Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Heft 71

Der PC-Stammvater PET

Textverarbeitung

Ordnung im Strom der Daten

computer kurs

Heft 71

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

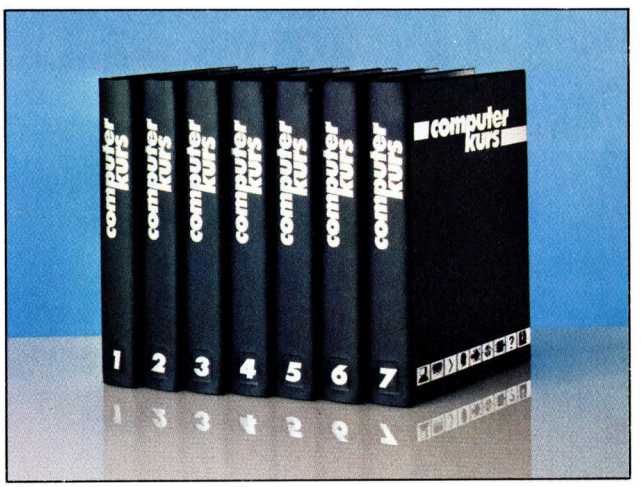
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

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

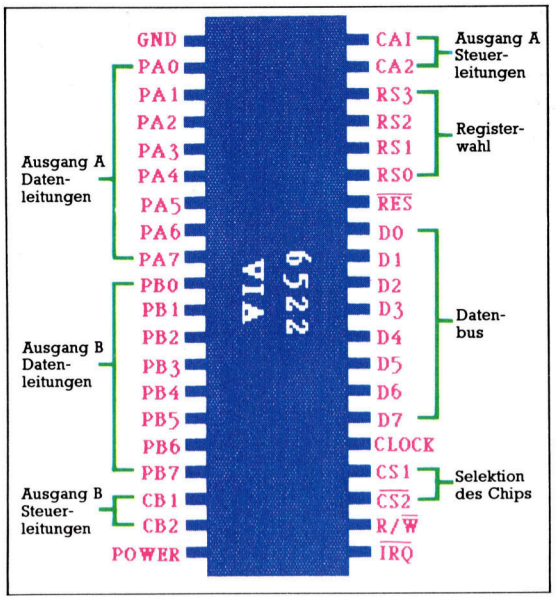
Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



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

Der Torhüter

Ohne spezielle Ein- und Ausgabeeinheiten läßt sich keine Kommunikation mit der Außenwelt herstellen. Wir sehen uns den PIO-Chip genauer an, der über die Ein- und Ausgabe wacht.



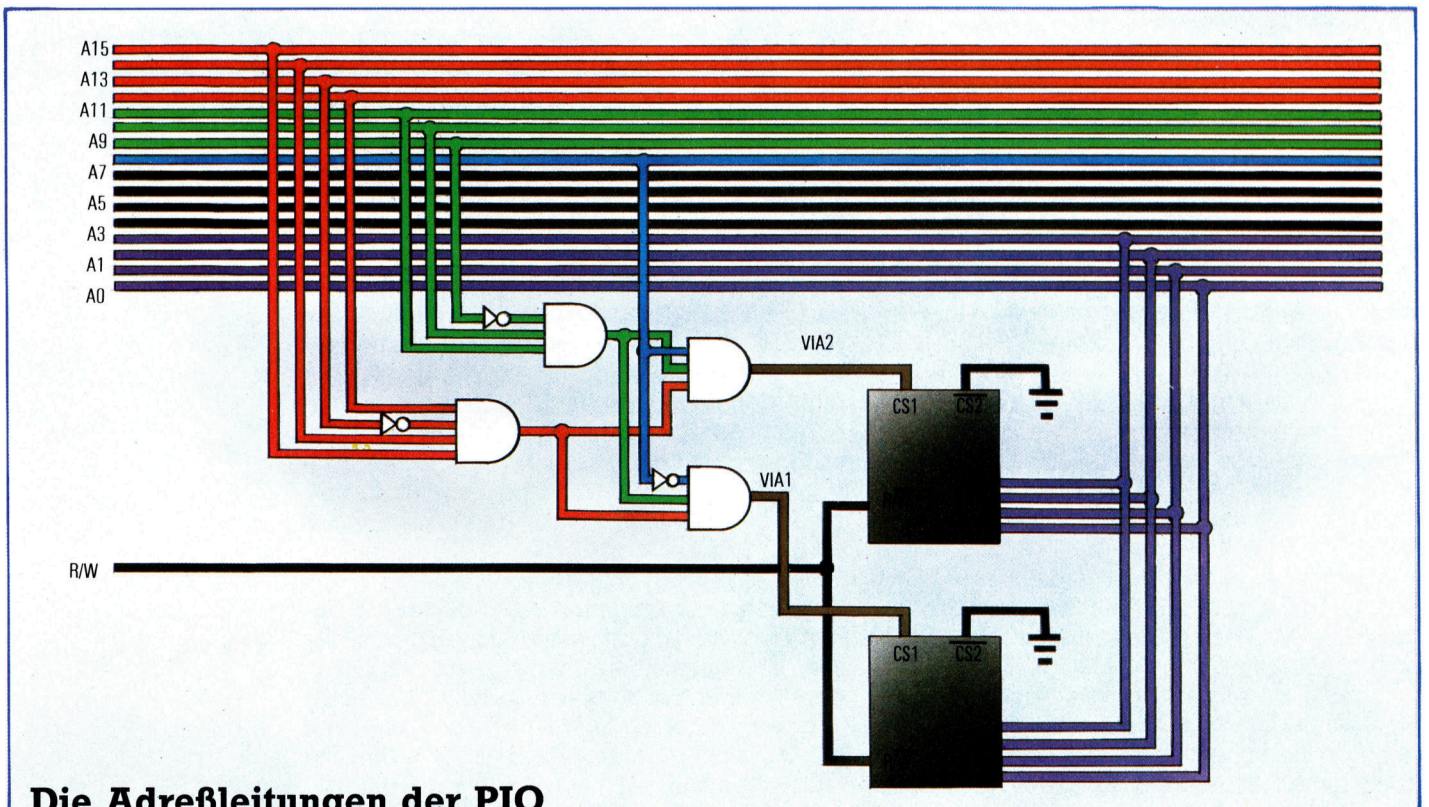
Der in den Acorn B eingebaute 6522 VIA-Chip ist ein typisches Beispiel für einen parallelen E/A-Chip – er besitzt zwei Acht-Bit-Ausgänge und Verbindungen zum Datenbus.

Wir haben bereits untersucht, wie ein Prozessor arbeitet und mit ROM und RAM Daten austauscht. Unser Beispielsystem war jedoch nach außen geschlossen und daher nicht ganz realistisch. Die Grundkombination Speicher/Prozessor kann auf vielerlei Weise mit anderen Geräten kommunizieren. Eingaben gehen dabei über Tastatur, Joystick und andere.

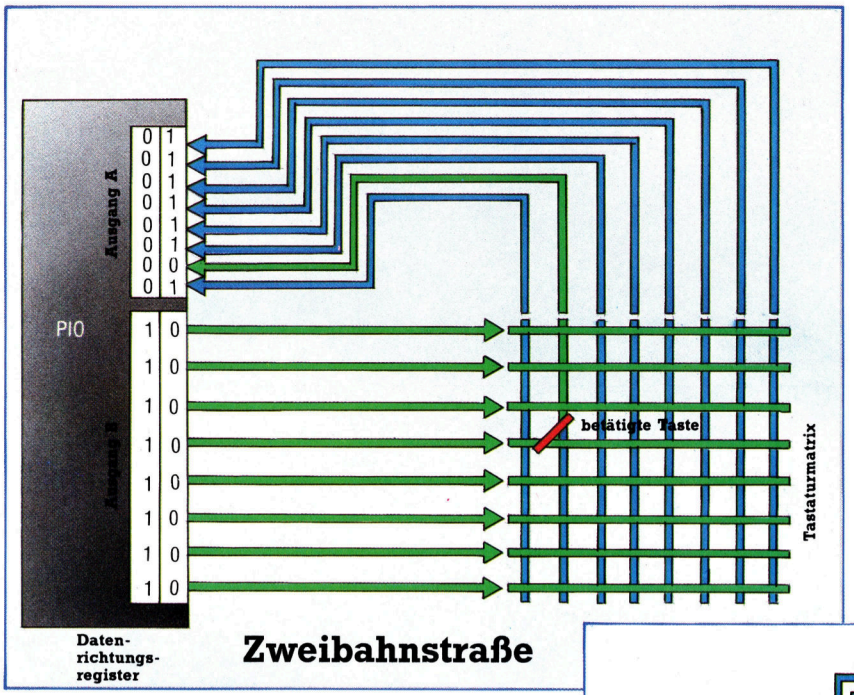
E/A-Geräte können nicht direkt an den Haupt-Datenbus angeschlossen werden, da die Daten dort nicht lange genug vorhanden sind, um von externen Geräten empfangen zu werden. Andererseits müssen eingehende Daten lange auf dem Datenbus bleiben, um vom Prozessor überhaupt verarbeitet werden zu können. Eine E/A-Schnittstelle muß Daten daher zumindest solange „festhalten", bis der E/A-Vorgang beendet ist, und ihn gleichzeitig synchronisieren (üblicherweise per Handshakeprozedur).

Die meisten E/A-Geräte bieten jedoch weit mehr als „Riegel" und Steuerleitungen. Wir müssen daher genauer auf die beiden E/A-Typen eingehen, mit denen die meisten Heimcomputer arbeiten. Der Schnittstellentyp wird hauptsächlich von der Methode bestimmt, mit der Peripheriegeräte und die Zentraleinheit Daten austauschen – seriell oder parallel. Es gibt daher auch zwei Gruppen von E/A-Geräten – PIOs und SIOs.

PIO-Chips werden über ihre internen Register programmiert. Um Zugang zu diesen Registern zu haben, muß der Prozessor sie in seinem Adreßbereich „finden" können. Unser Beispiel zeigt, wie zwei PIO-Chips in die Memory Map eingeblendet und über $DC00 und $DD00 angesprochen werden. Die oberen sieben Bits des Adreßbusses werden decodiert, die Leitung A8 wählt $DC00 oder $DD00 an.

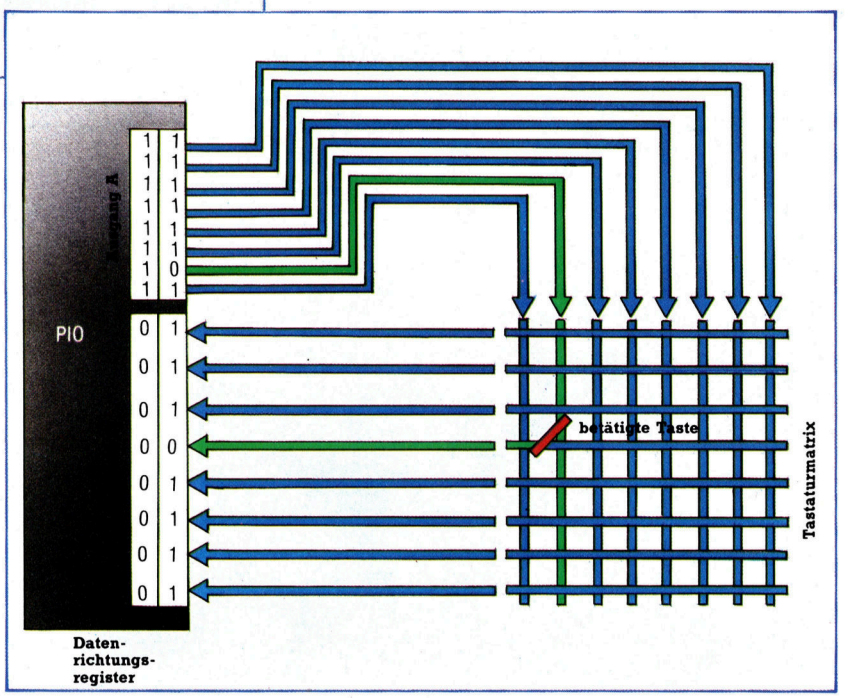


Die Adreßleitungen der PIO



Zweibahnstraße

Die Technik der Leitungsumkehrung ist eine von mehreren Möglichkeiten festzustellen, welche Taste einer Tastatur gedrückt wurde. Zunächst werden Nullen in die Matrixzeilen der Tastatur geladen. Ein Tastendruck setzt dann ein Bit dieses Registers auf Eins. Für die Spalten wiederholt sich der gleiche Vorgang in umgekehrter Richtung. Der so entstandene 16-Bit-Code kann das Zeichen der gedrückten Taste nun eindeutig identifizieren.



Die serielle Kommunikation wurde bereits ausführlich behandelt. Besonders die MIDI-Schnittstelle ist ein klassisches Beispiel für asynchrone Datenübertragung. Die Grundkomponenten einer derartigen Schnittstelle sind die Shift-Register, mit denen die parallel eingehenden Bits des Datenbusses in einen seriellen Datenstrom gewandelt werden und einkommende Daten paralleles Format erhalten. Auch testet die Schnittstelle die Prüfbits oder erzeugt sie.

Auf Heimcomputern ist der parallele PIO-Chip jedoch weit wichtiger. Der Name PIO wird nicht überall verwendet, VIA (Versatile Interface Adaptor), PIA (Peripheral Interface Adaptor) und CIA (Complex Interface Adaptor) bezeichnen alle die gleiche Einheit.

Heimcomputer mit voll ausgerüsteten Schnittstellen (etwa Acorn B und Commodore 64) besitzen zwei PIO-Chips. Einer davon steuert normalerweise die angeschlossenen E/A-Einheiten wie Tastatur und Joystick. Die zweite PIO läßt sich mit einem parallelen Drucker und einem User Port einsetzen und gibt Programmierern Zugang zum E/A-System.

Vielseitiger VIA-Chip

Unser erstes Bild zeigt den 6522 VIA-Chip des Acorn B. Mit seinen 40 Kontakten bietet er acht Verbindungen zum Datenbus und zwei Ausgänge im Acht-Bit-Format, die sich für Ein- und Ausgabe eignen. Die anderen Leitungen steuern Interrupts und Schreib/Lesevorgänge, registrieren Steuerbits und bedienen die Handshakeleitungen der E/A-Ausgänge.

Obwohl der PIO-Chip vom elektronischen Standpunkt aus gesehen extrem komplex ist, funktioniert er recht einfach. Der PIO-Chip eines typischen Acht-Bit-Micros hat zwei E/A-Ausgänge (A und B), bei denen sich die Datenrichtung der Bits mit Datenrichtungsregistern und separaten Handshakeleitungen individuell steuern läßt. Viele PIOs haben darüber hinaus noch Shift-Register, mit denen sie Daten über eine der Handshakeleitungen senden oder empfangen. Die Zeitsteuerung arbeitet dabei entweder mit einem der beiden internen Taktgeber oder mit den Impulsen eines externen Timers, die über eine andere Handshakeleitung hereinkommen.

Die einzelnen Bits eines Acht-Bit-Interrupt-Statusregisters zeigen an, ob von einer der vielen Quellen (Handshakesteuerleitungen, Timer, Shiftregister etc.) eine Interruptanforderung vorliegt. Bei programmierbaren Interrupts kann der Programmierer festlegen, welche der PIO-Interruptleitungen (falls nötig) die CPU unterbrechen soll.

Wie aber ist die PIO mit dem Grundsystem

Prozessor/Speicher verbunden? Die meisten Systeme mit E/A-Möglichkeiten verfügen über zwei Chips. Wenn die internen Register dieser beiden Chips bei $DC00 und $DD00 erscheinen sollen, müssen Daten an die PIO gesandt und von dort empfangen werden, das heißt, der Prozessor muß die internen PIO-Register wie Speicherstellen ansprechen und adressieren können. Damit jedoch PIO-Register im normalen Adreßbereich erscheinen, werden einige Adreßleitungen zur PIO geschaltet.

Über das Gatter

Die Abbildung der PIO-Adreßleitungen zeigt, wie die höchstwertigen acht Bits des Adreßbusses zu einer der beiden PIOs führen. Die oberen vier Bits werden über ein Gatter geleitet, das eine Eins erzeugt, wenn 1101 ($D) vorhanden ist. Die nächsten drei Bits lösen bei 110 eine Eins aus. Die Wahl zwischen $DC00 und $DD00 steuert das Adreßbit A8. Die untersten vier Bits sind direkt mit den Registerwahlkontakten der PIO verbunden. Auf diese Weise werden die internen Register der PIO 1 als Speicherstellen $DC00 bis $DC0F adressiert und die der PIO 2 von $DD00 bis $DD0F.

Eine PIO ist normalerweise für das System reserviert und stellt die Verbindung zu Tastatur und Joystick her. Es gibt mehrere Möglichkeiten, eine Tastatur mit 64 Tasten anzuschließen. Die einfachste arbeitet mit einer Leitungsmatrix zu je acht Zeilen und Spalten. Die Zeilenleitungen sind mit einem PIO-Ausgang verbunden, die Spaltenleitungen mit dem anderen. Ein Tastendruck verbindet die entsprechenden Zeilen- und Spaltenleitungen miteinander.

Die Technik der Leitungsumkehrung, mit der festgestellt wird, welche Taste gedrückt wurde, ist weitgehend softwaregesteuert. Sie arbeitet mit der Fähigkeit der PIO, die Ausgänge für Ein- und Ausgabe programmieren zu können. Dabei wird zuerst Ausgang B auf Ausgabe gesetzt und sein Datenregister mit Null geladen. Ausgang A ist auf Eingabe geschaltet, die Bits seines Datenregisters stehen auf Eins. Ein Tastendruck setzt nun das entsprechende Bit des Datenregisters A auf Null.

Der zweite Schritt kehrt die Datenrichtung der beiden Ausgänge um. Ausgang B ist nun bereit, Eingaben entgegenzunehmen. Die Bits seines Datenregisters stehen auf Eins – mit Ausnahme des Bits, das der Spaltenleitung der gedrückten Taste entspricht. Der jetzt in beiden Datenregistern gespeicherte 16-Bit-Code spricht die entsprechende Speicherstelle im ROM des Zeichensatzes an und ruft so den Acht-Bit-Code der gedrückten Taste ab.

Die beiden PIO-Chips können die Tastatur, einen parallelen Drucker und eine Anwenderschnittstelle mit dem Grundsystem verbinden. Im Bild unten ist zu sehen, daß die Tastatur einen PIO-Chip vollständig belegt. Einer der beiden Acht-Bit-Ausgänge der zweiten PIO dient als Druckerausgang, wobei ein Buffer zwischen Drucker und PIO inkorrekte Ausgaben abfängt. Der zweite Ausgang steht dann dem Anwender zur Verfügung.

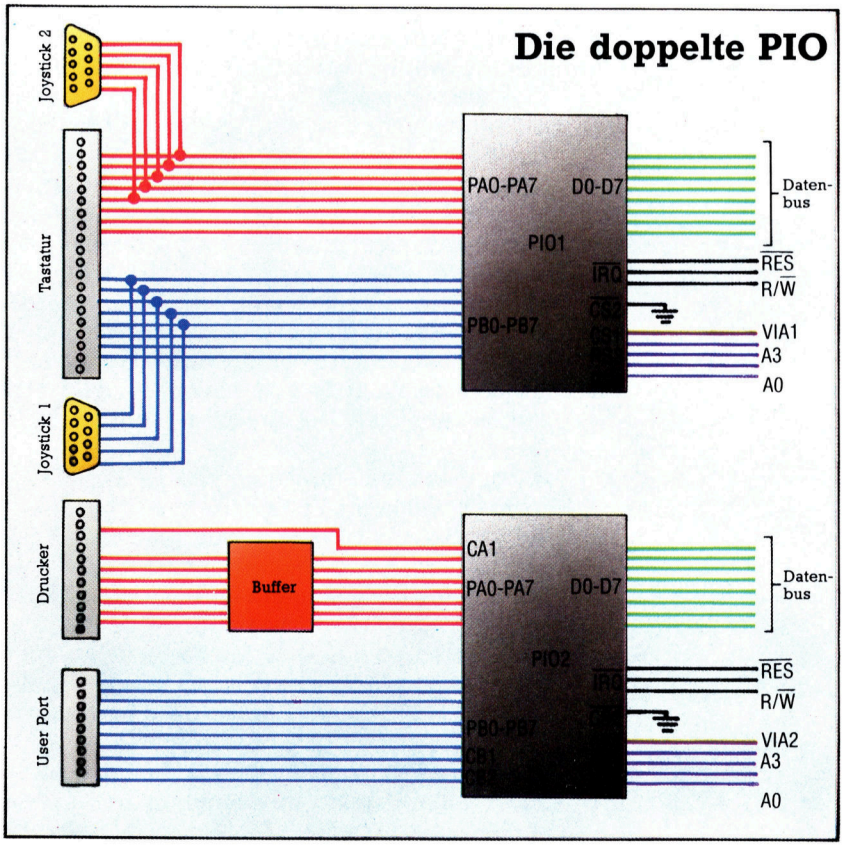
Das Bild zeigt, wie zwei PIO-Chips eine Schnittstelle bilden, die eine Tastatur, einen parallelen Drucker, Joysticks und den User Port mit dem Datenbus verbindet. Eine der Hauptaufgaben der Firmware oder Kernroutinen ist es, die PIO so zu programmieren, daß sie ein- und ausgehende Daten steuern kann. Für die meisten Anwendungen können Maschinencodeprogrammierer daher Kernroutinen verwenden, statt erst eigene Module entwickeln zu müssen. In einigen Fällen lohnt es sich jedoch, die PIO direkt anzusprechen – beispielsweise beim Einsatz des PIO-Timers oder des User Port.

Programmieren Sie die PIO

Der PIO-Chip besitzt mehrere interne Register, die der Hauptprozessor über das normale Adreßsystem ansprechen kann. Wenn diese Register mit bestimmten Werten geladen sind, werden in der PIO genau definierte Abläufe ausgelöst. Jede der acht Datenleitungen eines E/A-Ausgangs kann einzeln über das Datenrichtungsregister so programmiert werden, daß sie entweder als Eingangs- oder als Ausgangsleitung arbeitet. Auf diese Weise kann ein Ausgang mit einigen seiner Leitungen Eingaben entgegennehmen, während die anderen Leitungen Daten ausgeben.

Da E/A-Abläufe in fest definierten Zeitabschnitten ablaufen müssen, besitzt die PIO eigene Countdowntimer, die mit einem Wert geladen und bei jedem Taktsignal der Systemuhr dekrementiert werden. Das Auftreten eines „Underflow" (das heißt, der Zähler versucht, Null zu dekrementieren) löst ein Interruptsignal aus.

Synchrone Datenübertragung, bei der Daten in regelmäßigen Abständen empfangen oder gesendet werden, steuern meist die Timer der PIO.



Klassenbewußt

Wir untersuchen die vier „Speicherklassen" für Variablen der Programmiersprache C und initialisieren Arrays. In C haben Variablen einen exakt definierten Status, je nachdem, wie sie dem Speicher zugeordnet werden.

Außer einer Typenzuordnung haben alle C-Variablen auch eine Speicherklasse, die bestimmt, wie der Compiler sie verarbeitet und wie der Speicher für sie reserviert wird. Es gibt vier Klassen: automatic, extern, register und static. Die Klassenzuweisung geschieht ebenfalls mit diesen Schlüsselwörtern, die in der Variablendeklaration vor der Typenbezeichnung stehen:

```
extern double x, y;
```

Automatic ist die Standardklasse fast aller Variablen. Jede Variable, die innerhalb einer Funktion definiert wird, nimmt die Klasse automatic an und hat nur lokale Bedeutung. Bei jedem Aufruf der Funktion wird dieser Variablen dann neuer Speicherplatz zugewiesen, der beim Verlassen der Funktion wieder frei ist. Es ist nicht möglich, Variablen dieser Art zwischen zwei Aufrufen einer Funktion beizubehalten. Das gleiche trifft auch auf alle Variablen zu, die im Inneren eines Codeblocks definiert wurden und in Klammern {} eingeschlossen sind. Da main() ebenfalls eine Funktion ist, werden alle vorkommenden Variablen normalerweise als automatic angelegt.

Externe Variablen sind global und können von jedem Punkt des Programms aus angesprochen werden – unter bestimmten Umständen sogar von Funktionen, die in der Quelldatei der Deklaration nicht enthalten sind. Variablen, die außerhalb eines Funktionsmoduls deklariert werden, sind automatisch extern. Auch externe Variablen lassen sich an jeder beliebigen Stelle einer Funktion oder eines Blocks deklarieren. Sie bleiben nach dessen Ablauf erhalten und stehen nach ihrer Deklaration beliebig zu Verfügung.

Zwei externe Variablen, die in zwei oder mehr Quelldateien den gleichen Namen erhalten, werden beim Verbinden der Dateien als die gleiche Variable angesehen. Eine lokale Variable, die mit dem Namen einer globalen Variablen deklariert wird, „maskiert" diese im lokalen Bereich, so daß sich der Name im Inneren der Funktion (oder des Blocks) auf die lokale Variable bezieht und außerhalb auf die globale Variable.

Register-Variablen verhalten sich wie automatic-Variablen und werden in einigen Fällen auch ebenso behandelt. Die Zahl und Größe der für sie zur Verfügung stehenden Register mit schnellem Zugriff hängen vom Prozessor ab. Registervariablen sollten stets sparsam eingesetzt werden. Deklarieren Sie sie so spät wie möglich und geben Sie sie sofort frei, wenn sie nicht mehr benötigt werden. Registervariablen steuern oft Schleifen.

Static-Variablen sind normalerweise lokal, unterscheiden sich von lokalen Variablen aber dadurch, daß ihr Wert und Speicherplatz zwischen zwei Ausführungen einer Funktion (oder eines Blocks) erhalten bleibt. Statische Variablen können beispielsweise zählen, wie oft Funktionen aufgerufen werden. Auch können sie Daten „verstecken", da ihre Werte nicht wie externe Variablen von außen zugänglich sind. Eine static-Variable wird extern für eine Reihe von Funktionen deklariert und ist für diese Funktionen global verfügbar.

Es geht um Arrays

Arrays werden wie andere Variablen mit ihrer Größe (das heißt der Anzahl ihrer Elemente) deklariert. Die Größe steht hinter ihrem Namen in eckigen Klammern:

```
int intarray[100];
```

Damit wird Speicherplatz für die Arrayelemente intarray[0], intarray[1] etc. bis intarray[99] reserviert. Subscripts fangen immer bei Null an, wobei die Deklaration die Gesamtzahl der Elemente enthält. In diesem Fall gibt es daher kein intarray[100]. Static- oder external-Arrays werden in der Deklaration durch Hinzufügen einer in Klammern eingeschlossenen Werteliste initialisiert:

```
static int tage_im_monat[12] =
  {31,28,31,30,31,30,31,31,30,31,30,31};
```

Wenn die Liste nicht vollständig ist, werden die restlichen Elemente auf Null gesetzt. Ohne Initialisierung stehen alle Elemente eines statischen oder externen Arrays auf Null. Automatic-Arrays können nicht initialisiert werden. Da ihr Speicherplatz anfangs mit beliebigen Werten gefüllt ist, sollten Sie nicht davon ausgehen, daß ihre Elemente auf Null stehen.

Bei der Initialisierung von Arrays muß die Größe nicht angegeben werden, da C dafür au-

tomatisch die Zahl der angegebenen Werte nimmt. Die obenstehende Deklaration könnte auch so aussehen:

```
static int tage_im_monat[] =
  {31,28,31,30,31,30,31,31,30,31,30,31};
```

Besonders bei Strings und Arrays vom Typ char ist das sehr praktisch, da der initialisierte String nur in Anführungsstriche eingeschlossen wird. Beide Deklarationen sind gleichwertig.

```
static char st[] = "hallo";
```

ist gleichwertig mit

```
static char st[] = {'h','a','l','l','o'};
```

Beachten Sie jedoch, daß diese Strings nicht dynamisch sind. Die Länge des Strings kann nicht von der deklarierten Länge abweichen.

C kann Arrays mit fast jeder Zahl von Dimensionen verarbeiten. Bei Arrays mit zwei oder mehr Dimensionen wird jedes Subscript in ein eigenes Paar von eckigen Klammern eingeschlossen. Die Deklaration eines zweidimensionalen Arrays mit vier mal fünf Elementen sieht daher so aus:

```
int zweiarray[4][5];
```

Arrays lassen sich auch als Parameter an Funktionen übergeben, werden dann aber über eine Bezugsadresse angesprochen. Dabei wird die Adresse des ersten Elementes an die Funktion übergeben. Veränderungen, die von der Funktion im Array vorgenommen wurden, bleiben nach Verlassen der Funktion erhalten. Innerhalb einer Funktion muß die entsprechende Größe eines Arrays nicht deklariert werden, da sie bereits bekannt ist.

Unsere Beispielprogramme enthalten viele dieser Konzepte. Wir gehen davon aus, daß ein Zufallszahlengenerator und auch dessen Steuerfunktionen als Quelldateien vorhanden sind und beim Verknüpfen in die Programme eingebunden werden können. Das kurze Testprogramm ruft eine große Zahl von Zufallszahlen auf und prüft über die Zahlenhäufigkeit, ob eine gleichmäßige Verteilung besteht.

Listing 1, gespeichert in file 1

```
# define MULT 25173
# define MODULUS 65536
# define INCREMENT 13849
static int seed;

/* diese Deklaration geschieht extern, so dass die
Variable seed allen Funktionen dieser Datei
for (j = 0; j < NUM_OF_GROUPS; ++j)
{ printf ("%d—%d/n", groups[j]—
10, groups[j], frequencies [j]) }
zur Verfuegung steht, ausserhalb dieser Datei aber
nicht angesprochen werden kann. Der Name „seed"
steht daher in anderen Dateien wieder frei zur Ver-
fuegung */
randomise(s)

int s
{
   seed = s;
}
random (n)
int n;
/* ergibt eine Zufallszahl zwischen 0 and n */
{
   seed = (MULT * seed + INCREMENT) %
   MODULUS;
/* Die lineare Kongruenzmethode */
   return ((int) ((double) seed / (double)
   MODULUS *
   (double) n + 0.5));

/* Beachten Sie, wie die Ganzzahlenwerte mit Spezial -
formen in Fliesskommazahlen und wieder in Ganz-
   zahlen gewandelt werden. Die + 0,5 runden auf die
   naechsthoehere Ganzzahl */
}
double rnd()

/* ergibt eine Zufallszahl zwischen 0 and 1 */
{
   seed = (MULT * seed + INCREMENT) %
   MODULUS;
   return ((double) seed / (double) MODULUS);
}
```

Listing 2, gespeichert in file 2

```
# define SEED 17
# define SIZE 100
# define LIMIT 10000
# define NUM_OF_GROUPS 10
int groups [] = {10,20,30,40,50,60,70,80,90,100};

/* externe Array lassen sich initialisieren. Dieses Array
enthaelt die Gruppenbegrenzungen fuer die Zaehl-
frequenzen */

int frequencies [10];
/* dieses externe Array wird mit Nullen initialisiert */
main ()
int r;
{
      randomise (SEED);
      register int i;

/* Durch den Einsatz der Klasse register fuer die
Schleifenvariablen wird die Verarbeitung sehr schnell.
Deklarieren Sie die Variable so spaet wie moeglich */

      for (i = 0; i < LIMIT; ++i)
      {
/* 10000 Zufallszahlen zwischen 0 und 100 holen */
         r = random (SIZE);
         register int j;
/* testen, zu welcher Gruppe sie gehoeren */
         for (j = 0: j < NUM_OF_GROUPS;
            ++j)
            {
            if (r < groups[j])
{
/* Zufallszahl in die entsprechende Gruppe eintragen
und Schleife verlassen */
               ++ frequencies(j);
               break;
               }
            }
         }
/* eine Verteilungtabelle ausgeben, mit der die gleich-
maessige Verteilung gepruefft werden kann */
         for (j = 0; j < NUM__OF__GROUPS; ++j)
         { printf ("%d–%d=%d/n", groups[j]–
10, groups[j], frequencies [j]) )}
```

Kanalarbeiter

In unserer Serie über das Betriebssystem des Spectrum sehen wir uns die Steuerung der Kanäle an, über die Daten an den Bildschirm oder zum Drucker gesendet werden. Wir untersuchen dabei auch Systemvariablen, die in der Anzeige Veränderungen auslösen.

Strom	Kanal
0	K
1	K
2	S
3	P

Auf dem Sinclair Spectrum erfolgen Eingaben normalerweise per Tastatur, während Ausgaben hauptsächlich zum Bildschirm oder zum Drucker gehen. Im System des Spectrum wird jede dieser Hardwarekomponenten als Ein- oder Ausgabekanal bezeichnet. Daten, die in den Computer einfließen oder von dort kommen, heißen „Strom" (englisch: Stream). Datenströme lassen sich in unterschiedliche Hardwarekanäle lenken, wenn die angeschlossenen Geräte den Strom auf korrekte Weise verarbeiten können. Auf Ströme und Kanäle gehen wir jedoch erst in der nächsten Folge genauer ein.

In diesem Artikel untersuchen wir das E/A-System des Spectrum und sehen uns die Routine bei &0010 an, die ein Zeichen ausgibt. Jedes an diese Routine übergebene Zeichen wird je nach Kanalwahl an den Bildschirm oder den Drucker weitergeleitet. Unser Bild zeigt, wie auf einem nicht erweiterten Spectrum die Kanäle angelegt werden, und welche Nummern den entsprechenden Strömen zugeordnet sind. Die Kanäle werden mit einem einzelnen Buchstaben bezeichnet, die Ströme dagegen mit einer Nummer. Auf einem Standard-Spectrum sind nur die Ströme 0 bis 3 aktiv. Unsere kleine Tabelle am Rand zeigt, welche Kanäle welchen Stromnummern zugeordnet sind.

Um ein Zeichen an ein bestimmtes Gerät ausgeben zu können, müssen wir dem Spectrum zuerst mitteilen, welchen Strom er dafür aktivieren soll. Eine Ausgabe an den Bildschirm (Kanal S) geschieht über Strom 2. Mit einer ROM-Routine bei & 1601 informieren wir das OS, welchen Strom es einsetzen soll. Vor Aufruf der Routine wird die Stromnummer im Register A gespeichert. Nach Ablauf der Routine ist der Hardwarekanal dieser Stromnummer eröffnet. Der Kanal S wird mit folgenden Befehlen eröffnet:

```
LD    A,2
CALL &1601
```

Nach der Öffnung eines Kanals wird der Code des Zeichens, das dorthin gesendet werden soll, in das A-Register gestellt und mit RST die Routine bei &0010 aufgerufen.

In einigen Aspekten entspricht dies dem OSWRCH-Aufruf des Acorn B. Wir sprechen hier die ROM-Adresse jedoch direkt an und nicht über einen Vektor. Kanal S arbeitet in dem Bildschirmbereich, der normalen PRINT-Befehlen in BASIC zugänglich ist. Er kann aber auch die beiden unteren Bildschirmzeilen anspre-

Kanalisierter Spectrum

S – Oberer Anzeigebereich
K – Ausgabe im unteren Anzeigebereich (von BASIC für Eingabe- und Fehlermeldungen eingesetzt) und Tastatureingaben.
P – Drucker

Auf dem Spectrum werden Daten in „Strömen" (die in Kanälen „fließen") zu Peripheriegeräten übertragen oder von dort empfangen. Das Einschalten des Geräts ordnet die Kanäle K, S und P den Strömen (0 und 1), 2 und 3 zu. PRINT #2 (Strom 2 – Bildschirm) entspricht dem normalen Befehl PRINT. Beispiel: PRINT #2; „Diese Meldung erscheint auf dem Bildschirm".

chen, die der BASIC-Interpreter normalerweise für Fehlermeldungen und Prompts einsetzt. Unsere Tabelle zeigt, daß diese Zeilen ein Teil von Kanal K sind.

Zeichen, die in diesen Zeilen erscheinen sollen, müssen über Kanal K gesendet werden:

```
LD   A,0
CALL &1601
```

Wenn das A mit 3 geladen wird, eröffnen die gleichen Befehle den Druckerkanal.

Außer Schriftzeichen lassen sich auch Steuerzeichen über einen Kanal senden. Die Wirkung der Steuerzeichen hängt natürlich von dem gerade aktiven Kanal ab. Das bedeutet, daß wir auch die Maschinencodeentsprechungen von PRINT AT, PRINT INK, PRINT PAPER etc. abrufen können. Die folgende Tabelle zeigt die Funktionen einiger Steuerzeichen für den Kanal S oder K. Einige der angegebenen Zeichen haben auf dem Drucker (Kanal P) natürlich keine Wirkung.

Code	Parameter	Wirkung
8	–	Cursor ein Zeichen nach links
9	–	Cursor ein Zeichen nach rechts
10	–	Cursor eine Zeile nach unten
11	–	Cursor eine Zeile nach oben
12	–	Löschen
13	–	ENTER
16	n	INK n (braucht ein zus. Byte)
17	n	PAPER n (braucht ein zus. Byte)
18	n	FLASH n (braucht ein zus. Byte)
19	n	BRIGHT n (braucht ein zus. Byte)
20	n	INVERSE n (braucht zus. Byte)
21	n	OVER n (braucht zus. Byte)
22	nn	AT y,x (braucht zwei zus. Bytes für die X- und Y-Koordinaten)
23	n	TAB n (braucht zus. Parameter)

Einige Steuerzeichen brauchen als Parameter zusätzliche Bytes, die normalerweise dem entsprechenden BASIC-Befehl folgen. Für die Ausführung von PAPER 3 werden beispielsweise die Bytes 17 und 3 zum Kanal S gesendet. Der folgende Maschinencode entspricht PRINT AT 10,10; „A":

```
3E02     10    ld   a,2      ;open channel S by
CD0116   20    call #1601    ;selecting stream 2
3E16     30    ld   a,22     ;token for AT
D7       40    rst  #10      ;output a char routine
3E0A     50    ld   a,10     ;y co-ordinate
D7       60    rst  #10
3E0A     70    ld   a,10     ;x co-ordinate
D7       80    rst  #10
3E41     90    ld   a,65     ;ASCII code for 'A'
D7       100   rst  #10
C9       110   ret
```

Beachten Sie, daß in der Liste das Steuerzeichen für CLS fehlt. Das Löschen des Bildschirms geschieht über eine Routine bei &06DB. Vor dem Aufruf der Routine muß Kanal S eröffnet werden. Nach Ablauf der Routine muß der Kanal nochmals eröffnet werden, damit Daten auf dem Bildschirm erscheinen können. Die folgende Routine löscht den Bildschirm:

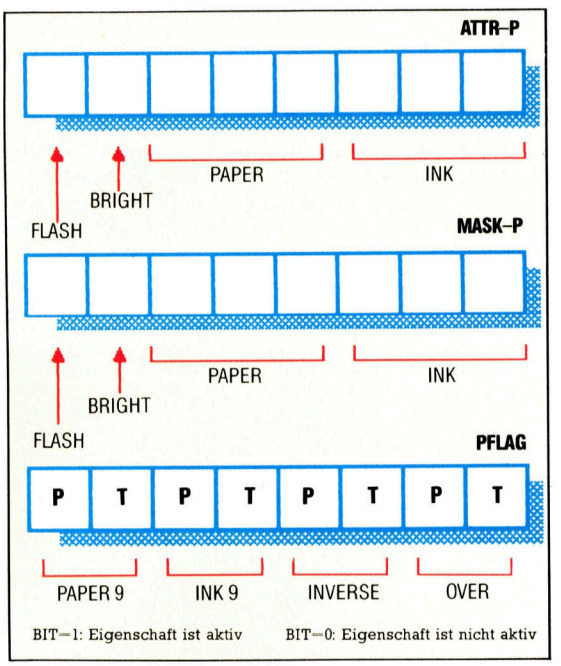
```
3E02     10    ld   a,2
CD0116   20    call #1601    ;open channel S
CDDB06   30    call #06db    ;clear the screen
3E02     40    ld   a,2
CD0116   50    call #1601    ;re-open channel S
C9       60    ret
```

Die Zeichenausgaberoutine hat die praktische Eigenschaft, daß dorthin übergebene Zahlen, die BASIC-Schlüsselwörter darstellen (Token), im Langtext dargestellt werden.

```
LD   A,249
RST &10
```

gibt daher auf Kanal S, K und P das Schlüsselwort RANDOMISE aus

Grafikbefehle wie PAPER, INK und BRIGHT, die mit der Zeichenoutputroutine ausgegeben werden, sind nur für die angegebene Folge von Ausgabezeichen aktiv – sie sind „temporäre Farbdaten". Wird der Befehl PAPER n außerhalb eines PRINT-Befehls ausgegeben, bleibt die entsprechende Farbe angeschaltet, bis ein weiterer PAPER-Befehl auftaucht.



Die beiden hierfür eingesetzten Systemvariablen heißen ATTR-P und MASK-P. Unsere Abbildung zeigt, wie sie verschiedene Aspekte der Anzeige steuern. Den drei Bits von ATTRP, die die Farbe von PAPER und INK festlegen, werden die in unserer Farbtabelle aufgeführten Werte zugeordnet. Wenn die Bits für FLASH oder BRIGHT auf Eins stehen, sind diese Eigenschaften aktiv. ATTR-P liegt bei der Adresse 23693.

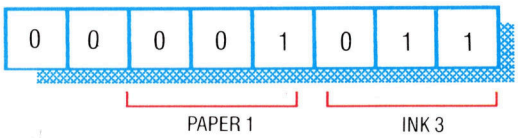
MASK-P befindet sich bei 23694. Jedes Bit (oder jede Bitgruppe) dieses Bytes, das auf Null steht, stellt sicher, daß der entsprechende Inhalt von ATTR-P die Bildschirmanzeige an dieser Position nicht verändert. Als praktisch erweist sich die Variable PFLAG bei 23697. Sie wird vom OS eingesetzt, um PAPER 9, INK 9,

Bits	Farbe
000	Schwarz
001	Blau
010	Rot
011	Magenta
100	Grün
101	Cyan
110	Gelb
111	Weiß

INVERSE und OVER anzuzeigen.

Zwei weitere Variablen – ATTR-T und MASK-T bei 23695 und 23696 – arbeiten ähnlich wie ATTR-P und MASK-P, steuern aber temporäre Farben (das heißt die Farbcodes, die mit PRINT-Befehlen oder Steuerzeichen über die Routine RTS &10 geschickt werden).

Der Einsatz von ATTR-P und MASK-P ist einfach. Für eine Farbänderung braucht man nur die entsprechenden Bits der Systemvariablen mit den Logikbefehlen des Maschinencodes AND und OR umzustellen. Für die Einstellung von PAPER 1:INK 3 sieht der Inhalt von ATTR-P so aus:



Geladen wird die Einstellung mit:

```
LD A,11
LD (23693),A
RET
```

Wie in BASIC haben die neuen PAPER- und INK-Befehle keine Auswirkung auf den bereits bestehenden Bildschirminhalt. Die neue PAPER-Farbe wird erst nach Ausführung eines CLS-Befehls aktiv.

Die Grafikroutinen des Spectrum sind Teil des BASIC-Interpreters. Beim Einsatz von PLOT und DRAW lassen sich durchaus auch Farbänderungen leicht mit den Variablen ATTR-P und MASK-P ausführen.

PLOT wird über Adresse &22E5 aufgerufen und erhält seine Koordinaten über das Registerpaar BC (B enthält die Y-Koordinate und C die X-Koordinate). Die Ausführung von PLOT 100,100 sieht im Maschinencode so aus:

```
LD B,100 ;Y-Koordinate
LD C,100 ;X-Koordinate
CALL &22E5 ;Ausführung
RET
```

Auch Farbänderungen lassen sich leicht durchführen. Die folgende Routine plottet einen roten Punkt auf den Bildschirm und versetzt ATTR-P vor seinem Rücksprung ins BASIC wieder in den ursprünglichen Zustand.

Kanal K arbeitet auf die gleiche Weise wie Kanal S. Alle Zeichen, die Sie in die unteren beiden Bildschirmzeilen schreiben, können jedoch von Meldungen des Betriebssystems oder des BASIC-Interpreters überschrieben werden.

Systemvariable LAST-K

Für Eingaben über diesen Kanal brauchen Sie nicht einmal eine ROM-Routine aufzurufen. Die Tastatur wird alle 20 Microsekunden abgefragt, wobei einige Systemvariablen anzeigen, ob eine Taste gedrückt wurde oder nicht. Die Systemvariable LAST-K (bei 23560) erhält den Zeichencode der zuletzt gedrückten Taste, während eine Systemvariable bei 23556 auf 255 steht, wenn keine Taste betätigt wurde. Die folgende Routine wartet auf einen Tastendruck (sie prüft, ob der Inhalt der Adresse 23556 nicht mehr auf 255 steht) und stellt den Code des eingegebenen Zeichens in das Register A. Nach dem Ablauf entspricht der Inhalt von LAST-K der zuletzt gedrückten Taste.

Weitere interessante Systemvariablen sind:

Variable	Adresse	Beschreibung
REPDEL	23561	Verzögerung der Tastenwiederholfunktion in Fünfzigstelsekunden. Legt Zeit bis zur Tastenwiederholung fest.
REPPER	23562	Verzögerung zwischen der zweiten und weiteren Wiederholungen – läßt sich mit POKE ansprechen.
PIP	23609	Länge des Tastaturklicks. Höhere Werte erzeugen ein „BEEP“.

Der Kanal P des Standard-Spectrum spricht den Drucker über Strom 3 an. Die Befehle

```
LD   A,3
CALL &1601
```

eröffnen Kanal P. Die darauf folgenden Zeichen werden auf dem Drucker ausgegeben. BASIC arbeitet oft mit Kanälen: PRINT und LIST sprechen Kanal S an, LPRINT und LLIST Kanal P und INPUT Kanal K. Man kann jedoch auch eigene Kanäle definieren, die dann auf Zusatzgeräte wie Microdrives, weitere Drucker und andere Hardware zugreifen. In der nächsten Folge gehen wir ausführlich darauf ein.

```
3A8D5C      10        ld   a,(23693)      ;get ATTR-P in A reg
F5          20        push af             ;and save on stack
E6F8        30        and  248            ;clear low 3 bits to 0
F602        40        or   2              ;set bit 1 to give red ink
0664        50        ld   b,100          ;y co-ordinate
0E64        60        ld   c,100          ;x co-ordinate
CDE522      70        call #22e5          ;call PLOT
F1          80        pop  af             ;restore original
328D5C      90        ld   (23693),a      ;ATTR-P contents
C9         100        ret
```

```
            10 ; GET-CHAR routine
3A045C      20 key:   ld   a,(23556)      ;check to see if
FEFF        30        cp   255            ;key is being pressed
28F9        40        jr   z,key          ;keep checking
3A085C      50        ld   a,(23560)      ;get LAST-K in A
C9          60        ret
```

High Fidelity

Nach der Einführung in die Hard- und Software der selbstgebauten MIDI-Schnittstelle entwickeln wir ein Programm, das aus dem Computer einen Recorder für digitale Aufzeichnung und Wiedergabe von Musikstücken macht.

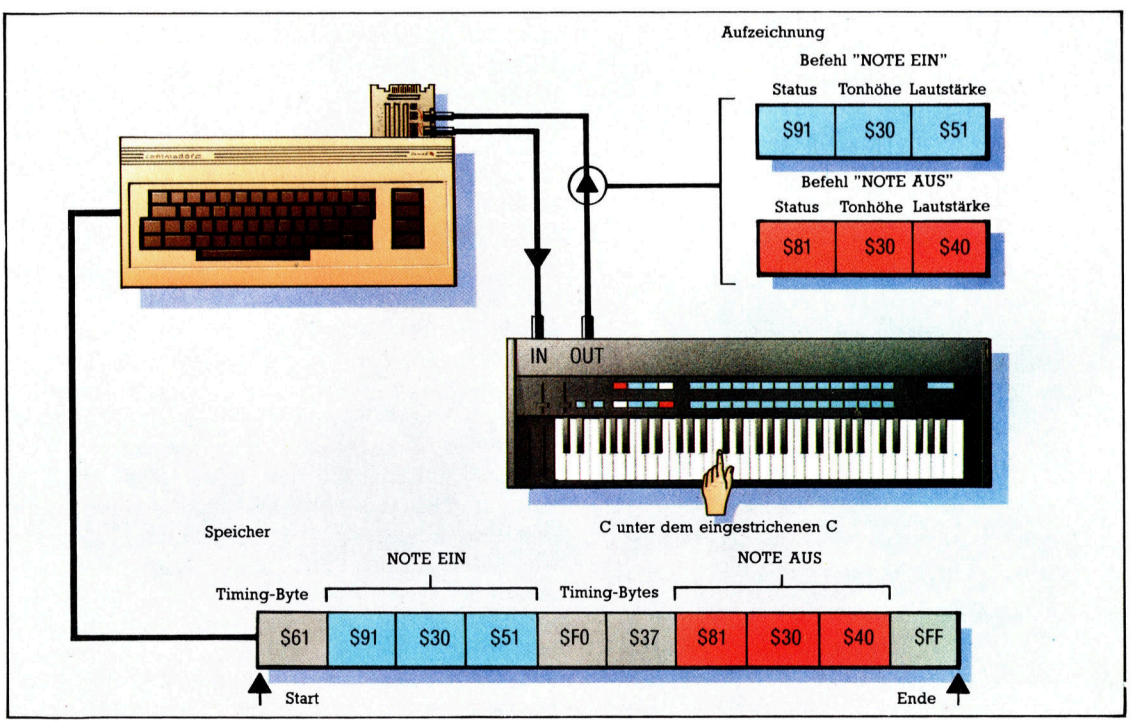
Beim Druck auf eine Taste oder ein anderes Bedienungselement eines MIDI-Instrumentes werden Daten in Byteform über die MIDI OUT-Buchse ausgegeben. Wir benötigen nun ein Programm zur Datenaufzeichnung und zum Bearbeiten empfangener Informationen. Auch der Computer selbst kann zum MIDI-Instrument werden, wenn er Daten an einen Synthesizer schickt, die von diesem in Töne umgesetzt werden. Der Rechner kann etwa als Sequenzer eingesetzt werden, in den man einzelne Abschnitte eines Musikstückes eingeben und über die Tastatur bearbeiten kann. Erst in der angestrebten Endfassung werden die Daten wieder über das MIDI-Interface an den ausführenden Synthesizer geschickt. Für MIDI ist das kein Problem – schließlich stehen 16 Kanäle zur Verfügung, über die mehrere Synthesizer oder Rhythmusgeräte gleichzeitig bedient werden können.

Zum Auftakt stellen wir ein Programm vor, mit dem der Computer (in Echtzeit) jedes auf einem MIDI-Instrument gespielte Stück speichern und – nach den entsprechenden Befehlen – wieder ausgeben kann. Die Aufzeichnung geschieht digital.

Digitale Aufzeichnung über ein MIDI-Interface unterscheidet sich grundlegend von einer Analogspeicherung auf Magnetband. Bei der analogen Bandaufzeichnung werden Töne in ein bestimmtes Orientierungsmuster der magnetischen Partikel auf der Tonbandoberfläche umgesetzt – beim Abspielen wird das Muster in Töne zurückverwandelt. Wenn Sie sich nun zwei Töne mit dazwischenliegender Pause von zwei Sekunden vorstellen, so würde das Band bei analoger Aufzeichnung auch während der Pause mit unveränderter Geschwindigkeit am Tonkopf vorbeilaufen – die Länge des leeren Bandabschnitts definiert in diesem Fall die Dauer der Pause.

Note per Tastendruck

Bei unserem digitalen MIDI-Aufzeichnungsverfahren liegt der Fall anders. Jede der beiden Noten entspricht einem Tastendruck, der zur Erzeugung eines oder mehrerer entsprechender MIDI-Befehle führt. Wenn Sie an obiges Analog-Beispiel denken, wird Ihnen schnell das Problem digitaler Echtzeit-Speicherung klar: Wir müssen in irgendeiner Form festhalten, wann der Computer die erste und wann er die zweite Note empfangen hat. Am



Das Bild zeigt, wie eine einzelne Note vom digitalen Aufzeichnungsprogramm gespeichert wird. Während der Aufzeichnung sendet der Synthesizer bei gedrückter Taste einen „Note-Ein"-Befehl, beim Loslassen wird "Note-Aus" geschickt. Diese Informationen werden im Speicher des Computers abgelegt. Auch die Anzahl der zwischen zwei Befehlen verstrichenen 2-Millisekunden-Intervalle wird festgehalten. In unserem Beispiel steckt diese Zeitinformation wegen der Pausenlänge in zwei Bytes. Auch für den Fall des Überlaufs (Pause länger als 239 ms) ist durch einen speziellen Bytewert vorgesorgt. Bei Abspielen werden die Verzögerungen entsprechend berücksichtigt.

Das Programm für den Commodore 64 ist einfach aufgebaut. In der Aufzeichnungsphase werden die über MIDI IN empfangenen Daten gespeichert. Die Aufzeichnung endet, wenn entweder der Speicher voll oder die Leertaste am Rechner gedrückt ist. Beim „Abspielen" werden die Daten über die MIDI OUT-Buchse wieder ausgegeben. Die Wiedergabe kann durch Druck auf die Leertaste gestoppt werden. Die erneute Aufzeichnung einer Tonfolge überschreibt vorher gespeicherte MIDI-Daten.

Programm zur digitalen Musikaufzeichnung

BASIC-Loader

```
10 FOR I = 49152 TO 49581
20 READ X:POKE I,X:S=S+X:NEXT
30 READ X:IF S=X THEN PRINT"OK SYS49152 TO RUN":END
40 PRINT"CHECKSUM ERROR"
50 END
100 DATA76,7,192,2,83,201,241,169,3
110 DATA141,0,222,169,21,141,0,222,169
120 DATA255,141,173,193,169,0,141,6
130 DATA220,169,8,141,7,220,169,17,141
140 DATA15,220,32,228,255,201,0,240
150 DATA249,201,69,208,1,96,201,82,240
160 DATA4,201,80,208,236,72,120,169
170 DATA127,141,0,220,169,173,133,247
180 DATA169,193,133,248,169,173,141,5
190 DATA192,169,241,141,6,192,160,0
200 DATA104,201,82,8,240,66,169,2,32
210 DATA69,193,32,44,193,201,255,208,4
220 DATA40,76,4,193,141,4,192,201,240
230 DATA173,4,192,240,12,32,246,192
240 DATA240,251,206,4,192,208,246,176
250 DATA223,32,44,193,72,173,0,222,41
260 DATA2,240,249,104,141,1,222,16,3
270 DATA32,53,193,202,208,233,174,3
280 DATA192,16,195,169,1,32,69,193,140
290 DATA4,192,162,255,32,246,192,240
300 DATA16,238,4,192,173,4,192,201,240
310 DATA144,6,32,21,193,140,4,192,173
320 DATA0,222,41,1,208,6,224,1,48,224
330 DATA16,243,173,1,222,48,11,224,0
340 DATA48,213,208,27,174,3,192,16,11
350 DATA201,248,176,223,201,240,176
360 DATA196,32,53,193,72,173,4,192,32
370 DATA21,193,140,4,192,104,32,21,193
380 DATA202,240,178,208,197,173,1,220
390 DATA73,239,208,18,104,104,40,208,4
400 DATA169,255,145,247,88,169,0,92,69
410 DATA193,76,37,192,173,13,220,41,2
420 DATA96,145,247,238,5,192,208,18
430 DATA238,6,192,208,13,104,104,104
440 DATA169,3,32,69,193,76,6,193,177
450 DATA247,230,247,208,2,230,248,96
460 DATA162,2,201,192,144,5,201,224
470 DATA176,1,202,142,3,192,232,96,10
480 DATA170,189,96,193,133,249,189,97
490 DATA193,133,250,160,0,177,249,240
500 DATA6,32,210,255,200,208,246,160,0
510 DATA96,104,193,134,193,146,193,158
520 DATA193,82,32,61,32,82,69,67,79,82
530 DATA68,44,80,32,61,32,80,76,65,89
540 DATA44,69,32,61,32,69,88,73,84,13
550 DATA0,147,82,69,67,79,82,68,73,78
560 DATA71,13,0,147,80,76,65,89,66,65
570 DATA67,75,32,13,0,79,85,84,32,79
580 DATA70,32,77,69,77,79,82,89,13,0
590 DATA240
600 DATA52178:REM*CHECKSUM*
```

Assembler-Listing

```
*=$C000
GETIN   = $FFE4          ;INPUT A CHARACTER
        CHROUT   = $FFD2 ;OUTPUT A CHARACTER
        JMP START
        NOBYTS *=*+1     ;NO. BYTES IN CURRENT MESSAGE
        CLOCKS  *=*+1    ;NO 2MS PERIODS SINCE LAST MESSA

        FREMEM *=*+2     ;- NO. OF FREE BYTES
        DATREG  = $DE01  ;TRANSMIT/RECEIVE REG ON ACIA
        STREG   = $DE00  ;STATUS/CONTROL REG FOR ACIA
        STOPKY = $EF
        MEM     = $F7    ;DATA MEMORY POINTER
        PTR     = $F9    ;STRING READ POINTER
        START   = *
;++++ SET UP ROUTINES ++++
        LDA #%03
        STA STREG        ;INITIALISE 6850
        LDA #%16
        STA STREG
        G10     LDA #$FF
```

einfachsten könnte man das entsprechend einer Bandaufzeichnung bewerkstelligen, indem man im Speicher eine bestimmte Anzahl „leerer" Bytewerte unterbringt. Das hieße aber reichlich verschwenderisch umgehen mit dem wertvollen Speicherplatz des Rechners.

Eine Alternative bietet die Speicherung zeitlicher Informationen in Byteform: Wenn 50 Zeiteinheiten nichts geschieht, läßt sich anstelle von fünfzig leeren Zellen auch die Zahl fünfzig festhalten, um damit die Länge der Pause zwischen den Noten zu definieren.

„NOTE EIN/AUS"

Die Speicherkapazität wird bei diesem Verfahren geschont, es ist allerdings sehr schwer vorherzusagen, wieviel Speicherplatz für die Aufzeichnung eines Musikstückes insgesamt gebraucht wird. Stücke mit schnellen Läufen belegen wegen der Vielzahl der „NOTE EIN/AUS" – Befehle viel mehr Bytes als langsame Stücke von gleicher Länge. Der Einsatz des „Pitch Wheels" belegt ebenfalls viel Speicherplatz, weil er eine Menge MIDI-Informationen beinhaltet.

Im hier abgedruckten Programm haben wir mit Hilfe der Variablen CLOCKS einen Zähler eingebaut, der die Intervalle zwischen den Befehlen festhält. Die Zählrate der Variablen bestimmt die Genauigkeit des Timings bei der Aufzeichnung – ein entscheidender Faktor für die Wiedergabequalität. Die optimale Rate wird willkürlich festgelegt – wir haben uns für etwa zwei Millisekunden Taktzeit entschieden. Das ist noch nicht so schnell, daß die Ausführungszeit der Unterprogramme unterschritten würde, andererseits ist aber auch die Exaktheit der Aufzeichnung ausreichend. Das festgelegte Zeitintervall wird als „Auflösung" bezeichnet, da es das kleinste Intervall der Aufzeichnung darstellt. Der erforderliche Taktimpuls wird vom CIA- oder VIA-IC des Rechners geliefert.

Die Timing-Bytes kommen

Die MIDI-Daten müssen für die Speicherung um die erforderlichen Timing-Bytes erweitert werden. Dazu wird jedem MIDI-Byte ein einzelnes Byte vorangestellt. Es gibt die Zahl der Intervalle an, die seit dem letzten übertragenen Befehl verstrichen sind. Dieses Einzelbyte kann bis zu 239 Einheiten festlegen und damit Werte zwischen $0 und $EF annehmen.

Wenn 240 Einheiten ohne Übertragung eines Befehls verstreichen, wird eine Überlaufmeldung gespeichert, die durch den Bytewert $F0 symbolisiert wird. Den Schluß einer Sequenz zeigt der Bytewert $FF an, die Werte $F1 bis $FE dienen in unserem Programm als interne Markierungszeichen.

Es folgt noch eine Version des digitalen Recorderprogramms für den Acorn B.

```
STA     LOWMEM          ;INIT TO EMPTY
        LDA #$00
        STA $DC06
        LDA #$08
        STA $DC07
        LDA #$11
        STA $DC0F       ;START TIMER B
        G20     JSR GETIN
        CMP #$00
        BEQ G20
        CMP #69         ;PRESS 'E' TO EXIT
        BNE G22
        RTS
G22     = *             ;NOT EXIT
        CMP #82
        BEQ G30
        CMP #80
        BNE G20
                        ;++++ REAL TIME PROG STARTS HERE
        G30     PHA     ;SAVE RECORD/PB COMMAND
        SEI
        LDA #$7F
        STA $DC00       ;SCAN LAST ROW OF KEYS
        LDA #<LOWMEM
        STA MEM
        LDA #>LOWMEM
        STA MEM+1
        LDA #<FRBYTS
        STA FREMEM
        LDA #>FRBYTS
STA     FREMEM+1
        LDY #$00
        PLA
        CMP #82
        PHP             ;SAVE Z FLAG = RECORD
        BEQ R00
                        ;
        P00 = *         ;PLAY ROUTINE
                        ;
        LDA #$02
        JSR STROUT
        P05 JSR READ
        CMP #$FF
        BNE P07         ;EXIT IF END REACHED
        PLP             ;ADJUST STACK
        JMP C20         ;EXIT PROCESS
        P07 STA CLOCKS
        CMP #$F0        ;CARRY SET = $F0 READ
        LDA CLOCKS
        BEQ P30         ;JMP IF NO TIME TO WAIT
        P10 JSR CHECK
        BEQ P10
        DEC CLOCKS
        BNE P10         ;JMP IF NOT TIMED OUT
        BCS P05         ;READ NEXT CLOCK IF $F0
                        ;
        P30 = *         ;READ NEXT MESSAGE BYTE
JSR     READ
PHA                     ;SAVE DATA BYTE
;
P35     = *             ;WAIT FOR TX REGISTER
LDA     STREG
AND     #$02
BEQ     P35             ;JMP IF STILL FULL
;
PLA                     ;GET DATA
STA     DATREG          ;SEND TO MIDI
BPL     P50             ;JMP IF NOT STATUS BYTE
JSR     GETNO           ;GET NO OF DATA BYTES
P50     DEX             ;DEC. BYTE COUNT
BNE     P30             ;GET NEXT DATA BYTE
LDX     NOBYTS
BPL     P05             ;GET CLOCK BYTE
;
R00     = *             ;RECORD ROUTINE
;
LDA     #$01
JSR     STROUT
STY     CLOCKS          ;CLOCKS = 0
R05     LDX #$FF        ;INIT TO NO DATA
R10     JSR CHECK
BEQ     R20
INC     CLOCKS
LDA     CLOCKS
CMP     #$F0
BCC     R20
JSR     STORE
STY     CLOCKS
;
R20     = *             ;CHECK FOR MIDI DATA
LDA     STREG
AND     #$01
BNE     R40             ;JMP IF DATA
CPX     #$01
BMI     R10             ;JMP IF WAITING
BPL     R20             ;JMP IF MID-MESSAGE
;
R40     = *             ;GET THE DATA
LDA     DATREG
BMI     R50             ;JMP IF STATUS BYTE
CPX     #$00
BMI     R10             ;JMP IF NOT CHANNEL DATA
BNE     R80             ;JMP IF MID-MESSAGE
LDX     NOBYTS          ;RESET BYTE COUNT
BPL     R60
;
R50     = *             ;STATUS BYTE PROCESS
CMP     #$F8
BCS     R20             ;IGNORE IF SYS REALTIME
CMP     #$F0
BCS     R05             ;JMP IF NOT CHANNEL STAT.
JSR     GETNO           ;GET NO OF DATA BYTES
;
```

```
R60     = *             ;RECORD THE DATA
PHA
LDA     CLOCKS
JSR     STORE
STY     CLOCKS          ;RESET COUNTER
PLA
;
R80     = *             ;RECORD MIDI DATA
JSR     STORE
DEX
BEQ     R10
BNE     R20
;
CHECK   = *
;
LDA     $DC01
EOR     #STOPKY         ;DOESN'T AFFECT CARRY
BNE     C40
PLA
PLA                     ;ADJUST STACK
PLP                     ;CHECK FOR REC/PB
BNE     C20             ;JMP IF PLAYBACK
LDA     #$FF
STA     (MEM),Y
C20     = *
CLI
LDA     #$00
JSR     STROUT
JMP     G20             ;WAIT FOR NEXT COMMAND
;
C40     = *             ;CHECK FOR TIMER B
LDA     $DC0D
AND     #$02            ;Z FLAG = NOT TIMED OUT
RTS
;
STORE   = *             ;STORE DATA IN MEMORY
;
STA     (MEM),Y
INC     FREMEM          ;CHECK MEMORY FREE
BNE     POINT
INC     FREMEM+1
BNE     POINT
PLA
PLA
PLA
LDA     #$03            ;ERROR MESSAGE
JSR     STROUT
JMP     C20             ;RESET PROGRAM
;
READ    = *             ;READ DATA FROM MEMORY
;
LDA     (MEM),Y
;
POINT   = *             ;UPDATE MEMORY POINTER
;
INC     MEM
BNE     P20
INC     MEM+1
P20     RTS
;
GETNO   = *             ;GET NO OF DATA BYTES
;           FOR STATUS IN ACC.
;
LDX     #$02            ;INIT TO 2 DATA BYTES
CMP     #$C0
BCC     N10
CMP     #$E0
BCS     N10
DEX                     ;1 DATA BYTE IF C0<=A<E0
N10     STX NOBYTS
INX                     ;ADD 1 FOR STATUS
RTS
;
STROUT  = *             ;PRINT STRING
;
ASL     A
TAX
LDA     MESTAB,X
STA     PTR
LDA     MESTAB+1,X
STA     PTR+1
LDY     #$00
M10     LDA (PTR),Y
BEQ     M70
JSR     CHROUT
INY
BNE     M10
M70     LDY #$00
RTS
;
;
MESTAB  = *             ;MESSAGE TABLE
;
.WORD   MESS0
.WORD   MESS1
.WORD   MESS2
.WORD   MESS3
;
MESS0   .BYTE 'R = RECORD,P = PLAY,E = EXIT',$0D,0

MESS1   .BYTE 147,'RECORDING',$0D,0


MESS2   .BYTE 147,'PLAYBACK',$0D,0


MESS3   .BYTE 'OUT OF MEMORY',$0D,0

LOWMEM  = *             ;START OF DATA MEMORY
FRBYTS  = LOWMEM - $D000
```

Grafikrahmen

Eins der größten Probleme der WIMP-Umgebungen ist die Übertragbarkeit der Programme auf andere Computer. Ein Grafikstandard für Kernroutinen bietet eine Lösung.

Die Erstellung einer WIMP-Umgebung für einen bestimmten Computertyp ist teuer. Überdies lassen sich diese speziell entwickelten Grafikanwendungen nicht auf andere Geräte übertragen. Übertragbarkeit würde bedeuten, daß das normale Betriebssystem mit einem „Grafikrahmen" arbeitet, der Fenster und Zeiger steuert und auch Programme ablaufen läßt. Nach Beendigung eines Vorgangs wird die Steuerung dabei nicht an das normale Betriebssystem zurückgegeben, sondern an den WIMP-Rahmen, der die Routinen des eigentlichen Betriebssystems (für Dateiverwaltung etc.) anspricht.

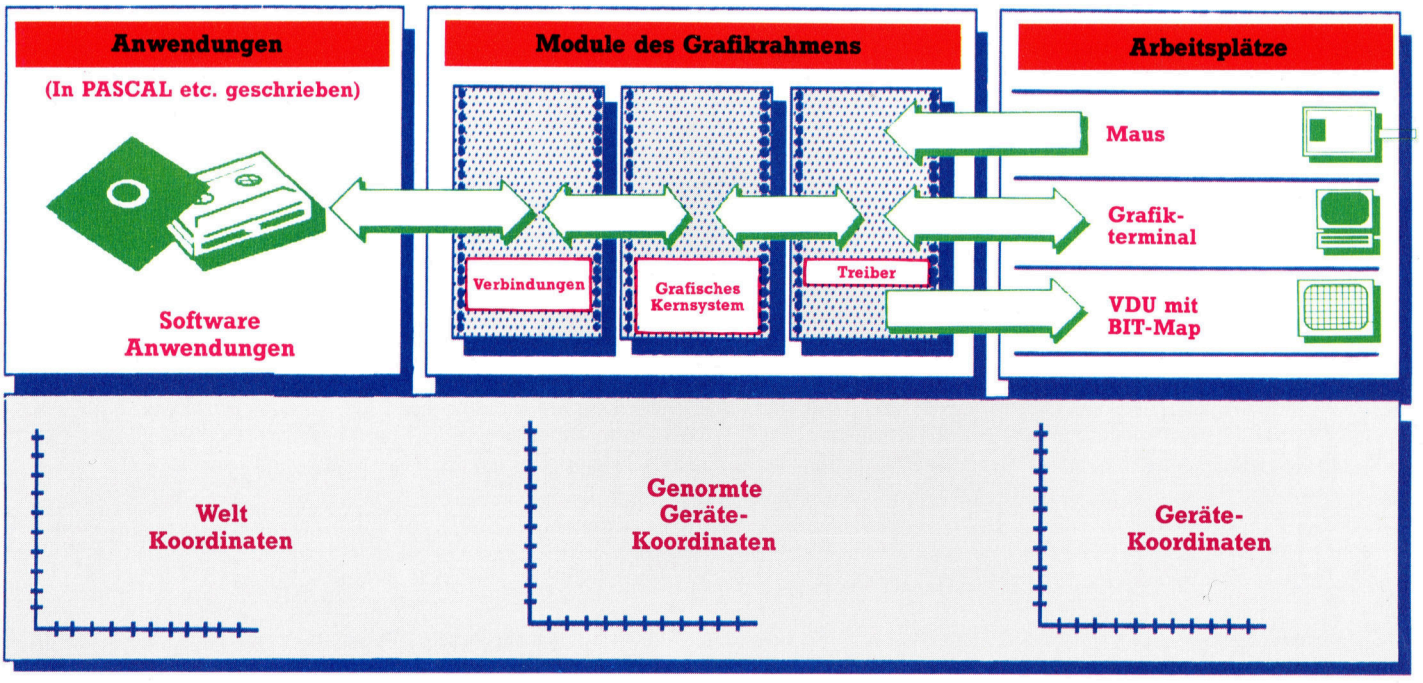
Die große Vielfalt der Grafiksoftware spricht zwar gegen eine Standardisierung dieser Umgebung, doch gibt es inzwischen Rahmenprogramme, die auf einem breiten Hardwarespektrum laufen, darunter TopView von IBM, MS-Window von Microsoft und GEM.

1977 wurden erstmals Bedingungen für ein „Grafisches Kern System" (GKS) veröffentlicht, das die Grafikprogrammierung systemunabhängig machen sollte. Inzwischen bieten mehrere Hersteller GKS an, doch blieb der Anwenderkreis wegen des hohen Preises auf CAD/CAM-Anwendungen und Universitäten beschränkt. Digital Research war die erste auf Microcomputer spezialisierte Firma, die für diesen Bereich das Rahmenprogramm GSX (Grafics System Extension) herausbrachte. GSX baut auf GKS auf, ist nicht kompatibel.

GSX ist eine Betriebssystemerweiterung der von DR entwickelten CP/M-86-Version und muß für jede Maschine angepaßt werden. Ist GSX erst einmal installiert, können Programme darauf zugreifen, ohne sich darum kümmern zu müssen, wie Hard-, Soft- oder Firmware diese Funktionen ausführen. GSX steuert die Hardware wie ein Betriebssystem, besitzt aber keine Anwenderschnittstelle (die mit viel Aufwand selbst programmiert werden muß).

Die britische Firma Prospero Software bietet eine Bibliothek von Grafikroutinen, die mit GSX arbeiten. Sie verzichten auf Skalen, Koordinatenmaps und Überläufe, die dem GSX-Programmierer das Leben schwer machen. Prospero liefert Compiler für ISO PASCAL und FORTRAN 77. Die GSX-Module werden in Programme dieser Sprachen eingebunden und laufen auf jeder Maschine mit installiertem GSX. Sie enthalten Treiber für Maus, Grafiktabletts und Tastatureingaben, aber auch für Drucker, Plotter, VDUs etc. Die GSX-Module plotten Punkte, Linien, Schattierungen und – sofern die Hardware vorhanden ist – auch mit unterschiedlichen Textskalen, Linienbreiten, Farben und Rotation.

GKS und andere Grafikschnittstellen werden übertragbar, indem sie die Anwenderkoordinaten in Gerätekoordinaten umsetzen, die die Hardware „verstehen" kann. In diesem Konzept sind nur die Gerätetreiber maschinenabhängig.



Doch selbst mit einer maschinenunabhängigen Grafikschnittstelle und der relativ bedienerfreundlichen Programmbibliothek von Prospero ist die Erstellung von Grafik immer noch sehr aufwendig. Für den User ist jedoch nur die Anwendung wichtig, ganz gleich, ob Grafik eingesetzt wird, die von GKS, GSX, Prospect, FORTRAN, PASCAL oder anderen Systemen gesteuert wird.

In einem kompletten WIMP-System bietet der Rahmen direkten Zugang zu allen Fähigkeiten des Computers, nicht nur zum Grafiksystem. Der Zugang sollte weiterhin nicht nur über Spezialmodule oder Betriebssystemroutinen möglich sein. Mit GEM und MS-Windows haben Digital Research und Microsoft diesen Weg eingeschlagen. Außer IBM arbeiten auch Apricot und RML (Nimbus) mit GEM und MS-Windows.

Das grafische Kernsystem, auf dem GEM aufgebaut, enstand aus dem Bedarf für übertragbare Grafiksoftware. Mit GSX hatte Digital Research eine kleinere Version von GKS geschaffen, die zwar dem Standard nicht entsprach, dafür aber billig war. Inzwischen wurde das Konzept von DR zum „Graphics Environment Manager" (GEM) weiterentwickelt. GEM ist daher das Ergebnis zweier Entwicklungslinien – zum einen des GKS-Standards und zum anderen des auf SMALLTALK aufgebauten „Schreibtischsystems", das durch den Macintosh von Apple bekannt wurde.

Arbeitsplatz mit Mäusen

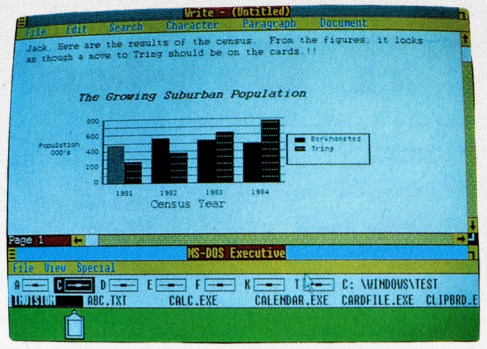
Ebenso wie die Anwendersoftware mit dem Betriebssystem CP/M unterschiedliche Hardware steuert, bietet GEM den Rahmen für die Maschinensteuerung eines WIMP-Systems. Eins der Grundkonzepte ist dabei der „Arbeitsplatz", der den Bereich von der Maus bis zum vollständigen Grafikterminal umfaßt.

GEM spricht jede Systemkomponente auf die gleiche Weise an, die Hardwareeigenschaften und Einschränkungen wurden in den Gerätetreibern der einzelnen Arbeitsplätze versteckt. So kann beispielsweise jedes Ausgabegerät – ob Plotter oder VDU – angewiesen werden, eine bestimmte Form zu zeichnen. Da die Hardwareeigenschaften jedoch Grafikfähigkeit und Auflösung stark beeinflussen, müssen die Grafikanwendungen bei geräteunabhängiger Steuerung radikal von der Hardware isoliert werden..

Jedes Gerät hat seinen eigenen Satz „Gerätekoordinaten". Farbmonitore arbeiten beispielsweise je nach Grad der Bildschirmauflösung auf den X- und Y-Achsen mit unterschiedlichen Maßstäben. Die Geräteeigenschaften und Koordinaten sind jedoch die einzigen Parameter, die sich in einem System, das mit GEM arbeitet, verändern.

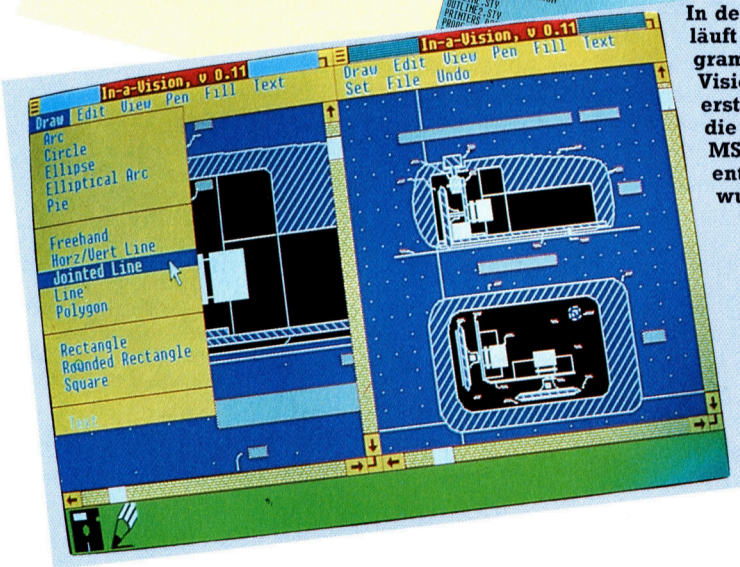
Gerätekoordinaten werden von einem Satz „normaler" Koordinaten abgeleitet, die GKS für alle Grafikvorgänge einsetzt. Diese genormten Gerätekoordinaten (NDC – Normalised Device Coordinates) werden wiederum in die Koordinaten der Anwendungssoftware – die sogenannten „World Coordinates" (WCs) – übernommen, die der Anwendungsprogrammierer auf jede beliebige Skala festlegen kann. Die NCD-Ebene dieses dreifachen Systems wird auf eine Skala von reellen Zahlen im Bereich von Null bis Eins übertragen. Der Auflösungsgrad findet nun nur noch in der Arithmetik der reellen Zahlen eine Einschränkung.

Die dabei entstehenden Schwierigkeiten werden von Computern der Groß-EDV durch hohe Geschwindigkeit und große Kapazitäten ausgeglichen. Auf Microcomputern ist das jedoch nicht so einfach. Da die Arithmetik der reellen Zahlen im Vergleich zur Ganzzahlenverarbeitung viel Zeit braucht, weicht GEM in dieser Hinsicht von GSK ab. Der NDC verfügt in GEM über eine Skala von Null bis 32767, so daß jeder Wert in einem 16-Bit Maschinenwort untergebracht und verarbeitet werden kann und Rechenzeit für Umwandlungen auf einem Minimum bleibt.



Farben können bei der Bearbeitung von Bildern und der Betonung von Meldungen viel ausmachen. Bei der Übernahme eines von einem Grafikprogramm entwickelten Diagramms in eine Textverarbeitung werden sie als Grauwerte dargestellt.

Eine Grafikumgebung unterstützt auch alle normalen Systemzugriffe. Das Bild zeigt das DIRectory des augenblicklich aktiven Laufwerks.



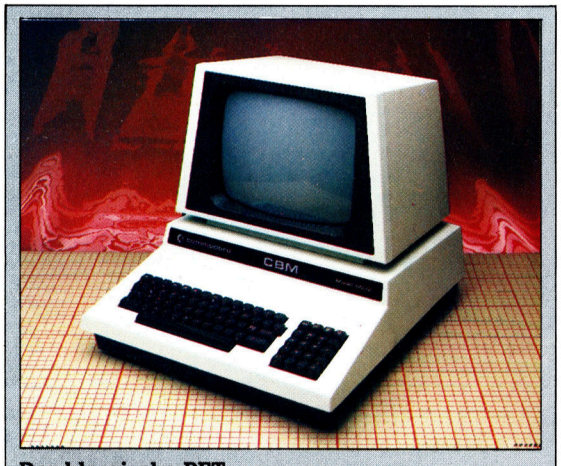
In den Fenstern läuft das Programm „In-a-Vision", die erste Software, die speziell für MS-Windows entwickelt wurde.

MS-Window ist eine von Microsoft entwikkelte Grafikumgebung, die speziell auf die Kompatibilität mit anderen Microsoftanwendungen ausgerichtet wurde. Die Bildschirminhalte zeigen den Einsatz des Systems. Im Gegensatz zum Macintosh von Apple arbeiten GEM und MS-Window mit Farbe.

In Ehren ergraut

Laut Lexikon ist ein „Commodore“ bei der Marine so etwas wie ein bewährtes altes Schlachtroß. Bis auf die Seefestigkeit kennzeichnet das durchaus treffend den ehrwürdigen Commodore PET, den Stammvater aller Personal-Computer.

In vieler Hinsicht gab der PET (Personal Electronic Transactor) von Commodore tatsächlich den Anstoß für den Microcomputer-Boom. Mit seinem Erscheinen 1977 wurde ein derart hoher Standard gesetzt, daß manche neuere Rechner vergleichsweise sogar als Rückschritt anzusehen sind. Schon äußerlich bietet der Ur-PET dafür ein schönes Beispiel: So ein robustes Metallgehäuse findet man heute meist nur noch bei teuren Bürorechnern. Ein anderer



Der klassische PET
Bei den ersten PETs war die Tastatur unzulänglich, während die nachfolgenden Versionen mit Schreibmaschinentasten mit Grafik-Symbolen (außer bei den Büromodellen) ausgestattet waren. Ein Schwarz-weiß-Monitor, später ein grüner 30-cm-Bildschirm, waren integriert.

Punkt ist das Netzteil – beim PET selbstverständlich eingebaut, bei vielen Heimcomputern dann lästigerweise ausgelagert.

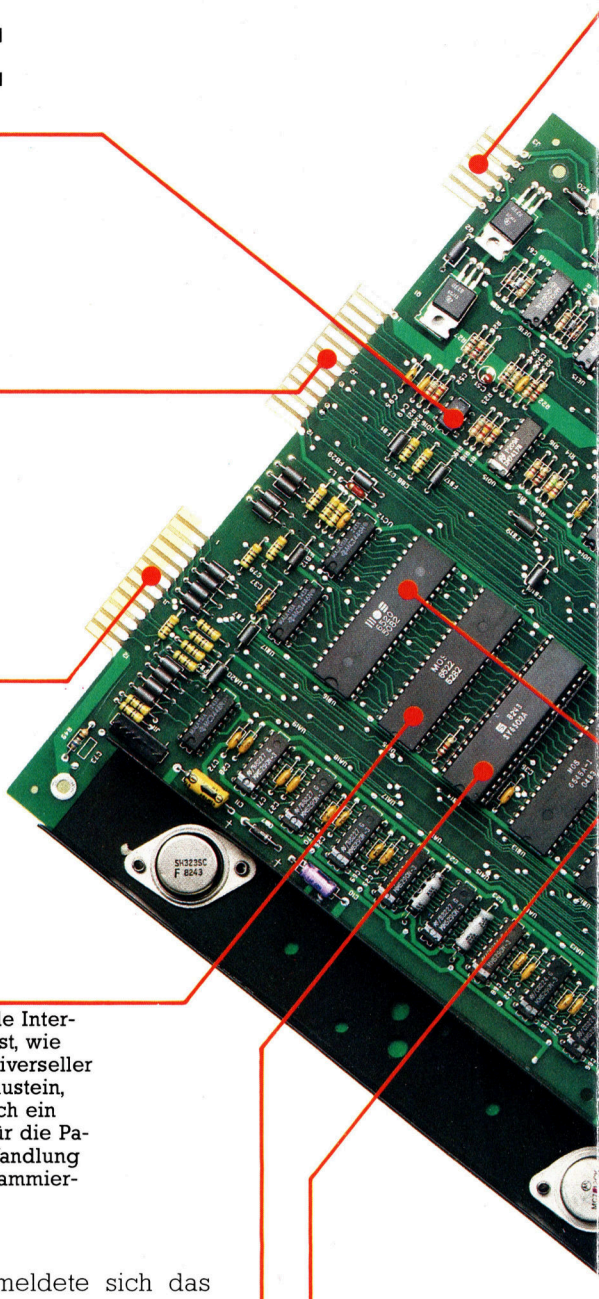
Zwar gab es mindestens zwei Jahre vor dem PET schon einige andere Acht- und auch 16-Bit-Rechner, aber dabei handelte es sich entweder um Bausätze oder um „Minimalsysteme“, die nur aus einer Leiterplatte mit einer Anzahl Chips bestanden. Der PET war der erste Computer, von dem man wirklich sagen konnte: „Stecker rein und fertig.“ Die ersten Versionen verfügten über einen eingebauten Cassettenrecorder mit Motorsteuerung, einen integrierten Monitor und ROM-gespeichertes BASIC – der Benutzer brauchte das Gerät nur einzuschalten, und schon meldete sich das System betriebsbereit:

```
COMMODORE BASIC VER. 1.0
7167 BYTES FREE
READY
```

Daraufhin konnte sofort ein Programm eingegeben und auf Cassette gesichert werden, ohne vorher diverse Komponenten zusammenstöpseln und das Betriebssystem vom Band laden oder es gar im Hex-Code byteweise eintippen zu müssen, was damals durchaus nicht unüblich war.

Das Commodore-BASIC wurde im Laufe der Jahre mehrfach überarbeitet, wobei die letzte Version dann gegenüber der Originalfassung so viele Erweiterungen enthielt, daß sie praktisch einen neuen Dialekt darstellte.

Eine andere wesentliche Besonderheit des PET war der Zeichensatz, der sowohl sämtliche ASCII-Codes wie auch eine Vielfalt von Block-



Timer
Dieser Zeitgeber-Chip gibt beim Einschalten einen Reset-Impuls ab, der die CPU in einen definierten Ausgangszustand versetzt und das Betriebssystem mit dem BASIC-Interpreter startet.

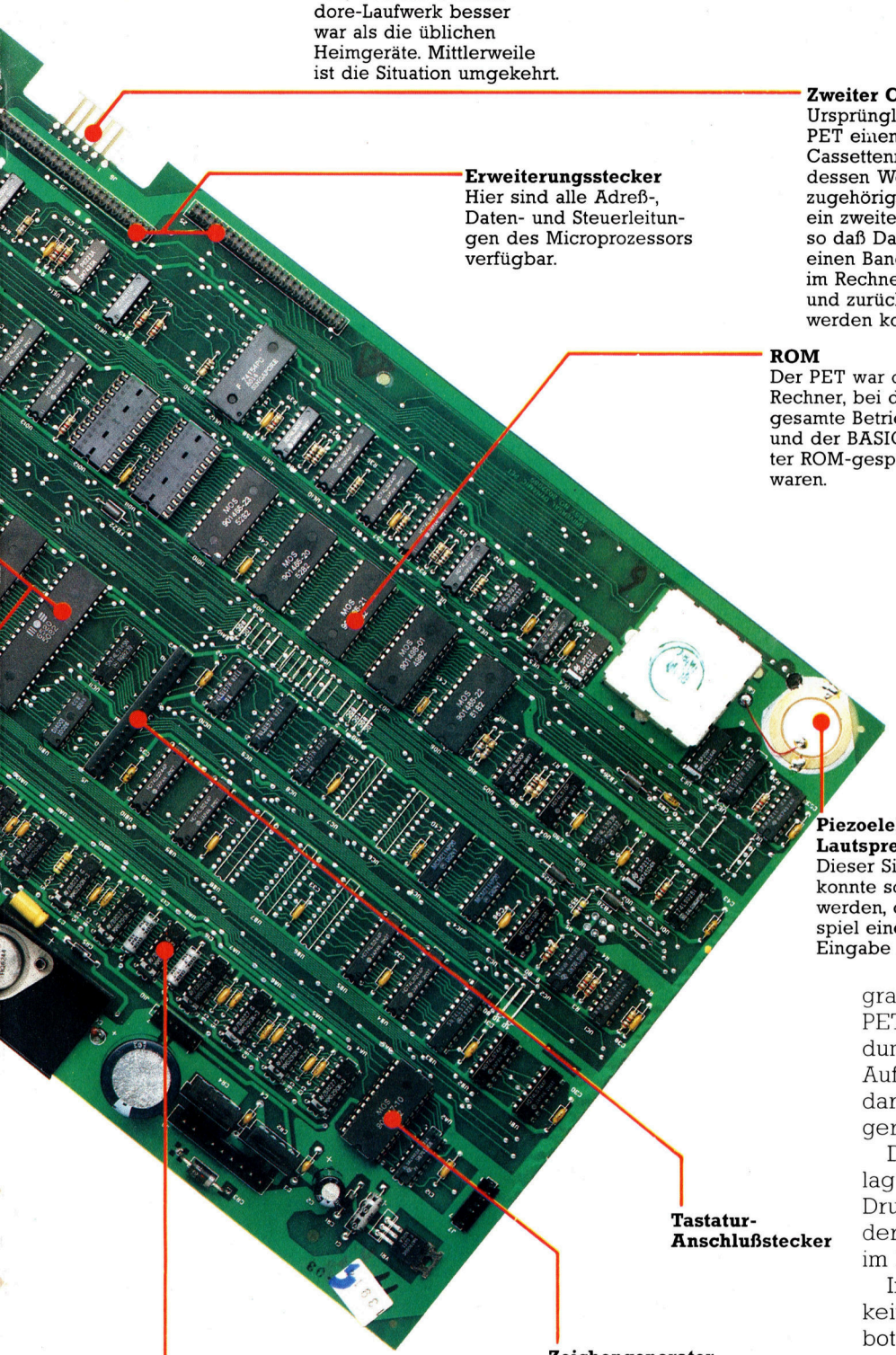
User Port
Diese Schnittstelle bietet einen parallelen 8-Bit-Port und Interface-Anschlüsse für einen externen Monitor. Vor allem läßt sich am User Port auch selbstentwickelte Zusatzelektronik betreiben.

IEEE488
Der PET war früher der einzige Microcomputer mit serienmäßigem IEEE488-Port. Darüber sind bis zu 15 Peripheriegeräte adressierbar. Der IEEE488-Bus ist auch gängiger Standard für die Meß- und Steuerungselektronik in wissenschaftlichen Labors.

VIA 6522
Dieser „Versatile Interface Adaptor“ ist, wie der 6520, ein universeller Schnittstellenbaustein, enthält aber noch ein Shift-Register für die Parallel/Seriell-Wandlung und zwei programmierbare Zeitgeber.

PIAs 6520
Über die „Peripheral Interface Adaptor“-Chips wird die Kommunikation mit der Peripherie einschließlich der Cassettenlaufwerke und der Tastatur abgewickelt.

Prozessorbaustein 6502
Da bei Commodore Chuck Peddle für den PET verantwortlich zeichnete, der zuvor bei MOS Technology den 6502 entwickelt hatte, war die Wahl der CPU keine Frage. Der 6502 ist nach wie vor im Heimcomputerbereich sehr verbreitet.



Recorder-Anschluß
Für den PET wurde ein spezieller Commodore-Recorder benötigt – anfänglich durchaus sinnvoll, weil das Commodore-Laufwerk besser war als die üblichen Heimgeräte. Mittlerweile ist die Situation umgekehrt.

Zweiter Cassetten-Port
Ursprünglich hatte der PET einen integrierten Cassettenrecorder. Nach dessen Wegfall war der zugehörige Anschluß für ein zweites Laufwerk frei, so daß Daten von dem einen Band eingelesen, im Rechner bearbeitet und zurückgeschrieben werden konnten.

Erweiterungsstecker
Hier sind alle Adreß-, Daten- und Steuerleitungen des Microprozessors verfügbar.

ROM
Der PET war der erste Rechner, bei dem das gesamte Betriebssystem und der BASIC-Interpreter ROM-gespeichert waren.

Piezoelektrischer Lautsprecher
Dieser Signalgeber konnte so programmiert werden, daß er zum Beispiel eine fehlerhafte Eingabe beanstandete.

Tastatur-Anschlußstecker

Zeichengenerator
Zusätzlich zu seinen 64 alphanumerischen Zeichen bot der PET 64 Grafiksymbole. Die Textwiedergabe erfolgte wahlweise in Groß- oder Kleinbuchstaben.

RAM-Chips
Die PETs waren standardmäßig mit 8–32 KByte RAM ausgestattet. Ein spezieller Umbau erlaubte später die Erweiterung auf 96 KByte und mehr.

Commodore „PET“ CBM 4032

Abmessungen:
480 × 440 × 300 mm

Zentraleinheit
6502, 1 MHz getaktet

Speicher
32 KByte RAM, 20 KByte ROM

Bildschirm
Integrierter grüner 30-cm-Bildschirm, 256 Zeichen und Grafiksymbole darstellbar; Textformat 25 Zeilen zu 40 Zeichen, niedrigauflösende Grafik (50 x 80)

Schnittstellen
IEEE-488-Anschluß, paralleler 8-Bit-User Port, 2 Cassettenrecorder, Buserweiterung

Mitgelieferte Sprachen
BASIC, Maschinencode mit Monitor

Weitere Sprachen
PASCAL, COMAL, LISP

Zubehör
Bedienungsanleitung

Tastatur
64 Schreibmaschinentasten mit Grafiksymbolen, getrennter Ziffernblock mit Rechenfunktionstasten

Dokumentation
Commodore erntete für seine Anleitungen nie besonders viel Beifall; allerdings ist in diesem Bereich inzwischen einiges besser geworden.

grafik-Symbolen umfaßte. Das brachte viele PET-Besitzer auf bemerkenswerte Anwendungsideen, trotz der relativ bescheidenen Auflösung. Ein Problem bestand allerdings darin, daß die Tastatur in keiner Weise normgerecht ausgelegt war.

Der ausgiebige Gebrauch der Blockgrafik lag auch deshalb nahe, weil eine Anzahl von Druckern zur Verfügung stand, die eine Wiedergabe ohne umständliche Programmierung im Bitmustermodus ermöglichten.

Infolge dieser verschiedenen Eigentümlichkeiten ist von dem großen PET-Softwareangebot nur wenig auf andere Maschinen übertragen worden. Umgekehrt gibt es auch nur wenig Übersetzungen von Fremdrechner-Programmen für den PET, weil die Anpassung meist mehr Aufwand erforderte, als alles neu zu schreiben. Das führte letztlich zu einer gewissen Isolation der PET-Familie und bewirkte, daß wichtige Entwicklungen der übrigen Branche nicht nachvollzogen wurden.

Die Neue Welt III

Mit diesem Artikel beenden wir das Simulationsspiel Neue Welt. Außerdem geben wir den Besitzern eines Schneider-Computers entsprechende Änderungshinweise für Listing.

Das auf dem Commodore 64 geschriebene Programm kann mit wenigen Änderungen auch auf den Schneider-Computern laufen. So sollte der Ausdruck PRINT CHR$(147) immer gegen CLS ausgetauscht werden. Der Ausdruck, mit dem auf einen Tastendruck gewartet wird, wie z. B.

<Zeile #> GET I$:IF I$="" THEN <Zeile #>

sollte wie folgt ersetzt werden:

<Zeile #> I$="":WHILE I$="":I$=INKEY$:WEND

Abschließend sollte die folgende Zeile eingefügt werden, damit der 40-Zeichen-Darstellungsmodus gewählt wird:

5 MODE 1

```
6530 REM LIFEBOAT
6535 IFM(1)=1THENRETURN
6536 PRINTCHR$(147)
6540 M(1)=1
6550 S$="A LIFEBOAT IS SIGHTED*":GOSUB9100
6552 S$="DRIFTING IN THE DISTANCE*":GOSUB9100
6554 PRINT:GOSUB9200
6556 S$="THROUGH THE TELESCOPE YOU SEE*":GOSUB9100
6558 S$="THAT IT CONTAINS:*":GOSUB9100
6560 PRINT:GOSUB9200
6562 S$="4 PEOPLE*":GOSUB9100
6563 GOSUB9200
6563 GOSUB9200
6564 S$="AND A LARGE CHEST!*":GOSUB9100
6566 PRINT:GOSUB9200
6568 S$="IF YOU CHANGE COURSE*":GOSUB9100
6570 S$="TO PICK THEM UP*":GOSUB9100
6572 S$="IT WILL TAKE YOU TWO EXTRA DAYS*":GOSUB91
00
6574 PRINT:GOSUB9200
6576 S$="DO YOU WANT TO PICK THEM UP (Y/N)*":GOSUB
9100
6578 INPUTI$:I$=LEFT$(I$,1)
6580 IFI$<>"Y"ANDI$<>"N"THEN6578
6585 IFI$="Y"THEN6600
6588 PRINT:GOSUB9200
6590 S$="THE LIFEBOAT DISAPPEARS....*":GOSUB9100
6592 PRINT:GOSUB9200
6594 S$=K$:GOSUB9100
6596 GETI$:IFI$=""THEN6596
6599 RETURN
6600 PRINT:GOSUB9200
6610 EW=EW+2/7
6625 IFCN<>16 THEN6630
6627 S$="YOU CAN'T PICK THEM UP*":GOSUB9100
6628 S$="YOU HAVE NO ROOM ON THE SHIP*":GOSUB9100
6629 GOTO6592
6630 X=16-CN:IFX>3THEN6635
6632 S$="YOU ONLY HAVE ROOM FOR*":GOSUB9100
6633 PRINTX;"MORE PEOPLE ON THE SHIP"
6634 PRINT:GOSUB9200
6635 S$="YOU PICK UP:*":GOSUB9100
6638 X=0
6640 FORT=1TO16
6645 IFTS(T,1)<>0THEN6679
6650 X=X+1
6655 IFX>4THENT=16:GOTO6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ";C$(TS(T,1))
6679 NEXT
6680 PRINT:GOSUB9200
6682 S$="THE CHEST CONTAINS:*":GOSUB9100
6685 FORT=1TO4
6690 X=INT(RND(1)*10)+10
6692 PRINTX;U$(T);"S OF ";P$(T)
6693 IFPA(T)=-999THENPA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO6592
```

Die Pest kann ausbrechen ...

```
6700 REM PLAGUE STRIKES
6705 IFM(2)=1THENRETURN
6706 PRINTCHR$(147)
6710 M(2)=1
6712 S$="A PLAGUE STRIKES*":GOSUB9100
6714 PRINT:GOSUB9200
6716 X=1
6718 FORT=1TO16
6720 IFTS(T,1)=2ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=0:T=16
6722 NEXT
6724 Y=1
6726 IFOA(1)<>0ANDOA(1)<>-999THENY=0
6730 Z=((X+Y)*10)+5
```

... die Widerstandskraft der Mannschaft hängt davon ab, ob ein Arzt angeheuert wurde und ob Medizin in ausreichender Menge an Bord ist.

```
6732 I$="YOU HAVE ":IFX=0ANDY=0THEN6740
6734 IFX=1THENS$="WITH NO DOCTOR*":GOSUB9100:I$="A
ND "
6736 IFY=1THEN S$=I$+"NO MEDICINE*":GOSUB9100
6740 S$="MANY OF THE CREW ARE AFFECTED*":GOSUB9100
6745 PRINT:GOSUB9200
6750 X=0
6755 FORT=1TO16
6756 IFRND(1)<.3THEN6775
6760 IFTS(T,2)=0ORTS(T,2)=-999THEN6775
6765 TS(T,2)=TS(T,2)-Z
6770 IFTS(T,2)<1THENTS(T,2)=-999:X=X+1
6775 NEXT
6776 IFY=1THEN6780
6777 S$="1/2 YOUR MEDICINE IS USED*":GOSUB9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT:GOSUB9200
6785 IFX=0THEN6797
6790 PRINT"AND";X;
6792 S$="CREW MEMBERS DIE*"
6794 IFX=1THENS$="CREW MEMBER DIES*"
6795 GOSUB9100
6796 PRINT:GOSUB9200
6797 S$=K$:GOSUB9100
6798 GETI$:IFI$=""THEN6798
6799 RETURN
```

Piraten können das Schiff angreifen ...

```
6800 REM PIRATES
6805 IFM(3)=1THENRETURN
6810 X=0
6812 FORT=1TO16
6814 IFTS(T,2)=0ORTS(T,2)=-999THENX=X+1
6815 NEXT
6816 IFX=16 THEN RETURN
6818 M(3)=1
6820 PRINTCHR$(147)
```

... sind Waffen an Bord, um den Angriff abzuwehren und die Verluste niedrig zu halten?

```
6822 S$="  PIRATES ATTACK THE SHIP!*":GOSUB9100
6824 PRINT:GOSUB9200
6825 K=2
6826 IFOA(2)=0OROA(2)=-999THENK=4
6828 S$="IN SPITE OF YOUR GUNS*"
6830 IFK=4THENS$="YOU HAVE NO GUNS*"
6832 GOSUB9100
6835 X=0
6836 FORT=1TO16
6838 IFTS(T,2)=0ORTS(T,2)=-999THEN6845
```

```
6840 X=X+1:TS(T,2)=-999
6842 IFX=KTHENT=16
6845 NEXT
6850 PRINTX;
6855 S$="OF THE CREW IS KILLED*"
6856 IFX>1THENS$="OF THE CREW ARE KILLED*"
6860 GOSUB9100
6865 PRINT:GOSUB9200
6890 S$=K$:GOSUB9100
6895 GETI$:IFI$=""THEN6895
6899 RETURN
6900 REM RUDDER
6905 IFM(4)=1THENRETURN
6910 PRINTCHR$(147)
6915 M(4)=1
6920 S$="TROUBLE WITH THE RUDDER!*":GOSUB9100
6925 PRINT:GOSUB9200
6928 X=4
6930 FORT=1TO16
6935 IFTS(T,1)=3ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=1:T=16
6938 NEXT
6940 S$="ALTHOUGH YOU HAVE A MECHANIC*"
6945 IFX=4THENS$="YOU HAVE NO MECHANIC AND*"
6950 GOSUB9100
6955 S$="YOUR JOURNEY WILL TAKE*":GOSUB9100
6960 PRINTX;"WEEKS LONGER"
6965 EW=EW+X
6967 PRINT:GOSUB9200
6969 S$=K$:GOSUB9100
6970 GETI$:IFI$=""THEN6970
6975 RETURN
7000 REM STORM
7005 IFM(5)=1THENRETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="YOU ARE BLOWN OFF COURSE*":GOSUB9100
7022 S$="IN A STORM!*":GOSUB9100
7025 PRINT:GOSUB9200
7028 X=2
7030 FORT=1TO16
7035 IFTS(T,1)=4ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=1:T=16
7038 NEXT
7040 S$="ALTHOUGH YOU HAVE A NAVIGATOR*"
7045 IFX=2THENS$="YOU HAVE NO NAVIGATOR AND*"
7049 GOTO6950
```

Wenn die Vorräte zur Neige gehen, können Sie sich entscheiden, an dieser Insel an Land zu gehen und die Vorräte wieder aufzufrischen.

```
7050 REM ISLAND
7055 IFM(6)=1THENRETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="YOUR CHARTS SHOW AN ISLAND*":GOSUB9100
7071 S$="WHERE YOU MAY BE ABLE TO*":GOSUB9100
7072 S$="RE-STOCK YOUR PROVISIONS*":GOSUB9100
7073 S$="BUT IF YOU GO THERE*":GOSUB9100
7074 S$="IT WILL ADD TIME TO YOUR JOURNEY*":GOSUB9
100
7075 PRINT:GOSUB9200
7080 S$="DO YOU WANT TO GO THERE*":GOSUB9100
7082 INPUTI$:I$=LEFT$(I$,1)
7084 IFI$<>"Y"ANDI$<>"N"THEN7082
7086 PRINT:GOSUB9200
7090 IFI$="N"THEN7145
7100 S$="YOU REACH THE ISLAND*":GOSUB9100
7105 S$="AND OBTAIN:*":GOSUB9100
7106 IFB$="N"THEN7110
7107 PRINT:GOSUB9200
7108 PRINT"NOTHING!":GOSUB9200
7109 S$="(REMEMBER THE ALBATROSS!)":GOSUB9100:GOTO
7130
7110 FORT=1TO4
7112 IFRND(1)<.25THEN7129
7115 X=INT(RND(1)*10)+5
7120 PRINTX;U$(T);"S OF";P$(T)
7122 IFPA(T)=-999THENPA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="BUT THE JOURNEY WILL NOW TAKE*":GOSUB9100
7135 X=INT(RND(1)*2)+1
7139 PRINTX;:S$="WEEKS LONGER*":GOSUB9100
7140 EW=EW+X
7145 PRINT:GOSUB9200
7150 S$=K$:GOSUB9100
7155 GETI$:IFI$=""THEN7155
7159 RETURN
```

Insgesamt gibt es sieben Faktoren, die zu einer Meuterei der Mannschaft führen können. Die Routine generiert einen Meutereifaktor MF, indem am Ende einer Reisewoche Bilanz gezogen wird.

```
7200 REM MUTINY
7210 MF=0
7215 IFH$="Y"THENMF=MF+30
7220 NC=0
7225 FORT=1TO16
7228 IFTS(T,1)=5ANDTS(T,2)<>0ANDTS(T,2)<>-999THENN
C=1:T=16
7230 NEXT
7235 IFNC=0THENMF=MF+30
7240 IFA$="Y"THENMF=MF-20
7245 IFB$="Y"THENMF=MF+30
7250 IFCN>12THENMF=MF+30
7255 IFWT>MOTHENMF=MF+30
7260 IFWK>8THENMF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
```

Steigt der Meutereifaktor über den Wert 75 (jedoch nicht über 100), wird der Spieler vor Unruhen unter der Mannschaft gewarnt.

```
7280 IFMF<75 THENRETURN
7282 PRINTCHR$(147)
7284 IFMF>100THEN7300
7285 S$="CONDITIONS ON THE SHIP*":GOSUB9100
7286 S$="ARE GETTING WORSE*":GOSUB9100
7287 S$="AND SOME OF THE CREW*":GOSUB9100
7288 S$="ARE TALKING OF MUTINY!*":GOSUB9100
7290 PRINT:GOSUB9200
7292 S$=K$:GOSUB9100
7294 GETI$:IFI$=""THEN7294
7299 RETURN
```

Steigt der Meutereifaktor über 100, rebelliert die Mannschaft. Der Kapitän wird auf offener See ausgesetzt. Das Spiel ist aus.

```
7300 PRINTCHR$(147)
7305 PRINT:GOSUB9200
7310 S$="THE CREW HAVE MUTINIED*":GOSUB9100
7312 S$="BECAUSE:*":GOSUB9100
7313 X=0
7314 IFH$<>"Y"THEN7320
7315 GOSUB9200:X=X+1:PRINTX;
7316 S$="THEY HAVE BEEN ON 1/2 RATIONS*":GOSUB9100
7318 S$="   FOR SOME OF THE VOYAGE*":GOSUB9100
7320 IFNC<>0THEN7325
7321 GOSUB9200:X=X+1:PRINTX;
7322 S$="THERE IS NO COOK*":GOSUB9100
7324 S$="   AND THE FOOD IS AWFUL*":GOSUB9100
7325 IFB$<>"Y"THEN7330
7326 GOSUB9200:X=X+1:PRINTX;
7327 S$="THE ALBATROSS WAS KILLED!*":GOSUB9100
7330 IFCN<13THEN7335
7331 GOSUB9200:X=X+1:PRINTX;
7332 S$="THE SHIP IS OVERCROWDED*":GOSUB9100
7335 IFMO>=WT THEN7340
7336 GOSUB9200:X=X+1:PRINTX;
7337 S$="THERE ISN'T ENOUGH GOLD*":GOSUB9100
7338 S$="   FOR THEIR WAGES*":GOSUB9100
7340 IFWK<=8 THEN7350
7341 GOSUB9200:X=X+1:PRINTX;
7342 S$="THEY HAVE BEEN AT SEA*":GOSUB9100
7343 S$="   FOR MORE THAN 8 WEEKS*":GOSUB9100
7350 PRINT:GOSUB9200
7360 S$="THE CREW SEIZE THE SHIP*":GOSUB9100
7362 GOSUB9200
7363 S$="AND SAIL AWAY*":GOSUB9100
7370 GOSUB9200
7372 S$="LEAVING YOU ADRIFT IN *":GOSUB9100
7373 S$="AN OPEN BOAT*":GOSUB9100
7374 S$="LET'S HOPE YOU ARE PICKED UP":GOSUB9100
7375 PRINT:GOSUB9200
7380 PRINT"        GAME OVER"
7382 END
```

Diese Unterroutinen werden an verschiedenen Stellen verwendet, um etwa die Bildschirmausgabe zu verlangsamen.

```
9100 REM SLOW PRINT OF S$
9110 FOR S3=1TO32
9115 IFMID$(S$,S3,1)="*"THENS3=32:GOTO9140
9120 PRINTMID$(S$,S3,1);
9130 FORS4=1TO20:NEXT
9140 NEXT:PRINT
9199 RETURN
9200 REM DELAY LOOP
9210 FORS5=1TO1000:NEXT
9299 RETURN
9300 REM REDUCE CREW STRENGTHS BY WF
9310 FORS1=1TO16
9315 IFTS(S1,2)=0THEN9340
9320 TS(S1,2)=TS(S1,2)-WF
9330 IFTS(S1,2)<1THENTS(S1,2)=-999
9340 NEXT
9399 RETURN
```

Das Schiff erreicht sein Ziel . . .

```
10000 REM ARRIVAL AT NEWWORLD
10001 PRINTCHR$(147):GOSUB9200
10005 S$="YOU ARRIVE AT THE NEW WORLD*":GOSUB9100
10006 PRINT:GOSUB9200
10007 S$="AS YOU APPROACH THE SHORE*":GOSUB9100
10009 S$="NATIVES PADDLE OUT TO MEET YOU*":GOSUB9100
10010 PRINT:GOSUB9200
10015 IFOA(2)=0THEN10050
10017 S$="THEY LOOK FIERCE AND ARE ARMED!!*":GOSUB
9100
10018 PRINT:GOSUB9200
10020 S$="DO YOU OPEN FIRE? (Y/N)*":GOSUB9100
10022 INPUTI$:I$=LEFT$(I$,1)
10024 IFI$<>"N"ANDI$<>"Y"THEN10022
10026 IFI$="N"THEN10050
10028 PRINT:GOSUB9200
10030 S$="MANY NATIVES ARE KILLED*":GOSUB9100
10032 S$="BUT DURING THE NIGHT*":GOSUB9100
10034 S$="OTHERS RETURN*":GOSUB9100
10036 S$="AND BURN YOUR SHIP!!!!*":GOSUB9100
10038 PRINT:PRINT:GOSUB9200
10040 S$="        GAME OVER*":GOSUB9100
10042 END
10044 GOTO10042
10050 S$="THEY TAKE YOU TO MEET THEIR*":GOSUB9100
10052 S$="CHIEF. HE HAS MET YOUR RACE*":GOSUB9100
10054 S$="BEFORE AND IS FRIENDLY.*":GOSUB9100
10056 GOSUB9200
10058 S$="THE CREW ARE FED AND RESTED*":GOSUB9100
10060 PRINT:GOSUB9200
10062 S$="THE NEXT DAY TRADING BEGINS*":GOSUB9100
10064 PRINT:GOSUB9200
10066 S$=K$:GOSUB9100
10068 GETI$:IFI$=""THEN10068
10069 RETURN
```

. . . und der Handel beginnt.

```
10070 PRINTCHR$(147):GOSUB9200:REM TRADING
10072 IFOA(2)=0THEN10080
10074 S$="THE CHIEF DOESN'T WANT YOUR*":GOSUB9100
10076 S$="GUNS - THEY WILL CAUSE TROUBLE*":GOSUB9100
10078 PRINT:GOSUB9200
10080 IFOA(3)<>0OROA(4)<>0OROA(5)<>0OROA(6)<>0THEN
10085 S$="YOU HAVE NO GOODS LEFT*":GOSUB9100
10090 S$="FOR TRADING*":GOSUB9100
10095 GOTO10038
10100 S$="IN EXCHANGE FOR ANY KNIVES*":GOSUB9100
10102 S$="SALT CLOTH OR JEWELS YOU HAVE*":GOSUB910
10104 S$="HE OFFERS YOU PEARLS CARVINGS*":GOSUB910
10106 S$="AND SPICES*":GOSUB9100
10108 PRINT:GOSUB9200
10110 S$="WHEN YOU LEFT PORT THESE WERE*":GOSUB910
10112 S$="WORTH:*":GOSUB9100
10114 S$="PEARLS - 2 GLD PCS EACH*":GOSUB9100
10116 S$="CARVINGS - 2 GLD PCS EACH*":GOSUB9100
10118 S$="SPICES - 1 GLD PC PER GRAM*":GOSUB9100
10120 PRINT:GOSUB9200
10122 S$="BUT THEIR VALUE MAY HAVE*":GOSUB9100
10124 S$="CHANGED WHEN YOU GET HOME*":GOSUB9100
10125 PRINT:GOSUB9200:S$=K$:GOSUB9100
10126 GETI$:IFI$=""THEN10126
10130 FORT=3TO6
10135 IFOA(T)=0THEN10200
10140 PRINTCHR$(147):GOSUB9200
10145 PRINT"YOU HAVE";OA(T);
10150 IFT=3THENS$="BAGS OF SALT*"
10151 IFT=4THENS$="BALES OF CLUTH*"
10152 IFT=5THENS$="KNIVES*"
10153 IFT=6THENS$="JEWELS*"
10155 GOSUB9100
10156 PRINT:GOSUB9200
10160 S$="FOR THESE THE CHIEF OFFERS YOU*":GOSUB9100
10165 PRINT"EITHER";OA(T)*EQ(T-2,1);"PEARLS"
10166 PRINT"     OR";OA(T)*EQ(T-2,2);"CARVINGS"
10167 PRINT"     OR";OA(T)*EQ(T-2,3);"GRAMS OF SPICE"
10168 PRINT:GOSUB9200
10170 S$="DO YOU WANT PEARLS,CARVINGS*":GOSUB9100
10172 S$="OR SPICES?*":GOSUB9100
10174 S$="(ENTER 1,2 OR 3)*":GOSUB9100
10175 INPUTI$
10176 I=VAL(I$):IFI<1ORI>3THEN10174
10180 AO(I)=AO(I)+OA(T)*EQ(T-2,I))
10190 PRINT:PRINT"THE ";T$(I);" ARE PUT ON THE SHIP"
10192 S$=K$:GOSUB9100
10194 GETI$:IFI$=""THEN10194
10200 NEXT
10210 PRINT:PRINT:GOSUB9200
10215 S$="      END OF TRADING*":GOSUB9100
10216 PRINT:GOSUB9200
10218 S$="YOU HAVE OBTAINED:*":GOSUB9100
10220 PRINTAO(1);"PEARLS"
10222 PRINTAO(2);"CARVINGS"
10224 PRINTAO(3);"GRAMS OF SPICE"
10226 PRINT:GOSUB9200
10228 S$=K$:GOSUB9100
10229 GETI$:IFI$=""THEN10229
10230 RETURN
```

Wollen Sie an einem Aufstand teilnehmen? Der Gewinn ist hoch – aber auch das Risiko, daß das Unterfangen mißlingt.

```
10300 REM REVOLUTION
10305 IFOA(2)=0THENRETURN
10310 PRINTCHR$(147):GOSUB9200
10315 S$="DURING THE NIGHT A RIVAL OF THE*":GOSUB9100
10316 S$="CHIEF VISITS THE SHIP IN SECRET*":GOSUB9100
10317 PRINT:GOSUB9200
10318 S$="HE WANTS TO BUY YOUR GUNS FOR*":GOSUB9100
10320 S$="A REVOLUTION*":GOSUB9100
10322 PRINT:GOSUB9200
10324 S$="HE OFFERS YOU 30 PEARLS PER GUN*":GOSUB9100
10326 S$="DO YOU SELL HIM THE GUNS?(Y/N)*":GOSUB9100
10328 INPUTI$:I$=LEFT$(I$,1)
10330 IFI$<>"N"ANDI$<>"Y"THEN10328
10332 IFI$="Y"THEN10400
10334 PRINT:GOSUB9200
10336 S$="THE CHIEF FINDS OUT AND IS*":GOSUB9100
10338 S$="PLEASED WITH YOU*":GOSUB9100
10340 S$="HE GIVES YOU FREE PROVISIONS*":GOSUB9100
10342 S$="FOR THE JOURNEY BACK*":GOSUB9100
10344 GOSUB9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="AND 50 PEARLS!!*":GOSUB9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB9200
10352 S$=K$:GOSUB9100
10354 GETI$:IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147):GOSUB9200
10405 IFRND(1)<.75THEN10450
10410 S$="THE REVOLUTION IS SUCCESSFUL*":GOSUB9100
10412 PRINT:GOSUB9200
10415 S$="THE NEW CHIEF REWARDS YOU WITH*":GOSUB900
10420 S$="FREE PROVISIONS FOR THE RETURN*":GOSUB9100
10425 S$="JOURNEY.*":GOSUB9100
10429 AO(1)=AO(1)+(OA(2)*30):REM ADD PEARLS
10430 OA(2)=0
10431 GOTO10350
10450 S$="THE REVOLUTION FAILS!!*":GOSUB9100
10452 PRINT:GOSUB9200
10455 S$="THE OLD CHIEF IS ANGRY WITH YOU*":GOSUB9100
10457 S$="HE BURNS YOUR SHIP AND STEALS*":GOSUB910
10458 S$="EVERYTHING!!*":GOSUB9100
10459 PRINT:GOSUB9200
10460 S$="    GAME OVER!!*":GOSUB9100
10462 END
10464 GOTO10462
10500 REM END OF VOYAGE
```

Das Schiff kehrt zum Heimathafen zurück, seine Waren werden begutachtet.

```
10501 PRINTCHR$(147):GOSUB9200
10505 S$="WITH A STRONG CREW AND GOOD*":GOSUB9100
10507 S$="WINDS THE JOURNEY BACK GOES*":GOSUB9100
10508 S$="WELL AND TAKES ONLY 8 WEEKS*":GOSUB9100
10512 WW=0
10514 FORT=1TO5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB9200
10520 S$="WAGE BILL FOR RETURN VOYAGE = *":GOSUB9100
10522 PRINTWW;"GOLD PIECES"
10524 PRINT:GOSUB9200
10526 S$="WHEN YOU GET BACK*":GOSUB9100
10528 S$="YOU SELL YOUR TRADING GOODS*":GOSUB9100
10530 S$="THE GOODS ARE NOW WORTH:*":GOSUB9100
10532 PRINT"PEARLS - ";V2(1);"GOLD PIECES"
10534 PRINT"CARVINGS - ";V2(2);"GOLD PIECES"
10536 PRINT"SPICES - ";V2(3);"GOLD PIECES"
10538 PRINT:S$="YOU GET A TOTAL OF*":GOSUB9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINTX;"GOLD PIECES"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="YOU NOW HAVE:*":GOSUB9100
10552 PRINTMO+X;"GOLD PIECES"
10555 PRINT:GOSUB9200
10556 S$="THE WAGE BILL FOR THE VOYAGE IS*":GOSUB9100
10557 PRINTWT+WW;"GOLD PIECES"
10559 PRINT:GOSUB9200
10560 S$="YOU END THE GAME WITH:*":GOSUB9100
10562 Z=MO+X-WT-WW
10565 PRINTZ;"GOLD PIECES"
10566 PRINT:GOSUB9200
10567 PRINT:S$="YOUR RATING IS:*":GOSUB9100:PRINT
10568 IF Z>3200 THEN S$="ARCH CAPITALIST*":GOSUB 9
100:END
10569 IF Z>2500 THEN S$="MASTER TRADER*"GOSUB9100:
END
10570 IF Z>2000 THEN S$="MERCHANT CLASS III*":GOSU
B9100:END
10571 IF Z>1000 THEN S$="A PROPER JONAH*":GOSUB 91
00:END
10572 S$="MORE OF A DUCK THAN A DRAKE*"
10573 GOSUB9100:END
```

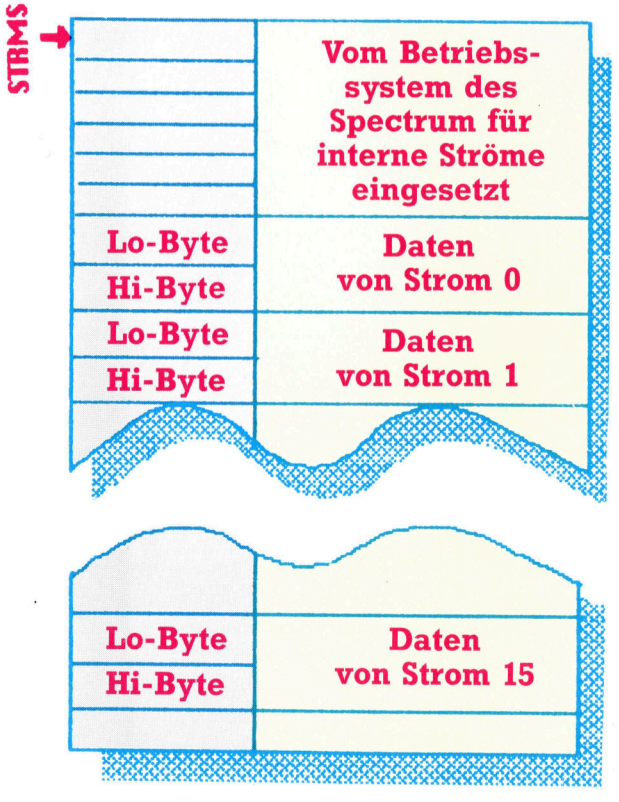
Stromlinien

In der vorigen Folge hatten wir gezeigt, wie das Betriebssystem des Spectrum seine Ein- und Ausgabekanäle anwählt und steuert. Nun betrachten wir die Datenströme, die über diese Kanäle „fließen".

Die Initialisierung des Spectrum ordnet beim Anschalten oder bei einem Reset Datenströme bestimmten Kanälen zu. Unsere Tabelle zeigt diese Zuordnungen:

Kanal	Strom	Kanalbeschreibung
K	0	Tastatur und unterer
K	1	Bildschirmbereich
S	2	oberer Bildschirmbereich
P	3	ZX-Drucker

Es gibt jedoch auch Ströme, die auf einem Standard-Spectrum nicht genutzt werden. Beachten Sie daher, daß sich die folgenden Beschreibungen auf einen Spectrum ohne Interface 1 beziehen. Wenn das lokale Netzwerk, die Microdrives oder die serielle Schnittstelle in Betrieb sind, ändert sich der gesamte Aufbau. Wir gehen auf die Ausbaustufen des Spectrum zu einem späteren Zeitpunkt genauer ein. Die weiteren Ströme haben die Nummern 4 bis 15 und können jedem E/A-Kanal zugeordnet werden, den Sie als Schnittstelle einsetzen wollen.



Das Betriebssystem des Spectrum speichert die Information, welche Ströme welchen Kanälen zugeordnet sind, in den 38 Bytes der Tabelle STRMS. Die Tabelle befindet sich im Bereich für Systemvariablen bei 23568. Die ersten sechs Tabellenbytes braucht das OS für eigene Zwecke.

Die weiteren Bytes sind paarweise jedem der 16 Ströme zugeordnet. Unser Bild zeigt den Tabellenaufbau.

Inhalte der Tabelle

Das folgende BASIC-Programm stellt den Inhalt der Tabelle dar:

```
10 FOR I=23568 TO 23604 STEP 2
20 PRINT PEEK I + 256 * PEEK (I+1)
30 NEXT I
```

Die ersten drei Einträge bezeichnen die internen Ströme des Spectrum. Alle darauffolgenden Werte sind Vektoren der Ströme 0 bis 15 und zeigen auf die Einträge einer weiteren Tabelle. Die Systemvariable CHANS (bei 23631) gibt an, wo diese Tabelle liegt. Man kann die Adresse mit

```
PRINT PEEK 23631 + 256 * PEEK 23632
```

herausfinden. Die Systemvariable CHANS sollte nie verändert werden, da sie auf den Anfang der Tabelle für Kanaldaten zeigt, die wichtige Informationen über jeden Kanal enthält. Der Wert der STRMS-Tabelle dient als Index für die Kanaldatentabelle.

Der entsprechende Eintrag gibt mit (Offset + 1) an, wo sich – vom Tabellenanfang aus – die Informationen über einen bestimmten Kanal befinden. Der STRMS-Tabelleneintrag für Strom 3 (Drucker) ist daher 16. Wenn Sie davon Eins abziehen, erhalten Sie den Offset von 15. Die Daten über Kanal 3 befinden sich also bei der Adresse

```
(PEEK 23361 + 256 * PEEK 23632) + 15
```

(vorausgesetzt, die Strom- und Kanalzuordnung wurde nach der Initialisierung nicht verändert).

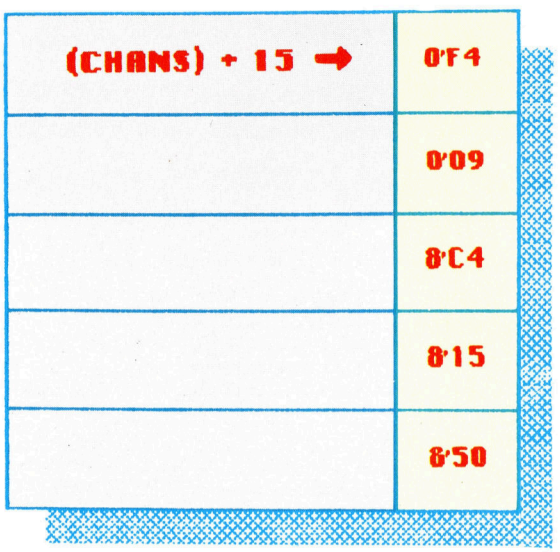
Ein Null-Eintrag der STRMS-Tabelle zeigt an, daß der Strom noch keinem Kanal zugeordnet wurde. Auf einem Standard-Spectrum ist das bei Strom 4 bis 15 der Fall.

Jeder Eintrag der Kanaldatentabelle hat das Format:

Lo-Byte	Adresse der Ausgaberoutine
Hi-Byte	des Kanals
Lo-Byte	Adresse der Eingaberoutine
Hi-Byte	des Kanals
n	Einzelnes Byte, das den Buchstaben des Kanalcodes im ASCII-Format darstellt, d. h. K, S oder P

Ein Eintrag der STRMS-Tabelle zeigt auf das zweite Byte des entsprechenden Eintrags in der Kanaldatentabelle. Daher besteht die Notwendigkeit, Eins zu subtrahieren. Diese Tabelle bildet das Herz des E/A-Systems. Wenn ein Kanal einen E/A-Vorgang nicht ausführen kann, zeigt der entsprechende Eintrag auf eine ROM-Routine, die die Fehlermeldung „Invalid I/O Device" ausgibt. Auf den reinen Ausgabekanälen (etwa Kanal S) zeigt der Adreßeintrag der Eingaberoutine auf dieses Fehlermodul. In jedem anderen Fall steht dafür die entsprechende ROM-Routine für Ein- oder Ausgabe zur Verfügung.

Sehen wir uns den Eintrag für Kanal P einmal genauer an. Normalerweise ist er ein reiner Ausgabekanal, der Daten zum ZX-Drucker sendet. Seine Tabellendaten haben wir grafisch dargestellt.



Der Eintrag für Kanal S liegt bei (CHANS)+5 und der von Kanal K bei (CHANS)+0. (CHANS) ist die in der Systemvariable CHANS gespeicherte Adresse.

Mit den Befehlen OPEN# und CLOSE# lassen sich nicht belegte Ströme den Kanälen zuordnen. Sie kommen jedoch erst in einem erweiterten System (z. B. mit Interface 1) voll zum Tragen. Hier das Einsatzformat dieser Befehle auf einem Standard-Spectrum:

```
OPEN #4,"S"
```

ordnet Strom 4 dem Kanal S zu. Wenn Sie diesen Befehl ausgeben und dann die STRMS Tabelle untersuchen, werden Sie unter Strom 4 einen Eintrag finden. Der Befehl kann auch Variablen verarbeiten:

```
10 LET n=4
20 LET c$="S"
30 OPEN #n,c$
```

führt exakt das gleiche aus, wie der einzeilige OPEN-Befehl. Nach Öffnen eines Kanals können Sie mit PRINT# und LIST# Daten über diesen neuen Strom senden.

```
40 PRINT#4;"Hallo"
```

gibt „Hallo" auf dem Bildschirm aus. LIST #4 listet dort das aktuelle Programm. Wenn ein Strom einem Kanal zugeordnet wurde, der sich für Ein- und Ausgabe eignet, kann auch der Befehl INPUT# eingesetzt werden. Die erweiterten PRINT- und INPUT-Anweisungen zeigen ihren Nutzen jedoch nur dann, wenn die zusätzlichen Kanäle des Interface 1 zur Verfügung stehen.

CLOSE und OPEN

Der Befehl CLOSE# löst Kanal- und Stromzuordnungen auf. Die Aussage von CLOSE#4 (nach dem obenstehenden OPEN-Befehl) koppelt daher Strom 4 von Kanal S ab. Danach erzeugt jeder Versuch, Daten in den „geschlossenen" Strom zu senden, eine Fehlermeldung.

Die Befehle PRINT#, LIST#, INPUT# und INKEY$# lassen sich auch mit Strom 0 bis 3 einsetzen. Diese Ströme sind ständig eröffnet, so daß der Befehl

```
10 CLOSE #2:REM sollte Kanal S schließen
20 PRINT #2; "Hallo"
```

keine Fehlermeldung erzeugt: Das OS stellt sicher, daß der Kanal wieder eröffnet ist, wenn Daten dorthin gesandt werden. Was löst nun der Befehl LIST# aus?

```
LIST#3
```

führt unter normalen Umständen den gleichen Ablauf aus wie LLIST – das Programm wird auf Kanal P (Drucker) ausgegeben. LIST#0 und PRINT#0 sind schon interessanter, da sie den unteren Bildschirmbereich ansprechen (der für normale PRINT-Befehle nicht erreichbar ist). Denken Sie jedoch daran, daß dieser Bereich von dem OS des Spectrum mit Meldungen übrschrieben wird. PRINT#3 entspricht unter normalen Umständen LPRINT.

Das Verhalten eines Kanals kann auch verändert werden. So wird Kanal P oft umgestellt, um über eine entsprechende Schnittstelle einen Centronics Drucker betreiben zu können. Wenn Sie in der Kanaldatentabelle den Eintrag für die Ausgaberoutine ändern, dann senden LPRINT und LLIST die Daten an eine Routine, die einen Centronics-Drucker steuert. Statt des Druckers lassen sich auch andere Geräte mit der Schnittstelle verbinden.

Hi-Res Printprogramm

Das folgende Programm leitet Daten, die über den Kanal P des Spectrum gesandt werden, zu einer Routine, die sie in hoher Auflösung anzeigt. Das Listing kann über einen Standardassembler eingegeben werden. Sie können den Hexcode auch direkt mit dem Monitor oder dem ebenfalls abgedruckten Hex-Ladeprogramm eingeben. Nach Eingabe und Aufruf des Hex-Ladeprogramms werden Sie nach der Anfangsadresse gefragt (geben Sie hier 65068 an). Sie können nun den Hexcode zweizeichenweise eingeben. Die Codezeile etwa

2130FE

wird dabei als

21 (ENTER)
30 (ENTER)
FE (ENTER)

eingetippt. Sie beenden das Programm, indem Sie S eintippen. Über die Prüfsumme (29155) können Sie nun feststellen, ob die Eingabe wirklich fehlerfrei ist.

Der neue, vom Programm gesteuerte Befehl hat das Format:

LRPINT AT, X,Y;
"MESSAGE"

wobei X zwischen 0 und 168 und Y im Bereich von 0 bis 248 liegen muß. Die „MESSAGE" kann auch eine Stringvariable oder eine Zahl sein. Falls der mit dem neuen Befehl gedruckte String über den Bildschirmrand hinausgeht, setzt er sich zwar am linken Bildschirmrand fort, jedoch auf der gleichen Zeile.

Hex-Ladeprogramm

```
  10 CLEAR 63999
  15 POKE 23658,8
  17 DEF FN H(H$)=16*(CODE (H$)-48-7*(H$
(1)>"9"))+(CODE (H$(2))-48-7*(H$(2)>"9")
)
  20 INPUT "Start address ";a
  25 LET Z=A
  30 INPUT (a), LINE a$
  35 IF A$="S" THEN  GO TO 90
  40 PRINT a,a$: POKE 23692,255
  50 FOR Z=1 TO LEN A$/2
  60 POKE a,FN H(a$(((z*2)-1) TO )): LET
 a=a+1
  70 NEXT z
  80 GO TO 30
  90 INPUT "CHECKSUM:";D
 100 LET C=0: FOR B=Z TO A-1: LET C=C+PE
EK B: NEXT B
 110 IF C=D THEN  PRINT "The code is OK!
": STOP
 120 PRINT "Oh dear , the code is incorr
ect": STOP
```

Assemblerlisting

```
                ORG   65068
          PIXAD EQU   22AAH
          UDG   EQU   23675
          CHARS EQU   23606
                ;     "Make the data in CHANS point to our routine"
2A4F5C    ENABL LD    HL,(23631)
010F00          LD    BC,15
09              ADD   HL,BC
013AFE          LD    BC,DO-IT
71              LD    (HL),C
23              INC   HL
70              LD    (HL),B
C9              RET
                ;     "Preserve regesters and call new print routine"
E5        DO-IT PUSH  HL
C5              PUSH  BC
D5              PUSH  DE
F5              PUSH  AF
CD46FE          CALL  DOIT1
F1              POP   AF
D1              POP   DE
C1              POP   BC
E1              POP   HL
C9              RET
                ;     "Check to see if a holdsan AT control code"
F5        DOIT1 PUSH  AF
3A1DFF          LD    A,(ATFLG)
FE00            CP    0
200B            JR    NZ,GETXP
F1              POP   AF
FE16      ATCHQ CP    22
201D            JR    NZ,CRCHQ
3EFF            LD    A,255
321DFF          LD    (ATFLG),A
C9              RET
FEFE      GETXP CP    254
2809            JR    Z,GETYP
F1              POP   AF
3216FF          LD    (XPOSI),A
211DFF          LD    HL,ATFLG
35              DEC   (HL)
C9              RET
F1        GETYP POP   AF
3217FF          LD    (YPOSI),A
3E00            LD    A,0
321DFF          LD    (ATFLG),A
C9              RET
                ;     "Check for end of print data (signified by 13)"
FE0D      CRCHQ CP    13
2001            JR    NZ,VCHRQ
C9        SKIPC RET
                ;     "If the code in a is    between 32 & 128 or if"
                ;     "it is a UDG code then  find its data in memory"
FE20      VCHRQ CP    32
38OC            JR    C,PRNT?
FEB0            CP    128
381C            JR    C,FCHR
FE90      UDGCQ CP    144
3804            JR    C,PRNT?
FEA5            CP    165
3804            JR    C,FUDGC
                ;     "For all other codes a ?will be printed"
3E3F      PRNT? LD    A,63
1810            JR    FCHR
                ;     "Find the data for the  UDG or character in mem"
D690      FUDGC SUB   144
210000          LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A7B5C          LD    HL,(UDG)
19              ADD   HL,DE
180C            JR    PRNIT
210000    FCHR  LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A365C          LD    HL,(CHARS)
19              ADD   HL,DE
010700    PRNIT LD    BC,7
09              ADD   HL,BC
221BFF          LD    (CHRAD),HL
                ;     "Check to see that the X& Y positions are valid"
3A16FF          LD    A,(XPOSI)
FEF9            CP    249
D231FF          JP    NC,ERRB
3A17FF          LD    A,(YPOSI)
FEA9            CP    169
3078            JR    NC,ERR5
ED4B16FF        LD    BC,(XPOSI)
CDAA22          CALL  PIXAD
321AFF          LD    (PIXPO),A
221BFF          LD    (DFADD),HL
                ;     "Get the first pixel slice of char being printed"
0608            LD    B,8
C5        PRNLP PUSH  BC
2A1BFF          LD    HL,(CHRAD)
7E              LD    A,(HL)
2B              DEC   HL
221BFF          LD    (CHRAD),HL
6F              LD    L,A
                ;     "If the pixel slice doesn't need moving within"
                ;     "the display byte then  jump forward"
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAE6FE          JP    Z,PUTIT
                ;     "Move the slice along in the display bytes to"
                ;     "the correct pixel position"
47              LD    B,A
2600            LD    H,0
CB3D      ROTLP SRL   L
CB1C            RR    H
A7              AND   A
10F9            DJNZ  ROTLP
                ;     "Put the slice at the   right screen location"
ED5B1BFF  PUTIT LD    DE,(DFADD)
1A              LD    A,(DE)
AD              XOR   L
12              LD    (DE),A
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAFAFE          JP    Z,PST
13              INC   DE
1A              LD    A,(DE)
AC              XOR   H
12              LD    (DE),A
1B              DEC   DE
2A1BFF    PST   LD    HL,(DFADD)
CD1EFF          CALL  ULINE
221BFF          LD    (DFADD),HL
C1              POP   BC
10C4            DJNZ  PRNLP
                ;     "Add 8 to the X positionso that the next char"
                ;     "is printed in a clear  space"
3A16FF          LD    A,(XPOSI)
C608            ADD   8
                ;     "If the end of the line is reached go back to"
                ;     "the start BUT DO NOT   move down a line"
FEF9            CP    249
DA12FF          JP    C,STAIT
3E00            LD    A,0
3216FF    STAIT LD    (XPOSI),A
C9              RET
00        XPOSI DEFB  0
00        YPOSI DEFB  0
0000      DFADD DEFW  0
00        PIXPO DEFB  0
0000      CHRAD DEFW  0
00        ATFLG DEFB  0
                ;     "Move the address in HL one pixel up the screen"
F5        ULINE PUSH  AF
7C              LD    A,H
25              DEC   H
E607            AND   7
200A            JR    NZ,END
7D              LD    A,L
D620            SUB   32
6F              LD    L,A
3804            JR    C,END
7C              LD    A,H
C608            ADD   8
67              LD    H,A
F1        END   POP   AF
C9              RET
                ;     "Create various errors"
CF        ERRB  RST   8
0A              DEFB  10
CF        ERR5  RST   8
04              DEFB  4
          FINIS END
```

Wilder Wort-Wirbel

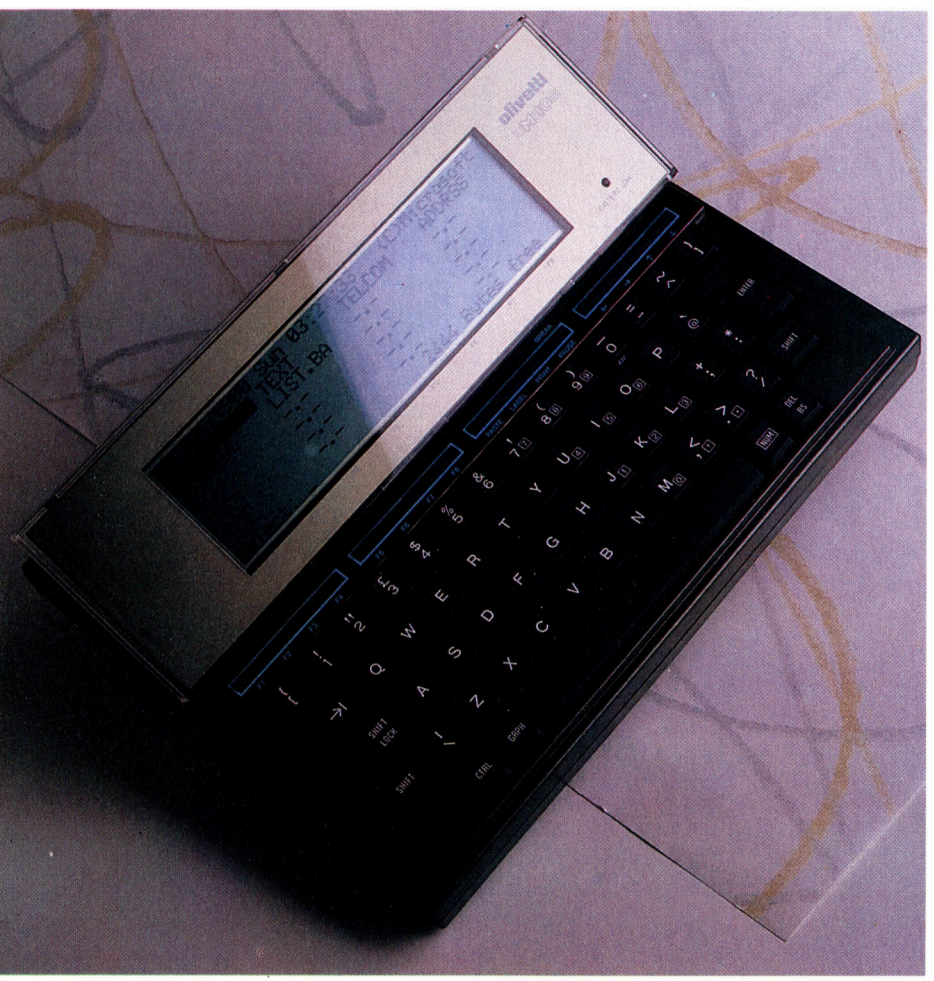
Die Textverarbeitung ist mit Abstand das beliebteste „ernsthafte" Einsatzfeld für Heimcomputer – entsprechende Programmpakete gibt es mittlerweile für fast jeden Rechner. Wir geben eine Übersicht.

Eine Textverarbeitung besteht aus Abläufen, bei denen der Benutzer voll engagiert ist, und mehr oder weniger selbständigen Operationen des Computers. Benutzerintensiv ist vor allem die Texteingabe über die Tastatur, während das Eröffnen von Textdateien (normalerweise ASCII-Files) im RAM oder auf Diskette, das Editieren und Ausdrucken nur wenig Anwenderaktivität beanspruchen. Im Korrekturstadium, wenn der Text bearbeitet und vielfältig umgestellt werden muß, sind Benutzer und Rechner gleichermaßen gefordert.

Einige Programme zeichnen sich besonders durch Perfektion bei der Textwiedergabe aus – bei MacWrite etwa lassen sich auf dem Schirm die verschiedensten Schrifttypen darstellen – andere dagegen durch spezielle Tricks beim Hantieren mit den Dateien. Es kommt aber weniger auf solche Äußerlichkeiten an als auf das Leistungsangebot im Bereich der anspruchsvollen Textbearbeitungsprobleme.

Benutzerfreundlichkeit und Flexibilität solcher Systeme sind inzwischen so selbstverständlich geworden, daß kaum noch jemand die komplexe Software dahinter würdigt. Dabei gibt es derartige Programme noch gar nicht sehr lange, und zwar aus mehreren Gründen: Erstens ist für die Textwiedergabe ein Bildschirm unerläßlich, und der gehörte vor zehn Jahren noch nicht zur üblichen Ausstattung eines Computers. Zweitens sind Digitalrechner primär für Zahlen- und nicht Textverarbeitung ausgelegt. Die Einführung geeigneter Codes wie des ASCII-Standards schafft zwar prinzipiell Abhilfe, aber die Speicherraumausnutzung ist dabei keinesfalls optimal.

Weil es für die Textverarbeitung kein Patentrezept gibt, sind die verschiedensten Systeme auf dem Markt, in jeder Form, Größe und Preislage – von 20 000 Mark für spezielle Anlagen bis zu 40 Mark für ein Heimcomputerprogramm. Hier sollen Beispiele aus allen Kategorien vorgestellt werden.



Kleine tragbare Rechner stoßen am Markt auf zunehmendes Interesse – sie bieten den Vorteil, daß sie auch „unterwegs" benutzbar sind. Die meisten dieser Computer werden, wie der abgebildete Olivetti M10, jetzt mit einem ROM-gespeicherten Textprogramm geliefert. Die Verarbeitungskapazität hält sich wegen des knappen Speicherraums, den die Batterieversorgung zuläßt, zwar sehr in Grenzen, aber für Briefe, Aktennotizen usw. sind derartige Maschinchen doch recht nützlich.

Gute Wiedergabe wichtig

Im kommerziellen Bereich haben sich Systeme wie die von IBM und Wang durchgesetzt. Sie bestehen aus einem Datensichtgerät, einer Tastatur mit Sondertasten für die Textverarbeitung, etwa einer „Paste"-Taste zum Einfügen von Passagen, und einem integrierten Speichermedium (Festplatte oder Diskette). Mit Ausnahme des „Joyce" von Schneider kosten diese Anlagen oft soviel wie ein Mittelklassewagen. Dafür bieten sie als Spezialausführungen eine Reihe von Annehmlichkeiten wie maßgefertigte Tastaturen, Vernetzungs-Schnittstellen und ergonomisch gestaltete Monitore. Bei der Textverarbeitung sitzt man oft stundenlang vor dem Bildschirm, so daß eine hinreichend große, scharfe und flimmerfreie Wiedergabe (mit mindestens 80 Zeichen pro Zeile) unerläßlich ist.

Bei den reinen Softwarepaketen gibt es enorme Unterschiede im Niveau. Manchmal wird als „Textverarbeitungssystem" bereits ein bildschirmorientierter Editor bezeichnet, wie er ohnehin meist zum Betriebssystem gehört. Damit lassen sich zwar notfalls mal ein paar kurze Abschnitte entwerfen und unmittelbar

ausdrucken, aber zu wesentlich mehr reicht es eigentlich nicht.

Eine Stufe höher stehen die „Bearbeitungsprogramme", die das Erstellen einer Textdatei und ihre Korrektur am Bildschirm erlauben. Sie kommen einem richtigen Textsystem schon näher; in einigen Fällen sind in dieser Kategorie bereits Suchroutinen zum Auffinden von Strings sowie Kommandos für das Kürzen und Einfügen von Blöcken (Cut/Paste) in einem Textfile vorgesehen. Ein typisches Beispiel dieser Art bietet die TEXT-Software bei den tragbaren Kyocera-Rechnern, die als „Olivetti M10" oder „Tandy Mod. 100" vertrieben werden. Sie umfaßt Cut-, Paste-, Kopier-, Such- und Druckroutinen. Die Textdateien sind in einem batterieversorgten CMOS-RAM mit geringem Leistungsbedarf gespeichert und bleiben auch bei abgeschaltetem Rechner erhalten.

RAM oder auf Diskette

Bei den Programmen, die die Bezeichnung Textverarbeitung wirklich verdienen, ist zwischen RAM- und Disketten-orientierten Systemen zu unterscheiden. Die Pakete auf RAM-Basis sind ein Zugeständnis an diejenigen Heimcomputerbesitzer, die keine Floppy haben und deshalb das Textprogramm selbst und – weil Cassettenrecorder zu langsam sind – auch die erstellten Texte vollständig im Arbeitsspeicher halten müssen. Die Textdatei wird dabei erst in der endgültigen Fassung auf Band überspielt. Die Leistungsfähigkeit der RAM-orientierten Systeme stößt aus naheliegenden Gründen an enge Grenzen. Das betrifft nicht nur den gesamten Textumfang, sondern oft auch die Länge der Passagen, die umgestellt werden können.

Die Programme auf RAM-Basis bieten trotzdem einen entscheidenden Vorteil: Sie arbeiten oftmals schneller als die diskettenorientierten, weil bei ihnen die Floppy-Zugriffszeiten entfallen. Ein gutes Beispiel für ein solches RAM-System ist „Tasword", das Texte bis zu 13 000 Zeichen (rund 2000 Wörter) zuläßt.

Bei den Diskettensystemen ist der Textumfang dagegen zumindest theoretisch nur durch die Speicherkapazität der Floppies begrenzt. Die Textdateien werden blockweise ins RAM geladen. Allerdings muß man mit längeren Wartezeiten rechnen, besonders beim Rücksprung von einem entfernten Textende auf den Anfang. Typische Diskettensysteme sind „WordStar" und „PerfectWriter". Diese Programme sind oft im Lieferumfang von Bürorechner-Paketen enthalten. Separat sind sie teuer, so daß sich die Anschaffung nur für professionelle Anwender lohnt.

Vorteilhaft bei den diskettenorientierten Programmen ist noch, daß sie ausgiebig das Betriebssystem einspannen, unter dem sie laufen (CP/M, MS-DOS usw.), und daher mehr Komfort beim Umgang mit Dateien als die Heimcomputerversionen bieten.

Der „Joyce" (PCW8256) ist das neueste Schneider-Produkt aus einer ganzen Palette preisgünstiger Microcomputer. Es handelt sich um ein spezielles Textverarbeitungssystem mit Z80-Prozessor und 256 KByte RAM, das komplett einschließlich Tastatur, Monitor mit integriertem Diskettenlaufwerk, Drucker und Software angeboten wird.

Angebot unter der Lupe

In den folgenden Artikeln soll eine Reihe einschlägiger Textverarbeitungsprogramme mit ihren Stärken und Schwächen besprochen werden. Zunächst geht es um WordStar – in diesem Bereich praktisch ein Standard, an dem alles andere gemessen wird. MacWrite wird als Beispiel für besonders elegante Textwiedergabe vorgestellt, und verschiedene Systeme auf RAM-Basis, wie „Tasword" und „Mini-Office", werden noch genauer unter die Lupe genommen.

Das „Microwriter"-Konzept ist für ein Textsystem höchst ungewöhnlich: Statt der üblichen Schreibmaschinentastatur gibt es nur sechs Knöpfe – vier für die Finger und zwei für den Daumen, mit denen durch Drücken in unterschiedlicher Kombination die Schriftzeichen einzutasten sind.

Die Vollendete

Den Abschluß unseres MIDI-Projekts bilden zusätzliche Tips und Tricks für die Programmentwicklung. Außerdem bringen wir ein Assembler-Listing für die Aufzeichnung, Speicherung und Wiedergabe von Musikstücken mit dem Acorn B.

Im letzten Abschnitt haben wir die Unterschiede zwischen analoger Speicherung von Musik auf Band und digitaler Aufzeichnung von MIDI-Daten kennengelernt. Dabei wurde auch gezeigt, wie mit der MIDI-Technik durch Pausen-Bytes Speicherplatz gespart wird.

Die dabei gewonnenen Erkenntnisse können wir bei der Programmierung für und mit MIDI häufig nutzen. Das Programm muß MIDI-Daten über die bereits entwickelte Schnittstelle in Echtzeit empfangen und senden können. Die Programme müssen so schnell arbeiten, daß jeder MIDI-Befehl vor dem Eintreffen des nächsten Befehls gespeichert bzw. verarbeitet ist.

Wegen der Echtzeit-Speicherung benötigen wir außerdem einen zuverlässigen Taktgeber im Computer, der die Aktivitäten des Programms bei Aufzeichnung und Wiedergabe synchronisiert und den Wert der im Speicher festzuhaltenden Pausenbytes (Zeit zwischen dem Eingang zweier aufeinander folgender Befehle) korrekt ermittelt.

Die im Commodore 64 und Acorn B verwendeten Ein/Ausgabechips sind ähnlich aufgebaut und besitzen beide ein 16-Bit-Timer-Register. Es kann so programmiert werden, daß es eine vorgegebene Zahl auf Null herunterzählt. Beim Erreichen der Null wird ein Interrupt-Flag gesetzt und der Timer wieder auf den Anfangswert gestellt. Diese Betriebsart nennt sich „free-run-modus" (Freilaufbetrieb).

Vom Programm wird der Timer folgendermaßen genutzt: Die Routine „check" prüft, ob der Zähler bereits auf Null ist, indem sie das IRQ-Flagregister des CIA/VIA-Chips abfragt und entsprechend das Z-Flag im Statusregister des Prozessor setzt. Die Teile des Programms, die zur Aufnahme und Wiedergabe dienen, rufen das Unterprogramm „check" auf und verwenden die Befehle BNE oder BEQ beim Verlassen des Unterprogramms, wobei sie gemäß dem Zustand des Z-Flags verzweigen.

Die check-Routine fragt auch die Tastatur ab, um zu prüfen, ob die Leertaste (zum Stoppen) gedrückt wurde. Beim Commodore 64 wird diese Abfrage unabhängig von der normalen, IRQ-gesteuerten Tastaturabfrage durchgeführt, wobei einfach die letzte Tastenreihe geprüft wird. Der Acorn fragt die Tastatur mit OSBYTE call &79 ab. Wenn das „check"-Unterprogramm zumindest einmal zwischen zwei aufeinander folgenden Zählerläufen aufgerufen wird, ist das ein recht sicheres Verfahren zur Aufzeichnung der MIDI-Befehle.

Es ist nicht notwendig, jeden der mit MIDI übermittelten Befehle zu speichern. Da das Aufnahme-/Wiedergabeprogramm eigene Taktimpulse erzeugt, sind die Echtzeitbefehle überflüssig und sollten unterdrückt werden. (Die meisten Tastaturinstrumente übertragen diese Meldungen nicht, wenn sie nicht über einen eigenständigen Taktgeber verfügen.)

Versehentlich übertragene allgemeine (System Common) und systemspezifische (System exklusive) Befehle müssen ignoriert werden. Diese Befehle können aber aus einer beliebigen Anzahl von Bytes bestehen. Daher muß ein Flag gesetzt werden, das den Empfänger dazu bringt, alle Daten bis zum Auftauchen eines Kanal-Statusbytes zu ignorieren. Das wird durch den Wert $FF im X-Register angezeigt. Die anderen in X gespeicherten Werte dienen als Zähler für die Menge der Datenbytes, die zur Komplettierung des aktuellen Channel-Befehls noch ausstehen.

Erweiterungsmöglichkeiten

Unser relativ unkompliziertes Programm soll ein kleiner Vorgeschmack auf Möglichkeiten und Probleme der MIDI-Datenverarbeitung in Echtzeit sein. Für den Ausbau zu Ihrem persönlichen MIDI-Programm stehen Ihnen noch viele Möglichkeiten offen:

1. Wird die Wiedergabe mitten im Stück angehalten, ertönen einige Noten weiter, da der erforderliche „Note Aus"-Befehl nicht gegeben wurde.
2. Für die Synchronisation von Geräten (etwa zwischen Cassettenrecorder und Rhythmusmaschine) sind keine Maßnahmen getroffen.
3. Eine Bearbeitung des Musikstückes ist nicht möglich.
4. Das Tempo von Aufzeichnung und Wiedergabe ist nicht variabel.
5. Es gibt keine Misch- oder Einfügungsmöglichkeiten.
6. Gleichzeitiges Abhören und Aufzeichnen eines Musikstückes ist nicht möglich.
7. Beim Herausziehen des MIDI-Kabels während einer Übertragung stürzt das Programm ab.
8. Es sind noch keine Vorkehrungen für das Speichern eines Musikstückes auf Band oder Diskette vorgesehen.

Aufzeichnungs/ Wiedergabeprogramm

```
  5 REM **   BBC MIDI PROJECT **
  6 REM ** DIGITAL RECORDING  **
  7 REM ** AND PLAYBACK       **
  8 *FX 14,6
 10 DIM start &2000
 20 FOR opt%=0 TO 3 STEP 3
 30 timer=&FE60
 40 osasci=&FFE3
 50 osrdch=&FFE0
 60  osbyte=&FFF4
 70 streg=&FEE0
 80 datreg=streg+1
 90 mem=&70
100 ptr=&72
110 P%=start
120 [
130  OPT opt%
140 .begin
150 JMP start1
160  .nobytes BRK
170  .clocks BRK
180  .freemem BRK
190  BRK
200 .start1
210 LDA #3
220 STA streg
230 LDA #&16
240 STA streg
250 .g10
260 LDA #&FF
270 STA lowmem
280 LDA #&40
290 STA timer+11
300 LDA #0
310 STA timer+4
320 LDA #&08
330 STA timer+5
340 .g20
350 LDA #0
360 JSR strout
370 .g21
380 JSR osrdch
390 CMP #ASC"E"
400 BNE g22
410 RTS
420 .g22
430  CMP #ASC"R"
440 BEQ g30
450 CMP #ASC"P"
460 BNE g21
470 .g30
480 PHA
490 LDA #lowmem MOD 256
500 STA mem
510 LDA #lowmem DIV 256
520 STA mem+1
530 LDA #frbytes MOD 256
540 STA freemem
550 LDA #frbytes DIV 256
560 STA freemem+1
570 SEI
580 LDY #0
590 PLA
600 CMP #ASC"R"
610 PHP
620  BEQ r00
630 .p00
640 LDA #2
650 JSR strout
660  .p05
670 JSR read
680 CMP #&FF
690 BNE p07
700 PLP
710 JMP c20
720 .p07
730 STA clocks
740 CMP #&F0
750 LDA clocks
760 BEQ p30
770 .p10
780 JSR check
790 BEQ p10
800 DEC clocks
810 BNE p10
820 BCS p05
830 .p30
840 JSR read
850 PHA
860 .p35
870 LDA streg
880 AND #2
890 BEQ p35
900 PLA
910 STA datreg
920 BPL p50
930 JSR getno
940 .p50
950 DEX
960 BNE p30
970 LDX nobytes
980 BPL p05
990 .r00
1000 LDA #1
1010 JSR strout
1020 STY clocks
1030 .r05
1040 LDX #&FF
1050 .r10
1060 JSR check
1070 BEQ r20
1080 INC clocks
1090 LDA clocks
1100 CMP #&F0
1110 BCC r20
1120 JSR store
1130 STY clocks
1140 .r20
1150 LDA streg
1160 AND #1
1170 BNE r40
1180 CPX #1
1190 BMI r10
1200 BPL r20
1210 .r40
1220 LDA datreg
1230 BMI r50
1240 CPX #0
1250 BMI r10
1260 BNE r80
1270 LDX nobytes
1280 BPL r60
1290 .r50
1300  CMP #&F8
1310  BCS r20
1320  CMP #&F0
1330 BCS r05
1340 JSR getno
1350 .r60
1360 PHA
1370 LDA clocks
1380 JSR store
1390 STY clocks
1400 PLA
1410 .r80
1420 JSR store
1430 DEX
1440 BEQ r10
1450 BNE r20
1460 .check
1470 STX &74
1480 PHP
1490 LDA #&79
1500 LDX #&E2
1510 JSR osbyte
1520 LDY #0
1530 PLP
1540 TXA
1550 BPL c40
1560 PLA
1570 PLA
1580 PLP
1590 BNE c20
1600 LDA #&FF
1610 STA (mem),Y
1620  .c20
1630 CLI
1640 JMP g20
1650 .c40
1660 LDX &74
1670 LDA timer+13
1680 AND #&40
1690 STA timer+13
1700 RTS
1710 .store
1720 STA (mem),Y
1730 INC freemem
1740 BNE point
1750 INC freemem+1
1760 BNE point
1770 PLA
1780 PLA
1790 PLA
1800 LDA #3
1810 JSR strout
1820 JMP c20
1830 .read
1840 LDA (mem),Y
1850 .point
1860 INC mem
1870 BNE p20
1880 INC mem+1
1890  .p20 RTS
1900 .getno
1910 LDX #2
1920 CMP #&C0
1930 BCC n10
1940 CMP #&E0
1950 BCS n10
1960 DEX
1970 .n10
1980 STX nobytes
1990 INX
2000 RTS
2010 .strout
2020 ASL A
2030 TAX
2040 LDA messtab,X
2050 STA ptr
2060 LDA messtab+1,X
2070 STA ptr+1
2080 LDY #0
2090 .m10
2100 LDA (ptr),Y
2110 JSR osasci
2120 INY
2130 CMP #&0D
2140 BNE m10
2150 LDY #0
2160 .m70 RTS
2170 ]
2180 mess0=P%
2190 $P%="R = record,P = play,E = exit"
2200 P%=P%+LEN($P%)+1
2210 mess1=P%
2220 $P%="recording"
2230 P%=P%+LEN($P%)+1
2240 mess2=P%
2250 $P%="playback"
2260 P%=P%+LEN($P%)+1
2270 mess3=P%
2280 $P%="out of memory"
2290 P%=P%+LEN($P%)+1
2300 lowmem=P%+8
2310 frbytes=&E000+(lowmem-start)
2320 [
2330 .messtab
2340 ]
2350  NEXT
2360 ?P%=mess0 MOD 256
2370 P%?1=mess0 DIV 256
2380 P%?2=mess1 MOD 256
2390 P%?3=mess1 DIV 256
2400 P%?4=mess2 MOD 256
2410 P%?5=mess2 DIV 256
2420 P%?6=mess3 MOD 256
2430 P%?7=mess3 DIV 256
2440 CALL begin
```

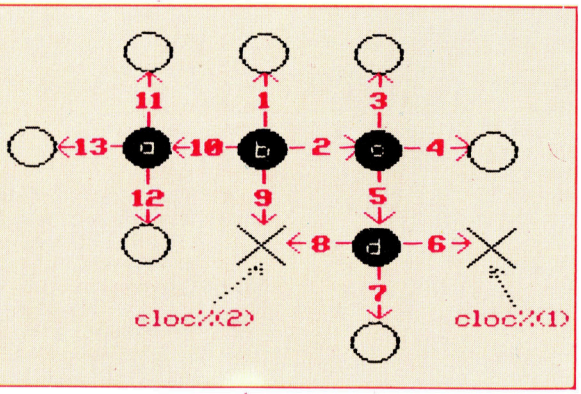
Steinzeit

Für einen Spielzug in Go gibt es weit mehr Möglichkeiten als etwa bei Schach. Wir programmieren das erste einer Reihe von Modulen, mit denen der Computer Entscheidungen fällen kann.

Wir beginnen die Programmierung „Künstlicher Intelligenz", durch die der Computer korrekte und taktisch kluge Züge „erlernt". Die erste Routine „Computer-Zug" ermöglicht dem Computer die Entscheidung, ob eine bestimmte Gruppe von Steinen gefährdet ist. Danach erkennt der Computer, mit welchem Zug er die Gruppe retten bzw. (bei einer gegnerischen Gruppe) einnehmen kann.

Verdeutlichen wir uns zunächst, wie andere Programme Spielzüge bestimmen. Bei Schach beispielsweise stehen bei jeder Brettposition durchschnittlich etwa 30 mögliche Spielzüge zur Wahl. Darauf basierend muß ein Computer jeden möglichen Zug prüfen und entscheiden, welcher davon der beste ist. Möglichst sollten die darauffolgenden gegnerischen Züge gleich mitgeprüft werden. Nehmen wir an, der Computer prüft die eigenen 30 Züge und 30 Antworten des Gegners, so müssen bis zu 900 (30x30) mögliche Positionen abgefragt werden. Gehen wir noch einen Schritt weiter, dann sollte der Computer seinen optimalen Gegenzug auf den Zug des Gegners ebenfalls berechnen. Somit wären jetzt bereits 27 000 Positionen zu ermitteln, in der nächsten Stufe 810 000 Positionen usw. Dieses „Vorausdenken" kann grafisch über die Verzweigung eines „Spielbaumes" dargestellt werden, bei dem an jedem Knotenpunkt die jeweils möglichen Züge abzweigen. Diese riesigen Zahlenmengen lassen sich über wirkungsvolle Techniken wie den „alpha-beta Algorithmus" erheblich reduzieren und von einem Computer mit hoher Verarbeitungsgeschwindigkeit handhaben.

Die durchschnittliche Anzahl möglicher



Kritische Lage
Die Gruppen-Berechnungsroutine sieht eine Gruppe als gefährdet an, wenn sie weniger als drei Freifelder hat. Das Zählen beginnt in unserem Beispiel bei Stein b. Zwei Array-Elemente, cloc%(1) und cloc%(2), dienen zum Speichern der Positionen von Freifeldern. Der Computer setzt dann bei Gefahr dorthin seinen Stein.

Viertes Modul

Acorn B:

```
   80 move%=move%+1:PROCblack_move
  240 DIM capture%(2),cloc%(2),tloc%(2)
 1360 atari1$="      ":atari2$="      ":T$
="***"
 2530 :
 2540 DEF PROCblack_move
 2550 atari1$="      "
 2560 location%=0
 2570 PROCgroup_evaluation:T$="GP "
 2630 IF location%=0 THEN ENDPROC
 2640 PROCmake_move(location%,black%)
 2650 PROCmessage(23,6,"")
 2660 ENDPROC
 2670 :
 2680 REM*****************************
 2690 :
 2700 DEF PROCgroup_evaluation
 2710 LOCAL C%,L%,P%,Q%,S%,hi,score
 2720 FOR P%=17 TO 255
 2730   C%=board%?P% AND colour%
 2740   IF C%=0 THEN 2850
 2750   PROCcount(P%,C%) : IF clib%>2 TH
EN 2850
 2760   tloc%(1)=cloc%(1)
 2770   tloc%(2)=cloc%(2)
 2780   L%=clib% : S%=cstn%
 2790   FOR Q%=1 TO L%
 2800     IF FNlegality(tloc%(Q%),black%
)<>0 THEN 2840
 2810     IF L%=2 AND clib%<3 THEN 2840
 2820     score=(8*S%/L%-clib%+2*L%)
 2830     IF score>hi THEN hi=score:loca
tion%=tloc%(Q%)
 2840     NEXT
 2850   NEXT
 2860 ENDPROC
 2870 :
 2880 REM*****************************
 4060 cloc%(1)=0 : cloc%(2)=0
 4280 IF clib%<3 THEN cloc%(clib%)=P%
```

Schneider CPC 464/664:

```
80 mve%=mve%+1:GOSUB 2540
240 DIM capture%(2),cloc%(2),tloc%(2)
1360 atari1$="      ":atari2$="      ":T$=
"***"
2530 :
2540 REM **** black move routine ****
2550 atari1$="      "
2560 location%=0
2570 GOSUB 2700:T$="GP ":REM group evalu
ate
2630 IF location%=0 THEN RETURN
2640 mmp%=location%:mmc%=black%:GOSUB 36
30:REM make move
2650 mp%=25:mm%=6:o$="":GOSUB 2160:REM m
essage
2660 RETURN
2670 :
2680 REM *************************
2690 :
2700 REM group evaluation routine
2710 hi%=-9999
2720 FOR p%=17 TO 255
2730 c%=PEEK(board+p%) AND colour%
2740 IF c%=0 THEN 2850
2750 cp%=p%:cc%=c%:GOSUB 4040:IF clib%>2
 THEN 2850
2760 tloc%(1)=cloc%(1)
2770 tloc%(2)=cloc%(2)
2780 gl%=clib%:gs%=cstn%
2790 FOR q%=1 TO gl%
2800 lp%=tloc%(q%):lc%=black%:GOSUB 3890
:IF ll%<>0 THEN 2840
2810 IF gl%=2 AND clib%<3 THEN 2840
2820 score=(8*gs%/gl%-clib%+2*gl%)
2830 IF score>hi THEN hi=score:location%
=tloc%(q%)
2840 NEXT q%
2850 NEXT p%
2860 RETURN
2870 :
2880 REM **********************
4060 cloc%(1)=0:cloc%(2)=0
4280 IF clib%<3 THEN cloc%(clib%)=sp%
```

Commodore 64:

```
80 MOVE%=MOVE%+1:GOSUB 2540
240 DIM CAPTURE%(2),CLOC%(2),TLOC%(2)
1360 A1$="      ":A2$="      ":T$="***"
2530 :
2540 REM BLACK-MOVE ROUTINE
2550 A1$="      "
2560 LOCAT%=0
2570 GOSUB 2700:T$="GP "
2630 IF LOCAT%=0 THEN RETURN
2640 MP%=LOCAT%:MC%=BLACK%:GOSUB 3630
2650 MP%=24:MM%=6:O$="":GOSUB 2160
2660 RETURN
2670 :
2680 REM*********************************
2690 :
2700 REM GROUP-EVALUATION ROUTINE
2710 HI=-9999
2720 FOR P=17 TO 255
2730 C%=PEEK(BOARD+P) AND COLOUR%
2740 IF C%=0 GOTO 2850
2750 CP%=P:CC%=C%:GOSUB 4040:IF CLIB%>2
GOTO 2850
2760 TLOC%(1)=CLOC%(1)
2770 TLOC%(2)=CLOC%(2)
2780 BL=CLIB%:BS=CSTN%
2790 FOR Q=1 TO BL
2800 LP%=TLOC%(Q):LC%=BLACK%:GOSUB 3890:
IF LL%<>0 GOTO 2840
2810 IF BL=2 AND CLIB%<3 GOTO 2840
2820 SCR=(8*BS/BL-CLIB%+2*BL)
2830 IF SCR>HI THEN HI=SCR:LOCAT%=TLOC%(
Q)
2840 NEXT
2850 NEXT
2860 RETURN
2870 :
2880 REM*********************************
4060 CLOC%(1)=0:CLOC%(2)=0
4280 IF CLIB%<3 THEN CLOC%(CLIB%)=SP%
```

Sinclair Spectrum:

```
  80 LET move=move+1: GO SUB 2540
 240 DIM c(2): DIM i(2): DIM j(2)
1360 LET x$="      ": LET y$="
  ": LET t$="***"
2530:
2540 REM black-move routine
2550 LET x$="      "
2560 LET location=0
2570 GO SUB 2700: LET t$="GP "
2630 IF location=0 THEN  RETURN
2640 LET mmp=location: LET mmc=b
lack: GO SUB 3630
2650 LET mp=21: LET mm=7: LET o$
="": GO SUB 2160
2660 RETURN
2670:
2680 REM **********************
2690:
2700 REM group-evaluation routine
2710 LET hi=-9999
2720 FOR p=17 TO 255
2730 LET c=PEEK (board+p)
2735 IF c>3 THEN  LET c=c-4: GO
TO 2735
2740 IF c=0 THEN  GO TO 2850
2750 LET cp=p: LET cc=c: GO SUB
4040: IF clib>2 THEN  GO TO 2850
2760 LET j(1)=i(1)
2770 LET j(2)=i(2)
2780 LET gl=clib: LET gs=cstn
2790 FOR q=1 TO gl
2800 LET lp=j(q): LET lc=black:
GO SUB 3890: IF ll<>0 THEN  GO T
O 2840
2810 IF gl=2 AND clib<3 THEN  GO
 TO 2840
2820 LET score=(8*gs/gl-clib+2*gl)
2830 IF score>hi THEN  LET hi=sc
ore: LET location=j(q)
2840 NEXT q
2850 NEXT p
2860 RETURN
2870:
2880 REM **********************
4060 LET i(1)=0: LET i(2)=0
4280 IF clib<3 THEN  LET i(clib)
=sp
```

Spielzüge pro Position (bei einem Brett mit 19 auf 19 Schnittpunkten) beträgt bei Go etwa 200, in unserem Programm (mit einem 15/15-Brett) etwa 125. Wenn wir unsere normale „Vorausdenk"-Technik anwenden, ergibt die Exponentialsteigerung 15 625 Gegenzugmöglichkeiten, 1 953 125 mögliche Positionen für den nächsten Computerzug, 244 140 625 Varianten für den nächsten Spielzug usw. Unabhängig von der Rechengeschwindigkeit eines Computers ist es möglich, diese Anzahl möglicher Positionen in einem vertretbaren Zeitraum vollständig durchzurechnen und prüfen zu lassen. Außerdem bereitet es einem gezielt denkenden Meisterspieler keine Probleme, selbst 20 oder 30 Bewegungen im voraus zu planen, um die Auswirkungen bestimmter Züge vorhersehen zu können.

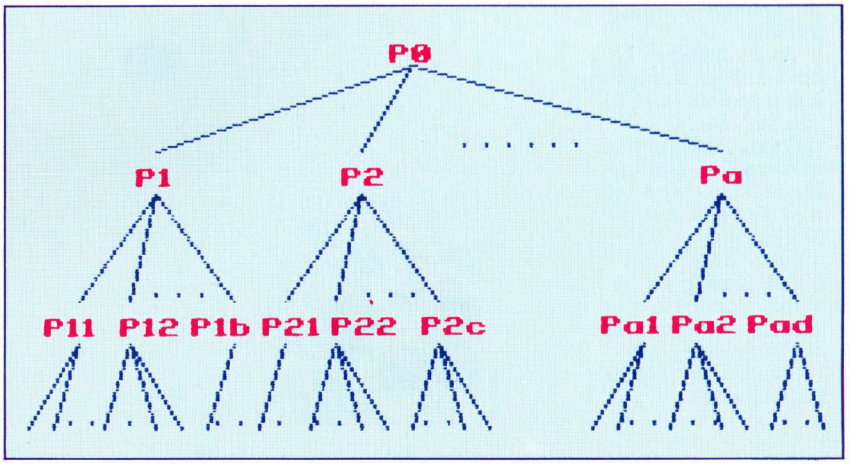
Sechs Berechnungsroutinen

Eine alternative Strategie besteht darin, eine Reihe von Routinen zu entwickeln, die die aktuelle Brett-Formation überprüfen, in der Hoffnung, einige gute Züge zu erkennen. Wir werden sechs Berechnungsroutinen erstellen, wobei die Berechnung direkt nach dem Auffinden eines mögliches Zuges angehalten wird. Logischerweise müssen die Routinen in korrekter Reihenfolge aufgerufen werden. Die Variable location% (anfangs 0) wird auf die entsprechende Brettposition gesetzt, sobald eine Routine einen möglichen Zug findet. Unsere angesprochene Hauptroutine black_move sieht dann wie folgt aus:

```
LET location%=0
CALL
erste Berechnungsroutine
IF location%=0 THEN CALL
zweite Berechnungsroutine
IF location%=0 THEN CALL
dritte Berechnungsroutine

IF location%=0 THEN CALL
Nte Berechnungsroutine
IF location%=0 THEN
keine Züge mehr auffindbar
```

Viele Strategiespiele, wie etwa Schach, arbeiten mit „Bäumen" zur Berechnung möglicher Züge und Gegenzüge für jede Figuren-Formation. Bei Go würde das bedeuten, daß für jeden Spielzug 200 neue Verästelungen für mögliche Gegenzüge berechnet werden müßten, für den übernächsten Zug bereits 40 000 Möglichkeiten. Wenn wir diese Steigerungsrate mit der eines Schachprogramms vergleichen, verstehen wir, was die Programmierung eines anspruchsvolleren Go-Spieles so kompliziert macht.



PROCblack_move ruft eine Berechnungsroutine namens PROCgroup_evaluation auf. Wird kein Zug gefunden (location% bleibt 0), wird die Routine beendet.

„Gruppen-Berechnungsroutine"

Daher kann nicht erwartet werden, daß das Programm gut spielt, obwohl der Computer eine angegriffene Gruppe bereits verteidigen kann. Denn die „Gruppen-Berechnungsroutine" hat immer höchste Priorität innerhalb von PROCblack_move und wird daher immer zuerst geprüft. Darin werden alle Gruppen auf dem Brett geprüft und deren Freifelder gezählt. Hat eine Gruppe nur ein oder zwei Freifelder, wird die Lage als kritisch erkannt (ein Zug vor „Atari"). Daher wird der Computer diese Gruppe schützen oder die gegnerische angreifen. Zuerst wird die Anzahl von Freifeldern jeder Gruppe gezählt. Bei nur einem oder zwei Freifeldern werden diese geschützt. Dabei ist unerheblich, ob es sich um eine eigene oder eine fremde Gruppe handelt, da beim Setzen eines Steines eine fremde Gruppe weiter eingeschlossen bzw. eine eigene Gruppe erweitert und damit geschützt wird.

Es wurde bereits erwähnt, daß die Routine PROCcount nicht nur die Steine einer Gruppe, sondern auch die Anzahl der Freifelder errechnet. Durch Anfügen von Zeile 4060 an PROCcount und Zeile 4280 an PROCsearch wird die Freifelder-Anzahl im Array clock%(2) abgelegt.

Jetzt kann PROCgroup_evaluation definiert werden. Die Schleife P% prüft jede Position auf dem Brett. Wurde eine Gruppe gefunden, werden deren Freifelder gezählt. Sind es weniger als drei, ist die Lage kritisch, und die Schleife 0% wird aufgerufen. Sie prüft, ob das Setzen auf eines der Freifelder erlaubt ist und ordnet einem korrekten Zug einen Wert zu. Diese Wertungs-Funktion ist noch nicht optimal — vielleicht experimentieren Sie selbst ein wenig. Beachten Sie, daß als Nebeneffekt beim Aufruf von FNlegality jetzt clib% und cstn% die Anzahl der Freifelder und Steine einer Gruppe enthalten, als wäre der Zug bereits ausgeführt. Die Ursprungswerte von clib% und cstn% wurden in L% und S% abgelegt.

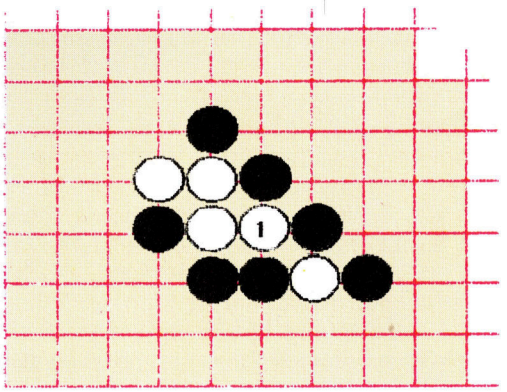
PROCgroup_evaluation

Zur Vereinfachung prüft PROCgroup_evaluation jeden Stein am Brett, was bedeutet, daß Steine einer Gruppe mehrmals gezählt werden. Diese Methode könnte verbessert werden, wenn sichergestellt ist, daß die Merker (die während PROCsearch als Anzeige für einen gezählten Stein gesetzt werden) am Ende von PROCcount nicht gelöscht werden. Freifeld-Merker müsssen aber nach jeder Suche gelöscht werden, da mehrere Gruppen gleiche Freifelder haben können. Auch muß sicher sein, daß die Merker nur dann stehen bleiben, wenn die count-Routine aus PROCgroup_evaluation aufgerufen wird. Die Positionen aller Freifelder und zugeordnete Wertungen müßten gespeichert und die Akzeptanz jedes Zuges müßte nach dem Löschen der Merker geprüft werden.

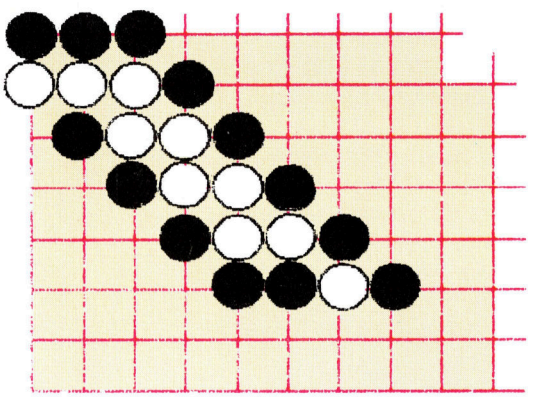
Im nächsten Kapitel wird eine „Fang"-Routine angefügt, die logische Züge bietet, wenn gerade keine Gruppe gefährdet ist. Damit wird der Computer in seinem Aktivpotential gestärkt und zu einem ernsten Gegner.

SHICHO

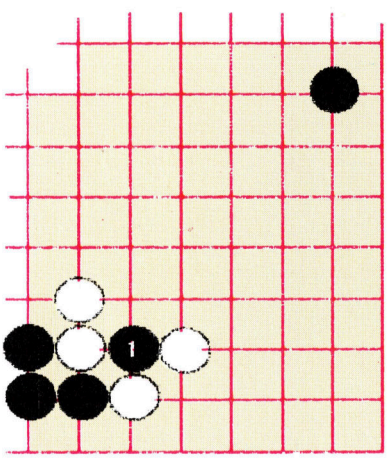
Gefährliche Diagonalen
Der weiße Stein 1 wird von Schwarz angegriffen. Anstatt den Stein zu opfern, versucht Weiß, die Diagonale nach oben links zu verlängern, um einen Verlust zu verhindern.



Spielbrettgrenzen
Weiß hat mit seinem Fluchtversuch einen schweren Fehler begangen. Sobald die weiße Gruppe den oberen Brettrand erreicht, kann sie von Schwarz gefangen werden. Statt zu fliehen, hätte Weiß gleich beim ersten Angriff seinen Stein aufgeben sollen. Die Diagonal-Muster, die auf der Flucht entstehen, werden SHICHO genannt.

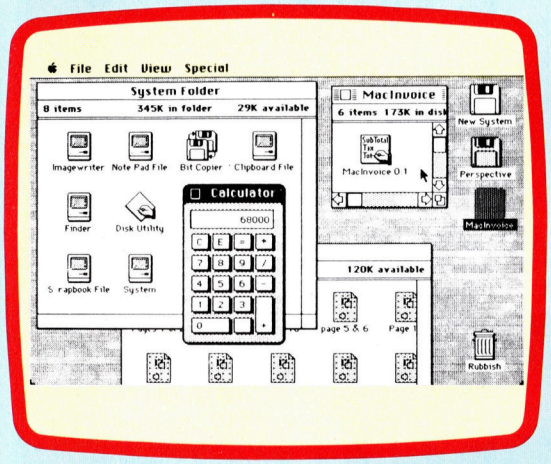


Durchbrechen von SHICHO
In dieser Situation könnte Weiß versuchen, den schwarzen Stein 1 einzunehmen, in der Hoffnung, daß Schwarz flieht und am oberen Brettrand gestoppt wird. Dabei würde ein Shicho geformt. Da Schwarz diagonal nach rechts oben flieht, könnte der obere schwarze Stein die Shicho-Formation durchbrechen, womit Schwarz gerettet wäre.



Fachwörter von A bis Z

Object Code = Objektprogramm
Aus dem Quellprogramm, das in einer höheren Programmiersprache geschrieben wurde, erzeugt ein Compiler ein Objektprogramm (in maschinennaher Sprache). Es steht nach dem Compilieren als Maschinenprogramm zur Verfügung und kann mit einem RUN-Befehl gestartet werden. Der Vorteil beim Erstellen eines solchen Objektprogramms liegt darin, daß es wesentlich schneller läuft als etwa ein BASIC-Programm, das während der Ausführung Zeile für Zeile vom Interpreter in die Maschinensprache übersetzt wird. Noch effizienter als vom Compiler erzeugte Objektprogramme arbeiten allerdings Programme, die unmittelbar im Maschinencode verfaßt sind.



Der Apple Macintosh war einer der ersten Rechner, bei dem die objektorientierte Programmierung in die Praxis umgesetzt wurde. Statt eine Serie von Befehlen einzugeben, braucht der Benutzer hier nur den Cursor mit der Maus auf das betreffende Piktogramm zu führen und die Kopftaste der Maus zu drücken. Der Rechner erledigt die gestellte Aufgabe dann ohne weiteres Zutun allein.

Object-Oriented = Objektorientiert
Normalerweise werden beim Programmieren alle Verarbeitungsschritte in Gestalt arithmetischer Anweisungen, Funktionen und Prozeduren vorgeschrieben. Bei der objektorientierten Programmierung wird der Ablauf dagegen nicht im Detail fixiert, sondern es wird in Form einer „Message" (Mitteilung) nur das zu lösende Problem spezifiziert. Der Rechner wählt den Lösungsweg.

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.

Octal Notation = Oktalschreibweise
Im Oktalsystem dient die Zahl Acht als Basis. Es gibt darin also nur die Ziffern Null bis Sieben. Jede Oktalstelle faßt deshalb gerade eine dreistellige Binärzahl. Das Oktalsystem war sehr praktisch für die alten „Stellenmaschinen", die nicht mit Bytes zu acht Bit, sondern mit „Stellen" zu sechs Bit als kleinster adressierbarer Einheit arbeiteten, weil jede „Stelle" als zweiziffrige Oktalzahl (Höchstwert 77≙ dez. 63) darzustellen war. Mit dem Einzug der Bytemaschinen wurde das Oktalsystem durch das Hexadezimalsystem mit der Basis 16 verdrängt.

One's Complement = Einer-Komplement
Das Einer-Komplement ist eine Zwischenstufe bei der Bildung eines Zweier-Komplements, das für die Binär-Arithmetik benötigt wird. Das binäre Einer-Komplement ergibt sich einfach durch Vertauschen von Nullen und Einsen, also durch „Invertieren" (oder NOT-Verknüpfung) einer Zahl; aus 0001 0011 (dezimal 19) wird damit 1110 1100. Wenn der Computer dieses Bitmuster wie üblich als ganze Zahl mit Vorzeichen und daher die „1" an erster Stelle als „Minus" liest, interpretiert er es als -19. Durch Erhöhen um Eins entsteht aus dem Einer- das Zweier-Komplement, dessen Addition bei Digitalrechnern die Subtraktion ersetzt.

Operand = Operand
Als Operanden werden speziell bei der Maschinencode- und Assemblerprogrammierung Größen bezeichnet, auf die „Operationen" wie Addition oder Subtraktion angewandt werden. In der CPU müssen die Operanden zur Ausführung arithmetisch/logischer Befehle in den vorgesehenen Registern bereitstehen.

Operating System = Betriebssystem
Das Betriebssystem ist eine Art Software-Schnittstelle zwischen Benutzer und Rechnerelektronik. Es gibt eine Vielzahl unterschiedlicher Systeme, die sich aber formal leicht zwei Kategorien zuordnen lassen. Die eine umfaßt die ROM-gespeicherten Systeme, die vor allem bei Heimcomputern häufig eingebaut sind.

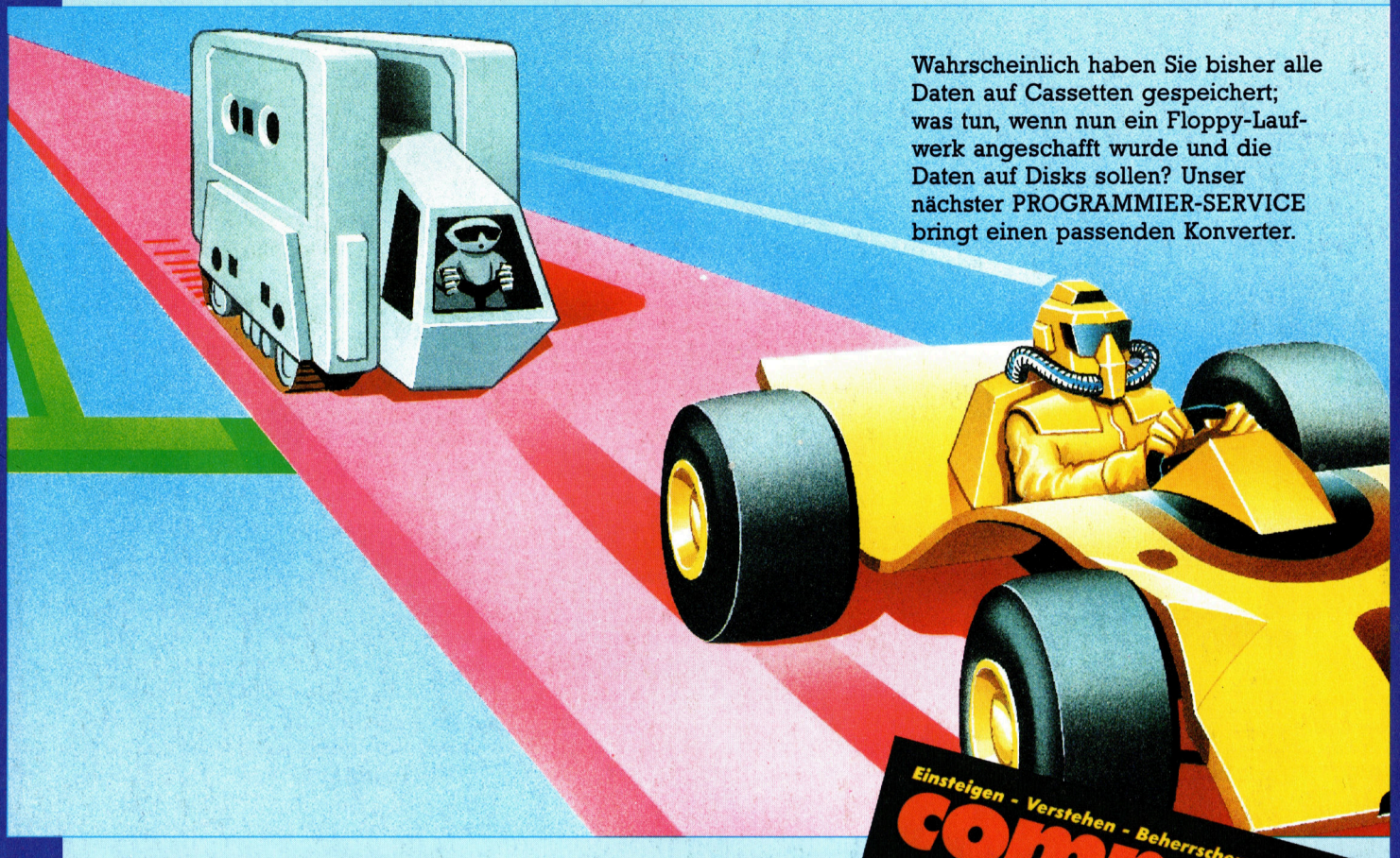
Die zweite Gruppe bilden Betriebssysteme, die nach dem Einschalten des Rechners erst von einem externen Speicher geladen werden müssen, wie etwa CP/M und MS-DOS. Dabei handelt es sich meistens um diskettenorientierte Systeme. Ihr Vorteil besteht insbesondere darin, daß sie ein hohes Maß an Kompatibilität zwischen hardwaremäßig vergleichbaren Computern gestatten. Bei Maschinen, die mit dem gleichen Betriebssystem arbeiten, sind im Prinzip auch die dafür geschriebenen Softwarepakete von einem zum andern Rechner übertragbar oder „portabel".

Operation = Operation
Durch eine mathematische Operation wird aus einer oder mehreren Ausgangsgrößen ein neues Ergebnis gewonnen. Die Symbole +, − und * beispielsweise repräsentieren arithmetische Operationen. Daneben spielen in der Datenverarbeitung aber noch weitere eine Rolle, etwa binäre Operationen, die letztlich alle auf der simplen Addition zweier Binärzahlen beruhen. Häufig benutzt werden auch Boolesche Operationen, bei denen Operanden durch logische Funktionen verknüpft werden.

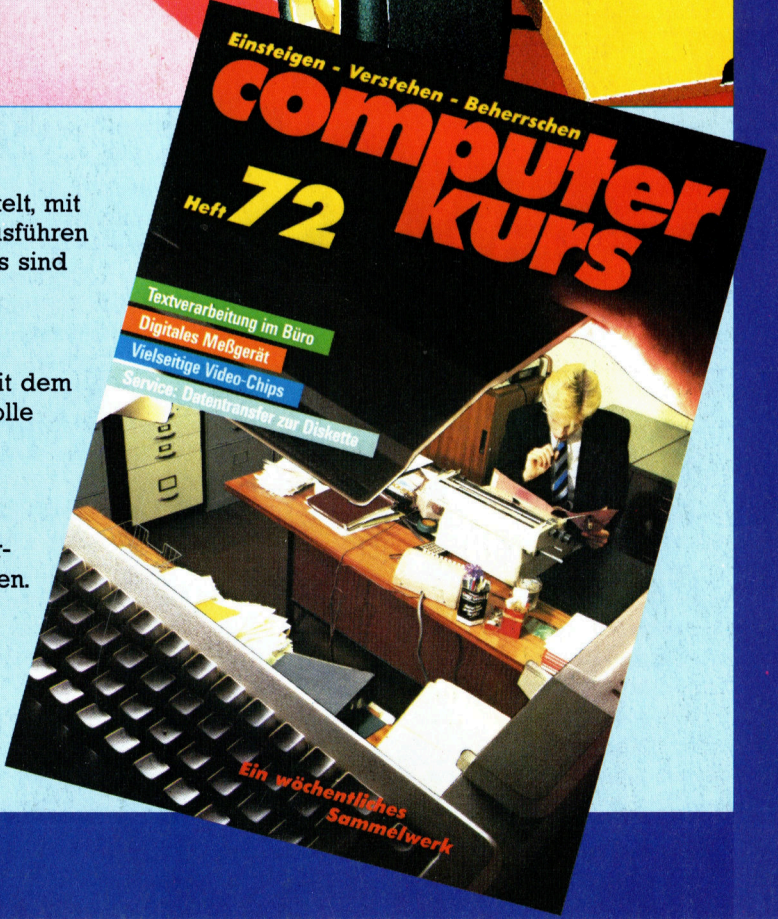
Bildnachweise
1961, 1966: Kevin Jones
1967, 1972: Mike Clowes
1970: Marcus Wilson-Smith
1974, 1982: Chris Stevens
1988: Caroline Clayton

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

computer kurs Heft 72



Wahrscheinlich haben Sie bisher alle Daten auf Cassetten gespeichert; was tun, wenn nun ein Floppy-Laufwerk angeschafft wurde und die Daten auf Disks sollen? Unser nächster PROGRAMMIER-SERVICE bringt einen passenden Konverter.



Zug um Zug

Im Go-Spiel werden nun Routinen entwickelt, mit denen der Computer strategische Züge ausführen kann. Durch die Besonderheiten des Spiels sind etwas aufwendigere Prozeduren nötig.

Digitalmessung

Der Selbstbau-Kurs befaßt sich diesmal mit dem Bau eines digitalen Meßgerätes zur Kontrolle von Widerständen und Spannungen.

Hinter der Maske

Die Z80-CPU bietet drei maskierbare Interrupts an, die Prozessorzeit „borgen" können.

Menü-Vorschläge

Eine gute Textverarbeitung erleichtert Büroarbeit immens. Wir stellen eines der bewährtesten Programme vor – „WordStar" von MicroPro.