Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

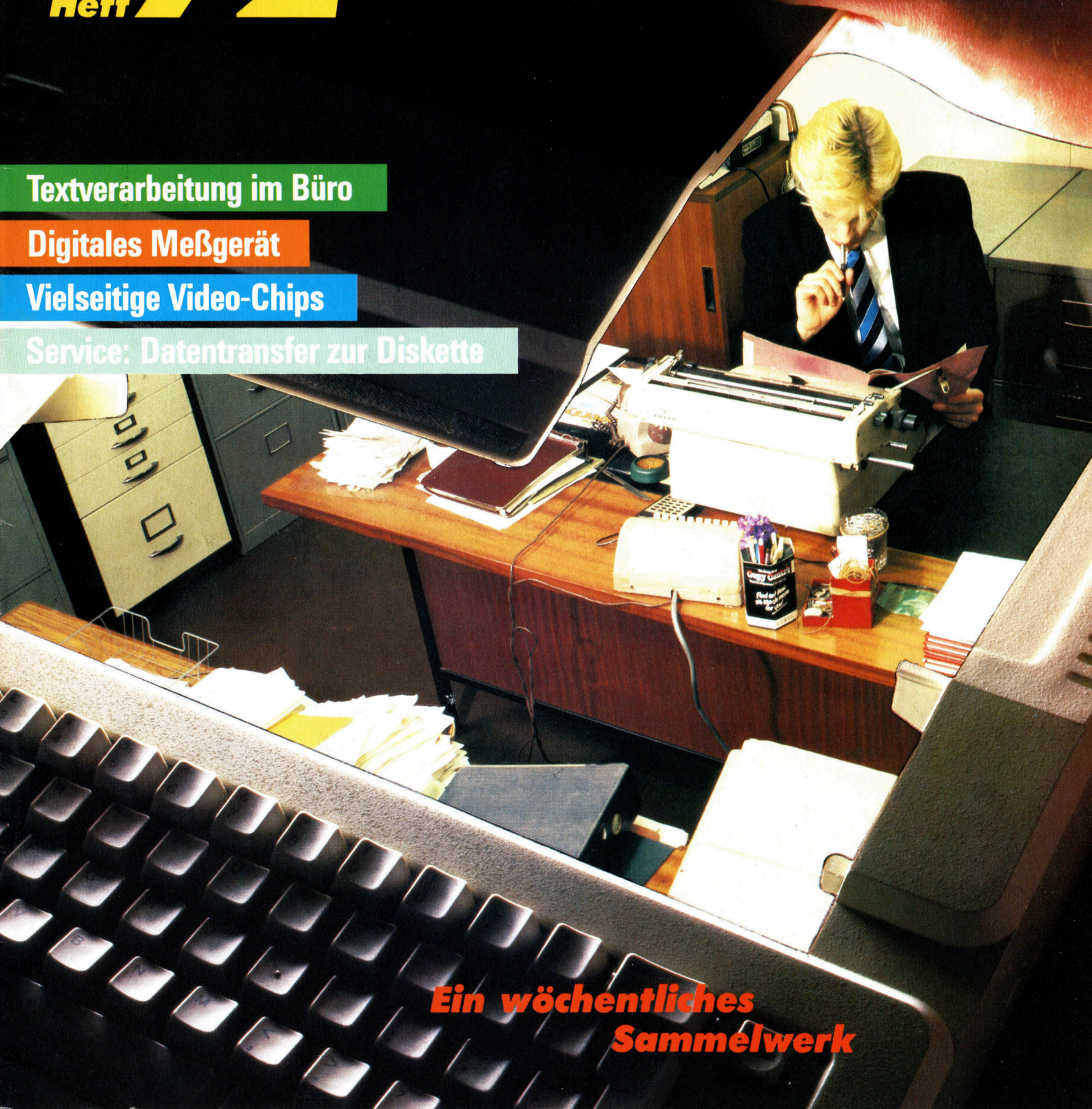
Heft 72

Textverarbeitung im Büro

Digitales Meßgerät

Vielseitige Video-Chips

Service: Datentransfer zur Diskette



Ein wöchentliches Sammelwerk

computer kurs

Heft 72

Inhalt

Bits und Bytes

Bandmaß 1989
ROM-Routinen beim Cassettenbetrieb

Hinter der Maske 2012
Interruptgesteuerte Module

Tips für die Praxis

Digitalmessung 1992
Konstruktion digitaler Meßgeräte

Software

Menü-Vorschläge 1994
Textverarbeitung „WordStar"

Arbeitsplatz am offenen Fenster 2010
Die GEM-Umgebung und ihre Möglichkeiten

Computer Welt

Senkrechtstarter 1997
Softwarehaus „Artic Computing"

Gespeicherter Schirm 2015
Die vielseitigen Video-Chips

Sprache

Wegweiser 1998
Deklaration und Einsatz der Pointer

Programmier-Service

Datentransfer vom Band zur Platte 2000
Konverter für den Commodore

BASIC 72

Wir machen einen Zug 2007
Strategien im Go-Spiel

Fachwörter von A—Z

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.

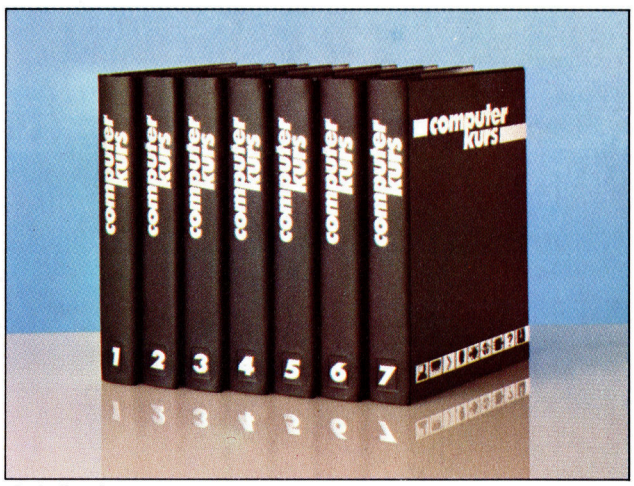
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

Bandmaß

Wir untersuchen, wie das Betriebssystem des Spectrum die Cassetten mit Hilfe von ROM-Routinen anspricht und zeigen, wie sich diese Routinen auch vom Maschinencode aus einsetzen lassen.

Der Standard Sinclair Spectrum besitzt nur eine Cassettenrecorder-Schnittstelle, über die alle Programme und Daten geladen und gesichert werden. Erst der Einbau eines Interface 1 eröffnet den Zugang zu Microdrives, serieller Datenübertragung und lokalen Netzwerken (die – wie die alternativen Kanäle des Acorn B – als zusätzliche Dateisysteme angesehen werden). Während der Acorn B seine Dateisysteme mit dem *-Befehl anwählt, arbeitet der Spectrum jedoch mit einer völlig andersartigen Syntax, um zwischen Befehlen, die für das Cassettensystem gedacht sind und alternativen Dateisystemen unterscheiden zu können. So sichert beispielsweise der Befehl

SAVE "fred1" LINE 1

ein BASIC-Programm auf Cassette, das beim Zurückladen automatisch auf Leitung 1 ausgeführt wird. Der gleiche Befehl für Microdrives ist weit umständlicher:

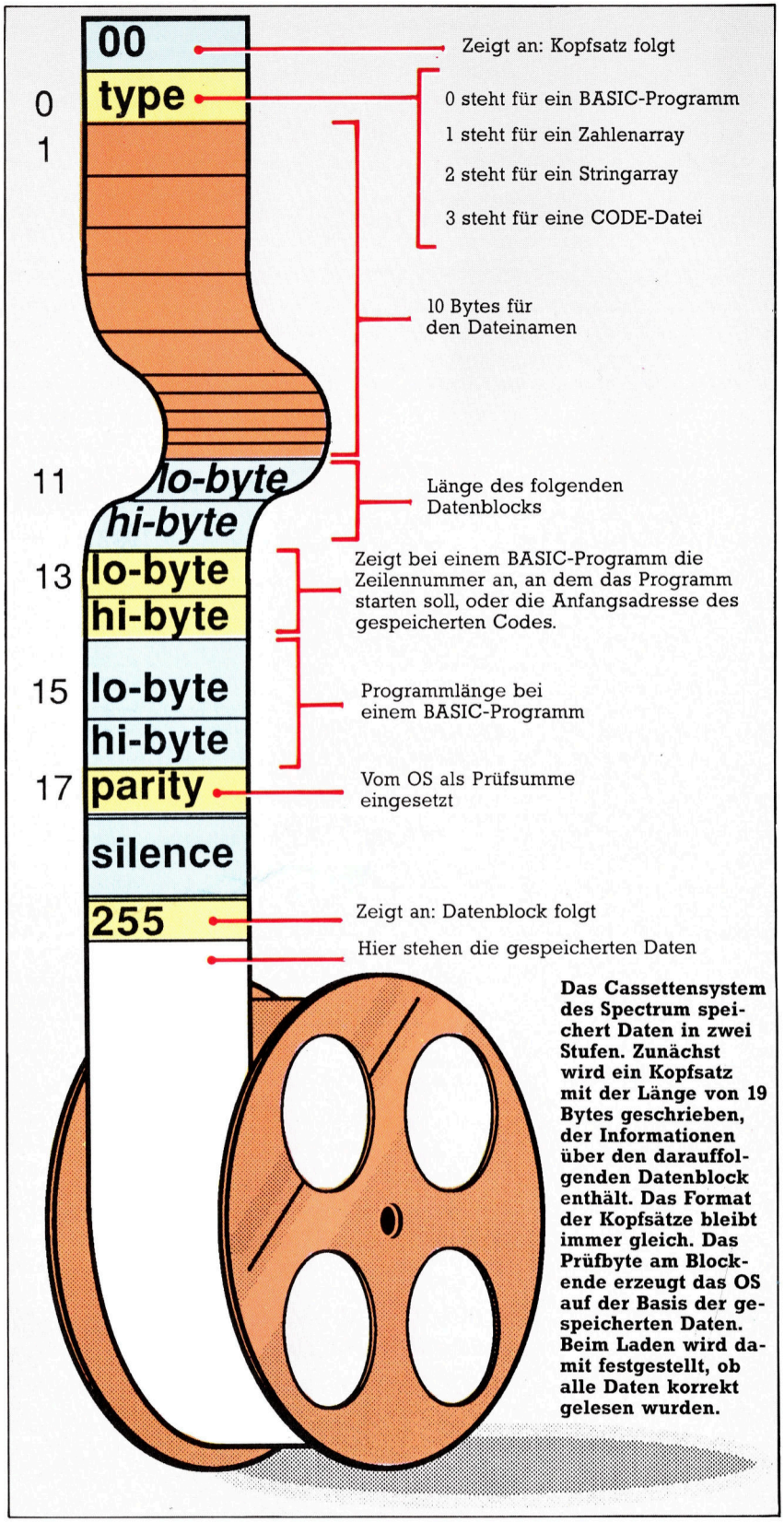
SAVE *"m";1;"fred1" LINE 1

Das „m" gibt an, daß Microdrives angesprochen werden, während die darauffolgende 1 die Nummer des Laufwerks festlegt. Mit der Funktionsweise des Microdrivesystems wollen wir uns jedoch erst später beschäftigen. Zunächst wird demonstriert, wie der Maschinencode das Cassettensystem steuern kann.

Leerzeichen füllen auf

Alle Cassettendaten werden auf die gleiche Art gespeichert – unabhängig von ihrer Struktur (siehe Bild). Die ersten 19 Bytes bilden den Kopfsatz mit Informationen über den darauffolgenden Datenblock. Die jeweils ersten und letzten Bytes des Kopfsatzes werden von Betriebssystemroutinen definiert und auf das Band geschrieben. Die anderen Bytes müssen vor Aufruf des Schreibmoduls definiert werden. Beim Laden zeigt das „Typbyte" dem OS an, welche Art Daten der Block enthält. Darauf folgen zehn Bytes mit dem Dateinamen. Enthält ein Dateiname weniger als zehn Zeichen, werden die restlichen Positionen mit Leerzeichen (ASCII-Code 32) gefüllt. Von Byte 11 an liefert der Kopfsatz dem OS weitere Informationen – beispielsweise in welchen Speicherbereich die Daten geladen werden sollen. Das letzte Byte des Kopfsatzes ist ein Prüfbyte, mit dem getestet wird, ob beim Lesen Fehler aufgetreten sind. In diesem Fall erscheint eine entsprechende Meldung auf dem Bildschirm.

Am Anfang des Hauptdatenblockes steht immer ein Byte mit dem Wert 255, das eben-



Das Cassettensystem des Spectrum speichert Daten in zwei Stufen. Zunächst wird ein Kopfsatz mit der Länge von 19 Bytes geschrieben, der Informationen über den darauffolgenden Datenblock enthält. Das Format der Kopfsätze bleibt immer gleich. Das Prüfbyte am Blockende erzeugt das OS auf der Basis der gespeicherten Daten. Beim Laden wird damit festgestellt, ob alle Daten korrekt gelesen wurden.

falls vom Schreibmodul eingesetzt wird. Danach kommen Daten, die mit einem weiteren, vom Schreibmodul erzeugten Prüfbyte abschließen. Der Inhalt der Prüfbytes hängt von den Daten des davorstehenden Blockes ab.

Wie werden die Cassettenroutinen nun von Maschinencodeprogrammen eingesetzt? Die Routinen für die Cassettensteuerung liegen im ROM des Spectrum zwischen &04C2 und &09F3. Sie sind recht umfangreich, da die Zeitabläufe und die Tonerzeugung für das Beschreiben des Bandes per Software erzeugt werden.

Zwar gibt es keine speziellen Systemvariablen für Cassettenfunktionen, doch zeigt die folgende Tabelle einige Variablen, die für diese Aufgabe eingesetzt werden.

Systemvariable	Beschreibung
TADDR (bei 23668 und 23669)	Entscheidet bei der Interpretation eines BASIC-Befehls, welcher Cassettenvorgang von den ROM-Routinen ausgeführt werden soll.
BORDCR (bei 23624)	Stellt nach einem Cassettenvorgang die Hintergrundfarbe wieder her.
XPTR (bei 23647 und 23648)	Zwischenspeicher für das Register IX

Auch die Systemvariablen VARS, ELINE und PROG werden vom Spectrum für die Cassettenspeicherung verwandt.

Die Einsprungbedingungen

Sehen wir uns zunächst an, wie Daten auf Band gespeichert werden (die Routine liegt bei &04C2). Im Normalfall wird diese Routine zweimal aufgerufen – einmal zur Speicherung des Kopfsatzes und ein zweites Mal zum Schreiben des Datenblocks. Unsere zweite Tabelle gibt die Einsprungbedingungen an.

Register	Kopfsatz	Datenblock
A	0	255
DE	17	Länge der Daten
IX	Anfangsadresse des Kopfsatzes oder des Datenblocks	

Dabei zeigt das IX-Register auf die Daten, die geschrieben werden sollen. In dem folgenden Programmbeispiel schreiben wir einen Datenblock von 100 Bytes (Anfang bei ROM-Adresse 0000) auf die Cassette.

```
        ;routine to save 100 bytes, starting address
         0000, to tape
        ;
        ;send header
3E00            ld   a,0        ;indicate a HEADER BLOCK
DDE5            push ix         ;save ix on stack
DD21D328        ld   ix,header  ;add. of header block
111100          ld   de,17      ;no of header bytes
CDC204          call #04c2      ;write header to tape
        ;send data
DD210000        ld   ix,0000    ;add of 1st byte to save
116400          ld   de,100     ;no of bytes to be saved
3EFF            ld   a,255      ;indicate a DATA BLOCK
CDC204          call #04c2      ;write data to tape
DDE1            pop  ix         ;restore IX value
C9              ret             ;back to BASIC
03      header: defb 3          ;data type 3=CODE block
54455354        defm "TESTPROG" ;f/name+spaces=10 chars
64              defb 100        ;lo-byte of data-length
00              defb 0          ;hi-byte of data-length
00              defb 00         ;lo-byte of start add.
00              defb 00         ;hi-byte of start add.
00              defb 00         ;used for BASIC only
00              defb 00         ;for start line no.
```

Der Aufruf dieser Routine sichert den angegebenen Speicherbereich. Im Gegensatz zu den BASIC-Bandroutinen gibt diese Routine keine Meldung auf dem Schirm aus. Da die Banddaten verschiebbar sind, bietet die Verifizierung keine Schwierigkeiten.

Laden Sie mit dem Befehl

```
LOAD "TESTPROG" CODE 40000
```

das Programm wieder, und vergleichen Sie die geladenen Bytes mit den Speicherstellen des ROM von 0 bis 99. Bedenken Sie aber, daß der Kopfsatz zwar mit dem Typbyte anfängt, das erste auf Band geschriebene Byte jedoch vom Betriebssystem geliefert wird (0 für Kopfsatz und 255 für Datenblock).

Durch eine kleine Veränderung lassen sich auch BASIC-Programme vom Maschinencode aus sichern. Der Kopfsatz am Ende des vorigen Listings muß dann lauten:

```
00      header: defb 0          ;0=BASIC program
73737373        defm "sssss     ;fname filled to 10 chars
                sssss"
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) prog+variables
00              defb nn         ;lo-byte) start line
00              defb nn         ;hi-byte) number
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) program only
```

Die Länge von Programm und Variablen erhalten Sie, wenn Sie den Wert PROG von ELINE abziehen. Die Programmlänge ergibt sich, wenn Sie PROG von VARS subtrahieren. Falls das Programm nicht sofort nach dem Laden starten soll, brauchen Sie im Listing nur die Zeilennummer des Programmanfangs auf 32768 zu setzen. Auf ähnliche Weise kann man auch den Befehl SAVE SCREEN$ vom Maschinencode aus simulieren. Zu diesem Zweck muß man die Anfangsadresse des Codes, der gesichert werden soll, auf %4000 setzen und die Länge mit &1B00 angeben.

Mit den Techniken, die wir früher im Kurs behandelt haben, läßt sich die Routine leicht ausbauen. So können Sie beispielsweise einen Prompt erzeugen und die Routine mit einem einfachen Zusatzmodul auf einen folgenden

Tastendruck warten lassen.

Auch die ROM-Routine (bei&0556), die Daten vom Band lädt, wird zweimal aufgerufen: einmal für den Kopfsatz und ein zweites Mal für den Datenblock. Die folgende Tabelle zeigt die Bedingungen für den Einsprung und die Codeverifizierung:

Register	Laden	Verifizieren
C-Flag	1	0
A	0 für Kopfsatz, 255 für Datenblock	
IX	Zeigt auf die Speicheradresse, von der an die Bytes geladen werden sollen	
DE	Anzahl der zu ladenden Bytes; D muß im Bereich von 0 bis 254 liegen	

Die Routine belegt im Arbeitsspeicher 34 Bytes – den Platz für zwei Kopfsätze. Um eine Datei an eine bestimmte RAM-Adresse laden zu können, muß zunächst ein zweiter Kopfsatz mit den Steuerdaten der Banddatei angelegt werden. Dieser zweite Kopfsatz wird dann mit dem vom Band gelesenen Kopfsatz verglichen und so festgestellt, ob dies überhaupt die gewünschten Daten sind.

C-Flag auf Wert setzen

Beim Aufruf der Routine (bei &0556) muß das C-Flag auf den in der Tabelle aufgeführten Wert gesetzt werden. Codeabschnitte, die man verifizieren möchte, sollten bei der Adresse anfangen, auf die das IX-Register zeigt. Beim Verlassen der ROM-Laderoutine enthält das C-Flag das Ergebnis des Ladevorgangs. Steht C auf Eins, war das Laden erfolgreich. Wenn Sie jedoch einen Kopfsatz laden wollen und auf dem Band zuerst einen Datenblock finden, wird das C-Flag auf Null gesetzt. (Ladefehler werden vom OS abgefangen.) Hier ein Beispiel für das Laden eines Kopfsatzes:

```
        ;load a header from tape into RAM
        ;
37              scf             ;set carry 1=LOAD
3E00            ld   a,0        ;0 indicates a header
DDE5            push ix         ;save ix on stack
DD2148EE        ld   ix,61000   ;load header to 61000
111100          ld   de,17      ;no of bytes in header
CD5605          call #0556      ;do it
DDE1            pop  ix         ;restore ix
C9              ret
```

Unmittelbar nach dem Laden wird der Kopfsatz mit dem im RAM angelegten Kopfsatz verglichen und überprüft, ob Dateiname, Dateityp und – falls nötig – die Länge des Blocks übereinstimmen. Das folgende Programm prüft nur den Namen. Falls er mit dem angegebenen identisch ist, wird der darauffolgende Datenblock bei der zuvor definierten Adresse ins RAM geladen.

```
        ;locate and load TESTPROG
        ;
DDE5            push ix         ;save IX on stack
37      loop:   scf             ;set carry for LOAD
3E00            ld   a,0        ;0=BASIC file
111100          ld   de,17      ;no of bytes in header
DD21CA2B        ld   ix,head2   ;load header into head2
CD5605          call #0556
D27B2B          jp   nc,loop    ;if no header then again
060A            ld   b,10       ;no of bytes in fname
11BA2B          ld   de,head+1  ;point to desired f/name
21CB2B          ld   hl,head2+1 ;point HL to found fname
1A      name:   ld   a,(de)     ;get header char in a
BE              cp   (hl)       ;comp with found header
2007            jr   nz,no      ;different so exit loop
13              inc  de         ;get next char
23              inc  hl
05              dec  b          ;decrement counter
20F7            jr   nz,name    ;same so check next char
1802            jr   ok         ;all chars checked + ok
18DB    no:     jr   loop       ;check failed, try again
        ;found right header so prepare to load data
DD21B92B ok:    ld   ix,head    ;get head in ix
DD6E0D          ld   l,(ix+13)  ;get address to
DD660E          ld   h,(ix+14)  ;load data into
E5              push hl         ;push address onto stack
DDE1            pop  ix         ;and get it into ix
3EFF            ld   a,255      ;indicate load data
        ;next instruction requires user to insert data
         length
11AF2B          ld   de,nn      ;insert data length
37              scf             ;indicate a LOAD
CD5605          call #0556      ;doit
DDE1            pop  ix         ;restore ix
C9              ret
        ;now follow details of desired f/name
03      head:   defb 3          ;indicates CODE block
54455354        defm "TESTPROG" ;f/name
00              defb 0          ;use for data length
00              defb 0          ;data-length hi-byte
48              defb 72         ;lo-byte for load
EE              defb 238        ;address of 61000
00              defb 0          ;used for BAS. prog only
00              defb 0          ; ditto
        ;now follows space for header loaded for
         checking
        head2:  defs 17
```

Auch hier werden keine Meldungen an den Schirm ausgegeben. Die Routine lädt einen Datenblock namens TESTPROG in den in HEAD angegebenen Speicherbereich. Zusätzlich sollte man prüfen, ob auch der Dateityp stimmt. Ähnlich wie bei LOAD und SAVE (in BASIC) können auch diese ROM-Routinen durch Drücken der Break-Taste einfach beendet werden.

Wir haben am Schluß dieser Folge ein kurzes Programm abgedruckt, das einen Kopfsatz vom Band liest und Informationen über die Datei auf dem Bildschirm ausgibt – Dateilänge, Anfangsadresse etc. Der Maschinencode ist in DATA-Befehlen untergebracht. Er lädt einen Kopfsatz und untersucht ihn ohne weitere Zwischenschritte von BASIC aus.

```
5 CLEAR 59999
10 FOR I=0 TO 19
20 READ A:POKE (60000+I),A
30 NEXT I
40 RANDOMIZE USR 60000
50 LET type=PEEK 60020
60 LET length=PEEK 60031+256*PEEK 60032
70 LET start=PEEK 60033+256*PEEK 60034
80 LET L$=""
90 FOR I=60021 TO 60030: LET L$=L$+CHR$(PEEK I): NEXT I
100 PRINT "Name: ";L$
110 IF type=0 THEN LET f$="BASIC"
120 IF type=1 THEN LET f$="Number Array"
130 IF type=2 THEN LET f$="String Array"
140 IF type=3 THEN LET f$="CODE"
150 PRINT "Type: ";f$
160 IF type=0 THEN PRINT "Auto Run line number: ";start
170 IF type<>0 THEN PRINT "Start address: ";start
180 PRINT "Length: ";length
190 GOTO 40
200 DATA 221,229,62,0,55,221,33,116,234,17,17,0,205,86,
    5,48,241,221,225,201
```

Digitalmessung

Mit dieser Folge starten wir einen neuen Abschnitt im Selbstbau-Kurs, bei dem es um die Konstruktion und Arbeitsweise digitaler Vielfach-Meßgeräte gehen wird.

Normalerweise arbeitet ein Computer vollständig digital. Auch die Tastatur und der Videoausgang als „Schnittstelle" zum Menschen funktionieren nur durch Digitalschaltungen. Die Nutzbarkeit eines Computers jedoch wächst entsprechend seinen Fähigkeiten, mit der Außenwelt in Verbindung zu treten – einer Außenwelt, die größtenteils nicht dem Funktionskonzept der Maschine entspricht, sondern analog strukturiert ist: Spannungen sind immens variabel, ebenso wie Temperaturen, Höhen, Gewichte usw. Zum Erfassen von Daten oder zur Steuerung externer Geräte muß daher eine Umwandlung digitaler in analoge Werte und umgekehrt vorgenommen werden.

Ein Computer, der mit einem Temperaturfühler verbunden ist, muß beispielsweise Digitalzahlen erzeugen, die der gemessenen Temperatur entsprechen – andernfalls können die Meßdaten nicht vernünftig ausgewertet werden. Dazu wird ein „Analog-Digitalwandler (A/D-Wandler)" eingesetzt.

Das Gegenstück dazu, eine Einheit zur Verwandlung digitaler Werte in analoge Signale, wird immer dann eingesetzt, wenn ein digitales Gerät ein analoges Gerät steuern soll. Als Beispiel kann uns ein Rechner dienen, der für die Erzeugung von Musik programmiert wurde. Der Computer selbst erzeugt dabei nur einen digitalen (binären) Wert, der in entsprechende akustische (analoge) Signale umgesetzt werden muß. Das besorgt ein D/A-Wandler, der binäre Werte in Klänge verwandelt.

„Do it Yourselfer" gefordert

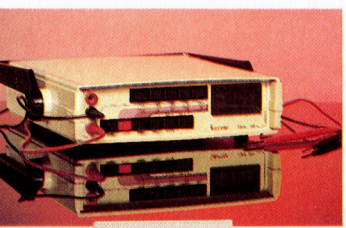
Ein Multimeter ist ein Gerät zum Messen von Widerständen (in Ohm), Stromstärke (in Ampere) oder elektrische Spannung (in Volt). In dieser Folge wollen wir ein digitales Multimeter bauen, das diese Werte sehr exakt feststellen und die Ergebnisse digital auf einem LED- oder LCD-Display darstellen kann. Ein solches Vorhaben würde die Fähigkeiten der meisten „Do-it-Yourselfer" übersteigen, wenn nicht in den letzten Jahren die Entwicklung hochintegrierter Chips hier vieles erleichtert hätte. Unser DVM (DigitalVoltMeter) soll so aufgebaut sein, daß es nicht nur allein funktionsfähig ist, sondern seine Meßergebnisse zur Weiterverarbeitung auch an den Computer übergibt.

Bei einem herkömmlichen Voltmeter wird ein beweglich über einer Skala aufgehängter Zeiger verwendet, der durch seinen Ausschlag die gemessene Spannung, Stromstärke oder den Widerstand anzeigt. Ein Netzwerk aus Vor- und Nebenwiderständen sorgt dafür, daß die Anzeige den unterschiedlichen Meßbereichen (Volt, Ampere und Ohm) entspricht, obwohl die Grundfunktion des Meßwerkes immer die gleiche ist: Strom durchfließt eine Spule, die sich in einem Magnetfeld befindet. Die Stärke des fließenden Stromes bestimmt, wieweit sich die Spule aus ihrer Ruhelage entgegen der Kraft einer Feder bewegt. Ein mit der Spule verbundener Zeiger macht diese Abweichung aus der Ruhelage sichtbar.

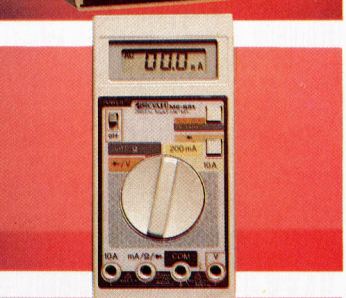
Wenn eine unbekannte Spannung digital gemessen werden soll, gibt es sehr viel größere Schwierigkeiten. Bevor wir uns näher mit diesem Problem auseinandersetzen, müssen wir genauer analysieren, wie ein digitaler Wert in einen analogen umgewandelt wird:

Digital/Analogwandlung ist eigentlich ein recht einfacher Vorgang. Stellen Sie sich vor, Sie wollten ein Acht-Bit-Wort (eine Reihe von acht Binärziffern) in eine analoge Spannung

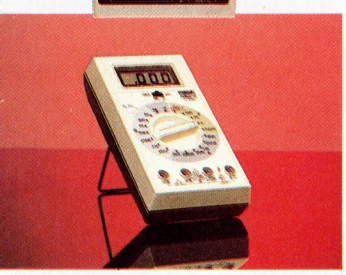
Multimeter-Marktübersicht
Hier sollen am Beispiel dreier Digitalmultimeter die Leistungsmerkmale der verschiedenen Geräteklassen dargestellt werden:



Lascar LMM 100	**Laborgerät**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	25 Bereiche
Meßgenauigkeit:	0,1 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Meßwerterhalt, Polarität, Batterieanzeige und Anzeige bei Bereichsüberschreitung.



Soar ME-531	**Meßgerät mit automatischer Bereichswahl**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	Autom. Bereichswahl
Meßgenauigkeit:	0,8 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Polarität, Überlastanzeige, Diodenprüfung



Precision Gold M-5010	**Einfaches Digitalmultimeter**
Anzeige:	3 1/2-stellig LCD
Meßbereich:	29 Bereiche
Meßgenauigkeit:	0,25 % (DC Volt)
Versorgung:	Batterie
Weitere Merkmale:	Durchgangsprüfer, Überlast- und Polaritätsanzeige, Diodenprüfung

zwischen 0 und 1 Volt umwandeln. Ein Acht-Bit-Wort kann einen Wert zwischen 0 und 255 darstellen (binär 00000000 bis 11111111). Der 1-Volt-Bereich läßt sich also in 256 Spannungsschritte oder Schritte von 0,0039 Volt auflösen.

Binäre Werte lassen sich „analogisieren", indem man sie zum Schalten von Spannungen nutzt. Die Zeichnungen stellen zwei Grundschaltungen für diese Funktion dar. In beiden Fällen werden die mechanischen Schalter elektronisch durch binäre Logikwerte bestätigt. Eins bedeutet dabei „geschlossen", Null „offen". Jede Binärziffer schließt einen der Schalter und vergrößert so den Stromfluß.

Die Analog/Digitalwandlung ist schwieriger. Wie bei der D/A-Wandlung bestimmt auch hier die Bitzahl die Genauigkeit (Auflösung) der Umwandlung oder Messung. Für unser Digitalvoltmeter ist eine Genauigkeit von acht Bit völlig ausreichend.

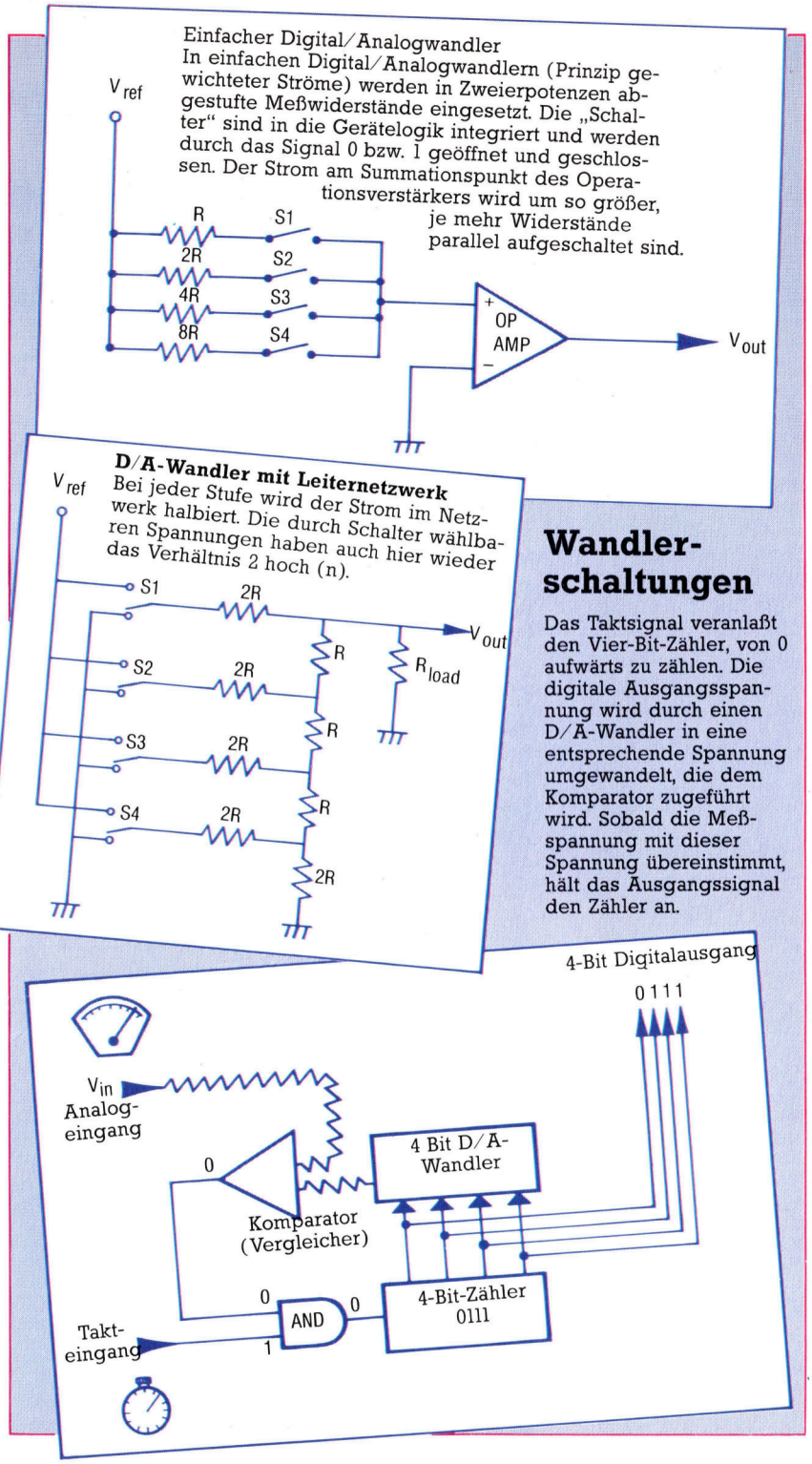
Die Schaltung soll in einem Basis-Arbeitsbereich von 0 bis 2 Volt funktionieren. Acht-Bit-Auflösung erlaubt dann Spannungsstufen von 0,0078 Volt. Der Meßbereich läßt sich durch ein einfaches Widerstandsnetzwerk auf 0 bis 20 Volt bzw. 0 bis 200 Volt erweitern.

Analog/Digitalwandlung ist erheblich schwieriger und damit auch teuerer als der umgekehrte Vorgang. Der Aufwand hängt davon ab, wie groß die Auflösung sein soll und wie schnell die Wandlung durchgeführt werden muß. Falls man Audio-Signale in einem Frequenzbereich von bis zu 15 kHz in digitale Signale umwandeln will, müssen in jeder Sekunde bis zu 30 000 Meßwerte genommen werden. Noch größer ist diese „Abtastrate" bei der Digitalisierung von Videosignalen. Zum Glück muß ein Digitalvoltmeter nur Gleich- und Wechselspannungen mit vergleichsweise niedriger Frequenz messen.

Integrierte Komparator-ICs

Die am häufigsten verwendete Methode ist der Vergleich einer bekannten Spannung (Referenzspannung) mit der unbekannten Spannung (Meßspannung). Integrierte Komparator-ICs sind für diesen Zweck ideal – bei entsprechender Beschaltung erzeugen sie bei Übereinstimmung der beiden angelegten Spannungen ein Ausgangssignal. Ein Binärzähler wird auf den Anfangswert 0 gesetzt und zählt von dort aufwärts, wobei er (durch die oben besprochene D/A-Technik) eine Spannung am Ausgang erzeugt, die irgendwann einmal zufällig mit der Meßspannung übereinstimmt. Das Ausgangssignal des Komparators hält dann den Zähler an, und die Meßspannung kann berechnet werden.

Es gibt zwar eine ganze Reihe integrierter Multimeter-Schaltungen, sie lassen sich aber im allgemeinen nur schwer mit einem Computer verbinden – ihre Ausgangsleitungen sind nur für eine LED- bzw. LCD-Anzeige geeignet.



Wir haben uns daher für den Einsatz des ICL 7135 von der Firma Analog Systems entschieden. Der binär-codierte dezimale Ausgangswert dieses ICs eignet sich nicht nur für die Ansteuerung eines Sieben-Segment-Displays, sondern auch für die Rechner-Kopplung.

Das IC hat einen sehr hohen Eingangswiderstand. Der Differenzeingang erlaubt die Messung sowohl positiver als auch negativer Spannungen. Außerdem gibt es einen automatischen Nullabgleich. Bereichsüber- und -unterschreitung sowie die Polarität der angelegten Spannung werden angezeigt.

Menü-Vorschläge

‚WordStar' – ein Programm von MicroPro aus den siebziger Jahren, das aber in seiner Klasse immer noch als führend gilt. WordStar ist stark CP/M-orientiert und stellt mit seinem vielseitigen Leistungsangebot ein außerordentlich wirkungsvolles Werkzeug dar.

In den letzten zehn Jahren wurden hunderte von Textverarbeitungsprogrammen geschrieben – jedes angeblich besser als alle Vorgänger. Das bekannteste Programm in diesem Bereich ist aber nach wie vor eins der ältesten Angebote: WordStar von MicroPro. Es wurde ursprünglich für CP/M-Rechner entwikkelt und 1978 kurz nach der Gründung von MicroPro erstmals vorgestellt. Später wurde das Programm für PC-DOS und MS-DOS umgeschrieben, um den großen Markt der IBM PCs und zahlloser Kompatible zu erreichen. Der Grund für den langjährigen Erfolg von WordStar ist in den vielfältigen Textgestaltungsmöglichkeiten zu sehen, denn das Programm ist für heutige Verhältnisse nicht einmal besonders benutzerfreundlich. Damals war es das erste, das auf dem Bildschirm die Unterstützung durch ‚Help'-Menüs anbot.

‚Preliminary Commands'

Wie schon erwähnt hat WordStar seine Wurzeln in CP/M und hat dessen starke wie auch schwache Seite geerbt. WordStar wird nach dem Laden auf der Liste der ‚transienten' CP/M-Segmente geführt, profitiert also von allen Vorteilen des CP/M-Diskettenbetriebs. Und wie CP/M macht auch WordStar ausgiebig Gebrauch von Steuerzeichen.

Nach Laden und Programmstart erscheint auf dem Bildschirm das Eröffnungsmenü (Opening Menu). Es enthält eine Liste von Wahlmöglichkeiten und das Dateiverzeichnis der gerade angesprochenen Diskette mit sämtlichen darauf gespeicherten Files. WordStar-Dateinamen haben das gleiche Format wie bei CP/M, nämlich acht Zeichen mit nachfolgendem Punkt und maximal drei Ergänzungsbuchstaben. Das Eröffnungsmenü enthält 13 Befehle. Unter ‚Preliminary Commands' (Vorbereitungen) finden Sie Befehle zum Wechseln des Laufwerks (Change Logged Disk Drive), zum Ein- und Ausschalten des Inhaltsverzeichnisses (File Directory On/Off), zur Wahl der Hilfsinformations-Stufe (Help Level) usw. Nach dem Booten ist automatisch Stufe 2 eingestellt. Der Anwender hat dabei die Möglichkeit, zwischen vier Hilfsstufen zu wählen (von 0 – alle Menüs und Erklärungen werden angezeigt bis 3 – alle Erklärungen werden unterdrückt).

Wesentliche Merkmale

Als Maßstab für einen Vergleich verschiedener Textverarbeitungspakete sind unten elf Punkte zusammengestellt. Anhand dieser Kriterien lassen sich die vorgestellten Systeme dann einheitlich bewerten.

Zeilenumbruch
Das Programm schaltet am Zeilenende, wenn ein Wort über den Rand stehen würde, automatisch auf den Beginn einer neuen Zeile um.

Blockbearbeitung
Der Benutzer kann im Text ‚Blöcke' definieren, mit denen er unabhängig vom übrigen Schriftsatz beliebig hantieren kann.

Bildschirmhilfe
Der Benutzer wird mittels Bildschirm-Informationen angeleitet, wie die verfügbaren Befehle einzusetzen sind.

80-Zeichen-Bildschirm
Das Textverarbeitungsprogramm sollte möglichst große Textausschnitte auf dem Schirm sichtbar machen können – 80 Zeichen pro Zeile sind Minimum.

Wortzähler
Am Wortzähler ist abzulesen, wieviel Text bisher eingegeben wurde.

Suchen und Ersetzen
Vorgegebene Zeichen, Wörter oder Sätze werden gesucht und auf Wunsch ausgewechselt.

Druckgetreues Textbild
Der Schriftsatz kann auf dem Bildschirm formatiert und genauso dargestellt werden, wie er später auf dem Papier erscheint.

Serienbriefe
Serienbriefprogramme gibt es als integralen Bestandteil oder als Zusatz zur Textverarbeitung. Sie ermöglichen Standardtexte mit individueller Anschrift und Anrede zu versehen.

Rechtschreibkontrolle
Ähnlich wie für die Serienbrief-Gestaltung werden auch Zusatzprogramme zur Prüfung der Orthografie angeboten. Dabei wird jedes Wort des Textes anhand eines gespeicherten Lexikons kontrolliert.

Schriftarten
Typenauswahl ist mehr eine Sache des Druckers. Viele Textverarbeitungssysteme unterstützen zumindest Fett- und Kursivschrift.

Dateikettung
Die Länge von Textdateien ist durch die Speicherkapazität begrenzt. Daher ist es bei der Erstellung umfangreicher Schriftstücke vorteilhaft, wenn Dateien verbunden werden können.

Wandlungen eines Stars

Angesichts der großen Beliebtheit von WordStar hat MicroPro die ursprüngliche CP/M-Fassung mehrfach für unterschiedliche Bedürfnisse umgeschrieben. In den letzten Jahren fand vor allem die Version ‚WordStar Professional' große Verbreitung, die unter MS-DOS läuft und somit beispielsweise für die Apricot-Rechner und den IBM PC in Frage kommt.

Für gehobenere Ansprüche hat MicroPro für IBM-kompatible PCs den ‚WordStar 2000' herausgebracht – ein Programm, das äußerlich noch eine gewisse Ähnlichkeit mit seinen Vorfahren hat, aber im übrigen als ‚Rolls Royce der Textverarbeitung' gelten kann. Es umfaßt fünf Disketten mit insgesamt zwei Megabyte an Programmen und Dateien. Das neuste Programm von MicroPro trägt die Bezeichnung „Easy" und ist vornehmlich für Einsteiger in die Textverarbeitung gedacht.

Aber auch die CP/M-Version hat durchaus noch Zukunft: MicroPro hat unter der Bezeichnung ‚Pocket WordStar' eine „Taschenausgabe" geschaffen, die für die Schneider-Modelle CPC 464 und 664 gedacht ist. Da diese Rechner für hochauflösende Grafik eingerichtet sind, bleibt von ihrem 64K-RAM zu wenig Platz frei, um mit dem vollständigen WordStar-Paket sinnvoll arbeiten zu können.

Sie können nun nach Wunsch eine –Textdatei (Document File) eröffnen bzw. bearbeiten oder eine Programm-Datei, je nachdem, ob Sie einen Text oder ein Programm erstellen wollen. Weiterhin enthält das Menü eine Anzahl von Dateibefehlen zum Ausdrucken, Umbenennen, Kopieren und Löschen von Dateien.

Zum Schluß werden noch Systemkommandos und ‚WordStar Options' angeboten. Die ‚System Commands' bilden die Brücke zu CP/M; man kann damit WordStar verlassen und ein auf der Diskette gespeichertes Programm starten oder auf die Betriebssystem-Ebene übergehen. Die verfügbaren Optionen hängen von der Software-Ausstattung ab. Vom Menü lassen sich die Programme „Mail Merge" oder „Spell Star" aufrufen. Mail Merge dient zum Erstellen von Serienbriefen mit individuellen Anreden und Adressen und ‚SpellStar' zur Kontrolle der Schreibweise aller Wörter in einer Textdatei anhand eines gespeicherten Lexikons. Jedes Wort, das nicht buchstabengetreu in diesem ‚Duden' wiederzufinden ist, wird dem Benutzer zur Überprüfung präsentiert.

Textdatei eröffnen

Nachdem Sie mittels des Kommandos ‚D' eine Textdatei neu oder wieder eröffnet haben, wird auf das Hauptmenü umgeschaltet. Dies zeigt in der Statuszeile das aktivierte Laufwerk und den Namen der gerade bearbeiteten Datei, ferner die aktuelle Cursorposition sowie eine Information, ob die Einfügungsfunktion tatsächlich eingeschaltet ist oder nicht (Insert On/Off).

In der Hilfsstufe 3 erscheint zudem eine umfangreiche Befehlsliste. Aus der im Menü erklärten Cursorsteuerung läßt sich wiederum die CP/M-Herkunft ablesen. Die alten CP/M-Rechner hatten nämlich keine speziellen Cursortasten, und die Steuerung erfolgte, indem man gleichzeitig ‚Control' und eine zweite Taste drückte – für links/rechts/oben/unten wurde durch die Tasten S/D/E/X ein Cursorkreuz simuliert.

Mit „CTRL B" formatieren

Des weiteren enthält die Befehlsliste unter anderem das Kommando „CTRL B" zum Formatieren. Damit läßt sich ein Absatz auf dem Bildschirm wunschgemäß ausrichten. Bei diesem Vorgang gibt das Programm Trennvorschläge. Die Schirmdarstellung entspricht dem Ausdruck, der später erstellt wird, es sei denn, der Drucker akzeptiert die Formatierung nicht.

Zusätzlich erscheinen im Hauptmenü die Titel der Untermenüs. Das ‚Help'-Menü unterstützt Sie mit einer ausführlichen Beschreibung von Anwendung und Funktion einzelner WordStar-Befehle. Das ‚Quick'-Menü bietet u. a. Kommandos zum Suchen und Ersetzen von Texten (Find and Replace) sowie für schnelle Lösch- und Wiederholfunktionen.

Das ‚Onscreen'-Menü enthält vielseitige Befehle für die Wahl des Textformats am Bildschirm durch Setzen von Randstellern, Tabulatorstopps, Zeilenabstand usw. Sehr nützlich ist auch die Seitenumbruch-Anzeige: Das Seitenende wird durch eine punktierte Linie markiert. Ferner gibt es eine ‚Wordwrap'-Funktion für den automatischen Zeilenumbruch. Sie sorgt dafür, daß am Zeilenende bei einem Wort, das über den Rand stehen würde, ein Wagenrücklauf mit Zeilenschaltung erfolgt.

Das dritte Untermenü, das Block-Menü, enthält mit einer Reihe von ‚Block Operations' die nützlichsten und leistungsfähigsten WordStar-Kommandos und bietet die Möglichkeit, ganze Textblöcke innerhalb eines Schriftstückes neu zu positionieren. Die Blöcke werden einfach durch Control-Kommandos zu Beginn und Ende eines Abschnitts definiert und auf dem Bildschirm durch eine negative Darstellung kenntlich gemacht (‚Reverse Video'). Nach dem Abgrenzen können Sie einen Block beliebig verschieben, löschen, an eine andere Stelle der Textdatei kopieren oder sogar in eine andere Datei auf der Diskette übertragen. Und diese Freizügigkeit gilt bei WordStar nicht nur für Zeilenblöcke, sondern genauso für Kolumnen, z. B. bei senkrechten Tabellen oder mehrspaltigen Schriftsätzen.

Im Block-Menü stehen auch die Save-Kommandos zum Sichern von Text, wahlweise mit anschließender Rückkehr zur Eingabe, zum

ZEILENUMBRUCH
WordStar bietet Zeilenumbruch bei der Texteingabe an, formatiert den bereits eingegebenen Text nach Verändern der Randbreiten nicht automatisch.

BLOCKBEARBEITUNG
WordStar erlaubt die Definition von Blöcken beliebiger Größe und freizügigen Umgang damit.

BILDSCHIRMHILFE
Die Hilfsmenüs sind ein typisches WordStar-Kennzeichen und mit ausschlaggebend für die große Beliebtheit.

80-ZEICHEN-BILDSCHIRM
WordStar unterstützt nicht nur das 80-Zeichen-Format, sondern läßt sogar eine Gesamtzeilenlänge von 255 Anschlägen zu.

WORTZÄHLER
Ist nicht vorgesehen.

SUCHEN UND ERSETZEN
Bei WordStar gibt es verschiedene ‚Find and Replace'-Befehle; es sind Strings mit bis zu 30 Zeichen lokalisierbar.

DRUCKGETREUES TEXTBILD
Wegen der Formatierungsmöglichkeiten am Bildschirm hat WordStar viel Beifall geerntet.

SERIENBRIEFE
WordStar war eins der ersten Textverarbeitungssysteme, das die Gestaltung von Serienbriefen ermöglichte.

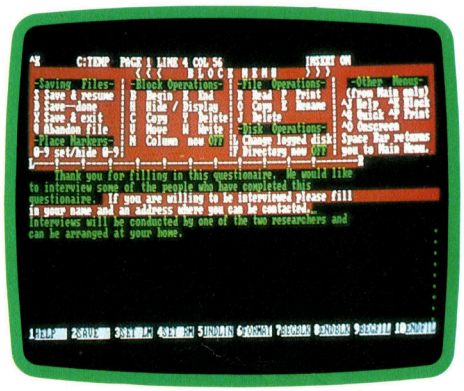
RECHTSCHREIBKONTROLLE
Für die Prüfung der Orthografie sind im ‚SpellStar'-Lexikon rund 20 000 Wörter gespeichert.

SCHRIFTARTEN
Zu den Schriftvarianten gehören Fett- und Kursivdruck.

DATEIKETTUNG
Einzeldateien können ausgedruckt werden, Suchen und Ersetzen über mehrere Dateien ist nicht vorgesehen.

A la Carte

Charakteristisch für WordStar sind Hilfsmenüs im oberen Teil des Bildschirms. Erfahrene Benutzer verzichten meist darauf, für Neulinge aber bedeuten sie eine große Erleichterung, da ständig eine Liste verfügbarer Befehle im Blickfeld liegt.

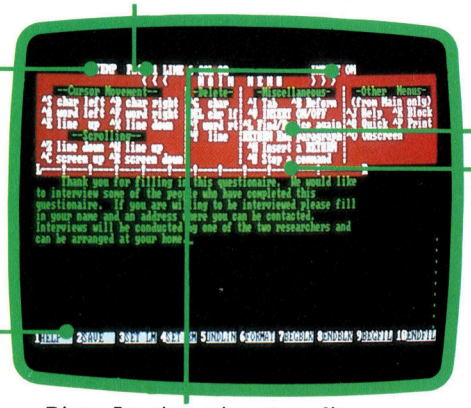


Jedes der hier abgebildeten Menüs zeigt einen andern Befehlsvorrat. Das Eröffnungsmenü (links oben), das unmittelbar nach dem Laden von WordStar präsentiert wird, enthält überwiegend Systemkommandos für die Dateiorganisation. Im Hauptmenü (rechts) sind vor allem Kommandos für die Texteingabe aufgeführt.

Hier sieht der Benutzer, welches Laufwerk und welche Datei aufgerufen wurden.

Bei Wahl von Hilfsstufe 3 zeigt das Hauptmenü diese vollständige Befehlsliste mit Erläuterungen.

Seiten-, Zeilen- und Spalten-Nummer geben die aktuelle Cursorposition an.



Diese Anzeige erinnert an die eingeschaltete Einfügungsfunktion.

Bei den IBM-kompatiblen legt WordStar Befehle auf Funktionstasten; die Zuordnung ist der untersten Bildschirmzeile zu entnehmen.

Dieser „Maßstab" gibt die Zeilenlänge und die Tabulatorstopps an – hier das Standardraster, das sich beliebig ändern läßt.

Hauptmenü oder zum Betriebssystem. Wenn Sie einen Befehl aufrufen wollen, der nicht im gerade angezeigten Menü steht, ist das jederzeit auch ohne Wechsel der Menüdarstellung möglich. Mit dem Befehl „CTRL KX" beispielsweise wird die gerade bearbeitete Datei gespeichert und anschließend das Betriebssystem aufgerufen.

Zum Abschluß noch ein Blick auf die Punktkommandos, die zwischen Textzeilen eingeschoben werden und mit einem Punkt beginnen – daher der Name. Sie betreffen die Formatsteuerung beim Ausdrucken und Details wie Seitenzahlen, obere und untere Randbreite usw. Der Befehl „.pl" mit nachfolgender Zahlenangabe legt beispielsweise die Anzahl der Zeilen pro Seite fest. Die Punktkommandos sind eigentlich nur Steuerzeichen für den Drucker; WordStar kümmert sich allerdings bis zum Druckbeginn nicht um die Steuerzeichen.

Mit seiner Leistungsfähigkeit hat WordStar fast ein Jahrzehnt in der Textverarbeitung den ersten Platz gehalten. Und seit dem „Abstieg" von CP/M aus den professionellen Höhen in den Bereich von Heimcomputern (wie den Schneider-Rechnern) übernimmt WordStar auch hier eine dominierende Rolle.

Popularitätseffekte

WordStar ist ein interessantes Beispiel für ein Softwarepaket, das Eigendynamik entfalten kann, wenn es erst einmal allgemeine Anerkennung gefunden hat. Nachdem WordStar sich als führendes Textverarbeitungssystem für CP/M-Rechner etabliert hatte, wurde es von vielen Computerherstellern als Bestandteil von Komplettpaketen übernommen, so daß der Benutzerkreis immer mehr wuchs.

Einen etwas anderen Verbreitungseffekt löste der Osborne 1 aus, bei dem nicht nur WordStar, sondern auch das Tabellenprogramm SuperCalc zum Lieferumfang gehörte – und beide liefen unter CP/M. Daher konnten auch beide auf die gleichen Dateien zugreifen. Dies Konzept fand soviel Anklang, daß andere Softwarehersteller sich ebenfalls mit dem Prinzip der ‚gemeinsamen' Dateien anfreundeten. Daraus entstanden schließlich integrierte Softwarepakete wie Lotus 1-2-3. Heute werden fast alle Micros mit Softwarepaketen inklusiv Textverarbeitung angeboten. Der Rahmen reicht dabei von aufwendigen und extrem teuren Systemen wie der ‚Perfect Software' von Thorn EMI bis zu Sparversionen wie ‚Tasword II'.

Senkrechtstarter

‚Artic Computing' wurde mit hundert Mark Anfangskapital gegründet. Inzwischen hat sich daraus ein Softwarehaus mit einem Jahresumsatz von über drei Millionen Mark entwickelt.

Richard Turner schrieb 1980 seine ersten Programme, „Battleship" und „Star Trek", für den ZX80. Er hatte sich für Strategiespiele anstelle der verbreiteten Arcadespiele entschieden: „Beim ZX80 wurde jedesmal der Bildschirm gelöscht, wenn sich irgendwas bewegte. Deshalb waren Denkspiele die einzige Möglichkeit – Arcadespiele kamen erst später für den Spectrum in Frage."

Der erste große Erfolg war das ZX-Schach, das Turner auf der ersten ZX-Messe im Sommer 1981 vorstellte. Der Jungunternehmer hatte dafür die letzten Kräfte mobilisiert: „Noch in der Nacht vorher kopierten wir wie wild mit sieben ZX81-Rechnern die Cassetten und stopften sie in Plastiktüten, zusammen mit den Anleitungen, die wir in der Schule kopiert hatten." Das ZX-Schach war sehr erfolgreich, und Turner behauptet, er habe auf der Messe über fünftausend Mark kassiert.

Wenig später wurde die Firma Artic Computing aus der Taufe gehoben. Die Aktivitäten kamen aber erstmal zum Stillstand, da Turner ein Ford-Stipendium annahm und begann, am Imperial College in London Elektrotechnik zu studieren. Dieses Studium hielt er ein Jahr durch, bis er eine kurzfristige Beurlaubung beantragte, um sich seiner Firma anzunehmen. Seine Professoren sahen ihn nie wieder.

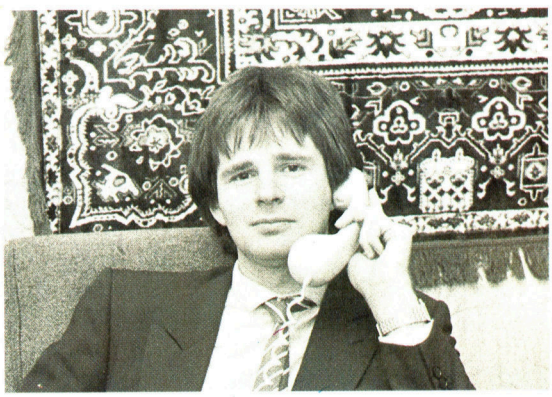
Beginn im Schlafzimmer

Firmenresidenz der Artic war ursprünglich Richard Turners Schlafzimmer im elterlichen Haus in Hull, aber als die Produktpalette auf 93 Softwaretitel angewachsen war, hielt der Chef angemessene Geschäftsräume doch für unumgänglich. So bezog das Unternehmen im Juni 1983 die Büros in Brandesburton/Humberside. Das Lieferprogramm wurde gestrafft und mehr Personal eingestellt.

Artic bereitet eine eigene Vertriebskette für Großbritannien vor; die ‚Artic Software Stationen' sollen dann nicht nur die Spiele von Artic, sondern auch Fremdprodukte verkaufen. Das erste Geschäft wurde im Juli 1984 in Acton/West London eröffnet und dient gleichzeitig als Londoner Zweigniederlassung. Auffällig ist, daß dieser Laden weder in einem ausgesprochenen Geschäftsviertel noch in der Nähe des West End-Einkaufszentrums liegt. Zu dieser Standortwahl sagt Jeff Raggett, Londoner Vertriebsleiter: „So ein Laden kostet in einer Hauptstraße pro Woche weit über tausend Mark, und hier natürlich viel weniger – sonst müßten wir einen Haufen Cassetten allein für die Miete umsetzen. Viele Leute halten eigene Läden sowieso für Blödsinn. Wir sehen aber jetzt, was sich an den Mann bringen läßt und kriegen mit, was die Kunden an den Spielen hauptsächlich reizt."

Ein anderer Verkaufsgag sind die Artic-Thekenständer, die bis zu 64 Cassetten aufnehmen können. Sie werden bei Zeitungshändlern aufgestellt, um den Kunden weite Wege zu den größeren Geschäften zu ersparen.

Hier ein paar der bekanntesten Spiele von Artic, darunter die Bestseller ‚Bear Bovver' und ‚World Cup', eine Spectrum-Version des beliebten Fußballspiels.



Der Gründer von ‚Artic Computing', Richard Turner, startete sein Unternehmen mit knapp 100 Mark Anfangskapital.

Artic möchte auch die Exportmärkte möglichst selbst in die Hand nehmen, wobei zunächst das europäische Festland angestrebt wird. Für die USA hat Artic eine wechselseitige Vertriebsvereinbarung mit den etablierten Softwarehäusern Softsync und International Publishing Corporation getroffen.

Bestseller von Artic waren „Bear Bovver", (mit über 40 000 verkauften Cassetten), „Galaxians" und „Gobbleman". Etwas neueren Datums ist das Spiel „World Cup", von dem binnen drei Wochen über fünftausend Kopien umgesetzt wurden.

Funktionen zur Stringbearbeitung

strcmp(s1,s2) – s1 und s2 sind Pointer auf char (strings). Zurückgeliefert wird ein Ganzzahlenwert. Bei s1<s2 ist er unter, bei s1=s2 gleich und bei s1>s2 über Null.
strncmp(s1,s2,n) – Wie strcmp, nur werden höchstens n Zeichen verglichen.
strlen(s) – s ist ein Pointer auf char. Die Funktion liefert die Stringlänge als Ganzzahl.
index(s,c) – s ist ein Pointer auf char und c ist char. Liefert Pointer auf erstes Vorkommen von c in s; oder NULL, falls es c in s nicht gibt.
rindex(s,c) – ähnlich wie index, nur wird die Suche vom rechtsaußen stehenden Zeichen aus durchgeführt.
strcat(s1,s2) – Hängt eine Kopie von s2 an das Ende von s1 und liefert s1. Es wird vorausgesetzt, daß Länge von s1 für alle Zeichen ausreicht.
strncat(s1,s2) – Hängt maximal n Zeichen an das Ende von S1 und liefert s1.
strcpy(s1,s2) – Kopiert Inhalt von s2 bis zu dem '/' in s1. Vorausgesetzt wird, daß Länge von s1 ausreicht. Der alte Inhalt von s1 wird gelöscht. Zurück kommt der Wert von s1.
strncpy(s1,s2,n) – Kopiert maximal n Zeichen von s2 in s1 und hängt ein ‚/' an, außer n ist größer als s2.

Wegweiser

Mit dem Variablentyp „Pointer" lassen sich in der Programmiersprache C absolute Speicheradressen ansprechen. Wir sehen uns Deklaration und Einsatz der Pointer genauer an und untersuchen dann Methoden der Stringverarbeitung.

Da C als Ersatz für Assembler entwickelt wurde, bietet diese Programmiersprache Möglichkeiten, die andere Hochsprachen nicht haben. So kann sich C beispielsweise direkt auf Maschinenadressen beziehen. Zwar kennt auch PASCAL den Variablentyp „Pointer", doch ist der Bezug zu Maschinenadressen dort nicht genau definiert. In C enthalten die Pointertypen direkte Maschinenadressen, die auf einer Multitasking-Maschine jedoch kaum praktischen Nutzen haben.

Pointervariablen können aber Hardware direkt steuern. Jede Variable, egal welchen Typs, besitzt eine Adresse, die einem Pointer zugeordnet werden kann. Speziell bei Strings ist es oft praktischer, mit Pointern auf einzelne Elemente zuzugreifen, statt sie mit Arraysubscripts anzusprechen.

Pointervariablen werden mit dem Zeichen * deklariert.

```
int *p;
```

deklariert die Pointervariable p. Beachten Sie, daß die mit int deklarierten Pointer nur auf einen bestimmten Variablentyp (den „Basistyp") zeigen können. Wenn Sie einen Pointer auf den Variablentyp char setzen wollen, müssen Sie daher

```
char *p;
```

eingeben etc. Da der * hier nur die Bedeutung „einen Pointer anlegen" hat, zeigt der Pointer noch auf keine Adresse.

Mit dem Zeichen & werden Pointer mit bestimmten Werten initialisiert. Steht & vor einer Variablen, liefert sie die Adresse der Variablen und nicht deren Wert.

```
P=&i;
```

initialisiert p als Pointer auf i. Doch auch * kann den Wert anzeigen, auf den eine Pointervariable zeigt.

```
i=*p;
```

speichert den Wert, auf den p zeigt in i,

```
i=p;
```

dagegen die Adresse, auf die p zeigt. Für die Bearbeitung absoluter Adressen brauchen Sie jedoch eine „cast" (Formatierung), um Fehlermeldungen des Compilers zu vermeiden:

```
p=(int*) 1000;
```

Pointervariablen lassen sich auch mit den Operatoren für In- und Dekrementierung (++ und – –) einsetzen. Sie berücksichtigen dabei automatisch den Typ der Variablen, auf die der Pointer zeigt. Bei einem System mit Ganzzahlen im Vier-Byte-Format zeigt p daher auf das erste Element eines Ganzzahlenarrays, p++ inkrementiert die Adresse in p um Vier und zeigt so auf das nächste Element. Das gleiche würde passieren, wenn wir p=p+1 verwenden: Hier wird bei der Inkrementierung die Größe des angesprochenen Elementes berücksichtigt und nicht nur Eins addiert. Damit wird klar, daß Pointer nicht mit Ganzzahlen identisch sind, selbst wenn sie Ganzzahlen enthalten.

Pointer können bei Funktionsaufrufen als Parameter übergeben werden. Dies schafft die Möglichkeit, Werte oder Bezug (reference) anzusprechen – das heißt, der Wert des Pointers wird innerhalb der Funktion zwar an eine lokale Variable übergeben, doch zeigt der Poin-

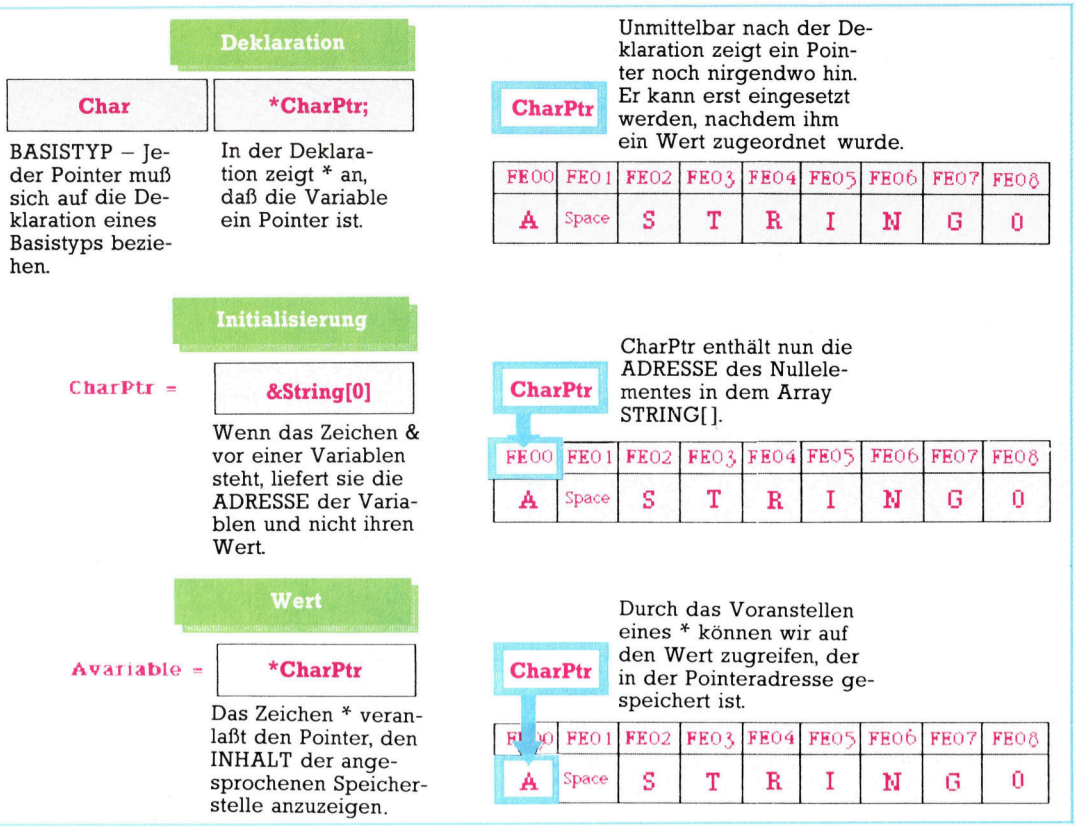
Der Bubblesort in C

```
bubble(a,n)
int a[],n,putinorder();
/* a ist ein Array mit n Ganzzahlen, die sortiert
   werden sollen */
{
int i,j;
   for (i=o;i<n—1;++i)
      for (j=n—1;i<j;— —j)
         putinorder(&a[j—1],&a[j]);
/* Die Adressen der Arrayelemente als Para-
   meter uebergeben */
}
putinorder(x,y)
int *x,*y,swap();
/* x und y sind Ganzzahlenpointer */
{
   if(*x>*y)
      swap(x,y);
}
swap(x,y)
int *x,*y;
{
   int temp;
   temp = *p;
   *p=*q;
   *q=temp;
/* beachten Sie, daß die Adressen der beiden
   Ganzzahlen zwar als Wert uebergeben wer-
   den, die Funktion sich darueber aber auf die
   Variablen beziehen kann, die sonst ausser-
   halb ihres Bereiches liegen wuerden */
{
```

Zahlenpointer
Unser Programm zeigt, welch enge Beziehung in C zwischen Pointern und Arrays bestehen. Der traditionelle „Bubble Sort Algorithmus" arbeitet mit einem Ganzzahlenarray. C-Pointer sind für den Zugriff auf Arrayelemente oft besser geeignet als Subscripts.

Auf den Punkt

Das Bild zeigt drei Stadien des Pointers CharPtr beim Ansprechen unterschiedlicher Elemente des Stringarrays STRING[]. Beachten Sie, daß CharPtr nach seiner Initialisierung eine Adresse zurückliefert, *CharPtr jedoch den in dieser Adresse gespeicherten Wert. Pointer lassen sich in- und dekrementieren und können so auf verschiedene Elemente des Arrays zeigen. Der Pointerwert wird dabei auf den in der Deklaration angegebenen Basisdatentyp eingestellt. So veranlaßt der Basistyp char Schritte im Ein-Byte-Rhythmus, während int auf den meisten Systemen Zwei-Byte-Schritte auslöst.



ter immer noch auf das gleiche Datenelement, wie in dem aufrufenden Programm. Änderungen des gespeicherten Wertes bleiben auch nach dem Rücksprung erhalten.

Zwischen Pointern und Arrays besteht eine enge Beziehung. Bei der Deklaration eines Arrays ist der Arrayname (ohne Subscript) ein Pointer auf das erste Arrayelement. Bei int a[100], *p ist daher p=a; das gleiche wie p=&a[0]. Der Zugriff auf Elemente eines Arrays oder Subarrays läßt sich mit Pointern oft schneller ausführen als mit Subscripts.

Strings sind eindimensionale Arrays von char. Da sie die Grundlage vieler Anwendungen bilden, wurde C mit Spezialmöglichkeiten und Standardfunktionen ausgerüstet, die die Stringbearbeitung sehr erleichtern. So steht in C hinter einem Array von char (mit einem String) immer das Zeichen /0 (ASCII-Code für Null). Da dies an jeder Arrayposition auftreten kann, entsteht selbst bei den auf feste Längen programmierten Arrays der Eindruck dynamischer Stringverwaltung. Bedenken Sie jedoch, daß auch das Zeichen /0 eine Stelle des Arrays belegt. Die Arraylänge muß daher um Eins höher sein als die maximal vorgesehene Zeichenzahl.

In doppelte Anführungszeichen eingeschlossene Stringkonstanten können jedem Array von char mit ausreichender Länge zugeordnet werden. /0 wird automatisch ergänzt. Die Variablenzuordnung geschieht über das Array selbst oder mit einem Pointer auf char.

```
char *s;
*s="abc";
```

veranlaßt, daß die vier Zeichen a, b, c und /0 in vier aufeinanderfolgenden Speicherstellen abgelegt werden, die bei der in s gespeicherten Adresse anfangen (S zeigt auf das Zeichen a). Nach ++s zeigt s auf b etc.

Pointer sind normale Variablen und belegen ebenfalls eine Speicherstelle mit Adresse. Es ist daher möglich, Pointer auf Pointer zeigen zu lassen. Die Programmierung kann sich dadurch zwar schwieriger gestalten, doch gibt es einige Bereiche, in denen Pointerarrays praktisch sind.

Es gibt zwei Spezialparameter, die an die Hauptfunktion übergeben werden können und Zugang zu den Befehlsparametern geben: argc liefert die Zahl der in der Befehlszeile enthaltenen Argumente (durch Leerzeichen voneinander getrennt) und argv ein Array von Pointern auf die Strings mit argc-Elementen. Jedes Element in argv zeigt auf das erste Zeichen dieses Befehlsstrings. Das folgende Programmbeispiel verdeutlicht diesen Vorgang es stellt in diesem Fall die Befehlsargumente zeilenweise dar:

```
main(argc,argv)
int argc;
char *argv[];
/* Deklaration erzeugt Pointerarray */
{
   int i;
   for (i=1;i<argc;i++)
      printf("%s/n",argv[i]);
/* Beachten Sie, daß das % Format einen
Pointer auf einen String erfordert */
}
```

Floppy-Konverter für

Commodore-Cassetten-Programme können auch mit einer Diskettenstation benutzt werden. Dazu brauchen Sie nur dieses Konverterprogramm in Maschinensprache.

Bis jetzt wurde bei den meisten hier veröffentlichten Programmen davon ausgegangen, daß Sie zum Laden und Speichern von Daten einen Cassettenrecorder benutzen. Inzwischen verfügen jedoch die meisten Anwender über eine Diskettenstation.

Wenn Sie eine Floppy besitzen, können Sie Ihre Programme natürlich von Hand umschreiben. Aber warum sollte der Rechner das nicht für Sie tun? Das folgende Maschinenprogramm modifiziert die Cassettenbefehle der Commodore 64- und VC 20-Programme automatisch.

Für den Acorn B ist ein solches Programm nicht nötig, weil er eine eventuell vorhandene Floppy automatisch anspricht. Electron und ZX 81 haben keine Floppys. Beim Dragon gibt es drei mögliche Konfigurationen,die verschieden behandelt werden müßten. Ähnliches gilt für den Spectrum und Schneider-Computer.

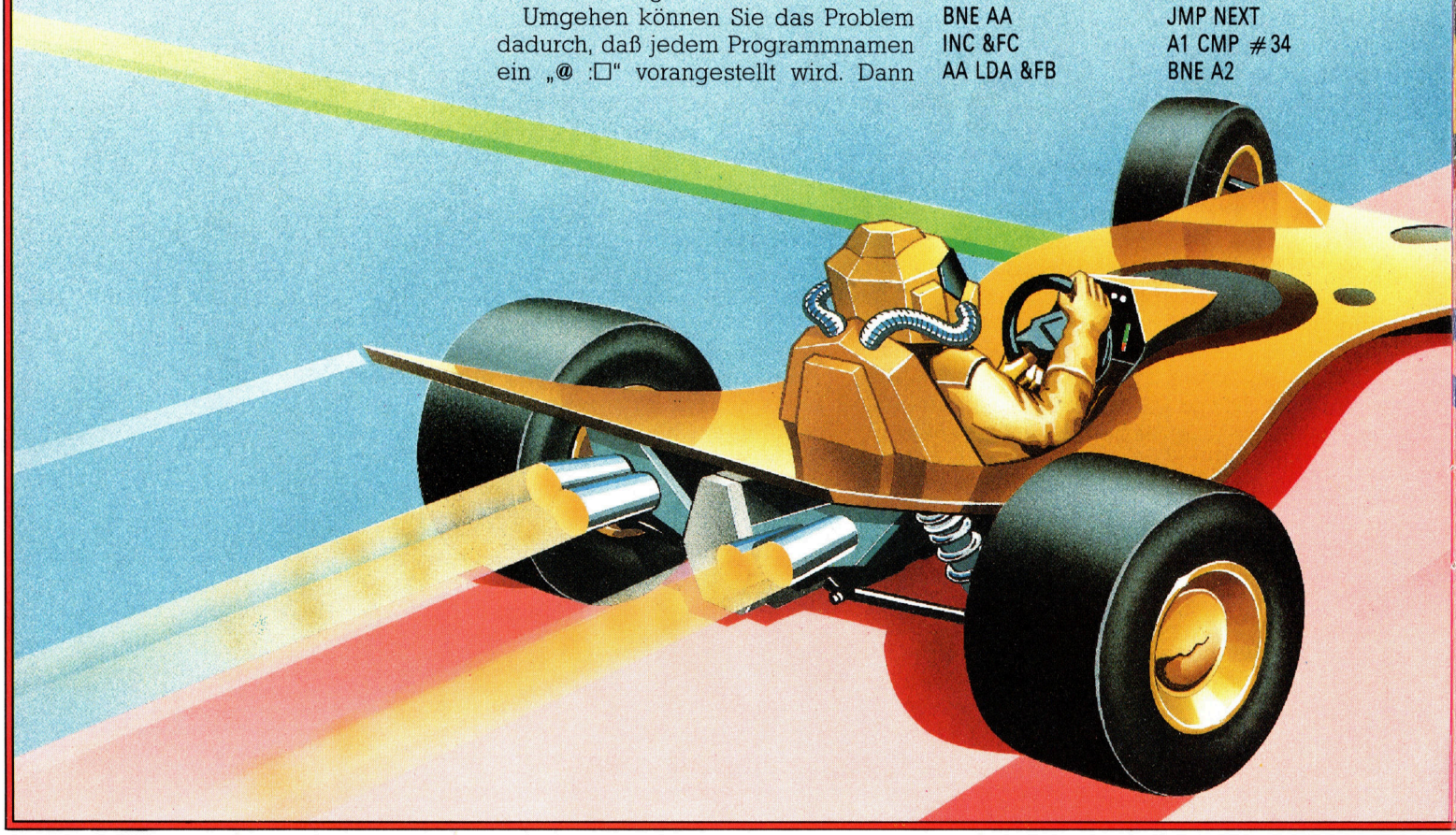
Commodore

Das folgende Programm wandelt alle Commodore-Cassettenprogramme so um, daß sie statt des Recorders die Floppy ansprechen. Dazu muß zuerst einmal jedem Cassettenbefehl ein „,8" angefügt werden. Aber das Programm geht noch weiter, denn zwischen dem Schreibvorgang auf Floppy oder Cassette gibt es Unterschiede: Bei einer Cassette kann man ein Programm einfach überschreiben, oder unter gleichem Namen beliebig häufig abspeichern. Bei der Floppy ist das anders: Jeder Versuch dieser Art führt zu einer Fehlermeldung.

Umgehen können Sie das Problem dadurch, daß jedem Programmnamen ein „@ :□" vorangestellt wird. Dann wird die alte Version überschrieben. Unser Konversionsprogramm stellt jedem Programmnamen automatisch das „@ :□" voran.

Manchmal sind Programmnamen auch als Strings gespeichert. Die Fähigkeit zum Suchen und Modifizieren dieser Strings hätte die Länge des Maschinenprogramms verdoppelt und wurde daher nicht vorgesehen. Falls Ihr Programm nach diesem Verfahren arbeitet, müssen Sie selbst in die Tasten greifen und jedem Namen ein „@ :□" voranstellen.

```
ORG□49152
LDA #Ø
STA &Ø33E
CLC
LDA &2B
ADC #3
STA &FB
LDA &2C
ADC #Ø
STA &FC
NEXT LDY #Ø
INC &FB
BNE AA
INC &FC
AA LDA &FB
INY
LDA (&FB),Y
BNE WG
JMP STOP
WG LDA &FB
ADC #3
STA &FB
LDA &FC
ADC =Ø
STA &FC
A4 LDX #Ø
STX &Ø33E
JMP NEXT
A1 CMP #34
BNE A2
```



den Commodore

```
CMP &2D
BNE BB
LDA &FC
CMP &2E
BNE BB
JMP STOP
BB LDA (&FB),Y
CMP #Ø
BNE A1
INY
LDA (&FB),Y
BNE WG
CMP #147
BEQ LOOP
CMP #148
BEQ LOOP
CMP #149
BEQ LOOP
CMP #159
BEQ LOOP
JMP NEXT
LOOP INY
```

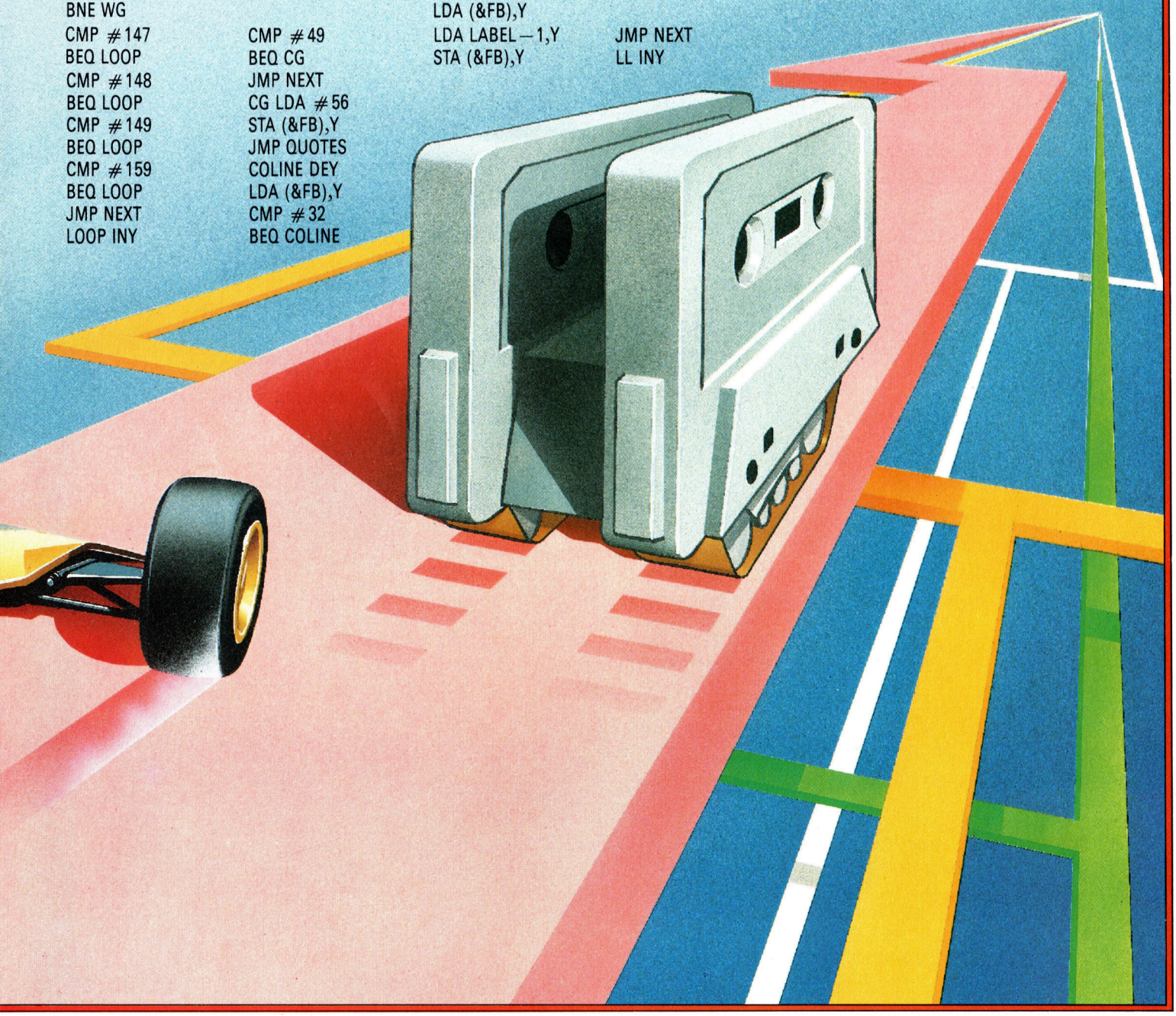
```
LDX &Ø33E
CPX #Ø
BNE A4
LDX #1
STX 3Ø33E
JMP NEXT
A2 LDX &Ø33E
CPX #1
BEQ NEXT


CMP #49
BEQ CG
JMP NEXT
CG LDA #56
STA (&FB),Y
JMP QUOTES
COLINE DEY
LDA (&FB),Y
CMP #32
BEQ COLINE
```

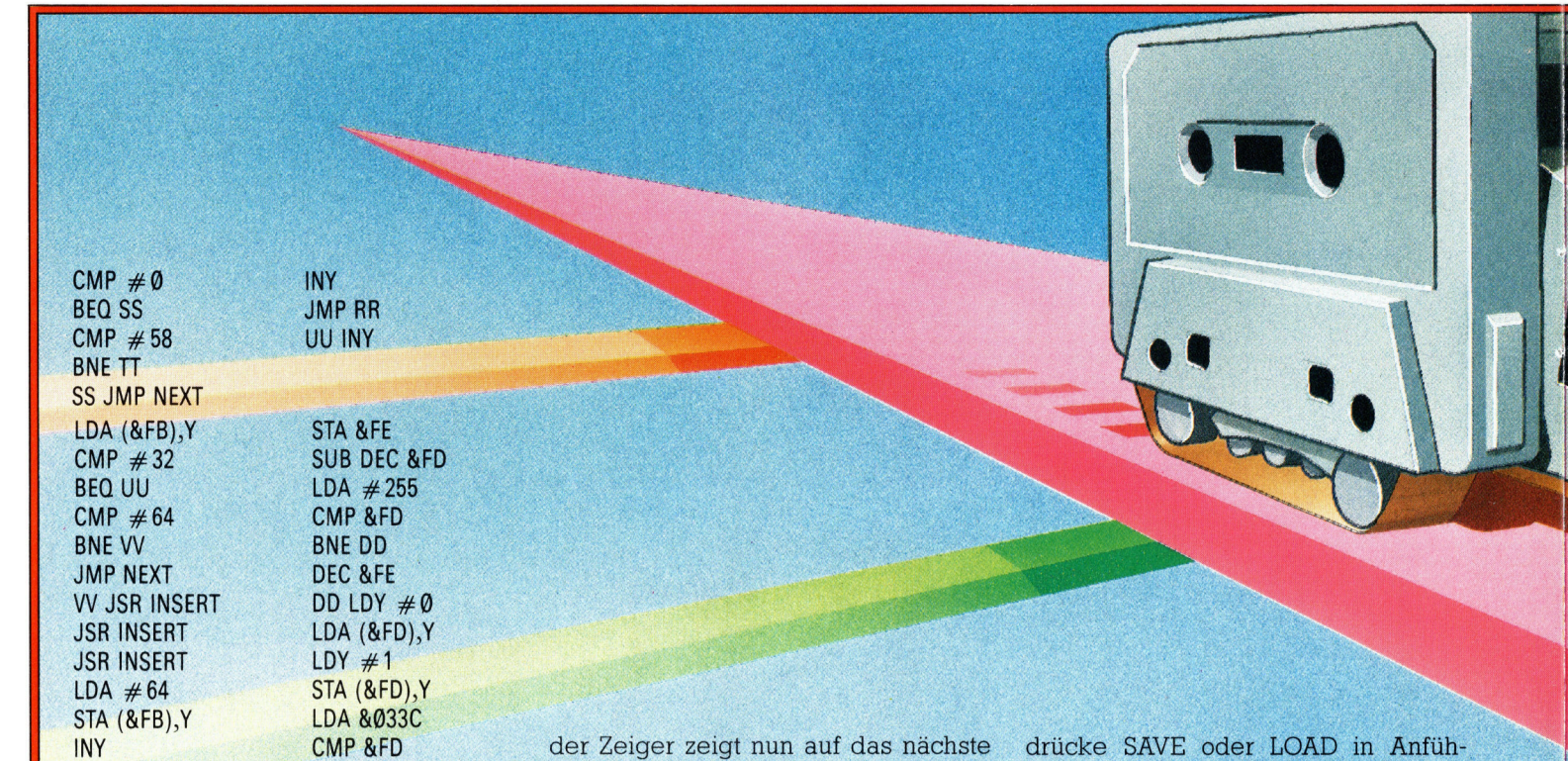
```
LDA (&FB),Y
CMP #32
BEQ LOOP
CMP #Ø
BEQ COLINE
CMP #58
BEQ COLINE
CMP #44
BEQ COMMA
JMP LOOP
COMMA INY
LDA (&FB),Y
LDA LABEL-1,Y
STA (&FB),Y
```

```
CPY #Ø
BNE KK
INY
CC JSR INSERT


JMP NEXT
LL INY
```

```
INY
CPY #8
BNE CC
JMP NEXT
KK LDA (&FB),Y
CMP #34
BEQ LL
CMP #36
BEQ LL
QUOTES LDY #1
RR LDA (&FB),Y
```

```
JSR INSERT
JSR INSERT
LDA #44
STA (&FB),Y
INY
LDA #56
STA (&FB),Y


TT CMP #34
BEQ UU
```





```
CMP #Ø          INY
BEQ SS          JMP RR
CMP #58         UU INY
BNE TT
SS JMP NEXT
LDA (&FB),Y     STA &FE
CMP #32         SUB DEC &FD
BEQ UU          LDA #255
CMP #64         CMP &FD
BNE VV          BNE DD
JMP NEXT        DEC &FE
VV JSR INSERT   DD LDY #Ø
JSR INSERT      LDA (&FD),Y
JSR INSERT      LDY #1
LDA #64         STA (&FD),Y
STA (&FB),Y     LDA &Ø33C
INY             CMP &FD
LDA #58         BNE SUB
STA (&FB),Y     LDA &Ø33D
INY             CMP &FE
LDA #32         BNE SUB
STA (&FB),Y     INC &2D
JMP NEXT        BNE EE
INSERT STY &FF  INC &2E
CLC             EE LDY &FF
LDA &FC         RTS
STA &Ø33D       STOP JSR &A533
PP TYA          RTS
ADC &FB         LABEL BYT &22
BCC QQ          BYT &4Ø
INC &Ø33D       BYT &3A
QQ STA &Ø33C    BYT &2Ø
LDA &2D         BYT &22
STA &FD         BYT &2C
LDA &2E         BYT &38
```

Funktionsbeschreibung

Durch das Programm wird zuerst eine 0 in den Akkumulator geladen und bei Adresse 033E gespeichert. Damit wird ein Byte gelöscht, das später als Flag gebraucht wird.

Die Zeiger für den Anfang des BASIC stehen bei 43 und 44 (2B/2C hex). Die ersten vier Befehle speichern diese Zeiger in FB/FC, einem freien Bereich der Speicherseite Null (Zero-Page), wo sie Veränderungen zugänglich sind.

Danach wird das Y-Register mit 0 geladen. Der Befehl INC &FB erhöht den Inhalt von FB um Eins, das heißt, der Zeiger zeigt nun auf das nächste Byte des BASIC-Programms. Ist das Ergebnis dieser Operation Null, dann bleibt der folgende Sprungbefehl inaktiv, und auch das höherwertige Byte des Zeigers wird inkrementiert.

Als nächstes lädt der Akkumulator das niederwertige Byte des Zeigers, das mit dem niederwertigen Byte in 45 und 46 (2D/2E hex) verglichen wird. Bei Übereinstimmung werden die höherwertigen Bytes verglichen. Stimmen auch sie überein, ist offensichtlich das Programmende erreicht. Dann erfolgt ein Sprung auf STOP am Ende der Routine.

Solange diese Bedingung nicht erfüllt ist, wird durch den Befehl LDA (&FB),Y das Byte geladen und auf das FB/FC zeigen. Da Y immer noch auf Null gesetzt ist, wäre der indizierte Befehl eigentlich unnötig. Wir benötigen in diesem Fall jedoch eine indirekte Adressierung, und jede indirekte Adressierung muß indiziert sein.

Bei Commodore können zwei Grafik-Symbole, die von der Tastatur aus erreichbar sind, nämlich CLEAR SCREEN und INSERT, mit den Recorder-Befehlen LOAD und SAVE verwechselt werden. Die ASCII-Codes für die Grafikzeichen sind nämlich identisch mit den Token der Befehle. Sie lassen sich dadurch auseinanderhalten, daß die Grafikzeichen immer in Anführungszeichen stehen, Befehle dagegen nie. Selbst wenn die Ausdrücke SAVE oder LOAD in Anführungszeichen auftauchen, kann es keine Verwirrung geben, denn dann bleiben sie als Text in ASCII stehen und werden nicht in Befehls-Token verwandelt.

Die Speicheradresse 033E

Unsere Routine muß daher prüfen, ob das Byte, das wir untersuchen, in Anführungszeichen steht oder nicht. Das geschieht, indem man ein Flag setzt, in diesem Fall die Speicheradresse 033E, die anfangs mit Null definiert wurde. Das Flag sollte gesetzt sein, bevor das Programm auf das erste Anführungszeichen in einer Zeile stößt. Kommt das zweite Anführungszeichen, wird es wieder zurückgesetzt. In den späteren Programmteilen kann dieses Flag bei Bedarf geprüft werden. Die Routine, die nach Anführungszeichen sucht, beginnt mit CMP #0. Dabei kennzeichnet 0 das Ende einer Zeile. Man berücksichtigt dadurch den Fall, daß nur ein Anführungszeichen innerhalb einer Zeile steht. Wird eine 0 gefunden, dann wird BNE AB nicht ausgeführt und 0 in das X-Register sowie nach 033E geladen, wodurch das Flag zurückgesetzt ist. JMP NEXT prüft dann das nächste Byte des BASIC-Programms.

Wird keine 0 gefunden, dann verzweigt der Prozessor nach AB und vergleicht das Byte mit 34, dem ASCII-

computer kurs

Computer Welt

Heft		Seite
61	Die gedruckte Seite	1691
	Ein Paukenschlag	1703
62	Zurück zu den Anfängen?	1709
	Computerbilder	1722
	Auf der Rennbahn	1734
63	Sieg oder Platz	1747
	Kostenprobleme	1759
64	Wetten, daß . . .?	1765
	Form der Zukunft	1785
65	Höret und staunet	1793
	Die Erfolgsformel	1818
66	Herzspezialisten	1833
	Computer-Casino	1842
67	Modellentwurf	1849
68	Chip-Tourismus	1877
69	Schreiben und Lesen	1905
70	Dynamische RAMs	1952
71	Der Torhüter	1961
72	Senkrechtstarter	1997
	Gespeicherter Schirm	2015

Basic

Heft		Seite
61	Mann über Bord	1686
	Platz für Manöver	1704
62	Austausch von Variablennamen	1712
	Alleine auf hoher, weiter See	1728
63	Globaler Austausch	1740
	Große Sorgen	1750
64	Piraten voraus!	1774
65	Zeit für Go	1796
	Säbelrasseln	1808
66	Go in Prozeduren	1821
	Die Neue Welt	1840
67	Fairer Tausch	1852
	Go total	1869
68	Auf die Plätze . . .!	1879
	Handelsbilanz	1896
69	Die Neue Welt I	1926
70	Die Neue Welt II	1944
	Stein auf Stein	1954
71	Die Neue Welt III	1976
	Steinzeit	1986
72	Wir machen einen Zug	2007

Software

Heft		Seite
61	CP/M-Befehle	1684
	Baumschule	1698
62	Namenserweiterung	1720
	Schatzsuche	1727
	Was tun mit Essen und Trinken?	1731
63	Systemkopie	1742
63	Brot und Spiele	1758
64	Objekt-Listing	1768
	Vielversprechend	1780
65	Programmierhilfen	1798
	Aus- und Umwege	1810
	Gesicherte Zwerge	1820
66	Vom Plan zur Tat	1824
	Katzenjammer	1838
67	Gestatten, Dr. LOGO	1854
	Von Baum zu Baum	1872
69	Dog and Bucket	1910
	Fenster zur Welt	1920
70	Blitter und WIMPS	1942
71	Grafikrahmen	1972
	Wilder Wort-Wirbel	1982
72	Menü-Vorschläge	1994
	Arbeitsplatz am offenen Fenster	2010

Hardware

Heft		Seite
61	Einkaufsbummel	1681
63	Das Top-Modell	1737
65	Der Texter	1805
67	Fast MSX-Standard	1861
69	Fürs Heimkino	1917
71	In Ehren ergraut	1974

Tips für die Praxis

Heft		Seite
61	Es ist soweit!	1694
62	Auf Armeslänge . . .	1724
63	In Bewegung	1755
64	Sinclairs neue Kraft	1772
	Kurvenberechnung	1788
65	Abtaster	1815
66	Endmontage	1834
67	Aus Arm mach RAM	1864
	Gummiband-Betrieb	1875
68	MIDI-Rock	1898
69	Klänge vom Chip	1907
	Soundcheck	1922
70	Auf Bachs Spuren	1958
71	High Fidelity	1969
	Die Vollendete	1984
72	Digitalmessung	1992

Bits und Bytes

Heft		Seite
61	Der FX-Effekt	1688
	Der erste Block	1706
62	Dateizugriff	1714
63	Byte für Byte	1753
	Zwei Versionen	1762
64	Kooperation	1770
	Wir unterbrechen dieses Programm	1790
65	Ereignisreich	1803
	Letzter Aufruf	1812
66	Der Prozessor 6510	1829
66	Tor zur Außenwelt	1845
67	Kontakt zum CIA	1859
	Farbenfreude	1867
68	Geborgte Zeit	1883
	Bilderrahmen	1901
69	Grafik mit Dreh	1929
70	Die vielen Talente des Spectrum	1938
71	Kanalarbeiter	1966
	Stromlinien	1979
72	Bandmaß	1989
	Hinter der Maske	2012

Peripherie

Heft		Seite
64	Reden ist Silber	1777
66	Der Jet-Set	1836
70	Starkes Echo	1933

Forth

Heft		Seite
61	Wortspiele	1696

Fortran

Heft		Seite
62	Frühe Formeln	1717
63	Zahlenakrobatik	1744
64	Aus alt mach neu	1782

Cobol

Heft		Seite
65	COBOL für Micros	1800
66	Befehlsabteilung	1826
67	Geschäftssprache	1856

C

Heft		Seite
68	C wie Computer	1885
69	Das hohe C	1914
70	In Funktion	1935
71	Klassenbewußt	1964
72	Wegweiser	1998

Programmier-Service

Heft		Seite
68	Computerhilfe in Geldfragen	1887
70	Auf Nummer Sicher gehen	1946
72	Datentransfer vom Band zur Platte	2000

Index Band 6

A

Seite

Abenteuerspiele
  Dog and Bucket:
    Baumstruktur für Requisiten 1731—3
    Entscheidungsbaum 1698—9
    Interaktionsbaum 1810—11
    Listing 1910—13
    Objekt-Listing 1768—9
    Programmgenerator 1798
    Sortierroutinen 1872—4
    Strukturplan 1838—9
ACIA (Asynchronous Communications Interface Adaptor) 1907, 1923, 1925
ACLS (Analogue Concept Learning System) 1849
Acorn B 1681, 1682
  A/D-Wandler 1815
  BASIC I und BASIC II 1762
  BASIC-ROM 1762—4, 1770—1
  Betriebssystem 1688—90
  BIOS-Calls 1812—14
  Econet VS 61
  Ereignissystem 1803—4
  Fehlerbehandlung 1763—4
  Interrupts 1790—2
  Namal Type & Talk 1777
  OSBYTE-Aufrufe 1688—90
  OSFILE-Aufrufe 1714—16
  OSFIND-Routinen 1753—4
  OSWORD-Routinen 1706—8
  Robotarm-Steuersoftware 1724—6
  Selbstbau-MIDI 1898—1900, 1907—9
  Tracer-Programm 1864—6, 1875—6
  Variablen-Austausch 1704, 1712—13
Acorn Electron 1682
A/D-Wandler 1992—3
  Acorn B 1815
Amsoft Speech Synthesizer 1778, 1779
Amstrad 1703
  siehe auch Schneider
Analog/Digital-Wandler, siehe A/D-Wandler
Apple II
  Sampling-System 1794
Apple Lisa 1942
Apple Macintosh 1692, VS 71
  PageMaker 1692
  WIMP-Betriebssystem 1920
Array
  in COBOL 1856
  in FORTRAN 1744—5
Artic Computing 1997
ASM 1685
Atari 520ST 1693
  600/800XL 1681, 1683
Ausbildung, Computer in der CAL (Computer-Assisted Learning) 1709—11
  Computerunterricht 1700
  Form der Zukunft 1785—7
  Kostenprobleme 1759—61
Ausgang, serieller des C64 1845

B

Seite

Bank Switching (Bankauswahlverfahren) 1829
BASIC
  BASIC I und BASIC II mit Acorn B 1762, 1764
  Handelssimulationsspiel 1686—7, 1728—30, 1750—2, 1774—6, 1808—9, 1840—1, 1852—3, 1896—7
    Acorn B-Listing 1944—5
    Schneider-Listing 1976—8
    Spectrum-Listing 1926—8
Go-Programm 1796—7, 1821—3, 1869—71, 1879—82, 1954—7, 1986—8, 2067—9
  -Interpreter 1798
  -Pointer des C64 1830
  Programm-Speicherung 1704—5
  Variablen-Austausch-Hilfsprogramm 1712—13
Bauanleitungen
  digitales Vielfach-Meßgerät 1992—3
  MIDI-Interface 1898—1900, 1907—9, 1922—5, 1958—60
  Robot-Arm 1694—5
    Steuer-Software 1724—6, 1755—7, 1772—3, 1788—9
  Tracer 1815—17, 1834—5, 1864—6, 1875—6
BCPL 1885
BDOS (Basic Disk Operating System) 1742, 1743
Befehlsinterpreter (CCP) 1742
Befehlszähler (Program Counter) 1878
Betriebssystem (Operating System) VS 71
  Acorn B 1688—9, 1762—4, 1770—1
  Commodore 64 1829—32
  Spectrum 1938—41
  Spectrum-Cassettensystem 1989—91
  Spectrum-Interrupts 2012—14
  WIMP-System 1920—1
BIOS (Basic-Eingabe/Ausgabe-System) 1742, 1743
  -Calls 1812—13
Bit image manipulator (Blitter) 1737, 1738, 1942—3
Bit-Mapping 2015
Blitter (Bit Image Manipulator) 1737, 1738, 1942—3
Boolesche Algebra
  Logiksymbole VS 62
Bootstrap 1947
  Bootstrap Loader (Urlader) 1684
Briefkasten (Mail Box) VS 65
Buchse, serielle des C 64 1847
Buffer
  Tastatur- 1798
Bus, serieller des C 64 1845

C

Seite

C 1885—6
  Funktionen 1935—7
  Hisoft-C 1885
  logische Operatoren 1914—16
  Pointer 1998—9
  Variablenklassen 1964—5
CAI (Computer-Aided Instruction) 1709
CAL (Computer-Assisted Learning) 1709—11
Cassette
  Konverterprogramm für Commodore 2000—6
  Spectrum-Cassetten 1989—91
CBL (Computer-Based Learning) 1709
CCP (Console Command Processor) 1742, 1743
CD-ROM-Disc 1787
Chip
  CIA-, des C 64 1859—60
  PIO- 1961—3
  Sound- 1793
  Überblick 1877—8
  Video-Interface- (VIC II) 1867—8
CIA (Complex Interface Adaptor) 1846, 1962
  -Chip des C 64 1859—60
COBOL
  für Micros 1800—2
  Geschäftssprache 1856—8
  Verarbeitungsteil 1826—8
COMAL (Common Algorithmic Language) 1700, 1798
Commodore
  Cassetten-Konverterprogramm 2000—6
  Musiksynthese 1793
Commodore 16 1681, 1683
Commodore 64 1681, 1683
  BASIC-Pointer 1830
  Betriebssystem 1829—32
  CIA-Chip 1859—60
  E/A-System 1845—8, 1859—60
  Fließkommaroutinen 1901—4, 1929—32
  Memory Map 1829
  OS-RAM 1829
  Parawedge-Programm 1859—60
  Peripherieschnittstellen 1845—8
  RAM-Vektoren 1830
  Robotarm-Steuerprogramm 1755—7
  Sampling-System 1794
  Selbstbau-MIDI-Interface 1898—1900, 1907—9
  Splitscreen-Programm 1883—4
  Variablen-Austausch 1704, 1712—13
  VIC II 1867—8, 1883—4
Commodore Amiga 1737—9, 1943
Commodore PET 1974—5
Commodore Plus/4 1681, 1683
Compact Disk, siehe Laserplatte
Control Character (Steuerzeichen) 1720, 1721
Counter (Taktzähler-Register) 1878
CP/M (Control Program for Micro-Processors)
  Befehle 1684—5
  Concurrent 1781
  CP/NET 1781
  Dateien 1720—1, 1743
  Disketten kopieren 1742—3
  Hauptmodule 1780—1
CPU (Zentraleinheit)
  Architektur 1877
  Schreiben und Lesen 1905—6

D

Seite

D/A-Wandler 1992—3
Datei
  Namenserweiterung 1720—1
  OSFILE-System mit Acorn B 1714—16
Daten
  Stack-Datenstruktur 1879
  -Transfer vom Band zur Platte 2000—6
Datenbus 1877
Datenströme
  Spectrum 1979—81
Decision Support System (Entscheidungshilfesystem) VS 65
Decoder 1878
  3-Bit- 1906
Demodulation VS 67
Digital Research
  CP/M-Version 1781
  GEM 1942—3
  GSX 1972
Digitalisierarm, siehe Tracer
Digitiser (Digitalisiergerät) 1691
Diskette VS 64
  Datentransfer vom Band 2000—6
  Kopieren mit CP/M 1742—3
DMA (Direct Memory Access) 1883
Dongle 1820
Drucker
  Tintenstrahl- 1836—7
  Zeilen- VS 61
Druckindustrie
  Die gedruckte Seite 1691—3
  PageMaker 1692

E

Seite

E/A (Ein- und Ausgabe)
  E/A-System des C 64 1845—8
  Kanalsteuerung bei Spectrum 1966—8
  OS-Routinen 1845
  PIO-Chip 1961—3
Echo Synthesizer 1933—4
Einer-Komplement (One's Complement) VS 71
Einloggen (Logging On) VS 62
Entscheidungsbaum 1698—9
  Entscheidungshilfesystem (Decision Support System) VS 65
Epson SQ-2000 Tintenstrahldrucker 1836—7
Ereignissystem 1707, 1803—4
Erweiterung
  Namens-, mit CP/M-Dateien 1720—1
  -Schnittstelle 1847
  -Steckleiste des C 64 1845
Expertensysteme
  Computer-Casino 1842—4
  Expert-Ease 1849
  Generator 1849—51

F

Seite

FCB (File Control Block) 1743
Fehlerbeseitigung 1798
Fenster
  WIMP 1920—1, 1942—3, 1972—3
Fließkomma-Arithmetik
  Routinen für den C 64 1901—4, 1929—32
Flip-Flop
  J-K- 1905
Floppy
  Cassetten-Konverterprogramm für

Commodore 2000—6
FORTH 1798
String-Verarbeitung 1696—7
FORTRAN
Arrays 1744—5
Codeknacker-Programm 1783
Compiler 1784
Mängel beseitigen 1782—4
Programmaufbau 1717—19
Subroutinen 1746
Zahlenverarbeitung 1744—6
FX-Aufrufe mit Acorn B 1678—80

G

Seite

Gatter
Logik- VS 62, 1877
GEM (Graphics Environment Manager) 1942—3, 1973, 2010—11
Geordnetes Zahlenpaar (Ordered Pair) VS 72
GKS (Grafisches Kern System) 1972—3
Glückspiele
Computer-Casino 1842—4
Expertensysteme 1843—4
Fußballergebnisse 1818—19
Pferderennsport 1734—6
Wahrscheinlichkeitsrechnung 1765—7, 1818—19
Grafik
Commodore Amiga 1738
dreidimensionale, mit dem C 64 1901—4, 1929—32
Gummiband-Modus 1875—6
Illusionen 1722—3
Splitscreen-Programm 1883—4
VIC-II 1867—8, 1883—4
Großrechner (Mainframe) VS 65
GSX (Grafics System Extension) 1972
Guild of Software Houses (GOSH) 1758

H

Seite

Hacker
Logging On VS 62
Handshaking 1792
Handshake-Leitung 1859, 1962
Hauptplatine (Motherboard) VS 68
Haushaltsbudget
Programm 1887—95
Hopper, Grace 1717

I

Seite

IBM VS 65
ICL Perq Arbeitsstation 1942, 1943
„Illustrator" 1798
Initialisation (Initialisierung) 1684
Intel 4004 VS 66
Interpreter
BASIC— 1798
Interrupt
Ereignis 1803—4
IRQ (Maskable Interrupt Request) 1790, 1830
ISR (Interrupt Service Routine) 1790
NMI (Non-Maskable Interrupt) 1790, 1846
Spectrum 2012—14
und OS des Acorn B 1790—2
Interval-Timer 1707, 1804, 1813
IRQ (Maskable Interrupt request) 1790, 1803, 1830, 1884
ISR (Interrupt Service Routine) 1790

J

Seite

Job-sharing
zwischen VIC-II und 6510 1883—4

K

Seite

Kernel (Kern) 1845
Kildall, Gary 1781, 1854
Kommerzielle Programme
COBOL 1800—2, 1826—8, 1856—8
Künstliche Intelligenz
Casino-Expertensysteme 1842—4
Expertensystem-Generator 1849—51

L

Seite

Laserplatte 1787, 1917, VS 72
-spieler, Pioneer LD-700 1917, 1919
„Last One" 1798—9, 1824—5
Lernsoftware
CAL (Computer-Assisted Learning) 1709—11
siehe auch Ausbildung; Schule
Lichtgriffel (Light pen) VS 61
LIFO (Last In First Out) 1879
Light pen (Lichtgriffel) VS 61
Line Printer (Zeilendrucker) VS 61
LISP VS 61
Local Area Network (Lokalnetz) VS 61
Lochkarten 1802
Logging on (Einloggen) VS 62
Logic (Logik) VS 61
Negative VS 69
Logic Gate (Logikgatter) VS 62, 1877
Logic State (Logischer Zustand) VS 62
Logic Symbols (Logiksymbole) VS 62
LOGO 1700—2, 1742
Dr. LOGO 1854—5
Lokalnetz (Local Area Network) VS 61
Loop (Schleife) VS 63
Low-Level Language (Maschinenorientierte Sprache) VS 63

M

Seite

Machine Independent (Maschinenunabhängig) VS 63
Macro (Makrobefehl) VS 64
Magnetic Card (Magnetkarte) VS 64
Magnetic Disk (Magnetplatte) VS 64
Makrobefehl (Macro) VS 64
Mail Box (Briefkasten) VS 65
Mainframe (Großrechner) VS 65
Management Information System (Management-Informationssystem) VS 65
Maschinenorientierte Sprache (Low-Level Language) VS 63
Maschinenunabhängig (Machine Independent) VS 63
Mass Storage (Massenspeicher) VS 65
Mathematische Operationen mit COBOL 1826—7
Matrix VS 66
Maus (Mouse) VS 68
Memory Hierarchy (Speicherhierarchie) VS 66
Memory Map
Commodore 64 1829
Spectrum 1938—41
Metacomco 1738
Microcomputer VS 65
Microprocessor (Microprozessor) VS 66
6510 1829—32, 1906, 1906
Motorola-68000 1833
MicroSoft
MS-Window 1973
Microtext 1798
Microwriter 1983
MIDI (Musical Instrument Digital Interface) 1794, 1795, VS 66
digitale Klangaufzeichnung 1969—71
Programmentwicklung 1984—5
Selbstbaukurs 1898—1900, 1907—9, 1922—5
Minicomputer VS 65, VS 67
MMI (Schnittstelle Mensch-Maschine) 1942, 1958—60
Modem VS 67
MODULA-2 1921
Modular Programming (Modulare Programmierung) VS 67
Modulation VS 67
Modulus (Modul) VS 67
Monitor VS 67
Motherboard (Hauptplatine) VS 68
Motorola 1833
68000-Microprozessor 1833
Mouse (Maus) VS 68
MP/M (Multi-Processing Monitor Control Program) 1781
MSX (Micro-Soft Extended BASIC)
-Standard-Computer 1683
Multimeter, digitales
Selbstbaukurs 1992-3
Multi-Tasking
mit Commodore Amiga 1378—9
und CP/M 1781
Musiksynthese
Backgroundmusik 1813
digitale Klangaufzeichnung 1969—71
Entwicklung 1793—5
Musik-Paket „Echo" 1933—4
Programme 1794, 1984—5
Selbstbau-MIDI-Interface 1898—1900, 1907—9, 1922—5, 1958—60

N

Seite

Namal Type & Talk 1777—9
Name VS 69
NAND VS 69
Negative Logik (Negative Logik) VS 69
Nesting (Verschachtelung) VS 69
Network (Netzwerk) VS 70
Network Architecture (Netzarchitektur) VS 70
Netzwerk (Network) VS 70
Nicht (NOT) VS 70
NMI (Non-Maskable Interrupt) 1790
Noise (Rauschen) VS 70
NOR VS 70
NOT (Nicht) VS 70
Numerical Analysis (Numerische Auswertung) VS 70

O

Seite

Object Code (Objektprogramm) VS 71
Object-Oriented (Objektorientiert) VS 71
Octal Notation (Oktalschreibweise) VS 71
ODER (OR) VS 72
One's Complement (Einer-Komplement) VS 71
Operand VS 71
Operating System (Betriebssystem) VS 71
Operation VS 71
Optical Disc (Optische Speicherplatte) VS 72
Optimisation (Optimierung) VS 72
Optokoppler 1900, 1907
OR (ODER) VS 72
Ordered Pair (Geordnetes Zahlenpaar) VS 72
OSBYTE-Aufrufe 1688—90, 1803
Oscilloscope (Oszilloskop) VS 72
OSFILE-Aufrufe 1714—16
OSFIND-Routinen 1753—4
OSWORD-Routinen 1706—8
OSWRCH 1812
Oszilloskop (Oscilloscope) VS 72

P

Seite

Pad 1696
PageMaker 1692
Paging-Register 1762
Parawedge-Programm für den C 64 1859—60
Peripheriegeräte
Tintenstrahldrucker 1836—7
Tracer, Selbstbaukurs 1815—17, 1834—5
Pferderennen
Buchmacher und Computer 1747—9
Glückspiel-Programme 1734—6
PIA (Peripheral Interface Adaptor) 1962
PILOT 1711
PIO-Chip 1961—3
Pioneer PX-7 1917—19
PIP (Peripheral Interchange Program) 1685
Piraten 1820
Platine
Haupt- (Motherboard) VS 68
Überblick 1877—8
Portabilität VS 63
Program Counter (Befehlszähler) 1878
Programmgenerator 1798—9, 1824—5
Expertensysteme 1849—51
Programmierhilfen
Programmgenerator 1798—9, 1824—5
Programmpiraterie 1820
Schutz für BASIC-Programme 1947—51
Programmspeicher 1780

Programmiersprache
  BCPL 1885
  COMAL
  Modula-2 1921
  PILOT 1711
SMALLTALK 1920—1, 1942—3
  und Computerunterricht 1700—2
  siehe auch BASIC; C; FORTH; LOGO; PROLOG
Programmierung
  modulare VS 67
  objektorientierte VS 71
PROLOG 1700
Prompt 1684

Q

Seite

„Quill" 1798

R

Seite

RAM
  dynamisches 1952—3
  statisches und dynamisches 1905
  -Vektoren des C 64 1830
Rauschen (Noise) VS 70
Rechenmaschine, Leibniz- 1827
Register 1877
  Taktzähler- 1878
Roboter
  Robot-Arm 1694—5, 1788—9
  Robotarm-Steuersoftware 1724—6, 1755—7, 1772—3
  -maus VS 68
Roland MP-401-MIDI VS 66
ROM (Read Only Memory) 1905, 1906

S

Seite

Sampling 1794
Schleife (Loop) VS 63
Schneider
  CPC464 1682, 1703
  CPC664 1703, 1781
  Dr. LOGO 1854—5
  FORTRAN für 1784
  Go-Programmprojekt 1870—1
  Interrupt-Sound-System 1793
  Joyce (PCW8256) 1703, 1805—7, 1983
  Sprachsynthesizer 1778, 1779
Schnittstelle
  Erweiterungs- 1847
  RS232- 1848
Schule
  CAL (Computer-Assisted Learning) 1709—11
  Computerunterricht 1700
  Form der Zukunft 1785—7
  Kostenprobleme 1759—61
Scrolling 1813—14
Sequenzer 1969
Sequenzing 1794
Serielle Buchse
  des C 64 1845, 1847
Servomotor
  elektrische Verbindungen 1694—5
  Steuer-Software 1724—6, 1755—7
Sicherheit
  Dongle 1820
  Schutz für BASIC-Programme 1946—51
Simulationspiel 1686—7, 1728—30, 1750—2, 1774—6, 1808—9, 1840—1, 1852—3, 1896—7
  Acorn B-Listing 1944—5
  Schneider 1976—8
  Spectrum 1926—8
Sinclair QL 1681, 1682
Sinclair Spectrum 1681, 1682
  Assemblerpakete 1940
  Cassettensystem 1989—91
  Datenströme 1979—81
  E/A-Kanalsteuerung 1966—8
  Interrupts 2012—14
  Memory Map 1938—41
  Robotarm-Steuerprogramm 1772—3
  Sampling 1794
  Tonerzeugung 1793
Sinclair Spectrum Plus 1682
SMALLTALK 1920—1, 1942—3
Soft Aid 1758
Softwarehäuser
  Artic Computing 1997
Sound, siehe Ton
Sound-Chip 1793
Spectravideo 318 & 328 1861—3
Spectrum, siehe Sinclair
Speicher
  -Bausteine 1905—6
  Massen- VS 65
  Programm- 1780
  Speicherhierarchie (Memory Hierarchy) VS 66
Spiele
  Go 1796—7, 1821—3, 1869—71, 1879—82, 1954—7, 1986—8, 2007—9
  Handelssimulationsspiel 1686—7, 1728—30, 1750—2, 1774—6, 1808—9, 1840—1, 1852—3, 1896—7, 1926—8, 1944—5, 1976—8
  Quo Vadis 1727
  Shadow of the Unicorn 1820
  SoftAid 1758
  siehe auch Abenteuerspiele; Glückspielen
Sprachsynthese
  Amsoft Speech Synthesizer 1778, 1779
  Namal Type & Talk 1777—9
Stack (Stapel)
  -Datenstruktur 1879
„Stand-Alone"-Microcomputer 1692
Steckleiste, Erweiterung
  für den C64 1845
Steuereinheit 1878
Steuerzeichen (Control Characters) 1720, 1721
String
  Verarbeitung mit FORTH 1696—7
Strom (Stream) 1966
  Datenströme im Spectrum 1979—81
Super-Mini VS 65
„Sycero" 1798—9, 1824—5
Synthesizer
  digitale Klangaufzeichnung 1969—71
  Echo 1933—4
  Entwicklung 1795
  Programmentwicklung 1984—5
  Selbstbau-MIDI-Interface 1898—1900, 1907—9, 1922—5, 1958—60

T

Seite

Tabellenkalkulationen
  mit COBOL 1856
Taktzähler-Register (Counter) 1878
Tandy-2000
  FORTRAN für 1784
Tastatur-Buffer 1798
Tatung Einstein
  Microtext 1798
Texture-mapping 1722
Textverarbeitung
  Programmpakete 1982—3
  Schneider Joyce 1805—7, 1983
  WordStar 1994—6
TFCB (Transient File Control Buffer) 1780
Tintenstrahldrucker 1836—7
Ton
  „Beep"-Tongenerator 1793
  Sound-Chip 1793
  siehe auch Musiksynthese
Toshiba HX-10 1683
TPA (Transient Program Area) 1780
Tracer, Digital
  Programmentwicklung 1864—6, 1875—6
  Selbstbauprojekt 1815—17, 1834—5
Tragbarer Computer (Portable) 1982
Transistor 1952

U

Seite

Uhr, interne
  Bildschirm-Uhr für den C 64 1831
  Interval-Timer 1707
Unix 1885
Unterbrechung, siehe Interrupt
Urlader (Bootstrap Loader) 1684
User Port
  des C 64 1845, 1846
Utilities
  Variablen-Austausch-Programm 1704—5, 1712—13, 1740—1

V

Seite

Variablen
  -Austausch-Hilfsprogramm 1704—5, 1712—13, 1740—1
Vektorentabelle des C 64 1830
Verschachtelung (Nesting) VS 69
VIA (Versatile Interface Adaptor) 1790, 1792, 1962
VIC-II, siehe Video-Interface-Chip
Video-Chip 2015—16
Video-Funktionen mit Commodore Amiga 1737
Video-Interface-Chip Commodore 64 1867—8, 1883—4, 2016
VoltMeter, digitales Selbstbaukurs 1992—3
Vorprozessor 1782

W

Seite

Wahrscheinlichkeitsrechnung 1765—7
  Bayessche Formel 1818—19
Warmstart 1721
Wildcard-Zeichen 1721
WIMP-Betriebssystem 1920—2, 1942—3
  Grafikstandard 1972—3
WordStar 1994—6

X

Seite

Xerox Dynabook-Projekt 1920—1
  Star 1942, 1943

Y

Seite

Yamaha CX5M 1794
  DX7 1795

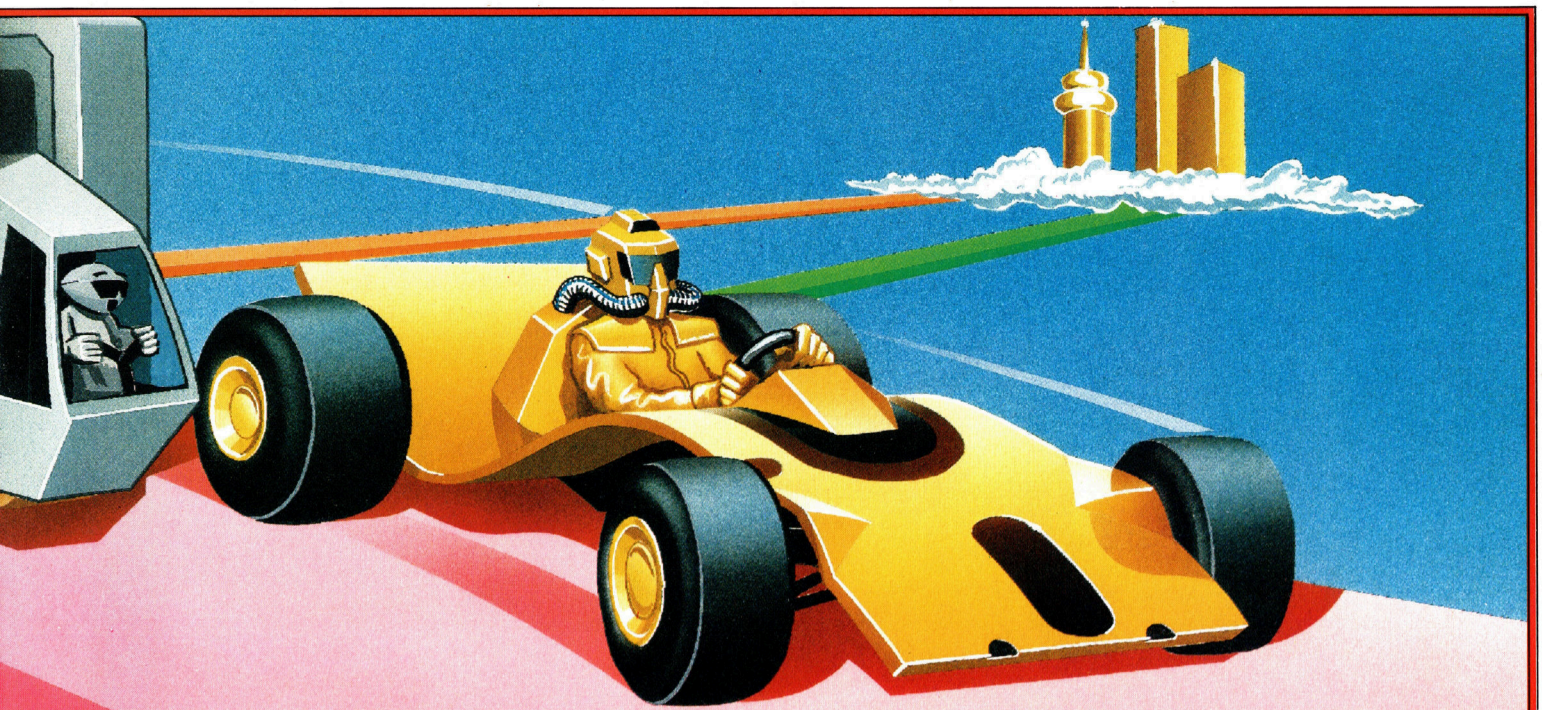
Z

Seite

Zählregister 1878
Zeilendrucker (Line Printer) VS 61
Zentraleinheit
  -Architektur 1877
  Schreiben und Lesen 1905—6
Zilog 1877

Alle zwölf Hefte erscheint ein solcher Teilindex. Der Gesamtindex erscheint mit dem letzten Heft — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.

Stichwörter, die auf die vorletzte Heftseite hinweisen, sind als VS und mit der betreffenden Heftnummer gekennzeichnet.



Code für
Anführungszeichen.
Wird es gefunden, erfolgt
bei BNE keine Verzweigung.
Stattdessen wird der Inhalt von
033E ins X-Register geladen und
mit 0 verglichen. Ist er Null, wird 1 ins X-Register geladen und zurück nach 033E geschrieben. Im anderen Fall führt BNE AD zur Reset-Routine. In beiden Fällen folgt aber ein JMP NEXT, das uns zur Prüfung des nächsten Bytes führt.

Wird kein Anführungszeichen gefunden, dann führt BNE AC auf ein LDX &033E, das heißt, der Inhalt von 033E wird ins X-Register geladen. Dann wird X mit 1 verglichen, um herauszufinden, ob das Flag gesetzt ist. Wenn ja, führt BEQ NEXT auf die Prüfung des nächsten Bytes. Wenn nein, wird nach einem Cassetten-Befehl gesucht.

Da das nächste Byte des BASIC-Programms noch im Akkumulator ist, können wir es durch CMP mit 147 vergleichen, dem Token für LOAD. Ist es gleich 147, dann führt die folgende Verzweigung, BEQ LOOP, auf das Label LOOP. Ist es ungleich 147, vergleichen wir mit 148, dem Token für SAVE, 149 (VERIFY)
und 159 (OPEN).
Wird keins dieser
Token gefunden, führt ein
JMP NEXT auf den Anfang der
Routine, um das nächste Byte zu untersuchen. Andernfalls durchläuft der Prozessor die LOOP-Routine.

Das Programm verzweigt

INY inkrementiert das Y-Register, so daß durch LDA (&FB),Y das nächste Byte nach dem Cassettenbefehl in den Akkumulator geladen wird. Die Routine prüft dann mit CMP #32, ob das folgende Zeichen ein Leerzeichen ist. In ASCII ist das Leerzeichen durch die 32 symbolisiert.

In diesem Fall verzweigt das Programm durch BEQ LOOP auf den Anfang der LOOP-Routine, um zu prüfen, ob auch das nächste Zeichen ein Leerzeichen ist. Andernfalls wird durch CMP
#0 geprüft, ob das
Zeilenende erreicht
ist. Wenn ein Zeilenende
erkannt wird, verzweigt das Programm durch BEQ COLINE auf die Routine COLINE.

Liegt weder ein Leerzeichen noch Zeilenende vor, wird durch Vergleich mit 58 – dem ASCII-Wert des Doppelpunktes – überprüft, ob ein Doppelpunkt im BASIC-Text steht. Auch in diesem Fall wird nach COLINE verzweigt.

Als nächstes vergleicht #CMP 44

das vorliegende Byte mit 44 (ASCII-Wert des Komma). Wenn das Byte als Komma identifiziert wird, verzweigt das Programm durch BEQ COMMA auf die Routine COMMA. Damit sind die wesentlichen Tests abgeschlossen, und das Programm kann durch ein JMP LOOP an den Anfang der LOOP-Routine zurückspringen.

Danach verzweigt der Prozessor dann schließlich zur Routine QUOTES (Anführungszeichen).

Der Sinn der Übung ist, hier eine „,8" einzufügen, so daß aus den Cassettenbefehlen SAVE „NAME" oder SAVE P$ die Diskettenbefehle SAVE „NAME",8 oder SAVE P$,8 werden.

Zu diesem Zweck wird der Akku mit 44, also ASCII für Komma geladen, und das Komma mit STA (&FB),Y in das Programm eingefügt. Dann wird Y inkrementiert und dieselbe Prozedur mit 56, ASCII für 8, wiederholt.

Die QUOTES-Routine beginnt mit LDY #1 und setzt damit Y auf 1 zurück. Danach wird das erste Byte nach dem Recorder-Befehl mit LDA (&FB),Y geladen und mit 0 bzw. 58 verglichen, um zu prüfen, ob das Ende des BASIC-Befehls schon erreicht wurde.

Hier wird eine spezielle Programmtechnik angewandt, bei der mit BEQ

Wenn ein Komma gefunden wurde, durchläuft der Prozessor die COMMA-Routine. Hier wird zuerst das Y-Register inkrementiert und mit LDA (&FB),Y das Komma folgende Zeichen in den Akkumulator geladen.

Cassettenrecorder im Einsatz

Das Byte wird sodann mit 49 verglichen, dem ASCII-Code für 1. Dabei bedeutet die 1 hinter einem Ausgabebefehl, daß das Gerät #1, also der Cassettenrecorder, benutzt werden soll. 2 steht für den Bildschirm, 4 für den Drucker und 8 für die Diskettenstation. Wird also keine 1 gefunden, kann diese Routine sofort wieder zum Programmanfang zurückspringen.

Ist das auf Komma folgende Byte tatsächlich 1, wird der Akkumulator durch LDA #56 mit dem ASCII-Zeichen für 8 geladen. Die 8 wird dann durch ein STA (&FB),Y anstelle der 1 in das BASIC-Programm geschrieben.

SS auf einen unbedingten Sprungbefehl verzweigt wird, nämlich JMP NEXT. Das ist an dieser Stelle notwendig, weil ein relativer Sprung nicht weiter als 128 Bytes zurück oder 127 Bytes nach vorne reichen kann. Tatsächlich folgt auf relative Sprunganweisungen nur ein Byte, während es bei absoluten Sprüngen zwei sind.

Die große Prüfung

Die nächste Prüfung gilt dann einem Anführungszeichen. Wenn durch CMP #34 eines gefunden wurde, führt BEQ UU zur UU-Routine. Wird kein Anführungszeichen gefunden, dann wird Y inkrementiert und mit JMP RR die Schleife geschlossen. Diese Schleife wird vom Prozessor also so oft durchlaufen, bis er auf ein Anführungszeichen oder das Ende des BASIC-Befehls (Doppelpunkt oder Anführungszeichen) trifft.

Obwohl das Y-Register in jedem Fall inkrementiert werden muß, kann man kein Byte dadurch sparen, daß INY vor der Verzweigung eingefügt wird. INY beeinflußt das Zero-Flag, das vom BEQ UU für die Verzweigung benötigt wird.

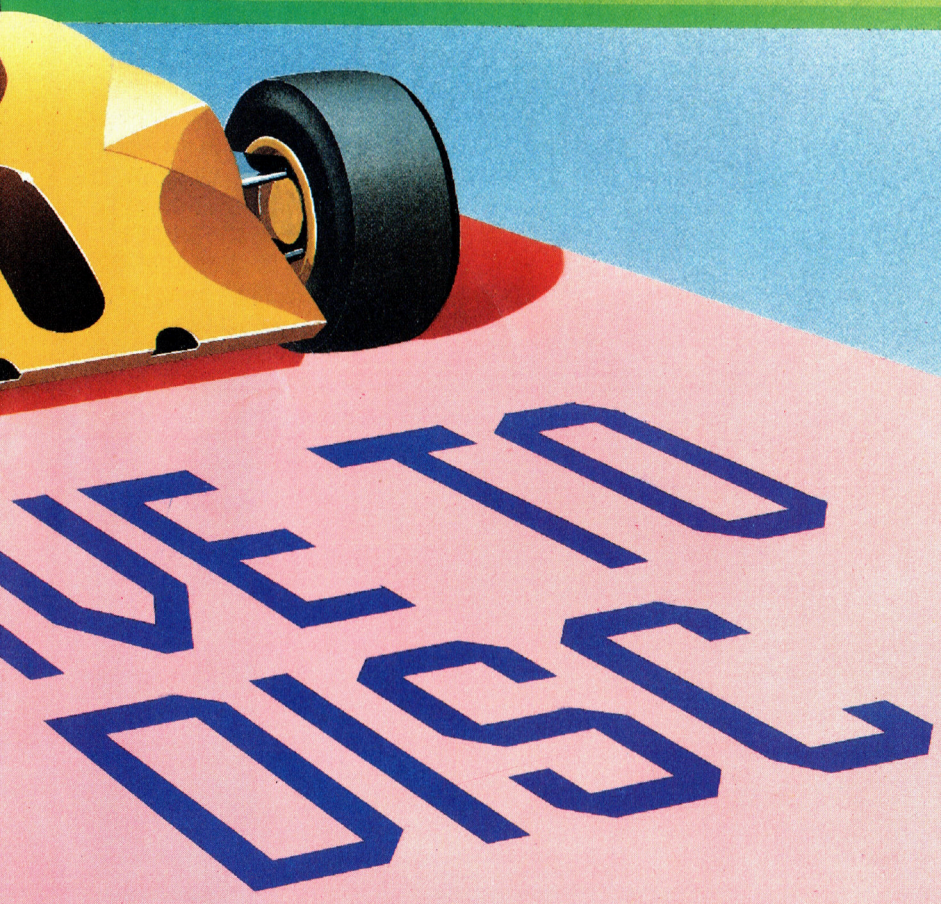
Das nächste Zeichen wird dann mit 32 (ASCII-Code für Leerzeichen) verglichen. Gegebenenfalls wird Y solange inkrementiert, bis etwas anderes als ein Leerzeichen gefunden ist.

Als nächstes wird nach einem @ gesucht. Falls der Programmname bereits ein @ :□ enthält, der das Überschreiben ermöglicht, sollte dieses Zeichen nicht wiederholt werden. In diesem Fall (wenn das Byte 64, also ASCII @ ist) erfolgt ein Rücksprung auf NEXT. Ansonsten gelangen wir nach VV, wo die INSERT-Routine dreimal aufgerufen wird, um Platz für die Bytes 64, 58 und 32 zu schaffen, den ASCII-Code für @: und ein Leerzeichen. Ein JMP NEXT leitet danach zur Prüfung des restlichen BASIC-Programms weiter.

Die INSERT-Routine ist von den bisher beschriebenen Programmteilen schon öfter aufgerufen worden. Ihre Aufgabe ist es, den Rest des BASIC-Programms im Speicher vom erreichten Punkt ab um ein Byte nach oben zu schieben.

Dazu wird erst einmal der Inhalt des Y-Registers nach FF auf der Zeropage abgelegt. Beim Verlassen von INSERT wird dieser Wert nämlich wieder gebraucht, andernfalls wüßte man nicht mehr, wo im Programm man sich befindet. Andererseits wird das Y-Register auch innerhalb der INSERT-Routine benötigt, weshalb sein Inhalt gleich zu Anfang übernommen und nach dem Ablauf der Routine auch wieder ins Y-Register zurückgeladen werden muß, damit sich das erwünschte Ergebnis auch einstellt.

Als nächstes wird das Carry-Flag mittels CLC zurückgesetzt, damit die nötigen Additionen ausgeführt werden können. Außerdem wird das höherwertige Byte des Zeigers aus FC mit LDA &FC und STA $033D nach 033D übertragen.



Standortbestimmung

Der Inhalt des Y-Registers, der uns ja sagt, wieweit hinter einem Recorderbefehl wir uns befinden, wird in den Akku gebracht, und zum niederwertigen Byte des Zeigers in FB und FC addiert, der noch auf den Recorderbefehl selbst zeigt. Das Ergebnis der Addition zeigt also auf das letzte Byte des Programms, das nicht verschoben werden soll, und wird in 033C gespeichert. Ein Übertrag bei der Addition führt zum Ignorieren der Verzweigung BCC. Das höherwertige Byte des Zeigers in 033D wird dann inkrementiert.

Die Systemvariable, die auf den Anfang der BASIC-Variablen zeigt – ein Byte hinter dem Schluß des BASIC-Programms – wird für spätere Operationen von 2D und 2E nach FD und FE kopiert.

Auf das Label SUB folgt ein kurzer Programmteil zur Dekrementierung

von FD und FE. Nach dem ersten Durchlauf zeigen dann FD/FE auf das Ende des BASIC-Programms. Dieser Programmteil wird so oft wiederholt, bis das gesamte BASIC-Programm Byte für Byte abgearbeitet ist.

Dazu wird das Y-Register mit 0 geladen, in den Akkumulator kommt das Byte, auf das FD/FE zeigt. Dann wird Y auf 1 inkrementiert, und mit STA (&FD),Y eine Speicheradresse höher wieder abgelegt.

Dabei wird durch Vergleich zwischen den Zeigern in 033C/033D bzw. FD/FE geprüft, ob der Durchlauf abgeschlossen ist. Falls die Werte nicht gleich sind, wird das nächste Byte durch einen Sprung nach SUB verschoben.

Großer Zeigersalat

Stimmen beide Zeiger überein, ist die Verschiebung vollständig. Man muß dann nur noch den Zeiger auf das Ende des BASIC-Programms berichtigen. Das geschieht durch INC &2D, BNE EE, INC &2E.

Zum Schluß muß durch LDY &FF das Y-Register wiederhergestellt werden. RTS führt dann zu dem Befehl nach dem Aufruf dieser Routine.

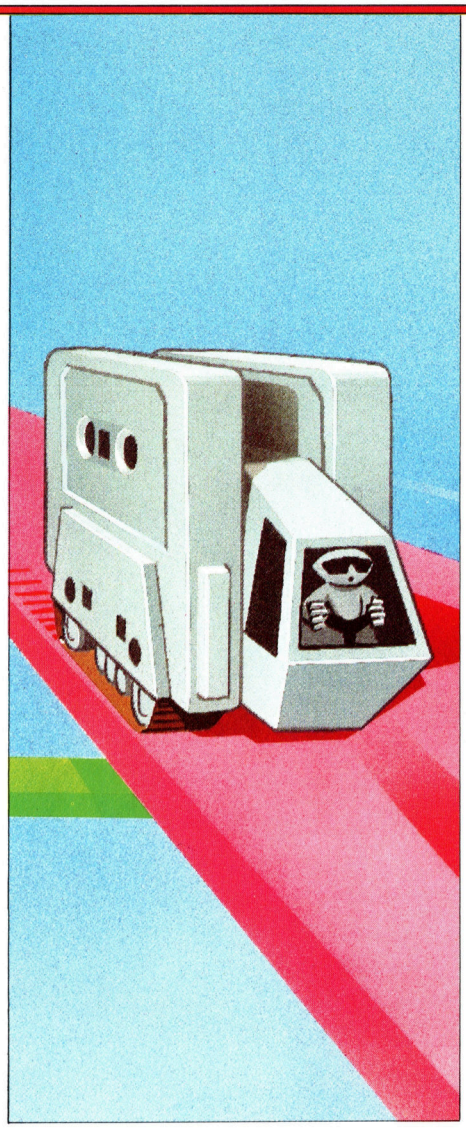
Durch das Herumschieben des BASIC-Programms im Speicher sind natürlich auch die Zeiger auf die Zeilenanfänge durcheinander gekommen. Sie stehen immer am Anfang der vorhergehenden BASIC-Zeile.

Glücklicherweise gibt es eine ROM-Routine, die das korrigiert. Zu diesem Zweck muß nur das Unterprogramm A533 mit JSR &A533 aufgerufen werden. RTS bringt uns dann wieder ins BASIC.

Was jetzt noch folgt, ist eigentlich kein Maschinencode mehr – es sind Daten. Dabei ist BYT ein Assembler-Befehl, der ein Byte im Speicher für den folgenden Wert freihält.

Zum Aufrufen des Programms dient:
SYS 49152

Vergessen Sie aber nicht, das BASIC-Programm, das Sie auf Floppybetrieb umstellen wollen, zuerst zu laden. Alles andere ist Zeitverschwendung – der Rechner stürzt sonst nämlich einfach ab. Wenn Sie dieses Programm speichern wollen, sollten Sie einen Maschinencode-Monitor benutzen.



Mikrotip

Es gibt noch einen Fall, mit dem das Floppy-Konverterprogramm Schwierigkeiten hat, nämlich dann, wenn der Programmname in einem String gespeichert ist, also etwa
SAVE P$

Das Programm setzt ',8' hinter das $, und der Befehl wird beim ersten Mal ausgeführt. Da der Programmname aber in einer Zeile wie
P$="NAME"

definiert wird, kann das Programm nicht ,@ :□' hinzufügen, wie es beim Überschreiben nötig wäre. Der nächste Schreibversuch führt deshalb zu einer Fehlermeldung.

Unser Maschinenprogramm funktioniert auch auf dem VC 20. Allerdings ist hier der Speicher hinter 49152 nicht geschützt. Das Programm sollte deshalb bei 7168 beginnen. Sie schützen diesen Bereich vorher mit

POKE 51,255
POKE 52,27
POKE 55,255
POKE 56,27
CLR

Auch die ROM-Routine, die die Zeiger auf die Zeilenlänge berichtigt, liegt hier anders: Sie beginnt bei C533, der Befehl lautet also JSR &C533.

Wenn Sie keinen fertigen Assembler für Ihren VC 20 haben, müssen Sie das Programm über einen Maschinencode-Monitor eingeben. Den folgenden Hex-Maschinencode können Sie direkt eingeben, ohne das Programm selbst zu assemblieren:

```
A9 ØØ 8D 3E Ø3 18 A5 2B 69 Ø3 85 FB A5 2C
63 ØØ 85 FC AØ ØØ E6 FB DØ Ø2 E6 FC A5
FB C5 2D DØ Ø9 A5 FC C5 2E DØ Ø3 4C 6Ø
1D B1 FB C9 ØØ DØ 21 C8 B1 FB DØ Ø8 C8
B1 FB DØ Ø3 4C 6Ø 1D A5 FB 69 Ø3 85 FB
A5 FC 69 ØØ 85 FC A2 ØØ 8E 3E Ø3 4C 12 1C
C9 22 DØ ØF AE 3E Ø3 EØ ØØ DØ ED A2 Ø1
8E 3E Ø3 4C 12 1C AE 3E Ø3 EØ Ø1 FØ A8 C9
93 FØ ØF C9 94 FØ ØB C9 95 FØ Ø7 C9 9F FØ
Ø3 4C 12 1C C8 B1 FB C9 2Ø FØ F9 C9 ØØ FØ
1C C9 3A FØ 18 C9 2C FØ Ø3 4C 7D 1C C8
B1 FB C9 31 FØ Ø3 4C 12 1C A9 38 91 FB 4C
DD 1C 88 B1 FB C9 2Ø FØ F9 CØ ØØ DØ 11
C8 2Ø 1C 1D B9 63 1D 91 FB C8 CØ Ø8 DØ
F3 4C 12 1C B1 FB C9 22 FØ Ø7 C9 24 FØ Ø3
4C 12 1C C8 2Ø 1C 1D 2Ø 1C 1D A9 2C 91 FB
C8 A9 38 91 FB AØ Ø1 B1 FB C9 ØØ FØ Ø4 C9
3A DØ Ø3 4C 12 1C C9 22 FØ Ø4 C8 4C DF
1C C8 B1 FB C9 2Ø FØ F9 C9 4Ø DØ Ø3 4C 12
1C 2Ø 1C 1D 2Ø 1C 1D 2Ø 1C 1D A9 4Ø 91
FB C8 A9 3A 91 FB C8 A9 2Ø 91 FB 4C 12 1C
84 FF 18 A5 FC 8D 3D Ø3 98 65 FB 9Ø Ø3 EE
3D Ø3 8D 3C Ø3 A5 2D 85 FD A5 2E 85 FE C6
FD A9 FF C5 FD DØ Ø2 C6 FE AØ ØØ B1 FD
AØ Ø1 91 FD AD 3C Ø3 C5 FD DØ E7 AD 3D
Ø3 C5 FE DØ EØ E6 2D DØ Ø2 E6 2E A4 FF 6Ø
2Ø 33 C5 6Ø 22 4Ø 3A 2Ø 22 2C 38
```

Das Programm wird mit SYS 7168 aufgerufen.

Vergessen Sie aber nicht, das BASIC-Programm, das Sie auf Floppybetrieb umstellen wollen, zuerst zu laden: Der Rechner stürzt sonst nämlich einfach ab! Wenn Sie dieses Programm zudem speichern wollen, sollten Sie auch hier einen Maschinencode-Monitor benutzen.

Wir machen einen Zug

Im letzten Artikel entwickelten wir eine Routine, die eine Zugauswahl ermöglichte. Im folgenden entwerfen wir eine weitere Routine, damit der Computer strategische Züge ausführen kann.

Das Spiel Go ist wahrscheinlich das schwierigste auf den Computer umsetzbare Brettspiel. Die ungewöhnliche Brettgröße und die Vielzahl möglicher Züge verhindern die Anwendung der üblichen Baumstruktur zur Vorausberechnung, wie sie bei vielen Spielen, wie Schach, Backgammon oder Dame, eingesetzt wird.

In einem Spiel wie Schach gibt es eine begrenzte Anzahl möglicher Züge (ca. 30 je Position), so daß alle Zugmöglichkeiten durchgerechnet werden können und der Computer garantiert einen Zug findet. Leider ist dieses System bei Go nicht verwendbar. Unsere Gruppenauswertungsroutine testet nur Züge bei Gruppen mit weniger als drei Freifeldern. Gibt es auf dem Brett keine Gruppen in dieser Situation, wird kein Zug gefunden. In zukünftigen Artikeln werden wir uns mit anderen Auswertungsroutinen befassen, die auf einfachen Vergleichstechniken basieren, also auch nur Züge für bestimmte Situationen finden. Wir brauchen demnach eine Auswertungsroutine.

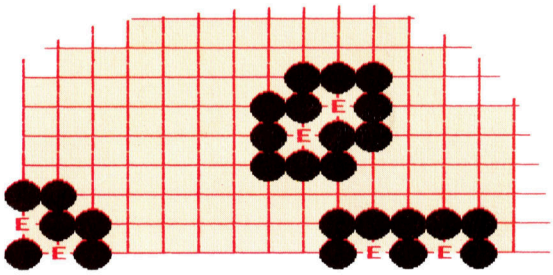
Der einfachste Weg hierzu wäre, per Zufall jeden denkbaren Zug zu spielen. Eine solche Routine sieht wie folgt aus:

```
5000 DEF PROCrandom_move
5010 LOCAL L%
5020 :
5030 FOR L%=17 TO 255
5040 IF FNlegality (L%, black%) = 0 THEN
     location% = L%
5050 NEXT
5060 ENDPROC
```

Diese Routine gibt, sofern möglich, einen legalen Zug aus. Sie kann aber verbessert werden, indem die Schleife an einer zufälligen Brettposition gestartet wird und einem gesetzten Stein in Abhängigkeit seiner Freifelder eine Punktzahl zuordnet – dies wird automatisch durch FNlegality berechnet und in der Variable clib% ausgegeben. Trotzdem ist der Computer aber immer noch nicht in der Lage, „sinnvolle" Züge zu machen.

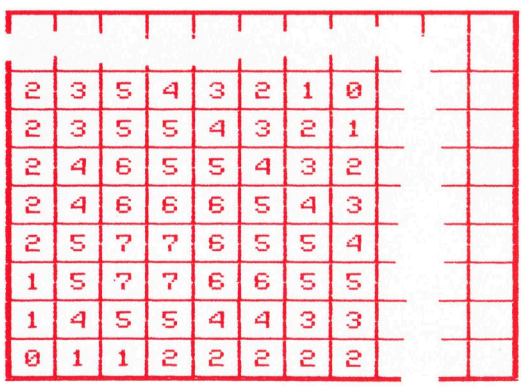
Die gewünschte Routine sollte zufällige Züge spielen, aber nur auf wichtigen Positionen. Ein möglicher Weg zu diesem Ziel ist die „Bewertung" der Positionen. Diese Technik wird auch bei Spielen wie Othello verwendet, wo bestimmte Brettpositionen eine größere Bedeutung haben als andere.

Obwohl die Entscheidung über die Vor- und Nachteile spezieller Positionen schwierig ist, gelten bestimmte Bereiche auf dem Brett als wichtiger als andere. Stellen Sie sich beispielsweise die Anzahl notwendiger Steine zum Einschließen von zwei separaten Freifeldern vor. Wie Sie sich erinnern, werden sie „Augen" genannt, und jede Gruppe mit zwei oder mehreren Augen ist sicher. Die Anzahl der notwendigen Steine zum Formen einer „sicheren" Gruppe ist davon abhängig, ob sie sich in einer Ecke, am Rand oder in der Mitte des Brettes befindet. In einer Ecke braucht man nur sechs Steine, was offensichtlich besser ist, als zwölf Steine in der Mitte des Brettes.



Die ersten Züge werden daher in oder nahe den Eckpositionen gesetzt. Von dort werden die Spieler normalerweise versuchen, an den Seiten des Brettes (auf der dritten oder vierten Linie) Gebiete zu gewinnen. Bei einem Angriff hat man dann den Vorteil, den Eckstein als Basis zur Bildung von zwei Augen verwenden zu können.

Mit diesen taktischen Grundüberlegungen sollte das folgende Diagramm der Brett-„Gewichtung" einen Sinn ergeben.



Diese Struktur ist für alle vier Ecken des Brettes symmetrisch und wird in den Zeilen 1020 bis 1230 initialisiert.

Die den verschiedenen Positionen zugeordneten Werte wurden aufgrund der eben beschriebenen Taktik gewählt, sind jedoch nicht unbedingt optimal. Sie können die Werte nach Belieben ändern. Ändern Sie lediglich die Zeilen 60 und 80 wie folgt:

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

Fünftes Modul

Acorn B:

```
220 DIM board% 255,weight% 255
1010 :
1020 DEF PROCread_weights
1030 LOCAL A%,B%,C%,D%,X%,Y%,V%
1040 RESTORE 1150
1050 FOR Y%=1 TO 8
1060     FOR X%=Y% TO 8
1070         A%=16*Y%+X% : C%=16*Y%+(16-X%)
1080         B%=16*X%+Y% : D%=16*X%+(16-Y%)
1090         READ V%
1100         weight%?A%=V%:weight%?(272-A%)
=V%
1110         weight%?B%=V%:weight%?(272-B%)
=V%
1120         weight%?C%=V%:weight%?(272-C%)
=V%
1130         weight%?D%=V%:weight%?(272-D%)
=V%
1140     NEXT:NEXT
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA   4,5,5,4,4,3,3
1170 DATA     7,7,6,6,5,5
1180 DATA       7,6,5,5,4
1190 DATA         6,5,4,3
1200 DATA           4,3,2
1210 DATA             2,1
1220 DATA               0
1230 ENDPROC
1240 :
1250 REM*********************************
1350 PROCread_weights
2620 IF location%=0 THEN PROCfind_any_m
ove:T$="RND"
 2630 IF location%=0 THEN end%=TRUE:ENDP
ROC
 2820 score=(8*S%/L%-clib%+2*L%)*weight%
?tloc%(Q%)
 3510 :
3520 DEF PROCfind_any_move
3530 LOCAL L%,hi,score
3540 hi=-9999
3550 FOR L%=17 TO 255 :score=RND(1)+wei
ght%?L%
 3560    IF (L% AND 240)=0 OR (L% AND 15)
=0 OR score<=hi THEN 3580
 3570    IF FNlegality(L%,black%)=0 AND c
lib%>2 THEN hi=score:location%=L%
 3580    NEXT
3590 ENDPROC
3600 :
3610 REM*********************************
3800 board%?P%=0:IF C%=black% THEN weig
ht%?P%=0
```

Commodore 64:

```
220 BOARD=49152:WEIGHT=BOARD+256
305 GOSUB 363
362 :
363 REM READ-WEIGHTS ROUTINE
364 RESTORE:FOR X=1 TO 4:READ Y:NEXT
365 FOR Y=1 TO 8
366 FOR X=Y TO 8
367 A%=16*Y+X:C%=16*Y+(16-X)
368 B%=16*X+Y:D%=16*X+(16-Y)
369 READ V%
370 POKE WEIGHT+A%,V%:POKE WEIGHT+272-A%
,V%
371 POKE WEIGHT+B%,V%:POKE WEIGHT+272-B%
,V%
372 POKE WEIGHT+C%,V%:POKE WEIGHT+272-C%
,V%
373 POKE WEIGHT+D%,V%:POKE WEIGHT+272-D%
,V%
374 NEXT:NEXT
375 DATA 0,1,1,2,2,2,2,2
376 DATA   4,5,5,4,4,3,3
377 DATA     7,7,6,6,5,5
378 DATA       7,6,5,5,4
379 DATA         6,5,4,3
380 DATA           4,3,2
381 DATA             2,1
382 DATA               0
383 RETURN
384 :
385 REM*********************************
1350 GOSUB 363
2620 IF LOCAT%=0 THEN GOSUB 3520:T$="RND
"
2630 IF LOCAT%=0 THEN FIN%=-1:RETURN
2820 SCR=(8*BS/BL-CLIB%+2*BL)*PEEK(WEIGH
T+TLOC%(Q))
3510 :
3520 REM FIND-ANY-MOVE ROUTINE
3540 HI=-9999
3550 FOR I=17 TO 255:SCR=RND(0)+PEEK(WEI
GHT+I)
3560 IF (I AND 240)=0 OR (I AND 15)=0 OR
 SCR<=HI GOTO 3580
3570 LP%=I:LC%=BLACK%:GOSUB 3890:IF LL%=
0 AND CLIB%>2 THEN HI=SCR:LOCAT%=I
3580 NEXT
3590 RETURN
3600 :
3610 REM*********************************
3800 POKE BOARD+RP%,0:IF RC%=BLACK% THEN
 POKE WEIGHT+RP%,0
3805 SK%(STACK%)=RP%:STACK%=STACK%+1
```

Für die Schneider-, C64- und Spectrum-Versionen verfahren Sie wie im Abschnitt BASIC-Dialekte beschrieben. Um unterschiedliche Wertetabellen einzufügen, ändern Sie zuerst Zeile 220:

220 DIM board%255,weight1%255,weight2%255

Die PROCread_weight-Routine kann dann so geändert werden, daß eine Bewertungstabelle im weight1%-Array und die andere in weight2% abgelegt wird. Fügen Sie jetzt in Zeile 60 weight%=weight1% und weight%=weight2% in Zeile 80 ein, verwendet der Computer für jeden Spieler eine unterschiedliche Tabelle.

Jetzt können wir die PROCfind_any_move-Routine integrieren. Sie ähnelt der Routine für zufällige Züge, verwendet jedoch die Bewertungstabelle, um einen „sinnvollen" Zug zu finden. Zusätzlich wird die Variable clib% überprüft, um zu gewährleisten, daß der Computer nicht 10 setzt, daß eine seiner Gruppen auf dem Brett mit weniger als drei Freifeldern verbleibt. Diese Routine wird in Zeile 2620 aufgerufen. Kann die neue Routine keinen möglichen Zug finden, wird in Zeile 2630 die Spiel-

ende-Variable gesetzt und der Programmlauf beendet.

Da wir nun schon einmal Bewertungstabellen erstellt haben, können wir sie auch zur Optimierung anderer Auswertungsroutinen verwenden. So kann etwa in der Gruppenauswertungsroutine die Punktzahl mit dem Bewertungswert multipliziert werden (Zeile 2820). Auch hier lassen sich die Faktoren beliebig ändern.

Verhindert werden muß aber auch, daß das Programm auf die Positionen zuvor eingenommener Steine setzt. Basierend auf unseren Berechnungen werden die ersten Steine zumeist auf die „besseren" Brettpositionen gesetzt. Nimmt man einige Steine ein, werden diese Positionen wieder frei. Der Computer bemerkt dies natürlich und fängt sofort an, Steine in diese Bereiche zu setzen, obwohl sie von Ihnen eingeschlossen sind. Um das zu verhindern, wurde Zeile 3800 eingefügt, die in einfacher Form eine dynamische Bewertung ermöglicht. Das bedeutet, daß die Bewertungswerte sich im Verlauf des Spieles ändern. Beim Schach-Spiel zum Beispiel wollen Sie die Bewertungen auf Basis der Königsposition festlegen. Während der ersten Züge wird dann den Eckpositionen eine höhere Bedeutung zugeordnet, um den König in eine sichere Position zu bringen. Gegen Spielende dagegen ändern sich die Werte dahingehend, daß der König eine Zentralposition einnehmen kann.

In unserem Go-Programm ist Zeile 3800 Bestandteil der PROCremove-Routine, die verlorene Steine vom Brett entfernt. Handelt es sich um schwarze Steine, ändert sich der Positionswert auf 0. Nun ist es sehr unwahrscheinlich, daß der Computer noch auf diese Position einen Stein setzt.

Schneider CPC 464/664:

```
220 board=&A000:weight=&A100
1010 :
1020 REM read weights routine
1040 RESTORE 1150
1050 FOR y%=1 TO 8
1060 FOR x%=y% TO 8
1070 a%=16*x%:c%=16*y%+(16-x%)
1080 b%=16*x%+y%:d%=16*x%+(16-x%)
1090 READ v%
1100 POKE(weight+a%),v%:POKE (weight+272
-a%),v%
1110 POKE(weight+b%),v%:POKE (weight+272
-b%),v%
1120 POKE(weight+c%),v%:POKE (weight+272
-c%),v%
1130 POKE(weight+d%),v%:POKE (weight+272
-d%),v%
1140 NEXT x%,y%
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA     4,5,5,4,4,3,3
1170 DATA       7,7,6,6,5,5
1180 DATA         7,6,5,5,4
1190 DATA           6,5,4,3
1200 DATA             4,3,2
1210 DATA               2,1
1220 DATA                 0
1230 RETURN
1240 :
1250 REM ***********************
1350 GOSUB 1020:REM read weights
2620 IF location%=0 THEN GOSUB 3520:T$="
RND":REM any move
2630 IF location%=0 THEN over%=1:RETURN
2820 score=(8*gs%/gl%-clib%+2*gl%)*PEEK(
weight+tloc%(q%))
3510 :
3520 REM find any move routine
3540 hi=-9999
3550 FOR i%=17 TO 255:score=RND(1)+PEEK(
weight+i%)
3560 IF (i% AND 240)=0 OR (i% AND 15)=0
OR score<=hi THEN 3580
3570 lp%=i%:lc%=black%:GOSUB 3890:IF ll%
=0 AND clib%>2 THEN hi=score:location%=i
%
3580 NEXT i%
3590 RETURN
3600 :
3610 REM ***********************
3800 POKE (board+rp%),0:IF rc%=black% TH
EN POKE (weight+rp%),0
```

Sinclair Spectrum:

```
220 LET board=64000:LET weight
=board+256
1020 REM read-weights routine
1040 RESTORE 1150
1050 FOR y=1 TO 8
1060 FOR x=y TO 8
1070 LET a=16*y+x: LET c=16*y+(1
6-x)
1080 LET b=16*x+y: LET d=16*x+(1
6-y)
1090 READ v
1100 POKE weight+a,v: POKE weigh
t+272-a,v
1110 POKE weight+b,v: POKE weigh
t+272-b,v
1120 POKE weight+c,v: POKE weigh
t+272-c,v
1130 POKE weight+d,v: POKE weigh
t+272-d,v
1140 NEXT x: NEXT y
1150 DATA 0,1,1,2,2,2,2,2
1160 DATA     4,5,5,4,4,3,3
1170 DATA       7,7,6,6,5,5
1180 DATA         7,6,5,5,4
1190 DATA           6,5,4,3
1200 DATA             4,3,2
1210 DATA               2,1
1220 DATA                 0
1230 RETURN
1250 REM ***********************
1350 GO SUB 1350:REM read weights
2620 IF location=0 THEN  GO SUB
3520: LET t$="RND"
2630 IF location=0 THEN  LET end
=1: RETURN
2820 LET score=(8*gs/gl-clib+2*g
l)*PEEK (weight+j(q))
3510:
3520 REM find-any-move routine
3540 LET hi=-9999
3550 FOR i=17 TO 255: LET score=
RND+PEEK (weight+i)
3560 IF INT (i/16)=0 OR i-16*INT
 (i/16)=0 OR score<=hi THEN  GO
TO 3580
3570 LET lp=i: LET lc=black: GO
SUB 3890: IF ll=0 AND clib>2 THE
N  LET hi=score: LET location=i
3580 NEXT i
3590 RETURN
3610 REM ***********************
3800 POKE board+rp,0: IF rc=blac
k THEN  POKE weight+rp,0
```

BASIC-Dialekte

Das Programm kann mit folgenden Änderungen so modifiziert werden, daß der Computer gegen sich selbst spielt. Durch Initialisierung von zwei Bewertungstabellen, verwendet der Computer diese abwechselnd, so daß man die Wirkung der Tabellen vergleichen kann.

Schneider CPC 464/664:

```
60  mve% = mve% + 1: black% = 2:
    white% = 1: weight = weight1:
    GOSUB 2540
80  mve% = mve% + 1: black% = 1:
    white% = 2: weight = weight2:
    GOSUB 2540
220 board = &A000: weight1 = &A100:
    weight2 = &A200
```

Spectrum:

```
60  LET move = move + 1: LET black = 2:
    LET white = 1: LET weight = weight1:
    GO SUB 2540
80  LET move= move + 1: LET black = 2:
    LET white = 1: LET weight = weight2:
    GO SUB 2540
220 LET board = 64000: LET weight1 =
    board + 256: LET weight2 = board +
    512
```

Commodore 64:

```
60  MOVE% = MOVE% + 1: BLACK% = 2:
    WHITE% = 1: WEIGHT = W1: GOSUB
    2540
80  MOVE% = MOVE% + 1: BLACK% = 1:
    WHITE% = 2: WEIGHT = W2: GOSUB
    2540
220 BOARD = 49152: W1 = BOARD +
    256: W2 = BOARD + 512
```

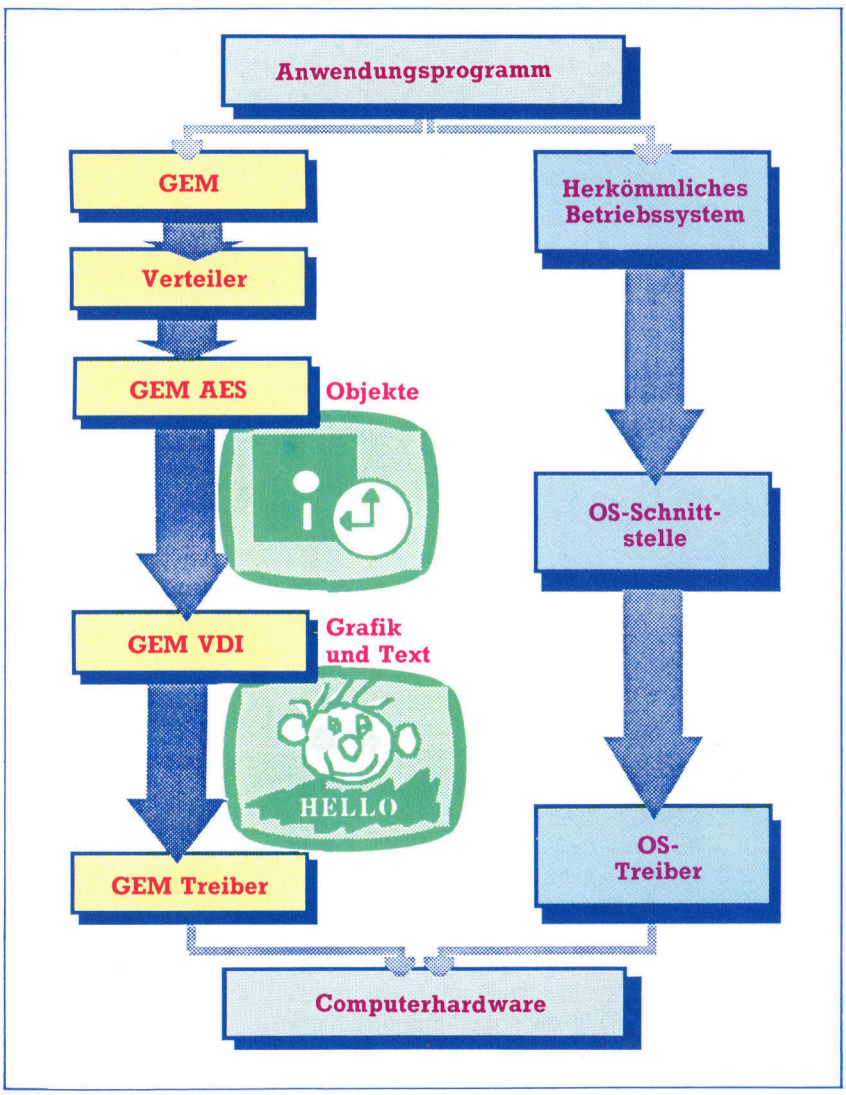
Arbeitsplatz am offenen Fenster

In der vorigen Folge erfuhren wir, daß GEM durch Verwendung des internationalen GKS-Standardrahmens einen hohen Grad an Übertragbarkeit erreicht. Wir sehen uns nun die GEM-Umgebung genauer an und untersuchen Abläufe, Programmstrukturen und Möglichkeiten für den kommerziellen Einsatz.

Die leichte Bedienung von GEM läßt kaum etwas von den außerordentlich komplizierten Vorgänge in seinem Inneren ahnen. Das VDI (Virtual Device Interface) zur Entwicklung von maschinenunabhängigem Code besteht aus zwei Teilen. Ein Teil – das GDOS – spielt die Rolle des herkömmlichen Betriebssystems, und bearbeitet im wesentlichen Eingaben des Anwenders. Der zweite Teil (die Gerätetreiber) ist auf die aktuelle Hardware eingestellt.

Während „Microsoft Window" fast ausschließlich bei kommerziellen Computersystemen eingesetzt wird, findet man GEM auch bei kleineren Systemen. Bei dem neuen Atari 520ST (mit dem Motorola 68000) gehört GEM bereits zu Lieferumfang, und auch auf dem Apricot und dem RML Nimbus ist es verfügbar.

Erfahrene Anwendungsprogrammierer programmieren normalerweise nicht für eine bestimmte Maschine, sondern für ein Betriebssystem. WIMPs, die auf GKS aufbauen (darunter auch GEM), sprechen ihre Grafikkomponenten nach einem festen Standard an. Unter GEM bzw. GKS entwickelte Programme sind damit theoretisch auf jede Hardware übertragbar, die die entsprechenden Kernroutinen für Grafik besitzt. GEM, GSX und GKS bieten einen Softwarerahmen, der sich um das bestehende System und die Grafikhardware legt. Dabei steuert das maschineneigene Betriebssystem ganz normal Tastatureingaben und Dateiverwaltung, während der zusätzliche Softwarerahmen die Grafik übernimmt.



Obwohl Digital Research sein Produkt auch GEM DOS nennt, führt GEM selbst keine der üblichen DOS-Funktionen aus. Befehle, die nicht per Tastatur kommen, werden in normale Betriebssystemaufrufe umgewandelt, die Aktualisierung der Bildschirmanzeige oder anderer Grafikperipherie dagegen steuert das darunterliegende GSX-System – das Herz von GEM. Dieser Vorgang läuft zusätzlich zu dem herkömmlichen Befehlszeileninterpreter ab.

Bei jedem Ablauf, der mit Anzeigedaten arbeitet, wird der normale zeichenweise Bildaufbau durch einen Grafikstrom mit Bit-Map Format ersetzt, der dem entsprechenden Gerätetreiber Daten liefert. Jede Hardwarekomponente besitzt einen eigenen Treiber, der ihre Geräteeigenschaften (Eingabe, Ausgabe, Auflösung, Farbe etc.) berücksichtigt.

Ein Fenster für „Hallo"

Die Programmierung unter GEM ist recht kompliziert. Ohne Standardmodule, die Ereignisse steuern, Fenster aktivieren oder deaktivieren (und darunterliegende Inhalte wiederherstellen) etc., muß der GEM-Programmierer jede Routine aus GEM/GSX VDI Grundelementen aufbauen. So ist unter GEM beispielsweise der Aufbau eines Fensters mit einem einfachen Text, wie PRINT „Hallo!" in BASIC schon eine umfangreiche Programmieraufgabe. Zunächst wird die Größe und Position des Fensters mit Standardwerten initialisiert. Diese Werte müssen veränderbar sein, damit das Fenster ver-

schoben oder auf eine andere Größe gebracht werden kann. Weiterhin muß der Programmierer Attribute wie Text- und Hintergrundfarben, Schriftart, Schriftgröße etc. bestimmen.

Die Aktivierung eines anderen Fensters ist ein „Ereignis", das den aktuellen Ablauf überlagern kann. Die Ausführung wird dann entweder für die Dauer der Überlagerung gestoppt oder läuft im Hintergrund weiter (MS-Windows, Concurrent DOS und die neue Multitasking-Version von GEM). Bei der Reaktivierung sollte der Vorgang wieder als oberes Fenster auf der Anzeige stehen und mit seinem Inhalt die anderen Bildschirmfenster überlagern.

Jedes Programm – auch das kleinste – muß den gesamten Code für die eigene Fenstersteuerung enthalten, darunter Routinen, die ständig den aktuellen Inhalt des Fensters und seine Position in Bezug zu nicht dargestellten Daten feststellen. Das bedeutet aber auch, daß Verwaltungsroutinen rund zehn KByte des Quellentextes belegen, bevor das Programm überhaupt etwas ausführen kann.

Für GEM oder GSX gibt es keine Standardprozeduren, die diese Verwaltungsroutinen erledigen. Softwarehäuser, die mit GEM arbeiten, programmieren in MODULA-2, PASCAL, C, BCPL oder Assembler, haben aber trotz all ihrer Erfahrung zuweilen immer noch große Probleme mit GEM. Wenn Sie vorhaben, in ein paar hundert BASIC-Zeilen eine einfache GEM-Anwendung für Ihren Heimcomputer zu schreiben, können Sie davon ausgehen, daß diese Aufgabe etwas Zeit erfordern wird. Wenn Sie jedoch in einer der erwähnten Sprachen programmieren können, steht Ihnen eine Reihe von Produkten zur Verfügung, die brauchbare Programmierschnittstellen zu dem darunterliegenden WIMP-System bieten.

Mehrere GEM-Angebote

Die englische Firma TDI brachte einen der ersten einsatzfähigen „Tool Kits" für WIMP auf den Markt. Der MODULA-2/ST bietet den Besitzern eines Atari 520ST die Möglichkeit, unter GEM in der Sprache zu programmieren, die von Nikolaus Wirth für Lilith entwickelt wurde.

Für ernsthafte GEM-Anwender stehen mehrere Systeme zur Verfügung:

- Ein Atari 520ST (mit GEM), MODULA-2 und dem TDI Toolkit sind schon recht günstig zu haben;
- Ein Apricot, GEM und ein guter MODULA-2, PASCAL, BCPL oder C-Compiler kosten etwas mehr;
- Ein CP/M–86 oder MS-DOS System mit GSX, Pro-PASCAL und der Prospect Modulbibliothek für die Grafikschnittstelle sind zwar teuer, aber erprobt und vielseitig einsetzbar;
- Der IBM PC, AT oder ein kompatibles Gerät, ein 8086 Assembler oder C und die GEM-Programmierhilfen von Digital Research eignen sich für erfahrene Programmierer mit Systemkenntnissen und viel Zeit und Geld.

Allen, die keine Zeit zum Programmieren haben, bietet Digital Research eine Reihe von Anwendungen, die mit der GEM-Umgebung arbeiten. GEM Desktop ist eins der „Schreibtischsysteme", die (mit der neuen Multitaskingversion von GEM) mehrere Anwendungen steuern, Dateien verwalten, drucken, plotten und die Funktionen der Echtzeituhr und des „Taschenrechners" übernehmen. GEM Write ist eine WIMP-ähnliche Textverarbeitung, GEM Paint ein Zeichenpaket für den künstlerischen Bereich. Jedes der Systeme ist einzeln oder im Paket erhältlich.

Die Gesichter von GEM

GEM besteht aus zwei Hauptteilen: dem AES „Applications Environment Services" (Steuerung der Anwendungsumgebung) und dem VDI „Virtual Device Interface" (virtuelle Geräteschnittstelle). GEM und AES enthalten eine Bibliothek von Subroutinen (mit Rahmenmodulen für Fenstersteuerung, Eingaben per Maus, Meldungsanzeige und Objektgrafik) und einen Verteiler mit Multitasking – im Augenblick lassen sich drei Vorgänge gleichzeitig steuern, die neue GEM-Version ist für zwölf Vorgänge ausgelegt.

Das VDI des GEM enthält im wesentlichen das aus GKS hervorgegangene GSX ‚Grafics System Extension' (Grafik Systemerweiterung), das mit RASTER-OPS und FACES erweitert wurde. RASTER-OPS sind bitweise logische Vorgänge (AND, OR, XOR) mit Quellen- und Datenblöcken, während FACES Schriftarten in dynamisch ladbare Dateien speichert.

Diese maschinennahen Module sind zweiteilig. Das GDOS „Graphics Davice Operating System" (Betriebssystem für die Grafikperipherie) entspricht dabei der normalen Betriebssystemschnittstelle. Den zweiten Teil bilden die austauschbaren Gerätetreiber.

Das GDOS enthält alle grundlegenden hardwareunabhängigen Grafikfunktionen (die separaten Treiber steuern die einzelnen Gerätekomponenten). Ebenso wie herkömmliche Betriebssysteme in ihren Anwendungsprogrammen den systematischen Speicherzugriff etc. steuern, so bietet auch GDOS ein standardisiertes und übertragbares Grafiksystem, das unabhängig von der eingesetzten Hardware arbeitet.

Eine einzige Subroutine spricht das VDI mit folgenden fünf Argumenten an:

- **Steuerarray**
- **Array der Eingabeparameter**
- **Eingabearray für Punktkoordinaten**
- **Array der Ausgabeparameter**
- **Ausgabearray für Punktkoordinaten**

In der Programmiersprache C gibt es Verbindungselemente, die als mnemotische Kürzel dienen und die Komplexität der zahllosen GEM-Module vereinfachen. So sprechen beispielsweise alle Funktionsaufrufe, die mit v_ anfangen, VDI Routinen an.

Hinter der Maske

Die Z80-CPU bietet drei maskierbare Interrupts, mit denen sich Maschinencodeprogrammierer Prozessorzeit „borgen" können. Wir untersuchen, welche Rolle Interrupts im Betriebssystem des Spectrum spielen und wie mit Vektoren eigene Maschinencodemodule angesprochen werden.

Da die interruptgesteuerten Module des Spectrum OS sehr komplex sind, sollten Sie vorsichtig damit umgehen. Ein unbeabsichtigt „abgeschaltetes" Interrupt kann das Funktionieren der Maschine behindern – es könnten beispielsweise keine Tastatureingaben mehr möglich sein – oder das System während der Ausführung eines BASIC-Programms abstürzen lassen.

Die Z80-CPU des Spectrum arbeitet mit zwei Arten von Interrupts – maskierbare und nicht maskierbare (NMI). Der Unterschied ist leicht beschrieben: Sie können die CPU so programmieren, daß sie maskierbare Interrupts ignoriert. Auf nicht-maskierbare Interrupts wird der Prozessor jedoch in jedem Fall reagieren.

Die NMI-Interrupts lassen sich auf dem Spectrum nicht besonders gut einsetzen, da die entsprechende Bearbeitungsroutine im ROM nicht zuverlässig arbeitet. Die Designer des OS wollten es dem Anwender ursprünglich ermöglichen, über die Speicherstellen &5CB0 und &5CB1 eine Adresse angeben zu können, die von der CPU bei Empfang eines NMI automatisch angesprochen wird. Der Z80 führt bei einem NMI jedoch standardmäßig den Maschinencode bei &66 aus und verursacht damit normalerweise einen Systemreset. Aus diesem Grund konzentrieren wir uns auf den Einsatz der maskierbaren Interrupts.

Die Z80-CPU besitzt mehrere Interruptmechanismen. Wir werden hier jedoch nur auf die Arten bzw. Modi eingehen, die für das OS des Spectrum Bedeutung haben.

50 Interrupts/sec

Bei der Initialisierung (d. h. beim Anschalten der Maschine oder bei Ausgabe des Befehls NEW) setzt die CPU automatisch den Interruptmodus 1 (IM1). Die Sinclair ULA (Uncommitted Logic Array – nicht festgelegte Anordnung von Logikschaltungen) liefert 50 Interruptimpulse pro Sekunde an die CPU. Im IM1-Modus führt die CPU bei jedem Empfang eines Interruptsignals den Befehl RST &0038 aus. Damit wird ein Sprung auf die Routine für Tastaturabfragen ausgelöst und der FRAMES-Zähler bei 23672 und 23674 inkrementiert. (Diese drei Bytes bilden einen 24-Bit-Zähler, der alle zwanzig Millisek. aktualisiert wird.)

Nach Interpretation einer BASIC-Zeile wartet der BASIC-Interpreter auf ein Interrupt, bevor er sich die nächste Zeile vornimmt. Wenn man die Interrupts abschaltet, bricht die BASIC-Programmausführung ab.

Nach Ausführung des Befehls DI (Disable Interrupts – Interrupts abschalten) ignoriert die CPU alle maskierbaren Interrupts. Der Befehl EI (Enable Interrupts) aktiviert sie wieder.

Im OS des Spectrum gibt es mehrere Routinen, bei deren Ausführung die Interrupts abgeschaltet sein müssen. Dies sind normalerweise zeitabhängige Routinen wie das BEEP-Modul des Tongenerators und die Lade- und Speicherroutinen der Cassettenstation.

Interrupts können auch von dem ZX Drucker, dem Interface 1 oder den Microdrives zeitweilig abgeschaltet werden. Nach Beendigung der Routine werden sie wieder aktiviert.

Während der Interruptabschaltung wird der FRAMES-Zähler nicht inkrementiert, das heißt, er „verliert Zeit". Da praktisch bei jedem Programmablauf Interrupts eintreten können, soll-

Für alle Fälle

Obwohl man fast immer davon ausgehen kann, daß der Datenbus beim Auftreten eines Interrupts ein 255 enthält, gibt es Peripheriegeräte, die diesen Wert ändern. Das Problem tritt nicht auf, wenn Sie den Interruptmodus 2 (IM2) ohne Peripheriegeräte verwenden.

Als alternative Methode, alle nur möglichen Werte des Datenbusses abzufangen, füllen sie eine Speicherseite mit identischen Werten und laden vor dem Anschalten von IM2 die Seitennummer in das I-Register. Beim Eintreten eines Interrupts holt sich der Z80 die Adresse der Bearbeitungsroutine aus einer beliebigen Position der angegebenen Seite – die exakte Position bestimmt der Datenbus.

Zu diesem Zweck wird das I-Register zunächst mit &FC geladen und die Bytes von &FC00 bis &FD00 auf den Wert &FB gesetzt. Danach schalten Sie IM2 an. Wenn der Datenbus beim nächsten Interrupt &C3 enthält, holt der Z80 sich die Adresse der Interrupt-Bearbeitungsroutine aus der Speicherstelle &FCC3 – den &FC-Teil der Adresse liefert das I-Register. Da alle 256 Bytes der Seite &FC (und Byte Null der Seite &FD) auf dem gleichen Wert stehen, lautet die Vektoradresse in jedem Fall &FBFB.

Dazu noch zwei wichtige Anmerkungen: Die Interrupt-Bearbeitungsroutine muß immer bei einer Adresse mit identischen Lo- und Hi-Bytes liegen (z. B. &C4C4 oder &FDFD). Berücksichtigen Sie weiterhin die Möglichkeit, daß der Adreßbus beim Auftreten eines Interrupts &FF enthält. In diesem Fall sucht der Z80 die Bearbeitungsroutine bei den Adressen nnFF (Lo-Byte) und (nn+1)00 (Hi-Byte) – (nn ist der Wert des I-Registers). Setzen Sie daher immer auch das erste Byte der darauffolgenden Seite.

ten Sie die Interrupts bei zeitkritischen Modulen möglichst abschalten. Vor der Rückkehr ins BASIC müssen sie jedoch unbedingt wieder aktiviert werden.

Um Interrupts praktisch einsetzen zu können, muß zuerst ein flexibler Interruptmodus eingestellt werden. Dafür eignet sich am besten der Interruptmodus IM2, der flexibler als IM1 ist. Während IM1 immer mit RST auf die Adresse &0038 springt, läßt sich IM2 auf jeden beliebigen Interruptvektor leiten.

Der Interruptvektor enthält die Startadresse der Bearbeitungsroutine, die beim Auftreten eines maskierbaren Interrupts ausgeführt wird. Dabei zeigt das I-Register der CPU an, wo sich der Vektor überhaupt befindet. Die Adresse der Interrupt-Vektorroutine ergibt sich aus dem Inhalt des I-Registers und des Datenbusses zum Interruptzeitpunkt. In einigen Systmen setzt die Interruptquelle außerdem ein Byte auf den Datenbus, das der CPU mitteilt, welches Gerät die Unterbrechung ausgelöst hat.

Die Sinclair ULA arbeitet nicht mit dieser Methode, doch ist der Spectrum so konstruiert, daß der Datenbus den Wert 255 enthält, wenn sich keine weitere Eingabe darauf befindet. Das I-Register liefert nun das höherwertige Byte der 16-Bit-Adresse des Interruptvektors und der Datenbus das niederwertige (in diesem Fall &FF). So befindet sich der Interruptvektor immer an der Grenze einer Speicherseite: Das niederwertige Adreßbyte liegt in &nnFF und das höherwertige in &(nn+1)00 (nn ist der Inhalt des I-Registers).

Wenn I beispielsweise den Wert &FB enthält, befindet sich die Vektoradresse bei &FBFF. Das niederwertige Vektorbyte (bei &FBFF) liefert dann das Lo-Byte der Interrupt-Bearbeitungsroutine und &FC00 das Hi-Byte.

Achtung! Hardwareprobleme

Bearbeitungsroutinen für Interrupts dürfen nicht an jeder beliebigen Speicherposition liegen – so sind zum Beispiel die ersten 16 KBytes dem ROM zugeordnet und stehen damit nicht zur Verfügung, um hier Routinen unterzubringen, die die Interruptsteuerung beeinflussen. Außerdem können Hardwareprobleme auftreten, wenn der Wert des I-Registers zwischen 64 und 127 liegt.

Sehen wir uns nun an, wie der Interruptmodus 2 angeschaltet und der Interruptvektor mit der Adresse der Bearbeitungsroutine eingesetzt wird:

```
F3          di                  ;disable interrupts
210000      ld   hl,ADDRESS     ;get ISR address in HL
22FFFB      ld   (#FBFF),hl     ;get address into vector
3EFB        ld   a,#FB          ;hi-byte of vector...
ED47        ld   i,a            ;...into I register
ED5E        IM   2              ;set int mode 2
FB          ei                  ;re-enable interrupts
C9          ret                 ;back to BASIC
```

Das Listing setzt natürlich voraus, daß sich bei ADDRESS eine Routine befindet, die den Interrupt bearbeiten kann – ist dies nicht der Fall, stürzt das Programm ab. Idealerweise sollte die Bearbeitungsroutine auch alle Aufgaben übernehmen, die sonst vom normalen Interruptmodus des Spectrum ausgeführt werden. Dies geschieht am besten über einen Aufruf der Routine bei &38. Das folgende Assemblerprogramm stellt den Interruptmodus auf IM2 um und veranlaßt die CPU, bei jedem Interrupt die Routine bei ADDRESS auszuführen (die hier nur die üblichen Interruptfunktionen des Spectrum erledigt).

```
            org  60000             ;specify start add
     vector: equ  #FEFF            ;FEFF is vector add
216FEA change: ld   hl,addres      ;get address into HL
22FFFE      ld   (vector),hl       ;set up vector
F3          di                     ;disable interrupts
3EFE        ld   a,#FE             ;set up the...
ED47        ld   i,a               ;...I register
ED5E        im   2                 ;change int mode
FB          ei                     ;re-enable interrupts
C9          ret                    ;back to BASIC
F3     addres: di                  ;turn off interrupts
FF          rst  #38               ;normal IM1 procedure
FB          ei                     ;re-enable interrupts
C9          ret
```

Der Aufruf der Routine bei Adresse CHANGE setzt den neuen Interruptmodus und den Vektor. Danach wird die Routine bei ADDRESS alle Fünfzigstelsekunde angesprochen. Im Augenblick erledigt diese Routine zwar nur die üblichen Standardaufgaben, doch werden wir in Kürze einige interessante Funktionen dafür programmieren.

Die folgende Routine kann jederzeit den normalen Interruptmodus wiederherstellen:

```
F3          di
3E3F        ld   a,#3f             ;reset I register...
ED47        ld   i,a               ;...to its usual value
ED56        im   1                 ;normal mode
FB          ei                     ;re-enable interrupts
C9          ret
```

Unseren eigenen Module werden als Subroutinen angelegt und mit CALL aufgerufen:

```
F3     addres: di
F5             push af             ;save registers...
C5             push bc
D5             push de
E5             push hl
FF             rst  #38            ;call normal ISR
F3             di                  ;ISR did EI, so DI again
CD0000         call PROG_ADD       ;call our own routine
E1             pop  hl             ;restore registers...
D1             pop  de
C1             pop  bc
F1             pop  af
FB             ei
C9             ret
```

Interruptroutinen verlangsamen den Spectrum. Denken Sie auch daran, die Interrupts am Anfang Ihrer eigenen Interruptroutinen auszuschalten, damit während der Ausführung keine weitere Unterbrechung eintreten kann.

Musik mit Unterbrechung

Der Tondatencompiler stellt ein Menü mit drei Bearbeitungsmöglichkeiten dar. „C-COMPILE“ verwandelt die Melodie (in den DATA-Befehlen zwischen Zeile 10 und 900 gespeichert) in einen Codeblock, der von der Interruptroutine eingesetzt wird. ‚P‘ spielt Ihnen die Melodie zur Prüfung vor. ‚R‘ verzweigt ins BASIC zurück.

Beachten Sie, daß Sie bei Option ‚C‘ angeben müssen, in welchem Speicherbereich Sie die Tondaten unterbringen wollen. Speichern Sie sie oberhalb von RAMTOP (der Wert muß bereits geändert sein, damit Platz für die Daten frei ist). Wenn die Töne compiliert und gespeichert sind, werden die Basisadresse und die Programmlänge angezeigt. Am besten notieren Sie sich diese beiden Werte, da Sie sie später noch brauchen. Zur Speicherung des Codes geben Sie von BASIC aus SAVE „NOTEDATA“ CODE (Basisadresse), (Programmlänge in Bytes) ein. Für den Einsatz Ihrer eigenen Melodien brauchen Sie nun nur noch die Zeilen 10 bis 90 zu löschen und eigene Daten zwischen den Zeilen 10 und 900 zu speichern.

Basisadresse der Routine ist 65021. Stellen Sie daher vor dem Laden des Codes sicher, daß RAMTOP heruntergesetzt wurde (CLEAR 65000). Nach Assemblierung und Laden des Programmcodes übertragen Sie den vom Tondatencompiler erzeugten Code in den Speicher (siehe unten). Stellen Sie auch hier sicher, daß über RAMTOP ausreichend Platz zur Verfügung steht.

10 LET L=65152: LET V=(* Basisadresse der Tondaten *)
20 POKE L+1,INT (V/256):
POKE L,V-PEEK(L+1)*256

RAND USR 65041 schaltet nun auf IM2, und die Musik fängt an zu spielen. RAND USR 65071 aktiviert wieder IM1 und stoppt die Musik.

Tondatencompiler

```
  1 REM >>> INTERRUPT MUSIC <<<
  2 REM >>Note Data  Compiler<<
  3 REM
  6 REM *NOTE DATA IN STANDARD SPECTRUM BEEP FORM
AT *
  7 REM
  9 RESTORE
 10 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,9,2,10,1,10,2,10,1,12,5,14
 20 DATA 1,14,2,7,1,7,1,7,1,6,1,7,2,16,1,14,2,14,
1,10,2,9,1,9,2,9,1,10,5,12
 30 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,12,2,11,1,19,2,19,1,19,5,19
 40 DATA 1,17,2,16,1,19,1,19,1,18,1,19,2,14,1,19,
1,19,1,18,1,19,2,12,1,11,2,12,1,11,3,12
 50 DATA 3,10,1,9,1,8,1,9,2,14,1,12,3.2,9,3.2,5,3
.2,2,3.2,7,5,5
 60 DATA 1,5,1,7,1,9,1,10,2,16,1,14,3.2,12,3.2,17
,3.2,16,3.2,14,5,12
 70 DATA 1,12,2,14,1,14,1,14,1,13,1,14,3,16,3,9
 80 DATA 2,17,1,17,1,19,1,17,1,19,5,21
 90 DATA 1,21,2,19,1,17,2,14,1,10,3.2,9,3.2,5,3.2
,7,3.2,4,5,5
997 REM
998 REM *DO NOT ERASE LINE 999*
999 DATA 255,255
1000 REM
2000 REM **** MENU ****
2010 BORDER 1: PAPER 1: INK 6
2020 CLS
2030 PRINT AT 5,4;"INTERRUPT MUSIC COMPILER";AT 10
,5;"C -- Compile note data";AT 12,5;"P -- Play tun
e in beeps";AT 14,5;"R -- Return to BASIC"
2040 LET a$=INKEY$
2050 IF a$="c" OR a$="C" THEN  GO TO 5000
2060 IF a$="p" OR a$="P" THEN  GO TO 3000
2070 IF a$="r" OR a$="R" THEN  CLS : STOP
2080 GO TO 2040
3000 RESTORE
3010 READ d,f: IF d=255 AND f=255 THEN  GO TO 2000
3020 BEEP d/10,f-6: GO TO 3010
5000 RESTORE : CLS : INPUT "BASE ADDRESS OF NOTE D
ATA? ";dd1: POKE 23301,INT (dd1/256): POKE 23300,d
d1-(256*PEEK 23301): CLEAR dd1-1: LET dd1=PEEK 233
00+256*PEEK 23301
5005 LET f=0: LET d=dd1
5010 READ delay,pitch
5015 LET delay=INT (delay*6)
5020 LET freq=(1.0594631^pitch)*256
5030 LET bip=INT ((437500/freq)-30.125)
5035 IF delay=255*6 THEN  POKE d,255: GO TO 6000
5037 LET f=f+1: PRINT AT 10,10;"NOTE )";f
5040 POKE d,delay
5060 POKE d+1,bip-(INT (bip/256)*256)
5070 POKE d+2,INT (bip/256)
5080 LET d=d+3
5090 GO TO 5010
6000 PAUSE 50: CLS : LET len=(d+2)-dd1
6010 PRINT "LOCATION OF NOTE DATA IN MEMORY=";dd1'
''"NUMBER OF BYTES OF NOTE DATA....";len
6020 INPUT "PRESS 'ENTER' TO RETURN TO MENU "; LIN
E a$: GO TO 2000
```

Ladeprogramm in BASIC

```
 10 RESTORE 9000
 20 CLEAR 64000
 30 LET cqs=0: FOR f=65021 TO 65156
 40 READ dat: POKE f,dat
 50 LET cqs=cqs+dat
 60 NEXT f
 70 IF cqs<>18850 THEN  PRINT "ERROR IN DATA !!!"
: STOP
 80 PRINT "OK NO PROBLEMS IN DATA"
 90 STOP
9000 DATA 0,0,221,229,229,197,213,245,195,52,254,2
41,209,193,225,221,225,195,56,0,33,255,253,34,253,
253,243,62,253,237,71,237,94,42,128,254,34,126,254
,62,1,50,130,254,61,50,131,254,251,201,243,237,86,
251,201,58,130,254,254,0,202,8,254,58,131,254,254,
0,202,75,254,61,50,131,254,195,8,254,42,126,254,12
6,254,255,202,108,254,50,131,254,35,126,35,95,126,
35,34,126,254,103,107,17,4,0,205,181,3,243,195,8,2
54,42,128,254,34,126,254,62,1,50,130,254,61,50,131
,254,195,8,254,0,0,0,0,0,0,
```

Assemblerlisting

```
                org  65021
0000    vect:   defw 0                  ;2 bytes for vector
DDE5    inter:  push ix
E5              push hl
C5              push bc
D5              push de
F5              push af
C334FE          jp   start              ;jp to music routine
F1      enprg:  pop  af
D1              pop  de
C1              pop  bc
E1              pop  hl
DDE1            pop  ix
C33800          jp   56                 ;jp to ROM ISR
21FFFF  on:     ld   hl,inter           ;set up vector
22FDFD          ld   (vect),hl
F3              di
3EFD            ld   a,253
ED47            ld   i,a                ;get vect hi-byte in I
ED5E            im   2                  ;set int mode 2
2A80FE  initi:  ld   hl,(datad)         ;point to data
227EFE          ld   (locat),hl
3E01            ld   a,1                ;switch = 1 = play note
3282FE          ld   (swtch),a
3D              dec  a                  ;delay = 0
3283FE          ld   (delay),a
FB              ei
C9              ret                     ;back to BASIC
F3      off:    di
ED56            im   1                  ;set int mode 1
FB              ei
                ret
C9      ;
3A82FE  start:  ld   a,(swtch)          ;get switch in a
FE00            cp   0
CA88FE          jp   z,enprg            ;if swtch 0 then end
3A83FE          ld   a,(delay)
FE00            cp   0
CA4BFE          jp   z,nwnot            ;if delay 0 then play
3D              dec  a                  ;delay=delay-1
3283FE          ld   (delay),a          ;store new delay value
C388FE          jp   enprg              ;goto end routine
2A7EFE  nwnot:  ld   hl,(locat)         ;get data location
7E              ld   a,(hl)             ;get next delay in a
FEFF            cp   255                ;255 = data finished
CA6CFE          jp   z,reset            ;data ended so reset
3283FE          ld   (delay),a          ;store new delay
23              inc  hl                 ;point to next databyte
7E              ld   a,(hl)             ;get it in a
23              inc  hl                 ;point to next databyte
5F              ld   e,a                ;store byte1 in e
7E              ld   a,(hl)             ;store byte2 in a
23              inc  hl                 ;point to next byte
227EFE          ld   (locat),hl         ;store next-byte-address
67              ld   h,a                ;get frequency data...
6B              ld   l,e                ;...in HL
110400          ld   de,4               ;note duration of 4
CDB503          call 949                ;call ROM beep routine
F3              di                      ;ROM-BEEP did EI, so DI
C388FE          jp   enprg              ;goto end routine
2A80FE  reset:  ld   hl,(datad)         ;reset various...
227EFE          ld   (locat),hl         ;...pointers
3E01            ld   a,1
3282FE          ld   (swtch),a
3D              dec  a
3283FE          ld   (delay),a
C388FE          jp   enprg              ;goto end routine
0000    locat:  defw 0                  ;reserve memory for...
0000    datad:  defw 0                  ;various pointers...
00      swtch:  defb 0
00      delay:  defb 0
```

Gespeicherter Schirm

Bei der Betrachtung der Komponenten eines Computers untersuchen wir zwei unterschiedliche Arten von Video-Chips und sehen, wie Binärmuster in Bildschirminhalte umgewandelt werden.

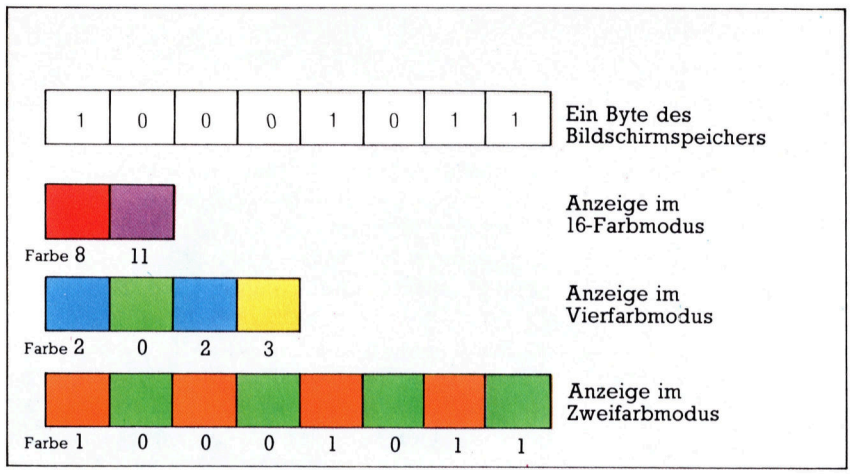
Qualität und Vielseitigkeit der Bildschirmdarstellung sind wichtige Verkaufsargumente für Computer. Wieviel Farben sind möglich? Wieviel Zeichen lassen sich in einer Zeile darstellen? Lassen sich Sprites steuern? Viele dieser Fragen beantwortet der Video-Chip des Gerätes.

Ein Video-Chip (auch CRTC = „Cathode Ray Tube Controller" – Bildschirmsteuerung – genannt) führt zunächst einmal Grundfunktionen aus: Er nimmt die Bytes der im RAM gespeicherten Anzeigedaten und bringt sie als Formen und Farben auf den Bildschirm. Unterschiede zwischen Videochips beziehen sich hauptsächlich auf die Technik, mit der sie Daten in Bildinformationen umwandeln.

Die erste Art von Video Chip hält Anzeige und Farbdaten gemeinsam im Speicher und macht damit die Umwandlung in ein Bild recht einfach. Mit dieser Methode arbeiten unter anderem Schneider-Computer. Chips dieser Art unterstützen üblicherweise eine ganze Palette unterschiedlicher Anzeigearten und können Zeichengröße, Farbenzahl etc. beeinflussen. Unser erstes Bild zeigt, wie unterschiedliche Anzeigenarten ein einzelnes Byte interpretieren. Obwohl die Arbeit des Video-Chips durch diese Technik sehr vereinfacht wird, und auch das Mischen von Text und hochauflösender Grafik möglich ist. belegt diese Methode mit seinem vollgepackten Bildschirmspeicher jedoch einen großen RAM-Bereich. Eine Anzeige mit 160 mal 256 Pixel und 16 Farben braucht beispielsweise 20 KBytes.

Grafiken aus Zeichen

Als Speicherchips noch teuer waren, ließ sich diese Methode nicht selbstverständlich einsetzen. So entstand eine alternative Anzeigetechnik, die auf dem Commodore 64 heute noch Anwendung findet. Statt jedes Pixel mit seiner Farbinformation im Speicher zu halten, baut dieser Video Chip die Grafik aus einzelnen Zeichen auf. Dabei entspricht die Position eines Zeichens je einem Speicherbyte. Jedes Byte enthält einen Zeichencode, über den der Video Chip eine ROM-Tabelle mit der Bildschirmmatrix von acht mal acht Pixel abruft.



Die Videosteuerung des Acorn B oder der Schneider-Computer bringt alle Bildschirminformation (Pixel AN/AUS und Farbdaten) zusammen im Speicher unter. Beide Computer besitzen mehrere Anzeigearten, die die gespeicherten Bitmuster auf unterschiedliche Weise interpretieren. In einem 16-Farben-Modus sind vier Bits für die Speicherung des Farbcodes nötig. Ein Byte kann daher nur je zwei Pixel darstellen. Im Zwei- oder Vierfarbmodus werden nur ein (bzw. zwei) Bits für den Farbcode eines Pixels benötigt. In diesen Darstellungsarten kann ein Byte vier bzw. acht Pixel speichern.

Eine Zeichenanzeige mit 40 mal 25 Zeichen belegt hier weniger als ein KByte. Die Farbdaten dieses Systems sind in weiteren 1000 Bytes gespeichert, die den Farbcode für die entsprechenden Zeichenbytes liefern.

Auch das Anzeigesystem des Schneider CPC greift auf im ROM gespeicherte Zeichendefinitionen zu, spricht sie aber vom Betriebssystem aus an, wenn Zeichen zum Bildschirmspeicher geschickt werden sollen.

Bildschirmanzeigen, die mit der zweiten Methode arbeiten, haben normalerweise einen zweiten, hochauflösenden Modus, bei dem die Speicherbits den Pixeln des Bildes direkt entsprechen. Diese Art der Grafikaufbereitung wird „Bit-Mapping" genannt. Die Farbinformation ist in diesem Fall nicht wie im ersten System in den Anzeigebits abgelegt, sondern in einem separaten Speicherbereich untergebracht. Das heißt, jedes Byte des zweiten RAM-Bereichs von 1000 Bytes enthält die Farbdaten für ein Feld von acht mal acht Pixeln.

Auf dem Commodore 64 ist damit zwar hochauflösende Grafik möglich, doch lassen sich Text und Grafik nicht mischen – es sei denn, Sie entwickeln eigene Routinen, die die Zeichendefinitionen in die Bit-Map schreiben. Dieser Ansatz wurde interessanterweise vom Spectrum übernommen, der seinen Bildschirm als einfache Bit-Map speichert und mit einem separaten Farb-RAM die Farben der einzelnen Zeichenfelder (acht mal acht Pixel) steuert.

Auf dem Spectrum lassen sich Text und hochauflösende Grafik problemlos mischen, da das Betriebssystem die Zeichendefinitionen in die Bit-Map schreibt.

Auch die Sprite-Grafik wird vom Video-Chip gesteuert. Normalerweise werden Sprites als Bitmuster im RAM definiert und ihre Eigenschaften (Farbe, Größe etc.) über die internen Register des Video-Chips gesteuert. Da ein einziger Chip außer der normalen Grafik auch die Sprites darstellen muß, lassen sich Video-Chips wie der VIC des Commodore 64 auf die Erzeugung von Interrupts programmieren, wenn Sprites mit beispielsweise anderen Sprites zusammenstoßen.

Verschiedenes Scrollen

Das Scrollen des Bildschirms führen die beiden hier beschriebenen Video-Chips auf unterschiedliche Weise aus. Der zweite Chiptyp steuert das Scrollen per Software. Sein Bildschirmspeicher enthält nur wenig Bytes, die sich schnell mit Maschinencoderoutinen umstellen lassen. Beim ersten Chip mit seiner großen Zahl von Bildschirmbytes wäre das software-gesteuerte Scrollen viel zu langsam. Diese Chips haben daher ein Spezialregister für die Anfangsadresse des Bildschirmspeichers und steuern das Scrollen über eine Änderung dieses Registerinhalts. Bei umsichtigem Einsatz dieser Methode kann die Anzeige in alle vier Richtungen gescrollt werden.

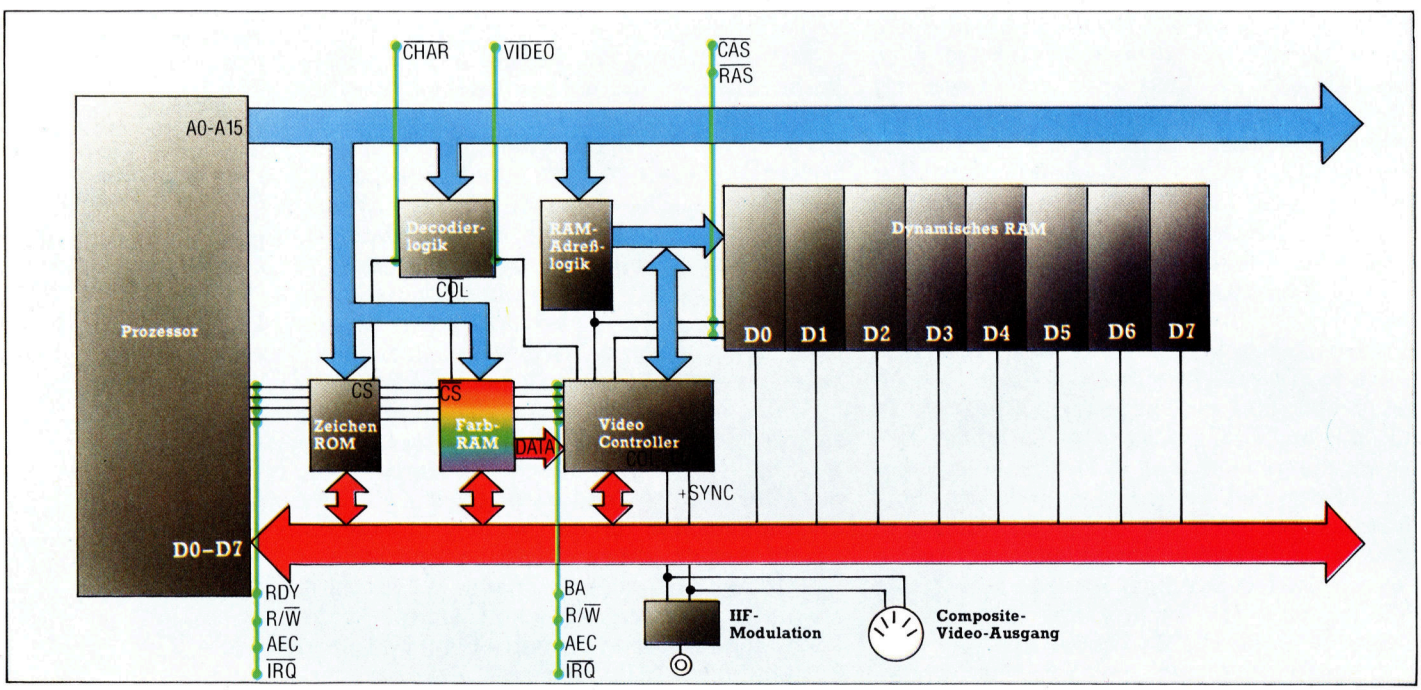
Der in den Acorn B eingebaute Video-Chip hat weit weniger zu tun, als beispielsweise der VIC-Chip des Commodore 64. Er kann seine Speicherzugriffe in die Phase des Taktzyklus verlegen, in dem der Prozessor den Speicher nicht anspricht. Damit verlangsamt der Video-Chip die Abläufe des Computers nicht. Außer einem Video-Chip besitzt der Acorn B aber noch eine ULA, die das gesamte System mit Zeitsignalen versorgt, die Beziehung zwischen logischen und physischen Farben berechnet und die RGB-Ausgabe liefert.

Im Gegensatz dazu sind mit dem VIC-Chip mehrere Speicherzugriffe nötig. Einige davon – beispielsweise die RAM-Auffrischung und das Abrufen von Zeichendaten haben keine Auswirkungen auf die Prozessorabläufe, da sie in Pausen der Systemuhr ausgeführt werden.

Immer im Bild

Das Bild zeigt den Anschluß des VIC-Chip an das System des Commodore 64. Der VIC ist direkt mit dem Datenbus verbunden und arbeitet mit den gleichen Adreßleitungen (im Multiplexverfahren) wie das dynamische RAM, das er ebenfalls über die Steuerleitungen CAS und RAS auffrischt. Einige Abläufe des Video-Chips lassen sich ohne Unterbrechung des Prozessors ausführen. Weiterhin stellt die Steuerung des Adreßbusses (AEC = Address Bus Enable Control) sicher, daß die Adreßbustreiber des Prozessors während der Phase der Systemuhr abgeschaltet sind, in der der Video-Chip den Speicher anspricht.

Der Zugriff auf die 1000 Byte des Bildschirmspeichers und die Spritebereiche muß jedoch öfter geschehen, als in jeder zweiten Phase. Es ist daher notwendig, den Prozessor auszuschalten, wenn der Video-Chip für den Abruf seiner Daten Speicherzyklen braucht. Das Ausschalten geschieht über eine Verbindung der VIC-Leitung BA (‚Bus Available' = Bus verfügbar) mit dem READY-Kontakt des Prozessors. Ihm stehen drei weitere Zyklen zur Verfügung, um einen bereits laufenden Speicherzugriff abzuschließen. Im vierten Taktzyklus des Prozessorzugriffs bleibt AEC jedoch auf „Low" und gibt dem VIC so die Möglichkeit, den Speicher anzusprechen.



Fachwörter von A bis Z

Optical Disc =
Optische Speicherplatte
Für dieses Speichermedium ist der Name ‚Laserplatte' geläufiger. Die digitale Information ist dabei in Form winziger Vertiefungen (Pits) auf konzentrischen oder spiraligen Spuren mit einem starken Laser in die spiegelnde Plattenoberfläche „eingebrannt". Bei der fotoelektrischen Abtastung mit einem schwachen Laser werden die Pits, die das Licht streuen, als Einsen interpretiert, die Reflexe der unversehrten Oberflächenbereiche dagegen als Nullen.

Laserplatten werden derzeit hauptsächlich als ‚Compact Disk' (CD) alternativ zur Schallplatte bzw. als Bildplatte für die Video-Wiedergabe eingesetzt. In Gestalt der ‚CD-ROMs' eröffnet diese Technologie aber auch im Computerbereich weitreichende Möglichkeiten. Eine einzige Compact Disk kann bis zu zwei Gigabyte speichern.

Für Laserplatten bieten sich im Computerbereich interessante Einsatzmöglichkeiten – nicht nur für die große Videoplatte zur Bildwiedergabe in Verbindung mit Computergrafik, sondern auch für die Compact Disk als ‚CD-ROM' zur Speicherung von Programmen. Der jetzige Stand der Technik gestattet allerdings noch nicht das Löschen und Beschreiben der Platten durch den Benutzer.

Die Sache hat einstweilen noch den Haken, daß sich die Platten vom Anwender weder löschen noch beschreiben lassen; sie sind daher vorerst nur für käufliche Software und nicht als Massenspeicher für den Schreib/Lese-Betrieb geeignet.

Bei Spielhallenautomaten werden Bildplatten schon seit längerer Zeit eingesetzt. Sie können dort beispielsweise Ihr Flugzeug (als Rechner-erzeugtes Sprite) durch eine ‚echte' Filmlandschaft von der Laserplatte steuern, die den Sprites als Hintergrund unterlegt wird. Das ergibt eine Wirklichkeitsnähe, die sonst nicht zu erreichen ist.

Optimisation = Optimierung
Die ‚Optimierung' eines Programms zielt auf die Lösung, die den gestellten Anforderungen am besten gerecht wird. Je nach Lage der Dinge kann die optimale Lösung sehr unterschiedlich aussehen – einmal soll die Geschwindigkeit und ein anderes Mal vielleicht die Speicherplatznutzung Vorrang haben.

Zur Optimierung gehört die Ausschaltung überflüssiger Umwege bei der Programmausführung. Ein gutes Beispiel dafür ist die Initialisierung von Variablen außerhalb von Schleifen oder häufig benutzten Unterprogrammen: Wenn einer Variablen einmal ein Wert zugewiesen ist, kostet es nur Zeit, die Vereinbarung immer wieder unnötig zu bestätigen, solange keine Veränderung eintritt.

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.

OR = ODER
Das OR als Boolesche Verknüpfung liefert dann den Ausgangswert ‚TRUE', wenn ein oder beide Eingänge auf ‚TRUE' liegen. Gebräuchlich ist dafür auch die Bezeichnung ‚inklusives' OR zur Unterscheidung vom ‚exklusiven' OR (EXOR), bei dem nur einer von den beiden Eingängen den Wert ‚TRUE' führen darf, wenn sich der Ausgangszustand ‚TRUE' ergeben soll.

Ordered Pair =
Geordnetes Zahlenpaar
‚Geordnet' heißt ein Zahlenpaar dann, wenn die Reihenfolge der Elemente fixiert ist. Bei der Angabe kartesischer Koordinaten in der Form (x,y) beispielsweise, wird stets die erste Zahl als horizontaler, die zweite als vertikaler Achsenabschnitt interpretiert, so daß die Zuordnung nicht unbedingt jedesmal neu spezifiziert werden muß.

Oscilloscope = Oszilloskop
Ein Oszilloskop (auch Oszillograf) dient zur Darstellung des zeitlichen Verlaufs von elektrischen Signalen, speziell von schnellen Schwingungsvorgängen. Die Kurvenform wird dabei mit einem Elektronenstrahl auf dem Leuchtschirm einer Kathodenstrahlröhre geschrieben. Der Strahl wird jedoch anders als beim zeilenweisen Aufbau eines Fernsehbildes durch die Signalspannung nicht helligkeitsmoduliert, sondern vertikal abgelenkt, während er mit einer wählbaren Geschwindigkeit von links nach rechts über den Schirm läuft. So ergibt sich eine Leuchtspur, die den Spannungsverlauf als Funktion der Zeit wiedergibt. Oszilloskope sind im Labor und beim Service ein unerläßliches Hilfsmittel, ob es nun um die Analyse eines Schallsignals oder die Synchronisation von Schaltimpulsen geht.

Bildnachweise
1992: Marcus Wilson-Smith
1993, 2016: Kevin Jones
1997, U3: Ian McKinnell
2000–2005: Garry Banks
2010: Caroline Clayton

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

computer kurs Heft 73

Attraktiver Ausbau

Die Grundausstattung des Acorn Electron ist, was die Schnittstellen betrifft, recht karg ausgefallen. Wirkungsvolle Abhilfe schafft das Erweiterungsmodul „Plus 3".

Bewegliche Typen

Die Sprache C läßt sich über anwenderdefinierte Datentypen noch verbessern und erweitern. Beispiel: Ein Bridgeprogramm.

Gesteuerte Treiber

Die Routinen des Interface des Spectrum lassen sich auch für Maschinenprogramme einsetzen.

Computer-Karrieren

In diesem Bericht geht es um die Hard- und Softwareingenieure.