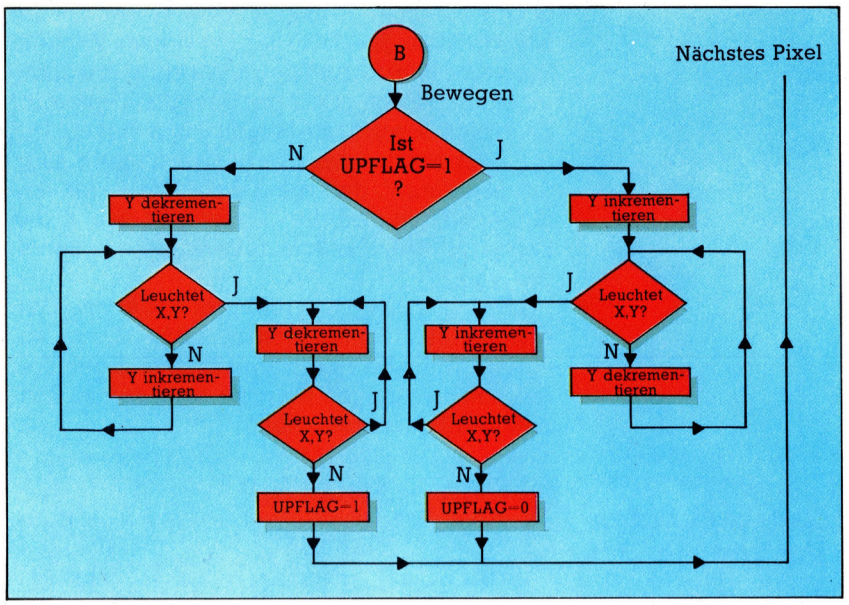
Je nach Status von „RTFLAG" bewegt sich die Routine nun nach rechts oder links. Da die rechte oder linke Grenze des Umrisses nur schwer zu finden ist, muß der Anwender für die X-Koordinate Höchst- und Niedrigstwerte festsetzen. Mit diesem Trick läßt sich der Umriß auch streifenweise füllen.

Das zweite Flußdiagramm hat folgenden Inhalt: Wenn X beim Inkrementieren (d. h. Bewegung nach rechts) seinen Höchstwert erreicht, bereitet die Routine das Ausfüllen des linken Umrißteils vor, indem sie X und Y auf die Anfangswerte und beide Richtungsflags auf Null setzt. Erreicht X beim Dekrementieren „XMIN", ist die Routine beendet. Sind die Höchst- und Niedrigstwerte nicht erreicht, wird die nächste Linie gefüllt.

Der Algorithmus für die Berücksichtigung unterschiedlicher Linientypen (flach, steil etc.)







sieht zwar kompliziert aus, sein Grundprinzip ist jedoch einfach. Nehmen wir an, die aufsteigende Füllroutine trifft auf eine Grenzlinie mit flachem Steigungswinkel.

Im oberen Bild geht der Algorithmus zunächst ein Pixel zurück. Bei einer Rechtsbewegung muß er sich noch um eine weitere Pixelposition zurückbewegen, damit die nächste Füllinie auf einem dunklen Pixel anfangen kann. Ein anderes Problem entsteht, wenn eine absteigende Füllinie auf eine Grenzlinie mit steilem Steigungswinkel trifft.

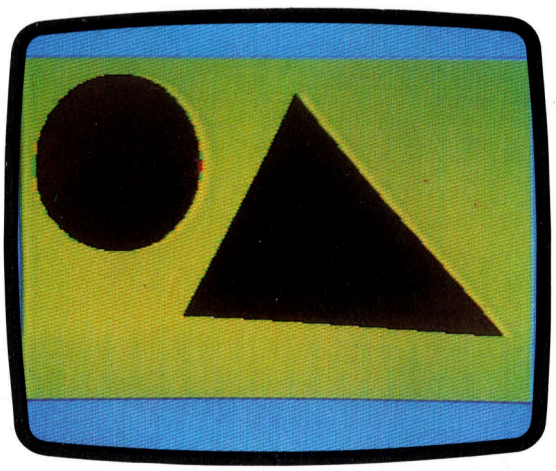
Bei einer Rechtsbewegung muß die Routine hier drei Pixel zurückgehen, um das nächste dunkle Pixel zu finden, von dem an sie wieder aufsteigen kann. Bewegt sich die Füllinie jedoch nach links (im unteren Bild zu sehen), muß sie zunächst drei Pixel abwärts gehen, um auf das nächste dunkle Pixel über der Grenzlinie zu stoßen, von dem aus wieder eine Aufwärtsbewegung möglich ist. Wir müssen daher für jede Richtung zwei Schleifen vorsehen, mit denen wir wiederum in beide Richtungen nachforschen können, wo das nächste dunkle Pixel liegt.

Im Flußdiagramm für das Fillsub-Programm wurden die Labels des Quellentextes eingesetzt, damit sich die entsprechenden Abläufe leichter finden lassen. An einigen Stellen muß der Füllalgorithmus testen, ob ein Pixel leuchtet. Das Programm verwendet dafür die Unterroutine „POINT", die ähnlich wie Plotsub die Adresse eines Punktes aus den X- und Y-Koordinaten errechnet. Statt jedoch das Bit mit einem logischen OR auf Eins zu setzen, stellt POINT mit dem logischen AND fest, ob das angesprochene Bit Eins ist. Ist das Ergebnis des AND-Vorgangs nicht Null, dann steht das Bit auf Eins. Dieses Ergebnis wird in PTFLAG gespeichert und später im Hauptprogramm getestet. Da ein Großteil von POINT die gleichen Abläufe wie Plotsub enthält, können Leser, die mit dem Maschinencode des 6502 vertraut sind, Plotsub ändern, so daß nicht nur die Bits an- und ausgeschaltet, sondern auch die Bitwerte getestet werden können.

Fillsub-Routine

Sehen wir uns an, wie die Fillsub-Routine eingesetzt wird. Zunächst müssen an Fillsub folgende Parameter übergeben werden:

- Die Koordinaten des Anfangspunktes (müssen innerhalb des Umrisses liegen, der gefüllt werden soll).



Nach der Eingabe umfangreicher Maschinencodeprogramme kann der Commodore 64 nun grundlegende Aufgaben in hochauflösender Grafik ausführen, die auf den meisten anderen Heimcomputern als Standard vorhanden sind. Statt sich derart anzustrengen, können Sie sich natürlich auch das Extended-BASIC-Modul kaufen.

● Die maximalen und minimalen Werte der X-Koordinate. Bei Umrissen mit spitzen Winkeln kann sich Fillsub durchaus über eine Grenzlinie hinausbewegen, wenn der gesetzte Grenzwert noch nicht erreicht ist. Setzen Sie in diesem Fall die Grenzwerte etwas weiter nach innen. Achten Sie darauf, daß die X-Koordinate des Anfangspunktes und die Grenzwerte von X das Format Hi-Byte/Lo-Byte haben (siehe Beispielprogramm).

Obwohl Fillsub die anderen Routinen (Plotsub, Linesub und Circsub) nicht direkt anspricht, lädt das Beispielprogramm diese drei Programme, um die Umrisse zeichnen zu können, die Fillsub dann füllt.

Ladeprogramm für Plotsub/II

Hier ist eine angepaßte Version der ersten Plotsub-Routine. Legen Sie damit auf Cassette oder Diskette die neue Objectcodedatei „PLOTSUB.HEX" ab.

```
10 FORI=49408TO49408+314
20 READA:POKEI,A:S=S+A:NEXT
30 READCC:IFCC<>STHENPRINT"CHECKSUM ERROR"
100 DATA1,0,3,6,0,5,0,0,6,5,5,69,38,2
110 DATA72,138,72,152,72,173,0,193,240
120 DATA83,169,0,133,251,169,4,133,252
130 DATA162,3,160,0,173,2,193,145,251
140 DATA136,208,251,230,252,202,48,8
150 DATA208,244,145,251,160,231,208
160 DATA238,173,1,193,240,24,169,0,133
170 DATA251,169,32,133,252,162,32,160
180 DATA0,169,0,145,251,136,208,251
190 DATA230,252,202,208,246,173,24,208
200 DATA41,240,9,8,141,24,208,173,17
210 DATA208,9,32,141,17,208,76,125,193
220 DATA173,24,208,41,240,9,4,141,24
230 DATA208,173,17,208,41,223,141,17
240 DATA208,104,168,104,170,104,96,72
250 DATA138,72,152,72,173,4,193,141,7
260 DATA193,173,3,193,41,248,141,6,193
270 DATA173,3,193,41,7,141,8,193,173,5
280 DATA193,41,7,141,10,193,162,3,78,5
290 DATA193,202,208,250,173,5,193,141
300 DATA9,193,169,0,141,11,193,141,12
310 DATA193,162,5,173,11,193,24,109,9
320 DATA193,141,11,193,202,208,243,162
330 DATA6,14,12,193,14,11,193,144,3
340 DATA238,12,193,202,208,242,173,11
350 DATA193,24,109,6,193,141,11,193
360 DATA173,12,193,109,7,193,141,12
370 DATA193,173,11,193,24,105,0,141,11
380 DATA193,173,12,193,105,32,141,12
390 DATA193,173,11,193,24,109,10,193
400 DATA141,11,193,173,12,193,105,0
410 DATA141,12,193,173,11,193,133,251
420 DATA173,12,193,133,252,169,1,141
430 DATA13,193,56,169,7,237,8,193,240
440 DATA7,170,14,13,193,202,208,248
450 DATA160,0,177,251,13,13,193,145
460 DATA251,76,125,193
470 DATA37523:REM*CHECKSUM*
```

In Zeile 15 zeigt DN=8 an, daß die Objectcodedateien (Plotsub.Hex etc.) von der Diskette geladen werden sollen. Ändern Sie dies in DN=1 um, wenn Sie den Code von Cassetten laden wollen. Legen Sie nun entweder eine Cassette mit allen drei Objectcodedateien an (in der Reihenfolge, die von Zeile 20 bis 30 aufgeführt ist), oder setzen Sie den folgenden Code als Zeile 22, 26 und 28 ein:

INPUT "CASSETTE WECHSELN UND RETURN DRUECKEN"; A$

Fillsub-Beispielprogramm

```
10 REM **** FILLSUB DEMO PROGRAMM ****
15 DN=8:REM FUR CASSETTE DN=1
20 IFA=0THENA=1:LOAD"PLOTSUB.HEX",DN,1
25 IFA=1THENA=2:LOAD"LINESUB.HEX",DN,1
27 IFA=2THENA=3:LOAD"CIRCSUB.HEX",DN,1
30 IFA=3THENA=4:LOAD"FILLSUB.HEX",DN,1
40 GOSUB1000:REM SET HIRES
50 REM **** DRAW TRIANGLE ****
60 XA=100:YA=150:XB=300:YB=160:XC=170:YC=20
80 X1=XA:Y1=YA:X2=XB:Y2=YB:GOSUB2000
90 X1=XC:Y1=YC:GOSUB2000
100 X2=XA:Y2=YA:GOSUB2000
102 REM **** DRAW CIRCLE ****
103 XC=60:YC=60:R=50:GOSUB4000
120 REM **** FILL TRIANGLE ****
130 XS=170:YS=130:REM START POINTS
140 MIN=100:MAX=299:REM LIMITS
150 GOSUB3000
161 REM **** FILL CIRCLE ****
162 XS=60:YS=60:REM START POINT
163 MIN=10:MAX=109
164 GOSUB3000
```

Fillsub-Beispielprogramm (Fortsetzung)

```
 200 GETA$:IFA$=""THEN200:REM AWAIT KEYPRESS
 210 POKE49408,0:SYS49422:REM RESET SCREEN
 220 PRINTCHR$(147):REM CLEAR SCREEN
 225 PRINT"END OF ROUTINE"
 230 END
1000 REM **** SET HIRES ****
1010 POKE49408,1:POKE49409,1
1020 POKE49410,7
1030 SYS49422
1040 RETURN
2000 REM **** LINESUB ****
2010 MHI=INT(X1/256):MLO=X1-256*MHI
2020 NHI=INT(X2/256):NLO=X2-256*NHI
2030 POKE49920,MLO:POKE49921,MHI
2040 POKE49922,NLO:POKE49923,NHI
2050 POKE49924,Y1:POKE49925,Y2
2060 SYS 49934
2070 RETURN
3000 REM **** FILLSUB ****
3010 SH=INT(XS/256):SL=XS-SH*256
3020 HAX=INT(MAX/256):LAX=MAX-256*HAX
3030 HIN=INT(MIN/256):LIN=MIN-256*HIN
3040 POKE50955,SL:POKE50956,SH
3050 POKE50957,YS
3060 POKE50958,LIN:POKE50959,HIN
3070 POKE50960,LAX:POKE50961,HAX
3080 SYS50967
3090 RETURN
4000 REM **** CIRCSUB ****
4010 CHI=INT(XXC/256):CLO=XC-256*CHI
4020 POKE50497,CLO:POKE50498,CHI
4030 POKE50499,YC:POKE50500,R
4040 SYS 50521
4060 RETURN
```

Ladeprogramm für Fillsub

```
10 REM **** BASIC LOADER FOR FILLSUB ****
20 FORI=50944 TO 51375
30 READA:POKEI,A:CC=CC+A:NEXT
40 READA:IFCC<>A THEN PRINT"CHECKSUM ERROR":END
100 DATA11,0,6,8,0,3,6,5,136,39,16,60
110 DATA0,60,10,0,109,0,0,1,10,0,16
120 DATA173,11,199,141,20,199,173,12
130 DATA199,141,21,199,172,13,199,169
140 DATA1,141,18,199,141,19,199,140,5
150 DATA193,173,20,199,141,3,193,173
160 DATA21,199,141,4,193,32,131,193
170 DATA173,19,199,208,8,200,192,200
180 DATA240,19,76,82,199,136,192,0,144
190 DATA11,32,246,199,173,22,199,208,3
200 DATA76,46,199,173,18,199,208,31
210 DATA173,20,199,56,233,1,141,20,199
220 DATA173,21,199,233,0,141,21,199
230 DATA205,15,199,208,65,173,20,199
240 DATA205,14,199,208,57,96,173,20
250 DATA199,24,105,1,141,20,199,173,21
260 DATA199,105,0,141,21,199,205,17
270 DATA199,208,34,173,20,199,205,16
280 DATA199,208,26,173,11,199,141,20
290 DATA199,173,12,199,141,21,199,172
300 DATA13,199,169,0,141,19,199,141,18
310 DATA199,76,46,199,173,19,199,208
320 DATA28,136,32,246,199,173,22,199
330 DATA208,4,200,76,191,199,238,19
340 DATA199,136,32,246,199,173,22,199
350 DATA208,247,76,46,199,200,32,246
360 DATA199,173,22,199,208,4,136,76
370 DATA219,199,206,19,199,200,32,246
380 DATA199,173,22,199,208,247,76,46
390 DATA199,72,138,72,152,72,140,2,199
400 DATA173,20,199,141,0,199,173,21
410 DATA199,141,1,199,173,1,199,141,4
420 DATA199,173,0,199,41,248,141,3,199
430 DATA173,0,199,41,7,141,5,199,173,2
440 DATA199,41,7,141,7,199,162,3,78,2
450 DATA199,202,208,250,173,2,199,141
460 DATA6,199,169,0,141,8,199,141,9
470 DATA199,162,5,173,8,199,24,109,6
480 DATA199,141,8,199,202,208,243,162
490 DATA6,14,8,199,46,9,199,202,208
500 DATA247,173,8,199,24,109,3,199,141
510 DATA8,199,173,9,199,109,4,199,141
520 DATA89,199,173,8,199,24,105,0,141,8
530 DATA199,173,9,199,105,32,141,9,199
540 DATA173,8,199,24,109,7,199,133,251
550 DATA173,9,199,105,0,133,252,169,1
560 DATA141,10,199,56,169,7,237,5,199
570 DATA240,7,170,14,10,199,202,208
580 DATA250,160,0,177,251,45,10,199
590 DATA141,22,199,104,168,104,170,104
600 DATA96
610 DATA50785:REM*CHECKSUM*
```

Assemblerprogramm

```
                ;
                ;++++++++++++++++++++
                ;++++++++++++++++++++
                ;++                ++
                ;++  FILLSUB 64    ++
                ;++                ++
                ;++++++++++++++++++++
                ;++++++++++++++++++++
                ;
                ;++++ PLOTSUB VALUES ++++
                ;
                PLTSUB = $C183
                XLO    = $C103
                XHI    = $C104
                YLO    = $C105
                MPBLO  = $00
                MPBHI  = $20
                PTR    = $FB
                *      = $C700
                ;
                ;++++ FILLSUB VARIABLES ++++
                ;
                PXLO   *=*+1
                PXHI   *=*+1
                PYLO   *=*+1
                PHBLO  *=*+1
                PHBHI  *=*+1
                PREMX  *=*+1
                PYBYTE *=*+1
                PREMY  *=*+1
                PROWLO *=*+1
                PROWHI *=*+1
                PBPOS  *=*+1
                ;
                XSTLO  *=*+1
                XSTHI  *=*+1
                YST    *=*+1
                XMINLO *=*+1
                XMINHI *=*+1
                XMAXLO *=*+1
                XMAXHI *=*+1
                RTFLAG *=*+1
                UPFLAG *=*+1
                FXLO   *=*+1
                FXHI   *=*+1
                PTFLAG *=*+1
                ;      *
                ;++++ TRANS START PT TO FX,FY ++++
                ;
AD 0B C7                LDA XSTLO
8D 14 C7                STA FXLO
AD 0C C7                LDA XSTHI
8D 15 C7                STA FXHI
AC 0D C7                LDY YST
                ;
                ;++++ SET FLAGS ++++
                ;
A9 01                   LDA #$01
8D 12 C7                STA RTFLAG
8D 13 C7                STA UPFLAG
                ;
                ;++++ PLOT POINT ++++
                ;
                NEXT
8C 05 C1                STY YLO
AD 14 C7                LDA FXLO
8D 03 C1                STA XLO
AD 15 C7                LDA FXHI
8D 04 C1                STA XHI
20 83 C1                JSR PLTSUB
                ;
                ;++++ INC./ DEC Y COORD ++++
                ;
AD 13 C7                LDA UPFLAG
D0 08                   BNE DECRY
C8                      INY
C0 C8                   CPY #$C8        ;HAS Y REACHED MAX
F0 13                   BEQ NEXLIN
4C 52 C7                JMP TESTPT
                DECRY
88                      DEY
C0 00                   CPY #$00        ;HAS Y REACHED MIN
90 0E                   BCC NEXLIN
                TESTPT
20 F6 C7                JSR POINT       ;IS POINT LIT ?
AD 16 C7                LDA PTFLAG
D0 03                   BNE NEXLIN      ;IF YES BRANCH
4C 2E C7                JMP NEXT
                ;
                ;++++ START NEXT LINE ++++
                ;
                NEXLIN
AD 12 C7                LDA RTFLAG
D0 1F                   BNE INCRX       ; TEST RIGHT FLAG
AD 14 C7                LDA FXLO
38                      SEC
E9 01                   SBC #$01
8D 14 C7                STA FXLO        ;DEC X LOBYTE
AD 15 C7                LDA FXHI
E9 00                   SBC #$00
8D 15 C7                STA FXHI        ;DEC X HIBYTE

CD 0F C7                CMP XMINHI
D0 41                   BNE MOVE
AD 14 C7                LDA FXLO        ;HAS X REACHED XMIN ?
CD 0E C7                CMP XMINLO
D0 39                   BNE MOVE
60                      RTS             ;END OF ROUTINE
                ;
                INCRX
AD 14 C7                LDA FXLO
18                      CLC
69 01                   ADC #$01        ;INC X LOBYTE
8D 14 C7                STA FXLO
AD 15 C7                LDA FXHI
69 00                   ADC #$00        ;INC X HIBYTE
8D 15 C7                STA FXHI
                ;
CD 11 C7                CMP XMAXHI
D0 22                   BNE MOVE
AD 14 C7                LDA FXLO        ;HAS X REACHED XMAX ?
CD 10 C7                CMP XMAXLO
D0 1A                   BNE MOVE
                ;
AD 0B C7                LDA XSTLO
8D 14 C7                STA FXLO
AD 0C C7                LDA XSTHI       ;RESET START POINT
8D 15 C7                STA FXHI
AC 0D C7                LDY YST
                ;
A9 00                   LDA #$00
8D 13 C7                STA UPFLAG
8D 12 C7                STA RTFLAG      ;RESET FLAGS
4C 2E C7                JMP NEXT
                ;
                ;++++ FIND START OF NEXT LINE ++++
                ;
                MOVE
AD 13 C7                LDA UPFLAG
D0 1C                   BNE DOWN        ;TEST UPFLAG
88                      DEY             ;Y=Y-1
                AGAIN1
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 04                   BNE CONT1
C8                      INY
4C BF C7                JMP AGAIN1
                CONT1
EE 13 C7                INC UPFLAG      ;SET TO ONE
                AGAIN2
88                      DEY
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 F7                   BNE AGAIN2
4C 2E C7                JMP NEXT
                ;
                DOWN
C8                      INY             ;Y=Y+1
                AGAIN3
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 04                   BNE CONT2
88                      DEY
4C DB C7                JMP AGAIN3
                CONT2
CE 13 C7                DEC UPFLAG      ;SET TO ZERO
                AGAIN4
C8                      INY
20 F6 C7                JSR POINT
AD 16 C7                LDA PTFLAG
D0 F7                   BNE AGAIN4
4C 2E C7                JMP NEXT
                ;++++ END OF MAIN PROGRAM ++++
                ;
                ;++++ TEST POINT SUBROUTINE ++++
                ;
                POINT
48                      PHA
8A                      TXA
48                      PHA             ;PUSH REGS ONTO STACK
98                      TYA
48                      PHA
                ;
8C 02 C7                STY PYLO
AD 14 C7                LDA FXLO
8D 00 C7                STA PXLO;TRANSFER COORDS
AD 15 C7                LDA FXHI
8D 01 C7                STA PXHI
                ;
                ;++++ CALCULATE ADDRESS OF POINT ++++
                ;
AD 01 C7                LDA PXHI
8D 04 C7                STA PHBHI
AD 00 C7                LDA PXLO
29 F8                   AND #$F8
8D 03 C7                STA PHBLO
AD 00 C7                LDA PXLO
29 07                   AND #$07
8D 05 C7                STA PREMX
                ;
AD 02 C7                LDA PYLO
29 07                   AND #$07
8D 07 C7                STA PREMY
                ;
A2 03                   LDX #$03
                SHIFT
4E 02 C7                LSR PYLO
CA                      DEX
D0 FA                   BNE SHIFT
AD 02 C7                LDA PYLO
8D 06 C7                STA PYBYTE
                ;
A9 00                   LDA #$00
8D 08 C7                STA PROWLO
8D 09 C7                STA PROWHI
A2 05                   LDX #$05
                FIVE
AD 08 C7                LDA PROWLO
18                      CLC
6D 06 C7                ADC PYBYTE
8D 08 C7                STA PROWLO
CA                      DEX
D0 F3                   BNE FIVE
A2 06                   LDX #$06
                MULT
0E 08 C7                ASL PROWLO
2E 09 C7                ROL PROWHI
CA                      DEX
D0 F7                   BNE MULT
                ;
AD 08 C7                LDA PROWLO
18                      CLC
6D 03 C7                ADC PHBLO
8D 08 C7                STA PROWLO
AD 09 C7                LDA PROWHI
6D 04 C7                ADC PHBHI
8D 09 C7                STA PROWHI
                ;
AD 08 C7                LDA PROWLO
18                      CLC
69 00                   ADC #MPBLO
8D 08 C7                STA PROWLO
AD 09 C7                LDA PROWHI
69 20                   ADC #MPBHI
8D 09 C7                STA PROWHI
                ;
AD 08 C7                LDA PROWLO
18                      CLC
6D 07 C7                ADC PREMY
85 FB                   STA PTR
AD 09 C7                LDA PROWHI
69 00                   ADC #$00
85 FC                   STA PTR+1
                ;
A9 01                   LDA #$01
8D 0A C7                STA PBPOS
38                      SEC
A9 07                   LDA #$07
ED 05 C7                SBC PREMX
F0 07                   BEQ BITON
AA                      TAX
                POWER
0E 0A C7                ASL PBPOS
CA                      DEX
D0 FA                   BNE POWER
                ;
                ;++++ TEST FOR BIT ON ++++
                ;
                BITON
A0 00                   LDY #$00
B1 FB                   LDA (PTR),Y     ;LOAD ADDRESS CONTENTS
2D 0A C7                AND PBPOS
8D 16 C7                STA PTFLAG      ;STORE RESULT
                ;
68                      PLA
A8                      TAY
68                      PLA             ;PULL REGS OFF STACK
AA                      TAX
68                      PLA
                ;
60                      RTS
```

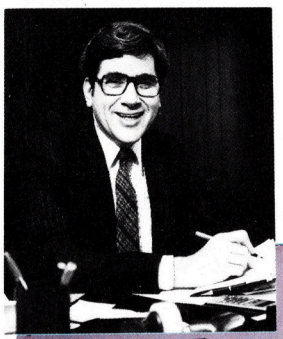
Qualitätskontrolle

Die Firma Softsel ist weltweit einer der einflußreichsten Großhändler von Computer-Hard- und Software. Wenn Sie kommerzielle Programme oder Computerspiele amerikanischer Hersteller einsetzen, ist die Software mit großer Wahrscheinlichkeit auch durch die Hände von Softsel gegangen.

Bei der ständig wachsenden Zahl neuer Programmpakete ist es für den Software-Händler unmöglich, alle neuen Produkte persönlich zu beurteilen, da das ausführliche Testen sehr zeitaufwendig ist. Eine zu flüchtige Bewertung birgt jedoch immer die Gefahr, daß man unverkäufliche und zu teure Pakete einkauft. Die Firma Softsel nimmt Händlern dieses Risiko ab, indem sie jedes angebotene Programmpaket ausführlich testet, bevor sie es in ihren Katalog aufnimmt.

Simon Rhodes, Softsels Marketingdirektor für Großbritannien, erläutert den Ablauf folgendermaßen: „Neue Programmpakete werden zunächst von der technischen Abteilung auf ihre Benutzerfreundlichkeit hin untersucht – es wird getestet, ob das Paket gut programmiert ist, ob die Dokumentation in Ordnung ist und die Grafik anspricht. Danach beurteilt unsere Werbe- und Verkaufsabteilung, ob das Programm mit genügend Werbeaufwand gefördert wird."

Außer Qualitätskontrollen bietet Softsel noch Verkaufsförderung und die Rücknahme nicht verkaufter Programme. Zwar sind die Programme direkt bei den Herstellern billiger, durch Großeinkäufe kann Softsel jedoch Rabatte bieten, die diesen Preisunterschied auf ein Minimum reduzieren. Simon Rhodes sagt dazu: „Natürlich erhalten die Händler einen besseren Preis, wenn sie direkt bei den Herstellern kaufen. Auf lange Sicht gesehen ist es jedoch teurer, da sie mit Hunderten verschiedener Firmen verhandeln müssen, statt nur mit einem Unternehmen."



Herb Blumstein, Geschäftsführer

Simon Rhodes, Marketingdirektor

Gründung in Amerika

Softsel wurde 1980 von Robert Leff und David Wagman gegründet, die früher in der Datenverarbeitung von Transaction Technology – einer Tochtergesellschaft des Finanzierungsgiganten Citicorp – zusammengearbeitet hatten. Das rapide Wachstum von Softsel zeigte, daß Leff und Wagman mit ihrem Konzept genau richtig lagen. Vier Jahre nach der Gründung beschäftigte Softsel weltweit 350 Mitarbeiter und setzte im letzten Geschäftsjahr 87 Millionen Dollar um. Allein in den Vereinigten Staaten unterhält die Firma vier große Läger – Atlanta, Chicago, Los Angeles und New York – und bietet den Händlern landesweit 4500 Programmpakete an.

Im September 1982 streckte Softsel ihre Fühler auch nach England aus und gründete schon ein halbes Jahr später das Zweigunternehmen mit dem Namen „Softsel Computer Products". SCP hat seinen Sitz in Feltham, in der Nähe des Flughafens Heathrow, und beliefert Europa und den Mittleren Osten mit mehr als 2500 verschiedenen Produkten.

Umfangreiches Programmangebot

Die Zukunft der Firma scheint gesichert. So plant die englische Tochtergesellschaft eine Ausweitung der kommerziellen Software, die augenblicklich etwa die Hälfte ihres Katalogs ausmacht. Softsel wird aber auch den zweiten Hauptbereich der Softwareproduktion – die Spielprogramme – nicht vernachlässigen.

Auch auf die übrigen europäischen Länder möchte Softsel seinen Einfluß ausdehnen. So gibt es in Deutschland eine Filiale mit Sitz in München, und auch die Computerfreaks in Frankreich und Italien profitieren von dem umfassenden Programmangebot.

Fachwörter von A bis Z

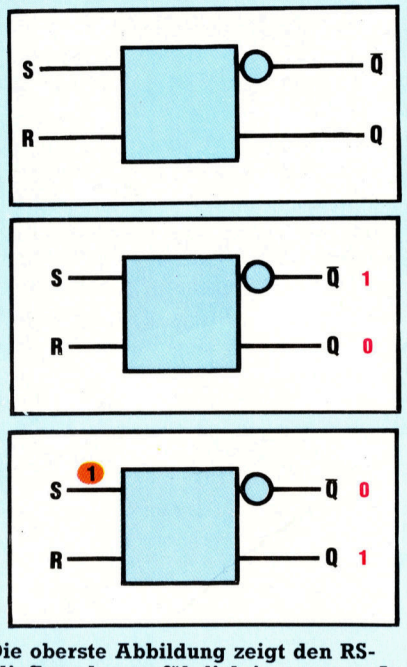
Filtering = Filterung
Im allgemeinen versteht man unter Filterung die Signalbeeinflussung durch Unterdrückung bestimmter Frequenzanteile. Der bei HiFi-Anlagen gängige Rumpelfilter stellt zum Beispiel einen „Hochpaß" dar, der nur Signalanteile oberhalb einer vorgegebenen Frequenz durchläßt: Beim Aufdrehen des Filters wird diese Grenzfrequenz angehoben, so daß die hochfrequenten Töne bevorzugt werden.

Bei der Übertragung digitaler Signale ist die Rauschfilterung ein wesentliches Hilfsmittel zur Fehlerunterdrückung. Dabei finden Frequenzfilter wie auch Prüf- und Paritätscodes Verwendung. Ferner haben viele Rechner einen Netzfilter zum Fernhalten von Hochspannungsspitzen („Spikes" oder Transienten), die häufig beim Schalten von hohen induktiven Lasten wie Aufzugsmotoren oder großen Kühlaggregaten auftreten. Die Spikes werden vom Spannungsregler des Rechners allein meist nicht aufgefangen und können Resets, unter Umständen sogar Schäden an Bauteilen verursachen.

Eine Filterung (oder „Maskierung") von Information ist durch logische Operationen möglich, indem zum Beispiel nur das jeweils höchstwertige Bit übertragen wird.

Flag = Kennzeichenbit
Ein „Flag" zeigt einen Betriebszustand oder ein Operationsergebnis an. Die CPU verfügt über ein Flag-Register (auch Status- oder Konditionsregister genannt), dessen einzelne Flags je nach Prozessoraktivität und Resultat unterschiedlich gesetzt werden. Wenn ein 8-Bit-Prozessor beispielsweise 236 und 101 addiert hat, wird das Zero-Flag gestrichen, weil das Ergebnis nicht Null lautet. Gleichzeitig wird das Carry-Flag gesetzt, weil die Summe die Kapazität des 8-Bit-Akkumulators übersteigt und ein Übertrag (Carry) erforderlich ist. Einige Assembler-Befehle des Prozessors haben je nach Zustand dieser Flags unterschiedliche Wirkungen, so daß logische Entscheidungen möglich sind.

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.



Die oberste Abbildung zeigt den RS-Flipflop, der ausführlich in unserem Logikkurs beschrieben wurde. Die beiden Ausgänge Q̄ und Q weisen jeweils unterschiedlichen Zustände auf. Das zweite Diagramm stellt den Anfangszustand (RESET) dar, und das dritte verdeutlicht eine Schaltung, wenn der Impuls auf dem SET-Eingang liegt.

Flip-Flop = Flipflop
Da Computer mit Binärzahlen arbeiten, sind Schaltkreise mit zwei stabilen, definiert umschaltbaren Zuständen für den Aufbau von Speicher- und Logikelementen von großer Bedeutung. Solche „bistabilen" Schaltungen werden als Flipflops bezeichnet. Die Grundform ist der RS-Flipflop, der am einfachsten durch kreuzweises Verbinden der Ein- und Ausgänge zweier NAND-Gatter realisiert wird.

Floating Point Notation = Gleitkommadarstellung
Jeder Rechner kennt mindestens zwei Formate für die interne Zahlendarstellung: ganze Zahlen und Gleitkommazahlen. Bei dem ganzzahligen Format werden jeder Variablen zwei Bytes für eine 16-Bit-Zweierkomplementdarstellung zugewiesen. Der Wertebereich ist daher

Dezimal −32 768 bis +32 767
Binär 1000 0000 0000 0000 bis
0111 1111 1111 1111

Im Gleitkommaformat wird eine Zahl dagegen folgendermaßen dargestellt:

$\pm$ Zahl = $\pm$ Mantisse × Basis $^{\pm\text{Exponent}}$

z. B. +317440 =
+0,60546875 × 2^{+19}

Weil die interne Zahlenbasis bei einem Binärsystem immer 2 ist, braucht sie nicht extra notiert zu werden, so daß nur Mantisse und Exponent mit dem jeweiligen Vorzeichen (Sign = S) zu speichern sind:

Exponent		Mantisse			
S		S			
0	0010011	0	1001101	10000000	00000000
Byte 0		Byte 1	Byte 2	Byte 3	

Mantisse und Exponent werden als Zweierkomplement-Zahlen dargestellt, so daß das höchstwertige Bit das Vorzeichen enthält (0 für +, 1 für −). Der Exponent wird bei der gebräuchlichen „normalisierten" Darstellung so gewählt, daß die Mantisse nie unter 1/2, aber stets unter 1 liegt.

Der große Vorteil des Gleitkommaformats liegt darin, daß große und kleine Zahlen gleich genau und kompakt darstellbar sind. Mehr Stellen in der Mantisse erhöhen die relative Genauigkeit, mehr Stellen im Exponent erweitern den Bereich.

Bildnachweise
1177: Paul Chave
1178: Liz Dixon, Computer Weekly
1179, 1183, 1187, 1200, 1201: Kevin Jones
1180, 1181, 1204, U3: Ian McKinnell
1184, 1185, 1192, 1193, 1196, 1197: Liz Dixon
1190, 1191, 1194: Chris Stevens
1195: Caroline Clayton
1198: Kevin Jones, Ian McKinnell
1199: Jill Furmanovsky, Time Out

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

computer kurs Heft 44

Nachfolgemodell

Nachfolger des erfolgreichen Schneider CPC 464 ist der CPC 664 mit integriertem Floppy-Laufwerk. Das Gerät wurde wieder von Amstrad entwickelt.

Dynamische Daten

Diese Folge unseres PASCAL-Kurses befaßt sich mit Pointern, dem Heap und verketteten Strukturen, um große Datenmengen verarbeiten zu können.

Auf der Suche

Die umfassende Untersuchung des „Such-Konzepts“ zur Lösung von Problemen beschreibt dieser Beitrag.

Dschungelfieber

Eine gelungene Mischung aus Action- und Abenteuerspiel ist „Sabre Wulf“. Für den ZX Spectrum und Acorn B.

Magische Maus

Computersteuerung per Maus wird immer beliebter. Wir stellen „Magic Mouse“ für den C 64 vor.