Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 70

Das ECHO-Musikpaket

Dynamische RAMs

Service: Programmschutz

Bytes im MIDI-System

Ein wöchentliches Sammelwerk

computer kurs

Heft 70

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.

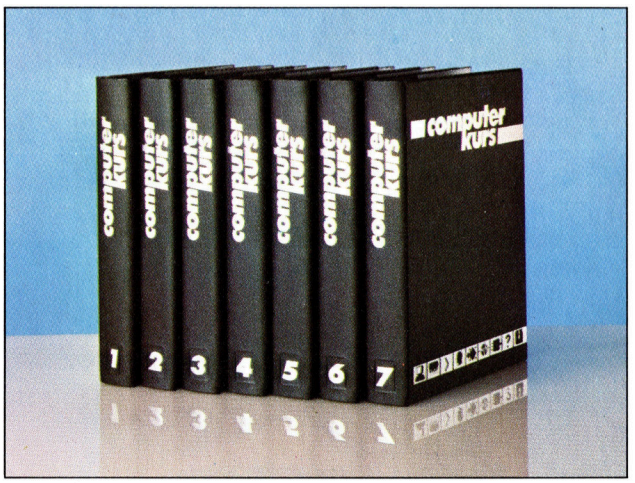
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

Starkes Echo

Für den Commodore 64 und den Acorn B wird das Musik-Paket „Echo“ angeboten, das den Soundchip im Rechner über eine Klaviertastatur anspricht.

Das „Echo“-Paket der Firma Leasalink Viewdata Ltd. (LVL) besteht aus einer Klaviatur mit mehr als drei Oktaven, der zugehörigen Software, einem Handbuch und einem zwanzigadrigen Kabel für den Anschluß an den User Port. Der Kabelstecker paßt aber nur beim Acorn B unmittelbar; für den Commodore 64 wird ein Adapter mitgeliefert.

Die Echo-Hardware ist nicht besonders aufwendig – die Funktionen werden im wesentlichen softwaremäßig realisiert, weil sich so die Herstellungskosten deutlich niedriger halten lassen.

Es sind zwei verschiedene Betriebsarten wählbar, die als „Orgelmodus“ bzw. „Synthesizermodus“ bezeichnet werden. Im Orgel-Modus lassen sich über die einzelnen Tasten der Rechnertastatur diverse Instrumentalregister von Hawaii-Gitarre bis zu Cello und Harpsichord anwählen. Auf dem oberen Teil des Bildschirms werden Steuerparameter zur Beeinflussung des Klangcharakters der Register angeboten. Beim C64 etwa kann man hier Tremolo oder Vibrato anwählen.

Besondere Effekte

Auch bei der Acorn-Version erfolgt die Steuerung von der Rechnertastatur aus, doch werden hier für die Wahl von Klangeffekten die Funktionstasten verwendet. Beim Acorn haben die Software-Entwickler die vorhandenen Funktionstasten und die Geschwindigkeit des Prozessors auch noch anderweitig nutzbar gemacht: Der Keyboarder kann während des Spiels Passagen hervorheben oder besondere Effekte erzielen, indem er über eine der Funktionstasten Sound-Varianten wie Woodblock, Baß, Cymbal usw. aufruft. Leider ist keine Möglichkeit vorgesehen, beim Spielen auf der Tastatur Begleitrhythmen laufen zu lassen.

Am unteren Bildschirmrand finden sich Möglichkeiten zur Beeinflussung der Tonhöhe (Pitch). Bei der Acorn-Version erfolgt die Tonhöhenverschiebung nach oben oder unten mit Hilfe der entsprechenden Cursortasten, beim C64 dagegen mit den Tasten „<“ und „>“. Es spricht nicht gerade für das Niveau der Software, daß bei der Commodore-Version beide Änderungsrichtungen auf dem Schirm mit „PITCH MINUS“ bezeichnet sind.

Das abgebildete „Echo“-Manual von LVL ist ein einfaches Musik-Keyboard für den Commodore 64 und den Acorn B, das direkt auf den Klangbaustein der Rechner zugreift und nicht die Anschaffung einer teuren Schnittstelle erfordert.

Unbefriedigend ist außerdem, daß die Klangfarben der Register nur sehr entfernt an die Instrumente erinnern, nach denen sie benannt sind. Diese Kritik trifft für Geräte zur elektronischen Musikerzeugung in den unteren Preisklassen zwar leider allgemein zu, aber die „Echo“-Soundeffekte wirken ganz besonders unecht. So haben das Geigen-, Bratschen- und Cello-Register beim C64 den gleichen Klangcharakter – nur die Tonhöhe ist unterschiedlich.

Bei beiden Rechnerversionen ist ein Synthesizermodus vorgesehen. Die zugehörigen Steuerparameter sind auf den jeweiligen Klangbaustein zugeschnitten. Daher ist beim C64 eine Hüllkurven- und Filterprogrammierung möglich. Wie beim Orgelmodus sind die Parameter durch Druck auf entsprechend belegte Tasten der Rechnertastatur nach Wunsch wählbar.

Beim Acorn B wird im Synthesizermodus vom ENVELOPE (Hüllkurven)-Befehl Gebrauch gemacht. Vier benutzerseitig definierbare Klangspektren stehen zur Verfügung. Anders als beim Orgelmodus besteht hier freie

Programmiermöglichkeit. Auf dem Schirm erscheinen 14 ENVELOPE-Parameter, die sich mit den Links/Rechts-Cursortasten einzeln anwählen und mit den Tasten für Auf/Ab ganz einfach ändern lassen.

Die Kommunikation zwischen Computer und Manual erfolgt, indem der Rechner die einzelnen Tasten softwaremäßig zyklisch abfragt. Dazu werden nacheinander die Zahlen von 0 bis 36 in die untersten sechs Bits des User-Port-Datenregisters gestellt. Sind mehrere Tasten gleichzeitig gedrückt, wird nur der höchste Ton akzeptiert. Das macht ein flüssiges Spiel über die Echo-Tastatur praktisch unmöglich, denn die vorherige Taste muß vollständig freigegeben sein, damit der Anschlag der nächsten Taste zuverlässig zur Kenntnis genommen wird.

Obwohl mit diesem Paket schon vielerlei Musikmöglichkeiten bestehen, kann es dem Profi und auch dem anspruchsvollen Amateur nicht genügend Niveau bieten.

Echo Synthesizer

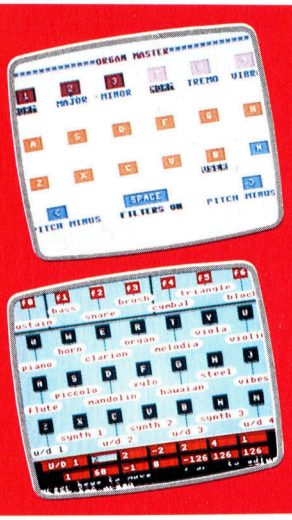
TASTATUR:
37 Klaviertasten

SCHNITTSTELLEN:
2adriges Kabel für den User Port des Acorn B mit Adapter für den Commodore 64

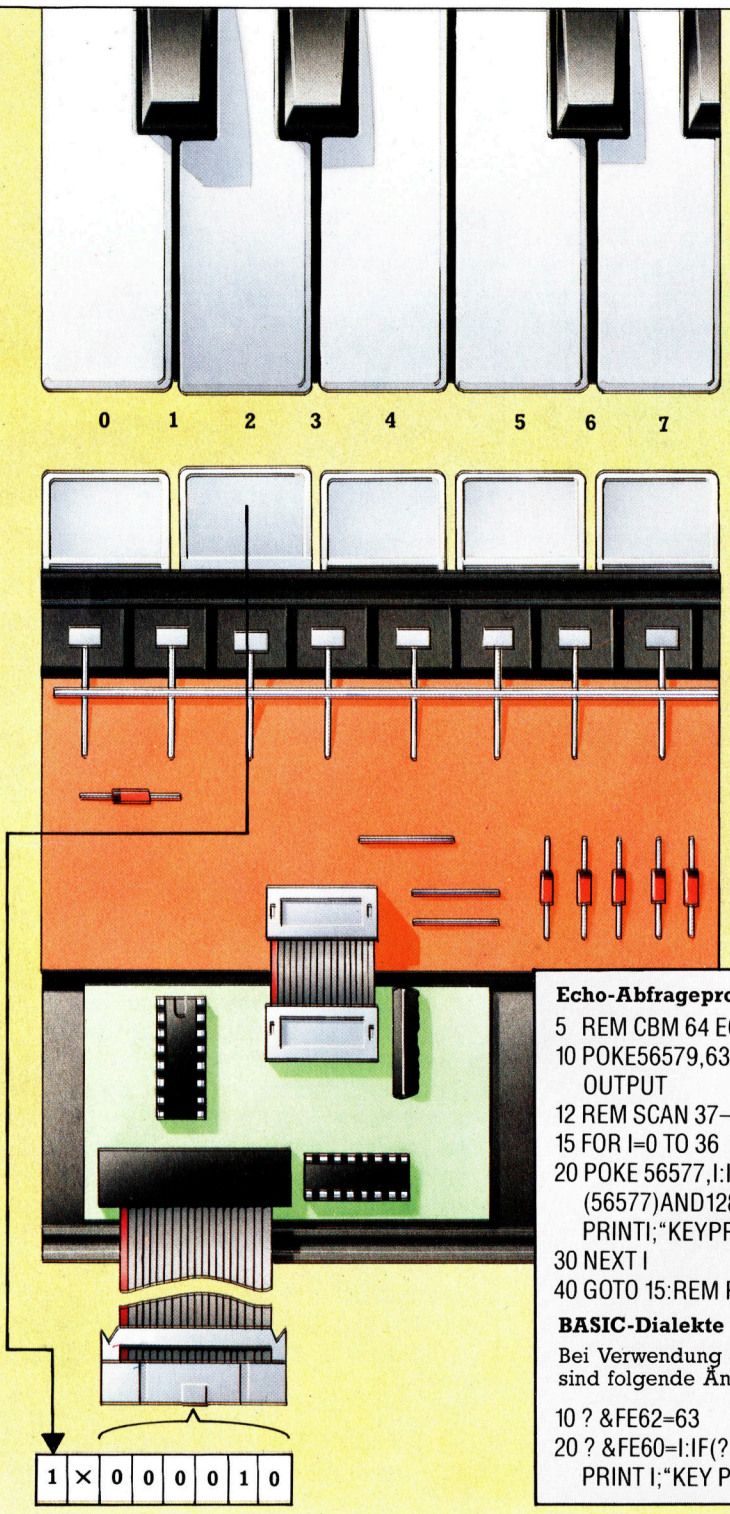
SOFTWARE:
Die Software für den Acorn B ist auf Cassette und Diskette, die C64-Version nur auf Cassette verfügbar

STÄRKEN:
Das Echo-Paket erlaubt bei beiden Rechnern die volle Nutzung der Möglichkeiten des Klangbausteins.

SCHWÄCHEN:
Durch die unzulängliche Software wird die praktische Brauchbarkeit des Systems leider stark beeinträchtigt; allerdings ist eine verbesserte Fassung angekündigt.



Die Abbildung zeigt oben den Commodore-64-Bildschirm für den Orgelmodus, unten das Acorn-Menü für den Synthesizer-Betrieb. Die Echo-Softwareversionen sind für die beiden Rechner gleichartig aufgebaut, unterscheiden sich jedoch in der Anzahl der Wahlmöglichkeiten, sind aber kaum ausreichend.



Echo-Abfrage

Die Echo-Tastatur wird über den User Port angeschlossen und softwaremäßig abgefragt. Das funktioniert eigentlich ganz einfach: Die untersten sechs Bits des User-Port-Datenregisters werden als Ausgänge für die Datenübertragung zum Manual hin definiert, während das höchstwertige Bit (Bit 7) die Rückmeldung von der Echo-Tastatur annimmt.

Um einen Tastenanschlag am Manual zu erkennen, sind zwei Schritte erforderlich: Zunächst muß durch POKE eine Zahl zwischen 0 und 36 ins Datenregister des User Port gesetzt werden, je nachdem welche der 37 Tasten an der Reihe ist. Die Tasten werden mit Null beginnend von links her gezählt. Für jede Taste ist dann zu prüfen, ob in Bit 7 des Datenregisters eine „0" oder eine „1" ansteht, wozu der Registerinhalt mit 128 geUNDet wird. Lautet das Ergebnis nicht Null, war die Taste gedrückt.

Um das ganze Manual abzufragen, müssen Sie mit Hilfe einer Schleife fortlaufend alle 37 Tasten durchgehen.

Echo-Abfrageprogramm für C 64

```
5  REM CBM 64 ECHO SCAN
10 POKE56579,63:REM SET BITS 0-5 FOR
   OUTPUT
12 REM SCAN 37-KEY KEYBOARD
15 FOR I=0 TO 36
20 POKE 56577,I:IF(PEEK
   (56577)AND128)<>0 THEN
   PRINTI;"KEYPRESSED"
30 NEXT I
40 GOTO 15:REM REPEAT
```

BASIC-Dialekte

Bei Verwendung der Tastatur am Acorn B sind folgende Änderungen anzubringen:

```
10 ? &FE62=63
20 ? &FE60=I:IF(?FE60 AND 128)<>0 THEN
   PRINT I;"KEY PRESSED"
```

C in Funktion

Die Programmiersprache C arbeitet hauptsächlich mit Funktionen, die selbst definiert oder vom System vorgegeben werden. Wir untersuchen die Struktur und die Rolle der Funktionen anhand eines Programms.

Die Sprache C baut zwar grundlegend auf dem Konzept der Funktion auf, ist aber dennoch keine Funktionssprache, da die Funktionen als Prozeduren definiert sind.

Mit „Funktion" ist hier ein Vorgang gemeint, dem bestimmte Werte (Argumente oder Parameter) übergeben werden und der nach deren Verarbeitung einen einzelnen Wert zurückliefert. Die übergebenen Werte brauchen keine Grundtypen wie Ganzzahlen oder reelle Zahlen zu sein, sondern können auch strukturierte Typen wie Arrays darstellen.

Auch Blöcke spielen eine große Rolle. In C ist ein „Block" ein selbständiger, von Klammern ({}) umschlossener Codeabschnitt. (Zum Vergleich: PASCAL zeigt Blöcke durch begin . . . end an). Wichtig ist auch der „Bereich" einer Variablen, der Programmabschnitt also, in dem die Variable eingesetzt wird. Da nur wenige BASIC-Versionen mit lokalen Variablen arbeiten, werden viele BASIC-Programmierer sich erst daran gewöhnen müssen, daß der Einsatz von lokalen Variablen auf bestimmte Programmbereiche beschränkt ist. In C können Variablen an jeder beliebigen Stelle deklariert werden. Sie müssen daher genau wissen, von wo Sie sich einfach darauf beziehen können.

Hier die Grundregeln für die Deklaration von C-Variablen:

- Wird eine Variable am Anfang einer Funktionsdefinition deklariert, dann gilt sie für den gesamten Bereich der Funktionsdefinition, also auch für jede andere Funktion, die im Inneren der Hauptfunktion deklariert ist. Da das Programm selbst aus der Funktion main() besteht, gelten am Anfang deklarierte Variablen im gesamten Programmbereich – sie sind „global".

Gut aufgebaute Programme enthalten soweit wie möglich völlig eigenständige Module, die nur mit lokalen Variablen und formalen Parametern arbeiten. Wenn die Ausführung einer Funktion außer lokalen Variablen noch andere Daten einsetzt, wird das „Nebenwirkung" genannt. So ist beispielsweise der Betrieb eines Ein-/Ausgabegerätes eine Nebenwirkung – wie auch der Einsatz einer globalen Variablen. Einige Nebenwirkungen sind positiv – das heißt, sie haben keine Auswirkungen auf die Ausführung anderer Programmteile. Negative Nebenwirkungen stören jedoch den Programmablauf und sollten vermieden werden.

- Eine im Inneren eines gesamten Blockes deklarierte Variable hat nur für den Rest des Blockes Gültigkeit.

Die Variablendeklaration hat besondere Bedeutung, da es Teil der „Philosophie" von C ist, Programme aus Modulen aufzubauen. Jedes dieser Module wird von einer eigenen Datei

dargestellt. Die Regeln, mit denen die Funktionen anderer Dateien in den Bereich einer Variablen eingeschlossen werden können, sind jedoch sehr kompliziert.

Es gibt zwei Möglichkeiten, Argumente oder Parameter an Funktionen zu übergeben:

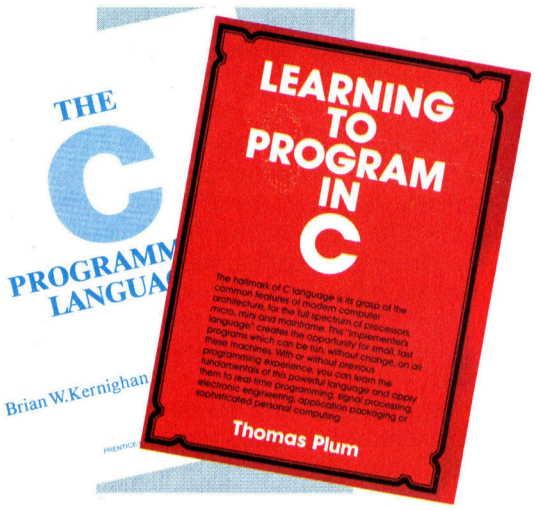
- **Wert:** Durch Anlegen einer neuen Variablen. Variablen dieser Art haben für die Funktion lokale Bedeutung und werden mit den von der aufrufenden Routine übergebenen Werten initialisiert.
- **Bezug:** Hier wird die Adresse der Variablen übergeben, die den Parameter enthält. Auf diese Weise kann die gleiche Variable von der Funktion und der aufrufenden Routine verwandt werden.

C übergibt alle Parameter als Wert. Wir werden jedoch später sehen, daß die Sprache breitgefächerte Möglichkeiten hat, Adressen und Pointer zu verarbeiten und damit auch die Referenzmethode einsetzen kann.

Der formale Aufbau einer Funktionsdefinition lautet:

```
funktion_typ funktion_name( formale_argu-
   ment_liste);
variablen_deklarationen;

   inhalt der funktion;
}
```



Für die wachsende Beliebtheit von C ist auch die Firma Prentice-Hall verantwortlich, die umfassendes Lesematerial über die Sprache herausgab, darunter die erste Sprachdefinition von Kernighan und Ritchie. Der hohe Preis dieses dünnen Buches mag vielleicht abschrecken, doch bleibt es bis zur Definition eines internationalen Standards die Grundlagenbeschreibung der Sprache. Ein weiteres wichtiges von Prentice-Hall herausgegebenes Buch, „Learning to Program in C", liegt als deutsche Übersetzung vor: „Das C Lernbuch" Verlage Carl Hanser und Prentice-Hall, ISBN 3-446-14165-0.

Die Angabe **funktion_typ** kann weggelassen werden, wenn der zurückgegebene Wert vom Typ **int** ist. Die **formale_argument_liste** enthält eine Liste der Variablen, in die die Werte beim Einsatz der Funktion gespeichert werden.

Die entsprechende Funktion wird folgendermaßen aufgerufen:

```
funktion_name (aktuelle_argument_liste)
```

Der Aufruf kann an jedem Punkt erfolgen, an dem ein Wert des von der Funktion gelieferten Typs eingesetzt wird. Wenn der zurückgegebene Wert nicht vom Typ **int** ist, muß der Funktionsname vor dem Aufruf als Variable des entsprechenden Typs deklariert werden. Die aktuelle Argumentliste enthält alle Werte (oder Variablen mit diesen Werten), die an die formalen Argumente der Funktion übergeben werden. Die aktuellen und formalen Argumente müssen natürlich in beiden Listen vom Typ und von der Position her übereinstimmen.

Mit **return** übergibt eine Funktion den gewünschten Wert an die aufrufende Routine. Es ist eine beliebige Zahl dieser Befehle möglich, doch sollte vor Verlassen der Funktion mindestens einer ausgeführt werden.

Das folgende Programm liest eine Reihe von reellen Zahlen ein und zeigt die größte und die kleinste an. Das Programm enthält drei Funktionen. **ausgabekopf** hat keine Argumente und auch keinen **return**-Befehl. Das bedeutet, daß der von der Funktion zurückgegebene Wert nicht definiert ist. Da die aufrufende Routine ihn nicht benötigt, wird er einfach „weggeworfen". Die Funktion hat Nebenwirkungen, die jedoch alle positiv sind.

Werte für Variablen

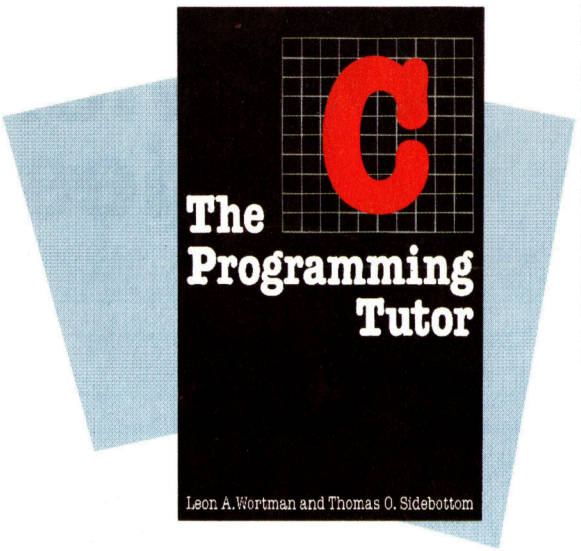
Das Programm arbeitet weiterhin mit der Standardfunktion **scanf**, die Eingaben auf ähnliche Weise formatiert wie **printf** die Ausgaben. Der erste Parameter der Funktion ist ein String mit den Formatinformationen der Eingabewerte. Die weiteren Argumente von **scanf** sind die Variablen, in die die Werte eingelesen werden. Da Parameter sich jedoch nur als Werte übergeben lassen, können sie an die aufrufende Routine auch keine Werte zurückgeben. Die Parameter müssen daher statt der Variablen selbst die Adressen der Variablen enthalten. Vor jeder beim Aufruf von **scanf** angegebenen Variable muß daher der Adreßoperator & stehen.

```
#include<stdio.h>
main ()
{
   int count, size_of_input,
   double smallest, largest, number, min(),
      max();
/* beachten sie die deklaration der funktionen
   min and max */
/* beachten sie weiterhin, daß im inneren
   eines blockes deklarierte variablen außer-
   halb des blockes nicht verfuegbar sind */
   printheading();
/* nach aufruf der funktion wird der zurueckge-
   gebene wert als int angesehen und 'wegge-
   worfen', d. h. nirgendwo anders eingesetzt
      */
   printf ("/nsize of input:");
   scanf ("%d",&size_of_input);
/* herausfinden, wieviele zahlen eingelesen
   werden sollen */
   printf ("Jetzt %d reelle Zahlen eingeben :/n",
      size_of_input);
/* die erste zahl holen, die im augenblick die
   kleinste und groesste ist */
   scanf ("%if", &number);
   largest=smallest=number;
/* jetzt die anderen zahlen holen */
```

```
    for (count=2; count<=size_of_input;
    ++count)
    {
       scanf("%if", &number);
       smallest = min (smallest, number);
       largest=max (largest, number);
    }
    printf ("/nsmallest is %f/nlargest is
    %f/n", smallest, largest);
}
/* hier die funktionsdefinitionen */
printheading(,
/* da kein typ gebraucht wird, int annehmen */
{
    printf ("/n%s/n%s/n%s/n",
       "Zuerst die Groesse des Zahlensets ein-
          geben",
       "dann entsprechend viele reelle Zahlen
          eingeben.",
       „Die groesste und kleinste Zahl werden
          dargestellt");
}
double min(x,y);
double x,y;
/* beachten sie die parameterdeklaration */
{
    if (x<y)
       return (x);
    else
       return(y);
}
double max (x,y);
double x,y;
{
    if (x>y)
       return(x);
    else
       return(v);
```

Viele Compiler verstehen „Compileranweisungen", die in den Programmtext eingebaut sind. Bei C wurde dieses Konzept weiter ausgebaut als bei den meisten anderen Sprachen. Jeder C-Compiler enthält daher einen „Vorprozessor", der bestimmte „Steuerzeilen" als Befehle erkennt und das Programm verändert, bevor es überhaupt dem Compiler übergeben wird. Sie sind am vorangehenden # auszumachen.

Die Vorprozessoranweisung **#include<dateiname>** oder **#include"dateiname"** befiehlt dem Vorprozessor, an dieser Stelle den Inhalt der angegebenen Datei einzuschließen. Einige Systeme kennen keinen Unterschied zwischen spitzen Klammern <> und Anführungszeichen "", andere suchen dadurch jedoch an unterschiedlichen Plätzen nach den angegebenen Dateien. Spitze Klammern werden normalerweise für Systemdateien eingesetzt und Anführungszeichen für selbstdefinierte Dateien. Die Funktionen **printf** und **scanf** sind in der Systemdatei **stdio.h** enthalten. Alle Programme, die damit arbeiten, müssen daher die Anweisung **#include<stdio.h>** enthalten.



Das von Prentice-Hall herausgegebene Buch „The C Programming Tutor" ist eine ausgezeichnete Einführung in die Sprache. Klar und deutlich geschrieben, werden dem Leser komplizierte Themen nahegebracht. Das Buch ist besonders für Leser geeignet, die außer BASIC keine anderen Programmiersprachen kennen (ISBN 0-13-110024-6).

Da diese Aufrufe geschachtelt werden dürfen, können auch eingeschlossene Dateien mit **#include** weitere Dateien einbeziehen.

Die Anweisung **#define** macht den Einsatz von Macros möglich. Die einfachste Form ist dabei die Definition einer Konstanten. **#define EOF–1** ersetzt beispielsweise jedes Auftreten des Namens EOF durch –1.

```
#define PI 3.14159
#define EQ ==
```

Derart definierte Namen werden in Großbuchstaben geschrieben und normale Variablen in Kleinbuchstaben.

Auch Parameter können in Macrodefinitionen wie

```
#define SUMSQ(x,y)((x*x)+(y*y))
```

vorkommen, so daß eine Art Funktionsdefinition wie in BASIC vorliegt. Das Auftreten von SUMSQ(3,4) würde dabei durch ((3*3)+ (4*4)) ersetzt werden. Die Klammern sind aus Sicherheitsgründen mit aufgeführt: Das Macro könnte nun in einer Situation eingesetzt werden, in der Klammern nötig sind.

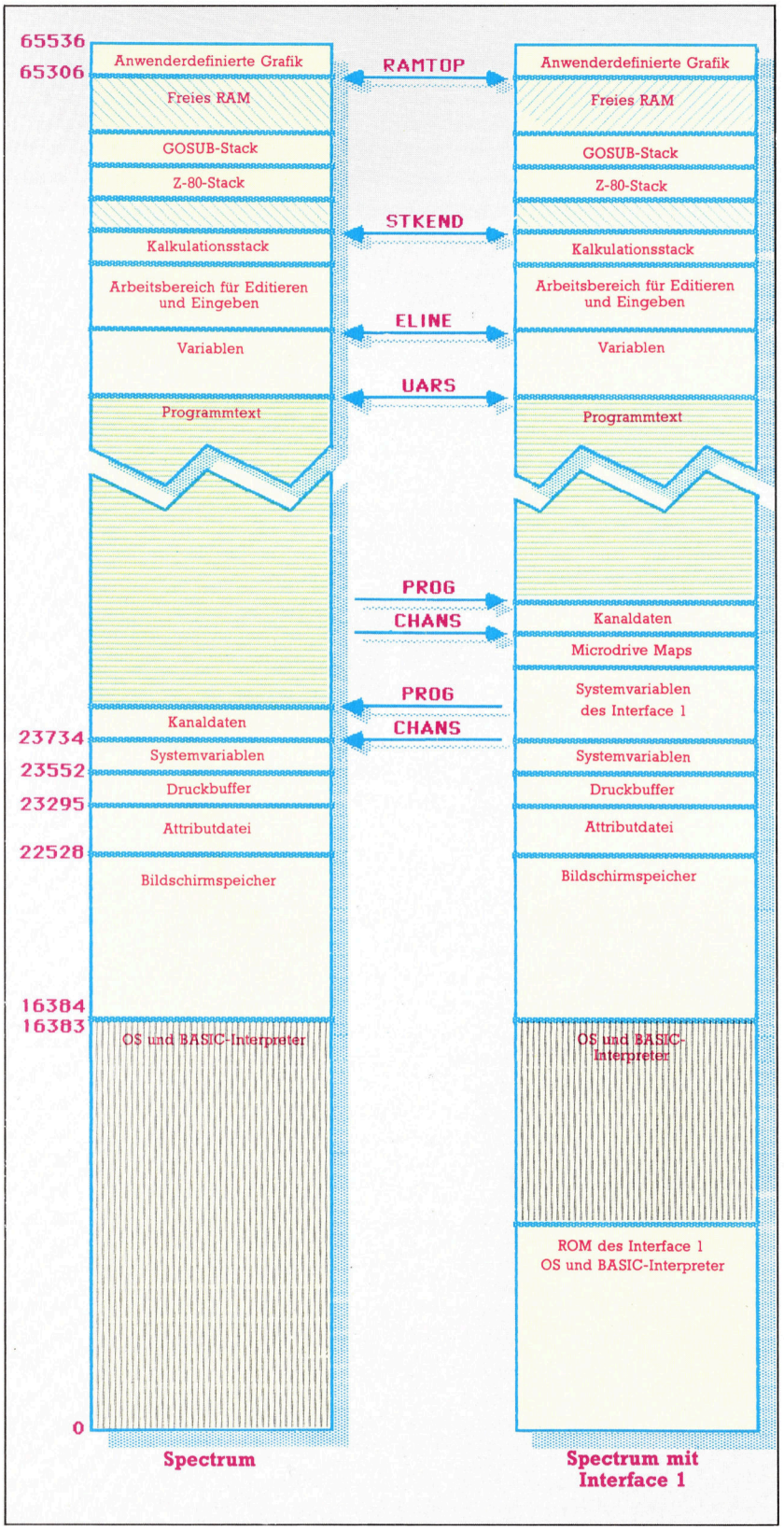
Oft wird auch die Anweisung **#if...#else...#endif** verwandt, die die bedingte Compilierung steuert. Wenn der konstante Ausdruck nach der **#if**-Anweisung mit wahr (nicht-Null) bewertet wird, werden die darauf folgenden Codezeilen compiliert. Ergibt die Bewertung des Ausdrucks falsch (Null), werden die auf **#else** folgenden Zeilen compiliert. Diese Struktur ist besonders bei der Programmentwicklung praktisch, wenn zusätzliche **printf**-Befehle Klarheit über den Programmablauf bringen sollen. Hier können Anweisungen wie **#define DEBUG 1** eingesetzt werden, die bei Bedarf mit

```
#if DEBUG
   printf(werte . . .)
#endif
```

die gewünschten Werte ausdrucken. Nach Beendigung der Fehlersuche muß nur **#define DEBUG 0** angegeben werden, und eine erneute Compilierung schließt diese Anweisungen nicht mehr mit ein.

Die vielen Talente des Spectrum

Die Memory Map des 48K Spectrum (und Spectrum+) zeigt deutlich, wie das Interface 1 die Belegung einiger RAM-Bereiche verändert. Die Systemvariablen erstrecken sich bis 23733.



Wir beginnen eine Serie über das Betriebssystem des Sinclair Spectrum und untersuchen zunächst die Memory Map des Computers. Dabei wird deutlich, welche Positionen sich für die Unterbringung von Maschinencode am besten und einfachsten eignen.

Der Sinclair Spectrum hat im Gegensatz zum Acorn B kein separates Betriebssystem-ROM. OS und BASIC-Interpreter des Spectrum (bzw. Spectrum+) befinden sich auf dem gleichen Chip. Das OS des Spectrum ist ein kompakter Satz von Routinen, der etwa halb soviel Codes umfaßt wie das OS des Acorn B. Es macht etwa sieben KByte der 16 KByte des ROM-Chips aus,

Trotz des langlebigen Betriebssystems und seines kompakten Codes hat der Spectrum im Vergleich zum Acorn B einen großen Nachteil. Da sein OS – von wenigen Ausnahmen abgesehen – nicht mit Vektoren arbeitet, ist eine Änderung bestimmter Abläufe recht kompliziert. Das Problem läßt sich jedoch umgehen. Wir kommen später darauf zurück.

In unserer Serie über das OS des Acorn B haben wir erfahren, daß das Betriebssystem die Routinen für Ein- und Ausgabe, Anzeige, Speicherung und Interrupts steuert und generell die Schnittstelle zwischen Hardware und dem Anwenderprogramm oder einem BASIC-Interpreter darstellt. Sehen wir uns zunächst die Memory Map des Spectrum im Bild an.

Den ersten Speicherblock von 16 KByte belegt das ROM mit BASIC-Interpreter und Betriebssystem. Das OS befindet sich in den unteren sechs bis sieben KBytes des Chips und der BASIC-Interpreter in der oberen Hälfte. Wenn das Gerät mit dem Interface 1 arbeitet, werden die unteren acht KByte des Speichers, ähnlich wie der Bereich von &8000 bis &BFFF auf dem Acorn B, durch „Paging" angesprochen. Beim Zugriff auf die Routinen des Interface 1, zum Beispiel zur Steuerung der Microdrives oder der seriellen Schnittstelle, wird das normale ROM ausgeblendet und statt dessen das ROM des Interface 1 angesprochen.

Den restlichen Speicher nimmt das RAM ein. Beim Einsatz des Interface 1 gibt es allerdings Unterschiede zum normalen Aufbau.

Der Bildschirmspeicher enthält die Daten der Abbildungen, die auf dem Bildschirm an-

gezeigt werden sollen. Die zugehörigen Farbdaten und Informationen wie BRIGHT und FLASH werden in der Attributdatei gespeichert, da sie sich jeweils auf ein Zeichenfeld der Anzeige beziehen. Der Druckbuffer besteht aus 256 Speicherbytes, die vom ZX-Drukker abgefragt werden, wenn der Befehl für eine Hardcopy eingegeben wird.

Die Systemvariablen werden vom OS und vom BASIC-Interpreter benutzt. Sie enthalten die wichtigsten Systemdaten. Eine vollständige Liste dieser Variablen ist im Anwenderhandbuch des Spectrum abgedruckt. Leider wurde sie nicht in das Handbuch des Spectrum+ übernommen.

Im Bereich der Systemvariablen treten die Unterschiede zwischen einem Spectrum, der mit oder ohne das Interface 1 arbeitet, am deutlichsten zutage. Das ROM des Interface 1 benötigt für seine Abläufe einen Arbeitsbereich und außerdem Systemvariablen, die es selbst erzeugt und hinter den normalen Systemvariablen des Spectrum ablegt. Da dabei jedoch die Aufteilung des restlichen Speichers ins „Rutschen" gerät, müssen einige Bereiche umorganisiert werden.

Auch die „Microdrive Maps" gibt es nur bei Verwendung des Interface 1. Sie geben an, welche Bereiche des Bandes belegt sind.

Bei den „Kanaldaten" hat man begrenzte Möglichkeiten, die Befehle LLIST, LPRINT, INPUT# und PRINT# über Vektoren einzusetzen. Sie können beispielsweise einen Maschinencodetreiber für einen Standarddrucker schreiben und die Kanaldaten so ändern, daß bei der Ausgabe von LLIST oder LPRINT Ihr Code und nicht die normalen Routinen des BASIC-Interpreters zum Einsatz kommen. In der nächsten Folge beschäftigen wir uns mit Abläufen dieser Art. Die Kanaldaten werden ebenfalls vom Interface 1 benötigt.

Editieren und Eingabe

Im Bereich Programmtext speichert das System den Text von BASIC-Programmen und deren Variablen.

Der Arbeitsbereich für „Editieren und Eingabe" wird vom System für das Editieren einzelner Programmzeilen oder die Eingabe von Daten und Programmzeilen eingesetzt. Da die Programmzeilen nach Beendigung des Editierens oder der Eingabe an den entsprechenden Positionen des Programms untergebracht werden, stehen diese Bereiche danach zur freien Verfügung.

Über diesem Arbeitsbereich liegen mehrere Stacks. Den „Kalkulationsstack" verwendet der BASIC-Interpreter für arithmetische Berechnungen. Mit dem Z80-Stack arbeitet die CPU auf ähnliche Weise, wie der 6502 die Speicherseite Eins benutzt: Dort werden die Rücksprungadressen der Maschinencodesubroutinen und zwischengespeicherte Daten untergebracht. Der GOSUB-Stack wird wiederum vom BASIC-Interpreter gebraucht. Er speichert die Rücksprungadressen der GOSUB-Befehle in BASIC. Schließlich gibt es noch einen Bereich für anwenderdefinierte Grafik mit den vom Anwender definierten Zeichen.

Die Memory Map zeigt deutlich, daß für selbstgeschriebene Maschinencodeprogramme kein Speicher reserviert wurde. Der Spectrum kann Maschinencode in drei Bereichen unterbringen: In einem REM-Befehl der Zeile 1. Die Speicherposition der ersten Zeile kann aus dem Wert der Systemvariable PROG errechnet werden. Wenn Sie auf PROG fünf addieren, erhalten Sie die Adresse des ersten Zeichens eines REM-Befehls in Zeile 1.

Maschinencode kann aber auch im Druckbuffer oder im Bereich zwischen RAMTOP und dem Speicher für anwenderdefinierte Zeichen abgelegt werden. Auf dem Acorn B ähnelt die letzte Methode dem Einsatz des Speichers zwischen HIMEM und dem Anfang des Video RAMs. Die folgende Tabelle zeigt die Vorteile der beiden Methoden:

Position	Spectrum	Mit Interface 1
REM in Zeile 1	✓	✗
Druckbuffer	(✓)*	(✓)*
Über RAMTOP	✓	✓

* Sollte bei Betrieb eines Druckers nicht verwandt werden

Es ist leicht zu erkennen, daß nur die letzte Möglichkeit in Frage kommt, da die beiden anderen in gewissen Situationen Probleme verursachen. Jeder Bereich oberhalb von RAM-TOP kann von BASIC nicht überschrieben werden. Er läßt sich außerdem leicht reservieren. Die genaue Position von RAMTOP ist in einer Zwei-Byte-Variablen bei den Adressen 23730 und 23731 gespeichert. Der Wert hat das übliche Z80-Format: Lo-Byte gefolgt Hi-Byte. Der Befehl liefert die aktuelle Position von RAMTOP:

```
LET ramtop=PEEK23730 + 256*PEEK23731
```

Nachdem Sie die Position von RAMTOP herausgefunden haben (auf einer standardmäßigen 48-K-Maschine liegt sie auf 65305), sollten Sie abschätzen, wieviele Byte Ihr Maschinencodeprogramm verlangt. Mit der Formel: Wert = RAMTOP − (Zahl-der-Programmbytes + 1) können Sie den neuen Wert errechnen und RAMTOP mit dem CLEAR-Befehl einfach heruntersetzen:

```
CLEAR Wert
```

Von einem BASIC-Programm aus löst dieser Befehl zunächst ein CLS aus, stellt dann die PLOT-Position auf den Ausgangswert zurück und ruft RESTORE auf. Außerdem wird der GOSUB-Stack gelöscht und verlegt. Sie sollten mit diesem Befehl vorsichtig umgehen und ihn möglichst am Programmanfang unterbringen.

Nach Ausführung von „CLEAR Wert" enthalten die Adressen 23730 und 23731 die neue Position von RAMTOP. Das erste Byte, das Ihnen für den Maschinencode zur Verfügung steht, befindet sich bei der Adresse (Wert + 1). Auf einer Standardmaschine mit 48 KByte reserviert der Befehl CLEAR 59999 etwa fünf KByte für Ihre Programme (Anfang bei 60000). Sie können Ihren Code mit POKE in den reservierten Bereich schreiben.

Keine OSBYTE und OSWORD

Sehen wir uns nun einige Systemvariablen an, die Informationen über die Speicheraufteilung enthalten. Zugriff auf die Variablen erhalten Sie mit PEEK oder POKE in die entsprechenden Speicherstellen. Es gibt jedoch keine Systemroutinen wie OSBYTE oder OSWORD, die auf dem Acorn B zur Verfügung standen.

- **chars**: Diese Variable liegt bei 23606 und 23607 und zeigt auf eine Speicheradresse, die sich 256 Bytes unter der Information über den Zeichensatz befindet. Normalerweise zeigt die in dieser Zwei-Byte-Variablen gespeicherte Adresse auf eine ROM-Adresse, die 256 Bytes unter dem ersten Byte der Definition des Leerzeichens (CHR$(32)) liegt. Die aktuelle Adresse der Variablen kann mit

```
LET chars=PEEK 23606 + 256*PEEK 23607
```

abgefragt werden. Wenn Sie einen eigenen, im RAM gespeicherten Zeichensatz verwenden wollen, brauchen Sie nur mit dem Wert dieser Adresse darauf zu zeigen.

- **vars**: Diese Variable (in 23627 und 23628) enthält die Anfangsadresse der gespeicherten BASIC-Variablen. Sie können diesen Wert zwar mit PEEK abfragen, sollten ihn aber nicht ändern, da der Spectrum dann seine Variablen nicht mehr „findet". Die Zeile

```
LET vars=PEEK 23627 + 256*PEEK 23628
```

ergibt die Anfangsadresse der Variablen.

- **prog**: Diese Systemvariable (bei 23635 und 23636) enthält die Anfangsadresse des BASIC-Programms und entspricht damit der Systemvariablen PAGE des ACORN B. Sie wird mit

```
LET prog=PEEK 23635 + 256*PEEK 23636
```

abgefragt. Der Befehl

```
PRINT vars-prog
```

zeigt die Länge des Programms an.

- **eline**: Die Variable ist an den Adressen 23641 und 23642 untergebracht und enthält die Anfangsadresse von Texten, die in das Gerät eingegeben werden. Sie können damit feststellen, wieviel Speicher die Variablen belegen. Wenn Sie vars bereits ermittelt haben, ergibt

```
LET eline=PEEK 23641 + 256*PEEK 23642
PRINT eline-vars
```

den von den Variablen belegten Speicher.

- **chans**: Diese Variable (bei 23631 und 23632) liefert die Anfangsadresse der Kanalinformation. Wir gehen in dieser Serie noch genauer auf die Kanäle ein.
- **stkend**: Diese Variable liegt an den Adressen 23653 und 23654. Zusammen mit dem Wert von RAMTOP gibt sie Auskunft darüber, wieviel Speicherplatz für BASIC-Programme und Variablen noch zur Verfügung steht.

```
LET stkend=PEEK 23653 + 256*PEEK 23654
```

errechnet den aktuellen Variablenwert und

```
PRINT ramtop — stkend
```

ergibt den verbleibenden freien Speicherplatz. Sie können den freien Speicher aber auch mit einer ROM-Routine errechnen, die jedoch nicht so präzise arbeitet wie die oben geschilderte Methode. Wenn Sie den aktuellen Wert von RAMTOP bereits haben, ergibt

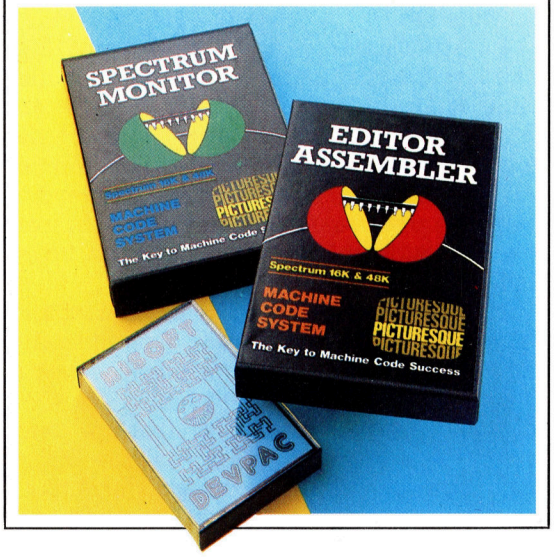
```
PRINT ramtop — USR 7962
```

eine ungefähre Angabe über den noch freien Speicherplatz.

Assemblerzeit

Für die Besitzer eines Sinclair Spectrum, die mit Maschinencode arbeiten wollen, gibt es eine Reihe von Assemblerpaketen. Eines der bekanntesten ist DEVPAC 3 von Hisoft. Es wird auf Cassette geliefert und enthält den GENS3 Assembler und den Monitor/Disassembler MON3. Auch eine Microdrive-Version ist verfügbar. DEVPAC 3 ist zwar nicht besonders bedienerfreundlich, aber sehr preisgünstig.

Weiterhin gibt es die Doppelpakete Editor Assembler und Spectrum Monitor von Picturesque. Das System von Picturesque hat den Vorteil, daß es problemlos auf einem 16K Spectrum läuft. Die Handbücher dieser Pakete sind besser als die von Hisoft, und das gesamte System macht einen weit professionelleren Eindruck.



Alles mit Maßen

Dem Spectrum fehlt unter anderem eine komfortable Möglichkeit, die Länge eines Programms zu berechnen. Wir haben hier ein Dienstprogramm abgedruckt, das die Länge des im Speicher abgelegten BASIC-Programms mit den Systemvariablen PROG und VARS berechnet und in einem REM-Befehl der Zeile 1 ablegt. Das Modul stellt nur die Anforderung, daß die erste Zeile des Programms ein REM mit mindestens fünf Zeichen enthält:

```
1 REM 00000 Bytes lang
```

Die Ausführung des Maschinencodes schreibt in diese Zeile die Programmlänge, etwa:

```
1 REM 00801 Bytes lang
```

Das erste Listing enthält das Assemblerprogramm. Wenn Sie keinen geeigneten Assembler haben, können sie den Maschinencode auch einfach über das BASIC-Ladeprogramm einlesen.

In dem BASIC-Ladeprogramm setzt CLEAR 61999 den Wert der Systemvariablen RAMTOP auf 61999. Das erste „sichere" Byte befindet sich daher bei der Adresse 62000, und hier beginnt auch der Maschinencode. RAMTOP bleibt bis zur Ausführung eines weiteren CLEAR n oder eines PRINT USR 0 auf dem eingestellten Wert. Auch ein Ab- und Anschalten der Maschine löscht den Wert. Da NEW keinen Einfluß auf RAMTOP hat, kann der Maschinencode von BASIC nicht überschrieben werden, solange der geänderte RAMTOP aktiv ist.

Wenn sich das Programm im Speicher befindet, kann es mit USR aktiviert werden. Das Argument der Funktion USR(n) gibt die Adresse der Maschinenroutine an, die Sie aufführen wollen. Diese Adresse kann jede beliebige Speicheradresse sein. Die folgenden Befehle rufen daher alle einen Maschinencode bei Adresse 62000 auf und können unseren Programmcode aktivieren:

```
RANDOMISE USR(62000)
PRINT USER(62000/
LET L=USR(62000)
```

Das mit PRINT und LET eingesetzte USR liefert ein Ergebnis an BASIC zurück, das weiter verarbeitet werden kann. Das Resultat befindet sich beim Rücksprung des Maschinencodes zum BASIC in dem Registerpaar BC. Wenn Ihre Routine den Wert des BC-Registers nicht berührt, wird die Adresse der Routine zurückgeliefert.

Assemblerprogramm

```
62000                10              ORG 62000
62000  2A4B5C        20              LD    HL,(23627)   ;get VARS in HL
62003  ED5B535C      30              LD    DE,(23635)   ;get PROG in DE
62007  AF            40              XOR   A            ;clear carry flag
62008  ED52          50              SBC   HL,DE        ;get program length into HL
62010  DD2A535C      60              LD    IX,(23635)   ;start of program into IX
62014  DD23          70              INC   IX
62016  DD23          80              INC   IX           ;increment IX to point to
62018  DD23          90              INC   IX           ;character after REM token
62020  DD23         100              INC   IX           ;in first program line
62022  DD23         110              INC   IX
62024  011027       120              LD    BC,10000     ;start getting ASCII codes
62027  CD6BF2       130              CALL  PRINT        ;of each digit in program
62030  DD23         140              INC   IX           ;length and store in REM
62032  01E803       150              LD    BC,1000      ;statement
62035  CD6BF2       160              CALL  PRINT
62038  DD23         170              INC   IX
62040  016400       180              LD    BC,100.
62043  CD6BF2       190              CALL  PRINT
62046  DD23         200              INC   IX
62048  010A00       210              LD    BC,10
62051  CD6BF2       220              CALL  PRINT
62054  DD23         230              INC   IX
62056  010100       240              LD    BC,1
62059  AF           250  PRINT:     XOR   A            ;clear carry and xero counter
62060  B7           260  L1:        OR    A
62061  ED42         270              SBC   HL,BC        ;subtract power of ten from
                                                       ;program length...
62063  3803         280              JR    C,FINISH     ;and jump if negative
62065  3C           290              INC   A            ;inc A reg if not negative
62066  18F8         300              JR    L1           ;repeat
62068  09           310  FINISH:    ADD   HL,BC        ;restore prog length to
                                                       ;positive value for the next
                                                       ;power of ten
62069  C630         320              ADD   A,48         ;get ASCII code of digit
62071  DD7700       330              LD    (IX),A       ;store ASCII code in REM
62074  C9           340              RET                ;finished
```

Ladeprogramm in BASIC

```
  10 REM 00000 Bytes Long
  20 CLEAR 61999
  30 FOR i=0 TO 74
  40 READ a: POKE 62000+i,a
  50 NEXT i
1000 DATA 42,75,92,237,91,83,92,175,237,82,221,42,
83,92,221,35,221,35,221,35,221,35,221,35,1,16,39,2
05,107,242,221,35,1,232,3,205,107,242,221,35,1,100
,0
1010 DATA 205,107,242,221,35,1,10,0,205,107,242,22
1,35,1,1,00,175,183,237,66,56,3,60,24,-8,9,198,48,
221,119,0,201
```

Anmerkung: Um sicherzustellen, daß unser Programm die korrekte Position findet, an der die Ziffern mit der Programmlänge eingesetzt werden, sollten Sie zwischen der Zeilennummer und dem Schlüsselwort REM keine Leerzeichen einsetzen.

Blitter und WIMPS

Für unsere Untersuchung der Ursprünge des Graphics Environment Manager (GEM) von Digital Research sehen wir uns einige der älteren kommerziell verfügbaren WIMP-Systeme an.

Die Forschung an dem von Xerox gegründeten PARC beeinflußte viele unterschiedliche Bereiche der Sprach- und Hardwareentwicklung. Die Folgen sind in der Entwicklung von PASCAL und Modula-2 deutlich erkennbar. Viele der frühen Produkte sind immer noch verfügbar. So hat sich GKS, das „Grafics Kernel System" für die maschinenunabhängige Grafikprogrammierung, in eine komplexe Anwendung entwickelt, die auf den CAD-CAM-Bereich beschränkt ist. Am anderen Ende der Skala erschien die Betriebssystemerweiterung GSX. Die verwendeten Abkürzungen:

ETH — ElektroTechnische Hochschule (Professor Niklaus Wirth)
PARC — Xerox Palo Alto Research Center
UCSD — University of California at San Diego
GKS — Graphics Kernel System (standardised in 1977)
DR — Digital Research Ltd
GSX — Graphics System Extension (DR)
GEM — Graphics Environment Manager (DR)
MS — Microsoft Corporation
GDI — Graphics Device Interface (MS)

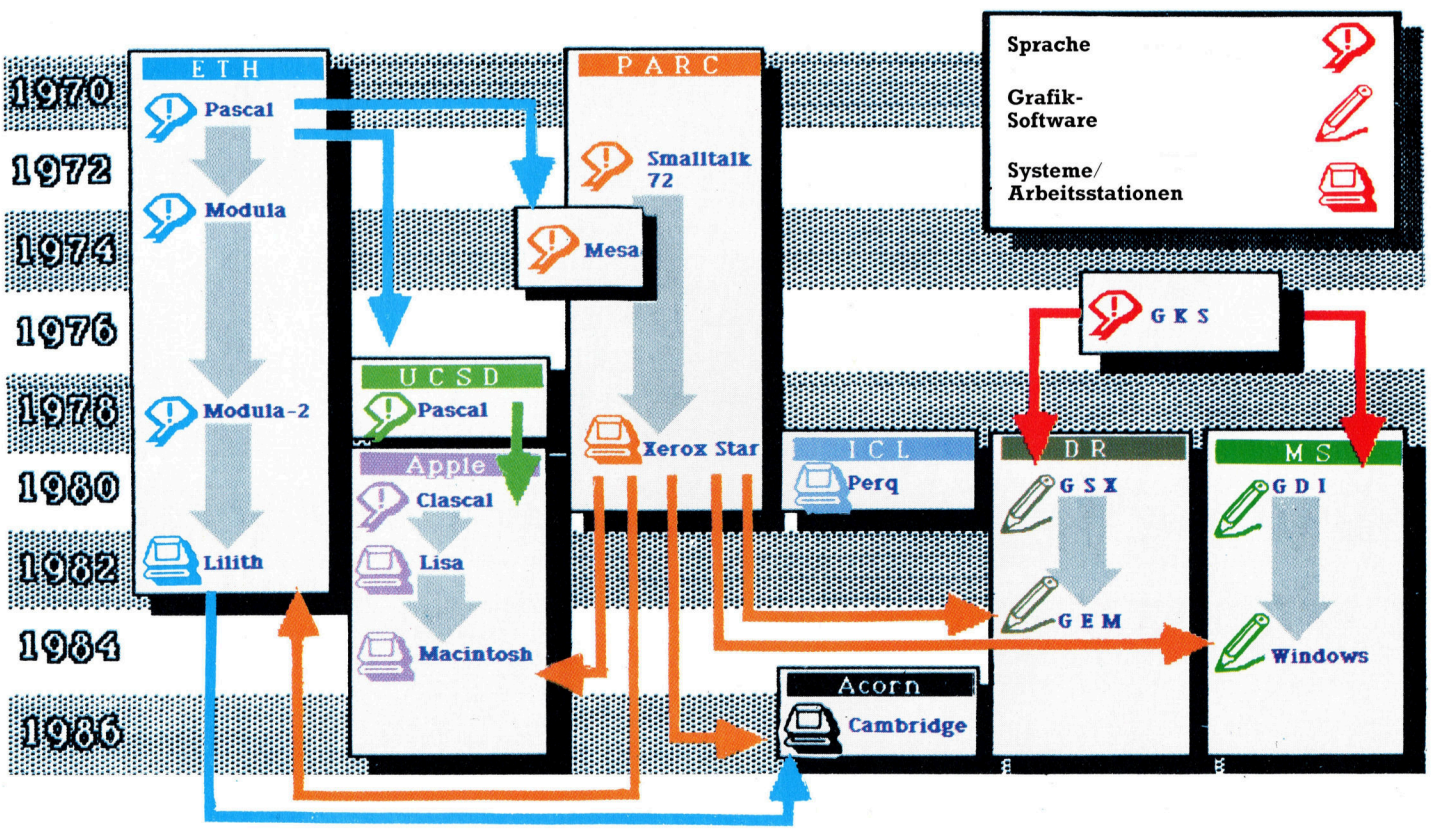
SMALLTALK und seine Nachfolger GEM, Macintosh und MS-Window stellen gewaltige Anforderungen an die Hardware. Wie schon in der letzten Folge erwähnt, sollten die Entwicklungsingenieure des von Xerox gegründeten PARC Forschungszentrums Maschinen für diese neuen Systeme konstruieren.

Der Xerox Star wurde 1980 vorgestellt. Er basierte auf der Forschung im Bereich Schnittstelle Mensch-Maschine (MMI) und der objektorientierten Umgebung von SMALLTALK. Die Eingabe wurde über eine Maus mit drei Knöpfen ausgeführt, und er arbeitete mit Fenstern und Piktogrammen auf einem hochauflösenden Bildschirm, der im Bit-Map-Verfahren gesteuert wurde. Aufgrund fehlender Software und des hohen Preises hatte der Star jedoch nie Erfolg.

Das Gerät machte dennoch die Computerwelt auf die Möglichkeiten von WIMP aufmerksam. Die Firma ICL baute daraufhin in England die Perq Arbeitsstation, die viele Eigenschaften des Star kopierte. Perq war auf den wissenschaftlichen und akademischen Bereich ausgerichtet und verfügte speziell über CAD-Grafik und die Fähigkeit, Text und Grafik mischen zu können. Das System hatte einen hochauflösenden, monochromen Bildschirm, arbeitete mit Fenstern, Menüs und Zeigern und kostete etwa 40 000 Mark. Das Gerät wurde in England etwa 2500mal verkauft (hauptsächlich wegen seiner Netzwerkkompatibilität mit den Großcomputern von ICL) und machte sich damit bezahlt.

Apples Lisa war jedoch der erste Computer, der die WIMP-Technologie 1983 ins Licht der Weltöffentlichkeit rückte. Die Lisa war im Diskettenzugriff sehr langsam, und es gab nur wenig Software für das Gerät. Die Markteinführung wurde außerdem von der Vorstellung des IBM PC kommerziell „überschattet".

Für die Grafiksteuerung von Piktogrammen, Menüs und Fenstern ist eine Bildschirmanzeige mit Hochgeschwindigkeit und im Bit-Map-Verfahren unbedingt notwendig. Das Zentrum von GEM bildet daher ein System, das Bilder sehr schnell über den Schirm bewegen kann. Obwohl SMALLTALK mit einem Interpre-



ter und nicht mit einem Compiler arbeitete, mußten die Grafikroutinen dennoch extrem effektiv sein. Doch selbst ohne die Grafiksteuerung ist die Verwaltung von sich überlagernden Fenstern und die Aktualisierung der sich ständig verändernden Inhalte nicht gerade einfach.

Der Ablauf mehrerer gleichzeitig aktiver Programme (durch Timesharing der CPU simuliert) verringert die für den Bildschirmaufbau zur Verfügung stehende Zeit nochmals. Zwar können moderne 16/32-Bit-Prozessoren wie der Motorola 68000 (im Macintosh, Atari 520 ST und Amiga eingesetzt) Geschwindigkeiten von etwa zwei mips (Millionen Instruktionen pro Sekunde) erreichen, doch stellt allein die Menge des Programmcodes für die Bildschirmsteuerung schon hohe Anforderungen. Dabei mußten neue Techniken entwickelt werden, die mit den Betriebssystemen mithalten konnten.

Hardware-Algorithmen

Um überhaupt vertretbare Ausführzeiten zu erreichen, wurden „Hardware-Algorithmen" eingebaut bzw. Spezialchips eingesetzt, die der CPU die umfangreichen Bitmanipulationen abnahmen. Jetzt hatte die CPU genug Zeit für die Programmausführung und brauchte ihre wertvollen Kapazitäten nicht für die Bildschirmsteuerung einzusetzen. Vielleicht das beste Beispiel für diese Methode ist der Amiga von Commodore.

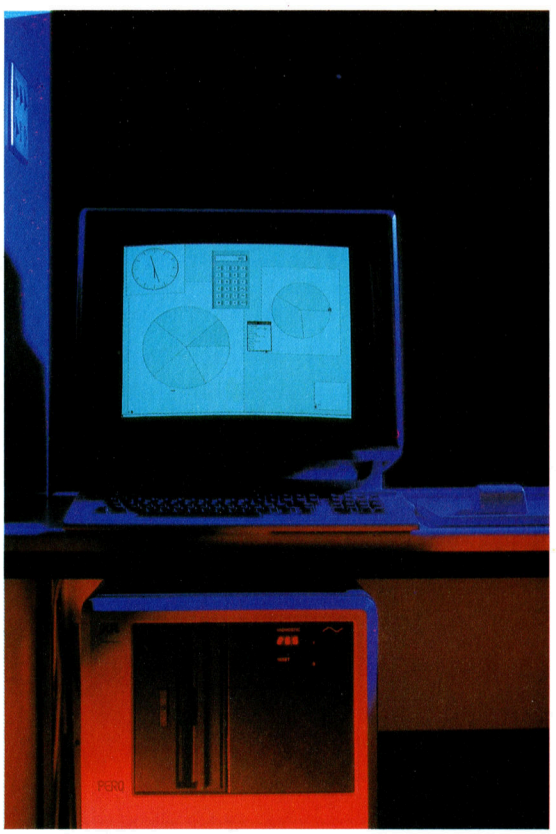
Der Amiga besitzt einen 68000, an den drei Spezialchips zur Steuerung der E/A-Kanäle angeschlossen sind. Diese Chips – Portia, Agnes und Daphne genannt – beeinflussen die beeindruckende Leistung der Maschine wesentlich. Eine hochauflösende Anzeige kann beispielsweise in einer Sekunde (mit beigeordneten Klängen) so oft aktualisiert werden, daß ein schneller, gleitender Bewegungsablauf entsteht.

Obwohl moderne Konstruktionstechniken viel dazu beigetragen haben, daß sich Peripheriechips dieser Art leicht einsetzen lassen, liegt der Ursprung der Algorithmen in der Vorarbeit von Xerox. Der entsprechende SMALLTALK-Algorithmus hieß BITBLT („BIT Block Transfer") und wurde oft als „bit-blit" oder „blitter" bezeichnet. BITBLT steuert seine Bilder nicht, indem er mit einem Zeichenalgorithmus ständig den Bildschirmspeicher ändert, sondern arbeitet unabhängig vom Hauptprozessor direkt mit dem Bildschirmspeicher. Dabei „fragt" er die Speicherstellen auf ähnliche Weise ab, wie ein Elektronenstrahl den VDU-Schirm abtastet.

Seine hohe Geschwindigkeit erreicht BITBLT jedoch nicht nur durch das Abtasten des Speichers, sondern durch die Verarbeitung von Bildschirmbytes statt Bits. Auf den ersten Blick scheint das eine Einschränkung zu sein: Wie lassen sich einzelne Bits ändern, wenn nur Gruppen von acht Bits angesprochen werden können?

Blitter lösen dieses Problem, indem sie mit AND, OR und XOR in Hochgeschwindigkeit bitweise Logikoperationen am Bildschirmspeicher vornehmen. Der Algorithmus kann eine Reihe von Bytemasken erzeugen, die über die bestehende Anzeige gelegt werden. Diese logischen Vorgänge sind extrem schnell. Besonders XOR hat die Fähigkeit, überschriebene Hintergründe wiederherstellen zu können.

Der Star von Xerox bietet eine Reihe von Fähigkeiten, die aus den Konzepten von SMALLTALK entwickelt wurden. Das System bietet verschiedene Schriftarten in mehreren Größen und Grafikeigenschaften, die in einer WIMP-Umgebung laufen.



Der Perq von ICL war der erste englische Computer, der mit den ursprünglich von Xerox entwickelten Techniken arbeitete. Die hochauflösende Anzeige besitzt außergewöhnliche grafische Eigenschaften. Die Maschine wurde hauptsächlich an Forschungsinstitute verkauft. Einer der ersten Anwender war das Institut für Künstliche Intelligenz an der Edinburgh University, das die grafischen Fähigkeiten des Perq unter anderem für Forschung auf dem Gebiet der Sichtmöglichkeiten von Robotern einsetzte.

Die Neue Welt II

Wir behandeln die notwendigen Änderungen, um das Programm auf einem Acorn B zu starten.

Unser Simulationsspiel ist in BASIC geschrieben, so daß es mit geringfügigen Änderungen auch auf dem Acorn B lauffähig ist. Die Hauptänderungen betreffen den Befehl zum Löschen des Bildschirms und die Routine zum Abfragen der Tastatur. An den Stellen des Listings, an denen der Befehl PRINT CHR$(147) steht, sollten Acorn-B-Benutzer den Befehl CLS einsetzen.

Schleife initialisieren

In der hier gezeigten Commodore-64-Version wird eine Schleife initialisiert, die mit GET Zeichen von der Tastatur liest. Sie bricht ab, sobald ein anderes Zeichen als ein Nullstring übermittelt wird. Diese Konstruktion taucht an verschiedenen Stellen auf:

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

und ist zu ersetzen. Das Acorn-B-BASIC verfügt über einen speziellen Befehl mit derselben Wirkung:

<Zeile #> I$=GET$

Jeder Darstellungs-Modus des Acorn B hat einen anderen Speicherplatzbedarf. Damit die Maximalmenge an BASIC-Speicher für das Programm zur Verfügung steht, sollten Sie beim Acorn B Modus 7 verwenden. Dieser Modus ist beim Einschalten des Computers automatisch aktiviert. Um Fehler zu vermeiden, sollte folgende Zeile eingefügt werden:

5 MODE 7

Bald kann das große Abenteuer beginnen. Viel Spaß!

```
5300 REM END OF WEEK REPORT
5305 PRINTCHR$(147)
5310 S$="           CAPTAINS LOG*":GOSUB9100
5312 S$="           -------- ---*":GOSUB9100
5314 GOSUB9200
5316 PRINT:PRINT"          END OF WEEK";WK
5318 GOSUB9200
```

Diese Schleife überprüft, ob während der Woche Mannschaftsmitglieder gestorben sind, und reduziert die Anzahl CN.

```
5320 X=0
5325 FORT=1TO16
5330 IFTS(T,2)<>-999THEN5350
5335 X=X+1
5340 TS(T,1)=0:TS(T,2)=0
5345 CN=CN-1
5350 NEXT
5355 IFX=0THEN5400
5358 PRINT:PRINT
5360 S$="DURING THE PAST WEEK*":GOSUB9100
```

Ist die gesamte Mannschaft gestorben (CN=0), so ist das Spiel zu Ende.

```
5365 IFCN>0THEN5390
5367 S$="ALL THE REMAINING CREW DIED*":GOSUB9100
5369 GOSUB9200:PRINT:GOSUB9200
5373 S$="YOUR SHIP DRIFTS HELPLESSLY*":GOSUB9100
5374 GOSUB9200
5376 S$="ON THE EMPTY OCEAN.....*":GOSUB9100
5378 GOSUB9200
5380 PRINT:PRINT
5382 S$="       GAME OVER*":GOSUB9100
5384 PRINT:PRINT
5386 GOSUB9200:END
5388 GOTO5386
5390 PRINTX;
5392 IF X=1 THEN S$="CREW MEMBER HAS DIED*"
5394 IFX<>1 THEN S$="CREW MEMBERS HAVE DIED*"
5396 GOSUB9100
5398 GOSUB9200
5400 PRINT:PRINT
```

Sind irgendwelche Vorräte ausgegangen? Eine entsprechende Meldung wird ausgegeben.

```
5405 S$="YOU HAVE NOW RUN OUT OF "
5410 FOR T=1TO4
5415 IFPA(T)<>-999THEN5440
5420 PRINTS$;P$(T)
5425 PA(T)=0
5428 S$="                    AND "
5440 NEXT
5450 GOSUB9200
5455 REM CALC NEW JL
```

Die Stärke der Mannschaft wird über das Array TS (,) ermittelt. Sinkt sie unter halbe Maximalstärke, wird die Reisezeit erhöht.

```
5460 X=0
5465 FORT=1TO16
5470 X=X+TS(T,2)
5475 NEXT
5480 IFX>799THEN5494
5481 PRINT:PRINT
5482 S$="THE CREW IS BELOW FULL STRENGTH*":GOSUB91
00
5483 S$="SO JOURNEY MIGHT TAKE LONGER*":GOSUB9100
5489 EW=EW+(800-X)/800
5490 REM JL=JL+INT(EW)
5491 JL = 8 + INT(EW)
5492 PRINT
5494 S$=K$:GOSUB9100
5496 GETI$:IFI$=""THEN5496
5499 RETURN
```

Um während der Fahrt unvorhersehbare Ereignisse eintreten zu lassen, wird eine Zufallszahl zum Aufruf einer Routine verwendet.

```
5500 REM RANDOM EVENT GENERATOR
5502 IFRC=RMTHENRETURN
5503 REM EXIT IF ALL RANDOM EVENTS DONE
5504 PRINTCHR$(147)
5505 GOSUB9200
5510 X=INT(RND(1)*RM)+1
5515 REM GENERATE RANDOM NO. BETWEEN 1 AND RM
5520 IFRR(X)=1THENRETURN
5522 REM TRY AGAIN IF THIS EVENT ALREADY DONE
5523 RR(X)=1:RC=RC+1
5524 REM SET IND. TO SHOW THIS EVENT DONE AND INCR
EMENT EVENT COUNT
5525 ON X GOTO 5540,5570,5570,5570,5570,5600,5700,
5800,5850,5900,5950,6000,6050
5528 REM GO TO APPROPRIATE CODE FOR THIS EVENT
5530 PRINT:S$=K$:GOSUB9100
5535 GETI$:IFI$=""THEN5535
5539 RETURN: REM RETURN TO MAIN JOURNEY LOOP
```

Es gibt negative Ereignisse ...

```
5540 REM EVENT 1 - MAN OVERBOARD!
5542 PRINT
5543 S$="  DURING THE WEEK*":GOSUB9100
5545 PRINT:GOSUB9200
5546 S$=" 1 PERSON WAS LOST OVERBOARD*":GOSUB9100
5548 S$=" IN A STORM*":GOSUB9100
5550 PRINT:GOSUB9200
5552 S$="YOUR CREW IS NOW REDUCED TO*":GOSUB9100
5554 PRINTCN-1;"MEMBERS"
5558 FORT=1TO16
5559 REM SEARCH FOR CREW MEMBER TO LOSE
5560 IFTS(T,2)=0ORTS(T,2)=-999THEN5566
5562 TS(T,2)=-999:REM DEAD
5564 T=16
5566 NEXT
5568 GOTO5530
5570 REM EVENTS 2 TO 5 - PROVISIONS LOST
5572 X=X-1:REM X NOW POINTS TO PROVISION(1-4)
5574 IFPA(X)=0 OR PA(X)=-999 THENRETURN
5576 REM NO ACTION IF THIS PROVISION ALREADY EXHAU
STED
5578 PRINT
5580 S$="  DURING THE WEEK*":GOSUB9100
5582 PRINT:GOSUB9200
5584 PRINT" SOME OF YOUR ";P$(X)
5586 S$=" WAS WASHED OVERBOARD*":GOSUB9100
5588 PRINT:GOSUB9200
5590 S$="YOU NOW HAVE APPROXIMATELY*":GOSUB9100
5592 PA(X)=PA(X)-INT(PA(X)/(INT(RND(1)*3)+2))
5593 REM REDUCE PROV AMOUNT BY 1/2 1/3 OR1/4
5594 PRINTINT(PA(X)/(CN*PN(X)));
5595 PRINT"WEEKS ";P$(X);" LEFT"
5599 GOTO5530
```

... aber auch gute. So kann etwa die Fleischration mit Fisch aufgebessert werden.

```
5600 REM EVENT 6 - CATCH SOME FISH
5605 X=0
5610 FORT=1TO16
5615 IFTS(T,2)=-999THENX=X+1
5620 REM COUNT DEAD THIS WEEK
5625 NEXT
5630 IFCN-X<1THENRETURN
5635 REM NO ACTION IF ALL CREW DEAD
5640 PRINT
5645 S$="  DURING THE WEEK*":GOSUB9100
5646 PRINT:GOSUB9200
5650 S$="ONE OF THE CREW CAUGHT*":GOSUB9100
5655 X=INT(RND(1)*10)+11
5660 REM 10 TO 20 KILOS
5662 PRINTX;"KILOS OF FISH"
5665 PRINT:GOSUB9200
5670 S$="YOUR MEAT SUPPLY IS NOW*":GOSUB9100
5675 S$="ENOUGH FOR APPROXIMATELY*":GOSUB9100
5678 IFPA(3)=-999THENPA(3)=0
5680 PA(3)=PA(3)+X
5685 PRINTINT(PA(3)/(CN*PN(3)));"WEEKS"
5690 GOTO5530
```

Ist nur noch wenig Wasser an Bord, kann die folgende Routine weiterhelfen. In Zeile 5735 wird eine Zufallszahl zwischen 10 und 20 generiert. Dieser Wert entspricht der Wassermenge, die zu PA(4) addiert wird.

```
5700 REM EVENT 7 - CATCH SOME WATER
5705 PRINT
5710 S$="  DURING THE WEEK*":GOSUB9100
5715 PRINT:GOSUB9200
5720 S$="A RAINSTORM REFILLED YOUR*":GOSUB9100
5725 S$="WATER BARRELS*":GOSUB9100
5730 PRINT:GOSUB9200
5735 X=INT(RND(1)*10)+11
5736 REM 10 TO 20 BARRELS
5740 S$="YOUR WATER SUPPLY IS NOW*":GOSUB9100
5745 S$="ENOUGH FOR APPROXIMATELY*":GOSUB9100
5748 IFPA(4)=-999THENPA(4)=0
5750 PA(4)=PA(4)+X
5755 PRINTINT(PA(4)/(CN*PN(4)));"WEEKS"
5760 GOTO5530
5800 REM EVENT 8 - GOOD WINDS
5805 PRINT
5810 S$="STRONG FOLLOWING WINDS ALL WEEK*":GOSUB91
00
5815 PRINT:GOSUB9200
5820 S$="YOU HAVE MADE GOOD SPEED*":GOSUB9100
5825 S$="AND YOUR JOURNEY TIME IS*":GOSUB9100
5830 S$="REDUCED BY 1/2 A WEEK*":GOSUB9100
5835 EW=EW-.5
5839 GOTO5530
```

Gutes Wetter kann die Arbeitsmoral verbessern. Das entsprechende Array – TS(,) – wird durchsucht und jeder Wert um eine Zahl zwischen 5 und 15 erhöht.

```
5850 REM EVENT 9 - GOOD WEATHER
5855 PRINT
5860 S$="GOOD WEATHER ALL WEEK*":GOSUB9100
5865 PRINT:GOSUB9200
5870 S$="THE CREW IS FEELING HAPPIER*":GOSUB9100
5875 GOSUB9200
5880 S$="                AND HEALTHIER!*":GOSUB9100
5882 FORT=1TO16
5884 IFTS(T,2)=0 OR TS(T,2)=-999THEN5888
5886 TS(T,2)=TS(T,2)+INT(RND(1)*11)+5
5888 NEXT
5889 GOTO5530
```

Die nächsten drei Routinen betreffen die Ladung. Waffen können verrosten, Kleidung durch Mäuse angefressen werden, und Medizin-Flaschen zerbrechen.

```
5900 REM EVENT 10 - LOSE MEDICINE
5905 IF OA(1)=0 OR OA(1)=-999THENRETURN
5910 PRINT
5915 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
5920 S$="MEDICINE BOTTLES HAVE BROKEN*":GOSUB9100
5925 OA(1)=INT(OA(1)/2)
5930 PRINT:GOSUB9200
5935 S$="YOU NOW HAVE ONLY*":GOSUB9100
5940 PRINTOA(1);"BOTTLES LEFT"
5945 GOTO5530
5950 REM EVENT 11 - RUSTY GUNS
5955 IF OA(2)=0 OR OA(2)=-999THENRETURN
5960 PRINT
5965 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
5970 S$="GUNS HAVE GONE RUSTY*":GOSUB9100
5972 S$="AND NO LONGER WORK*":GOSUB9100
5975 OA(2)=INT(OA(2)/2)
5980 PRINT:GOSUB9200
5985 S$="YOU NOW HAVE ONLY*":GOSUB9100
5990 PRINTOA(2);"GUNS LEFT"
5995 GOTO5530
6000 REM EVENT 12 - LOSE CLOTH
6005 IF OA(4)=0 OR OA(4)=-999THENRETURN
6010 PRINT
6015 S$="YOU DISCOVER THAT HALF YOUR*":GOSUB9100
6020 S$="BALES OF CLOTH HAVE BEEN*":GOSUB9100
6022 S$="GNAWED BY MICE*":GOSUB9100
6024 S$="AND ARE NOW WORTHLESS*":GOSUB9100
6025 OA(4)=INT(OA(4)/2)
6030 PRINT:GOSUB9200
6035 S$="YOU NOW HAVE ONLY*":GOSUB9100
6040 PRINTOA(4);"BALES LEFT"
6045 GOTO5530
6050 REM EVENT 13 TO     - ALBATROSS
6055 PRINT:A$="Y"
6060 S$="AN ALBATROSS FLIES OVER THE SHIP*":GOSUB9
100
6062 GOSUB9200
6065 S$="THIS IS A GOOD OMEN*":GOSUB9100
6068 S$="AND THE CREW IS HAPPY*":GOSUB9100
6070 PRINT:GOSUB9200
6075 IFPA(3)<(CN*PN(3)*(JL-WK+1))THEN6090
6080 REM NOT SHORT OF MEAT
6085 GOTO6122
6090 S$="YOU'RE RUNNING SHORT OF MEAT*":GOSUB9100
6095 S$="AND THE BIRD WEIGHS 10 KILOS!*":GOSUB9100
6100 PRINT:GOSUB9200
6105 S$="WOULD YOU LIKE TO CATCH IT?*":GOSUB9100
6110 INPUTI$
6112 PRINT:GOSUB9200
6115 IFLEFT$(I$,1)="Y"THEN6133
6120 S$="PROBABLY JUST AS WELL!*":GOSUB9100
6122 PRINT:GOSUB9200
6125 S$="THE ALBATROSS FLIES AWAY....*":GOSUB9100
6130 GOTO5530
6133 IFOA(2)=0OROA(2)=-999THEN6180
6135 S$="A SHOT IS FIRED.............*":GOSUB9100
6138 GOSUB9200:GOSUB9200
6140 IFRND(1)<.5THEN6150
6145 S$=".........BUT MISSES!*":GOSUB9100
6148 GOTO6122
6150 S$="AND THE BIRD FALLS TO THE DECK!*":GOSUB91
00
6155 PRINT:GOSUB9200
6160 IFPA(3)=-999THENPA(3)=0
6162 PA(3)=PA(3)+10:B$="Y"
6165 S$="YOU NOW HAVE 10 MORE KILOS*":GOSUB9100
6167 S$="OF MEAT......*":GOSUB9100
6170 S$="BUT YOU MAY NOT HAVE MUCH*":GOSUB9100
6172 S$="GOOD LUCK FROM NOW ON!!*":GOSUB9100
6174 GOTO5530
6180 S$="YOU CAN'T - YOU HAVE NO GUNS*":GOSUB9100
6190 GOTO6122
```

Die bisherigen Ereignisse haben geringfügige Auswirkungen auf die Fahrt, doch es gibt sechs wichtige Begebenheiten, die hier per Zufall ausgewählt werden.

```
6500 REM MAJOR CONTINGENCIES
6508 X=INT(RND(1)*10)+1
6510 ON X GOSUB6530,6700,6800,6900,7000,7050
6520 RETURN
```

Auf Nummer Sicher

Wollen Sie Ihre Programmiertricks vor neugierigen Augen schützen? Oder vielleicht einem Programm den gewissen professionellen „Touch“ verleihen? Wir zeigen Ihnen, wie das geht.

Wenn ein Programm erst einmal läuft, kann man daran gehen, ihm auch ein etwas professionelleres Aussehen zu verschaffen. Es geht aber nicht nur um das gute Erscheinungsbild, sondern auch darum, nicht jeden Anwender gleich mit der Nase darauf zu stoßen, welche Tricks Sie beim Programmieren benutzt haben.

Der beste Schutz für ein BASIC-Programm vor unerlaubtem Kopieren sind eingebaute „Fallen“. Maschinenprogramme lassen sich mit raffinierteren Methoden schützen, grundsätzlich können aber die Techniken zum Schutz eines BASIC-Programms auch auf Maschinenprogramme übertragen werden.

Welche Tricks Sie nun anwenden, um einen „Piraten“ am Kopieren oder auch nur LISTen zu hindern, hängt natürlich von Ihren Anforderungen ab. Für sehr einfache Programme sollten Sie den Aufwand allerdings nicht übertreiben; denn: Den absolut sicheren Kopierschutz gibt es nicht. Viele werden Ihre Schutzversuche nur als zusätzliche Herausforderung betrachten und sofort versuchen, Sie zu überlisten. Oft wollen Anwender Programme für eigene Zwecke modifizieren. Und so mancher wird einfach nur wissen wollen, wie Sie ein spezielles Problem gelöst haben.

Achtung Falle!

Schon ganz einfache Fallen können ein Programm schützen. Natürlich kann ein Fachmann sie umgehen, aber auch er hat seine Mühe damit. Mühsam ist es allerdings auch für den, der das Programm schreibt, besonders dann, wenn die Schutzmaßnahmen schon eingegeben und auch aktiv sind. Die Fehlersuche kann dadurch überaus mühsam werden. Also: Nicht in die eigene Falle gehen!

Bei diesen einfachen Methoden werden die üblichen SAVE-, LIST- und anderen Befehle außer Betrieb gesetzt. Das geschieht aber erst, wenn das Programm gestartet wird. Das kann ein Vorteil sein – die Fehlersuche hat man bei einem funktionierenden Programm ja bereits hinter sich. Allerdings kann auch der Programm-Pirat vor dem RUN schon etwas sehr Wesentliches herausfinden . . .

Besseren Schutz bietet ein Programm, das sich nach dem Laden selbsttätig startet. Beim Spectrum und Acorn geht das, wie wir sehen werden, mit den entsprechenden SAVE- und LOAD-Befehlen. Beim Commodore führt Shift-Run/Stop zum automatischen Start des ersten Programms auf der Cassette.

Diese Befehle werden normalerweise im Direktmodus eingegeben. Sie können aber auch in einem separaten Programm stehen, das vorab geladen wird. Ein solches Programm nennt man „Bootstrap“. Je nach seinen Aufgaben kann es in BASIC oder Maschinencode geschrieben werden. In der einfachsten Form lautet es so:

```
10 LOAD"Name des naechsten Programms".
```

Dieser Einzeiler lädt einfach das Programm, das in Anführungszeichen steht.

Natürlich ist eine solche einzelne Zeile sinnlos. In der Praxis macht ein Bootstrap darum auch bedeutend mehr: Häufige Anwendungen sind die Ausgabe des Titelfensters und Copyrights. Bootstraps erzeugen aber auch Bedienungshinweise, legen Variablen an – und enthalten nicht zuletzt die ausgefeiltesten Vorkehrungen für den Programmschutz.

Dazu gehören einige der wirkungsvollsten Techniken, mit denen ein BASIC-Programm geschützt werden kann – die Veränderung von Systembefehlen. Die wahre Stärke von Bootstraps liegt aber in der Fähigkeit, ein Maschinenprogramm vom BASIC aus zu starten.

Bootstraps werden häufig in kommerziellen Programmen genutzt: Diese bestehen oft aus vielen Teilprogrammen, die bei Bedarf unter der Aufsicht eines oder mehrerer Bootstraps nachgeladen werden. In BASIC führt das normalerweise zum Überschreiben des ladenden Programms. Wenn Sie diese Technik also selbst anwenden wollen, müssen alle Maschinenprogramme zuerst geladen werden und an einen „sicheren“ Platz gebracht worden sein.

Bei Programmen, die aus mehreren Modulen bestehen, kann der Programmschutz auch auf einer gegenseitigen Abhängigkeit einzelner Module aufbauen. Dabei prüft ein Modul die Daten eines anderen. Fehlt etwas, kommt es zum Systemabsturz oder

einem sonstigen Fehler im Ablauf, der eine problemlose Verwendung dieses speziellen Sicherungsprogramms unmöglich macht..

Wie kann man nun ein Bootstrap einsetzen, um ein anderes Programm zu starten? Für Tandy- oder Dragon-Rechner ist das nicht ganz einfach, da hier spezielle Maschinenroutinen aufgerufen werden müssen, die von BASIC aus nicht zugänglich sind.

Bei einem Spectrum-Rechner (zu Acorn kommen wir noch) ist es einfacher: Innerhalb des Bootstraps muß nur der richtige Befehl benutzt werden. Eine einzige Zeile bringt Sie schon an das Ziel Ihrer Bemühungen und den ungebetenen Programmpiraten an den Rand der Verzweiflung.

Spectrum

```
990 LOAD "Naechstes Programm"
```

Dieses Programm muß aber zuvor mit dem Selbststartbefehl SAVE „Naechstes Programm" LINE 1 gespeichert werden (für „LINE 1" wird die erste Programmzeile eingesetzt). Bei einer höheren Zeilennummer könnte man sogar noch ein Copyright oder Maschinencode einfügen, der zur Sicherheitsprüfung ganz einfach aus REM-Zeilen gePEEKt wird.

Commodore

Das selbsttätige Starten eines Commodore-Programms läßt sich nur dadurch bewerkstelligen, daß der LOAD- und RUN-Befehl in den Tastaturbuffer geschafft werden, um von dort die Betätigung dieser Tasten zu simulieren und die Steuerung vom Bootstrap zurück zum BASIC zu übergeben.

Genau das können Sie mit dem folgenden Programm erreichen. Es ist um die Commodore-System-Routine „vector" aufgebaut: Ein Bytepaar gibt dem Betriebssystem an, welchen Teil des eigenen Codes es für einen speziellen Befehl nutzen soll. Das erste Listing gilt für den Commodore 64:

```
10 N$="NAME"
20 POKE 49189, LEN(N$)
30 FOR Z=1 TO LEN(N$)
40 POKE 49189+Z, ASC
   (MID$(N$,Z,1))
50 NEXT Z
60 FOR Z=679 TO 736
70 READ X
80 POKE Z,X
90 NEXT Z
100 FOR Z=49152 TO 49188
110 READ X
120 POKE Z,X
130 NEXT Z
200 POKE 770,167: POKE 771,2:
   SYS49152
210 PRINT "OKAY": GOTO210
220 DATA 169,47,133,0,169,55,133,1,32,
   138,255,169,1,141,32,208
230 DATA 169,48,141,119,2,169,76,141,
   120,2,169,207,141,121,2
240 DATA 169,13,141,122,2,169,82,141,
   123,2,169,213,141,124,2
250 DATA 169,13,141,125,2,169,7,133,
   198,108,0,160
260 DATA 162,1,160,1,169,1,32,186,255,
   162,38,160,192,173,37,192
270 DATA 32,189,255,169,167,133, 251,
   169,2,133,252,162,5,160,3,
   169,251
280 DATA 32,216,255,96
```

Dieses Programm läuft auf dem VC 20 ohne Erweiterung:

```
10 N$="NAME"
20 POKE 7205, LEN(N$)
30 FOR Z=1 TO LEN(N$)
40 POKE 7205+Z, ASC
   (MID$(N$,Z,1))
50 NEXT Z
60 FOR Z=679 TO 723
70 READ X
80 POKE Z,X
90 NEXT Z
100 FOR Z=7168 TO 7204
110 READ X
120 POKE Z,X
130 NEXT Z
200 POKE 770,167: POKE 771,2: SYS7168
210 PRINT "OKAY": GOTO210
220 DATA 32,135,255
230 DATA 169,48,141,119,2,169,76,141,
   120,2,169,207,141,121,2
240 DATA 169,13,141,122,2,169,82,141,
   123,2,169,213,141,124,2
250 DATA 169,13,141,125,2,169,7,133,
   198,108,0,192
260 DATA 162,1,160,1,169,1,32,186,255,
   162,38,160,28,173,37,28
270 DATA 32,189,255,169,167,133,251,
   169,2,133,252,162,5,160,3,169,251
280 DATA 32,216,255,96
```

Dieses Programm sollten Sie für späteren Gebrauch SAVEn. Danach wird es wieder geladen und der Name des Programms, das selbsttätig starten soll, in Zeile 10 eingegeben. Wenn Sie mit Cassetten arbeiten, muß dieses Programm nach dem Bootstrap geSAVEt werden. Durch das Starten des Bootstrap-Programms wird es in den Speicher übertragen. Wenn es danach wieder auf Cassette abgespeichert wird, sollte das Bootstrap-Programm den Namen des selbststartenden Programms tragen. Nach der Anzeige von OKAY die Tasten RUN/STOP und RESTORE drücken, um das Programm abzubrechen. Nun das zweite Programm laden und mit PEEK die Speicherplätze 45 und 46 prüfen (hier beginnen die Variablen). An den Anfang des zweiten Programms wird folgende Zeile gesetzt:

```
0 POKE 45,X: POKE 46,Y
```

X und Y sind die Werte, die Sie durch PEEK abgefragt haben. Sie sollten die Zeile mit RETURN eingeben und im Direktmodus PEEK wiederholen, um nachzusehen, ob sich die Werte geändert haben. Passen Sie Zeile 0 nun an die neuen Zahlen an. Dieser Vorgang wird so oft wiederholt, bis sich die Werte nicht mehr ändern.

Jetzt kann das zweite Programm geSAVEt werden. Davor können Sie aber auch noch andere Sicherheitsvorkehrungen einbauen, etwa um POKEs zu verhindern.

Wir geben Ihnen etwas später in diesem Artikel noch einige wertvolle Tips, an denen sich Programmpiraten die Zähne ausbeißen dürften.

Acorn B

Für LOAD und RUN haben die Acorn-Computer einen speziellen Befehl, der vom Bootstrap in dieser Form genutzt werden kann:

```
990 CHAIN "NAME DES FOLGENDEN
    PROGRAMMS"
```

Das ist bereits alles! Andere Programme können also ganz einfach durch eine entsprechende Zeile im vorhergehenden Programm aufgerufen werden.

Selbsttätiger Programmstart allein reicht aber noch nicht. Sie müssen auch dafür sorgen, daß ein Programm weder angehalten noch geändert und auch nicht geLISTet werden kann.

Dazu wird das System so verändert,

daß es auf den Aufruf der Befehle, wie etwa auf LIST, anders als gewohnt reagiert.

Jeder Computer wird mit einem Betriebssystem und, jedenfalls in unserem Bereich, auch mit einem BASIC-Interpreter geliefert. Die meisten Systeminformationen befinden sich im ROM, einige werden beim Einschalten aber auch in das RAM des Computers geladen. Diese Werte können verändert und damit das Verhalten des Rechners durch den Programmierer umgestaltet werden.

Wenn man sich in dieses „Innenleben" eines Rechners vorwagt, braucht man in jedem Fall eine Tabelle der Systemvariablen und Betriebssystem-Routinen. Wer noch weitergehen möchte, muß sich zusätzlich Informationen über die Speicheraufteilung seines Rechners verschaffen.

Beim Einschalten des Computers wird ein Teil der Systemdaten vom ROM ins RAM geladen, um bestimmte Variablen veränderlich zu machen. Sobald Daten jedoch im RAM stehen, kann der Programmierer auf sie zugreifen und sie für seine speziellen Zwecke nutzen.

Ein besonderer Typ von Systemvariablen ist der RAM-„Pointer" oder „Vector" (Zeiger). Normalerweise besteht er aus zwei benachbarten Speicherplätzen, in denen die Adresse eines Systemablaufs festgehalten ist. Bei Änderung dieser Adresse wird das Betriebssystem mit jedem Aufruf der Routine fehlgeleitet.

Für uns sind einige Zeiger von besonderem Interesse. Dazu gehören LIST- und SAVE-Zeiger, Interruptzeiger sowie die Zeiger für WARMSTART und RESET.

Es ist unmöglich, einen vollständigen Programmschutz zu erreichen – das gilt besonders dann, wenn Sie sich nicht all zu sehr mit Maschinensprache beschäftigen möchten. Durch die Verschiedenheiten der Betriebssysteme sind die Methoden für jeden Rechner anders. Trotzdem möchten wir Ihnen ein paar Vorschläge machen.

Spectrum

Zu den einfacheren Methoden gehört es, ein unveränderliches Copyright in Ihr Programm einzubauen. Man kann dann mit PEEKs prüfen, ob der Vermerk noch vorhanden ist, und bei jeder Veränderung einen System-Reset erzwingen.

Zuerst müssen Sie die Adresse des BASIC-Speicherbereichs herausfinden. Diese Adresse (die Systemvariable PROG) steht in den Speicherplätzen 23635 und 23636 und kann mit PRINT PEEK 23635 + 256*PEEK 23636 bestimmt werden. Wenn Sie den Wert von PROG kennen, können Sie eine beliebige Zahl N in (PROG+1) POKEn. Zeile N wird dann die erste Programmzeile.

Es wird spannend

Interessant wird es, wenn N auf 0 gesetzt wird – Zeile 0 eines Programms wird man nämlich nicht mehr los. Nehmen wir an, Zeile 10 sähe so aus:

```
10 REM (c) MEIER 1986
```

Geben Sie im Direktmodus ein:

```
POKE (PEEK 23635 + 256*PEEK 23636) + 1,0
```

– das war's: aus Zeile 10 wird Zeile 0. Natürlich kann dieser Ablauf auch von einem Bootstrap-Programm ausgeführt werden.

Bei einem selbststartenden Programm scheint der einfachste Weg zum LISTen der Druck auf die BREAK-Taste zu sein. BREAK erscheint im unteren Bildschirmabschnitt, danach kann das Programm mit LIST angezeigt werden.

Aber was tun, wenn der untere Bildschirmabschnitt die Eingabe nicht annimmt? Wenn Sie sich die Liste der Systemvariablen im Handbuch des Spectrum ansehen, werden Sie sehen, daß DF SZ (Speicherplatz 23659) die Anzahl der Zeilen des unteren Schirmabschnittes (normalerweise 2) enthält. Wenn Sie nun eine 0 in die Adresse 23659 POKEn, stürzt der Rechner bei jedem Eingabeversuch in diesen Abschnitt unweigerlich ab.

Dies kann allerdings nicht im Direktmodus geschehen, sondern muß in ein Programm integriert werden. Dennoch macht ein solcher Eingriff keinerlei Schwierigkeiten, denn er findet in nur drei kurzen Programmzeilen Platz.

```
10 POKE 23659,0
20 PRINT AT 5,5;RND
30 GOTO 20
```

Nach einem RUN stürzt der Rechner bei Druck auf BREAK augenblicklich ab. Wenn Sie diese Technik einsetzen, darf das Programm nicht mit Screen-Meldungen wie INPUT? oder scroll? arbeiten – das würde den gleichen Effekt haben. Zur weiteren Dateneingabe müssen Sie INKEY$-Befehle verwenden.

Aus Hacker-Sicht gibt es einen besonders geeigneten Befehl, um ein Programm vom selbsttätigen Start abzuhalten: Es wird nicht geladen (LOAD), sondern mit MERGE in den Rechner übernommen. Aber auch das kann verhindert werden – „Bytes" können nicht geMERGEt werden. Wenn also ein Programm als CODE gespeichert wird (in Byteform), ist kaum etwas zu machen.

Das Abspeichern von CODE beginnt bei 23552, dem Anfang der Systemvariablen. Die Anzahl der Bytes ist N-23552, wobei N jede Zahl sein kann, die größer als STKEND ist (Startadresse des freien Speicherberei-

ches). Nach dem Handbuch ist diese Adresse PEEK 23653 + 256*PEEK 23654. Falls das gesamte User-RAM zu speichern ist, wird N auf 65535 gesetzt. Das ist die größte Anzahl von Bytes, die auf Cassette übernommen werden kann. Folgende Zeile wird dafür an den Beginn des Programms gesetzt:

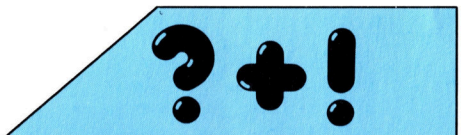
```
1 SAVE "PROGNAME" CODE 23552, N-23552
```

Dann eine zweite Zeile mit POKE einfügen, um das Programm bei Eingabe von BREAK zum Abstürzen zu bringen:

```
2 POKE 23659,0
```

Jetzt geben Sie GOTO 1 ein, und SAVEn Sie das Programm. Durch den Befehl LOAD „NAME DES PROGRAMMS" startet das Programm selbsttätig.

Der normale LOAD-Befehl funktioniert bei sehr großen N-Werten nicht – etwa bei N=65000 im 48K-Rechner. Er verbraucht Speicherplatz sowohl auf der Cassette als auch im Rechner. Weil aber bereits kaum Platz für Ihr eigenes Programm vorhanden ist, paßt auch sonst nichts mehr in den Speicher – ein Cassetten-Kopierprogramm etwa. Damit ist Ihr Programm dann sicher vor Benutzung durch „böse Buben" und Hacker.



Wie können Maschinenprogramme von BASIC aus so geladen werden, daß der Vorgang für den Anwender einfach und problemlos ist?
Mit einem Bootstrap-Programm (siehe unten). Beim Acorn ist die Startadresse N des Maschinenprogramms in Hex anzugeben.

Spectrum

```
10 CLEAR N–1
20 LOAD "Maschinencodefile" CODE
```

Programm mit SAVE „loader name" LINE 10 speichern, damit es selbsttätig startet. Danach das Maschinenprogramm mit SAVE „Maschinencodefile" CODE N,B. Hierbei ist B die Anzahl der Bytes.

Acorn B

```
*LOAD ""N
```

Dragon

```
10 CLEAR 200,N–1
20 CLOADM "MASCHINEN
CODEFILE"
```

Commodore

Programme für den selbsttätigen Start und die Abschaltung der Tastatur sind leicht zu entlarven – es müssen also noch zusätzliche Maßnahmen getroffen werden. Dazu zählt die „back-delete"-Technik, mit der man POKE-Befehle verbergen kann.

Mit diesem Verfahren läßt sich nahezu alles verstecken – sogar die zur Sicherung nötigen Daten!

Die „back-delete"-Methode stützt sich auf DELete-Befehle, die in Anführungszeichen hinter dem nichtaktiven Teil einer REM-Zeile stehen. Hier ein Beispiel, das ohne Leerzeichen eingegeben werden muß:

```
99POKE45,0:POKE46,20:RUN:
REM""SYS4096
```

Bringen Sie nach dem Eintippen den Cursor auf das zweite Anführungszeichen, und geben Sie mit SHIFT und INST/DEL 27 Leerzeichen ein. Danach mit INST/DEL ohne SHIFT 27 × DELete (Löschen) eingeben, als Symbol erscheint dabei ein umgedrehtes T. Zeile mit RETURN abschließen. Als nächstes gehen Sie wieder zum Editieren in die Zeile und löschen das zweite Anführungszeichen. Wieder mit RETURN abschließen. Wenn Sie alles richtig gemacht haben und die Zeile jetzt zu LISTen versuchen, wird nur dies angezeigt:

```
10 SYS4096
```

Versuchen Sie, die Reihe der versteckten DELetes durch Eingabe von weiteren Leerzeichen und DELete-Symbolen zu editieren.

Mit diesem einfachen Verfahren lassen sich ganze Zeilen oder auch nur Teile einer Programmzeile unsichtbar machen.

Sofern diese Zeilen am Anfang und am Schluß eines Listings stehen, treten keine verräterischen Lücken auf – allerdings kann das kurzzeitige Auftauchen der versteckten Zeichen dem Eingeweihten einen Hinweis geben. Außerdem – das Verfahren beeinflußt nur den Bildschirm. Ein schnell gemachter Programmausdruck bringt den Trick ans Tageslicht!

Manchmal reichen schon kleine Hinweise auf das Vorhandensein eines Maschinenprogramms, um ungeübte „Programmdiebe" entsprechend abzuschrecken.

Bei Verwendung eines Bootstrap-Programms gibt es aber neben dem selbsttätigen Start noch mehr Möglichkeiten des Schutzes: Mit einer zusätzlichen Programmzeile im Selbststart-Programm können Sie vom Bootstrap genutzte Speicherplätze durch PEEK verändern. Das führt dazu, daß das Programm nicht ohne Bootstrap geladen oder gestartet werden kann. Falls Sie das Bootstrap-Programm schon vorher verwendet haben, bauen Sie in das zweite Programm folgende Zeile ein.
Auf dem C 64:

```
1 IF PEEK (679) <> 169 OR PEEK (680)
<> 47 THEN NEW
```

Auf dem VC 20:

```
1 IF PEEK (679) <> 32 OR PEEK (680) <>
135 THEN NEW
```

Zusätzlich sollten Sie geeignete Tastaturabschaltungen vornehmen, damit das laufende Programm nicht gestoppt werden kann.

Acorn B

Die besten Möglichkeiten zur Programmsicherung bietet Maschinensprache. Das Problem beim Einsatz eines BASIC-Programms besteht darin, daß alles davon abhängt, ob das Programm einmal gestartet wurde und ob es nun Variablen verändert, den Speicherbereich verschiebt oder beim Druck auf BREAK abstürzt. Der Programmknacker muß es nur vor RUN LISTen, und alle Tricks und Kniffe werden sofort sichtbar. Hier kann ein kurzes Bootstrap-Programm in Maschinensprache – ein „Header" – weiterhelfen. Wenn man jetzt ein LIST versucht, erhält man nur die Bildschirmanzeige „Bad program".

Es gibt also auch eine „wasserdichte" Möglichkeit des Programmschutzes, die wohl jeden – außer Maschinensprache-Fachleute – total abschreckt. Und alles ohne Maschinensprache!

Sie brauchen einfach nur das letzte Byte Ihres Programms zu verändern. Bei BASIC-Programmen ist das immer &FF (Hex). Wenn Sie dieses Byte än-

dern, findet der Rechner beim LIST-Versuch das Programmende nicht mehr und erzeugt darauf die Meldung „Bad program". Damit das Programm beim Ablauf richtig arbeitet, muß aber END in der letzten Zeile stehen. Und so geht's!

In Hex-Form

Zuerst geben Sie diese Zeile ein:

```
PRINT ~ PAGE, ~ TOP
```

Wundern Sie sich nicht über das ~-Zeichen, es erscheint im MODE 7 des Acorn B als ÷ und bedeutet auch das gleiche – eine dezimale Zahl wird in die Hex-Form umgewandelt. Sie werden auf dem Bildschirm zwei Zahlen sehen, nämlich PAGE und TOP (PAGE=Anfang des BASIC-Programms, TOP=Ende des Programms). PAGE ist normalerweise 0E00 (oder 1900, wenn Sie ein Acorn-Disketten-System haben). TOP hängt von der Größe des Programms ab. Nun geben Sie ein:

```
?(TOP–1)=0
```

Damit wird das letzte Byte des Programms auf Null gesetzt. Nun muß das Programm nur noch mit *SAVE abgespeichert werden:

```
*SAVE "Name des Programms" _MMMM_NNNN
```

MMMM und NNNN sind die beiden Zahlen, die Sie vorher für PAGE und TOP ermittelt haben. Damit ist Ihr Programm recht gut geschützt. Wer immer noch an Ihr Programm heran möchte, der muß inzwischen einen gehörigen Aufwand treiben.

Dragon

Um ein BASIC-Programm zum selbsttätigen Start zu bewegen, brauchen Sie beim Dragon eine etwas kniffelige Maschinensprach-Routine für das Speichern auf Cassette.

Einfacher ist es, das Programm als Maschinen-Code mit CSAVEM zu speichern und ein Mini-Maschinenprogramm einzubauen, das die BASIC-Zeiger wieder richtig einstellt. Dann läuft das Programm, ohne daß der Rechner vorher wieder zum direkten Eingabemodus zurückkehrt. Die zu verstellenden BASIC-Zeiger sind zuerst einmal die Startadresse des Programms, die Sie so herausfinden:

```
PEEK(25)*256 + PEEK(26)
```

Die Endadresse bekommen Sie auf ähnliche Weise:

```
PEEK(27)*256 + PEEK(28)
```

Ein weiterer wichtiger Zeiger, der das LISTen des Programms verhindern kann, steht in den ersten beiden Bytes des BASIC-Programms. Wenn beide Bytes auf 0 gesetzt sind, kann der Dragon das Programm weder LISTen noch mit RUN starten.

Um diese Schutzfunktion zu nutzen, sollten Sie Ihr Programm zuerst sehr gründlich auf Fehler oder Mängel prüfen. Danach wird es mit RENUMber so numeriert, daß 10 zur ersten Zeile wird. Beim Eingeben der folgenden Zeilen müssen Sie besonders darauf achten, daß Zeile 1 fehlerlos ist – dabei kommt es besonders auf das Leerzeichen an:

```
1 REM□AAAAAAAAAAAAAAAAAAAAAAAA
2 ST=PEEK(25)*256 + PEEK(26):
    A$="308C10EC81DD19EC81ED
    8CEEEC84DD1B7E85A5"
3 FOR K=1 TO 38 STEP 2: POKE ST + 6 +
    K/2, VAL("&H" + MID$(A$,K,2)): NEXT
4 POKE ST + 25, PEEK(25): POKE ST + 26,
    PEEK(26): POKE ST + 27,PEEK(ST): POKE
    ST + 28, PEEK(ST +1): END
```

Jetzt wird das Programm mit CSAVE ganz normal abgespeichert. Beim Starten mit RUN wird eine kurze Maschinenroutine durch die Zeilen 2 und 3 in die REM-Zeile 1 gePOKEt. Die Daten für den BASIC-Anfangszeiger gelangen aus Zeile 4 dorthin. Starten Sie das Programm mit RUN und LISTen Sie danach Zeile 1 – Sie werden sehen, daß sich deren Inhalt bereits gehörig geändert hat.

Um den „Einbruch" in das fertige Programm zu vermeiden, müssen noch die BREAK- und die RESET-Taste abgeschaltet werden. Nach dem Löschen der Zeilen 3 und 4 des Programms geben Sie dazu ein:

```
2 A$="E4ED04CBE4EC":FOR K=1 TO 12
      STEP 2:POKE
416–K/2,VAL("&H"+MID$(A$,K,2)):
NEXT:POKE113,0
```

Durch die FOR. . .NEXT-Schleife wird BASIC daran gehindert, die BREAK-Taste abzufragen. Das POKE am Schluß führt zu einem NEW, falls die RESET-Taste gedrückt wird. Für Programme, die den Befehl INPUT verwenden, ist dieses Verfahren allerdings nicht zu gebrauchen. Eine Eingabe mit diesem Befehl würde ebenfalls zu einem, in diesem Falle jedoch unerwünschten, Absturz führen.

Bevor das Programm abgespeichert werden kann, müssen in die REM-Befehle von Zeile 1 Informationen über das Ende des BASIC-Programms eingebaut werden. Das geht nur im direkten Eingabemodus, weil jede neue Programmzeile den Wert verändern würde. Geben Sie diese Zeilen also direkt ein:

```
ST=PEEK(25)*256+PEEK(26):POKE
    ST+29,PEEK(27):
POKE ST+30,PEEK(28):POKE ST,0:POKE
    ST+1,0
```

Danach ENTER drücken. Zum Abspeichern:

```
CSAVEM "Name des Programms",
ST,PEEK(27)*256+PEEK(28),ST+6
```

Damit wird das Programm in geschützter Form für den späteren Gebrauch bereitgehalten.

Zum Laden des Programms von der Cassette anstelle von CLOAD den Befehl CLOADM eingeben. Nach dem Laden findet der Dragon kein BASIC-Programm, das geLISTet oder mit RUN gestartet werden kann. Zum Starten EXEC und danach ENTER eintippen.

In den Commodore können mit einem Bootstrap-Programm mehrere Maschinenroutinen eingegeben werden. Allerdings müssen Vorsichtsmaßnahmen getroffen werden – das Programm verzweigt sonst zum Listing-Anfang und lädt das erste File immer wieder. Am Anfang des Programms muß eine „Flag"-Variable gesetzt werden:

```
10 IF A=0 THEN A = 1: LOAD"FILE1",1,1
20 IF A=1 THEN A = 2: LOAD"FILE2"1,1
```

Springt das Programm an den Anfang zurück, merkt es, daß der Variablen A ein Wert zugeordnet ist – es arbeitet in Zeile 20 weiter und lädt das nächste File.

Dynamische RAMs

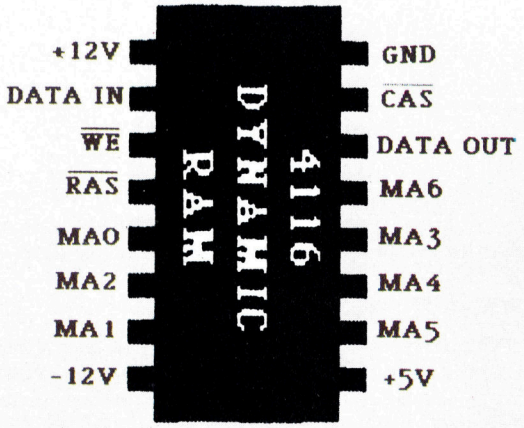
In der vorigen Folge wurde erklärt, wie ROM und statische RAMs vom Prozessor gesteuert werden. Heute untersuchen wir dynamische RAMs und gewinnen Einblick in die Arbeitsweise eines RAM-Chips.

In der letzten Folge hatten wir kurz erwähnt, daß es zwei Arten von RAMs gibt: dynamische und statische. Während statische RAMs mit kleinen logischen Mechanismen – den Flipflops – arbeiten, speichern dynamische RAMs ihre Datenbits als elektrische Ladung.

Beide RAM-Typen haben Vor- und Nachteile. So ist der Transistor, der in dynamischen RAMs eine Bit-Ladung speichert, weitaus kleiner als ein Flipflop, der die gleiche Information in statischen RAMs festhält. Aus diesem Grund lassen sich dynamische RAMs dichter packen. Da die Ladung dynamischer RAMs sich jedoch schon nach einigen Millisekunden „verflüchtigt", sind Zusatzschaltungen nötig, die sie periodisch „auffrischen". Statische RAMs haben dieses Problem nicht und sind eine Lösung für Systeme mit kleinem Arbeitsspeicher.

Statische RAMs werden normalerweise in Acht-Bit-Registern angeordnet. Dabei verfügt jeder Chip über acht Datenkontakte, die mit den acht Leitungen des Datenbusses verbunden sind. Dynamische RAMs haben jedoch oft einen anderen Aufbau. Jeder RAM-Chip stellt dabei eins der acht Datenbits einer Speicherstelle dar. Der Speicher enthält dann acht parallel geschaltete Chips dieser Art.

Das Bild unten zeigt den 4116 mit den typischen Kontakten eines 16-KBit-Chips. Für einen Arbeitsspeicher von 64 KByte verwendet man bei vielen Acht-Bit-Micros den dynamischen RAM Chip 4164 (64 KBit). Obwohl wir uns den Chip als ein Bit „breit" und 16 mal 1024 Bits „lang" vorstellen können, enthält er 128 Zeilen mal 128 Spalten.



Die typische Pinbelegung eines 16-KBit RAM: Anders als beim statischen RAM korrespondiert ein dynamisches RAM mit einem Bit des Datenworts – daher je eine einfache Data-in- und Data-out-Leitung. Die Verkabelung ist recht einfach.

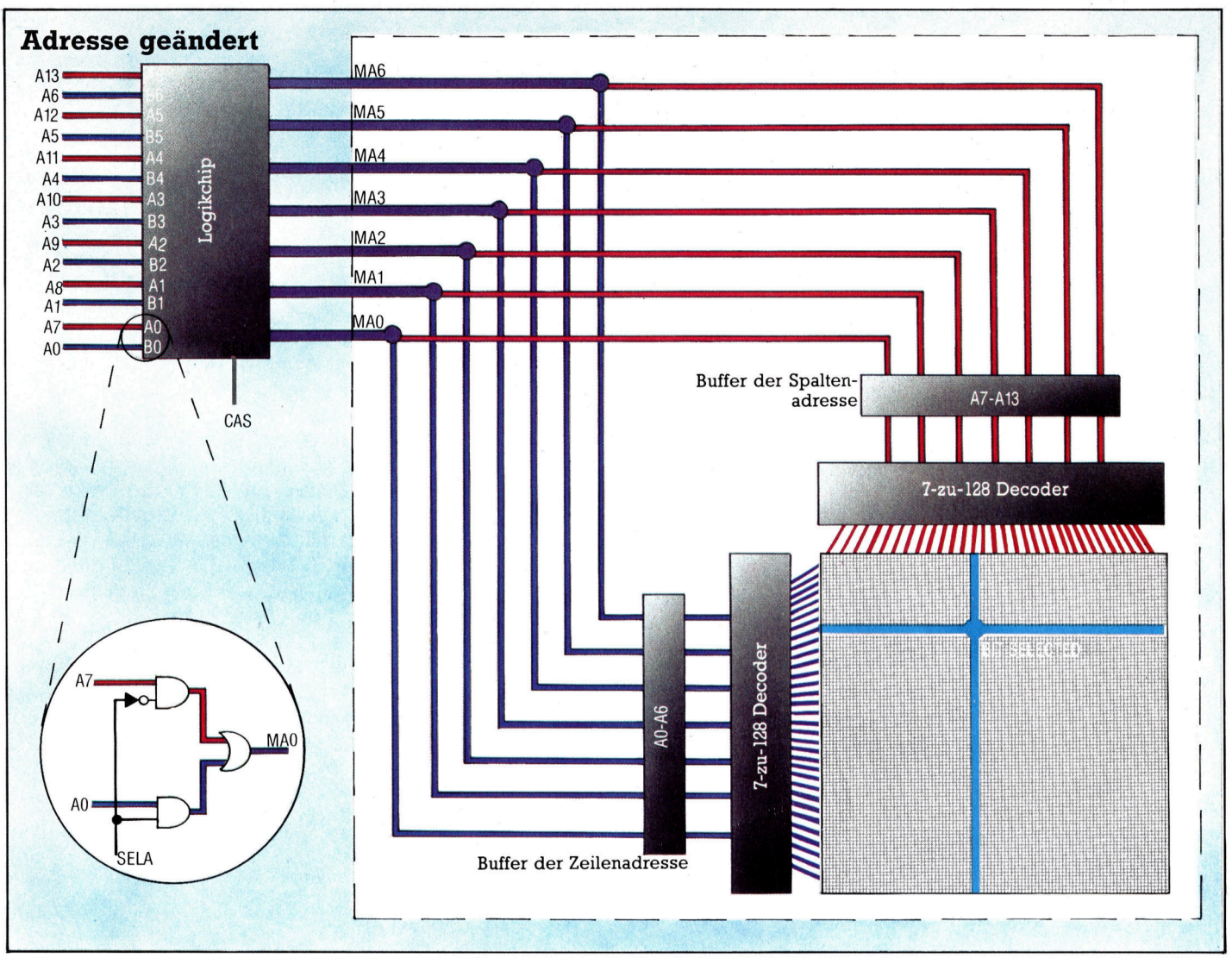
Bei einem Vergleich zwischen diesem Chip und einem statischen ROM fallen etliche Unterschiede ins Auge. Zunächst gibt es statt acht Datenkontakten nur zwei: „Data in" und „Data out". Bei einem 16-KBit-Chip wären 14 Adreßbits zu erwarten, die jedes Bit einzeln ansprechen. Der 4116 hat jedoch nur sieben. Außerdem gibt es zwei Leitungen mit seltsamen Bezeichnungen: $\overline{RAS}$ und $\overline{CAS}$ – „Row Adress Strobe" und „Column Adress Strobe". Mit diesen zwei Zeitsignalen läßt sich die Adresse in zwei Hälften darstellen (daher sieben statt vierzehn Adreßkontakte). Die $\overline{RAS}$-Leitung wird außerdem für die Auffrischung der dynamischen RAMs eingesetzt.

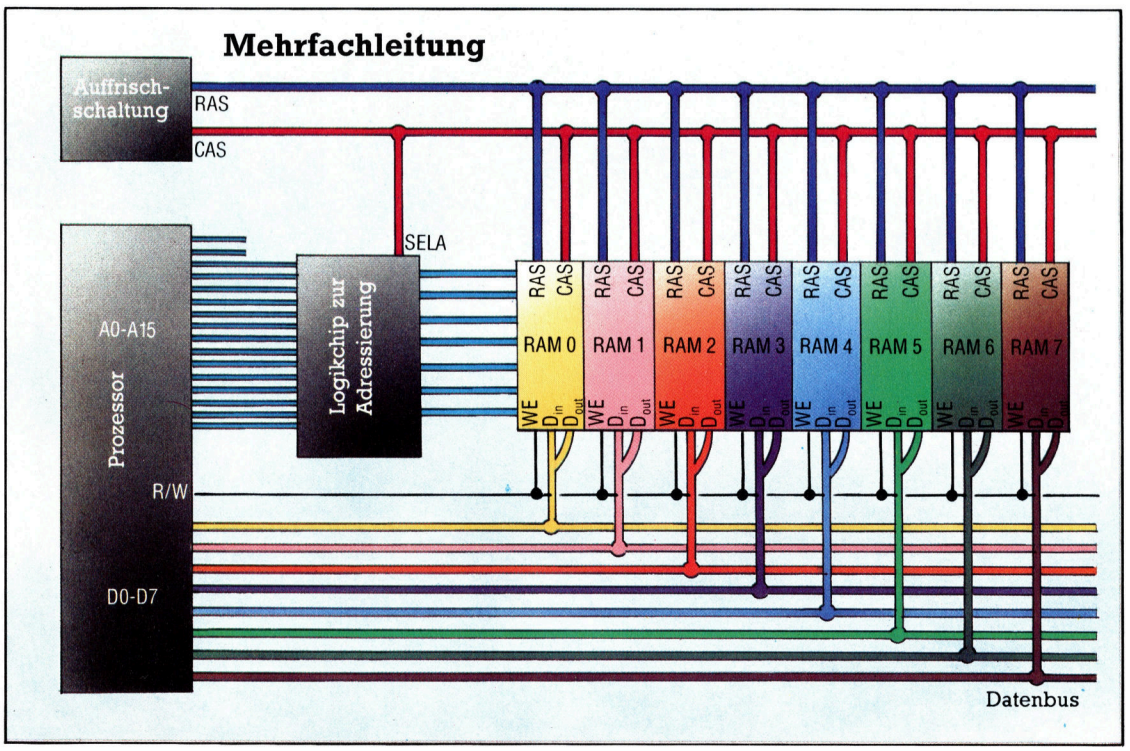
Auffrischung des Chips

Beim Auffrischungsvorgang werden alle Daten aus dem dynamischen RAM ausgelesen. Das Zurückschreiben stellt die Ladung wieder her. Da der Vorgang in unserem Beispiel zeilenweise abläuft, sind zur Auffrischung des Chips 128 Zyklen nötig. Die Auffrischung verlangsamt in vielen Fällen den Speicherzugriff des Prozessors, doch beträgt die Verzögerung selten mehr als fünf Prozent.

Die 14 Adressen eines 16-KByte-Speichers werden von unserem RAM in zwei Teilen zu je sieben Bits dargestellt. Das ermöglicht ein externer Logikchip, der die nieder- und höherwertigen Adreßleitungen mit dem CAS-Zeitsignal abstimmt. Unser Bild rechts oben zeigt, wie die niederwertigen Adreßleitungen (A0 bis A6) mit den Adreßkontakten des Chips verbunden sind, wenn CAS gesetzt ist. Ist CAS nicht gesetzt, sind die höherwertigen Adreßleitungen angeschlossen. Der RAM-Chip selbst enthält zwei Buffer, die ankommende Datensignale „einfrieren", so daß die Zeilen- und Spaltenadresse nacheinander ausgesandt werden können. Ein entsprechender Decoder (7 auf 128) setzt dann die hier erzeugten Adreßwerte zusammen.

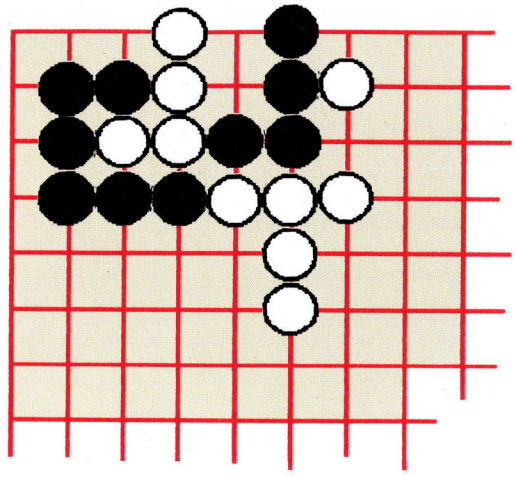
Unser Bild rechts unten zeigt, wie ein dynamisches RAM von 16 KByte mit den Adreß-, Daten- und Schreib-/Lese-Leitungen des Prozessors verbunden ist.





Ein Speicher mit der Kapazität von 16 KByte läßt sich aus acht 4116 Chips aufbauen, die parallel geschaltet sind und sich die Leitungen für Adresse, R/W, RAS und CAS teilen. Bei dieser Anordnung ist jedes dynamische RAM mit einer Leitung des Datenbusses verbunden.

Die 14 Adreßleitungen, die jedes Bit eines 16-KBit-RAM-Chips ansprechen können, erzeugen je eine 7-Bit-Spalten- und Zeilenadresse. Zur Vereinfachung der Schaltung besitzt der Chip selbst nur sieben Adreßleitungen. Der kreisförmige Bildausschnitt zeigt, wie einfach die logische Schaltung des externen Logikchips aufgebaut ist, die zwischen den beiden Adreßhälften umschaltet.



SEMEAI

Schlechte Position
Um Steine am Spielbrettrand zu erobern, kann man sie „erdrücken" – eine Taktik, bekannt unter dem Namen SEMEAI. In einem solchen Kampf ist die Anzahl der Freifelder sehr wichtig. So sieht es in unserem Beispiel für die weiße Gruppe schlecht aus, da sie nur über drei Freifelder verfügt, die schwarze Gruppe dagegen über vier.

Stein auf Stein

Im folgenden zeigen wir Ihnen das Modul zum Setzen der Steine. Dabei sind diverse Abfragen des Computers notwendig, die den Zug überprüfen, Meldungen ausgeben und die Spielbrettdarstellung aktualisieren.

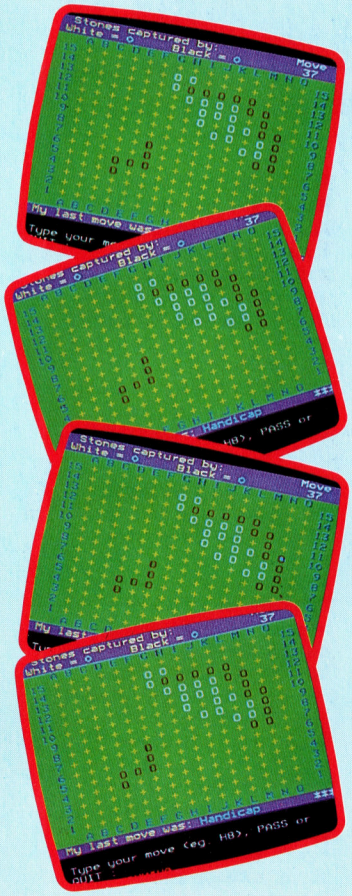
Durch den strukturierten und allgemeinen Aufbau des Programms ist es sehr einfach, das Listing für zwei Spieler zu modifizieren. Wie bisher bezieht sich die Programmbeschreibung auf die Acorn-B-Version; Zeilennummern und Struktur sind in den anderen drei Versionen beibehalten.

Die erste Routine, mit der wir uns befassen, ist FNlegality (Zeilen 3890–4000). Diese Funktion akzeptiert einen Zug (P%) von Farbe C% und gibt einen Wert aus, der anzeigt, ob der Zug legal ist. Diese Überprüfung gewährleistet, daß:

1. die Schnittstelle (P%) frei ist,
2. der Spieler keinen Stein in Ko einnimmt,
3. der gespielte Stein keinen „Selbstmordversuch" unternimmt.

Die Überprüfung, ob eine Position frei ist, erfolgt in Zeile 3910 durch Abfrage nach einer 0 im entsprechenden Brett-Byte. Zur Probe wird der Stein „provisorisch" gesetzt. Die Schleife L% zählt dann die Freifelder der angrenzenden gegnerischen Steine (unter Verwendung von PROCcount). Innerhalb der Schleife wird durch K% die Anzahl an Steinen, die durch den Zug P% beseitigt würden, verzeichnet.

- Wenn keine Steine beseitigt werden (K%=0) und der zu setzende Stein keine Freifelder hat (clib%=0), versucht der Stein „Selbstmord".
- Wenn ein Stein beseitigt wird (K%=1) und sich der zu setzende Stein in der Ko-Position befindet (P%=ko%), wird die Ko-Regel verletzt.



Drittes Modul

Acorn B:

```
 1380 ko%=FALSE:end%=FALSE
 2310 :
 2320 DEF PROCwhite_move
 2330 LOCAL C%,L%,P%,X%,Y%,A$,C$,Y$
 2340 atari2$="      "
 2350 A$=FNinput(21,8,4)
 2360 IF A$="PASS" THEN ko%=FALSE:PROCpr
int_board:GOTO 2490
 2370 IF A$="QUIT" THEN end%=TRUE:ENDPRO
C
 2380 X%=ASC(LEFT$(A$,1))-64
 2390 Y$=MID$(A$,2)
 2400 FOR C%=1 TO LEN(Y$)
 2410   C$=MID$(Y$,C%,1)
 2420   IF C$<"0" OR C$>"9" THEN PROCmes
sage(23,1,A$):GOTO2350
 2430   NEXT
 2440 Y%=VAL(Y$)
 2450 IF X%<1 OR X%>15 OR Y%<1 OR Y%>15
THEN PROCmessage(23,1,A$):GOTO 2350
 2460 P%=16*Y%+X% : L%=FNlegality(P%,whi
te%)
 2470 IF L%>0 THEN PROCmessage(23,L%,A$)
:GOTO 2350
 2480 PROCmake_move(P%,white%)
 2490 PROCmessage(23,0,"")
 2500 ENDPROC
 2510 :
 2520 REM*******************************
 3620 :
 3630 DEF PROCmake_move(P%,C%)
 3640 LOCAL A%,L%,N%
 3650 board%?P%=C%
 3660 FOR L%=1 TO 4 : A%=P%+dir%(L%) : I
F board%?A%<>colour%-C% THEN 3700
 3670   PROCcount(A%,colour%-C%):IF clib
%=0 THEN ko%=A%:N%=N%+cstn%:PROCremove(A
%,colour%-C%)
 3680   IF clib%=1 AND C%=black% THEN at
ari1$="Atari1":atari2$="      "
 3690   IF clib%=1 AND C%=white% THEN at
ari2$="Atari1"
 3700   NEXT
 3710 IF N%<>1 THEN ko%=FALSE
 3720 capture%(C%)=capture%(C%)+N%
 3730 PROCprint_board
 3740 ENDPROC
 3750 :
 3760 REM*******************************
 3770 :
 3780 DEF PROCremove(P%,C%)
 3790 IF board%?P%<>C% THEN ENDPROC
 3800 board%?P%=0
 3810 PROCremove(P%+dir%(1),C%)
 3820 PROCremove(P%+dir%(2),C%)
 3830 PROCremove(P%+dir%(3),C%)
 3840 PROCremove(P%+dir%(4),C%)
 3850 ENDPROC
 3860 :
 3870 REM*******************************
 3880 :
 3890 DEF FNlegality(P%,C%)
 3900 LOCAL A%,K%,L%,S%
 3910 IF board%?P%<>0 THEN =2
 3920 board%?P%=C%
 3930 FOR L%=1 TO 4
 3940   A%=P%+dir%(L%)
 3950   IF board%?A%=colour%-C% THEN PRO
Ccount(A%,colour%-C%):IF clib%=0 THEN K%
=K%+cstn%
 3960   NEXT
 3970 IF K%=0 THEN PROCcount(P%,C%):IF c
lib%=0 THEN S%=4
 3980 board%?P%=0
 3990 IF P%=ko% AND K%=1 THEN =3
 4000 =S%
 4010 :
 4020 REM*******************************
```

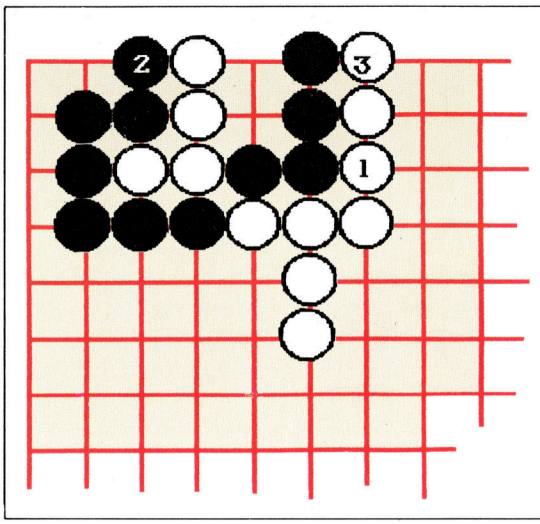
Die Ko-Abfrage ist eventuell etwas schwer zu verstehen. Wenn ein Spieler einen einzelnen Stein einnimmt, wird ko% von der make_move-Routine in den Zeilen 3670 und 3710 auf die Position dieses Steines gesetzt. Wenn der neu zu spielende Stein auf diesen Punkt positioniert wird und dieser Stein nur den zuletzt gespielten gegnerischen Stein erobert, muß die Ko-Regel verletzt sein.

Steine vom Brett

PROCmake_move beginnt damit, einen Stein der entsprechenden Farbe auf das Brett zu setzen, woraufhin eine Schleife den Status der umliegenden Gruppen überprüft. Besetzt der Stein das letzte Freifeld einer gegnerischen Gruppe, wird diese Gruppe durch Aufruf von PROCremove vom Brett entfernt. Entsprechend wird, wenn die Gruppe nur noch ein Freifeld hat, der entsprechende „atari"-String aktualisiert. Abschließend wird die Anzahl der vom jeweiligen Spieler eroberten Steine aktualisiert und die Darstellung mit der print_board-Routine erneuert.

Wird ein Stein oder eine Gruppe von Steinen eingenommen, wird PROCremove aufgerufen, um sie vom Brett zu nehmen. Diese Routine arbeitet ähnlich wie PROCsearch. Zuerst wird der Stein in Position P% vom Brett entfernt. Danach ruft sie sich selbst auf, um jeden Stein derselben Farbe, der direkt im Norden, Osten, Süden oder Westen der aktuellen Position liegt, zu entfernen.

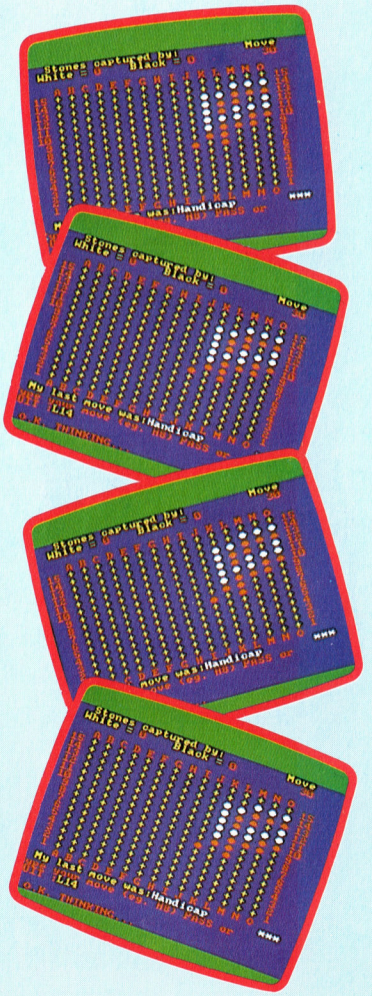
Auch hier wird bei der C64-, Spectrum- und Amstrad-Version der „stack" verwendet, um die Rekursion zu simulieren.

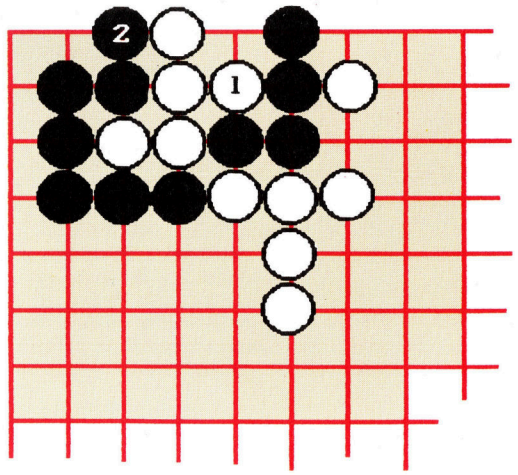


Die Position SEKI
SEMEAI-Kämpfe enden in der Regel mit einem Unterliegen der schwächeren Gruppe oder aber einer gefestigten Position, genannt SEKI. Hierzu muß Weiß die schwarze Gruppe von außen angreifen. In der gezeigten Situation hat Weiß seine Position gesichert. Auf die zwei verbleibenden Freifelder kann keiner setzen, ohne die eigene Gruppe zu gefährden.

Schneider CPC 464/664:

```
60 MOVE%=MOVE%+1:GOSUB 2320
1380 KO%=0:FIN%=0
2310 :
2320 REM WHITE-MOVE ROUTINE
2340 A2$="      "
2350 IP%=22:IM%=8:IW%=4:GOSUB 1990
2360 IF A$="PASS" THEN KO%=0:GOSUB 1730:
GOTO 2490
2370 IF A$="QUIT" THEN FIN%=-1:RETURN
2380 X%=ASC(LEFT$(A$,1))-64
2390 Y$=MID$(A$,2)
2400 FOR WC=1 TO LEN(Y$)
2410 C$=MID$(Y$,WC,1)
2420 IF C$<"0" OR C$>"9" THEN MP%=24:MM%
=1:O$=A$:GOSUB 2160:GOTO 2350
2430 NEXT
2440 Y%=VAL(Y$)
2450 IF X%>0 AND X%<16 AND Y%>0 AND Y%<1
6 GOTO 2460
2455 MP%=24:MM%=1:O$=A$:GOSUB 2160:GOTO
2350
2460 WP%=16*Y%+X%:LP%=WP%:LC%=WHITE%:GOS
UB 3890
2470 IF LL%>0 THEN MP%=24:MM%=LL%:O$=A$:
GOSUB 2160:GOTO 2350
2480 MP%=WP%:MC%=WHITE%:GOSUB 3630
2490 MP%=24:MM%=0:O$="":GOSUB 2160
2500 RETURN
2510 :
2520 REM*********************************
3620 :
3630 REM MAKE-MOVE ROUTINE
3640 N%=0
3650 POKE BOARD+MP%,MC%
3660 FOR K=1 TO 4:A=MP%+DIR%(K):IF PEEK(
BOARD+A)<>COLOUR%-MC% GOTO 3700
3670 CP%=A:CC%=COLOUR%-MC%:GOSUB 4040:IF
 CLIB%<>0 GOTO 3680
3675 KO%=A:N%=N%+CSTN%:RP%=A:RC%=COLOUR%
-MC%:GOSUB 3780
3680 IF CLIB%=1 AND MC%=BLACK% THEN A1$=
"ATARI":A2$="      "
3690 IF CLIB%=1 AND MC%=WHITE% THEN A2$=
"ATARI"
3700 NEXT
3710 IF N%<>1 THEN KO%=0
3720 CAPTURE%(MC%)=CAPTURE%(MC%)+N%
3730 GOSUB 1730
3740 RETURN
3750 :
3760 REM*********************************
3770 :
3780 REM REMOVE ROUTINE
3790 IF PEEK(BOARD+RP%)<>RC% THEN RETURN
3800 POKE BOARD+RP%,0
3805 SK%(STACK%)=RP%:STACK%=STACK%+1
3810 RP%=SK%(STACK%-1)+DIR%(1):GOSUB 378
0
3820 RP%=SK%(STACK%-1)+DIR%(2):GOSUB 378
0
3830 RP%=SK%(STACK%-1)+DIR%(3):GOSUB 378
0
3840 RP%=SK%(STACK%-1)+DIR%(4):GOSUB 378
0
3845 STACK%=STACK%-1:RP%=SK%(STACK%)
3850 RETURN
3860 :
3870 REM*********************************
3880 :
3890 REM LEGALITY ROUTINE
3900 LL%=0:LK%=0
3910 IF PEEK(BOARD+LP%)<>0 THEN LL%=2:RE
TURN
3920 POKE BOARD+LP%,LC%
3930 FOR K=1 TO 4
3940 LA%=LP%+DIR%(K)
3950 IF PEEK(BOARD+LA%)<>COLOUR%-LC% GOT
O 3960
3955 CP%=LA%:CC%=COLOUR%-LC%:GOSUB 4040:
IF CLIB%=0 THEN LK%=LK%+CSTN%
3960 NEXT
3970 IF LK%=0 THEN CP%=LP%:CC%=LC%:GOSUB
 4040:IF CLIB%=0 THEN LL%=4
3980 POKE BOARD+LP%,0
3990 IF LP%=KO% AND LK%=1 THEN LL%=3
4000 RETURN
4010 :
4020 REM*********************************
```





Schwarz gewinnt
Setzt Weiß während des SEMEAI-Kampfes auf eines der internen Freifelder, wird die Gruppe unweigerlich erobert. Nachdem Schwarz den mit 2 gekennzeichneten Stein gesetzt hat, verbleibt nur noch ein Freifeld, und der Kampf ist für Weiß verloren.

Befassen wir uns nun mit der Routine, die die Eingabe und Ausführung von Zügen gestattet – PROCwhite_move. Da wir bereits eine Anzahl allgemein verwendbarer Routinen haben, befaßt sich der größte Teil des Programmcodes mit der Überprüfung der Eingabe. Die Routine FNinput (Zeile 2350) erlaubt Ihnen, vier beliebige Zeichen einzugeben, so daß lediglich überprüft werden muß, ob Sie PASS, QUIT oder eine gültige Brettposition eingegeben haben.

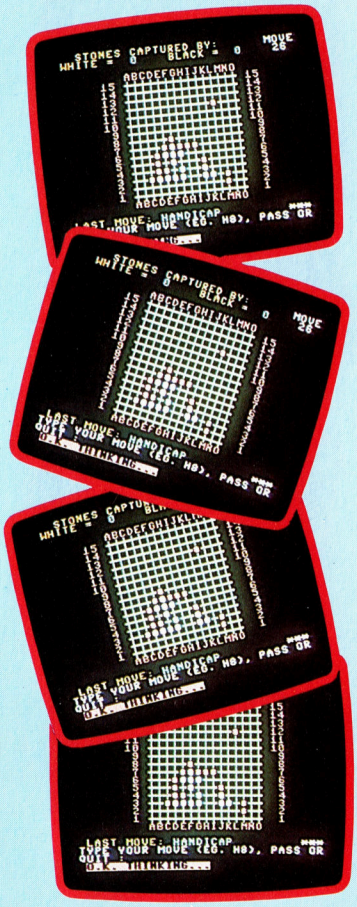
PASS bewirkt, daß das Ko-Flag zurückgesetzt wird, wogegen durch QUIT der logische Wert für das Spielende auf TRUE (WAHR) gesetzt wird, so daß das Programm nach Rücksprung die Hauptbewegungsschleife (Zeile 60 bis 90) verläßt. Entspricht die Eingabe weder PASS noch QUIT, unterteilt das Programm die Eingabe in horizontale (A bis O) und vertikale (1 bis 15) Koordinaten, die dann zu einer Brettposition P% zusammengesetzt werden.

Hauptbewegungsschleife

Wie Sie sehen, können wir hier nicht ohne weiteres die Funktion VAL verwenden. Dies liegt daran, daß die Eingabe auch nur 2 oder 3 Zeichen lang sein könnte und VAL (MID$(„A12",2)) immer noch das legale Ergebnis 1 ausgibt, obwohl am Ende des Strings ein Fragezeichen steht.

Ist die Eingabe gültig und FNlegality bestätigt den Zug als legal, ruft die Routine PROCmake_move auf, stellt in der Mitteilungszeile durch Aufruf von PROCmessage die „O.K. THINKING"-Meldung dar und bricht dann ab.

In diesem Programm haben wir anstelle der aktuellen Zahlen 1 und 2 die Konstanten black% und white% verwendet. Die einzige Ausnahme befindet sich in PROCprint_board, wo die Elemente des capture%()-Arrays die Anzahl eingenommener Steine jeder Seite anzeigt. Ein Vorteil dieser Methode ist, daß wir so



Commodore 64:

```
  60 mve%=mve%+1:GOSUB 2320
  90 IF NOT over% THEN 60
1380 ko%=0:over%=0
2310 :
2320 REM **** white move routine ****
2340 atari2$="      "
2350 ip%=21:im%=8:iw%=4:GOSUB 1990:REM i
nput
2360 IF a$="PASS" THEN ko%=0:GOSUB 1730:
GOTO 2490
2370 IF a$="QUIT" THEN over%=1:RETURN
2380 x%=ASC(LEFT$(a$,1))-64
2390 y$=MID$(a$,2)
2400 FOR c%=1 TO LEN(y$)
2410 c$=MID$(y$,c%,1)
2420 IF c$<"0" OR c$>"9" THEN mp%=25:mm%
=1:o$=a$:GOSUB 2160:GOTO 2350
2430 NEXT c%
2440 y%=VAL(y$)
2450 IF x%<1 OR x%>15 OR y%<1 OR y%>15 T
HEN mp%=25:mm%=1:o$=a$:GOSUB 2160:GOTO 2
350
2460 lp%=16*y%+x%:lc%=white%:GOSUB 3890:
REM check legality
2470 IF ll%>0 THEN mp%=25:mm%=ll%:o$=a$:
GOSUB 2160:GOTO 2350
2480 mmp%=lp%:mmc%=white%:GOSUB 3630:REM
 make move
2490 mp%=25:mm%=0:o$="":GOSUB 2160
2500 RETURN
2510 :
3620 :
3630 REM **** make move routine ****
3640 n%=0
3650 POKE (board+mmp%),mmc%
3660 FOR k%=1 TO 4:a%=mmp%+dir%(k%):IF P
EEK(board+a%)<>colour%-mmc% THEN 3700
3670 cp%=a%:cc%=colour%-mmc%:GOSUB 4040:
IF clib%=0 THEN ko%=a%:n%=n%+cstn%:rp%=a
%:rc%=colour%-mmc%:GOSUB 3780
3680 IF clib%=1 AND mmc%=white% THEN ata
ri2$="Atari"
3700 NEXT k%
3710 IF n%<>1 THEN ko%=0
3720 capture%(mmc%)=capture%(mmc%)+n%
3730 GOSUB 1730:REM print board
3740 RETURN
3750 :
3760 REM ************************
3770 :
3780 REM remove routine
3790 IF PEEK(board+rp%)<>rc% THEN RETURN
3800 POKE (board+rp%),0
3805 s(stack%)=rp%:stack%=stack%+1
3810 rp%=s(stack%-1)+dir%(1):GOSUB 3780
3820 rp%=s(stack%-1)+dir%(2):GOSUB 3780
3830 rp%=s(stack%-1)+dir%(3):GOSUB 3780
3840 rp%=s(stack%-1)+dir%(4):GOSUB 3780
3845 stack%=stack%-1:rp%=s(stack%)
3850 RETURN
3860 :
3870 REM ************************
3880 :
3890 REM legality routine
3900 ll%=0:lk%=0
3910 IF PEEK(board+lp%)<>0 THEN ll%=2:RE
TURN
3920 POKE (board+lp%),lc%
3930 FOR k%=1 TO 4
3940 la%=lp%+dir%(k%)
3950 IF PEEK(board+la%)=colour%-lc% THEN
 cp%=la%:cc%=colour%-lc%:GOSUB 4040:IF c
lib%=0 THEN lk%=lk%+cstn%
3960 NEXT k%
3970 IF lk%=0 THEN cp%=lp%:cc%=lc%:GOSUB
 4040:IF clib%=0 THEN ll%=4
3980 POKE (board+lp%),0
3990 IF lp%=ko% AND lk%=1 THEN ll%=3
4000 RETURN
4010 :
4020 REM ************************
```

einfach die Form, in der Brett-Bytes Farben repräsentieren, ändern können.

Die folgenden Zeilen verwandeln das Programm in eine Zwei-Spieler-Version (siehe BASIC-Dialekte für andere Computer):

```
60 move%=move%+1:black%=1:white%=2:
   PROCwhite_move
80 move%=move%+1:black%=2:white%=1:
   PROCwhite_move
```

Diese Zeilen ändern jeweils die Werte von black% und white% bei Aufruf von PROCwhite_move. Da alle Routinen so entworfen wurden, daß sie jeden Wert akzeptieren, gestattet der Computer jetzt Schwarz und Weiß abwechselnd zu spielen, wobei für beide PROCwhite_move verwendet wird, um die Gültigkeit eines Zuges zu überprüfen und ihn darzustellen. Im nächsten Abschnitt ändern wir diese Zeilen, so daß der Computer gegen Sie spielen kann. Bis dahin können Sie aber zumindest schon einmal gegen einen Ihrer Bekannten antreten.

BASIC-Dialekte

Mit den folgenden Zeilen können Sie das Programm für zwei Spieler verwenden. Der Computer fungiert als intelligentes Spielbrett, das die Gültigkeit von Zügen überprüft und den Punktestand vermerkt.

Commodore 64

```
60 MOVE%=MOVE%+1:BLACK%=1:WHITE%=2:
   GOSUB 2320
80 MOVE%=MOVE%+1:BLACK%=2:WHITE%=1:
   GOSUB 2320
```

Spectrum

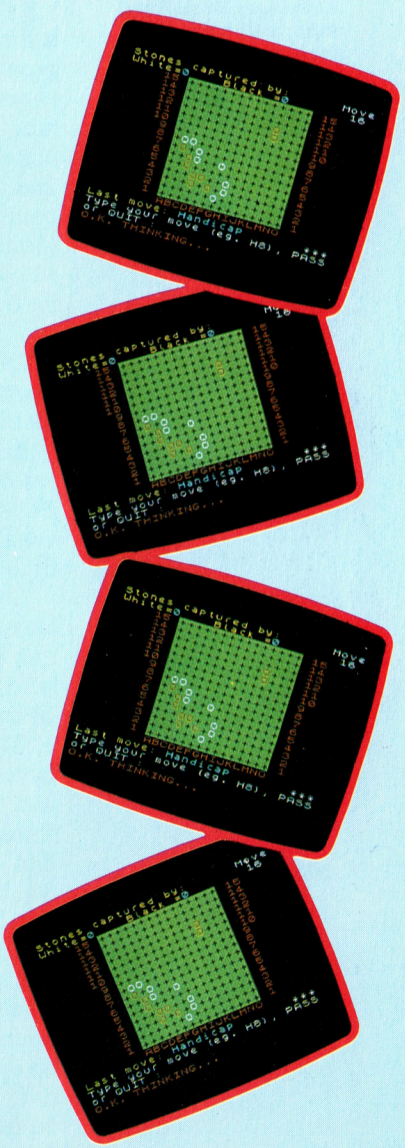
```
60 LET move=move+1:LET black=1:LET white=2:
   GO SUB 2320
80 LET move=move+1:LET black=2:LET white=1:
   GO SUB 2320
```

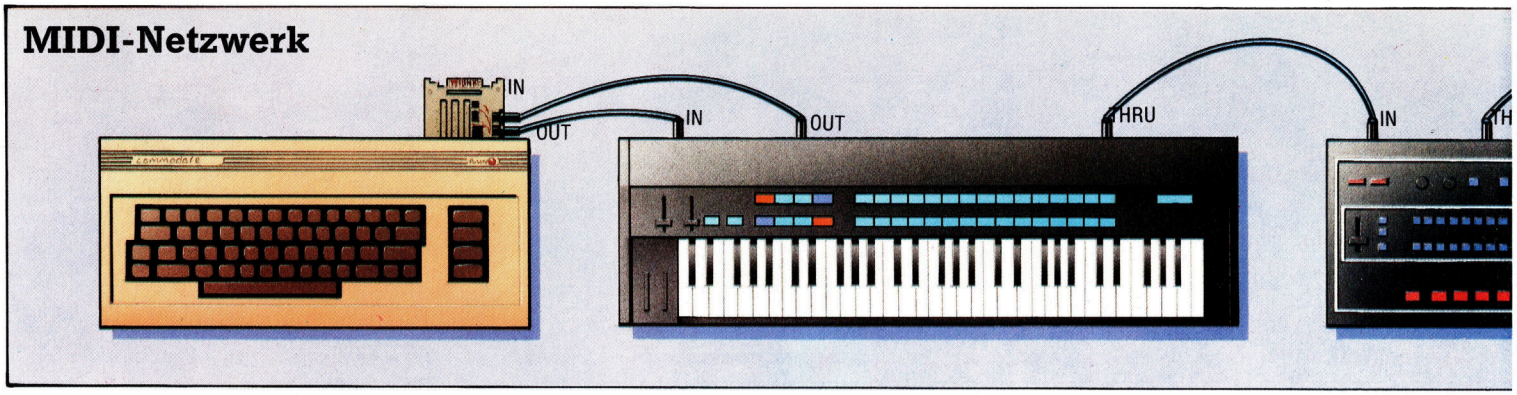
Schneider CPC 464/664

```
60 mve%=mve%+1:black%=1:white%=2:
   GOSUB 2320
80 mve%=mve%+1:black%=2:white%=1:
   GOSUB 2320
```

Sinclair Spectrum:

```
  60 LET move=move+1: GO SUB 232
0
1380 LET ko=0: LET end=0
2310:
2320 REM white-move routine
2340 LET y$="      "
2350 LET ip=19: LET im=9: LET iw
=4: GO SUB 1990
2360 IF a$="PASS" THEN  LET ko=0
: GO SUB 1730: GO TO 2490
2370 IF a$="QUIT" THEN  LET end=
1: RETURN
2380 LET x=CODE (a$(1))-64
2390 LET z$=a$(2 TO )
2400 FOR c=1 TO LEN z$
2410 LET c$=z$(c)
2420 IF c$<"0" OR c$>"9" THEN  L
ET mp=21: LET mm=2: LET o$=a$: G
O SUB 2160: GO TO 2350
2430 NEXT c
2440 LET y=VAL z$
2450 IF x<1 OR x>15 OR y<1 OR y>
15 THEN  LET mp=21: LET mm=2: LE
T o$=a$: GO SUB 2160: GO TO 2350
2460 LET lp=16*y+x: LET lc=white
: GO SUB 3890
2470 IF ll>0 THEN  LET mp=21: LE
T mm=ll: LET o$=a$: GO SUB 2160:
 GO TO 2350
2480 LET mmp=lp: LET mmc=white:
GO SUB 3630
2490 LET mp=21: LET mm=1: LET o$
="": GO SUB 2160
2500 RETURN
2510:
2520 REM ************************
3620:
3630 REM make-move routine
3640 LET n=0
3650 POKE board+mmp,mmc
3660 FOR k=1 TO 4: LET a=mmp+d(k
): IF PEEK (board+a)<>colour-mmc
 THEN  GO TO 3700
3670 LET cp=a: LET cc=colour-mmc
: GO SUB 4040: IF clib=0 THEN  L
ET ko=a: LET n=n+cstn: LET rp=a:
 LET rc=colour-mmc: GO SUB 3780
3680 IF clib=1 AND mmc=black THE
N  LET x$="Atari": LET y$="
"
3690 IF clib=1 AND mmc=white THE
N  LET y$="Atari"
3700 NEXT k
3710 IF n<>1 THEN  LET ko=0
3720 LET c(mmc)=c(mmc)+n
3730 GO SUB 1730
3740 RETURN
3750:
3760 REM ************************
3770:
3780 REM remove routine
3790 IF PEEK (board+rp)<>rc THEN
  RETURN
3800 POKE board+rp,0
3805 LET s(stack)=rp: LET stack=
stack+1
3810 LET rp=s(stack-1)+d(1): GO
SUB 3780
3820 LET rp=s(stack-1)+d(2): GO
SUB 3780
3830 LET rp=s(stack-1)+d(3): GO
SUB 3780
3840 LET rp=s(stack-1)+d(4): GO
SUB 3780
3845 LET stack=stack-1: LET rp=s
(stack)
3850 RETURN
3860:
3870 REM ************************
3880:
3890 REM legality routine
3900 LET ll=0: LET lk=0
3910 IF PEEK (board+lp)<>0 THEN
 LET ll=3: RETURN
3920 POKE board+lp,lc
3930 FOR k=1 TO 4
3940 LET la=lp+d(k)
3950 IF PEEK (board+la)=colour-l
c THEN  LET cp=la: LET cc=colour
-lc: GO SUB 4040: IF clib=0 THEN
  LET lk=lk+cstn
3960 NEXT k
3970 IF lk=0 THEN  LET cp=lp: LE
T cc=lc: GO SUB 4040: IF clib=0
THEN  LET ll=5
3980 POKE board+lp,0
3990 IF lp=ko AND lk=1 THEN  LET
 ll=4
4000 RETURN
4010:
4020 REM ************************
```





Üblicherweise sind MIDI-kompatible Geräte in Kettenform miteinander verbunden. Die meisten Instrumente haben dafür neben MIDI-IN- und MIDI-OUT-Anschluß noch eine MIDI-THRU-Buchse. Damit können mehrere Geräte ähnlich wie Computerkomponenten durch eine gemeinsame Busleitung gekoppelt sein. Dennoch reagiert jedes Instrument nur auf ganz bestimmte Daten, die über die Leitung kommen.

Auf Bachs Spuren

Auf die mechanisch/elektrische Konstruktion der MIDI-Schnittstelle folgt ein nicht minder wichtiger Abschnitt: Erst die Software erlaubt uns das Musizieren in „Echtzeit".

Jeder musikalische Vorgang („event") wird durch eine Gruppe von Daten symbolisiert, eine „message" (Nachricht). Die meisten Messages sind zwischen einem und drei Bytes lang. Eine Ausnahme bilden die systemspezifischen Informationen, auf die wir später noch zurückkommen. Das Anfangsbit (Most Significant Bit) des ersten Message-Bytes ist immer gleich Eins. In allen folgenden Bytes einer Message sind die MSBs dann gleich Null. Ein Byte, bei dem das MSB Eins ist, nennen wir „Statusbyte", die anderen sind Databytes. Die MIDI-Messages lassen sich in zwei Gruppen unterteilen – Kanal-Informationen und System-Informationen.

Kanal-Informationen (Channel-messages) werden gebraucht, weil MIDI-Geräte meist in Kettenform verbunden sind – jedes Instrument erhält alle Informationen, die von der „Zentrale" ausgehen.

Durch die Kanal-Informationen werden die Daten nur von dem Instrument akzeptiert, für das sie gedacht sind. Den Empfangsgeräten wird dafür eine Kanal-Nummer zwischen 1 und 16 zugeordnet.

Channel-Messages setzen sich aus Statusbytes im Bereich $80 bis einschließlich $EF und ein bis zwei Datenbytes zusammen. Die Kanalnummer wird in den vier niederwertigen Bits des Statusbytes codiert. Kanal 1 wird mit $0, Kanal 16 mit $F bezeichnet. Die übrigen drei Bits geben die Art der darauffolgenden Information an.

Eine Besonderheit der Channel-Messages ist es, daß zwischen Einzelinformationen kein neues Statusbyte geschickt werden muß, wenn es sich nicht vom vorhergehenden unterscheidet. Dadurch bleibt der aktuelle Status so lange erhalten, bis ein neues Statusbyte empfangen wird.

Besonders nützlich kann diese Eigenschaft sein, wenn viele aufeinanderfolgende Note-Ein- bzw. Note-Aus-Befehle übertragen werden sollen. Der Note-Aus-Befehl gleicht nämlich dem Befehl Note-Ein mit der Anschlagstärke = 0. Die Verwendung des gleichen Status für mehrere Note-Ein/Aus-Befehle spart Speicherplatz und Übertragungszeit.

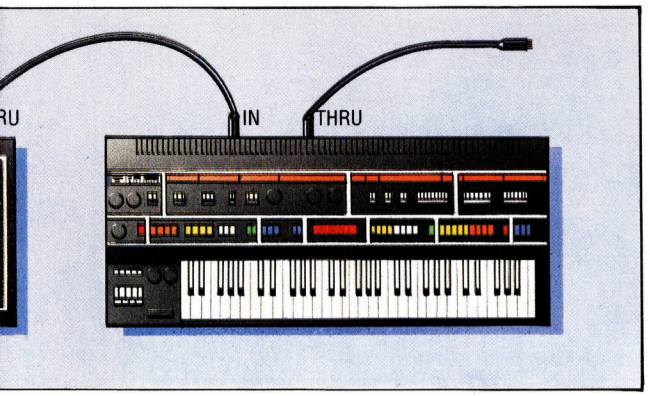
Stimmendefinition (voice assignement) ist der Vorgang, mit dem ein Noten-Befehl einem der freien Klangerzeuger (Stimmen) eines polyphonen (mehrstimmigen) Empfängers zugeordnet wird. Um die Reaktion eines Instruments auf die Kanal-Informationen richtig zu steuern, kann bei MIDI durch „mode messages" unter verschiedenen Betriebsarten gewählt werden. „Mode messages", die über die Fähigkeiten eines angeschlossenen Instrumentes hinausgehen (etwa wegen zu kleiner Stimmenzahl), sollten von diesem ignoriert werden.

Mehr Klanganweisungen

Die einfachste Betriebsart ist der „omni-mode", bei dem der Empfänger auf alle Informationen reagiert, ohne sich um vorangegangene Kanalnummern zu kümmern.

Die meisten polyphonen Synthesizer können nur eine bestimmte Anzahl von Klängen gleichzeitig erzeugen (meist sechs oder acht). Wie reagiert das Instrument nun, wenn mehr als acht Klanganweisungen eingehen?

Normalerweise arbeiten die einzelnen Stimmen nach dem Rotationsverfahren: Die zuerst angeforderte Note wird beendet, um Platz für den neuen Klang zu geben. Diese Betriebsart wird durch den Befehl „poly mode on" eingeschaltet. Im „mono mode" ist jede der verfügbaren Stimmen des Empfängers einem einzel-



nen MIDI-Kanal zugeordnet. Der Befehl „mono mode" selbst erreicht das Gerät auf dem allgemeinen Kanal, das zweite Datenbyte legt dabei die gewünschte Zahl von Kanälen fest.

Im „omni-mode" führt eine „mono-message" dazu, daß der Empfänger eine seiner Stimmen für allgemeine MIDI-Befehle freihält.

Systembefehle (system messages) haben keine Kanalnummern im Statusbyte, sie gehen an alle Instrumente des Netzwerkes. Es gibt Gruppen dieser Befehle: allgemeine, Echtzeitbefehle und systemspezifische Anweisungen.

Allgemeine Befehle (system common) haben Statusbytes zwischen $F1 und $F7. Sie bestehen aus einem Statusbyte, gefolgt von null, einem oder zwei Datenbytes.

Echtzeit-Befehle (system real-time) ergehen an alle Systemkomponenten. Ihre Statusbytes liegen zwischen $F8 und $FF. Sie werden hauptsächlich von Schlagzeugen und Taktgebern genutzt, um ihre Taktimpulse an den Systemtakt anzugleichen. Die meisten Synthesizer übergehen diese Befehle. Ausnahmen bilden Synthesizer, deren Taktgeneratoren eine Synchronisation vorsehen.

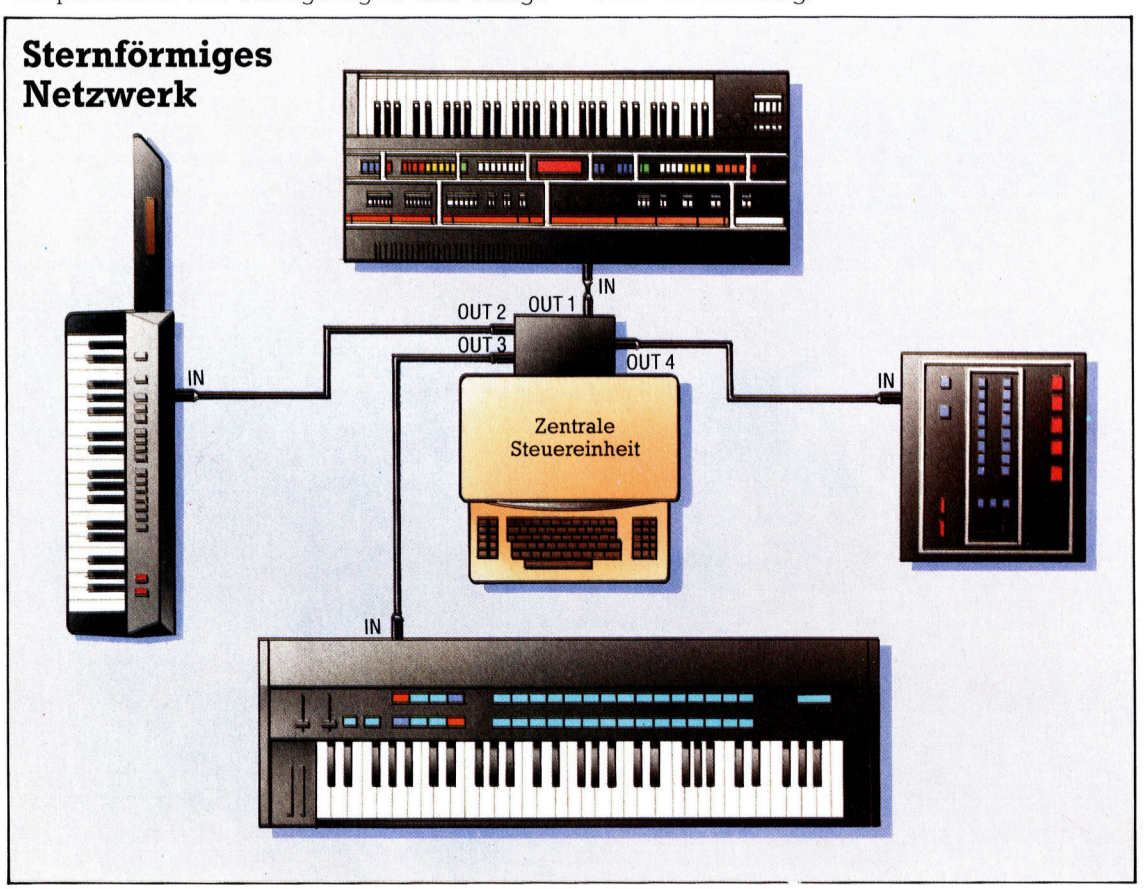
System-Echtzeit-Befehle unterscheiden sich von anderen Befehlen dadurch, daß sie nur aus einem Statusbyte ohne nachfolgende Datenbytes bestehen. Sie können jederzeit gesendet werden, selbst wenn sie dabei Befehle anderer Art unterbrechen.

Systemspezifische Befehle (system exclusive) beginnen mit dem Statusbyte $F0, auf das einige Datenbytes folgen. Abgeschlossen werden sie entweder durch das „end-of-exclusive" ($F7) oder irgendein anderes Statusbyte. Das erste Byte ist eine Herstellerkennung (ID). Entspricht sie nicht der Kennung des Empfängers, wird der Befehl ignoriert.

„System exclusive"

„System exclusive"-Informationen sind für den Datenaustausch von Geräten ähnlichen Typs gedacht. Normalerweise handelt es sich dabei um die Daten eines Synthesizer-Anpassungsprogramms. (Sie sollten nicht mit der channel-message $Cx verwechselt werden, die zwischen im Empfangsgerät vorhandenen Programmen wählt.)

Die nach „system exclusive"-Befehlen übermittelten Daten sind den Herstellern freigestellt – Voraussetzung ist nur ein gültiger ID-Code als Kennung.



Die ersten MIDI-Anlagen besaßen noch nicht die Fähigkeit, mehrere Kanäle getrennt anzusprechen – jedes angeschlossene Gerät hätte in der Kettenformation auf alle übermittelten Daten reagiert. Die frühen MIDI-Systeme wurden daher sternförmig mit der zentralen Steuereinheit verbunden. Für jedes Instrument gab es eine separate OUT-Buchse. Eine spezielle Schaltung sorgte dafür, daß die Geräte nur einzeln angesprochen wurden.

Kanal und Stimmen

Status	Data1	Data2	Bedeutung	Erklärung
$8x	p	v	Note aus p=Tonhöhe v=Lautstärke aus	1,2
$9x	p	v	a) v>0 Note ein p=Tonhöhe v=Lautstärke b) v=0 Note aus p=Tonhöhe	1,2 1,2
$Ax	p	pr	Anschlagstärke p=Tonhöhe pr=Anschlagstärke	1,3a
$Bx	n	c	a) n < $7A Wechsel der Steuerung n=Nummer des Steuergerätes c=Eingestellter Wert b) n=$7A Lokale Steuerung Ein (c=$7F) Aus (c=0) c) n=$7B, c=0 Alle Noten aus d) n=$7C, c=0 Omni-Mode aus (alle Noten aus) e) n=$7D, c=0 Omni-Mode ein (alle Noten aus) f) n=$7E Mono-Mode ein (alle Noten aus) c=Anzahl der Kanäle g) n=$7F, c=0 Poly-Mode ein (alle Noten aus)	4 6 6 6,7 6
$Cx	pp	–	Wahl des Anpassungsprogramms pp=Programmnummer	
$Dx	pr	–	Anschlagstärke (Kanal) pr=Anschlagstärke	3b
$Ex	l	m	Tonhöhenschwankung (Vibrato) l=7 niederwertige Bits m= 7 höchstwertige Bits	8

Erklärungen:

Solange nichts anderes vorgeschrieben ist, dürfen die Datenbytes jeden Wert zwischen 0 und $F7 haben. Die jeweilige Kanalnummer ist x, die erste Hex-Zahl des Statusbytes (x=0 für Kanal 1 bis x=SF für Kanal 16).

1. p ist die Höhe der Note, gemessen in Halbtonschritten. Das mittlere C hat die Tonhöhe $3C, alle Cs sind daher Produkte von $0C (dezimal 12). Eine normale Klaviertastatur reicht von $15 bis $6C.

2. Lautstärkewerte reichen von $01 bis $F7. Keyboards ohne Anschlagsensoren sollten mit dem voreingestellten Wert $40 arbeiten. Der Befehl „Note ein" mit Lautstärke 0 entspricht dem Befehl „Note aus" mit dem Lautstärkewert $40. Dieses Verfahren wird bei allen einfacheren Instrumenten ohne Dämpfungssensoren genutzt. Dadurch kann die Möglichkeit der Kanal-Befehle voll ausgenutzt werden.

3. „After Touch" ist ein Maß dafür, wie stark eine Taste nach dem Anschlag gedrückt wird. Im allgemeinen wird dadurch Modulation erzeugt, ohne ein „modulation wheel" zu benutzen. Es gibt zwei unterschiedliche Arten von Drucksensoren, für die entsprechend unterschiedliche Befehle vorgesehen sind:
a) Individueller bzw. polyphoner Anschlag erfordert für jede Taste am Instrument einen eigenen Drucksensor. Nur die gerade gedrückte Taste wird vom Meßwert dieses Sensors beeinflußt, das heißt, außer dem Meßwert muß auch die Tonhöhe gesendet werden. (In diesem Falle muß das Instrument auch mit mehreren separaten Modulationsschaltungen für den mehrstimmigen (polyphonen) Betrieb ausgerüstet sein.)
b) Es ist nur ein Drucksensor für die gesamte Tastatur vorgesehen. Der gemeldete Druck wird mit dem höchstmöglichen Druck (alle Tasten gleichzeitig betätigt) abgeglichen. Alle Stimmen des Instrumentes werden vom gemeldeten Wert gleichermaßen beeinflußt.

4. Diese Befehle kommen von anderen Steuereinheiten als der Tastatur. Diese Steuereinheiten mit den Nummern $0 bis $1F können Werte (c) zwischen $0 und $7F liefern. Meist handelt es sich dabei um Analog-Bedienungselemente. Die Steuergerät-Nummern von $20 bis $3F können dabei für die Übermittlung einer zweiten Bitgruppe genutzt werden, falls sehr hohe Auflösung der Steuerung erwünscht ist. Die Steuergerät-Nummern von $40 bis $5F sind Schaltgeräte – etwa Sustain-Pedale, Ein/Aus-Schalter oder Schalter für gebundene Klangerzeugung (Portamento). Dafür wird c entweder auf 0 (Aus) oder auf $7F (Ein) gesetzt. Damit entspricht die Funktion etwa einem Schalter-Joystick. Die Steuergerät-Nummern $60 bis $79 sind nicht belegt, während die Nummern zwischen $7A und $7F für Kanal-Modus-Befehle reserviert sind. Die Steuergerät-Nummern sind nicht auf eine bestimmte Art von Steuergerät beschränkt. Ausnahme ist nur die Nummer $1, die für das „modulation-wheel" bestimmt ist.

5. Lokale Steuerung wird aufgerufen, wenn die interne Kopplung zwischen dem Eingabeteil (normalerweise einer Tastatur) und den Schaltkreisen zur Klangerzeugung getrennt werden soll. Dadurch schickt die Tastatur etwa nur Daten über den MIDI-OUT-Anschluß, während die Klangerzeugung ausschließlich auf den MIDI-IN-Eingang reagiert.

6. Die Möglichkeit „Alle Noten aus" ist nur eine Option und sollte nicht als Ersatz des individuellen Abschaltens einzelner Noten verwandt werden. Es ist unverständlich, warum die MIDI-Spezifikationen das vorsehen – die vorangehenden Befehle werden dadurch nahezu völlig unbrauchbar.

7. Das dritte Byte des Mono-Modus gibt die Gesamtzahl der erforderlichen Kanäle an. Ist es Null, wird die Kanalzahl mit den im Empfangsgerät verfügbaren Stimmen gleichgesetzt.

8. Das „pitch wheel" unterscheidet sich insofern von anderen Steuereinheiten, als es positive und negative Werte annehmen kann. Die Mittelposition wird als l=$0 und m=$40 codiert und gesendet. Viele Empfänger haben nur eine Auflösung von 7 Bit und reagieren nur auf den m-Wert. l wird dann übergangen.

Systembefehle

Status	Data1	Data2	Bedeutung	Erklärung
$F0	beliebige Zahl		Systemspezifisch	1
$F1			Nicht definiert	
$F2	l	m	Zeiger für aktuelle Position im Stück l= 7 niederwertige Bits m= 7 höchstwertige Bits	2
$F3	s		Musikstück-Wahl s=Nummer des Stückes	
$F4			Nicht definiert	
$F5			Nicht definiert	
$F6			Befehl „Stimmen"	3
$F7			Systemspezifisch Ende	1
$F8	–	–	Timer	4
$F9			Nicht definiert	
$FA	–	–	Start	4
$FB	–	–	Fortsetzung	4
$FC	–	–	Stop	4
$FD			Nicht definiert	
$FE	–	–	Test „System aktiv?"	5
$FF	–	–	Nullstellung (Reset)	6

Erklärungen:

Die Befehle $F8 bis $FF sind Echtzeitsignale des MIDI-Systems und können jederzeit – selbst während der Übertragung anderer Befehle – gesendet werden.

1. Systemspezifische Befehle bestehen aus einer beliebigen Folge von Datenbytes, die mit „Systemspezifisch Ende" ($F7) oder einem anderen Statusbyte abgeschlossen wird.

2. Dieser Befehl dient zur willkürlichen Einstellung des „Position im Stück"-Zeigers, eines internen Registers, das die Anzahl der „beats" (1 beat=6 MIDI-Taktzyklen) seit dem Anfang des Musikstücks speichert.

3. Damit werden analoge Synthesizer zur Feineinstellung ihrer Oszillatoren (Schwingungserzeuger) aufgefordert – der Synthesizer wird gestimmt.

4. Signale zur Synchronisierung von Haupt- und Nebentaktgebern. Der Systemtakt arbeitet mit dem 24. Teil einer Viertelnote. Der Befehl „Fortsetzen" unterscheidet sich insofern vom „Start"-Befehl, als das Musikstück vom Punkt der Unterbrechung und nicht vom Anfang aus gespielt wird.

5. „System aktiv" ist eine Leernachricht und hat keinen Einfluß auf die MIDI-Signale. Bei MIDI muß alle 300 ms ein Befehl erfolgen. Wenn während eines längeren Zeitraumes keine Befehle eingehen, schaltet das MIDI-Empfangsgerät alle Noten ab und kehrt in den normalen Betriebszustand zurück.

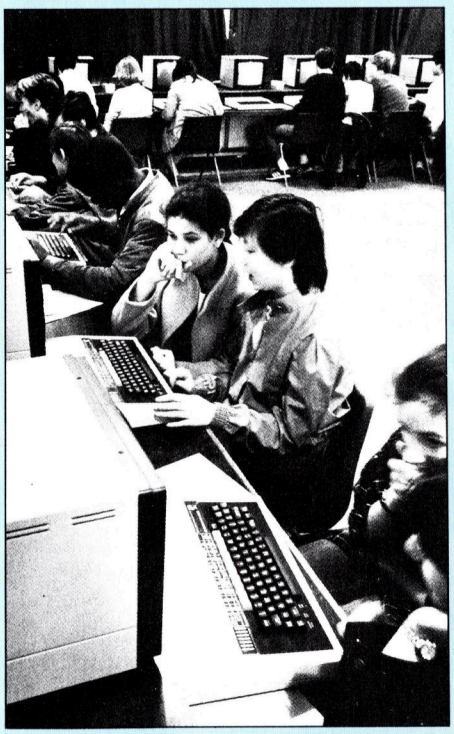
6. Dieser Befehl versetzt das gesamte System in den Grundzustand. Er sollte jedoch nicht automatisch beim Einschalten eines Gerätes erfolgen – das könnte zu einem endlosen Austausch von „Reset"-Befehlen zwischen zwei MIDI-Instrumenten führen.

Andere als die hier aufgeführten Befehle sollten nicht gesendet und von MIDI-Empfängern ignoriert werden.

Fachwörter von A bis Z

Network = Netzwerk
Als Netzwerk wird in der Datenverarbeitung ein Kommunikationssystem zur Verknüpfung mehrerer Rechner und Terminals bezeichnet. Der Datenverkehr kann dabei über Telefonleitungen, Richtfunkstrecken oder Satelliten erfolgen – die räumliche Ausdehnung spielt keine Rolle. Ein Netzwerk kann sich insgesamt auf nur einige PCs im gleichen Büro beschränken oder eine Anzahl Großrechner umfassen, die über die ganze Welt verstreut sind.

Die Bedeutung von Kommunikationsnetzen nimmt auch im Microcomputerbereich ständig zu. Benutzer können im Netzverbund nicht nur untereinander Daten austauschen, sondern bekommen vor allem Zugang zu der enormen Verarbeitungskapazität von Großrechnern und den zugehörigen umfangreichen Informationsangeboten von Datenbanken.



Computernetze werden in zunehmendem Umfang in der Ausbildung von Computeranfängern eingesetzt.

In der Schaltungstechnik versteht man unter einem Netzwerk eine abgegrenzte Bauelement-Gruppe mit spezifischen Eigenschaften. Unterschieden wird dabei zwischen „aktiven" Netzwerken mit verstärkenden Komponenten wie Transistoren und „passiven" Netzwerken, die nur passive Bauteile wie Kondensatoren und Widerstände enthalten. Wenn zwischen Ein- und Ausgangssignal eine lineare Beziehung besteht, spricht man von einem „linearen", andernfalls von einem „nichtlinearen" Netzwerk.

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.

NOR = NOR
NOR ist eine Abkürzung für die logische Operation NOT OR, also das negierte ODER – eine der sechs Booleschen Verknüpfungen, die in der Schaltalgebra Verwendung finden. Ein NOR-Gatter liefert den Ausgangswert TRUE, wenn beide Eingänge FALSE sind, und verhält sich damit umgekehrt wie ein NAND-Gatter. Werden Eingänge und Ausgang eines NOR-Gatters invertiert, ergibt sich die Funktion eine NAND-Gatters, so daß sich wie mit NANDs auch mit NOR-Gattern sämtliche Booleschen Verknüpfungen realisieren lassen.

NOT = Nicht
Das NOT oder die Inversion ist eine weitere Boolesche Operation. Sie bewirkt die Umkehrung eines logischen Zustands in sein Gegenteil – aus TRUE wird FALSE, aus FALSE wird TRUE. Im Unterschied zu den andern Logikgattern gibt es bei einem NICHT-Gatter nur einen einzigen Eingang.

Numerical Analysis = Numerische Auswertung
Bei der numerischen Bearbeitung mathematischer Aufgabenstellungen wird die Lösung durch Zahlenrechnen gesucht. Dabei sind meist Unmengen von Zahlen zu verarbeiten. Zum Bereich der numerischen Mathematik gehört auch die Optimierung der Vorgehensweisen mit dem Ziel, möglichst effiziente und allgemein anwendbare Verfahren zu entwickeln.

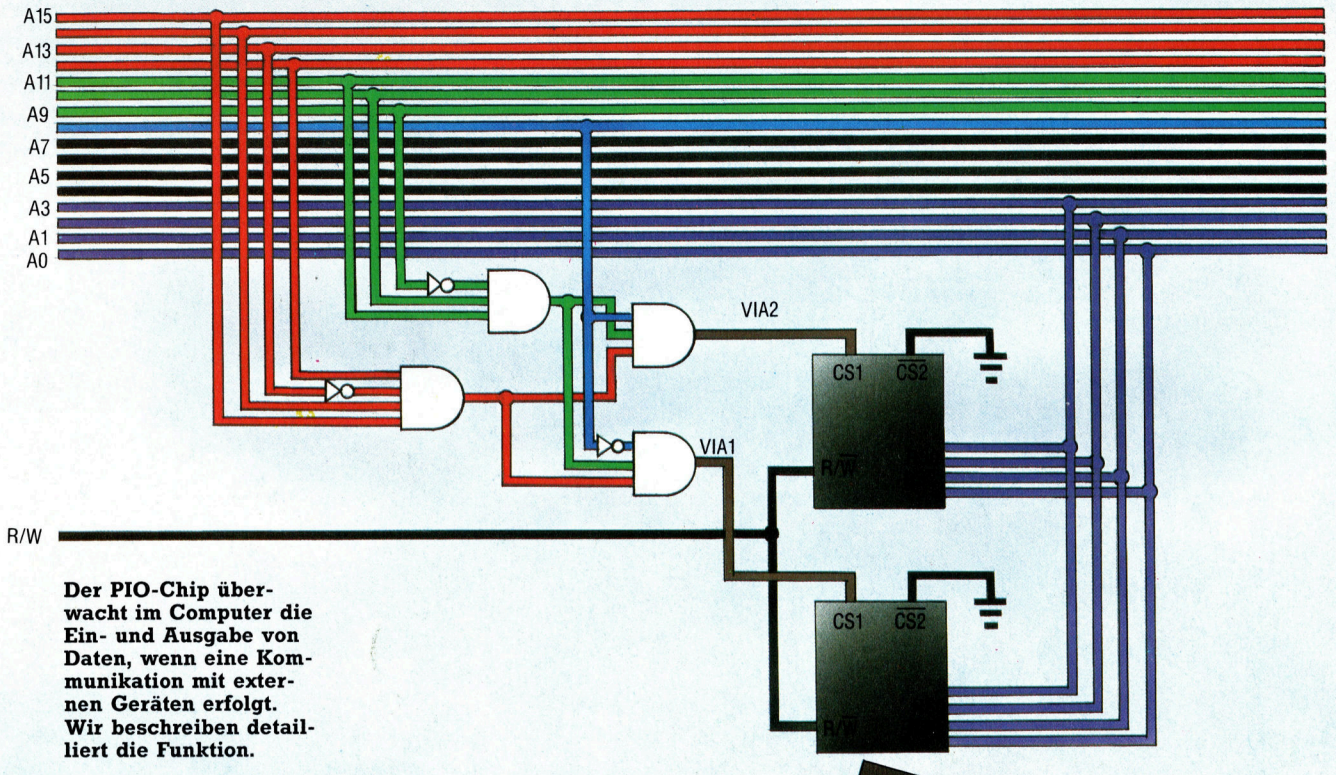
Network Architecture = Netzarchitektur
Die „Architektur" umfaßt die formale und funktionale Struktur eines Kommunikationsnetzes. Funktionale Gesichtspunkte sind unter anderem die Wahl des Übertragungsverfahrens einschließlich Codes und Fehlerkontrolle, die Datenflußsteuerung und die Art der Adressierung der einzelnen Netzknoten. Für den formalen Aufbau ist auch der Begriff „Netzwerktopologie" gebräuchlich – typische Konfiguration sind etwa das Ring- und das Sternnetz.

Noise = Rauschen
Bei den Nachrichtentechnikern und Elektrotechnikern heißt jeder unerwünschte Signalbeitrag Rauschen. Schon beim Schaltungsentwurf ist darauf zu achten, daß Rauscheinflüsse, die sich dem eigentlichen Signal störend überlagern, so klein wie möglich bleiben. Bei Digitalbausteinen der TTL-Familie z. B. wird die logische Eins durch eine Spannung von +5 V, die logische Null durch 0 V dargestellt. Rauschbeiträge können das Signal so stark verfälschen, daß die Zustände „1" und „0" an irgendeinem Gattereingang nicht mehr eindeutig erkennbar sind und Fehler auftreten. Es ist ein wesentliches Ziel der Elektronikentwickler, möglichst hohe „Störschwellen" zu erreichen, das heißt hohe Grenzwerte für die Rauschspannung, die noch ohne Beeinträchtigung der Zuverlässigkeit toleriert wird. Wenn Schaltkreise sehr viel Rauschen ohne Fehlfunktion verkraften, spricht man auch von extremer „Störsicherheit".

Bildnachweise
1933: Paul Chave
1934, 1953, 1958, 1959: Kevin Jones
1938: Liz Dixon
1940: Chris Stevens
1942, 1952: Mike Clowes
1943: Marcus Wilson-Smith
1946, 1949: Graeme Harris
1955: Liz Heany
1956: Caroline Clayton
1960: Tony Sleep

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

computer kurs Heft 71



Der PIO-Chip überwacht im Computer die Ein- und Ausgabe von Daten, wenn eine Kommunikation mit externen Geräten erfolgt. Wir beschreiben detailliert die Funktion.

High Fidelity

In Tips für die Praxis wird ein Programm entwickelt, das eine digitale Aufzeichnung und Wiedergabe von Musikstücken ermöglicht; es läuft auf dem Commodore C 64 ohne Probleme.

Klassenbewußt

Die nächste Folge der Sprache C beschäftigt sich mit den vier „Speicherklassen" dieser interessanten Programmiersprache.

Kanalarbeiter

In diesem Abschnitt über das Betriebssystem des Spectrum geht es um die Kanäle, die Daten zu den Peripherien senden.

Stammvater

Der PET (Personal Electronic Transactor) von Commodore gilt als Stammvater aller PCs. Wir stellen das Gerät vor.