Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 28

Serie: Assembler

Roboter zum Selberbauen

Random Access-Dateien

Der Osborne Vadem

Ein wöchentliches Sammelwerk

computer kurs

Heft 28

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. Bei Bestellungen aus Österreich oder Schweiz senden Sie Ihren Auftrag bitte auch an die Hamburger Adresse. Berechnung und Zahlung erfolgen in Landeswährung zum Ladenpreis.

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

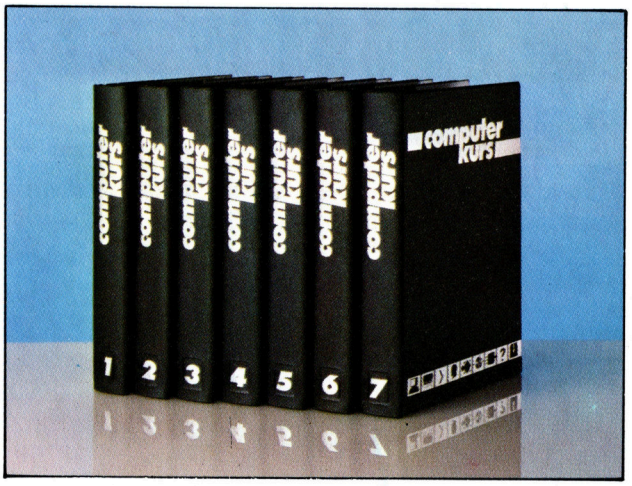
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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.

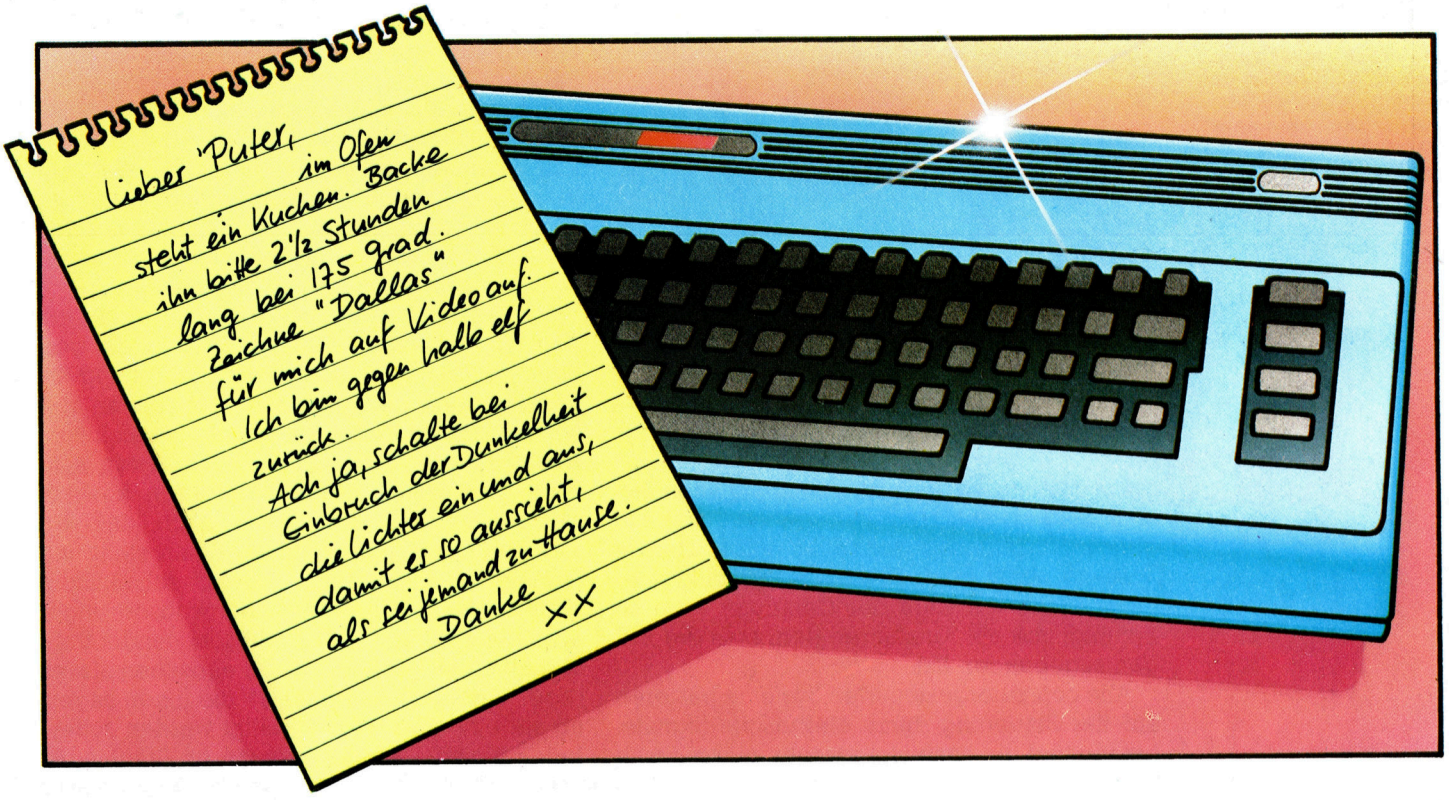
Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk RedaktionsService GmbH, Paulstraße 3, 2000 Hamburg 1

Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1, Tel.: 040/23 40 85



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

Steuern und Regeln



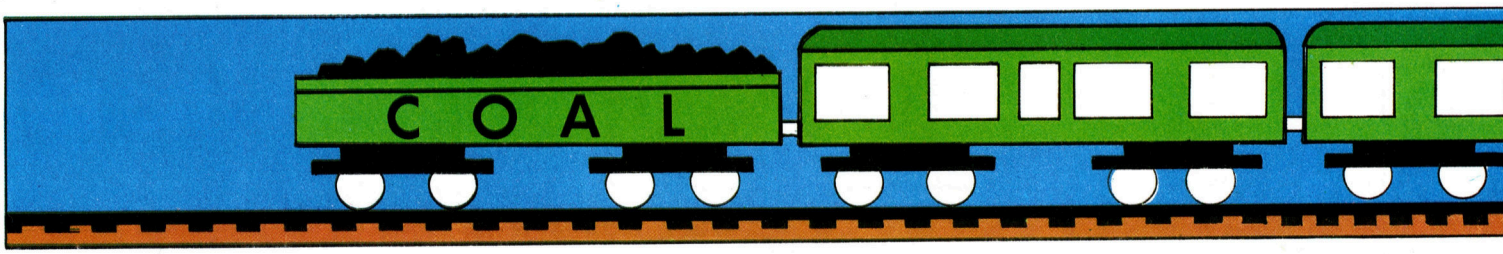
Heimcomputer-Besitzer können wählen, ob sie Software kaufen oder eigene Programme schreiben wollen. Doch nur wenige stellen Überlegungen an, ob sie Peripherien kaufen oder selbst bauen sollten. Und doch gibt es jetzt einige Komponenten auf dem Markt, die den Bau computergesteuerter Geräte ermöglichen.

Der einzige Grund, einen Microcomputer so zu programmieren, daß er am Morgen die Vorhänge öffnen oder während der Ferienzeit die Blumen gießen läßt, ist der Spaß an der Sache. Und das ist auch völlig in Ordnung. Warum sollte man nicht einige Stunden damit verbringen, eigene Software zu schreiben, wenn es Spaß macht?

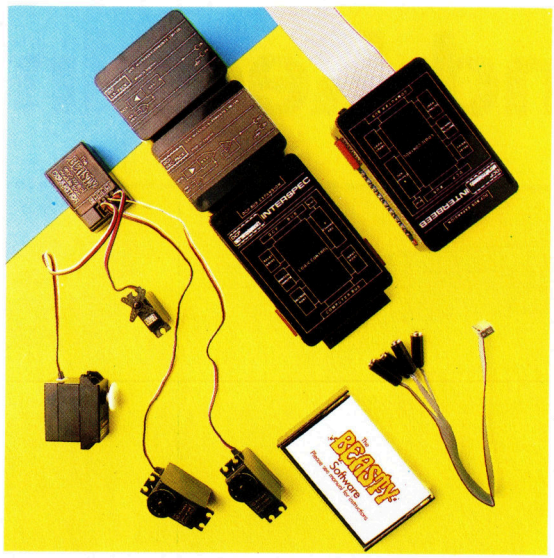
Gegenwärtig mag der Bau eigener Peripherie-Geräte als „Herumspielen" mit selbstgemachten technischen Einrichtungen betrachtet werden. Doch langfristig gesehen wird sich dies als sinnvolle und nützliche Betätigung erweisen. Fähigkeiten, die man heute erlernt, werden in der Zukunft von unschätzbarem Wert sein. Und schließlich wurde so manche große Computer-Gesellschaft (wie Apple oder Atari) in einer Garage gegründet, von Leuten, die ganz einfach mit elektronischem „Kram" und Gerät „herumspielten".

In einem computergesteuerten System werden viele Elemente benötigt. Da ist zunächst der Computer selbst und das, was gesteuert werden soll. Außerdem sind bestimmte Dinge notwendig, um die Steuersignale zu übermitteln, und ferner Software, die den Computer in die Lage versetzt zu entscheiden, was diese Meldungen oder Signale beinhalten sollen. Doch das ist nur ein Teil des Ganzen. Der Computer muß mit einer Vorrichtung ausgestattet

Durch das Verbinden eines Computers mit entsprechenden Geräten gibt es eine Fülle von Möglichkeiten für automatische, programmgesteuerte Erledigung verschiedener Aufgaben. Der Computer kann zu vorprogrammierten Zeiten oder bei Temperatur- oder Lichtveränderungen reagieren.



Schnittstellen für Steuerzwecke gibt es für viele Heimcomputer. Dabei handelt es sich zumeist um Schalteinheiten auf Relaisbasis. Der Computer kann die Stromzufuhr zu Geräten ein- oder ausschalten und erhält Signale, abhängig davon, ob der Sensor „an" oder „aus" ist.



sein, die ihm zu berechnen ermöglicht, was sein Steuerbefehl bewirkt hat, und ihm zugleich Korrekturmöglichkeiten gibt. Diesen Vorgang nennt man „Feedback" (Rückkoppelung). Ohne ihn wäre der Computer etwa so befähigt wie ein Autofahrer, dem man die Augen zugebunden hat.

Alle computergesteuerten Systeme basieren auf der Steuerung durch elektrische Signale. Leider sind die computerinternen elektrischen Signale zu schwach, um etwas bewirken zu können. Selbst eine winzige Glühbirne benötigt mehr Strom als irgendein Teil des Computers. Also ist eine grundlegende Voraussetzung, die minimalen Spannungen des Computers in größere zu verwandeln.

Der erste Schritt erfolgt im Computer selbst. Er braucht ein Gerät, mit dem Signale nach außen geschickt werden können. Dies geschieht, indem man einen bestimmten Teil des Computerspeichers ausschließlich für diesen Zweck benutzt. Über den „User Port" lassen sich bestimmte Informationen von außen lesen, ohne die interne Verarbeitung zu stören.

Einige Microcomputer sind standardmäßig mit einem solchen Interface ausgestattet, andere können nachträglich damit ausgerüstet werden. Der User Port selbst kann zum Ein- und Ausschalten von LEDs Anwendung finden. Doch für praxisnahe Systeme sind andere Komponenten erforderlich. Am sinnvollsten ist der Anschluß einiger elektronischer Teile sowie einer zusätzlichen Stromquelle, die die Steuerung von Relais erlaubt. Relais sind Schalter, die relativ große elektrische Ströme an- und abschalten können, selbst aber durch sehr kleine Ströme gesteuert werden.

Die meisten von Bastlern benutzten Relais arbeiten nur mit batteriebetriebenen Geräten. Es gibt wohl nur wenige Leute, die tatsächlich einen Bedarf an Relais für große Stromstärken haben. Da zudem höhere Spannungen sehr gefährlich sind, sollten ausschließlich auf technische Sicherheit überprüfte (VDE-getestete) Komponenten verwendet werden. Wer mit kleinen Spannungen arbeitet, kann zwischen fertig konfektionierten und selbst gebauten Relais-Schaltern, die an den Computer angeschlossen werden können, entscheiden.

Zur Zeit muß der Computer noch direkt mit den zu steuernden Geräten verbunden werden, was zweifelsfrei eine Einschränkung ist. Verschiedene Unternehmen arbeiten an der Lösung dieses Problems. Man entwickelt zum Beispiel Kabel, die nicht nur Strom leiten, sondern gleichzeitig Daten übertragen können. Mit diesem System ist es möglich, daß der Computer in einem zentralen Raum steht und Signale an sogenannte „Slaves" schickt, die ganz normal an beliebige Steckdosen angeschlossen werden. Der Computer sendet individuelle Anweisungen an die einzelnen Slaves (oder Verbraucherterminals), die ihnen sagen, ob sie ein- oder ausgeschaltet werden sollen. Jedes normale Haushaltsgerät, ob Stehlampe, Fernseher oder Heizofen, kann einfach an den Slave angeschlossen und so durch den Computer gesteuert werden.

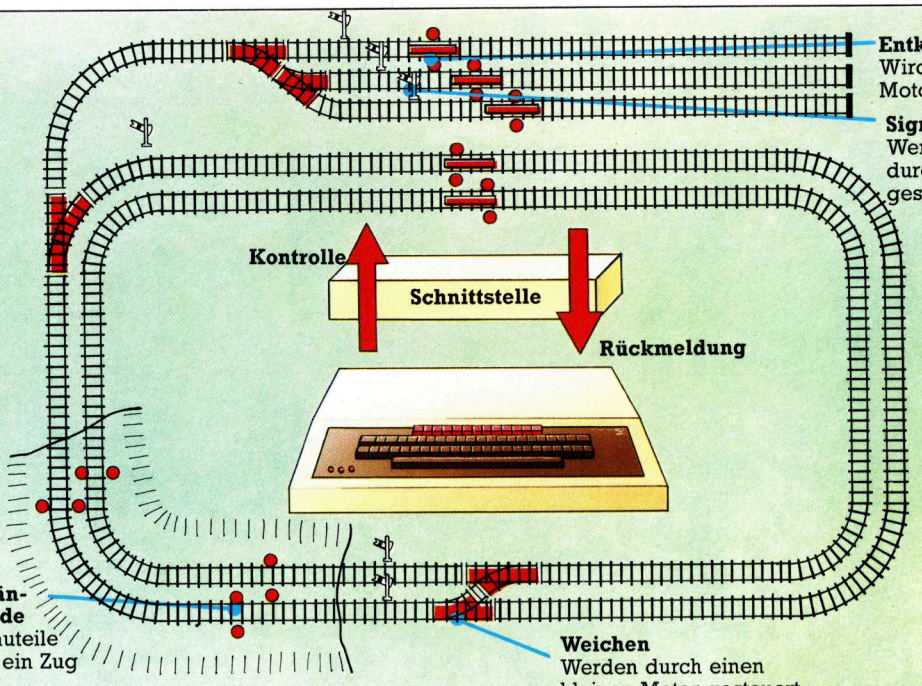
Rückkopplung ist das A und O

Es wurde schon darauf hingewiesen, daß fast alle computergesteuerten Systeme eine Rückmeldung benötigen, um die Funktionsweise des Systems überprüfen zu können. Auf den ersten Blick scheint nicht einsehbar, warum der Computer eine Rückmeldung dafür braucht, ob ein Licht ein- oder ausgeschaltet ist – sinnvoller wäre es, wenn der Computer wüßte, wann es draußen dunkel ist, und aufgrund dieser Information das Licht einschaltet.

Es gibt zwei Arten von Rückmeldungssignalen. Das kann einmal der Zustand „Ein" oder „Aus" sein, ohne daß Zwischenwerte erfaßt werden. Ein solches Signal kann ein einfacher Schalter sein, der meldet, ob ein Fenster geöffnet oder geschlossen ist oder ob die Türklingel betätigt wurde. User Ports können diese Ein/Aus-Signale lesen.

Eisenbahn-Steuerung

Bei jeder Art von Steuerung durch einen Computer – gleich ob Modelleisenbahn oder die elektrischen Anlagen eines ganzen Hauses – findet dieselbe Technik Anwendung. Man installiert eine Schleife, mit der der Computer über eine Schnittstelle Signale aussendet. Diese steuern Servomotoren, Lichter und so weiter. Das Gerät gibt dann eine Rückmeldung. Diese Rückmeldungs-Schleife ermöglicht eine exakte Steuerung durch den Computer.



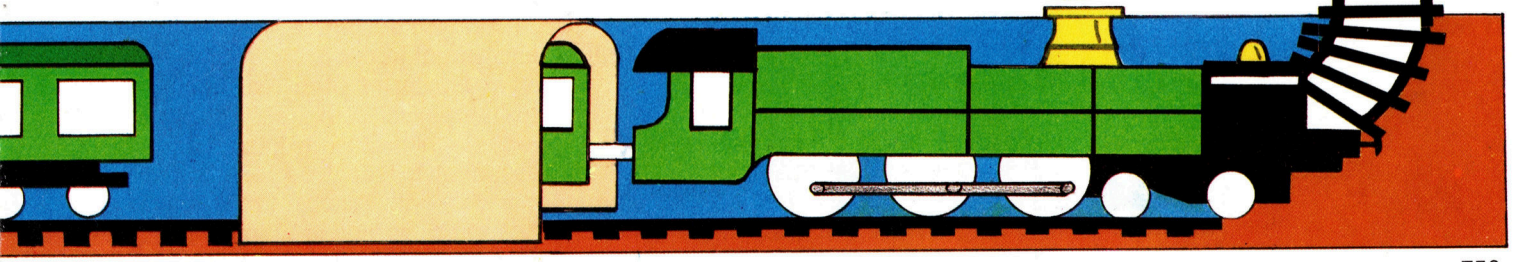
Nützlicher, aber auch komplizierter ist die als Analog-Signal bezeichnete Art von Rückmeldung. Ein solches Signal kann aus einer Reihe von Werten bestehen und so zur Temperaturmessung, zum Messen des Abstands eines Objekts oder wie weit es sich bewegt oder gedreht hat, verwendet werden. Aber auch die Gewichtsmessung oder die Messung der Spannungsabgabe einer Batterie ist möglich. Diese Art von Signal wird von einem Analog-Digital-Konverter (Umwandler) erkannt.

Allein von den Rückmeldungseinheiten hängt ab, was computergesteuert möglich ist. Ein Motor, der ein Rad dreht, kann von einem Computer überwacht werden, indem dieser feststellt, wie weit sich das Rad in einer bestimmten Zeit gedreht hat. Das funktioniert aber nicht, wenn eine Last auf das Rad gelegt wird oder die den Motor antreibenden Batterien schwächer werden, da das Rad sich dann langsamer dreht. Ein optischer Sensor könnte dem Computer in diesem Fall ein Signal übermitteln, wenn eine Umdrehung vollendet ist.

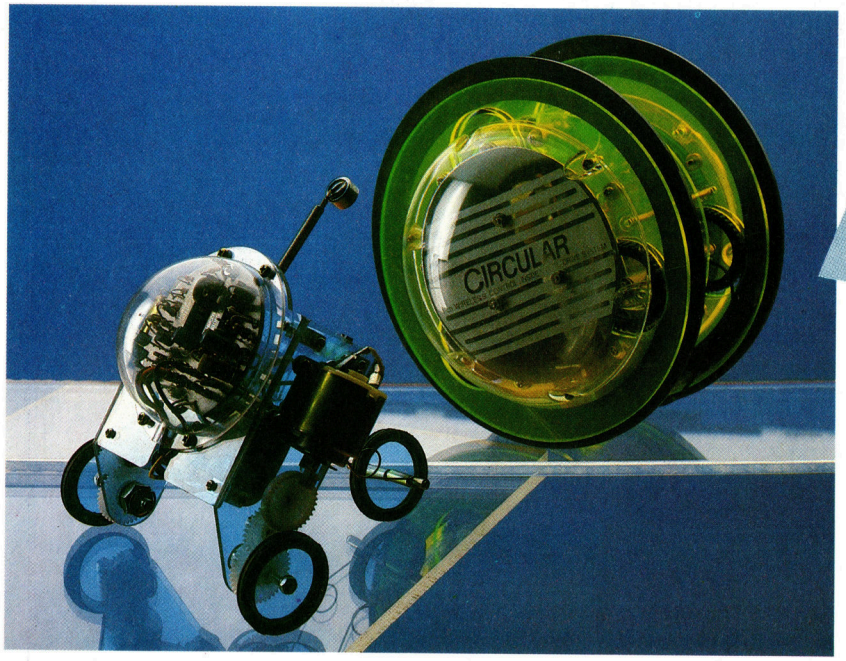
Einige Arten von Elektromotoren sind mit einer Rückmeldungseinheit ausgestattet. Das bedeutet: Der Computer sendet ein Signal, mit dem die Anweisung zur Bewegung in eine bestimmte Position gegeben wird, und der Motor arbeitet so lange, bis er diese erreicht hat. Es gibt zwei Arten dieser Motoren: Schritt- und Servomotoren. Ein Schrittmotor kann sich, wie jeder andere Motor, ständig drehen oder aber in jeder beliebigen Position angehalten werden. Er ist jedoch nicht sehr leistungsfähig und kann deshalb nur gering belastet werden. Servomotoren sind stark, können sich aber nur in einem kleinen Winkel bewegen, der normalerweise bei etwa 90 Grad liegt. Sowohl Schritt- als auch Servomotoren benötigen eine spezielle Steuereinheit, um über Computer angesprochen werden zu können.

Die letzte Gattung computergesteuerter Geräte, die hier vorgestellt werden soll, benötigt zur Steuerung wechselnde Spannungen. Ein Beispiel dafür wäre ein kleiner Elektromotor, der sich mit unterschiedlicher Geschwindigkeit abhängig von der zugeführten Stromspannung dreht. Das Gegenstück eines A/D-Wandlers, ein Digital-Analog-Wandler (kurz D/A), wandelt die digitalen Signale des Computers in wechselnde Stromspannungen um.

Der Einsatz von Computern zur Steuerung anderer Geräte ist dem Schreiben eigener Software vergleichbar. Man muß eine gute Idee mit technischem Wissen verbinden und umsetzen können, und man braucht Zeit. Die Ergebnisse mögen zwar nicht dem aktuellen technischen Stand entsprechen, doch es macht auf jeden Fall mehr Spaß, etwas „selbst zu tun", als fertige Massenprodukte zu kaufen.



Die Movits kommen



Die Abbildungen zeigen drei Roboter aus dem „Movit"-Angebot. Links ist die „Piper Mouse" zu sehen, daneben das „Circular" und rechts der „Memocon Crawler." Alle Movits bestehen aus transparentem, hochwertigen Kunststoff, so daß die internen mechanischen Abläufe verfolgt werden können.

Hinter dem Namen „Movits" steckt eine Serie preiswerter, vorprogrammierter „Roboter"-Bausätze, die für Elektronikfreaks geschaffen wurde. Ob es sich dabei um Roboter handelt oder nicht, wird nachfolgend erörtert.

Die Experten sind sich einig: Roboter werden der nächste Schritt in der Entwicklung zum computerisierten Zuhause sein. Schon heute hat der Fortschritt der Robotik ein Stadium erreicht, das dem der Microcomputer zu Anfang der siebziger Jahre vergleichbar ist. Neue Technologien ermöglichen die Entwicklung völlig neuer Produkte. Doch während die Entwicklung der Computer eher im Verborgenen stattfand, da sich nur wenige Elektronik-Enthusiasten der Möglichkeiten dieser neuen Maschinen bewußt waren, sind sich heute viel mehr Menschen darüber im Klaren, was die gegenwärtige Revolution der Microelektronik bewirkt.

Daher war es nur eine Frage der Zeit, bis ein Anbieter zu der Erkenntnis kam, daß man preiswerte „Roboter" für zu Hause anbieten könne. Die „Movits", von der Kaho Musen Company of Japan produziert, kosten zwischen vierzig und hundertvierzig Mark.

Jedes „Movit" wird als Bausatz geliefert. Extrateile sind nicht erforderlich, einmal von den Batterien (1,5 V) abgesehen. Das Werkzeug – Schraubendreher bzw. Miniaturschraubenschlüssel – wird mitgeliefert. Die kleinen Maschinen wirken sehr futuristisch. Sie sind aus transparentem Kunststoff gefertigt, und die sie bewegende Elektronik befindet sich unter einer durchsichtigen Plastikkuppel. Zur Zeit sind zwölf verschiedene Modelle lieferbar. Das preiswerteste heißt „Monkey" und wird über Geräusche gesteuert. Nach Aktivierung durch den Akustik-Sensor – der beispielsweise einen Ruf oder Händeklatschen wahrnehmen kann – führen zwei sich abwechselnd bewegende Arme den „Roboter" über ein Seil oder eine ähnlich schmale Beförderungsstrecke. Der Sensor besteht aus einem Kondensator-Mikrofon-Transistor, der mit einer Platine verbunden ist und das Signal verarbeitet. Bestimmte Signale aktivieren den Elektromotor, der mit einem Mechanismus verbunden ist und die Arme bewegt. Ein integrierter Timer sorgt dafür, daß „Monkey" rechtzeitig abgebremst wird.

Der infrarotsensor-gesteuerte „Line Tracer II" folgt jeder auf dem Boden befindlichen Linie, wenn ausreichender Kontrast vorhanden ist. Der Roboter wird von zwei Motoren angetrieben, die seine drei Räder bewegen. Auch die Steuerung der „Piper Mouse" erfolgt via Akustik-Sensor, der hierbei allerdings über eine Art „Hundepfeife" aktiviert wird. Nach jedem Pfiff führt das Movit eine Sequenz der vorprogrammierten Befehle aus: Linkswende, Stop, Rechtswende, Stop, vorwärts, Stop.

„Circular" wird ferngesteuert und über zwei Gleichstrom-Motoren angetrieben, die die großen Räder an der Außenseite in Bewegung setzen. An der Fernsteuerkontrolleinheit befinden sich zwei Knöpfe, jeweils einer für einen Motor. Für eine Drehung nach links oder rechts muß einer der Knöpfe, für eine Vorwärtsbewegung müssen beide betätigt werden. Zwei PCBs ermöglichen unabhängige Kontrolle, da sie jeweils unterschiedliche Signale der Steuereinheit empfangen können.

Die bisher beschriebenen Maschinen sind witzig und interessant zu bauen. Doch die dabei verwendete Technologie kann schwerlich als fortgeschritten betrachtet werden. Nach unserer Roboter-Definition sollte ja ein integriertes Steuersystem vorhanden sein, das pro-

grammiert werden kann, um eine bestimmte Abfolge von Aktionen auszuführen.

Nur der „Memocon Crawler" entspricht in etwa dieser Beschreibung. Auch er wird durch Gleichstrommotoren angetrieben. Das Steuersystem besteht aus einer Tastatur, die durch Kabel mit dem „Crawler" verbunden ist. Das Keyboard umfaßt fünf verschiedene Tasten für die einzelnen Befehle: vorwärts, links, rechts, Halt, Tongeneration und Einschalten des integrierten LEDs. Die Befehle werden in einem statischen RAM-Chip (Speicherkapazität 256 Bytes) abgelegt, wobei jedes Byte aus vier Bits besteht.

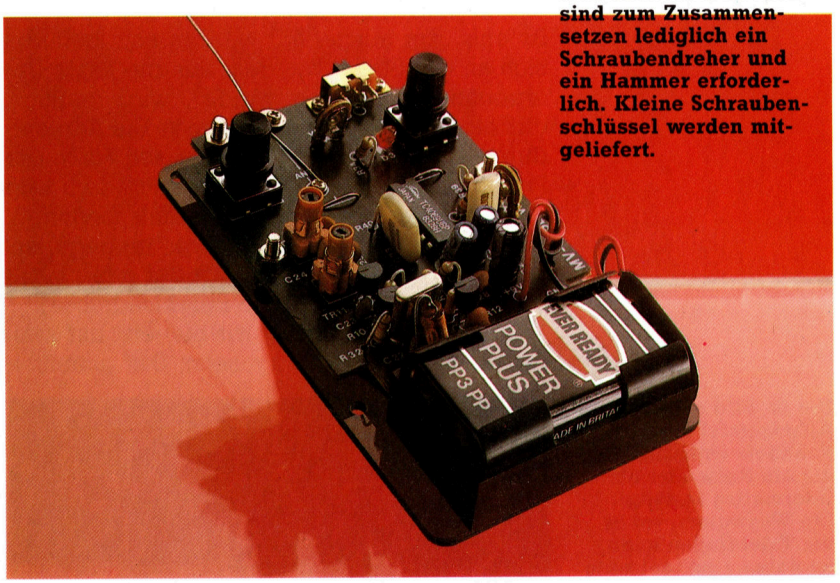
Viele Interessenten mögen sich dadurch abgeschreckt fühlen, daß die „Movits" in Bausatzform geliefert werden und zusammengebaut werden müssen. Doch trotz der beeindruckenden Vielzahl von Einzelteilen sind die „Roboter" sehr leicht zu bauen. Der zehnjährige Sohn des Vertriebsleiters einer großen Elektronikfirma baute die Mustermodelle für das Haus allein zusammen!

Ausführliche Bauanleitung

Die Roboterteile werden in Beuteln verpackt geliefert, die nach Baugruppen numeriert sind. Die Bauanleitungen zeigen die „Movits" in den einzelnen Bauabschnitten in Explosionszeichnungen. Sämtliche Teile sind einzeln gezeichnet und durch die Nummer auf dem Beutel auch entsprechend einfach erkennbar. Pfeile weisen darauf hin, wo das Bauteil eingesetzt werden soll. Für das Gros der Interessenten wird dieser Bauplan reichen. Wer aber noch mehr wissen möchte, kann den ausführlichen Begleittext lesen. Da die Elektronikplatine fertig montiert geliefert wird, reichen Schraubendreher, Zange und ein Hammer als Werkzeug.

Der Bau der „Movits" macht großen Spaß, und ihre Besitzer werden besondere Freude daran haben zu sehen, wie nach und nach ein Roboter entsteht und nach Fertigstellung über den Boden eilt. Der eine oder andere mag bezweifeln, daß es sich bei den „Movits" tatsächlich um Roboter handelt, und sie vielmehr als hochentwickeltes elektronisches Spielzeug betrachten wollen. Doch die „Movits" führen immerhin auf preiswerte Art in die Grundlagen der Robotik ein. Weitere Bausätze sind in Vorbereitung.

Das „Circular" ist wie die anderen Bausätze der Reihe leicht zusammenzusetzen. Sowohl die Platine für den Roboter selbst als auch die Steuerplatine für die Fernsteuerung sind fertig bestückt im Bausatz enthalten. Deshalb sind zum Zusammensetzen lediglich ein Schraubendreher und ein Hammer erforderlich. Kleine Schraubenschlüssel werden mitgeliefert.



Monkey

BEWEGUNG:
Durch zwei sich wechselweise bewegende Greifarme, die eine Hebelbewegung verursachen.

STEUERUNG:
Akustik-Sensor.

STROMVERSORGUNG:
Zwei 1,5-V-Batterien.

Line Tracer II

BEWEGUNG:
Auf drei Rädern, die durch zwei Gleichstrom-Motoren angetrieben werden.

STEUERUNG:
Infrarot-Sensor.

STROMVERSORGUNG:
Zwei 1,5-V-Batterien und eine 9-V-Batterie.

Piper Mouse

BEWEGUNG:
Auf drei Rädern, die durch zwei Gleichstrom-Motoren angetrieben werden.

STEUERUNG:
Hochfrequenz-Sensor.

STROMVERSORGUNG:
Zwei 1,5-V-Batterien und eine 9-V-Batterie.

Circular

BEWEGUNG:
Auf zwei Zahnkranz-besetzten Rädern, die durch zwei Motoren angetrieben werden.

STEUERUNG:
Fernsteuerung.

STROMVERSORGUNG:
Drei 1,5-V-Batterien und zwei 9-V-Batterien.

Memocon Crawler

BEWEGUNG:
Auf drei Rädern, die durch zwei Motoren angetrieben werden.

STEUERUNG:
Programmierbarer Speicher.

STROMVERSORGUNG:
Zwei 1,5-V-Batterien und eine 9-V-Batterie.

Arten der Adressierung

Ausgeklügelte Adressiermethoden unterstützen die Flexibilität der Assemblersprache. In dieser Folge sehen wir uns die verschiedenen Arten der direkten, indirekten und indizierten Adressierung an.

Jeder Assemblerbefehl hat einen direkten oder indirekten Bezug zum Speicherinhalt. Da sich Speicherbytes aber nur durch ihre Adressen voneinander unterscheiden, enthält jeder Befehl mindestens eine Adresse. Die Methode der Adressierung kann dabei direkt und eindeutig sein wie bei dem Befehl LDA $E349 („Lade den Akkumulator mit dem Inhalt der Adresse $E349"). Dabei werden der Akkumulator (ein Speicherbyte, dessen Adresse aus einem Namen statt einer Zahl besteht) und die Adresse $E349 direkt angegeben und ihre Beziehung zueinander durch die Struktur des Befehles definiert.

Der Bezug zu einer Adresse kann aber auch weniger deutlich sein. So scheint der Befehl RET („Rücksprung vom Unterprogramm") keine Adresse zu enthalten. Versteht man den Befehl jedoch als „Die Adresse des nächsten Befehls, der ausgeführt werden soll, befindet sich an der Stelle, von dem aus die letzte Unterroutine aufgerufen wurde", wird deutlich, daß eine Adressierung stattfindet. Die Adresse, deren Inhalt verändert werden soll (d. h. der Programmzähler – das Register, in dem sich die Adresse des nächsten Befehls befindet), wurde allerdings weder direkt angegeben, noch die Adresse, an der sich der neue Inhalt befindet (d. h. die nächste Befehlsadresse).

In dieser Serie haben wir bis jetzt nur zwei Adressiermethoden eingesetzt: Die „unmittelbare" Adressierung, wie in LD A, $45 und ADC #$31, und die „direkte" oder absolute Adressierung, wie in STA $58A7 und LD($696C),A. Mit diesen beiden Methoden scheinen sich (außer der „implizierten" Adressierung wie RTS oder RET) alle Adreßanforderungen erfüllen zu lassen.

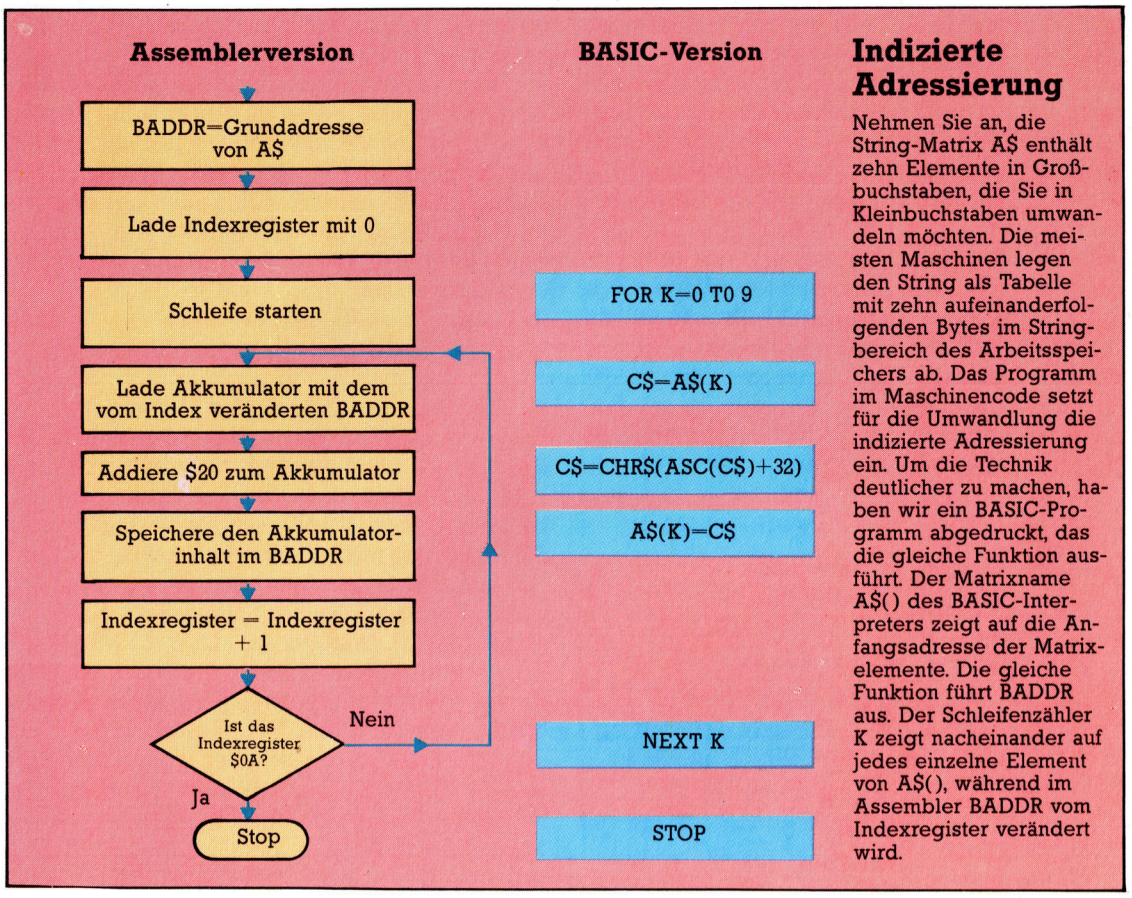
Die Seite 0 des Speichers – auch „Zero Page" genannt – hat für die Adressierung eine besondere Bedeutung. Das Befehlsformat bezieht sich dabei auf Adressen im Bereich $0000 bis $00FF, deren höherwertiges Byte die Zahl $00 ist. Befehle dieser Art bestehen aus nur zwei Bytes: einem für den Op-code und einem für das niederwertige Byte der Adresse. Wenn die CPU eine Ein-Byte-Adresse erhält, setzt sie automatisch $00 als höherwertiges Byte ein und spricht damit Seite 0 an. Der Vorteil der Zero-Page-Adressierung ist ihre hohe Ausführgeschwindigkeit.

Die Zero Page wird von den Microprozessoren Z80 und 6502 unterschiedlich eingesetzt. In der Assemblersprache des 6502 kann jeder Befehl, der den RAM-Bereich adressiert (zum Beispiel LDA), auch im Zero-Page-Format verwandt werden, wobei der CPU alle 256 Bytes dieser Speicherseite zur Verfügung stehen. Im Z80-Assembler gibt es jedoch nur den Befehl RST („Restart" oder „Reset"), der die Seite 0 (und dort auch nur acht Adressen) anspricht. Da RST in seiner Adressierung begrenzt ist und sein Op-code nur aus einem einzigen Byte besteht (die Adresse ist Teil des Bytes), hat er sehr kurze Ausführzeiten.

Die im Bild gezeigte Anzeigetafel eines Bahnhofs enthält Daten, die ein Reisender als Information benötigt. Ein Wegweiser, wo er die Tafel finden kann, ist daher eine indirekte Adressierung dieser Daten. In der Assemblersprache bedeutet indirekte Adressierung, daß die im Operanden des Befehls enthaltene Adresse die Adresse des Bytes enthält, in dem die zu verarbeitenden Daten gespeichert sind. Die Operandenadresse ist ein Zeiger.

Zeit sparen!

Es erscheint kleinlich, die Ausführzeiten verschiedener Assemblerbefehle gegeneinander abzuwägen, da selbst der Ablauf des langsamsten Befehls nur in Mikrosekunden meßbar ist. Es gibt jedoch Assemblerprogramme, bei denen die Einsparung einer Mikrosekunde pro Befehl den Erfolg oder Mißerfolg bestimmt. Spielprogramme mit bewegten, dreidimensionalen Bildern in hoher grafischer Auflösung können zum Beispiel für jedes neue Bild Millionen von Befehlen enthalten, die mit hoher Geschwindigkeit ablaufen müssen, damit die Zügigkeit des Spiels nicht behindert wird. Ein



Indizierte Adressierung

Nehmen Sie an, die String-Matrix A$ enthält zehn Elemente in Großbuchstaben, die Sie in Kleinbuchstaben umwandeln möchten. Die meisten Maschinen legen den String als Tabelle mit zehn aufeinanderfolgenden Bytes im Stringbereich des Arbeitsspeichers ab. Das Programm im Maschinencode setzt für die Umwandlung die indizierte Adressierung ein. Um die Technik deutlicher zu machen, haben wir ein BASIC-Programm abgedruckt, das die gleiche Funktion ausführt. Der Matrixname A$() des BASIC-Interpreters zeigt auf die Anfangsadresse der Matrixelemente. Die gleiche Funktion führt BADDR aus. Der Schleifenzähler K zeigt nacheinander auf jedes einzelne Element von A$(), während im Assembler BADDR vom Indexregister verändert wird.

nennenswerter Zeitgewinn entsteht bereits dann, wenn bei einem Befehl, der in einer Schleife 64 000mal eingesetzt wird, eine Mikrosekunde gespart werden kann.

Die indizierte Adressierung

Die „indizierte" Adressierung ist für die Assemblersprache wichtig, da sie den Aufbau matrixähnlicher Datenstrukturen ermöglicht. Ohne diese Strukturen können Speicherstellen nur einzeln und – wie in unseren Beispielen – mit Angabe der genauen Adresse angesprochen werden.

Die wesentlichen Elemente der indizierten Adressierung sind die Basisadresse und der Index. Wenn eine Datentabelle – zum Beispiel die ASCII-Codes – in fortlaufenden Bytes gespeichert werden soll, dann ist die Adresse des ersten Tabellenbytes die Basisadresse. Die Position jedes Bytes der Tabelle wird nun zu der Basisadresse in Beziehung gesetzt. Das erste Byte steht dabei auf Position 0, das zweite Byte auf Position 1, das dritte auf Position zwei etc. Der Bezug einer Byteposition zu seiner Basisadresse wird Index genannt, wobei sich die absolute Adresse jedes Tabellenbytes aus der Basisadresse und dem Byteindex errechnet.

Auch bei der indizierten Adressierung sind die Assembler des Z80 und des 6502 unterschiedlich strukturiert. Der Chip des 6502 enthält zwei Ein-Byte-Register (X und Y), die je einen Index enthalten, mit dem eine Basisadresse verändert werden kann. Die Länge der Tabelle ist damit auf maximal 256 Bytes begrenzt. Der Z80-Chip dagegen enthält zwei Zwei-Byte-Register (IX und IY), die die Basisadresse selbst enthalten. Die Register können erweitert und reduziert werden und zeigen damit auf die entsprechenden Bytes der Tabelle. Da IX und IY Zwei-Byte-Register sind, läßt sich jedes Byte, das von der CPU angesprochen werden kann, adressieren.

Die indirekte Adressierung

Die „indirekte" Adressierung setzt Pointeradressen ein. Stellen Sie sich als Beispiel einen Filmclub vor, dessen Mitglieder sich jede Woche treffen, um einen Film anzusehen, den der Clubpräsident ausgewählt hat. Da der Film in einem von zwölf Kinos gezeigt werden kann, schreibt der Präsident den Vorführort und die Zeit auf eine Karte, die er in einem Schaufenster des Stadtzentrums aushängt. Wenn die Clubmitglieder erfahren wollen, wo der Film der nächsten Woche gezeigt wird, brauchen sie nur den Laden mit der Information aufzusuchen, der auf den jeweiligen Aufführort „zeigt". Die Adresse des Ladens ist daher indirekt die Adresse des Kinos.

Die Befehle der indirekten Adressierung enthalten somit eine Zeigeradresse, die nicht verarbeitet wird, sondern auf den Inhalt einer anderen Speicherstelle zeigt. Besonders in Verbindung mit der indizierten Adressierung

hat die indirekte Adressierung entscheidende Vorteile. Nehmen Sie an, Sie wollen eine Assemblerroutine schreiben, die eine Datentabelle nach einem bestimmten Zeichen durchsuchen und Ihnen die Indexposition eines bestimmten Zeichens liefern soll. Nehmen Sie weiterhin an, daß in unterschiedlichen Bereichen des Speichers mehrere dieser Tabellen untergebracht sind, die alle mit der gleichen Routine behandelt werden sollen. Wenn Sie in die Routine eine indirekte Adressierung einbauen, bei der der Zeiger auf die Basisadresse der gewünschten Tabelle zeigt, können Sie jede Tabelle ansprechen. Sie müssen vor Aufruf der Routine nur den Zeiger auf die entsprechende Basisadresse setzen.

In der praktischen Anwendung werden statt der Grundbefehle Mischformen eingesetzt:

Befehl	Operand	Einsatzform
LDA	#$34	unmittelbar
LDA	$A2	Zero Page direkt
LDA	$967F	Absolut, direkt
LDA	$A2,X	Zero Page mit X indiziert
LDA	$967F,X	Absolut, mit X indiziert
LDA	$967F,Y	Absolut, mit Y indiziert
LDA	($A2,X)	Indirekt, mit X vorindiziert
LDA	($A2),Y	Indirekt, mit Y nachindiziert

Beachten Sie, daß die beiden letzten Befehlsversionen sowohl die Zero-Page-Adressierung als auch die indirekte und die indizierte Methode einsetzen. Wenn die Tabelle auf den ersten Blick auch etwas verwirren mag, so wird ihre Bedeutung mit ein wenig Übung schon bald deutlich werden, denn auch bei den Befehlen LDA und ADC konnten wir die unmittelbare und absolute Adressierung ohne Probleme einsetzen.

Unterscheidung gleicher Mnemotiks

In der Tabelle ist auch die Antwort auf die Frage enthalten: Wie kann der Op-code eines Befehls bestimmt werden, wenn das mnemotische Kürzel gleich lautet? Die Beispieltabelle zeigt, wie sich die Operandenformate jeder Adressierungsart voneinander unterscheiden. Die einzige Unklarheit entsteht bei der Verwendung symbolischer Adressen. Dabei ist nicht gleich deutlich, ob LDA SYMB1 eine Adresse der Zero Page oder eine absolute Adresse ist. Normalerweise löst das Assemblerprogramm diese Frage automatisch. Bei einer manuellen Assemblierung müssen Sie jedoch überprüfen, ob SYMB1 als Ein- oder Zwei-Byte-Zahl gedacht ist.

Lösung der Aufgaben

Das Monitorprogramm ist für Erweiterungen gedacht, und das Hinzufügen einer weiteren Menüoption ist daher recht einfach:

Die Version für den Spectrum

1) Folgende Änderungen sind bei der Initialisierung zu beachten:

```
1050 LET LT=5:DIM C$(LT):DIM0$
     (LT,24):DIM X$(16)
1150 LET C$="ADGQB":LET C1=-48
     :LET C2=10-CODE(C$(1))
1280 LET 0$(5)=" B-BINAERE ANZEIGE"
```

2) Die Eingaberoutine hat in Zeile 2000 die Anfangsadresse bereits angesprochen, diese in A$ als Hexzahl mit vier Ziffern umgewandelt und in DN als Dezimalzahl gespeichert. Die Unterroutine für die binäre Anzeige sieht folgendermaßen aus:

```
7000 REM**HEX&BIN DISP S/R**
7020 FOR P=DN TO (DN+15)
7040 LET NM=P:GOSUB 3100: PRINT H$,
7060 LET N=PEEK(P):LET NM=N
7080 GOSUB 3000: PRINT B$;" ";
7100 GOSUB 7300: PRINT B$
7120 IF P=65535 THEN LET P=DN+15
7140 NEXT P
7200 RETURN
7300 REM**BINARY BYTE S/R**
7310 LET B$=""
7320 FOR D=8 TO 1 STEP-1
7330 LET N1=INT(N/2)
7340 LET R=N-2*N1
7350 LET B$=STR$(R)+B$
7360 LET N=N1
7370 NEXT D
7380 RETURN
```

Version für den Acorn B/Commodore

Übernehmen Sie die Spectrum-Version mit folgenden Änderungen:

1) Verändern Sie die Initialisierung von LT und 0$ wie bei dem Spectrum und ergänzen Sie die Zeile

```
1150 durch C$(5)="B".
```

2) Zeile 600 übergibt die Programmsteuerung an die entsprechende Unterroutine. Verändern Sie diese auf dem Commodore 64 und auf dem Acorn B folgendermaßen:

```
600 ON CM GOSUB 5000, 5500, 6000,
    6500, 7000
```

3) Ändern Sie auf dem Acorn B die Zeile 7060 in

```
7060 N=?(P):NM=N
```

4) Ändern Sie auf dem Commodore die Zeile 7350 in

```
7350 B$=MID$(STR$(R),2)+B$
```

Lösungen

Die Übungsaufgaben aus dem letzten Teil des Kurses werden erklärt.

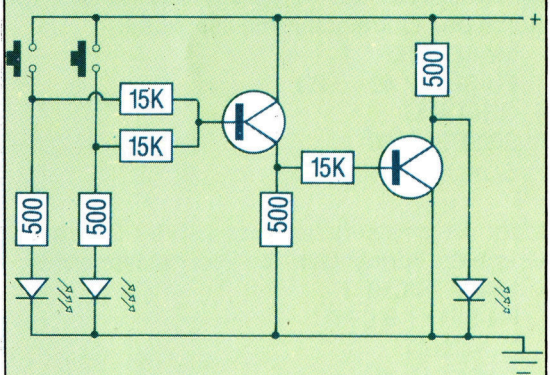
Wie gut kennen Sie sich schon in der Elektronik aus? Die hier von uns vorgeschlagenen Lösungen sind vielleicht anders als die von Ihnen gefundenen. Kein Grund zur Besorgnis – auch in diesem Fall können verschiedene Wege zum gleichen Ziel führen. Entscheidend ist nur, daß sich die Logik der von Ihnen gebauten Schaltung mit den Vorgaben der Wahrheits-Tabelle deckt!

1) Widerstände

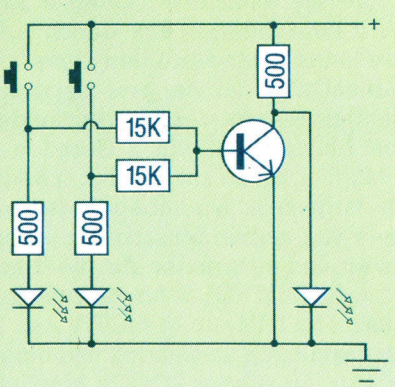
Die Werte der abgebildeten Widerstände sind: a) 6400 Kilo-Ohm und b) 150 Kilo-Ohm. Ein Widerstand von 150 Ohm wäre mit braun-grün-braunen Ringen versehen (In Richtung zum silbernen oder goldenen Ring hin gelesen).

2) NOR-Gatter

Der einfachste Weg zu einem NOR-Gatter ist die Verbindung aus einer OR- und einer NOT-Schaltung wie hier abgebildet:

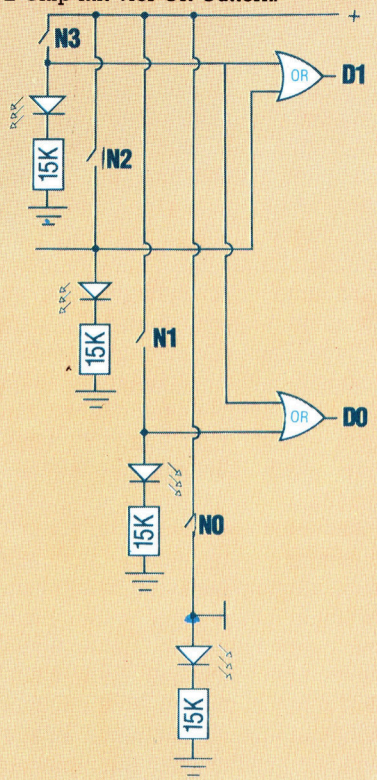


Noch einfacher geht es mit diesem Trick. Dazu wird – ähnlich wie im NOT-Gatter – das Ausgangssignal des Kollektors anstelle des Emittersignals verwendet.



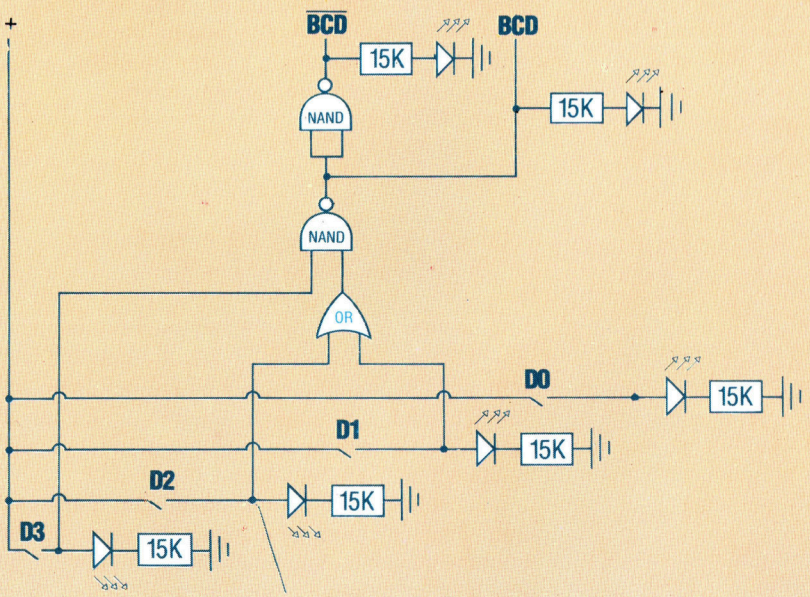
3) Dezimal-Binärwandler

Dafür wird nur ein einziges IC gebraucht – ein TTL-Chip mit vier OR-Gattern.



4) BCD oder nicht?

Die gültigen BCD-Ziffern reichen von 0000 bis 1001, alle höheren Codes sind ungültig. Bei unserer Lösung sieht der BCD-Prüfer so aus:



Räumliche Vorstellung

LOGO ist für die Darstellung von Mustern und Symmetrie besonders geeignet. In dieser Folge wird gezeigt, wie man mit Hilfe der Turtle räumliche Umwandlungen durchführt.

Es gibt vier Arten, eine zweidimensionale Figur zu verwandeln, ohne ihre Form zu verändern: Diese sind Translation, Drehung, Spiegelung und Gleitspiegelung. Im Diagramm wird gezeigt, wie sich die Lage einer Figur bei jeder dieser Umwandlungen verändert.

Man bezeichnet eine Figur als symmetrisch, wenn sie auf eine oder mehrere der vorgenannten Arten umgewandelt werden kann, in Lage und Form aber unverändert bleibt. Um endliche Formen, wie zum Beispiel Polygone und Buchstaben, spiegeln oder drehen zu können, müssen diese symmetrisch sein, da Translationen und Gleitspiegelungen ihre Lage verändern.

Nun soll zunächst eine Form auf einer Linie gespiegelt werden, die durch das Original führt und eine vorgegebene Richtung hat.

Am einfachsten ist es, wenn in der Prozedur zum Zeichnen der Figur die Turtle ihre Ausgangsposition- und -richtung beibehält. Die Aufgabe wird dann in zwei Teile untergliedert: Zunächst müssen die Koordinaten sowie die Richtung des Startpunktes für die Spiegelung, die mit dem Startpunkt der Originalfigur übereinstimmt, gefunden werden. Vor dem Zeichnen der Figur muß letzteres geändert werden. Das heißt: Hierbei sind lediglich alle Rechtswendungen in der Zeichenprozedur in Linkswendungen umzuwandeln und umgekehrt. Eine Möglichkeit besteht darin, sämtliche RT und LT in der Prozedur durch DREHEN zu ersetzen, das wie folgt definiert wird:

```
TO DREHEN :A
  RT :RICHT * :A
END
```

Jetzt läßt sich ein Quadrat so definieren:

```
REPEAT 4 [FD 50 DREHEN 90]
```

Um diesen Befehl einsetzen zu können, muß die globale Variable RICHT auf 1 gesetzt werden. MAKE "RICHT 1 QUADRAT zeichnet nun ein Quadrat. Um dieses auf der Y-Achse zu spiegeln, ist lediglich MAKE "RICHT (-1) einzugeben und anschließend QUADRAT.

Der Befehl zur Positionierung der Turtle vor dem Zeichnen der Spiegelung basiert auf trigonometrischen Formeln, die im Programm wie folgt umgesetzt werden:

```
TO SPIEGELN :A
  MAKE "H HEADING
  MAKE "XALT XCOR
  MAKE "WINKEL (ATAN :YALT :XALT)
    —90 + :A
  MAKE "R QUAT ( :XALT * :XALT +
    :YALT * :YALT)
  PU
  SETXY 0 0
  SETH :A + :WINKEL
  FD :R
  SETH 2 * :A — :H
  PD
  MAKE "RICHT :RICHT * (—1)
END
```

Mit dieser Prozedur kann man sehen, wie Spiegelungen auf verschiedenen Linien, die durch das Original führen, aussehen:

```
MAKE "RICHT 1
PU SETXY 40 70 PD
QUADRAT
SPIEGELN 60
QUADRAT
```

Liegt die gespiegelte Form auf der Spitze des Originals, spricht man von Spiegelsymmetrie.

```
MAKE "RICHT 1
PU SETXY 0 0 PD
QUADRAT
SPIEGELN 45
QUADRAT
```

Es ist möglich, mit Translationen und Gleitspiegelungen einen ganzen Mustersatz zu bewegen, ohne die einzelnen Elemente zu verändern. Wir konzentrieren uns jedoch auf Translationen längs einer einzelnen Linie.

Kombinationen der vier grundlegenden Umwandlungen lassen sieben verschiedene Arten von Mustern auf einer Geraden zu. All diese Möglichkeiten sind in dem zweiten Diagramm dargestellt. Wir haben Prozeduren zum Zeichnen von sieben Mustern für jedes Motiv entwickelt und verwenden für die Translation die Prozedur BEWEGEN, für die Drehung DREHEN und R.MOTIV, die in MOTIV alle Rechts- in Linksdrehungen umwandelt und umgekehrt.

Hier werden wieder LOGOs Möglichkeiten der Listenverarbeitung genutzt, um die Proze-

7 Muster

Die folgenden Programme laufen nur, wenn Sie bereits die Prozeduren UMSCHR.PROZ, UMSCHR.ZEILE, AENDER.WORD, POSITION, BEWEGEN und DREHEN eingegeben haben. Damit lassen sich sieben verschiedene Muster erzeugen:

```
TO MUSTER1 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  POSITION
  REPEAT 6 [MOTIV BEWEGEN]
END

TO MUSTER2 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R.MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 3 [MOTIV BEWEGEN DREHEN
    R.MOTIV DREHEN BEWEGEN]
END

TO MUSTER3 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R: MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 6 [MOTIV R.MOTIV BEWEGEN]
END
TO MUSTER4 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R.MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 6 [MOTIV DREHEN MOTIV
    DREHEN BEWEGEN]
END

TO MUSTER5 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R.MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 3 [MOTIV R.MOTIV BEWEGEN
    DREHEN MOTIV R.MOTIV DREHEN
    BEWEGEN]
END

TO MUSTER6 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R.MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 6 [MOTIV DREHEN R.MOTIV
    DREHEN BEWEGEN]
END

TO MUSTER7 :PROZ
  DEFINE "MOTIV TEXT :PROZ
  DEFINE "R.MOTIV UMSCHR.PROZ TEXT
    :PROZ
  POSITION
  REPEAT 6 [MOTIV R.MOTIV DREHEN
    MOTIV R.MOTIV DREHEN BEWEGEN]
END
```

Die erforderliche Zeichenprozedur für das Motiv ist:

```
TO BEIN
  FD 50
  RT 90
  FD 20
  BK 20
  LT 90
  BK 50
END
```

Ein alternatives Motiv wäre:

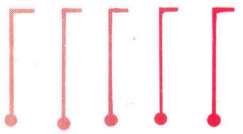
```
TO FIG
  RT 30
  FD 20
  LT 50
  FD 20
  RT 90
  FD 10
  REPEAT 4 [FD 20 RT 90]
  BK 10
  LT 90
  BK 20
  RT 50
  BK 20
  LT 30
END
```

LOGO-Dialekte

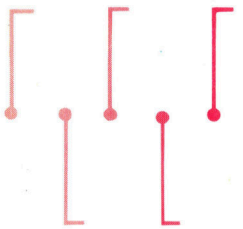
Das Atari-LOGO kennt weder ATAN noch TOWARDS. Das beeinträchtigt zwar die Prozeduren SPIEGELN und DREHEN, nicht aber die MUSTER-Programme.
TEXT und DEFINE sind nicht im Atari-LOGO-Wortschatz enthalten, können aber definiert werden. Man kann unter Verwendung des Editors R.MOTIV durch Modifizierung von MOTIV erstellen.

LOGO-Motive

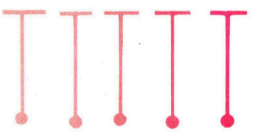
Translation



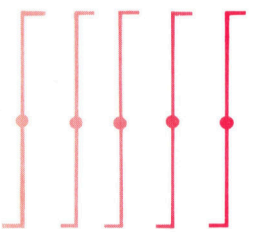
Gleitspiegelung



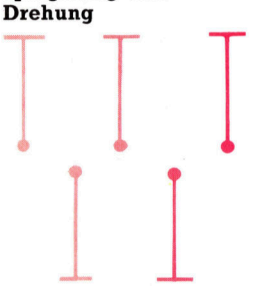
Zwei Spiegelungen



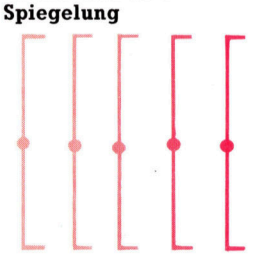
Translation und Drehung



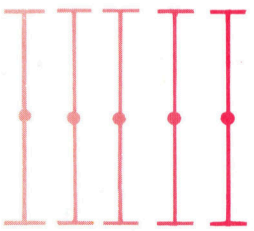
Spiegelung und Drehung



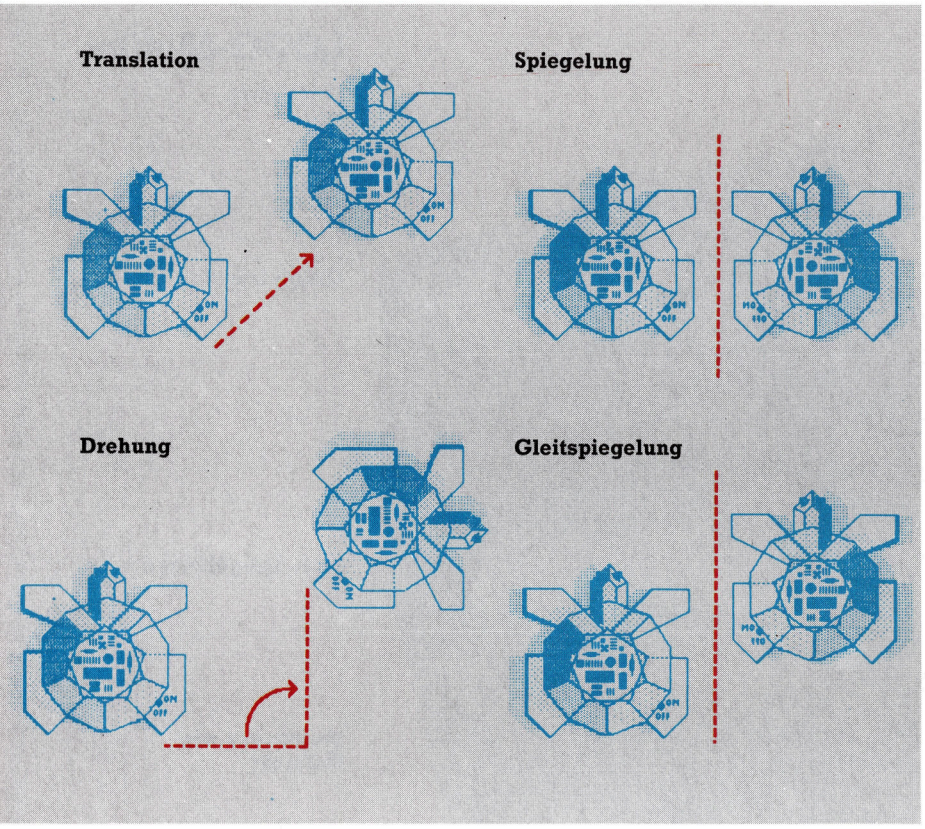
Translation und Spiegelung



Translation und zwei Spiegelungen



Die vier einfachen isometrischen Umwandlungen können zur Erzeugung sieben individueller Muster verschieden kombiniert werden. In allen Fällen beginnen wir mit dem „Beinmotiv". Die Translationen erfolgen in Richtung der X-Achse.



Isometrische Umwandlungen ändern die Lage, nicht aber die Form einer Figur. Es gibt vier Grundtypen der isometrischen Umwandlung: Translation, Drehung, Spiegelung und Gleitspiegelung. Translation ist eine einfache „Verschiebung" des Originals. Bei der Drehung wird die Figur um einen bestimmten Punkt gedreht. Die Spiegelung beinhaltet die Bewegung von Punkten über eine Spiegellinie derart, daß jeder Punkt der neuen Figur denselben Abstand zur Linie hat wie der entsprechende Punkt des Originals. Gleitspiegelung ist eine Kombination aus Spiegelung und Translation. Während Translation und Drehung den „Sinn" des Originals erhalten, verändern Spiegelung und Gleitspiegelung ihn. Als Beispiel sei das Spiegelbild eines Wortes genannt.

dur MOTIV umzuschreiben und gleichzeitig R.MOTIV zu erstellen.

```
TO UMSCHR :PROZ
  OUTPUT UMSCHR.PROZ TEXT :PROZ
END
```

UMSCHR liest den Inhalt einer spezifizierten Prozedur, ändert ihn und gibt ihn unter anderem Namen wieder aus. Voraussetzung dafür ist, daß die zu verarbeitende Prozedur aus gültigen LOGO-Befehlen besteht und keine Unterroutinen enthält. UMSCHR ruft die folgenden Prozeduren auf:

```
TO UMSCHR.PROZ :TEXT
  IF :TEXT = [] THEN OUTPUT []
  OUTPUT FPUT UMSCHR.ZEILE FIRST
    :TEXT
  UMSCHR.PROZ BUTFIRST :TEXT
END
```

Diese Routine zerlegt die Eingabeprozedur durch Aufruf der folgenden Prozedur in Einzelteile:

```
TO UMSCHR.ZEILE :ZEILE
  IF ZEILE = [] THEN OUTPUT []
  IF LIST? FIRST :ZEILE THEN OUTPUT
    FPUT
  UMSCHR.ZEILE FIRST :ZEILE
    UMSCHR.ZEILE
  BUTFIRST :ZEILE
  OUTPUT FPUT
  AENDER.WORD FIRST :ZEILE
  UMSCHR.ZEILE BUTFIRST :ZEILE
END
```

UMSCHR.ZEILE rechnet jede Zeile durch und leitet die einzelnen Worte an AENDER.WORD weiter. Die mit IF LIST? beginnende Zeile wird benötigt, wenn MOTIV einen REPEAT-Befehl enthält. Die AENDER.WORD-Prozedur:

```
TO AENDER.WORD : WORD
  IF (ANYOF :WORD = "RT :WORD =
    "RIGHT) THEN OUTPUT "LEFT
  IF (ANYOF :WORD = "LT :WORD =
    "LEFT) THEN OUTPUT "RIGHT
  OUTPUT :WORD
END
```

Diese Routine überprüft jedes einzelne Wort und erledigt die erforderlichen Änderungen. Für den ersten Versuch wird eine einfache Figur definiert:

```
TO DREIECK
  REPEAT 3 [FD 50 RT 120]
END
```

Nun muß man DEFINE "REF UMSCHR "DREIECK eingeben und danach REF.

```
TO REF
  REPEAT 3 [FD 50 LEFT 120]
END
```

Es ist möglich, eine umfassendere UMSCHR-Routine zu schreiben, die auch alle Unterroutinen, die durch die Hauptprozedur aufgerufen werden, umschreibt. Achten Sie dabei auf recursive Prozeduren.

Es ist ebenfalls möglich, Muster aus den Routinen für die vier Grundumwandlungen zu entwickeln. Bei diesen Befehlen finden drei Unterroutinen Anwendung:

```
TO POSITION
  HT
  PU
  SETXY — 125 0
  PD
END
```

Damit wird die Turtle auf der linken Seite des Bildschirms positioniert und ist zeichenbereit.

```
TO BEWEGEN
  PU
  RT 90
  FD 50
  LT 90
  PD
END
```

BEWEGEN führt die gewünschte Translation aus und DREHEN dreht die Figur um 180 Grad.

```
TO DREHEN
  RT 180
END
```

Will man diese Prozeduren verwenden, muß zunächst ein Figur-Programm (z. B. QUADRAT) eingegeben werden. Nach Eingabe von MUSTER1 "QUADRAT wird das erste Muster gezeichnet.

Kompakt und kompatibel?

Die in den USA ansässige Osborne Corporation baute den ersten tragbaren Microcomputer, den Osborne-1. Nach Bewältigung finanzieller Schwierigkeiten stellte die Firma vor einigen Monaten den Osborne Vadem vor. Das neue Gerät reiht sich in die Menge IBM-kompatibler Microcomputer ein.

Der erste tragbare Computer, Osborne-1, löste eine Revolution im Bereich der Microtechnik aus. Mit seinem eingebauten Monitor, zwei Diskettenlaufwerken und Schnittstellen für Modem und Drucker war er der erste netzunabhängige CP/M-Computer, der kommerziell eingesetzt werden konnte. Obwohl 10,5 Kilo Eigengewicht die Bezeichnung „tragbar" eigentlich nicht ganz trafen, erkannten andere Hersteller sehr schnell, welches Potential die neue Maschine barg. Und schon sehr bald fand sich die Osborne Corporation im harten Wettbewerb.

Die Probleme von Osborne begannen 1981, als IBM die erste Maschine seiner PC-Reihe vorstellte. Aufgrund des bekannten Namens entschieden sich viele Geschäftsleute für den IBM PC, und der Branchenriese konnte den Markt im Nu erobern. Wie viele andere Computerhersteller kündigte auch die Osborne Corporation eine PC-kompatible Maschine an. Das neue Modell – der Osborne Executive – sollte mit zwei Prozessoren ausgerüstet sein und unter CP/M und MS-DOS laufen. Da zu diesem Zeitpunkt jedoch eine weltweite Knappheit an 8086-Prozessoren bestand, wurde die Maschine ohne den Chip herausgebracht, mit dem sie PC-kompatibel gewesen wäre. In der Folge ging der Verkauf von Osborne-Computern so sehr zurück, daß die Firma 1983 Vergleich anmelden mußte. Osborne schaffte es jedoch zu überleben und ist mit einem anderen Konzept jetzt hauptsächlich im Bereich Forschung und Entwicklung tätig.

Vor einigen Monaten präsentierte Osborne einen neuen Computer, den Osborne Vadem. Mit rund vier Kilogramm, bzw. 4,7 Kilo mit Akkumulator, wiegt der Vadem wesentlich weniger als sein Vorgänger. Wenn die Tastatur vor den Bildschirm geklappt ist, hat das Gerät etwa die Größe von drei übereinandergelegten Telefonbüchern. Das Gehäuse besteht aus strapazierfähigem Plastik. Die Tastatur wird von zwei Klammern gehalten.

Über den Schreibmaschinentasten im QWERTY-Format befindet sich eine Plastikmembran, unter der die Funktionstasten und die „Ikonen" (Symbole für die eingebauten Programme) liegen. In das Hauptgehäuse wurde eine 23,7×8 Zentimeter große Flüssigkristallanzeige eingebaut.

Mit Kontrolltasten auf beiden Seiten und vier Steuertasten für den Cursor (in der rechten unteren Ecke) macht die Tastatur einen professionellen Eindruck. Sie wirkt dennoch ein wenig überladen, da versucht wurde, alle Fähigkeiten des IBM PC auf kleinstem Raum unterzubringen. So besitzt der IBM PC rechts der Tastatur einen Zehnerblock für Zahleneingaben. Beim Vadem mußte dieser Block in die Haupttastatur integriert werden. Die Taschenrechnerfunktionen können wie die anderen eingebauten Programme über die Funktionsleiste oberhalb der Tastatur angesprochen werden. Die Symbole stellen Programmodule für Taschenrechner, Modem und Disketten und Kalenderroutinen dar. Die Funktionsleiste macht keinen sehr stabilen Eindruck und steht damit im Kontrast zur Tastatur.

Mit der Flüssigkristallanzeige haben die

Der Osborne Vadem ist eine der ersten IBM-kompatiblen Maschinen, die mit einer LCD-Anzeige statt mit der standardmäßigen Kathodenstrahlröhre arbeiten. Wenn die Tastatur über den Bildschirm geklappt ist, läßt sich das kompakte Gerät mit dem Schulterriemen problemlos transportieren.

Konstrukteure den Elektrizitätsbedarf wesentlich verringert, da ein LCD-Schirm weitaus weniger Strom verbraucht als die üblichen Kathodenstrahlröhren. Hier liegt auch die größte Schwierigkeit der IBM-Kompatibilität. Das Problem ist nicht die fehlende Farbdarstellung der LCD-Anzeige (die sich leicht durch unterschiedliche Grautöne ersetzen läßt), sondern die Größe des Bildschirms.

Die normale Textdarstellung des IBM PC besteht aus 25 Zeilen mit je 80 Zeichen. Da es dem japanischen Hersteller der LCD-Schirme bis zur Einführung des Vadem nicht gelang, einen entsprechend großen Schirm zu entwikkeln, war Osborne vorerst gezwungen, den Vadem mit einer 80×16-Darstellung auszurüsten. Das bedeutet, daß sich viele für den IBM geschriebene Programme auf dem Vadem nicht einsetzen lassen. Osborne plant jedoch, das Gerät mit einem 80×25-LCD-Display auszustatten, sobald diese in ausreichender Menge zur Verfügung stehen.

Auf der rechten Seite des Computers befinden sich zwei $5^1/_4$-Zoll-Diskettenlaufwerke. Auf der gegenüberliegenden Seite liegen der Anschluß für das Netzgerät, der Betriebsschalter, ein Knopf zur Kontrasteinstellung der Anzeige und eine Aussparung für Batterien. Mit einem speziellen Nickel-Cadmium-Pack kann die Maschine ohne Netzanschluß laufen.

Anschlußmöglichkeiten

Auf der Rückseite des Vadem befinden sich Anschlüsse für die Ein- und Ausgabe. Es gibt eine Telefonbuchse für das eingebaute Modem, eine Centronics-Druckerschnittstelle und eine RS232-Schnittstelle für den Anschluß von seriellen Druckern und externen Modems.

Zum Booten der MS-DOS-Systemdiskette braucht nur das entsprechende Symbol der Funktionsleiste gedrückt zu werden. Die eingebauten Module lassen sich unabhängig vom Betriebssystem und den gerade von Diskette geladenen Programmen aufrufen.

Der Osborne Vadem verarbeitet fast alle IBM-PC-Programme. Wegen des kleineren Bildschirmformates müssen jedoch viele Befehle vorerst „blind" eingegeben werden. Unter den Programmen, die auf dem Vadem fehlerfrei laufen, befindet sich auch Lotus 1-2-3. Dieses Programm ist auf anderen PC-kompatiblen Geräten nur äußerst schwierig zum Laufen zu bringen, da es die eingebauten Routinen auf ungewöhnliche Weise einsetzt.

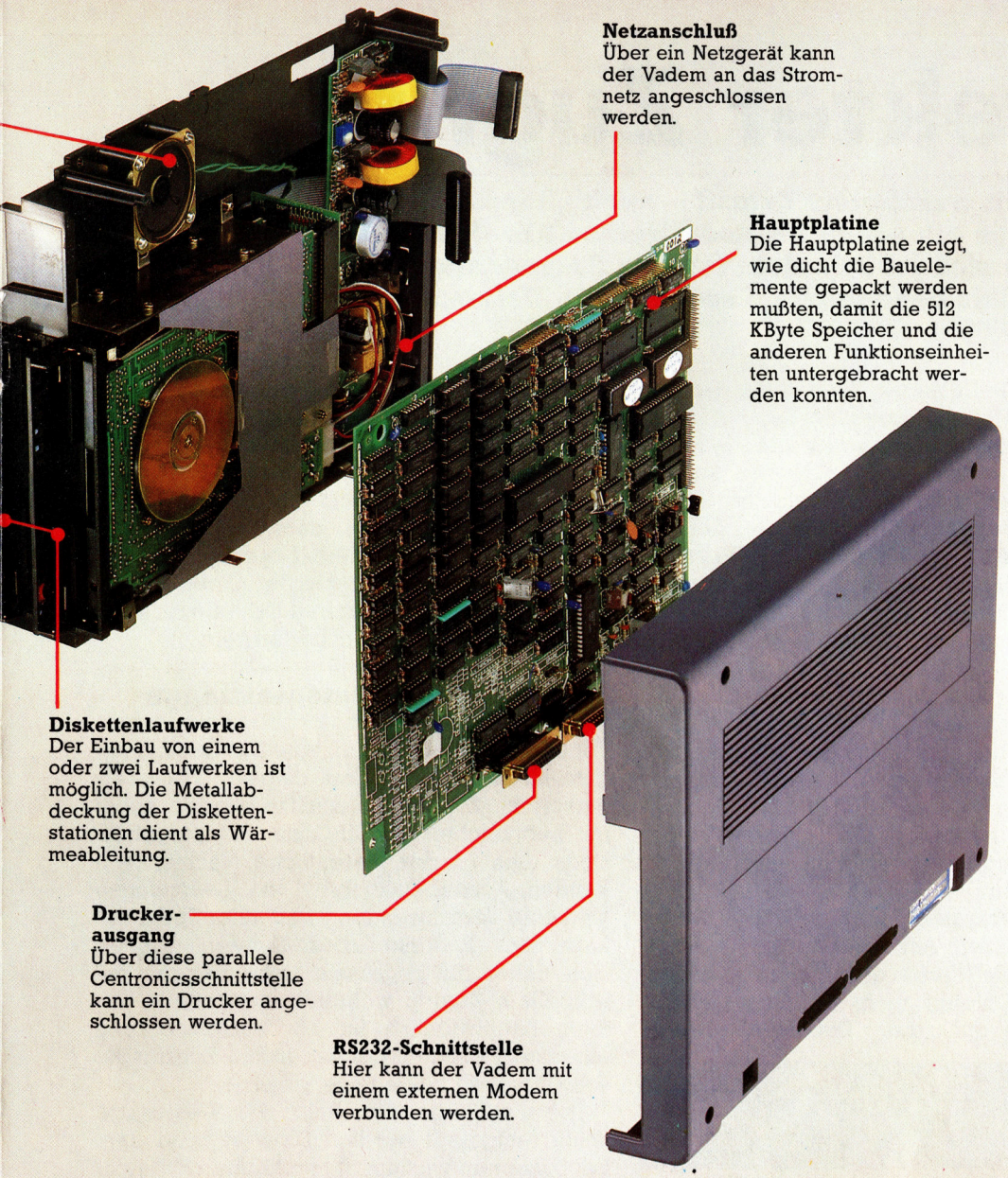
Es läßt sich nur schwer voraussagen, ob der Vadem ebenso erfolgreich sein wird wie der Osborne-1. An den Bildschirm muß man sich zunächst gewöhnen, da er wie alle LCD-Anzeigen für die Erzeugung lesbarer Zeichen viel Licht benötigt. Auf anderen kleineren Maschinen mit großformatigen Buchstaben ist dies kein Problem; die Zeichen des Vadem haben jedoch nur halbe Größe

Lautsprecher
Der kleine Lautsprecher erzeugt den Tastatur-„Beep".

LCD-Bildschirm
Durch die Flüssigkristallanzeige verbraucht der Vadem weitaus weniger Elektrizität als Geräte mit Kathodenstrahlröhren. Die Möglichkeit des Batteriebetriebs macht die IBM-kompatible Maschine zu einer interessanten Alternative.

Doppelte Funktion
Die Tastatur enthält weniger Tasten als die des IBM PCs, bietet aber die gleichen Funktionen. Einige Tasten lassen sich auch als Taschenrechner einsetzen.

Diskettenlaufwerke
Auf der rechten Seite des Gerätes können sich ein oder zwei Diskettenlaufwerke befinden. Unter dem Betriebssystem MS-DOS 2.11 lassen sich 360 KByte auf zweiseitigen Disketten mit doppelter Schreibdichte speichern.



Netzanschluß
Über ein Netzgerät kann der Vadem an das Stromnetz angeschlossen werden.

Hauptplatine
Die Hauptplatine zeigt, wie dicht die Bauelemente gepackt werden mußten, damit die 512 KByte Speicher und die anderen Funktionseinheiten untergebracht werden konnten.

Diskettenlaufwerke
Der Einbau von einem oder zwei Laufwerken ist möglich. Die Metallabdeckung der Diskettenstationen dient als Wärmeableitung.

Druckerausgang
Über diese parallele Centronicsschnittstelle kann ein Drucker angeschlossen werden.

RS232-Schnittstelle
Hier kann der Vadem mit einem externen Modem verbunden werden.

Osborne Vadem

ABMESSUNGEN
241×325×141 mm

ZENTRALEINHEIT
8086 16-Bit-Prozessor

SPEICHERKAPAZITÄT
Der Vadem ist wahlweise mit 256 oder 512K RAM erhältlich.

BILDSCHIRMDARSTELLUNG
LCD-Anzeige mit 16 Zeilen zu je 80 Zeichen oder 480×128 Pixel. Osborne hat den Einbau einer 80×25 Anzeige angekündigt, sobald diese verfügbar ist. Geräte mit altem LCD-Format sollen dann nachgerüstet werden können.

SCHNITTSTELLEN
Serielle RS232-Schnittstelle, RJ11-Telefonbuchse und paralleler Centronicseingang.

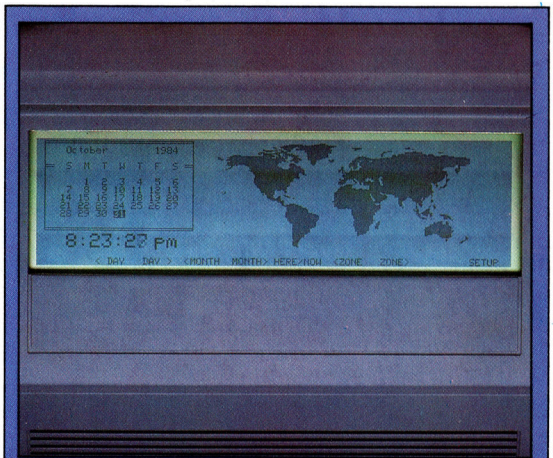
PROGRAMMIERSPRACHEN
Microsoft-BASIC wird auf Diskette mitgeliefert.

TASTATUR
62 Schreibmaschinentasten, eine Tastenfolie mit 10 Funktions- und 4 Symboltasten.

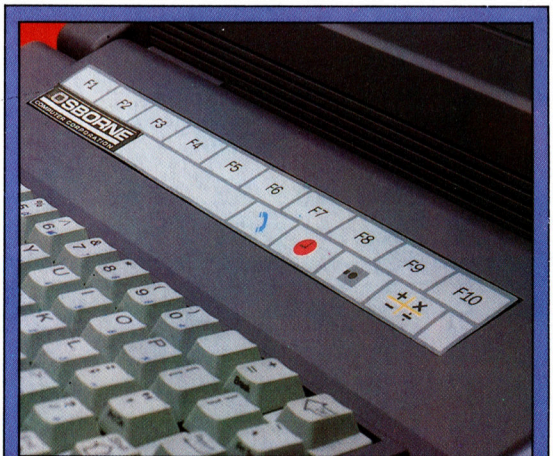
HANDBÜCHER
Die beiden Anwenderhandbücher sind wie alle Dokumentationen von Osborne ausgezeichnet geschrieben. Darin ist eine vollständige Beschreibung über die Funktionsweise der Maschine enthalten.

STÄRKEN
Der Vadem ist eine äußerst vielseitig und flexibel einsetzbare Maschine, die für vier bis fünf Stunden im Batteriebetrieb arbeiten kann.

SCHWÄCHEN
Der Bildschirm ist noch nicht voll IBM-kompatibel und teilweise schlecht zu lesen.



Zeitzonen
Das Bild zeigt die 80×16 Zeichen große LCD-Anzeige. Das Fenster unterhalb der Anzeige deutet an, daß für die Zukunft der Einbau eines Bildschirms mit 80×25 Zeichen vorgesehen ist. In der Anzeige ist der Terminkalender dargestellt.



Leicht berührt
Oberhalb der Tastatur befindet sich eine Folie, die der Membrantastatur des ZX81 ähnlich ist. Sie enthält zehn programmierbare Funktionstasten und sogenannte „Ikonen", mit denen sich die im ROM eingebauten Programme aufrufen lassen.

Direkter Zugriff

Dateien mit wahlfreiem Zugriff – auch „Random Access" genannt – ermöglichen ein schnelles und direktes Abrufen einzelner Datensätze. Sie eignen sich jedoch nicht für alle Anwendungen, da sie mehr Speicherplatz benötigen als sequentielle Dateien.

Sequentielle Dateien weisen einen Nachteil auf: Gespeicherte Daten können nur in der Reihenfolge gelesen werden, in der sie abgelegt wurden. Random-Access-Dateien kennen diese Beschränkung nicht. – Auf ihre Daten kann schnell, direkt und in beliebiger Reihenfolge zugegriffen werden, ohne daß alle vorangehenden Daten von Anfang an gelesen oder geschrieben werden müssen.

Mit Random-Access-Dateien läßt sich schneller und einfacher arbeiten als mit der umständlichen sequentiellen Speicherung. Die Unterteilung der Datei in Datensätze und Felder spielt dabei eine wichtige Rolle. So muß genau angegeben werden, welcher Datensatz angesprochen werden soll. Er wird dann in einen Buffer geladen, in dem einzelne Felder gelöscht, geändert oder auch ausgedruckt werden können.

Das Betriebssystem übernimmt beim Aufrufen einer Random-Access-Datei alle komplizierten Abläufe. Hierbei geschieht, was bei sequentiellen Dateien nicht möglich ist: Der Schreib/Lesekopf des Diskettenlaufwerks wird direkt an den Anfang des gewünschten Datensatzes gesetzt. Kurze Zugriffszeiten sind möglich, weil jeder Datensatz die gleiche Länge hat. Wenn beispielsweise jeder Datensatz 100 Bytes enthält und das Programm File 83 ansprechen möchte, dann setzt das Betriebssystem den Schreib/Lesekopf an den Anfang des 8300sten Bytes der Datei. Da die Länge der Diskettensektoren bekannt ist, kann die Position exakt berechnet werden.

Dateisysteme im Vergleich

	Random-Access-Dateien	Sequentielle Dateien
Pro	• Schneller Zugriff auf einzelne Datensätze.	• Speicherplatz optimal ausgenutzt. • Für Cassettensysteme geeignet.
Kontra	• Verschwendung von Speicherplatz. • Läuft nur mit Disketten.	• Langsam und umständlich zu handhaben.
Anwendungsbereich	• Begrenzte Datenmengen mit vorgegebener Struktur. • Zugriff auf eine kleine Anzahl verschiedener Datensätze. Beispiel: Katalog einer Bücherei. Mit wenigen Lesevorgängen soll ein bestimmtes Buch gefunden werden.	• Große Mengen unstrukturierter Daten. • Zugriff auf alle Datensätze einer Datei in einem einzigen Programmdurchlauf, Beispiel: Lohnabrechnung; sämtliche Kosten der Angestellten müssen bearbeitet werden. Es sind viele Lesevorgänge notwendig.

Einheitliche Datensatzlänge

Für die Dateistruktur muß eine Datensatzlänge gewählt werden, die dem längsten Datensatz entspricht. Kürzere Datensätze werden dabei mit Leerzeichen (ASCII-Code: 32) aufgefüllt. Doch dies ist der wesentliche Nachteil von Random-Access-Dateien: Die Leerzeichen sind eine Verschwendung wertvoller Speicherkapazität. Random-Access-Dateien eignen sich daher für begrenzte Datenmengen mit schnellem Zugriff, während sequentielle Dateien hauptsächlich für Massenspeicherung eingesetzt werden, bei denen die Zugriffsgeschwindigkeit keine Rolle spielt.

Die Feldlänge innerhalb der Datensätze sollte festgelegt werden. Dies ist besonders bei Systemen wichtig, die nicht nur auf Datensätze, sondern auch auf Felder wahlfrei zugreifen können. Der erste Schritt beim Aufbau einer Random-Access-Datei besteht daher immer aus der Erstellung einer Liste, in der die unterschiedlichen Felder mit ihren Längen aufgeführt sind. So sollte zum Beispiel das Namensfeld einer Anschrift mindestens zwanzig Zeichen fassen, während für eine Altersangabe zwei Bytes völlig ausreichen.

Da eine Dateistruktur fast immer ein Kompromiß zwischen der Menge der gespeicherten Information und der Anzahl der Datensätze ist, werden oft Codesysteme eingesetzt, die den Speicherbedarf verringern. So können etwa für die Farben Schwarz, Rot und Grün die Codes 1, 2 und 3 eingesetzt werden oder Datencodes wie 110785 für den 11. Juli 1985. Die Codes werden dabei nur intern verwandt und für die Bildschirmdarstellung wieder in lesbare Form umgewandelt.

Für die Länge der Datensätze sind noch weitere Faktoren wichtig. Viele Systeme geben eine maximale Datensatzlänge zwischen 128

Bytes und 2048 Bytes vor. Dabei ist es vorteilhaft, eine Länge zu wählen, die ein Vielfaches oder ein leicht teilbarer Abschnitt der Sektorengröße ist. Längen wie 64, 128, 256 oder 512 Bytes verhindern, daß die Anfänge der einzelnen Datensätze an unterschiedlichen Positionen der Sektoren stehen, und verringern damit durch kurze Wege die Zugriffszeit jeder einzelnen Abfrage.

Beim Ändern einzelner Datensätze werden die Daten zunächst gelesen, dann geändert und an die gleiche Stelle zurückgeschrieben. Datensätze kann man mit ihrer Nummer aufrufen. Da sich ein Anwender jedoch kaum die Nummern einzelner Datensätze merken wird, gibt es eine ganze Anzahl Techniken, Datensätze zu suchen. Oft wird das Namensfeld als Zugriffsschlüssel eingesetzt. Der Computer baut dabei einen Index auf, der den unterschiedlichen Namen die Nummern ihrer Datensätze zuordnet.

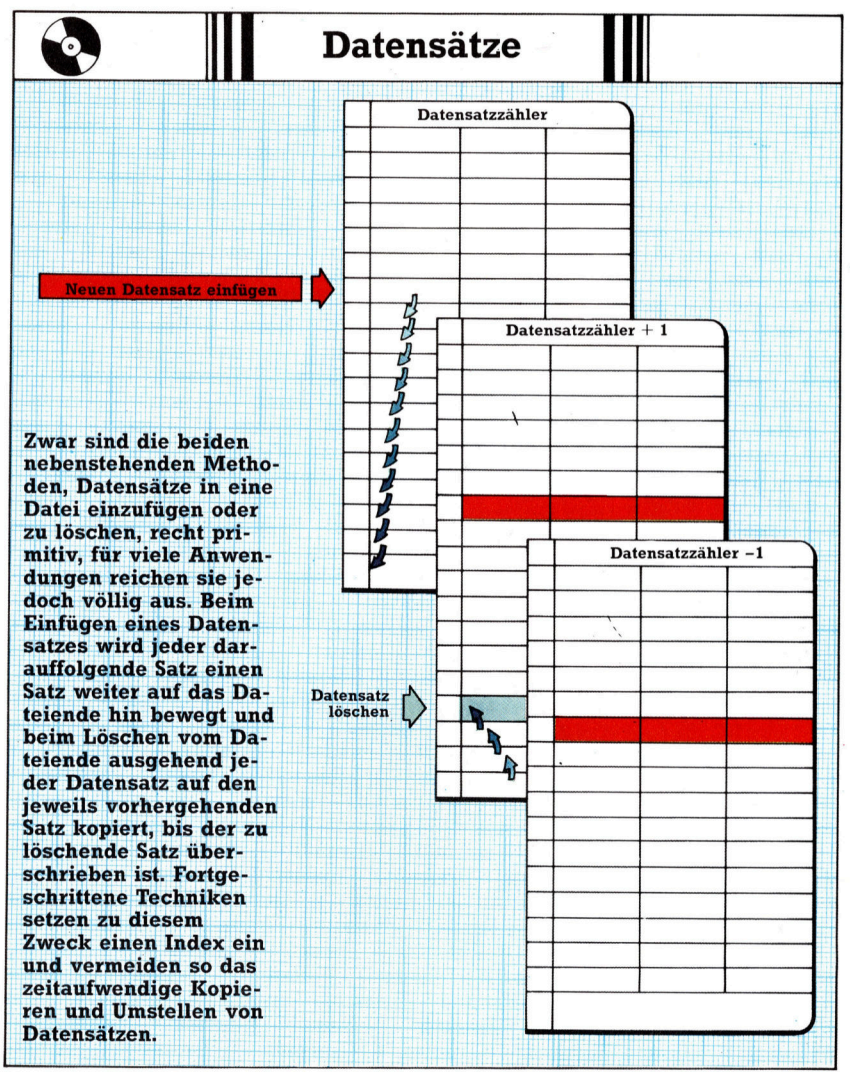
Random-Access-Dateien ohne Index müssen wie sequentielle Dateien Satz für Satz durchsucht werden, bis eine bestimmte Information gefunden ist. Ist die Datei jedoch nach bestimmten Kriterien sortiert, lassen sich zeitsparende Suchmethoden einsetzen. Nehmen wir an, wir suchen die Daten für „Müller" in einer Datei, deren Sätze nach den Namen sortiert sind. Wir lesen zuerst den Datensatz, der genau in der Mitte der Datei steht – und finden den Namen „Peters". Da „Müller" im Alphabet vor „Peters" steht, können wir nun alle Datensätze nach „Peters" ausschließen. Als nächstes lesen wir den Datensatz, der in der Mitte der ersten Hälfte steht. Hier lautet der Name „Heinemann", und der Bereich des gesuchten Datensatzes wäre damit weiter eingegrenzt und so weiter. Techniken dieser Art lassen sich noch weiter verfeinern. Viele Programme halten die Nummern der Datensätze mit vielen Zugriffen sogar direkt im Arbeitsspeicher und sprechen damit auch bei großen Datenmengen einzelne Datensätze schnell an.

Löschen und Einfügen

Vergleichsweise langsam kann das Löschen und Einfügen neuer Datensätze sein. Bei der einfachsten Methode, einen Datensatz zu löschen, wird der unmittelbar dahinter liegende Datensatz auf den freigewordenen Platz kopiert und die dort gespeicherte Information überschrieben. Dieser Vorgang wird mit allen folgenden Datensätzen wiederholt und schließlich der Zähler um eins vermindert. Auf ähnliche Weise lassen sich auch neue Datensätze an beliebigen Positionen einfügen. Dabei wird der letzte Datensatz einen Satz weiter geschoben, und alle davor liegenden Datensätze bis zu dem einzufügenden Satz werden auf den jeweils dahinter liegenden Satz kopiert. In der entstehenden Lücke wird die neue Information untergebracht.

Obwohl diese beiden Methoden nicht schnell sind, sind sie effektiver als ähnliche Vorgänge bei sequentiellen Dateien. Mit einer separaten Indexdatei läßt sich das Einfügen und Löschen einzelner Datensätze jedoch sehr beschleunigen. Dabei werden überflüssige Datensätze nur im Index als gelöscht gekennzeichnet, während der Datensatz selbst bestehen bleibt. Beim Einfügen neuer Datensätze werden als gelöscht gekennzeichnete Sätze überschrieben, und der Index wird auf den neuesten Stand gebracht.

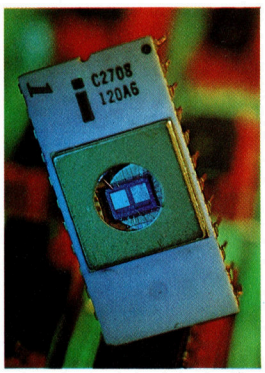
Eine Datenverwaltung nach dem Random-Access-System bietet noch weitere Vorteile. Für den Fall, daß einzelne Datensätze in falscher Reihenfolge abgelegt sind, enthalten einige Systeme Hilfsprogramme, die die gelöschten Sätze aus der Hauptdatei herausnehmen und die Datensätze reorganisieren und sortieren. Weiterhin enthält ein System, bei dem einzelne Datensätze nur als gelöscht gekennzeichnet sind, ein „Sicherheitsnetz", bei dem gelöschte Datensätze leicht reaktiviert werden können. Dieses „Sicherheitsnetz" besteht, solange die gelöschten Datensätze nicht überschrieben oder von einem Hilfsprogramm in der Hauptdatei gelöscht wurden.



Zwar sind die beiden nebenstehenden Methoden, Datensätze in eine Datei einzufügen oder zu löschen, recht primitiv, für viele Anwendungen reichen sie jedoch völlig aus. Beim Einfügen eines Datensatzes wird jeder darauffolgende Satz einen Satz weiter auf das Dateiende hin bewegt und beim Löschen vom Dateiende ausgehend jeder Datensatz auf den jeweils vorhergehenden Satz kopiert, bis der zu löschende Satz überschrieben ist. Fortgeschrittene Techniken setzen zu diesem Zweck einen Index ein und vermeiden so das zeitaufwendige Kopieren und Umstellen von Datensätzen.

EPROM-Brenner

Zur Programmierung eines EPROMs benötigt man einen speziellen EPROM-Brenner. Der hier beschriebene „Micron EPROM Programmer" ist eines der preiswerteren Geräte auf dem Markt.

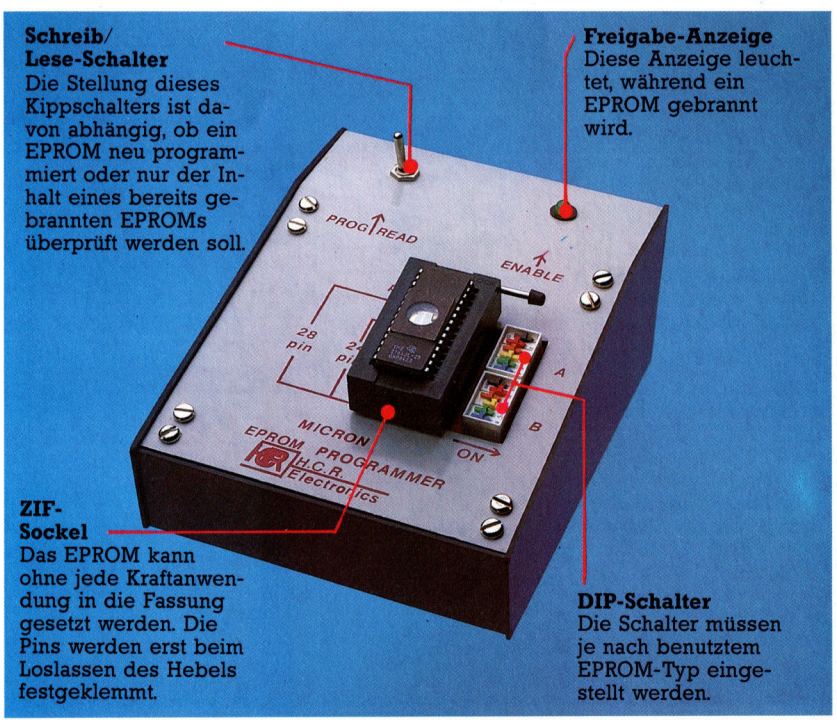


Durch das Quarzfenster des EPROMs ist der Siliziumchip zu sehen. Auch die Verbindungsdrähte zwischen dem Chip und den Anschlußstiften sind zu erkennen und mit etwas Mühe die Transistorfelder, die die Informationen enthalten.

Mit Hilfe dieses Geräts kann der Benutzer selbstgeschriebene Programme über den User Port des Acorn B in EPROMs „einbrennen", um sie auf den ROM-Steckplätzen des Rechners ständig griffbereit zu halten.

Einige Rechner verfügen über zusätzliche freie ROM-Steckplätze, die der Benutzer für spezielle Zwecke verwenden kann. ROMs mit Anwender-Programmen gibt es inzwischen in großer Auswahl bis hin zu Textverarbeitungs- und Datenbanksystemen oder Sprachen wie LOGO. Der Benutzer kann aber auch selbstgeschriebene Programme in ein EPROM (Erasable Programmable Read Only Memory = lösch- und programmierbarer Festwertspeicher) übertragen und dann in den Rechner einsetzen. Das EPROM wird mit einem speziellen Programmiergerät „gebrannt", das allgemein als „EPROM-Brenner" bezeichnet wird.

Wie alle ROMs besteht ein EPROM aus einer Leiterbahnen-Matrix, wobei an sämtlichen Zeilen/Spalten-Kreuzungen MOS-Speichertransistoren eingebaut sind. Ist ein Speichertransistor leitend programmiert, wird beim Anlegen eines Signals an die Zeile (Wortleitung) eine kreuzende Spalte (Bitleitung) spannungführend (logisch „1"). – Ist der Transistor gesperrt, bleibt die Spalte auf „0". Wenn die CPU über den Adreßbus ein Speicherbyte anfordert, wird auf diese Weise das Informationsmuster der acht Speichertransistoren, die auf der zugeordneten Leitung liegen, auf acht Spaltenleitungen übertragen und über den Datenbus dem Prozessor mitgeteilt.



Bei der EPROM-Herstellung sind die Speicherzellen im allgemeinen auf „1" gesetzt, das heißt, jede Adresse enthält den Hexadezimalwert FF. Die isoliert in Siliziumdioxid eingebetteten Gate-Elektroden der Speichertransistoren sind ungeladen, so daß alle Wortleitungen auf die Datenleitungen „durchgeschaltet" sind. Beim kurzen Anlegen von Überspannung (typisch sind 25V während 50ms) an einen Speichertransistor kommt es zu einem elektrischen Durchbruch im Siliziumdioxid, wobei sich das Gate so auflädt, daß der Transistor gesperrt wird. Wegen der isolierten Einbettung des Gate kann die Ladung nicht wieder abfließen, in der betreffenden Zelle ist daher dauerhaft eine logische „0" gespeichert. Beim Programmieren des EPROMs wird eine Wortleitung nach der anderen adressiert, zur Markierung der zu sperrenden Transistoren jeweils das gewünschte Bitmuster auf die Datenleitungen gegeben und dann der „Einbrenn"-Impuls angelegt.

Programmieren und Lesen

EPROMs können im Unterschied zu PROMs vom Anwender auch wieder gelöscht werden. Dazu ist auf dem Chip ein Quarzfenster angebracht. Bei UV-Bestrahlung wird die Siliziumdioxid-Isolation der Gates leitend, so daß sie sich entladen können. Damit sind die Speichertransistoren wieder durchgeschaltet und alle Bytes auf #FF gesetzt. Ultraviolettes Licht ist nicht unbedingt erforderlich. Es gibt auch „EEPROMs (Electrically Erasable . . .), die elektrisch gelöscht werden können.

Der „Micron EPROM Programmer" von HCR Electronics ist für EPROMs bis zu 16 KByte Speicherkapazität ausgelegt. Das kleine Gehäuse trägt einen speziellen Wechselsockel (ZIF = Zero Insertion Force), der EPROMs mit 24 oder 28 „Beinen" aufnehmen kann. Daneben sind zwei achtpolige DIP-Schalter plaziert, die je nach EPROM-Typ einzustellen sind. Links oben liegt ein Umschalter für Programmier- oder Lesebetrieb. Dieser Hebel verdient besondere Beachtung, weil Fehlbedienung zum Überschreiben eines frisch gebrannten EPROMs führen kann.

Das Gerät wird über ein Flachbandkabel mit dem User Port des Acorn B verbunden. Programme können mit LOAD und SAVE in einen Bufferbereich gebracht und dann auf ein EPROM übertragen werden. Dazu wird auf

Cassette eine menügesteuerte Software geliefert, natürlich mit der Empfehlung, zuerst hierfür ein EPROM zu brennen.

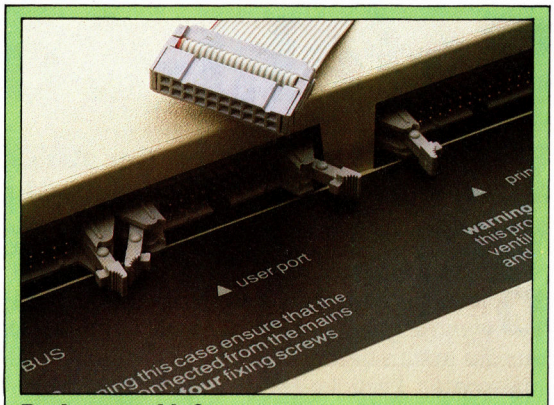
Der Bildschirm zeigt nach dem Laden der Software die anwählbaren Optionen: Vor dem Brennen des EPROMs ist Ihr Programm zunächst mit „L(oad)" in einen ab Byte #2000 reservierten Bufferbereich zu bringen. Anschließend wird der „P(rogram)"-Betrieb aufgerufen. Der Schirm zeigt dann eine Liste aller verwendbaren EPROM-Typen. Wenn Sie gewählt haben, folgt das Einstellschema für die 16 DIP-Schalter, wobei diejenigen blinken, die anders eingestellt werden müssen als beim vorherigen Programm. Zuletzt erscheint die Aufforderung, den Schreib/Lese-Schalter auf „Prog" zu kippen. Danach genügt ein Tastendruck, um des EPROM-Brennen zu starten. Die jeweils bearbeiteten Speicheradressen werden nun fortlaufend angezeigt.

Der Zeitbedarf für den Programmiervorgang hängt davon ab, wie viele Bytes auf #FF bleiben dürfen. Er liegt zwischen 74 Sekunden für ein kurzes Programm und maximal 14 Minuten für ein 16-KByte-EPROM. Mit dem Befehl „V(erify)" können Sie kontrollieren, ob Ihr Programm einwandfrei übertragen worden ist – die Software errechnet eine Prüfsumme und gibt fehlerhafte Adressen auf dem Schirm an. Ist alles in Ordnung, kann das Eprom vorsichtig in einen der vorgesehenen Stecksockel des Computers gesteckt werden.

Sie können für Ihr Programm einen Standard-Kopf festlegen, indem Sie mit dem Befehl „G" einen Namen bestimmen, der später (mit vorgesetztem *) über CALL aufgerufen werden kann. Bei einem BASIC-Programm sollten Sie den „G"-Befehl in Verbindung mit dem Zusatz „F(ill)" verwenden, damit unbenutzte Speicherplätze zur späteren Programmierung auf #FF bleiben. Maschinenprogramme können, auch wenn sie mit Kopf versehen sind, nur ab Adresse #2000 geladen werden.

Auch für Laien geeignet

Angesichts der Nützlichkeit solcher EPROM-Brenner ist es eigentlich erstaunlich, wie wenig sie bei Acorn-Besitzern verbreitet sind. Wahrscheinlich glauben viele, daß diese Geräte in das Reich der begeisterten Elektronikbastler gehören und für „normale" Heimcomputerbenutzer nicht in Frage kommen. Das ist schlicht falsch – so ein EPROM-Brenner ist auch für Laien einfach zu bedienen. Die Verwendung von EPROMs bietet viele Vorteile, wobei am wesentlichsten ist, daß Sie über das Betriebssystem ohne jede Verzögerung auf Ihre Programme zugreifen können, weil das Laden von Diskette oder Cassette entfällt. Außerdem sind die Programme im EPROM von BASIC-Programmen aus aufrufbar. Schließlich werden mit der Bereithaltung der Programme in EPROMs auch viele Störquellen ausgeklammert, die bei der Speicherung auf Magnetträgern die Zuverlässigkeit beeinträchtigen.



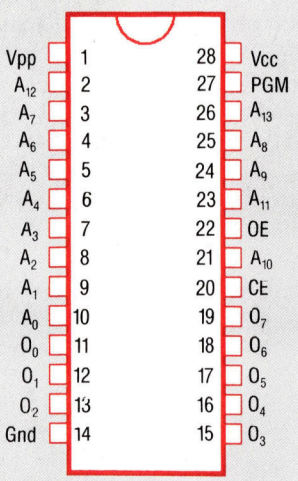
Rechneranschluß
Der EPROM-Brenner wird über ein zwölfadriges Flachbandkabel an den User Port des Acorn B angeschlossen.

EPROM-Brenner gibt es natürlich auch für andere Computer. Die für den Commodore 64 erhältliche Version besteht aus zwei Platinen, die durch ein Flachkabel miteinander verbunden werden sowie einem 2-KByte-EPROM, das die benötigte Software enthält. Die kleinere Platine wird in den Erweiterungssport des C 64 gesteckt. Auf der zweiten Platine befindet sich der eigentliche Programmierer, mit dem 8- und 16-KByte-EPROMs „geschossen" werden können. Die Bedienungsanleitung erklärt, was beim Programmieren zu beachten ist.

Kontaktsuche

Anschlußbelegung	
A_0-A_{13}	Adressen
CE	Chip-Freigabe
OE	Ausgangs-Freig.
O_0-O_7	Ausgänge
PGM	Programmieren
V_{pp} V_{cc}	Stromversorgung
Gnd	Masse (0 V)

Dies ist eine schematische Darstellung des 16-KByte-EPROMs TMS27128 von Texas Instruments mit 28 Anschlüssen, das beim HRC-EPROM-Brenner verwendet werden kann. 14 Pins werden für die Adressierung benötigt, weitere acht für den Datenbus. Mit „V_{cc}" (Pin 28) ist die normale Betriebsspannung, mit „V_{pp}" (Pin 1) die Überspannung für die Programmierung gemeint.



EPROM-Brenner

ABMESSUNGEN
140×105×65 mm

SCHNITTSTELLE
Flachbandkabel-Stekker für den User Port des Acorn B.

DOKUMENTATION
Drei fotokopierte Blätter mit einer Erläuterung der neun Systembefehle, die für Anfänger etwas schwierig sein kann. Dabei ist die Bedienung des Geräts eigentlich sehr einfach.

STÄRKEN
Die Handhabung ist problemlos; die menügesteuerte Software ist vom Benutzer auch ohne viel Verständnis für die technischen Vorgänge beim EPROM-Brennen anwendbar.

SCHWÄCHEN
Es gibt kaum Hinweise, wie der EPROM-Brenner mittels Erweiterung der Software eingesetzt werden kann. Maschinenprogramme, die nicht auf die Startadresse #2000 verlegt werden können, sind nicht ohne weiteres in ein EPROM zu programmieren.

Befehl-Codes

Die Commodore-Version der BASIC-Sprache kann wohl niemand als fortgeschritten bezeichnen, aber sie hat eine erstaunlich einfache Logik.

Das Commodore-BASIC gestattet die Verwendung langer Variablennamen, doch nur die ersten beiden Buchstaben werden vom Interpreter berücksichtigt. Somit sind Variablen wie KARIN und KAROLINA zulässig, werden jedoch als identisch angesehen. Das Ergebnis der folgenden Programmzeile ist:

```
100 KARIN=17:KAROLINA=2*KARIN
200 PRINT KARIN, KAROLINA
```

Ergebnis:

```
34          34
```

Dies gilt für alle Variablentypen: Fließkomma (Beispiel NUMMER), Integer (Beispiel NUMMER%), String (Beispiel NUMMER$) und Array (Beispiel NUMMER$(62,47)). Die Variablentypen selbst entsprechen dem Standard. Lediglich beim VC 20 können keine Integer-Variablen verwendet werden, da der Computer nicht über die Integer-Arithmetik verfügt.

Reservierte Wörter

Eine ärgerliche Konsequenz der Variablennamen-Regeln ist, daß ein gültig aussehender Variablenname ungültig sein kann, wenn seine zwei ersten Buchstaben ein reserviertes Wort ansprechen. START entspricht beispielsweise ST als Variablenname; ST ist jedoch ein reserviertes Wort.

Array-Variablen können bis zu 225 Dimensionen haben und sind in ihrer Größe nur durch den zur Verfügung stehenden Speicherplatz limitiert. Das erste Element in jedem Array ist Element(0). Das bedeutet, daß DIM EX(6) einen Bereich mit sieben Elementen reserviert: EX(0), EX(1), EX(2) . . . EX(6). Die DIM-Anweisung ist hier nicht unbedingt notwendig, da der Interpreter, wenn er eine eindimensionale Array-Variable findet, für die keine Dimensionierung vorgenommen wurde, einen Grundwert von 10 annimmt. Ist die Zuordnung eines solchen Bereiches größer als 10, erhält man einen BAD-SUBSCRIPT-Fehler.

Aufgrund der Arbeitsweise der meisten BASIC-Interpreter kann die Programmausführungsgeschwindigkeit durch Initialisierung der am häufigsten im Programm gebrauchten Variablen in der Reihenfolge ihrer Wichtigkeit erhöht werden. Dies ist mit Zuordnungsanweisungen oder mit der DIM-Anweisung möglich. Betrachten Sie folgende Zeile:

```
10 DIM A$(10,24),K,L,PUNKTE
```

Diese Zeile hat keinen sichtbaren Effekt, doch stellt sie einen schnellen Weg dar, die Variablen K, L und PUNKTE in der Variablentabelle weit vorne zu plazieren. Somit ist es für den Interpreter leichter, auf sie zuzugreifen, wodurch die Ausführungsgeschwindigkeit erhöht wird.

Ein Blick auf die Liste der BASIC-Schlüsselwörter offenbart einige Nachteile und Ergänzungen zum Microsoft-Befehlssatz. Der größte Nachteil dürfte das Fehlen von INKEY$ sein, und die wichtigsten Ergänzungen sind TIME$ und STATUS.

INKEY$, die Abfrage-Funktion für die Tastatur, wurde durch die Anweisung GET ersetzt. Ähnlich wie INKEY$ bewirkt diese Anweisung, daß das erste Zeichen des Tastatur-Buffers genommen und dessen ASCII-Wert ausgegeben wird. GET wird zumeist in Anweisungen wie der folgenden verwendet:

```
150 GET GT$:IF GT$="" THEN 150
```

Mit einer solchen Programmzeile wird die Programmausführung angehalten, bis eine Taste gedrückt wird. In diesem Fall beinhaltet GT$ dann das entsprechende Zeichen. Dies muß nicht immer so sein, da GET den Tastatur-Buffer abfragt und nicht die Tastatur. Enthält der Tastatur-Buffer bei der Ausführung von GET noch Zeichen, wird die Programmausführung dementsprechend nicht angehalten. Betrachten Sie hierzu das folgende Programm:

```
 50 FOR K=1 TO 100:PRINT K:NEXT K
 60 PRINT "DRUECKE EINE TASTE"
150 GET GT$:IF GT$="" THEN 150
200 PRINT "SIE DRUECKTEN DIE TASTE ";
    GT$
```

Wenn Sie dieses Programm laufen lassen, erscheinen die Zahlen von 1 bis 100 auf dem Bildschirm, gefolgt von der Eingabeaufforderung. Danach wartet der Rechner, bis Sie eine Taste drücken. Wenn Sie das tun, während die Zahlen auf dem Bildschirm ausgegeben werden, wird das gedrückte Zeichen im Tastatur-Buffer gespeichert und dann von der GET-Anweisung gefunden. Die Programmausführung wird also nicht mehr angehalten. Dies kann sehr ärgerlich sein – besonders in einem

Spiel, wo die verschiedensten Tasten recht hektisch gedrückt werden. Die Lösung dieses Problems ist, den Tastatur-Buffer kurz vor Ausführung der GET-Anweisung zurückzusetzen (zu „entleeren"), indem folgende Anweisung in das Programm integriert wird:

```
149 POKE KBPTR,0
```

KBPTR ist hierbei die Adresse des Tastatur-Zählers (198 beim Commodore 64).

TIME$ (normalerweise mit TI$ abgekürzt) ist die System-Uhr. Beim Einschalten des Computers wird sie mit "000000" initialisiert und gibt ab diesem Moment die Zeit in der Form „Stunden, Minuten und Sekunden" an bis zu "235959" – 23 Stunden 59 Minuten und 59 Sekunden seit Initialisierung – und wird dann wieder auf "000000" gesetzt. Diese Uhr steht dem Anwender wie jede andere Variable zur Verfügung und kann auf eine beliebige Zeit eingestellt werden (z. B. TI$="000000" oder TI$="084503").

Status-Anweisungen

Zusammen mit TI$ gibt es TI, das numerische Gegenstück. TI enthält den Wert der seit Initialisierung vergangenen Zeit in Sechzigsteln einer Sekunde (genannt „jiffys"), so daß TI einen Wert von 0 bis 5183999 enthalten kann (60*60*60*24-1). TI ist abhängig von TI$ und kann nicht initialisiert werden.

STATUS (abgekürzt als ST) ist eine Systemdefinierte Variable. Wenn bei einem Eingabe/Ausgabe-Gerät ein Fehler auftritt, enthält ST einen dem Fehler entsprechenden Wert. In einem Programm sähe das so aus:

```
330 IF ST > 0 THEN GOSUB 30000: REM
    I/O-FEHLERROUTINE
30000 REM FEHLERMELDUNG
30100 IF ST=16 PRINT "NICHT BEHEB-
    BARER LESEFEHLER"
```

Weiterhin gibt es CMD. Es ist ein nützlicher Bestandteil des Commodore-Befehlssatzes. Seine Funktion ist, eine Ausgabe vom Bildschirm zu einem bestimmten Ausgabekanal umzuleiten. Hierfür gibt es viele sinnvolle Anwendungen, wie beispielsweise:

```
OPEN 4,4:CMD 4:LIST
```

Damit wird das Listing des im Speicher befindlichen Programms nicht auf dem Bildschirm, sondern auf dem Drucker ausgegeben. Ist die Anweisung ausgeführt, sollte der Kanal wieder geschlossen werden:

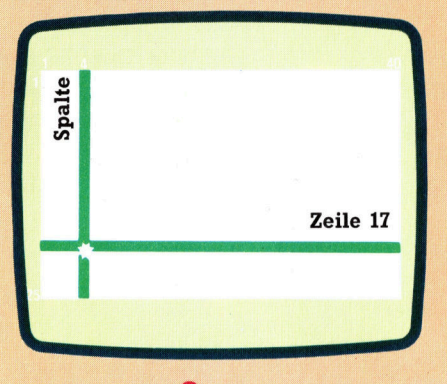
```
PRINT#4:CLOSE 4
```

Obwohl normalerweise das Fragezeichen (?) als Abkürzung für PRINT verwendet werden kann, ist es nicht möglich, PRINT# mit ?# abzukürzen – Sie müssen pR (p gefolgt von einem SHIFT R) als Abkürzung verwenden.

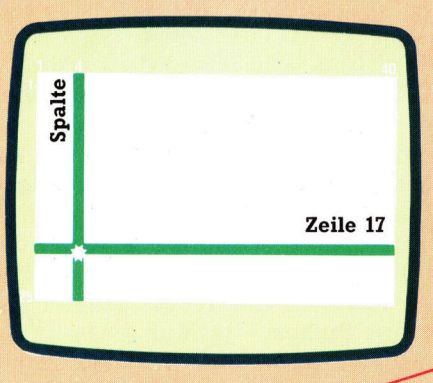
Fast alle Schlüsselwörter können mit ihrem Anfangsbuchstaben und dem zweiten Buchstaben in geSHIFTeter Form abgekürzt werden. Haben zwei oder mehr Schlüsselwörter dieselben beiden Anfangsbuchstaben, besteht die Abkürzung aus den ersten beiden Buchstaben und dann dem dritten Buchstaben in geSHIFTeter Form: READ, RESTORE und RETURN werden beispielsweise mit rE, reS und reT abgekürzt.

Der Bildschirm-Editor und das robuste, ihm zugrunde liegende anwenderfreundliche Betriebssystem sind wohl die bemerkenswertesten Details der Commodore-Computer. Um beispielsweise eine Programmzeile zu editieren, brauchen Sie die Zeile nur zu LISTen, mit dem Cursor direkt zu jedem beliebigen Punkt der Zeile zu gehen, den Text zu editieren und RETURN zu drücken. Dabei ist es völlig unwichtig, wo sich der Text auf dem Bildschirm befindet oder wo der Cursor innerhalb der Zeile positioniert ist. Sobald Sie die RETURN-Taste drücken, wird der Text der Bildschirmzeile, auf der sich der Cursor befindet, vom System aufgenommen.

Positionieren des Cursors




```
100 PRINT AT(17,4)"*"
```




```
50 POSITION$="
100 PRINT LEFT$(POSITION$,17)TAB(4-1)"*"

50 POSITION$="
100 ROW=17:COLUMN=4:GOSUB 1000:PRINT "*"
500 END
1000 PRINT LEFT$(POSITION$,ROW)TAB(COLUMN-1);:RETURN
```

Die Möglichkeit des Commodore, Cursor-Steuerbefehle in Strings abzulegen, gestattet, auf einfache Art Grafiken zu erstellen – besonders in Verbindung mit den komfortablen Bildschirm-Editier-Befehlen.

Wenn das BASIC des Commodore den PRINT-AT-Befehl enthalten würde, wäre die Positionierung des Cursors einfach.

Da dies nicht der Fall ist, müssen wir die programmierbaren Cursor-Möglichkeiten verwenden:

Initialisieren Sie POSITION$ mit einem HOME und 24 CRSR DOWNs. Fügen Sie dann die Reihen- und Spalten-Parameter in diesen Ausdruck ein.

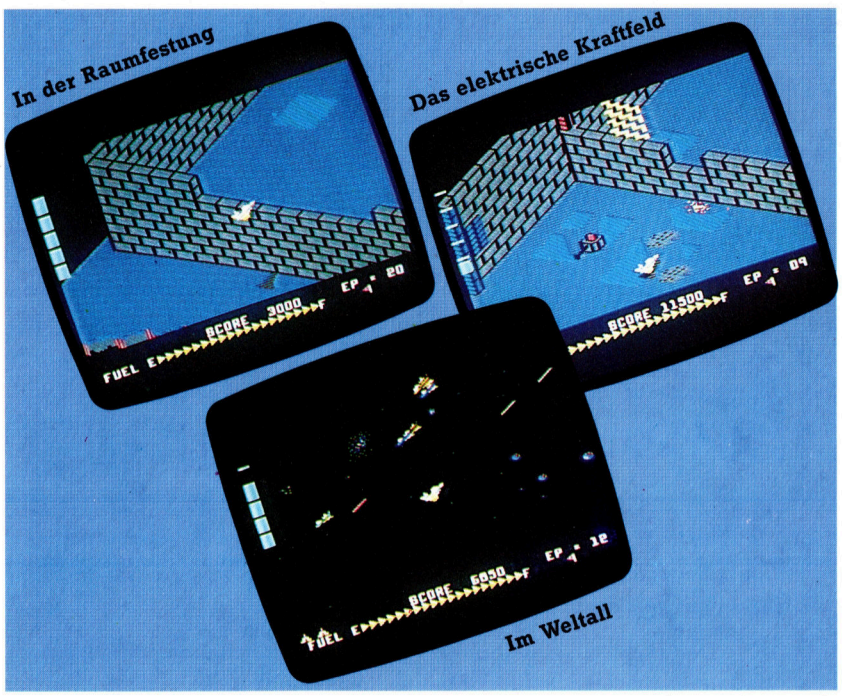
Wenn Sie häufig auf dem Bildschirm formatieren müssen, ist es sinnvoll, die entsprechenden Positionierungsbefehle in einer Unterroutine abzulegen, und dann die Variablen REIHE und SPALTE mit den Bildschirmpositionen zu initialisieren.

Raumfestung

Viele der bestverkauften Computerprogramme wurden ursprünglich für Arcadengeräte entwickelt und erst später für Heimcomputer adaptiert. Hier stellen wir ZAXXON vor, ein Programm, das für die Spielhallen produziert wurde und seit einiger Zeit für viele Heimcomputer erhältlich ist.

Hier werden drei verschiedene ZAXXON-Phasen gezeigt. Zu Spielbeginn fliegt der Spieler mit seinem Raumschiff in die erste der beiden Raumfestungen. Das Schiff muß in korrektem Winkel und richtiger Höhe gesteuert werden, um die Mauer zu überqueren. Innerhalb der Festung befindet sich eine weitere Mauer, die durch ein elektrisches Feld geschützt ist. Durch Zerstören der Waffentürme kann man Bonuspunkte erzielen. Anschließend gelangt man wieder in den Weltraum und muß gegnerischen Schiffen ausweichen. Sind alle Hindernisse der zweiten Festung überwunden, steht das große Finale mit dem Roboter ZAXXON an.

Spielziel bei ZAXXON ist es, ein Raumschiff an verschiedenen Hindernissen vorbeizusteuern und den Roboter ZAXXON zu zerstören. Bei Spielbeginn fliegt das Raumschiff durch die Weite des Alls. Plötzlich taucht vor dem Spieler eine von hohen Mauern umgebene „Raumfestung" auf. Ein Weiterkommen ist nur möglich, wenn man durch eine Lücke in der Mauer fliegt und den Hindernissen, die sich einem in den Weg stellen, ausweichen kann. Darunter befinden sich Raketen, Radarschirme sowie Treibstofftanks. Alles ist sehr realistisch in Quasi-3-D-Grafik dargestellt. Die



Treibstofftanks sind besonders wichtig: Zerstört man sie, wird der Treibstoffvorrat des eigenen Raumschiffes entsprechend erhöht.

Die Grafik dieses Programms ist ungewöhnlich gut. Für die Darstellung des Raumfahrzeugs verwendeten die Programmierer drei verschiedene Sprites – jeweils eines für das Fliegen nach unten, den Steigflug und die normale Flugrichtung. Beim Heben und Senken des Schiffes kommt die Wirkung der Sprites besonders gut zum Ausdruck. Die auf dem Boden befindlichen Hindernisse sind ebenfalls ausgezeichnet dargestellt. Besonders augenfällig ist der Schatten des Schiffs: Bei Höhenverlust wird der Schatten größer.

Wird das Schiff in dieser Spielphase durch feindliches Feuer zerstört, geht's zurück in die Ausgangsposition. Nach entsprechender Übung aber ist es unproblematisch, den Hindernissen auszuweichen. Das Zerstören der Radarschirme ist einen Versuch wert, da anschließend Abwehrraketen auf den Spieler abgefeuert werden. Bevor sie zerstört sind, müssen sie dreimal voll getroffen werden. Zuweilen ist das Steuern des Schiffes in der richtigen Höhe schwierig, da Fehleinschätzungen leicht möglich sind und man dann direkt gegen die ankommenden Raketen fliegt. Elektrischen Kraftfeldern weicht man durch Überfliegen aus und fliegt anschließend sofort nach unten. Dort warten Abwehrjäger.

Wurden die Abwehrvorrichtungen der Festung erfolgreich überwunden, gleitet das Schiff wieder in das Weltall, wo es von einem ganzen Schwarm feindlicher Raumschiffe attackiert wird. Hier stellt sich dem Spieler das Problem, ein dreidimensionales Spiel auf einem zweidimensionalen Bildschirm spielen zu müssen. Um den Gegner angreifen zu können, müssen Sie das Schiff im richtigen Moment hochziehen bzw. herunterdrücken.

Anschließend gelangt man wieder in die Raumfestung. Die Lücke in der Mauer ist diesmal erheblich schmaler, und die Ziele sind schwerer zu treffen. Dies ist der Augenblick, in dem der Spieler mit dem Roboter ZAXXON selbst konfrontiert wird, der über ein Rasterfeld auf ihn zukommt. Nach Erreichen dieser Phase stoppt das Schiff automatisch. Die einzige Abwehrmöglichkeit ist die, den Roboter zu zerstören, bevor er Ihr Schiff vernichtet. Dazu sind wiederum drei Volltreffer mit Automatik-Raketen erforderlich.

ZAXXON: Für Sinclair ZX Spectrum, C 64 und Atari

Hersteller: DataSoft/US Gold

Autor: Peter Adams

Joysticks: Erforderlich

Programm: Cassette (Spectrum/C 64), Diskette (Commodore 64, Atari Computer)

Das Wissen

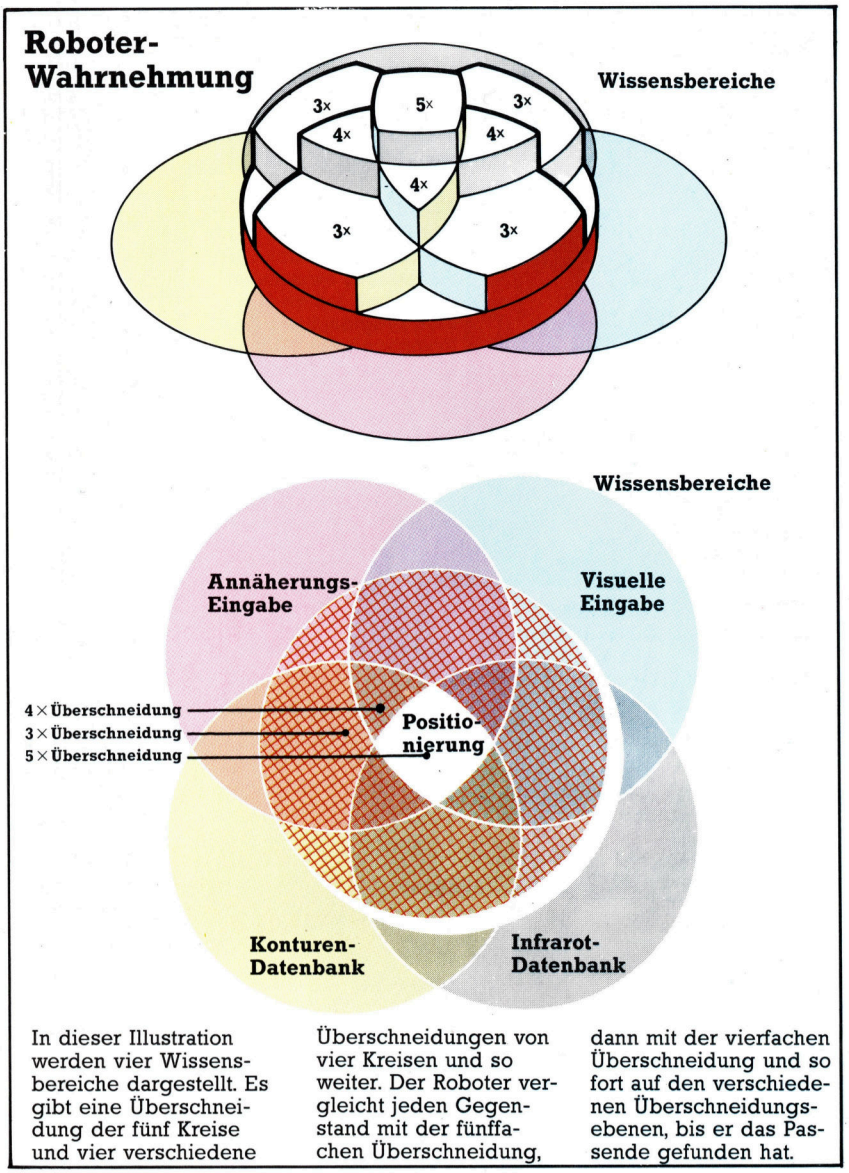
In den anderen Teilen dieser Serie wurden detailliert die Wahrnehmungs-Vorrichtungen vorgestellt, die für die „Intelligenz" eines Roboters wichtig sind. Wie diese Sensoren oder Sinne kombiniert werden können, um dem Roboter ein besseres Verständnis seiner Umgebung zu ermöglichen, zeigen wir hier.

Bei der Beschreibung der Wirkungsweise der einzelnen Sensoren, die ein Roboter benötigt, um zumindest einen Teil seiner Umgebung begreifen zu können, wurden die sensorischen Inputs (Sehen, Hören, Fühlen) einzeln erläutert. Zum Verständnis seiner Umgebung muß der Roboter jedoch imstande sein, die sensorischen Eingaben einander zuzuordnen. Das heißt, die Inputs des einen Sensors dienen zur Kontrolle der anderen usw. Erst so ist die Konstruktion eines vollständigen roboterinternen Weltbildes möglich.

Dieser Vorgang entspricht der Wahrnehmung des Menschen. Unsere Sinne arbeiten ebenfalls nicht isoliert voneinander. Wir nutzen unbewußt ständig einen Input (oder Reiz), um eine andere Wahrnehmung auf Richtigkeit zu überprüfen, und erhalten als Ergebnis ein vollständiges Bild unserer Umwelt. Bestes Beispiel dafür sind Experimente, die mit blind geborenen Menschen gemacht wurden. Sie bekamen durch einen chirurgischen Eingriff ihr Augenlicht. Die Patienten verstanden überraschend schnell, ihren Augensinn zu nutzen. Grund dafür ist, daß Blinde ein sehr gutes Wissen über die Umwelt haben, weil sie Gegenstände berühren, sich in der Umgebung bewegen können und „Beschreibungen" der Umwelt akustisch wahrnehmen. Ist ihnen das Augenlicht gegeben, verbinden sie ihr Wissen mit den neuen Eindrücken und können so die Dinge, die sie sehen, „wiedererkennen".

Soll ein Roboter ähnlich funktionieren, muß eine Interaktion seiner Sinne wie beim Menschen möglich sein. Ein Roboter, der Gegenstände aufheben soll, kann das natürlich auch „blind" tun. Doch besser wäre es, wenn er mit einem Sichtsystem ausgestattet wäre, da er dann auch falsch plazierte Objekte greifen oder sogar in einem anderen als dem pro-

Wir haben einen imaginären Roboter-Butler geschaffen, der eine Reihe sensorischer Wahrnehmungen zur Ausführung seiner Aufgaben miteinander verknüpfen muß. Sein größtes Problem besteht darin, daß die Gäste sich ständig bewegen. Das bedeutet: Er muß sein „Welt-Modell" entsprechend ständig anpassen.



In dieser Illustration werden vier Wissensbereiche dargestellt. Es gibt eine Überschneidung der fünf Kreise und vier verschiedene Überschneidungen von vier Kreisen und so weiter. Der Roboter vergleicht jeden Gegenstand mit der fünffachen Überschneidung, dann mit der vierfachen Überschneidung und so fort auf den verschiedenen Überschneidungsebenen, bis er das Passende gefunden hat.

Roboter stehen verschiedene Wissensquellen zur Verfügung: Das können vorprogrammierte Datenbanken sein (Sammlungen von Objektkonturen oder Infrarot-Mustern) oder Erfahrungs-Datenbanken (etwa die Karte, die der Roboter sich im Augenblick von seiner Umgebung geschaffen hat) und sensorische Wahrnehmungen oder Eingaben. Im Idealfall, wenn der Roboter seine Umgebung „kennt" und sie „versteht", ist durch Überschneidung der einzelnen Bereiche eine eindeutige Identifizierung jedes vom Roboter gesehenen Gegenstandes möglich.

grammierten Winkel den Arm bewegen könnte.

Einfachstes Beispiel dieses internen Modells sind die im Labyrinth herumlaufenden Roboter, die an anderer Stelle bereits vorgestellt wurden. Sie benutzen Sensoren, um die Wände des Labyrinths zu erkennen und erstellen eine komplette zweidimensionale Karte des Labyrinths. Übertragen wir dieses Prinzip auf einen Roboterarm, muß die Karte dreidimensional sein. Ist der Roboter zudem mit Sehvermögen ausgestattet, wird die Karte um Farben, Helligkeitsunterschiede und Muster ergänzt. Mit Spracherkennung ausgestattet erhielte der Roboter zusätzliche gesprochene Informationen für sein „Weltmodell".

Das Problem der Roboterkonstrukteure aber ist, daß die Welt nicht statisch ist, sondern sich ständig verändert. Der Roboter muß also entsprechend gebaut sein, um solchen Veränderungen folgen zu können.

Betrachten wir diese Schwierigkeit an einem Roboter, der lediglich für das Stapeln von Ziegeln programmiert wurde. Sind die Ziegel nicht gleich groß, müssen sie sehr genau aufeinandergesetzt werden. Befindet sich der Schwerpunkt einzelner Ziegel außerhalb der Mitte des Stapels, wird er irgendwann umstürzen. Doch ein Roboter kennt die Gesetze der Schwerkraft nicht. Und wenn der Stapel umfällt, weiß er nicht, was überhaupt geschieht und was er nun tun sollte.

Es gibt zwei Wege, dieses Problem zu lösen. Einmal ist es möglich, den Roboter mit einem Programm auszustatten, das jede Eventualität berücksichtigt und entsprechende Anweisungen enthält. Dabei aber ist das Verständnis des Roboters auf bestimmte, klar definierte Aufgaben begrenzt. Der Ziegel stapelnde Roboter wäre nur mit Anweisungen programmiert, die ihm ein exaktes Stapeln unter Berücksichtigung der Schwerkraft und des Gleichgewichts erlauben.

Heuristik auch für Computer

Die zweite Lösungsmöglichkeit ist die, den Roboter seine Umwelt durch Lernen begreifen zu lassen. Dieser Bereich der Computerwissenschaft wird als „Heuristik" bezeichnet. Der Roboter ist für die Ausführung einer bestimmten Aufgabe programmiert, erhält ein Feedback entweder durch einen Menschen oder einen Sensor und erfährt so, ob die Aufgabe richtig gelöst wurde. Aufgrund dieser Rückmeldung kann der Roboter sein Programm entsprechend der veränderten Situation modifizieren und eine Art „Handbibliothek" aufbauen, die künftig befragt wird.

Hat der Roboter seine Lektion „gelernt", muß diese als Computerprogramm gespeichert werden. Dieser Vorgang wird als „Wissens-Darstellung" bezeichnet. Die „Kenntnisse" eines Roboters können auf traditionelle Art in Form von Datencodierung gespeichert werden. Doch die Techniken der künstlichen Intelligenz bieten andere Möglichkeiten, darunter „Produktionsanweisungen", „semantische Netze" und „Raster".

Produktionsanweisungen sind beispielsweise IF... THEN-Befehle und einfache Kommandos. Das Wissen eines Roboters könnte also folgendermaßen gespeichert werden: IF (wenn) vor dir eine Ziegelwand ist, THEN (dann) kannst du nicht weiter gehen. Aus Regeln dieser Art lassen sich ganze Sequenzen bilden. Ihr Vorteil ist, daß der Programmierer sie leicht schreiben und leicht verstehen kann. Der „Haken" an der Geschichte ist aber, daß auch der Roboter sie verstehen soll. Deshalb muß er mit einer sogenannten „Folgerungs-Maschine" ausgestattet sein. Nur so kann er die Regeln interpretieren und daraus die erforderlichen Aktionen ableiten. Solche Programme können in konventionellen Programmiersprachen wie etwa BASIC geschrieben sein. Doch meistens werden Sprachen wie

PROLOG eingesetzt, die sich für diese Anwendung besser eignen. Dabei überprüft das Programm ständig eine bestimmte Zahl von Faktoren oder Umständen, mit denen eine oder mehrere Regeln übereinstimmen könnten. Ist dies der Fall, wird die entsprechende Bewegung ausgeführt. Ungültige, also nicht anwendbare Regeln, werden gelöscht.

Semantische Netze sind eine Form grafischer Strukturen, mit denen Wissen dargestellt werden kann. Man muß sie sich als Beziehungen zwischen verschiedenen Einzelelementen vorstellen. Der Grund dafür, warum man sie als semantische Netze bezeichnet und nicht als einfache Netze, ist: Die individuellen Verbindungen können unterschiedliche Bedeutungen haben. Ein „Bogen", der zwei „Knoten" miteinander verbindet, läßt sich unterschiedlich interpretieren.

Ein als „Tisch" bezeichneter Knoten kann mit einem als „Möbel" etikettierten Knoten verbunden werden. Die daraus abzuleitende Verwandtschaft besagt, daß der Tisch eine bestimmte Art Möbel ist. Wir haben es mit einer „Typen"-Verbindung zu tun.

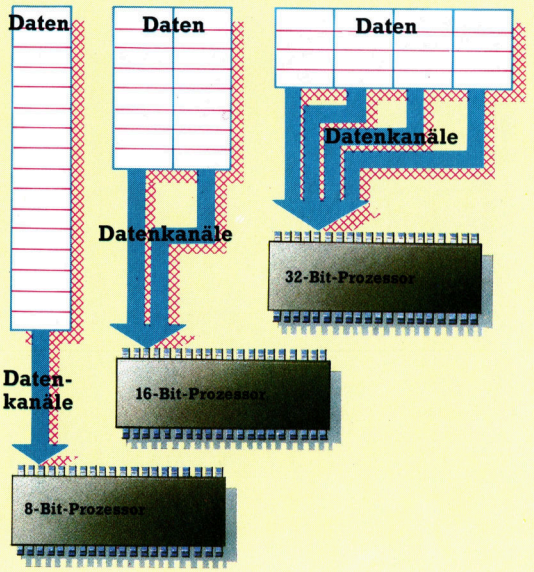
Diese Art der Wissensdarstellung kann in herkömmlichen Sprachen erfolgen. Man könnte das aber auch in BASIC versuchen und die verschiedenen Knoten und Verbindungen durch String-Variablen darstellen. Üblicher in der Anwendung sind aber Sprachen wie LISP, da mit diesen so komplexe Zusammenhänge leichter ausdrückbar sind.

Die Raster-Methode

Raster lassen sich mit leeren Fragebögen vergleichen, die speziell für jede mögliche Situation, mit der ein Roboter konfrontiert werden könnte, entwickelt wurden. Das Prinzip ist einfach und läßt sich in BASIC durch einfache zwei-dimensionale String-Arrays programmieren. Die erste Dimension beinhaltet die „Fragen", die zweite die „Antworten".

Bei diesem Verfahren hat der Roboter keine vollständige Kenntnis über einen Sachverhalt oder eine Situation, bevor er nicht jedes Fragebogen-Feld „ausgefüllt" hat. Erst danach kann er handeln. Diese Methode läßt sich durch eine entsprechend große Anzahl verschiedener Raster verfeinern. Eine Aufgabe des Roboters besteht dann darin, für eine gegebene Situation das Richtige auszuwählen.

Wir haben in diesem Beitrag eine Reihe von Roboter-Programmierungsmöglichkeiten betrachtet, die ein Verständnis der Umwelt erlauben. Doch in den Computer-Forschungs-Laboratorien der Universitäten wird weitergearbeitet, um die Schlüsselelemente wie Sensoren, Rückkoppelung, Lernen und Wissens-Darstellung zu verfeinern und zu verbessern. In einem anderen Teil werden wir uns mit Computer-Simulationen befassen, die dazu dienen, echte Vorgänge nachzuahmen.

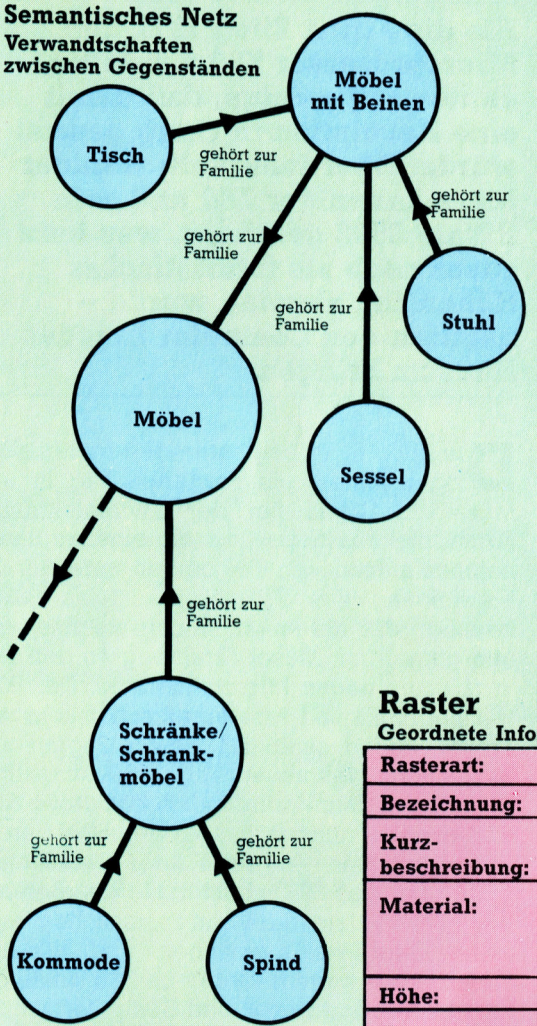


Parallele Eingaben

Die in das Datenverarbeitungssystem des Roboters gelangenden Informationen müssen schnell interpretiert werden, damit eine rasche Reaktion möglich ist. Die Länge der eintreffenden Daten-„Schlange" wird durch die verarbeitende mathematische Gleichung bestimmt, die die Verarbeitungsgeschwindigkeit steuert. Das von einem neueren Roboter erzeugte Datenvolumen übersteigt die Leistungsfähigkeiten solcher Prozessoren wie des Z80 oder 6502. Die Lösung: Man verwendet 16- oder 32-Bit-Prozessoren.

Wege zum Wissen

Semantisches Netz
Verwandtschaften zwischen Gegenständen



Wissen kann als zusammenhängende Information definiert werden. Computer speichern zwar Daten, können aber die Verwandtschaft der Daten untereinander nicht erkennen. Es müssen also Methoden entwickelt werden, um die Zusammenhänge erkennen zu können. Eine ist das semantische Netz, eine andere das Raster. Produktionsanweisungen sind Listen von Informationen und logischen Operatoren. Keine dieser Methoden ist für die Darstellung des Wissens ausreichend. Deshalb sind Kombinationen üblich. Hier wird ein semantisches Netz zur Darstellung detaillierten Wissens benutzt. Wichtig dabei ist, daß es sich um rein statische Darstellungen handelt, die keine Aktionsanweisungen enthalten.

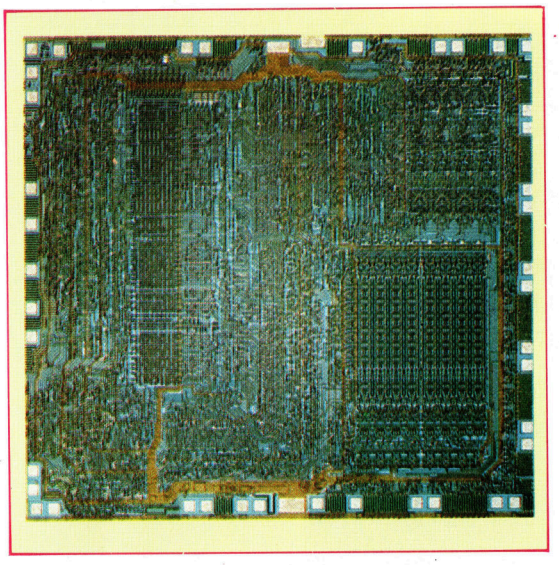
Raster

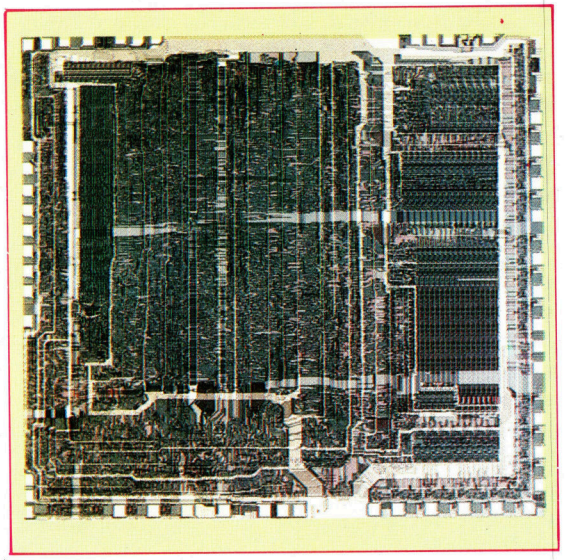
Geordnete Informationen

Rasterart:	1
Bezeichnung:	Stuhl
Kurzbeschreibung:	3 Beine Sitz
Material:	Holz Metall Plastik
Höhe:	0,5–1 m
Verwandtschaften:	Möbel mit Beinen und Sitzen

Produktionsanweisungen

Boolsche Regeln
IF Stuhl **THEN** (3 Beine **AND** Sitz) **AND** (Holz **OR** Metall **OR** Plastik)





Diese Mikroskopaufnahmen veranschaulichen die dichten Belegungsmöglichkeiten moderner Microelektronik: links eine stark vergrößerte Darstellung des Zilog-Renners Z80, rechts der neuere Z8000. Als 16-Bit-Prozessor hat der Z8000 eine höhere Packungsdichte als der Z80 mit seinen acht Bit, und außerdem ist an den Rändern des neuen 16-Bit-Chips eine wesentlich größere Anzahl von Kontaktierungsflächen vorgesehen.

Z-Prozessoren

Als die Firma Zilog 1977 den Microprozessor Z80 vorstellte, ahnten nur wenige, daß damit eine Revolution in Gang gesetzt würde. Aber innerhalb weniger Jahre haben der Z80 und sein Rivale 6502 geschafft, was kurz zuvor noch als futuristischer Höhenflug abgetan wurde – nämlich den Computer in jedes Haus zu bringen.

Zilog-Präsident Frank de Weeger

Die Zilog-Story beginnt in den frühen siebziger Jahren, als Frederico Faggin und Masatoshi Shima bei der Microelektronik-Firma Intel ausstiegen, um ein eigenes Unternehmen aufzubauen. Die beiden waren an der Entwicklung des Prozessorbausteins 8080A beteiligt, der als erster Einchip-Rechner gilt, und gingen mit dieser Erfahrung an den Entwurf eines neuen Microprozessors. Der 8080 war bei Profis und Amateuren schon sehr verbreitet, und deshalb lag es für Faggin und Shima nahe, den neuen Chip 8080-kompatibel auszulegen. Damit wurde auch das große Softwareangebot nutzbar, das für den 8080 bereits vorhanden war. Aufgrund ihrer Detailkenntnisse über den 8080 konnten sie den Befehlssatz durch Verwendung von zusätzlichen Registern, Zwei-Byte-Operationen und anderen Techniken erweitern. Der neue Z80 stellte daher für Hardwarefirmen und Computerbesitzer einen beträchtlichen Fortschritt dar.

Diese Hardware-Revolution wurde durch eine parallele Entwicklung auf dem Softwaregebiet noch unterstützt. Gary Kildall und John Torode schrieben 1972 ein Betriebssystem namens CP/M (Control Program for Microcomputers). Als Intel-Berater hatte Kildall seine CP/M-Version zwar auf den 8080 und 8085 ausgerichtet, es wurde aber bald das dominierende Floppy-Betriebssystem für Computer. Mit seiner leistungsfähigen 8080-kompatiblen Neuentwicklung lag Zilog genau richtig, um von dem CP/M-Boom zu profitieren.

Probleme durch 16-Bit-Prozessoren

In den folgenden Jahren ging es aber nicht so reibungslos weiter. Die Verkaufszahlen des Z80 sind zwar immer noch gewaltig – pro Monat stellt die Firma davon eine runde Million her –, aber Versuche, mit Weiterentwicklungen auf dem 16-Bit-Markt zu landen, stießen nur auf verhaltenes Interesse.

Als 16-Bit-Prozessor wurde zunächst der Z8000 eingeführt, der wegen seines reichhaltigen Befehlssatzes und seiner zahllosen Register allgemein als vielversprechend begrüßt wurde. Die Programmierung erwies sich jedoch als äußerst schwierig, und dazu kamen andere Probleme: Der Z8000 ist zwar mit dem noch unfertigen Z80000 (oder Z80K) kompatibel, aber nicht mit dem Z80. Daher kann er nicht von der umfassenden Software profitieren, die inzwischen für den Z80 auf dem Markt ist. Daraufhin tendierten Hersteller, die einen 16-Bit-Nachfolger für ihre Rechnerfertigung suchten, eher zu einem weniger anspruchsvollen Prozessor.

Da sich der Z8000 nicht durchsetzen konnte, machte Zilog einen neuen Anlauf. Das Ergebnis ist der 16-Bit-Prozessor Z800, der voll Z80-kompatibel ist und ab Jahresende lieferbar sein soll. Auch anderwärts zeigt sich Licht: Commodore stattet die neue 900er-Klasse mit dem Z8000 aus.

Arbeiten mit der Commodore-Floppy

Dir Firma Commodore kann auf jahrelange Erfahrung bei der Herstellung von 5 1/4-Zoll-Diskettenstationen zurückgreifen. Die Commodore-Laufwerke sind mit einem eigenen Prozessor und einem zusätzlichen RAM ausgestattet.

Intelligente Laufwerke sind teuer in der Herstellung. Trotzdem gelang es Commodore nach der Einführung des VC 20, eine preiswerte Version seiner PET-Floppy auf den Markt zu bringen: die 1540-Floppy. Das Gerät arbeitet sowohl mit dem VC 20 als auch mit dem C 64. Allerdings zeigte sich, daß jeder Datenzugriff mit POKE eingeleitet und beendet werden mußte. Mit diesem mühseligen Verfahren ist es glücklicherweise inzwischen vorbei: Commodore hat das Disketten-Betriebssystem (DOS = Disk Operating System) geändert. Die neue Version des Gerätes heißt 1541.

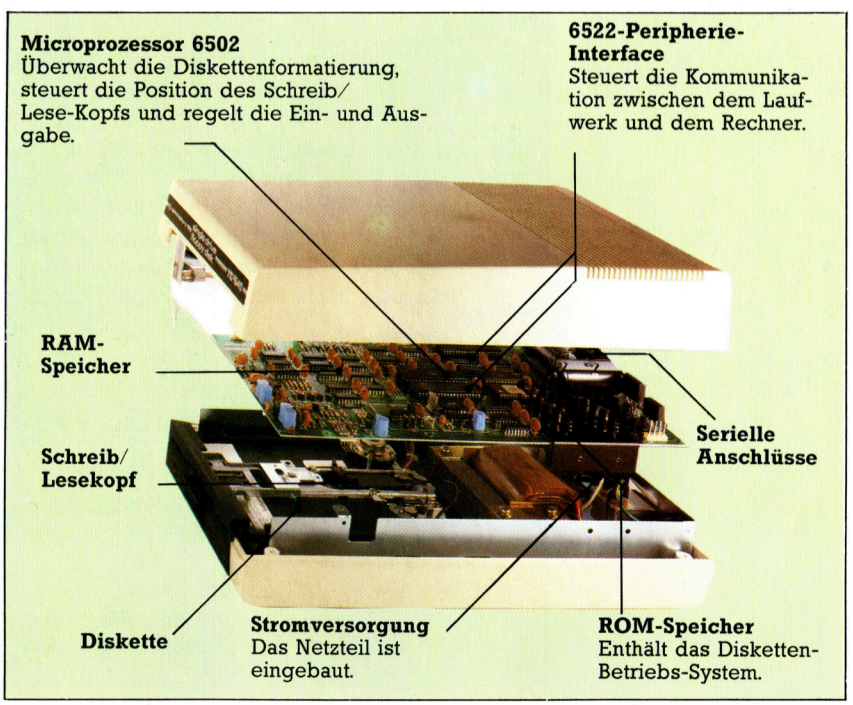
Die 1541 wird von einem 6502-Microprozessor gesteuert. Außerdem verfügt sie über zwei 6522-VIAs, zwei KiloByte RAM-Speicher und einen ROM-Speicher von acht KByte, der das Betriebssystem (DOS) enthält. Neben der Fehlerüberwachung bietet dieses umfangreiche Möglichkeiten zur Manipulation von Programmfiles (PRG), sequentiellen Datenfiles und „Random-Access"-Files. Eine serielle Version des IEEE-488-Interface schafft die Verbindung zum Rechner. Das Interface unterstützt dieselben Anweisungen wie die schnellere Commodore-Parallelschnittstelle, die zum Anschluß weiterer Peripheriegeräte dient. Geräte mit seriellem IEEE488-Eingang können direkt von der Floppy angesprochen werden.

Die Disketten werden einseitig mit 35 Spuren formatiert. Die äußeren Spuren sind in 21 Sektoren unterteilt, ganz innen liegen nur noch 17 Sektoren. Jeder Sektor enthält neben einem 256-Byte-Block noch die Kennung, Start- und Stopbits sowie Prüfsummen-Bits. Auf einer Diskette finden 683 Blöcke Platz. Der Anwender kann 664 davon frei belegen. Je nach Art der gespeicherten Files liegt die Diskettenkapazität somit bei ca. 170 KByte. Die Ablage von Daten auf der Diskette wird vom Betriebssystem durch Einrichtung eines Inhaltsverzeichnisses sowie einer sogenannten BAM (Block Availability Map = Verzeichnis verfügbarer Blöcke) geregelt. Die BAM liegt auf Spur 18, Sektor 0, und enthält auf 144 Bytes die Information, welche Blöcke frei und welche belegt sind. Das Inhaltsverzeichnis befindet sich ebenfalls auf Spur 18 im Sektor 1. Es besteht aus einer Liste von bis zu 144 Filenamen, die mit Zusatzinformationen über den Filetyp und die Länge der einzelnen Files (in Blöcken) versehen sind. Bei jedem Schreib- oder Löschvorgang werden BAM und Inhaltsverzeichnis automatisch aktualisiert.

Ein wichtiger Pluspunkt ist, daß die 1541 keinen Speicherplatz des Computers belegt – für einen zuverlässigen Massenspeicher ein nicht zu unterschätzender Vorteil. Das serielle Interface der 1541 macht sich allerdings durch eine vergleichsweise niedrige Geschwindigkeit der Floppy störend bemerkbar.

Commodore druckt im Handbuch der Floppy nicht einmal Angaben zur Datenübertragungsgeschwindigkeit oder Zugriffszeit ab. Zwar arbeitet die Floppy 50mal schneller als der Cassettenrecorder, hat damit aber trotzdem noch eine längere Zugriffszeit als jedes vergleichbare andere Floppy-Laufwerk. Wer auf große Speicherkapazität Wert legt, kann an den seriellen Port ein Parallel-Interface anschließen. Das steigert zwar die Arbeitsgeschwindigkeit nicht, macht aber die Palette der PET-Peripherie für den VC 20 bzw. C 64 zugänglich.

Die intelligente Commodore-Floppy läßt sich an den C 64 und an den VC 20 anschließen. Im tragbaren SX 64 ist sie bereits fest installiert. Das Gerät arbeitet unabhängig von der CPU des Rechners und belegt auch keinen Speicherplatz. Zur Laufwerkssteuerung dient der gleiche 6502-Prozessor in MOS-Technologie, der auch in den Computern selbst eingebaut ist.



Auf der Frontplatte der 1541-Floppy befinden sich zwei LEDs: Eine grüne für die Stromversorgung und eine rote, die den Betriebszustand anzeigt.
EIN = Liest von der bzw. schreibt auf die Diskette.
AUS = Bereitschaft
BLINKEN = Betriebssystem zeigt einen Fehler an.

Um die Fehlerursache herauszufinden, muß der Fehler-Kanal des Disketten-Betriebssystems gelesen werden. Das folgende Programm sorgt für den Ausdruck der Fehlermeldungen. Eine Liste einzelner Fehlercodes und ihrer Bedeutung ist im Handbuch abgedruckt.

```
10 REM***DISKETTEN-
   FEHLER-
   MELDUNGEN***
20 OPEN 15,8,15
30 INPUT# 15, EN, EM$,
   ET, ES
40 PRINT CHR$(147)
50 PRINT "FEHLER
   AUF"EN
60 PRINTEM$
70 PRINT"SPUR"ET
80 PRINT"SEKTOR"ES
90 CLOSE 15:END
```

Befehlssatz der Commodore-Floppy

Das Commodore-Diskettenbetriebssystem (DOS) verfügt über einen umfangreichen Befehlsvorrat. Die wichtigsten Befehle haben wir unten aufgelistet. Die 8 ist jeweils die Kennziffer des Diskettenlaufwerkes.

SAVE
Erzeugt Programm-Files (PRG), deren Namen bis zu 16 Zeichen lang sein dürfen. Inhalt: Programme oder sequentielle Daten.
Format:
SAVE"FILENAME",8

LOAD
In der Form
LOAD"FILENAME",8
lädt der Befehl das angegebene PRG-File in den Speicher des Rechners. Der Befehl
LOAD"FILENAME",8,1
lädt die Programme bzw. Daten genau zu den Speicherplätzen zurück, aus denen sie geSAVEd worden waren.
LOAD"$",8
lädt das Disketten-Inhaltsverzeichnis in den Speicher, wo es wie ein BASIC-Programm geLISTet werden kann. Dabei wird angezeigt:
Name der Diskette
Disk-Kennung (2 Zeichen)
Bis zu 144 File-Namen
Jeweiliger File-Typ (PGR bzw. SEQ)
Länge jedes Files in Blöcken
Anzahl der freien Blöcke

VERIFY
Vergleicht in der Form
VERIFY"FILENAME",8
das im Speicher befindliche File mit dem durch „FILENAME" spezifizierten File. Bei Abweichungen erfolgt eine Fehlermeldung. Dient als Test, ob korrekt geSAVEd wurde.

OPEN
Erzeugt einen Kanal für Datenübertragung, der mit einer „Logischen File Nummer" (LFN) von 1 bis 255 bezeichnet wird. Es können bis zu zehn Kanäle gleichzeitig offen sein. OPEN erzeugt zusätzlich ein „secondary address" (SA), mit der weitere Zugriffskriterien festgelegt werden. Für ein Laufwerk gibt es nur die SA 15, die den übergeordneten Befehlskanal anspricht.
OPEN hat die Form
OPEN LFN,8,SA

CLOSE
Hat das Format
CLOSE LFN
Es schließt das angegebene File. Files sollten bei Nichtbenutzung immer mit CLOSE geschlossen werden!

PRINT#, INPUT#, GET#
PRINT# ähnelt dem Befehl PRINT. Die Daten werden jedoch nicht auf den Bildschirm, sondern in das mit Open spezifizierte File übertragen. Es hat die Form:
PRINT#LFN,"DATA" oder
PRINT#LFN,A$,B$ . . .
INPUT# und GET# lesen entsprechend SEQ-Files. INPUT# liest String-Daten ein. Es arbeitet nur dann richtig, wenn die einzelnen Strings mit Komma oder Semikolon getrennt sind. Andernfalls werden sie als ein zusammenhängender String behandelt. GET# liest die Daten Bit für Bit ein, einschließlich Komma und Semikolon. Syntax-Beispiele:
INPUT#LFN,A$,B$ . . .
GET#LFN,A$,B$ . . .
Falls PRINT# in Verbindung mit einem OPEN des „Command"-Kanals (zum Beispiel OPENLFN,8,15) eingesetzt werden soll, lautet die Syntax
PRINT#LFN,8,15,"command string"
Dieser Befehl ist für den Umgang mit der Floppy besonders wichtig. Command- (Befehls) Strings dienen zur Steuerung der Floppy und machen zusätzliche Random-Access-Anweisungen zugänglich.

Disketten-Steuerungsbefehle

Hier eine Auflistung der Befehle, die mit PRINT# oder OPEN über den Command-Kanal eingesetzt werden können:

NEW
Formatiert und benennt die Diskette. Erstellt BAM und Inhaltsverzeichnis. Weist die zweistellige Disketten-Kennung (DI) zu.
Befehl: "N:DISKNAME,DI"

INITIALISE
Vergleicht das BAM auf der Diskette mit dem BAM im Speicher des Laufwerkes.
Befehl: "I"

VALIDATE
Löscht Datenblöcke, die mit REL-Befehlen verschoben wurden und nicht im Inhaltsverzeichnis vorhanden sind. Löscht auch Files, die nicht mit CLOSE geschlossen worden sind. Erstellt neues BAM.
Befehl: "V"

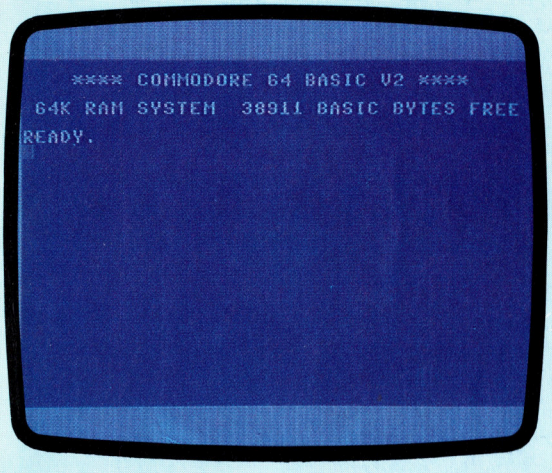
RENAME
Ändert den Namen eines Files im Inhaltsverzeichnis.
Befehl: "R:NEUNAME=ALTNAME"

SCRATCH
Löscht bezeichnetes File von der Diskette und aus dem Inhaltsverzeichnis.
Befehl:"S:FILENAME 1,FILENAME 2, . . ."

COPY
Stellt vom bezeichneten File eine Kopie auf derselben Diskette her.
Befehl: "C:NEUNAME+URNAME".
Der Befehl faßt auch mehrere SEQ-Files zu einem Gesamtfile zusammen.
Befehl:"C:GESAMTFILE=NAME1,NAME2, . . ."

Fachwörter von A bis Z

Cold Start = Kaltstart
Ob Ihr Rechner nun eine Sekunde oder tagelang abgeschaltet war – beim Einschalten gibt es jedesmal einen „Kaltstart". Dabei dauert es Sekundenbruchteile, bis alle Schaltkreise einen stabilen Betriebszustand erreicht haben. Über ein simples Zeitglied (Kondensator mit Ladewiderstand) wird etwa eine Zehntelsekunde nach dem Einschalten ein Impuls auf den Reset-Eingang des Prozessors gegeben, der sich in der Zwischenzeit stabilisiert hat.



So meldet der C 64, daß er betriebsbereit ist und genügend Bytes frei hat.

Der Reset-Impuls sorgt dafür, daß der Prozessor die „Startschwierigkeiten" überwindet und auf eine bestimmte ROM-Adresse springt, mit der das „Kaltstartprogramm" beginnt. Dieses durchläuft zunächst einen RAM-Funktionstest, fragt den verfügbaren Speicherraum ab und startet dann das Betriebssystem. Einen „Warmstart" veranlassen Sie dagegen durch Drücken der Reset-Taste. Dabei werden die Prozessor-Register zurückgesetzt, in manchen Fällen werden auch der Speicherinhalt gelöscht und das Betriebssystem aufgerufen.

Command Language = Kommandosprache
Sie kennen den Unterschied zwischen menü- und befehlsgesteuerten Programmen: Beim Menü wählen Sie durch einen Tastendruck aus einem Angebot von Varianten, während Sie bei der Befehlssteuerung ein Kommandowort eintippen müssen. Die Menütechnik zeichnet sich durch Anwenderfreundlichkeit aus. Bei Verfügbarkeit einer „Kommandosprache" läßt sich allerdings mit der Befehlssteuerung wesentlich mehr anfangen.

Eine Kommandosprache ermöglicht dem Benutzer die Kombination mehrerer Einzelaktionen in einem Befehlswort. Bei einer Datenbank brauchen Sie beispielsweise häufig Befehlsfolgen wie: GET = Hole den nächsten Datensatz, EXTRACT = entnimm das Feld TOTAL, MULTIPL(Y)iziere es mit dem DISCOUNT = Rabatt-Satz, UPDATE = aktualisiere den Datensatz mit dem neuen Wert und STORE = speichere ihn wieder in der Datei. Bei einer Kommandosprache reicht für diese Vorgänge ein Einzelbefehl, etwa MODIFY.

Hochentwickelte Datenbanksysteme wie dBaseII von Ashton-Tate gehen noch einen Schritt weiter: Dort können Sie ganze Programme mit Verzweigungen und Unterprogrammen in der Kommandosprache schreiben.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Bildnachweise
757, 758, 759: Andy Leslie
757, 779: Mike Brownlow
758, 763, 767, 778, 783: Ian McKinnell
759, 773, 780: Kevin Jones
760, 761, 769, 770, 771: Chris Stevens
763, 765, 786, 775, 776: Liz Dixon
774: Chris Stevens, Texas Instruments
779: Mike Brownlow

Comparator = Komparator
Ein Komparator ist ein Schaltkreis, der zwei Analogspannungen miteinander vergleicht und ein digitales Ausgangssignal abgibt.

Ein Komparator wird verwendet, um einen Analog/Digital-Wandler (AD) mit Hilfe eines Digital/Analog-Wandlers (DA) aufzubauen. Ein DA ist nämlich viel leichter realisierbar als ein AD. Der Ausgang des DA kommt an den einen Komparator-Eingang, die Meßspannung an den anderen. Über eine Steuerlogik mit Speicherregister wird an den DA eine binäre Zahlenfolge gelegt, die eine schrittweise Zunahme seiner Ausgangsspannung bewirkt. Erreicht sie die Höhe der Meßspannung, so „kippt" der Komparator „um" und hält mit seinem Ausgangssignal die Steuerlogik an – im Speicherregister steht dann der digitalisierte Meßwert zur Verfügung.

Compiler = Compiler
Ein Compiler ist ein Übersetzungsprogramm, das in einer höheren Programmiersprache geschriebene Programme oder „Quellprogramme" in den Maschinencode (in „Objektprogramme") umsetzt.

In einer Compilersprache entwickelte Programme sind wesentlich schneller als die im üblichen BASIC geschriebenen. Die meisten BASIC-Versionen arbeiten nämlich nur als „Interpreter", das heißt, sie übersetzen die Anweisungen während des Programmlaufs nacheinander in Maschinenbefehle. Zum Beispiel wird eine Schleife bei jedem Durchlauf neu vom BASIC in den Maschinencode übersetzt.

Ein Compiler entschlüsselt dagegen die Anweisungen nur einmal vor der ersten Programmbenutzung und speichert sie als Maschinenprogramm ab. Während des Programmlaufs fällt dann keine Übersetzungszeit mehr an. Für Heimcomputer werden verhältnismäßig selten Compiler angeboten, weil sie sehr viel Speicherraum beanspruchen und eine dialogorientierte Programmentwicklung dabei nur unter Vorschaltung eines Interpreters möglich ist.

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

computer kurs

Heft 29

Roboter-Funktionen werden auf dem Computer simuliert, bevor man z. B. eine Fertigungsstraße konstruiert.

Klein und doch leistungsstark

Der Apple IIc ist ein geschlossenes System ohne Erweiterungssteckleisten. So sind bereits alle notwendigen Schnittstellen und ein 128-KByte-Speicher im Gerät eingebaut.

Trigonometrie

Auch beim Programmieren im leichten BASIC kommt man oft um die Mathematik nicht herum. Wir erklären, wie man mit den Winkelfunktionen Sinus und Kosinus umgeht.

Wasser-Spiel

„River-Rescue“ ist ein spannendes Action-Spiel für Commodore, Atari und Sinclair.

Touchmaster

Für den Entwurf von Grafiken empfiehlt sich die Verwendung eines Grafik-Tabletts: Der Touchmaster paßt an viele Heimcomputer.

Schach auf Chips

Ein Vergleich der wichtigsten Schach-Programme.