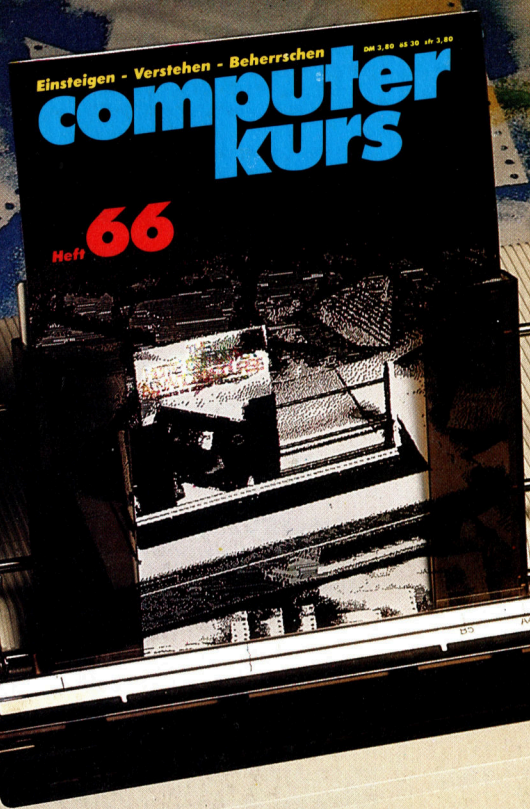
Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 66



EPSON SQ-2000

Toller Jet-Set-Druck auf Papier

Kalkulierte Glücksspiele

Der Arm wird fertig

Schnittstellen beim C 64

Ein wöchentliches Sammelwerk

computer kurs

Heft 66

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

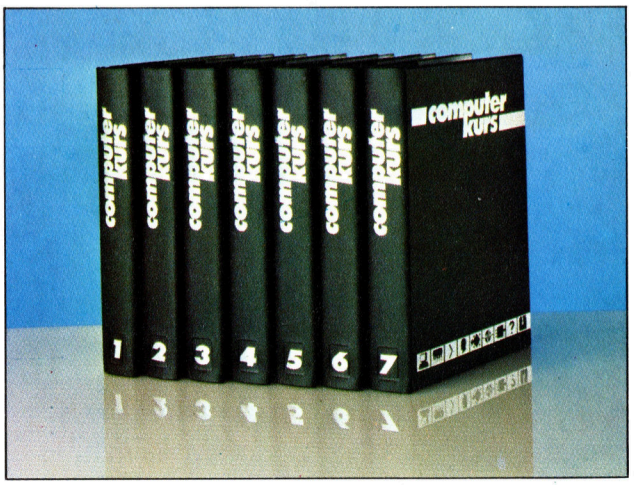
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

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

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



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

Go in Prozeduren

Beim Programmieren eines Spieles wie Go sollte man mit den E/A-Routinen beginnen. Unser Projekt startet mit der Ausarbeitung dieser Routinen auf dem Acorn B. Es werden Variablen initialisiert, und das komplette Spielbrett von Go wird auf dem Bildschirm dargestellt.

Go wird in vier Versionen erstellt – für den Acorn B, C64, Sinclair Spectrum und Schneider CPC 464/664. Die Zeilennummern stimmen, wo immer möglich, bei den einzelnen Versionen überein. Manchmal mußten aber zusätzliche Programmzeilen zur Implementierung einzelner Routinen auf die jeweilige Maschine eingefügt werden.

So muß ein User-Stack eingearbeitet werden, um die Listings möglichst ähnlich aufgebaut zu lassen. Wenn sich Text auf Routinen und Variablen im Listing bezieht, gilt dieser für die Acorn-B-Version; es wurde jedoch eine möglichst genaue Übereinstimmung mit den anderen drei Listings angestrebt.

Unser erster Programmabschnitt behandelt die Initialisierung von Variablen, erstellt das Spielbrett und enthält die Eingabe-Routinen. Die Zeilen 10 bis 140 bilden den Hauptteil des Programms. Vor Eingabe der eigentlichen Spiel-Schleife (Zeile 60 bis 90) werden die Routinen PROCinitialise und PROCintroduction aufgerufen.

PROCinitialise wird nur beim ersten Programmlauf zur DIMensionierung des Bretts (board%), Initialisierung des Cursors usw. benutzt. Das Brett wird als Reihe aus 255 Bytes dimensioniert, die eine Spielfläche von 15 mal 15 Feldern bilden (nicht 19 mal 19 Felder, wie normalerweise bei Go üblich).

Nützliche Konstanten

Dafür gibt es mehrere Gründe. Einer davon ist das Tempo. Ein Spielprogramm in BASIC kann erwartungsgemäß nicht sehr schnell sein; durch das verkleinerte Spielfeld wird die Ausführungszeit jedoch auf ein Drittel verkürzt.

Die in Zeile 190 und 200 initialisierten Variablen sind Konstanten. Wenn aber die Variablennamen anstelle der aktuellen Zahlen verwendet werden, kann eine Modifikation erheblich einfacher vorgenommen werden. Wollen Sie etwa mit zwei Personen oder gegen den Computer spielen, müssen nur die Werte von black% und white% geändert werden.

Normalerweise wird Go auf einem Holzspielbrett mit einer Rastereinteilung von 19 mal 19 überschneidenden Linien gespielt. In der Computerversion wurde die Brettgröße auf ein Raster von 15 mal 15 reduziert, um eine problemlose Darstellung des Spielfeldes auf dem Bildschirm zu ermöglichen.

PROCintroduction wird zu Beginn jedes Spiels verwendet, um den Bewegungszähler (move%), das Ende der Kondition (end%) usw. zu setzen. Dabei werden die Routinen PROCgame_init und PROCtitle_screen benutzt. Der Titelbildschirm ist ziemlich einfach gehalten. Sie können ihn aber beliebig ändern. Sie könnten z. B. das japanische Schriftzeichen für Go und fernöstliche Musik einarbeiten. Dafür stehen die Zeilen ab 5000 zur Verfügung.

PROCprint_board zeichnet das Spielbrett. FNinit_to_char wird von dieser Routine für die Konvertierung einer Integer-Brettkoordinate in Zeichen-Koordinaten für die Bildschirmdarstellung verwendet; Definition in Zeile 2260.

FNinput und PROCmessage sind zwei allgemein verwendbare Routinen zur Ein- und Ausgabe von Meldungen, basierend auf dem mess$-Array, das in PROCread_messages ini-

tialisiert wurde. Dieses Array enthält zehn Hinweise oder Meldungen, die während des Spiels ausgegeben werden.

Ein Problem beim Eintippen des Listings für den Spectrum tritt in den Zeilen 1480 und 1505 auf. Die unterstrichenen Zahlen in diesen Zeilen beziehen sich auf die Symbol-Grafiken des Spectrum. Drücken Sie Caps Shift zusammen mit 9, um in den Grafikmodus zu kommen. Jetzt geben Sie die angegebene Zahl ein. Zum Verlassen des Grafikmodus drücken Sie erneut Caps Shift und 9. Wird der angegebenen Zahl ein sh vorangestellt, müssen Sie auch hier gleichzeitig die Shift-Taste gedrückt halten.

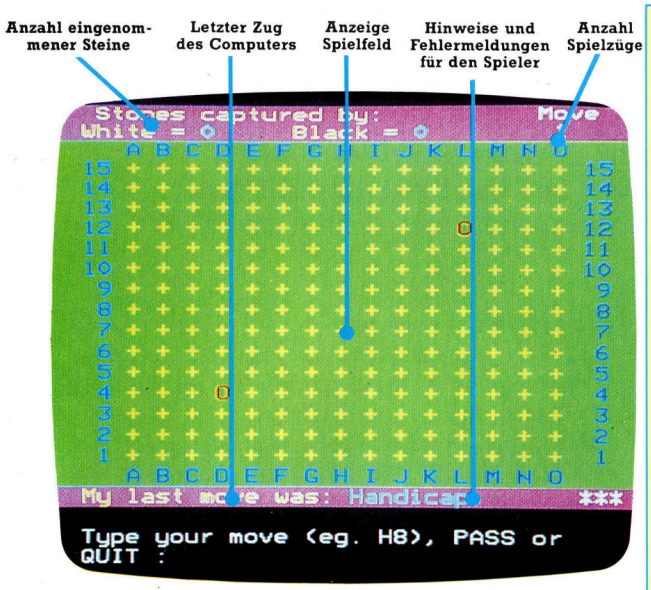
Variablenliste

Variable	Verwendung
black%	Wert 1:schwarzer Stein im Brett-Byte.
board%	Startwert für das 256-Byte-Brett im Speicher.
capture%(2)	Enthält die Zahl der von Schwarz und Weiß eingenommenen Steine.
colour%	Wert 3. Dient zum Setzen der Farb-Bits in einem Brett-Byte.
dir%(4)	Enthält die nötigen Offsets für Bewegungen um ein Feld nach Norden, Süden, Westen oder Osten.
liberty%	Wert 8. Wird in Gruppen-Suchroutinen zur Kennzeichnung zuvor gezählter Freifelder verwendet.
marker%	Wert 4. Wird in Gruppen-Suchroutinen zur Kennzeichnung zuvor gezählter Steine verwendet.
move%	Enthält die Zahl der Züge in jedem Spiel.
white%	Wert 2: weißer Stein im Brett-Byte.
atari1$	Enthält entweder 5 Leerzeichen oder „Atari“, wenn der letzte Zug des Computers diese Situation bewirkte.
atari2$	Enthält entweder 5 Leerzeichen oder „Atari“, wenn der letzte Zug des Spielers diese Situation bewirkte. Anmerkung: Atari bedeutet das Setzen eines Steines in der Form, daß eine oder mehrere gegnerische Gruppen mit nur einem Freifeld auf dem Brett verbleiben.
axis$	Teil der Spielbrett-Anzeige.
mess$	Enthält alle allgemeinen E/A-Meldungen. Wird entweder von PROCinput oder PROCmessage verwendet.

Erstes Modul

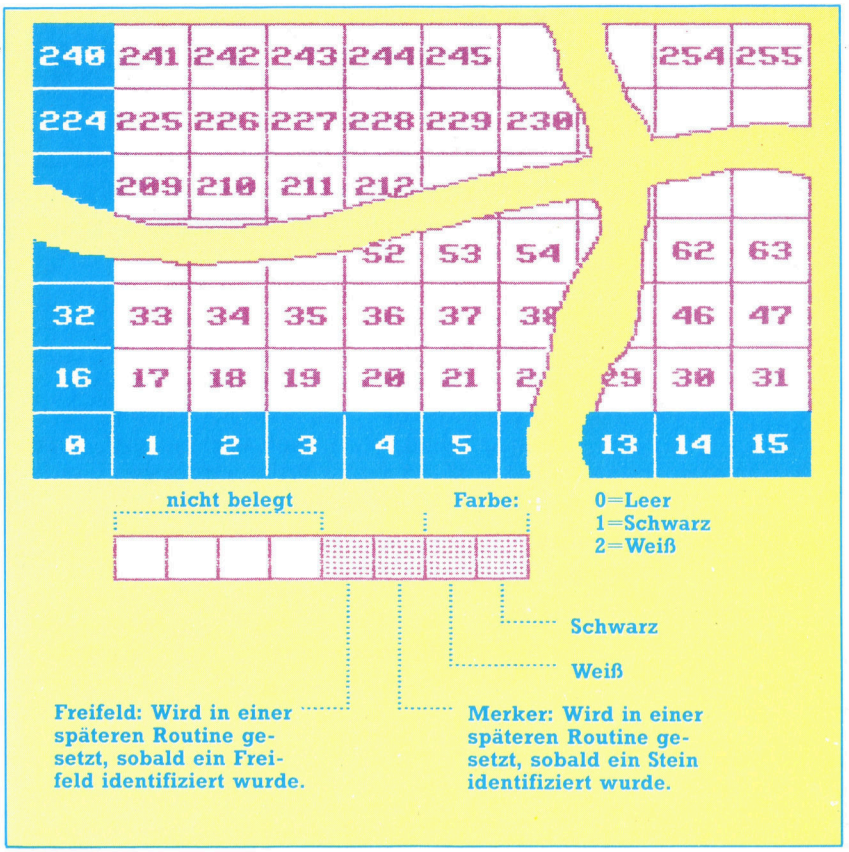
Acorn B:

```
   10 MODE 7
   20 *FX14,6
   30 PROCinitialise
   40 PROCintroduction
   50 CLS:PROCprint_board
   60 move%=move%+1:REM white moves here
   70 IF end% GOTO 100
   80 move%=move%+1:REM black moves here
   90 IF NOT end% GOTO 60
  100 ans$=FNinput(21,9,1)
  110 IF ans$="Y" GOTO 40 ELSE IF ans$<>
"N" GOTO 100
  120 PROCmessage(22,5,"")
  130 PRINT:END
  140 :
  150 REM*******************************
  160 :
  170 DEF PROCinitialise
  180 LOCAL L%
  190 black%=1 : white%=2 : colour%=3
  200 marker%=4 : liberty%=8
  210 @%=2
  220 DIM board% 255
  230 VDU 23;8202;0;0;0;
  240 DIM capture%(2)
  250 axis$=CHR$130+CHR$157+CHR$132+"
A B C D E F G H I J K L M N O"
  260 PROCread_messages
  290 DIM dir%(4)
  300 RESTORE 340
  310 FOR L%=1 TO 4
  320   READ dir%(L%)
  330   NEXT
  340 DATA 16,1,-16,-1
  350 ENDPROC
  360 :
  370 REM*******************************
  380 :
  390 DEF PROCread_messages
  400 LOCAL M%
  410 RESTORE 460
  420 DIM mess$(9)
  430 FOR M% = 0 TO 9
  440   READ mess$(M%)
  450   NEXT
  460 DATA "O.K. THINKING..."
  470 DATA "Illegal entry: "
  480 DATA "Stone already on point: "
  490 DATA "Illegal. Ko on point: "
  500 DATA "Illegal. Suicide on point: "
  510 DATA "    O.K. GAME OVER."
  520 DATA ""
  530 DATA "  How many handicap stones m
ay I          have (2-9) ?"
  540 DATA "Type your move (eg. H8), PAS
S or        QUIT :"
  550 DATA "Do you want to play again (Y
/N)?"
  560 ENDPROC
  570 :
  580 REM*******************************
 1260 :
 1270 DEF PROCintroduction
 1280 PROCgame_init
 1290 PROCtitle_screen
 1300 ENDPROC
 1310 :
 1320 REM*******************************
 1330 :
 1340 DEF PROCgame_init
 1360 atari1$="     ":atari2$="     "
 1370 location%=0:move%=1
 1380 end%=FALSE
 1390 capture%(1)=0:capture%(2)=0
 1410 ENDPROC
 1420 :
 1430 REM*******************************
 1440 :
 1450 DEF PROCtitle_screen
 1460 CLS
 1470 PRINT TAB(12,4);CHR$145;CHR$154;"h
7+$  h7k4"
 1480 PRINT TAB(12,5);CHR$145;CHR$154;"j
5k5  j5j5"
 1490 PRINT TAB(12,6);CHR$145;CHR$154;"p
ssp  pssp"
 1500 PRINT TAB(9,10);CHR$134;"by  Marcu
s  Jeffery"
```



Meldungen während des Spieles

Zusätzlich zur Darstellung des Spielfeldes erhält der Spieler Hinweise über die Zahl eingenommener Steine, die Anzahl der bisherigen Spielzüge und den Status des zuletzt erfolgten Spielzuges. Die untere Bildschirmhälfte enthält Meldungen und Fehleranzeigen. Bei der Programmentwicklung werden wir sehen, wie das Programm Fehler entdeckt und unerlaubte Züge (wie z. B. „Selbstmord" oder das Setzen von Steinen auf bereits besetzte Felder) unterbindet.




```
 1510 PRINT TAB(1,13);CHR$133;"You will
play the";
 1520 PRINT CHR$135;"white";CHR$133;"sto
nes,  and"
 1530 PRINT CHR$133;" the  computer  (be
ing weaker!)  will"
 1540 PRINT CHR$133;" have the";CHR$129;
"red";CHR$133;"stones with a  handicap"
 1550 PRINT TAB(13);CHR$133;"advantage."
 1560 hand%=VAL(FNinput(20,7,1))
 1570 IF hand%<2 OR hand%>9 THEN 1560
 1590 ENDPROC
 1600 :
 1610 REM*******************************
 1720 :
 1730 DEF PROCprint_board
 1740 LOCAL P%,X%,Y%
 1750 PRINT TAB(0,0);CHR$133;CHR$157;CHR
$131;" Stones captured by:";
 1760 PRINT TAB(32,0);CHR$135;"Move"
 1770 PRINT TAB(0,1);CHR$133;CHR$157;CHR
$131;"White =";CHR$134;capture%(2);
 1780 PRINT TAB(16,1);CHR$131;"Black =";
CHR$134;capture%(1);
 1790 PRINT TAB(33,1);CHR$135;move%
 1800 PRINT TAB(0,2);axis$
 1810 FOR Y%=15 TO 1 STEP -1
 1820   PRINT TAB(0,18-Y%);CHR$130;CHR$1
57;CHR$132;RIGHT$(" "+STR$(Y%),2);
 1830   FOR X%=1 TO 15
 1840     P%=board%?(16*Y%+X%)
 1850     IF P%=1 THEN PRINT CHR$129;"O"
;:GOTO 1880
 1860     IF P%=2 THEN PRINT CHR$135;"O"
;:GOTO 1880
 1870     PRINT CHR$131;"+";
 1880     NEXT
 1890   PRINT TAB(35,18-Y%);CHR$132,Y%
 1900   NEXT
 1910 PRINT TAB(0,18);axis$
 1920 PRINT TAB(0,19);CHR$133;CHR$157;CH
R$131;"My last move was:";
 1930 PRINT CHR$134;FNint_to_char(locati
on%);TAB(30);atari1$
 1940 PRINT TAB(16,22);atari2$;CHR$7
 1950 ENDPROC
 1960 :
 1970 REM*******************************
 1980 :
 1990 DEF FNinput(P%,M%,W%)
 2000 LOCAL S%,A$,I$
 2010 PRINT TAB(39,P%+1);STRING$(119,CHR
$127);
 2020 A$=""
 2030 PRINT TAB(3,P%);mess$(M%);CHR$134;
 2040 PRINT STRING$(W%,CHR$32);STRING$(W
%,CHR$127);
 2050 *FX21,0
 2060 I$=GET$ : IF ASC(I$)=13 THEN GOTO
2120
 2070 IF ASC(I$)<>127 GOTO 2090
 2080 IF S%>0 THEN S%=S%-1:A$=LEFT$(A$,S
%):GOTO 2110 ELSE GOTO 2060
 2090 IF I$>="a" AND I$<="z" THEN I$=CHR
$(ASC(I$)-32)
 2100 IF S%<W% THEN S%=S%+1:A$=A$+I$ ELS
E GOTO 2060
 2110 PRINT I$; : GOTO 2060
 2120 =A$
 2130 :
 2140 REM*******************************
 2150 :
 2160 DEF PROCmessage(P%,M%,A$)
 2170 LOCAL L%
 2180 FOR L%=P% TO P%+1
 2190   PRINT TAB(39,L%);STRING$(39,CHR$
127);
 2200   PRINT CHR$141;CHR$136;CHR$129;me
ss$(M%);A$;
 2210   NEXT
 2220 ENDPROC
 2230 :
 2240 REM*******************************
 2250 :
 2260 DEF FNint_to_char(P%)
 2270 IF P%=0 THEN ="Handicap"
 2280 =CHR$(P% MOD 16 + 64)+STR$(P% DIV
16)+"  "
 2290 :
 2300 REM*******************************
```

Jede Position auf dem Brett wird durch ein einzelnes Byte im Speicher repräsentiert. Das obenstehende Diagramm zeigt den Aufbau der 255 Bytes für die 255 Schnittstellen auf dem 15 mal 15 großen Brett. Bits innerhalb jedes Bytes enthalten Informationen über den Status des korrespondierenden Schnittpunkts. Die zwei niederwertigen Bits stellen die Farbe des Steines auf dem Schnittpunkt fest (oder zeigen an, daß der Schnittpunkt frei ist). Bit 2 und 3 werden von „Spielstatus-Routinen" verwendet, die wir später ausarbeiten werden.

Vom Plan zur Tat

Wir beschließen unseren zweiteiligen Bericht über Programmgeneratoren mit einem Blick auf zwei der populärsten gegenwärtig verfügbaren Systeme: Sycero und The Last One (TLO).

Software, die auch dem Laien eine eigenständige Programmentwicklung erlauben soll, muß hohen Anforderungen genügen:

- Sie sollte das „von oben nach unten"-Verfahren unterstützen. Dabei skizziert der Anwender sein Projekt zu Anfang nur grob und fügt Details erst im weiteren Verlauf ein.
- Sie sollte menügesteuert sein und Anwender mit verständlichen Anweisungen an verfügbare Möglichkeiten heranführen.
- Sie sollte Fehler unmittelbar erkennen und einfache Möglichkeiten zur Korrektur und Ergänzung bieten.
- Das erzeugte Programm sollte „transportabel" sein, also wenn irgend möglich auch auf anderen Computern laufen.
- Der Prozeß der Programmentwicklung sollte so gut dokumentiert sein, daß auch ein anderer Benutzer später Ergänzungen oder Verbesserungen vornehmen kann.

Nun ist ein Programmgenerator nicht nur ein Werkzeug für Anfänger. Ein zu niedriges Niveau kann Fortgeschrittene nicht nur zur Verzweiflung treiben, es kann auch durch überflüssige Vorsichtsmaßnahmen außerordentlich zeitraubend werden. Darüber hinaus besitzt ein erfahrener Anwender seinen eigenen Programmierstil, den ein Programmgenerator akzeptieren sollte. In allen aufgezählten Punkten bieten die beiden Systeme, TLO und der neuere Sycero, verblüffend gute Leistungen.

Informative Menüs werden von beiden durchgehend zur Verfügung gestellt. Sycero bietet Hilfsfenster für Cursorsteuerung und Grafik; Editorbefehle sind durch Drücken der Control- und H-Tasten jederzeit zugänglich. Bei dem TLO-System wird jedoch das „von oben nach unten"-Konzept besser realisiert. Es beginnt mit der Erstellung eines „Flußdiagramms", das aus REM-Zeilen besteht.

Ein- und Ausgabefenster werden während des Codierungsprozesses definiert; der erzeugte Code selbst ist in ASCII. Der Anwender muß den Programmgenerator verlassen, den Code laden und im binären Format speichern. Bei Sycero müssen die benötigte Dateidefinition und die Gestaltung des Bildschirms vorher festgelegt werden. Darauf folgt die Strukturierung der Eingabevariablen (also auch die Definition der Kommentare und Fehlermeldungen zur Bedienerführung), die Festlegung von Ausgabeformaten und ähnliches mehr. Die Einzelmodule werden unmittelbar vor der Codierung miteinander verbunden.

Dieser Ansatz bereitet dem Ungeübten Schwierigkeiten, hat aber den Vorteil, daß vor der Codierung eine sorgfältige Planung durchzuführen ist. Wird während der Codierung ein Fehler entdeckt, unterbricht der Generator die Ausführung, gibt eine Fehlermeldung aus und ermöglicht eine Berichtigung. Nach dem Codieren wird das Programm automatisch binär gespeichert.

Beide Generatoren erzeugen vollständig transportablen Code. Tatsächlich sollte der ASCII-Code von TLO auf jedem MSX-Rechner laufen, soweit es keine Probleme mit dem verfügbaren Speicherplatz gibt. Da aber beide Generatoren umfangreiche und ziemlich komplizierte Fehlersuchroutinen in die Programme integrieren, werden auch kürzere Programme schnell umfangreich.

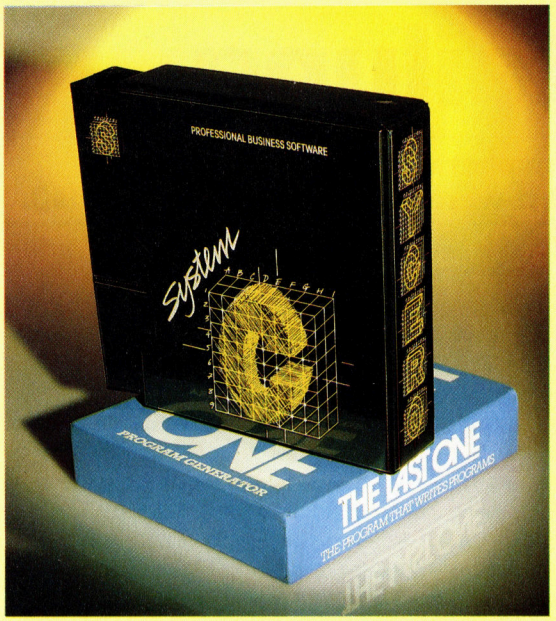
Gute Dokumentation

Wie bei MS-DOS üblich, müssen sowohl TLO als auch Sycero mit einem Installationsprogramm an die Hardware angepaßt werden. Unterstützt werden Doppellaufwerke und Hard-Disks. Bei TLO muß man außerdem eine Arbeitsdiskette anlegen, ein Vorgang, der dem Formatieren sehr ähnlich ist. Man hat dabei die Wahl zwischen einer Floppy oder einem Subdirectory auf der Hard-Disk. Die Zuweisung ist wesentlich, weil das Programm bei der Konstruktion des Flußdiagramms ständig auf die Arbeitsdiskette zugreifen muß. Ist das betreffende Flußdiagramm nicht verfügbar, muß man entweder von vorne anfangen oder den erzeugten BASIC-Text selbst ändern. Wie üblich sollte man sich gegen derlei Unglücksfälle durch Sicherheitskopien schützen.

Sowohl TLO als auch Sycero erzeugen eine umfangreiche Dokumentation, in der Bildschirmaufbau, die benutzten Variablen und weitere Informationen festgehalten werden. Sycero fügt noch Datum und Uhrzeit auf dem Bildschirm und im Listing hinzu, so daß alte und neuere Versionen leicht auseinanderzuhalten sind.

Nach der Installation zeigt TLO ein Hauptmenü mit acht Optionen:

Programm erstellen	Hilfsfunktionen
Programm ändern	Arbeitsdisk einlegen
Datei ändern	Codierung fortsetzen
Datei anlegen	Zurück zu BASIC



The Last One (TLO) und Sycero helfen dem Benutzer bei der Erstellung von BASIC-Programmen, die unabhängig vom benutzten Computertyp laufen. Die Wünsche des Anwenders müssen aber in einer sorgfältigen Vorausplanung präzise definiert werden, ehe eins der Programme sinnvoll genutzt werden kann.

Wählt man die erste Option, so wird man gefragt: Benötigt Ihr Programm Dateien? Wenn ja, müssen die nötigen Dateien spezifiziert werden. Danach erscheint das Menü zur Erstellung von Flußdiagrammen. Bei der neuesten Version hat man 20 Optionen, die vom offensichtlich abergläubigen Autor von 1 bis 12 und 14 bis 21 numeriert wurden:

Flußdiagramm anzeigen
Flußdiagramm ändern
Programm codieren
Flußdiagramme verknüpfen
Ende
Tastatureingabe
Daten anzeigen
Sprünge
Berechnungen
Spezielle Funktionen
Löschen
Dateizeiger setzen
Datei lesen
Datei schreiben
Datei suchen oder sortieren
Datei verknüpfen
Datensatz prüfen
Datei löschen
Databank-Funktionen
Mehrfach-Funktionen

Mit diesen Optionen kann man ein Flußdiagramm erzeugen, das im Falle einer Namens- und Adreß-Datei etwa so aussieht:

1. Verzweigung auf ein Menü mit drei Optionen
2. Zeiger auf Ende Adreßdatei setzen
3. Tastatureingabe für Adreßdatei
4. Daten in Adreßdatei schreiben
5. Fragen <Fertig?>. Verzweigung, falls ‚Nein'
6. Unbedingter Sprung
7. Verzweigung zum Menü
8. Zeiger auf Anfang Adreßdatei setzen
9. Suche Adreßdatei
10. Daten aus Adreßdatei ausgeben
11. Fragen <Weiter?>. Verzweigung, falls „Ja"
12. Unbedingter Sprung
13. Adreßdatei sortieren
14. Zeiger auf Anfang Adreßdatei setzen
15. Daten aus Adreßdatei lesen
16. Daten aus Adreßdatei ausgeben
17. Unbedingter Sprung
18. Ende

Jetzt kann codiert werden. Bis zu diesem Zeitpunkt durfte man das Programm nicht verlassen, ohne alles bisher Geschriebene zu verlieren. Erst nach Beginn der Codierung kann man unterbrechen und mit dem Befehl „Codierung fortsetzen" weitermachen. Während der Codierung werden auch die Sprungadressen festgelegt – das erste Menü verzweigt beispielsweise nach 2 (Daten schreiben), 7 (Daten lesen) oder 18 (Ende) – sowie Bildschirmanzeigen aufgebaut.

Vielseitigkeit geboten

Die Anzeigen können abgespeichert werden. Das ist nützlich, weil sie somit auch in anderen Programmen weiterverwendet und modifiziert werden können. Bedauerlicherweise weisen weder das Programm noch das Handbuch auf diese Möglichkeit hin. Bei Sycero erfolgt die Speicherung automatisch.

Syceros Hauptmenü bietet 13 Optionen, die der User etwa in der gleichen Reihenfolge benötigt, in der sie auch angezeigt werden:

System anpassen
Initialisierung
System-Dateifeld definieren
Bildschirm aufbauen
Bildschirm verarbeiten
Bericht definieren
Bericht verarbeiten
Programm definieren
Programm erzeugen
„Aktive" Datei erzeugen
Erzeugtes Programm starten
Hilfsfunktionen
Ende

Ist man bis zur Option „Programm erzeugen" vorgedrungen, ist die meiste Arbeit getan. Der vergleichsweise einfache nächste Schritt erfordert wenig Arbeit, falls die Software keinen Fehler entdeckt.

Der Hauptunterschied zwischen beiden Programmgeneratoren liegt in ihrer Vorgehensweise. Das Sycero-Handbuch sagt das ganz deutlich: „Der beste Weg, ein Programm zu entwickeln, besteht im Vorgehen ‚von oben nach unten'. Fangen Sie mit den groben Umrissen an, und fügen Sie danach Details ein."

TLO, das mit dem Entwurf eines Flußdiagramms beginnt, ist der klassischen Methode näher. Paradoxerweise veranlaßt aber gerade das den Benutzer oft, ohne die nötige Vorausplanung zu arbeiten. Im Gegensatz dazu ist Sycero ohne Vorarbeit kaum einzusetzen.

Sowohl Sycero als auch TLO geben dem Benutzer ein bequemes Werkzeug zur Erstellung von BASIC-Programmen in die Hand. Trotz ihrer Mängel sind sie nützlich bei der Konstruktion von Hilfsprogrammen für Datenbanken und Rechenprogramme mit einfachen Fragen und Antworten. Dabei ist der entstehende Code natürlich umfangreicher als bei Programmen, die direkt in BASIC erstellt worden sind.

Befehlsabteilung

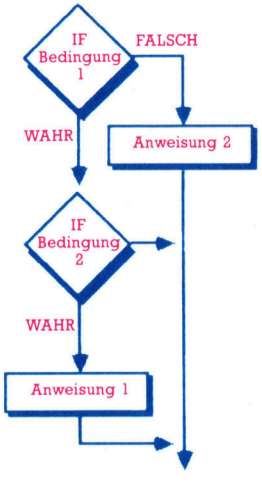
Wir untersuchen den vierten Programmbereich COBOLs, in dem Berechnungen und Abläufe ausgeführt werden: den Verarbeitungsteil.

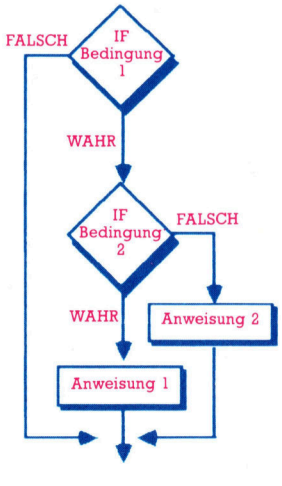
Der Verarbeitungsteil eines COBOL-Programms enthält die eigentlichen Programmabläufe. Sein Aufbau und die darin eingesetzten Mechanismen und Befehle sollen das Programm so eng wie möglich an das Umgangsenglisch anlehnen. Jeder Bereich dieses Teils besteht aus Paragraphen, die wiederum aus Sätzen aufgebaut sind und mit einem Punkt enden. Im allgemeinen entspricht ein COBOL-Satz einer BASIC-Anweisung. Sätze lassen sich in Bedingungen aufteilen, die wiederum andere Bedingungen genauer bestimmen können.

Das wichtigste Wort einer Bedingung oder eines Satzes ist das „Verb". Das am häufigsten eingesetzte Verb ist MOVE. Es überträgt Daten von einem Bereich in einen anderen und hat folgendes Format:

```
MOVE Name-1 TO Name-2.
```

In COBOL sind selbst einfache Abläufe recht umständlich. So gibt es zwei Arten von MOVE: Das alphanumerische MOVE überträgt von links Zeichen für Zeichen und fügt (falls nötig) Leerzeichen ein oder löscht sie. Das numerische MOVE positioniert das Dezimalkomma und führt die programmierten Änderungen durch. Je nach Formatanweisung (PICTURE) schneidet es Nullen ab oder fügt sie ein.





Probleme mit IF

Wenn in COBOL eine IF-Anweisung mit einem Punkt abgeschlossen wird, können Probleme entstehen. Die Befehlsfolge

```
IF Bedingung1 (dann)
   IF Bedingung2 (dann)
      Anweisung1
ELSE Anweisung2.
```

sollte wie in dem Ablaufdiagramm gezeigt ausgeführt werden. Wenn jedoch hinter beiden Anweisungen Punkte als Endkennung stehen, compiliert COBOL die Befehlsfolge völlig anders (siehe zweites Diagramm).

Das Problem ist umgangen, wenn die interne Verzweigung über einen separaten Paragraphen verfügt, der mit PERFORM ausgeführt wird.

```
IF Bedingung1 (dann)
   PERFORM Paragraph1
   ELSE Anweisung2
...
Paragraph1
   IF Bedingung2
      Anweisung1
```

Interessant sind MOVEs zwischen alphanumerischen und numerischen Datenelementen. Dabei sind einige MOVEs nicht erlaubt, speziell die mit besonderen PICTURE-Anweisungen und alphanumerische MOVEs. Es gibt zwar detaillierte Angaben über die Abläufe jedes einzelnen MOVEs doch sind die Einzelheiten sehr kompliziert. Für den Augenblick genügt es zu wissen, daß eine Anweisung, die wichtige Stellen ohne Fehlermeldung abschneiden könnte, mit Vorsicht zu genießen ist.

Mathematische Abläufe sind nicht die Stärke von COBOL, wenn auch einige Versionen eine 18stellige Rechengenauigkeit als Standard bieten. Zunächst gibt es die mathematischen Verben ADD, SUBTRACT, MULTIPLY und DIVIDE.

```
ADD Zahl-1 TO Zahl-2.
```

addiert beispielsweise den Inhalt von Zahl-1 zum Inhalt von Zahl-2, während

```
ADD Zahl-1 TO Zahl-2 GIVING Zahl-3.
```

die Summe von Zahl-1 und Zahl-2 in Zahl-3 ablegt. Der gleiche Vorgang läßt sich auch mehrfach ausführen:

```
ADD 5 TO Zahl-1, Zahl-2.
```

addiert beispielsweise 5 zu den aktuellen Werten von Zahl-1 und Zahl-2.

Die anderen mathematischen Verben haben das gleiche Format:

```
SUBTRACT Zahl-1 FROM Zahl-2
[GIVING Zahl-3].
MULTIPLY Zahl-1 BY Zahl-2 [GIVING Zahl-3].
DIVIDE Zahl-1 INTO Zahl-2
[GIVING Zahl-3]
[REMAINDER Zahl-4].
```

Beachten Sie, daß sich bei allen Beispielen der endgültige Wert in dem zuletzt aufgeführten Datenelement befindet.

Wie bei MOVE entstehen Probleme, wenn das Ergebnis eines mathematischen Vorgangs für den dafür reservierten Platz zu groß ist. Dabei werden zunächst die unwichtigen Stellen abgeschnitten, wenn nicht hinter dem Namen des Datenelementes das Wort ROUNDED steht. Das Abschneiden wichtiger Ziffernstellen ist ein Fehler, der von COBOL nicht verhindert wird. Es gibt jedoch die Möglichkeit, die Bedingung „ON SIZE ERROR" in mathematische Abläufe einzusetzen. Die Bedingung ist ohne Wirkung, wenn wichtige Stellen abge-

schnitten werden. Tritt jedoch ein Überlauf auf, bleibt der Wert intakt, und der in der Bedingung angegebene Ablauf wird ausgeführt.

Sie können sich vielleicht schon vorstellen, daß jede nur etwas kompliziertere Berechnung in COBOL sehr umfangreich wird. Das Verb COMPUTE kann diese Abläufe etwas vereinfachen, da es einen mathematischen Ausdruck einleiten kann, der einem BASIC- oder FORTRAN-Befehl entspricht:

```
COMPUTE Zahl-1 = (Zahl-2 + 3) * Zahl-3.
```

Einige Compiler erlauben sogar die Potenzierung, doch ist das nicht der Standard.

Auch bei COMPUTE sind Probleme möglich. Überläufe können hier nicht nur im Endergebnis, sondern bei jedem Zwischenwert auftreten. „ON SIZE ERROR" kann den Fehler zwar abfangen, teilt Ihnen jedoch nicht mit, wo der Überlauf aufgetaucht ist. Aus diesem und anderen Gründen verzichten viele COBOL-Puristen auf COMPUTE und arbeiten nur mit den normalen arithmetischen Verben, die zwar umständlicher, aber dafür sicherer sind.

Paragraphendschungel

Der Verarbeitungsteil kann eine beliebige Anzahl von Paragraphen enthalten, die jeweils einen eigenen Namen haben. Der Name steht im Bereich A von Spalte 7 an, und die zugehörigen Anweisungen befinden sich in dem Bereich B ab Spalte 12. Ein Paragraph entspricht etwa einer BASIC-Subroutine. Er ist jedoch normalerweise kein separates Modul oder Prozedur wie bei PASCAL und verfügt auch nicht über lokale Variable oder Parameter. Der Code wird üblicherweise von Anfang bis Ende ausgeführt und jeder Paragraph auch in dieser Reihenfolge bearbeitet.

COBOL kann Befehlsfolgen auf verschiedene Weise verändern. Dafür eignet sich besonders das Verb PERFORM mit seinen vielen Variationen. Hier zwei Beispiele:

```
PERFORM Paragraph-1.
```

funktioniert wie der Aufruf einer Prozedur oder eines GOSUB. Die Anweisungen im Inneren des Paragraphen werden ausgeführt, und der darauffolgende Befehl wird angesprochen.

```
PERFORM Paragraph-1 THRU Paragraph-n.
```

führt eine Reihe von Paragraphen aus, bevor es die Steuerung zurückgibt.

Hier eine weitere Steuermöglichkeit:

```
GOTO Paragraph-1.
```

Die Anweisung funktioniert zwar, sollte aber in COBOL (und auch in jeder anderen Sprache) möglichst vermieden werden.

Da die Sprache hauptsächlich im kommerziellen Bereich eingesetzt wird, bietet COBOL viele Lese- und Schreibvorgänge für Dateien, die auf Disketten oder Cassetten gespeichert sind. Das Standard-Cobol hat jedoch kaum Möglichkeiten für die interaktive Ein- und Ausgabe per Tastatur bzw. Bildschirm:

```
ACCEPT Datenbezeichnung.
```

steuert die Tastatureingabe und

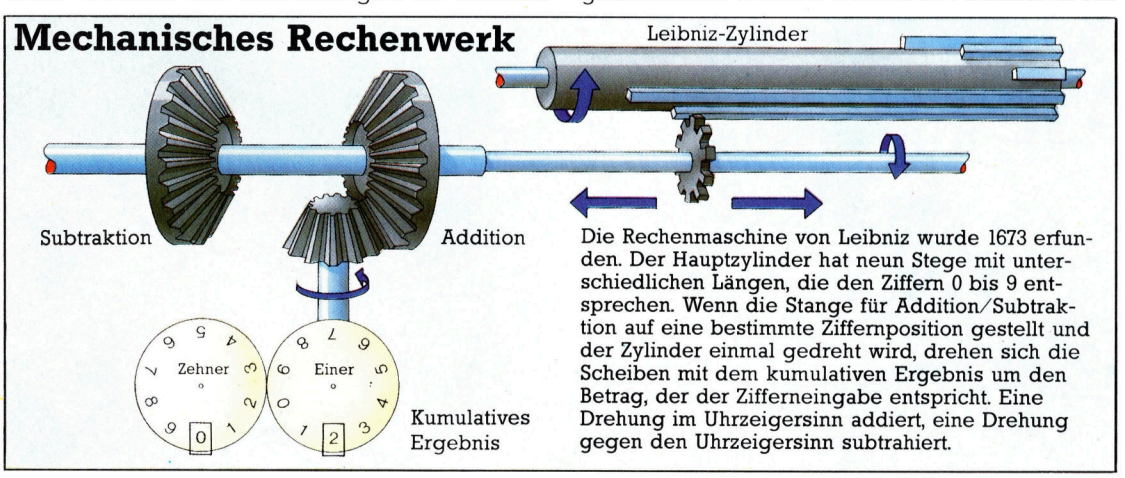
```
DISPLAY Datenbezeichnung.
```

die Bildschirmanzeige. Die Anweisungen arbeiten (wie die BASIC-Befehle INPUT und PRINT) im normalen Modus. Die meisten Microversionen von COBOL haben jedoch erweiterte E/A-Verben, mit denen die Bildschirmsteuerung möglich ist.

Die wichtigsten Eigenschaften von COBOL – wie auch jeder anderen modernen Programmiersprache – sind die Steuerstrukturen, besonders die Möglichkeiten für Verzweigung und Wiederholung. COBOL besitzt eine IF... THEN...ELSE-Struktur, die im Vergleich zu PASCAL jedoch recht begrenzt ist:

```
IF Bedingung
   Anweisung(en)
ELSE
   Anweisung(en).
```

Beachten Sie, daß das THEN nicht eingegeben werden muß und daß die IF-Anweisung wie jeder andere Befehl mit einem Punkt zu beenden ist. Das bedeutet, daß die Anweisungen im IF und ELSE-Teil nicht mit einem Punkt abgeschlossen werden dürfen. Bei normalen An-



Die Rechenmaschine von Leibniz wurde 1673 erfunden. Der Hauptzylinder hat neun Stege mit unterschiedlichen Längen, die den Ziffern 0 bis 9 entsprechen. Wenn die Stange für Addition/Subtraktion auf eine bestimmte Ziffernposition gestellt und der Zylinder einmal gedreht wird, drehen sich die Scheiben mit dem kumulativen Ergebnis um den Betrag, der der Zifferneingabe entspricht. Eine Drehung im Uhrzeigersinn addiert, eine Drehung gegen den Uhrzeigersinn subtrahiert.

Die Arbeit von Leibniz hat für die modernen Computerabläufe große Bedeutung und führte schon zu seiner Zeit zu der Entwicklung von Rechenmaschinen.

weisungen hat dies keine Auswirkung, bei Verzweigungen (etwa ein weiteres IF oder ein READ, das prüft, ob das Dateiende erreicht wurde) ist es jedoch wichtig. In all diesen Fällen beendet ein Punkt in der inneren Verzweigung auch die äußere.

IF bietet mehrere Möglichkeiten, Boolesche Bedingungen zu testen. Zunächst gibt es (wie in den meisten Sprachen) die relationalen Operatoren <, > und =, ebenso das NOT. NOT< hat beispielsweise dieselbe Bedeutung wie >=. Die Operatoren können als Symbole oder im Langtext geschrieben werden (Zahl-1 IS EQUAL TO 5). Mit AND oder OR lassen sich beliebig viele Bedingungen miteinander verknüpfen, zum Beispiel (Zahl-1 = 5) OR (Zahl-2 NOT = 6).

„Ebene 88"

COBOL ist eine der wenigen Sprachen, die zusammengesetzte Bedingungen durch eine Verknüpfung des gleichen Datenelementes mit mehreren Operatoren abkürzen können.

```
Zahl-1 < 10 AND > 6
Zeichen-1 = ‚A' OR ‚B' OR ‚C'
```

sind in COBOL völlig legal. COBOL besitzt zwar keinen Booleschen Datentyp, kann aber die „Ebene 88" der Dateneingabe einsetzen, bei der jedes elementare Datenelement einem Wert, einem Wertebereich (oder einer Werteliste) oder einem Booleschen Bedingungsnamen zugeordnet wird. Hier ein Beispiel:

```
77 Zahl-1 PIC 99.
88 Zahl-ist-gültig VALUES 1,3,24 THRU 67.
```

(In einigen COBOL-Versionen sind entweder nur eine Liste oder ein Bereich möglich, nicht aber beide.) Die Bedingung Zahl-ist-gültig ist Zahl-1 zugeordnet, wenn sie im Datenteil unmittelbar hinter Zahl-1 steht. Jeder in Zahl-1 gespeicherte Wert erzeugt nun in Zahl-ist-gültig ein entsprechendes TRUE oder FALSE:

```
IF Zahl-ist-gültig
   PERFORM Routine-gültig
ELSE
   PERFORM Routine-ungültig.
```

Wiederholungen (Schleifen) lassen sich in COBOL auf mehrere Weisen ausführen. Alle verwenden dabei Variationen von PERFORM. Hier das Grundformat:

```
PERFORM Paragraphenname UNTIL
Bedingung
```

wobei die Bedingung jede Kombination von Beziehungen oder Elementen der Ebene 88 (wie bei IF) sein kann. Die Anweisung arbeitet wie eine WHILE-Schleife in BASIC und anderen Sprachen. Dabei wird vor jeder Ausführung des Paragraphen die Bedingung getestet.

Der FOR...NEXT-Schleife in BASIC entspricht in COBOL eine Variation von PERFORM, bei der ein oder mehrere Datenelemente von einem Anfangswert an inkrementiert werden. Die folgende Anweisung führt den Code in Paragraph-1 fünfmal aus:

```
PERFORM Paragraph-1 VARYING Zahl-1
FROM 1 BY 1
UNTIL Zahl-1 > 5.
```

Beachten Sie, daß die Endbedingung jede Form haben kann, nicht nur numerische Werte.

Das nebenstehende COBOL-Programm zeigt den Einsatz von arithmetischen Verben. Es berechnet Pi mit der Leibniz-Serie. Diese algebraische Methode arbeitet mit dem Prinzip der unendlichen Folge, bei der jedes Element der Folge zunehmend kleiner wird.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PI-CALC.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 OBJECT-COMPUTER.
 SPECIAL-NAMES. CONSOLE IS CRT.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01  SCREEN PIC X(1920).
*
 01  DI-1 REDEFINES SCREEN.
     02 FILLER PIC X(160).
     02 DI-TX1 PIC X(160).
     02 DI-TX2 PIC X(13).
     02 DI-TERM PIC X(15).
     02 FILLER PIC X(136).
     02 DI-TX3 PIC X(6).
     02 DI-PI PIC X(15).
     02 FILLER PIC X(1415).
*
 01  DI-2 REDEFINES SCREEN.
     02 FILLER PIC X(333).
     02 DI-TERM2 PIC X(15).
     02 FILLER PIC X(142).
     02 DI-PI2 PIC X(15).
     02 FILLER PIC X(1415).
*
 01  WORK-AREA.
     02 PI PIC S9V9(14).
     02 TERM PIC S9V9(14).
     02 W PIC S9V9(14).
     02 N PIC 9999.
     02 N1 PIC 9999.
     02 N2 PIC 9999.
     02 ED PIC -9.9(12).
*
 01  CONSTANTS.
     02 TX1 PIC X(17) VALUE "CALCULATION OF PI".
     02 TX2 PIC X(12) VALUE "NEXT TERM IS".
     02 TX3 PIC X(5) VALUE "PI IS".
*
 PROCEDURE DIVISION.
 LA-START.
     DISPLAY SPACE.
     MOVE SPACE TO SCREEN.
     MOVE TX1 TO DI-TX1.
     MOVE TX2 TO DI-TX2.
     MOVE TX3 TO DI-TX3.
     MOVE 0.5 TO ED.
     MOVE ED TO DI-TERM.
     MOVE 3 TO ED.
     MOVE ED TO DI-PI.
     DISPLAY DI-1.
     MOVE 0.5 TO PI.
     MOVE 0.5 TO TERM.
     MOVE 3 TO N.

 LOOP.
     MOVE N TO N2.
     SUBTRACT 2 FROM N2.
     MULTIPLY N2 BY N2.
     MULTIPLY N2 BY TERM.
     MOVE N TO N1.
     SUBTRACT 1 FROM N1.
     MULTIPLY N BY N1.
     MULTIPLY 4 BY N1.
     DIVIDE N1 INTO TERM.
     IF TERM < 0.0000000000001 THEN GO TO HALT.
     ADD TERM TO PI.
     MOVE PI TO W.
     MULTIPLY 6 BY W.
     MOVE W TO ED.
     MOVE ED TO DI-PI2.
     MOVE TERM TO ED.
     MOVE ED TO DI-TERM2.
     DISPLAY DI-2.
     ADD 2 TO N.
     IF N < 100 GO TO LOOP.
 HALT.
     STOP RUN.
```

Der Prozessor 6510

Das Betriebssystem des Commodore 64 hat viele praktische, aber auch einige ungewöhnliche Aspekte. Wir betreiben eine Bildschirmuhr.

Systemprogrammierer müssen die „Memory Map", das heißt den Aufbau des Arbeitsspeichers, kennen. Der Microprozessor 6510 des Commodore 64 kann – je nach Konfiguration der Maschine – mit einer von mehreren Speicherstrukturen arbeiten, da er mit der Technik des „Bank Switching" Zugriff auf unterschiedliche Speicherblocks hat. Die Umschaltung geschieht dabei entweder über die Hardware (an die Kontakte 8 und 9 der Erweiterungsbuchse wird hohe (+5V) oder niedrige (0V) Spannung angelegt) oder über die Software (durch die Änderung des Inhalts der Adresse 1). Beide Methoden ändern radikal die Interpretation des Speicheraufbaus. Die Tabelle gibt einen Einblick in den standardmäßig vorgegebenen Speicheraufbau der 64 KBytes, die der 6510 ansprechen kann.

Wenn wir davon ausgehen, daß der Commodore 64 mit der Standardeinstellung arbeitet, dann fällt als erstes auf, daß der Computer 64 KByte RAM und 20 KByte ROM enthält. Da der 6510 von diesen 84 KBytes aber nie mehr als 64 adressieren kann, wird hier das Bank Switching eingesetzt. Die Memory Map zeigt, daß sich das ROM des BASIC-Interpreters zwischen $A000 und $CFFF befindet und das ROM für die Ein- und Ausgabefunktionen zwischen $E000 und $FFFF.

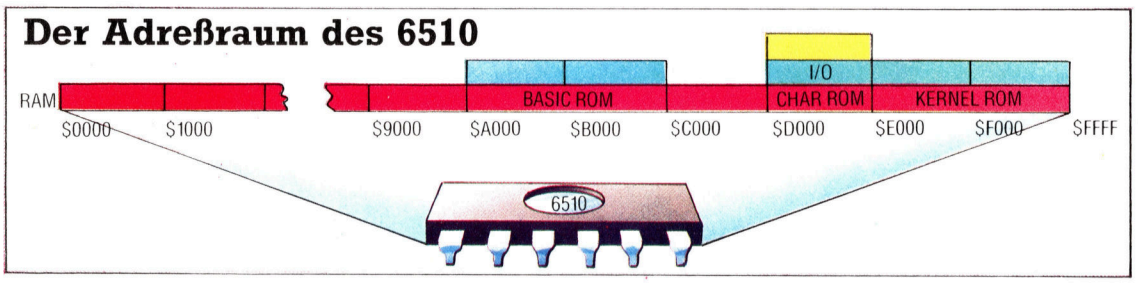
Schattendasein

Unter diesen beiden ROM-Blöcken verbergen sich zwei RAM-Blocks zu je acht KByte, die der 6510 normalerweise nicht ansprechen kann (ein POKE oder STA auf diese Adressen könnte dahin vordringen, PEEK liefert jedoch immer nur den Inhalt des darüberliegenden ROMs). Das RAM läßt sich lesen, wenn das entsprechende ROM mit Bank Switching ausgeschaltet wird. Denken Sie beim Einsatz der BASIC-Befehle PEEK und POKE jedoch daran, daß nicht auch das BASIC-ROM ausgeschaltet wird.

Wie jedes BASIC-Programm brauchen auch der Interpreter und die E/A-Routinen einen RAM-Bereich (das OS-RAM) zur Speicherung von Variablen. Das OS-RAM reicht von $0002 bis $03FF, wobei ein wichtiger Teilbereich – der Stack – zwischen $0100 und $01FF liegt.

Eile ist geboten

Das OS-RAM liegt auf der Zero Page, da dieser Speicherbereich den schnellsten RAM Zugriff bietet und die Geschwindigkeit der OS-Routinen außerordentlich wichtig ist. Wenn Sie in einem BASIC-Programm Maschinencoderoutinen einsetzen, sollten Sie diesen RAM-Bereich daher genau kennen. Sehen wir uns zwei wichtige Aspekte des OS-RAM genauer an: die BASIC-Pointer und überaus nützlichen Maschinencodevektoren.



Der 6510 Prozessor arbeitet mit 16-Bit-Adressen und kann daher maximal auf 65536 (64K) Speicherstellen zugreifen. Die insgesamt 84K (RAM und ROM) des Commodore 64 werden mit der Technik des „Bank Switching" adressiert, die bestimmte Adreßbereiche in der „Memory Map" ein- oder ausschaltet.

Der vordere „Block" zeigt, wie der 6510 seinen Speicher normalerweise interpretiert. Wenn spezielle „Banking"-Register gesetzt sind, lassen sich aber auch die darunterliegenden Bereiche ansprechen.

Die Memory Map des Commodore 64

Adressen	Funktionen
$0000 —$0001	Steuerregister der Memory Map des 6510
$0002 —$03FF	Betriebssystem-RAM
$0400 —$07F7	Bildschirm-RAM
$07F8 —$07FF	Sprite Pointer
$0800 —$9FFF	BASIC-Programmbereich (mit Variablen)
$A000 —$BFFF	ROM des BASIC-Interpreters
$C000 —$CFFF	4 KByte freies RAM
$D000 —$D02E	VIC II (6566/9) Chip-Steuerregister
$D02F —$D3FF	Wiederholung der VIV II Abbildung
$D400 —$D41C	SID (6581) Chip-Steuerregister
$D41D —$D7FF	Wiederholung der SID-Abbildung
$D800 —$DBE7	Farbnybbles (je 4 Bits)
$DBE8 —$DBFF	Nicht eingesetzte Nybbles
$DC00 —$DC0F	CIA # 1 E/A (6526) Chip-Steuerregister
$DC10 —$DCFF	Wiederholung der CIA # 1 Abbildung
$DD00 —$DD0F	CIA # 2 E/A (6526) Chip-Steuerregister
$DD10 —$DDFF	Wiederholung der CIA # 2 Abbildung
$E000 —$FFFF	ROM für Ein- und Ausgabe

Pointertabelle des C64 BASIC

Adressen	Pointer
$002B—$002C	Anfang des BASIC-Bereichs für Anwender
$002D—$002E	Anfang des Bereichs für BASIC-Variablen
$002F—$0030	Anfang des Bereichs für BASIC-Arrays
$0031—$0032	Ende der BASIC Arrays+1
$0033—$0034	Untergrenze für Strings
$0035—$0036	Dienstbereich
$0037—$0038	Ende des BASIC-Bereichs für Anwender

BASIC Pointer

Zuweilen kann eine Änderung der BASIC-Pointer sehr praktisch sein. Der normale Inhalt von $002B (dezimal 43) und $002C (dezimal 44) ist beispielsweise 1 und 8 und zeigt so die Anfangsadresse des BASIC an (im Format lo-hi). BASIC beginnt also bei 8*256+1=2049 oder $0801. Tatsächlich liegt der Anfang von BASIC zwar bei $0800, doch verlangt das Betriebssystem, daß das erste Byte auf Null steht. (Nebenbei: Wenn Sie mit dem Maschinencodemonitor des Commodore arbeiten, ist $0801 auch die Adresse, von der an Sie ein BASIC-Programm sichern müssen.)

Vor dem Laden von BASIC-Programmen können Sie mit diesen beiden Pointern die Untergrenze des BASIC verlegen. BASIC darf jedoch nur an einer Seitengrenze beginnen (das bedeutet, der Inhalt von $002B sollte auf Eins bleiben) und muß mit einer Null anfangen (denken Sie daran, daß eine „Seite" 256 Bytes enthält). Der direkte Befehl

POKE2560,0:POKE44,10:NEW

erhöht die Untergrenze des Speichers um zwei Seiten auf $2560. NEW ist die schnellste Methode, alle Pointer (zwischen $002D und $0038) zurückzusetzen. Durch die Erhöhung der BASIC-Untergrenze können sich zwei BASIC-Programme gleichzeitig im Speicher befinden. Sie laden dabei nur das erste Programm, erhöhen die BASIC-Untergrenze und laden dann das zweite Programm.

Oft wird auch die Obergrenze des Speichers herabgesetzt, um Platz für Maschinencodeprogramme zu schaffen. Die direkten Befehle

POKE56,159:POKE51,0:POKE52,159

verlegen die Obergrenze des BASIC-Speichers um eine Seite nach unten.

Ein RAM-Block, der auf diese Weise dem Zugriff von BASIC entzogen wurde, kann nun sicher für den Maschinencode eingesetzt werden, da das OS diesen Bereich nicht mehr für das Speichern von BASIC-Variablen verwenden kann.

Die RAM-Vektoren

Neben den BASIC-Pointern haben auch die RAM-Vektoren zwischen $0314 und $0333 große Bedeutung für den Maschinencodeprogrammierer. Ein RAM-Vektor ist einer zusätzlichen Weiche in einem Schienenstrang nicht unähnlich. Wenn ein Zug (oder der 6510 bei der Ausführung eines Programms) diesen Weg nimmt, wird er im Normalfall die Weiche passieren, ohne den Seitenpfad zu nehmen. Manchmal ist es jedoch praktisch, den Zug über die Seitenlinie zu leiten und zusätzliche Stationen passieren zu lassen, bevor er auf der normalen Route weiterfährt.

Nehmen wir als Beispiel den IRQ-Vektor (Interrupt ReQuest): Im normalen Operationsmodus des Commodore 64 veranlaßt einer der Timer des 6526 E/A-Chips jede Sechzigstelsekunde, daß die IRQ-Leitung des 6510 auf niedrige Spannung gesetzt wird. Nach Beendigung des gerade bearbeiteten Befehls reagiert der 6510 auf die IRQ-Anforderung und springt auf ein Dienstprogramm, dessen Code bei $FF48 liegt. Die IRQ-Behandlungsroutine prüft nun unter anderem, ob auf der Tastatur eine Taste gedrückt wurde.

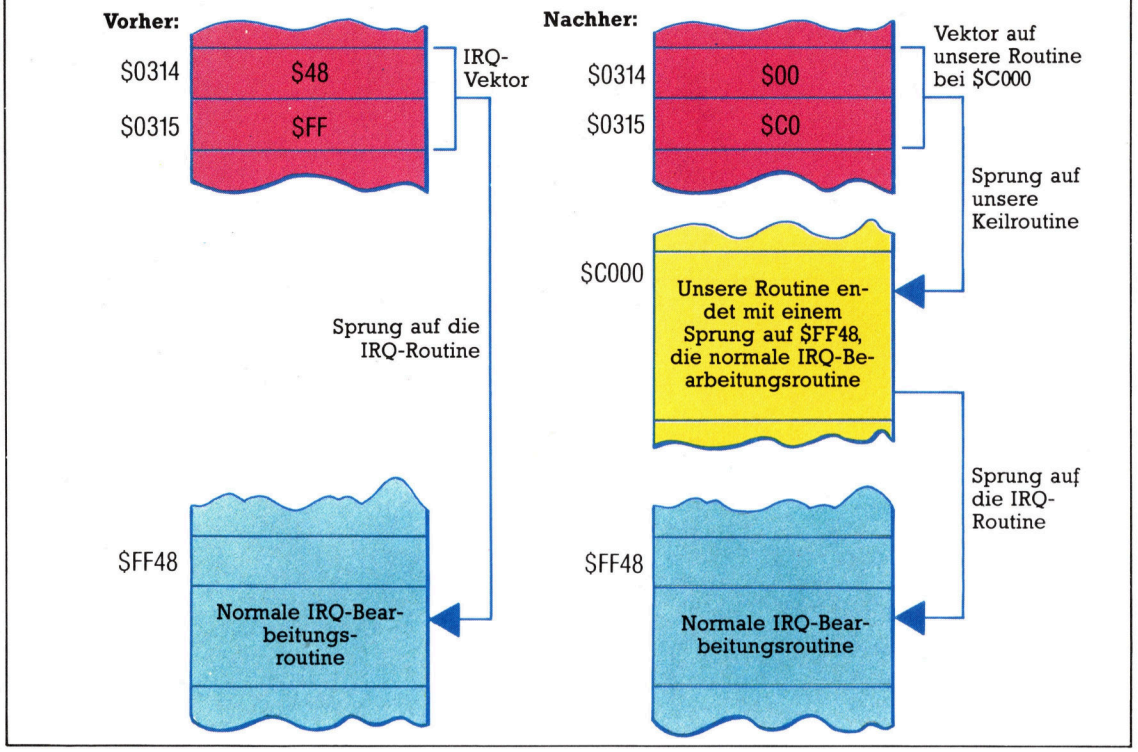
JMP ($0314)

ist einer der ersten Befehle der Routine. Er führt einen indirekten Sprung auf die Adresse aus, die in $0314 (niederwertiges Byte) und $0315 (höherwertiges Byte) gespeichert ist. Da diese beiden Bytes im RAM liegen, können wir ihren Inhalt ändern und den Vektor damit auf unseren eigenen Code zeigen lassen. Nach Ausführung unseres Codes wird der Microprozessor dann wieder auf das „Gleis" gesetzt, von dem wir ihn abgelenkt hatten. Wenn der Code nicht zu lang ist, kann der 6510 ihn jede Sechzigstelsekunde ausführen.

Vektorentabelle des C64

Adressen	Pointer
$0314 –$0315	IRQ Interrupt
$0316 –$0317	BRK Interrupt
$0318 –$0319	NMI Interrupt
$031A–$031B	E/A-Routine OPEN
$031C–$031D	E/A-Routine CLOSE
$031E–$031F	E/A-Routine CHKIN
$0320 –$0321	E/A-Routine CHKOUT
$0322 –$0323	E/A-Routine CLRCHN
$0324 –$0325	E/A-Routine CHRIN
$0326 –$0327	E/A-Routine CHROUT
$0328 –$0329	E/A-Routine STOP
$032A–$032B	E/A-Routine GETIN
$032C–$032D	E/A-Routine CLALL
$032E–$032F	Vom Anwender definiert
$0330 –$0331	E/A-Routine LOAD
$0332 –$0333	E/A-Routine SAVE

In Keilform



Der 6510 Prozessor wird jede Sechzigstelsekunde unterbrochen, um interne Organisationsaufgaben (das Abfragen der Tastatur auf Eingaben, etc.) auszuführen. Die Anfangsadresse dieser Bearbeitungsroutine liegt in dem Adreßpaar $0314 und $0315.

Wenn Sie den Inhalt dieser beiden Speicherstellen ändern, können Sie Ihren eigenen Code als „Keil" hineinschieben, der dann anstelle der normalen Bearbeitungsroutine ausgeführt wird. Normalerweise endet der Code eines derartigen „Keils" mit einem Sprung auf die normale IRQ-Bearbeitungsroutine. Sie stellen damit sicher, daß Ihr Code bei jedem Interrupt ausgeführt wird.

Die Zeit des Commodore 64

Das folgende Assemblerlisting und das Ladeprogramm in BASIC zeigen, wie ein „Keil" in die normale Bearbeitungsroutine für Interrupts eingesetzt wird. Die Keilroutine stellt am Bildschirmrand eine Uhr dar, die bei normalen Interrupts (die jede Sechzigstelsekunde eintreten) aktualisiert wird. Da die Uhr parallel zu anderen Vorgängen läuft, können Sie gleichzeitig mit BASIC-Programmen weiterarbeiten. Die Routine stellt zunächst die übergebene Anfangszeit ein und setzt dann den Inhalt von $0314 und $0315 auf den Anfang des „Keil"-Codes. Der Code aktualisiert die Zeit bei jedem IRQ-Interrupt. Letzter Befehl des Codes ist ein Sprung (JMP) auf die normale IRQ-Bearbeitungsroutine, deren Adresse in den Speicherstellen VECTOR und VECTOR+1 abgelegt wurde.

Statt den Quellentext einzugeben und zu assemblieren, können Sie das Programm auch mit DATA-Befehlen schreiben. Geben Sie dazu einfach das Ladeprogramm in BASIC ein und starten Sie es. Die Uhr erscheint am rechten oberen Bildschirmrand. Über eine Prüfsumme wird festgestellt, ob alle DATAs korrekt eingegeben wurden. Wenn das Programm mit der Meldung „CHECKSUM ERROR" abbricht, sollten Sie die DATAs überprüfen.

Die Uhr kann mit POKEs in die Speicherstellen 50129 (Stunden), 50130 (Minuten) und 50131 (Sekunden) gestellt werden. Im Ladeprogramm wird die Uhr von Zeile 1480 an auf den Anfangswert 4:30h gestellt. Sie können andere Zeiten einsetzen, wenn Sie die entsprechenden POKE-Werte ändern. SYS 50138 veranlaßt die Ausführung des Maschinencodes und setzt die Uhr in Gang. Die Uhr schaltet ab, wenn Sie die Tasten RUN/STOP und RESTORE gleichzeitig drücken, oder die spezielle Abschaltroutine im Maschinencodeprogramm mit SYS 50237 aktivieren.

Ladeprogramm in BASIC

```
1000 REM ** CLOCK BASIC LOADER **
1010 DATA173,166,2,240,10,169,128,13,14
1020 DATA221,141,14,221,48,8,169,127,45
1030 DATA14,221,141,14,221,169,127,45
1040 DATA15,221,141,15,221,173,208,195
1050 DATA41,128,141,208,195,173,209,195
1060 DATA32,28,197,13,208,195,141,11
1070 DATA221,173,210,195,32,28,197,141
1080 DATA10,221,173,211,195,32,28,197
1090 DATA141,9,221,169,0,141,8,221,120
1100 DATA173,20,3,141,214,195,173,21,3
1110 DATA141,215,195,169,76,141,20,3
1120 DATA169,196,141,21,3,88,96,120,173
1130 DATA214,195,141,20,3,173,215,195
1140 DATA141,21,3,88,96,173,216,195,201
1150 DATA6,240,3,76,8,197,169,255,141
1160 DATA216,195,173,213,195,240,243
1170 DATA173,11,221,170,41,128,208,5
1180 DATA169,1,76,111,196,169,16,141,38
1190 DATA4,173,212,195,141,38,216,163
1200 DATA13,141,39,4,173,212,195,141,39
1210 DATA216,138,41,16,32,14,197,141,28
1220 DATA4,173,212,195,141,28,216,138
1230 DATA32,22,197,141,29,4,173,212,195
1240 DATA141,29,216,169,58,141,30,4,173
1250 DATA212,195,141,30,216,173,10,221
1260 DATA170,32,14,197,141,31,4,173,212
1270 DATA195,141,31,216,138,32,22,197
1280 DATA141,32,4,173,212,195,141,32
1290 DATA216,169,47,141,33,4,173,212
1300 DATA195,141,33,216,173,9,221,170
1310 DATA32,14,197,141,34,4,173,212,195
```

```
1320 DATA141,34,216,138,32,22,197,141
1330 DATA35,4,173,212,195,141,35,216
1340 DATA169,46,141,36,4,173,212,195
1350 DATA141,36,216,173,8,221,105,48
1360 DATA141,37,4,173,212,195,141,37
1370 DATA216,238,216,195,108,214,195,74
1380 DATA74,74,74,24,105,48,96,41,15,24
1390 DATA105,48,96,160,255,56,200,233
1400 DATA10,176,251,105,10,141,217,195
1410 DATA152,10,10,10,10,13,217,195,96
1420 DATA42131:REM*CHECKSUM*
1430 CC=0
1440 FORI=50138TO50481
1450 READX:CC=CC+X:POKEI,X
1460 NEXT
1470 READX:IFCC<>XTHENPRINT"CHECKSUM ERROR"
1480 REM** TEST CLOCK **
1490 POKE50128,128:REM AM/PM
1500 POKE50132,8:REM COLOUR
1510 POKE50133,1:REM DISPLAY
1520 POKE50129,4:REM HOURS
1530 POKE50130,30:REM MINUTES
1540 POKE50131,0:REM SECONDS
1550 SYS50138:REM CALL ROUTINE
```

+0 TURN OFF THE WEDGE USE SYS50237
IN DIRECT OR PROGRAM MODE.

Assemblerlisting

```
;***********************************
;*                                 *
;*        CLOCK IRQ WEDGE          *
;*                                 *
;* 50128 = AM=0 / PM=128           *
;* 50129 = HOURS                   *
;* 50130 = MINUTES                 *
;* 50131 = SECONDS                 *
;* 50132 = CLOCK COLOUR            *
;* 50133 = DISPLAY ON=1/OFF=0      *
;*                                 *
;* WEDGE INSERT  SYS 50138         *
;* WEDGE REMOVE  SYS 50237         *
;*                                 *
;***********************************

IRQVEC = $0314   ; IRQ RAM VECTOR
CLOCK = $DD08    ; TOD REGISTER
D2CRA = $DD0E    ; VIA#2 CRA
D2CRB = $DD0F    ; VIA#2 CRB
PALNTS = $02A6   ; PAL/NTSC FLAG
RATE = $06       ; DISPLAY EVERY 6 IRQS
DIGIT = $30      ; SCREEN CODE FOR '0'
POINT = $2E      ; SCREEN CODE FOR '.'
SLASH = $2F      ; SCREEN CODE FOR '/'
COLON = $3A      ; SCREEN CODE FOR ':'
HZ50 = $80       ; 50 HZ MASK FOR TODIN
HZ60 = $7F       ; 60 HZ MASK FOR TODIN
AY = $01         ; SCREEN CODE FOR 'A'
PEE = $10        ; SCREEN CODE FOR 'P'
EM = $0D         ; SCREEN CODE FOR 'M'
WRITE = 127      ; MASK TO SET CLOCK IN CRB
SCNLOC = $041C   ; CLOCK ADDR ON SCREEN
COLLOC = $D81C   ; ADDR ON VIDEO MATRIX
TRNCLO = $0F     ; MASK FOR LOW NYBBLE

* = $C3D0         ; START ADDRESS CODE
AMPM *=*+1       ; AM/PM FLAG
HOURS *=*+1      ; HOURS VAL (FOR INITIALISE)
MINS *=*+1       ; MINUTES VALUE
SECS *=*+1       ; SECONDS VALUE
COLOR *=*+1      ; CLOCK COLOUR
DISPLY *=*+1     ; DISPLAY/HIDE FLAG
VECTOR *=*+2     ; STORAGE FOR OLD IRQ VEC
COUNT *=*+1      ; IRQ COUNTER
TEMP1 *=*+1

;
;INSERT WEDGE
;

   LDA PALNTS      ; PAL OR NTSC
   BEQ NTSC        ; BRANCH FOR NTSC
   LDA #HZ50       ; MUST BE PAL
   ORA D2CRA
   STA D2CRA       ; SET TOSIN FOR 50 HZ
   BMI PALDUN
NTSC
   LDA #HZ60       ; NTSC
   AND D2CRA
   STA D2CRA       ; SET TODIN FOR 60 HZ
PALDUN
   LDA #WRITE
   AND D2CRB       ; SET CLOCK NOT ALARM
   STA D2CRB
   LDA AMPM
   AND #128        ; MAKE AMPM VALUE VALID
   STA AMPM
   LDA HOURS       ; GET HOURS
   JSR BINBCD      ; CONVERT TO BCD
   ORA AMPM        ; OR WITH AM/PM FLAG
   STA CLOCK+3     ; STORE IN CLOCK
   LDA MINS        ; GET MINUTES
   JSR BINBCD      ; CONVERT TO BCD
   STA CLOCK+2     ; STORE IN CLOCK
   LDA SECS        ; GET SECONDS
   JSR BINBCD      ; CONVERT TO BCD
   STA CLOCK+1     ; STORE IN CLOCK
   LDA #00         ; ALWAYS SET 10THS TO 0
   STA CLOCK       ; START CLOCK
;
   SEI             ; DISABLE INTERRUPTS
   LDA IRQVEC
   STA VECTOR      ; SAVE OLD IRQ VECTOR
   LDA IRQVEC+1
   STA VECTOR+1
;
   LDA #<WEDGE
   STA IRQVEC
   LDA #>WEDGE     ; INSERT WEDGE
   STA IRQVEC+1
   CLI             ; ENABLE INTERRUPTS
   RTS

;
;REMOVE WEDGE
;
   SEI             ; DISABLE INTERRUPTS
   LDA VECTOR
   STA IRQVEC      ; RESTORE RAM VECTOR
   LDA VECTOR+1
   STA IRQVEC+1
   CLI             ; ENABLE INTERRUPTS
   RTS

;
;WEDGE STARTS HERE
;
WEDGE
   LDA COUNT       ;
   CMP #RATE       ; DO CLOCK THIS IRQ?
   BEQ CONT        ;
OUT
   JMP EXIT        ; NO
CONT
   LDA #$FF        ; RESET IRQ COUNTER
   STA COUNT
   LDA DISPLY      ; DISPLAY?
   BEQ OUT         ; NO...BRANCH

   LDA CLOCK+3     ; GET HOURS/AM/PM
   TAX             ; PUT A COPY IN X REG
   AND #$80        ; GET AM/PM
   BNE PM          ; BRANCH IF PM
   LDA #AY         ; DISPLAY 'A'
   JMP MERIDP
PM
   LDA #PEE        ; DISPLAY 'P'
MERIDP
   STA SCNLOC+10
   LDA COLOR       ; GET COLOUR
   STA COLLOC+10   ; SET COLOUR
   LDA #EM         ; DISPLAY 'M'
   STA SCNLOC+11
   LDA COLOR       ; SET COLOUR
   STA COLLOC+11
;
;DO HOURS
;
   TXA             ; GET HOURS
   AND #$10        ; JUST WANT HIGH DIGIT
   JSR HIDIGT      ; GET SCREEN CODE
   STA SCNLOC      ; DISPLAY IT
   LDA COLOR
   STA COLLOC      ; SET COLOUR
   TXA             ; GET BYTE AGAIN
   JSR LODIGT      ; GET LOW DIGIT
   STA SCNLOC+1    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+1    ; SET COLOUR
;
   LDA #COLON      ; HRS/MINS SEPARATOR
   STA SCNLOC+2
   LDA COLOR
   STA COLLOC+2
;
;NOW DO MINUTES
;
   LDA CLOCK+2     ; GET MINUTES
   TAX
   JSR HIDIGT      ; DO HIGH DIGIT
   STA SCNLOC+3    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+3    ; AND COLOUR
   TXA             ; GET BYTE AGAIN
   JSR LODIGT      ; DO LOW DIGIT
   STA SCNLOC+4    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+4    ; SET COLOUR
;
   LDA #SLASH      ; MIN/SEC SEPARATOR
   STA SCNLOC+5
   LDA COLOR
   STA COLLOC+5
;
;NOW DO SECONDS
;
   LDA CLOCK+1     ; GET SECONDS
   TAX
   JSR HIDIGT      ; DO HIGH DIGIT
   STA SCNLOC+6    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+6    ; AND COLOUR
   TXA             ; GET BYTE AGAIN
   JSR LODIGT      ; DO LOW DIGIT
   STA SCNLOC+7    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+7    ; AND COLOUR
;
   LDA #POINT      ; SECS/TENTHS SEPARATOR
   STA SCNLOC+8
   LDA COLOR
   STA COLLOC+8
;
;NOW DO TENTHS
;
   LDA CLOCK       ; GET TENTHS VALUE
   ADC #DIGIT      ; ADD $30 FOR SCREEN CODE
   STA SCNLOC+9    ; DISPLAY IT
   LDA COLOR
   STA COLLOC+9    ; AND COLOUR
;
;
EXIT
   INC COUNT       ; INCREMENT IRQ COUNTER
   JMP (VECTOR)    ; GO TO REST OF IRQ

;
;SUBROUTINES
;
HIDIGT
   LSR A
   LSR A           ; MOVE HIGH NYBBLE INTO LOW
   LSR A
   LSR A
   CLC
   ADC #DIGIT      ; ADD $30 FOR SCREEN CODE
   RTS

LODIGT
   AND #TRNCLO     ; MASK OFF HIGH NYBBLE
   CLC
   ADC #DIGIT      ; ADD $30 FOR SCREEN CODE
   RTS

;
;CONVERT BINARY TO BCD
;
BINBCD
   LDY #$FF
   SEC
D10
   INY
   SBC #10         ; SUBTRACT 10 UNTIL -VE
   BCS D10
   ADC #10         ; ADD 10 BACK ON
   STA TEMP1       ; STORE REMAINDER
   TYA             ; GET NUMBER PF 10S SUBTRACTED
   ASL A
   ASL A
   ASL A           ; SHIFT INTO HIGH NYBBLE
   ASL A
   ORA TEMP1       ; PUT REMAINDER IN LOW NYBBLE
   RTS
```

Herzspezialisten

Innerhalb der letzten Jahre hat sich Motorola Incorporated aus einer bescheidenen Außenseiterposition an die Spitze der Hersteller microelektronischer Bauteile vorgearbeitet.

Wie viele andere erfolgreiche Konzerne hat sich auch Motorola aus einem Ein-Mann-Unternehmen entwickelt: Paul Galvin gründete 1928 in Chikago die „Galvin Manufacturing Corporation", die auf die Produktion von Radioapparaten spezialisiert war. In den 30er Jahren wurde das Programm um Autoradios und Funkgeräte für die Polizei erweitert. Bereits in den 40er Jahren begann die Firma – jetzt als Motorola Incorporated – mit der Halbleiterproduktion.

1959 starb Paul Galvin, neuer Chef wurde sein Sohn Robert. In den folgenden Jahren geriet Motorola im Halbleiterbereich und bei der Unterhaltungselektronik unter starken Konkurrenzdruck. Die Wirtschaftskrise Mitte der 70er Jahre steigerte die Verluste der Gesellschaft noch weiter – eine Umorientierung wurde unvermeidlich. Neue Mitarbeiter wurden angeworben, darunter viele aus dem Hause des Erzrivalen Texas Instruments. Motorolas neuer Weg führte fort von den traditionellen Bereichen der Elektronik, auf denen das Unternehmen nicht mehr konkurrenzfähig war. Das Ziel hieß jetzt High-Tech-Microelektronik.

Gewagte Operationen

Um die riesigen Summen für Forschung und Entwicklung aufzubringen, verkaufte Motorola Teile der Firma. Im Gegenzug wurden Unternehmen aus Bereichen übernommen, in die Motorola vordringen wollte – ein riskantes Vorgehen, aber es gab keine andere Wahl.

Offensichtlich hat sich das Wagnis gelohnt. Zwar stand Motorola auch im zweiten Teil der 70er noch im Schatten anderer Unternehmen der Halbleiterbranche, heute ist das Ziel jedoch erreicht – der einstige Marktführer Texas Instruments wurde überrundet. Robert Galvin sagt dazu: „Die früheren Konkurrenten von Motorola existieren einfach nicht mehr. Sie haben sich nicht rechtzeitig den veränderten Gegebenheiten angepaßt."

Bis heute hat man bei Motorola allerdings Schwierigkeiten dabei, Produkte zum richtigen Zeitpunkt auf den Markt zu werfen. Mitte der 70er Jahre hatte der Motorola-Chip 6800 nicht nur den im erfolgreichen Apple-Heimcomputer eingebauten Prozessor Mostek 6502 als Konkurrenten, sondern auch noch den Intel 8085 und den Zilog Z80 für CP/M-Rechner. 1976 kam der Motorola 6809 heraus, damals der anerkannt beste Acht-Bit-Prozessor. Der Wettbewerb um den Massenmarkt war jedoch bereits verloren – der Chip wurde nur in wenigen Rechnern als „Herz" eingesetzt, etwa dem Tandy Color Computer und dem Dragon.

Der Schrittmacher

Trotzdem investiert die Gesellschaft nach wie vor kräftig in die Forschung. „Maximalen Fortschritt zum frühestmöglichen Zeitpunkt" heißt die Devise von Robert Galvin. Im Markt der 16-Bit-Prozessoren hat Motorola denn auch eine bessere Position: Der 68000-Microprozessor erschien bereits 1979. Heute sitzt dieser Chip in Apples Lisa und auch im Macintosh, im Atari ST und im Amiga. Sinclair hat ihn für den QL übernommen. Der Chip ist außerordentlich leistungsfähig, verfügt über 17 32-Bit-Register, einen 16-Bit-Datenbus und einen 24-Bit-Adreßbus, damit genau über das, was der Markt verlangt, auf dem heute Massen umgesetzt werden können, auf dem Gewinne erzielt werden.

In den Forschungszentren von Motorola in Phoenix (Arizona), Genf (Schweiz) und East Kilbridge (Schottland) werden weiterhin neue Produkte entwickelt. Die schottische Fabrik erzeugt MOS- und CMOS-Bauteile für einen breitgefächerten Anwendungsbereich. Heute ist Motorola in fünf Unternehmensbereiche aufgeteilt (Kommunikation, Halbleiter, Informationssysteme, Industrieelektronik und Militärtechnik).

Motorola-Chef Robert Galvin

Motorola-Zentrale in Illinois, USA

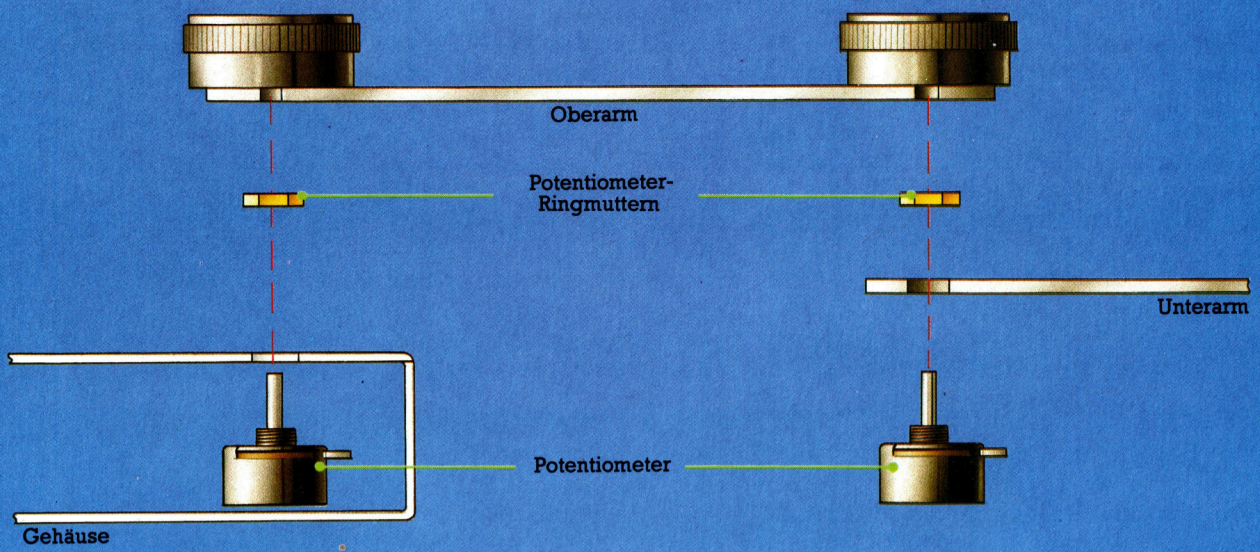
Endmontage

Weiter geht's mit dem Digitalisierarm. In diesem Bauabschnitt wird das Gerät montiert und mit den Potentiometern sowie einer Visiereinrichtung ausgestattet.

Zusammenbau des Arms

Als Gelenke für die beiden Armteile dienen uns zwei Potentiometer. Einer der beiden sitzt in dem kleinen Kunststoffgehäuse oben auf der Grundplatte, der zweite verbindet den Ober- mit dem Unterarm. Zuerst müssen Sie die Potentiometer-Achsen auf 15 mm kürzen. Die Potis werden mit ihren Ringmuttern am Gehäusedeckel bzw. am Unterarm montiert.

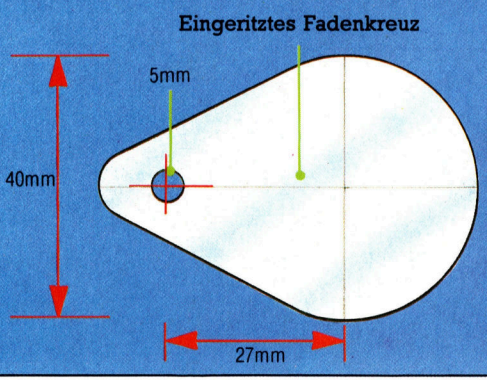
Die Anschlußfahnen der Drehregler sollten im Gehäuse zur benachbarten Seitenwand zeigen, bei dem am Arm montierten Poti weisen sie in dessen Längsrichtung (siehe Zeichnung). Danach können die Potentiometer-Achsen in die Drehknöpfe gesteckt und dort mit den Madenschrauben fixiert werden. Drehen Sie dabei die Achsen so, daß die abgeflachte Seite zur Madenschraube hin zeigt.

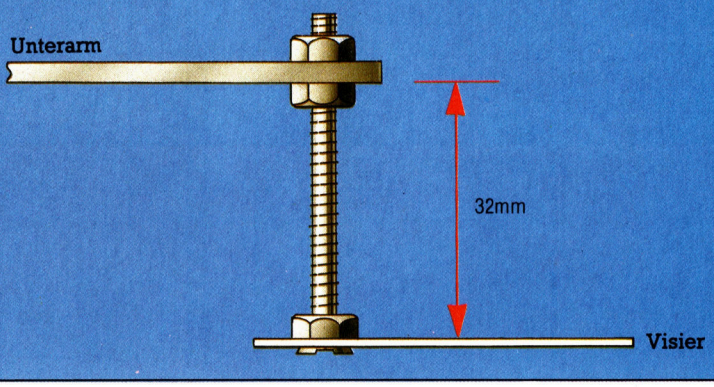


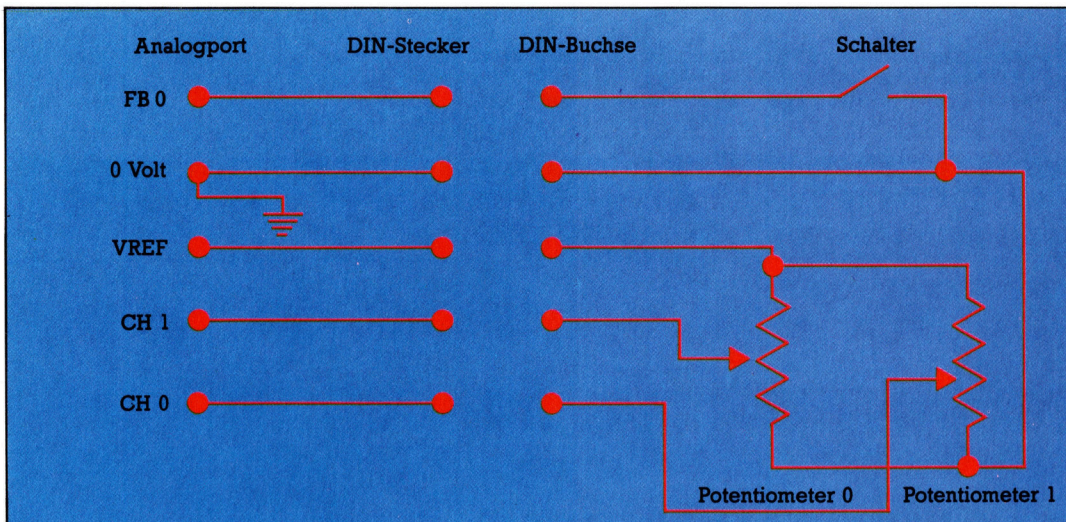
Visier montieren

Schneiden Sie das Visier aus relativ dünnem transparenten Kunststoff, wie er für die Hüllen von Cassetten verwendet wird. Das eingeritzte Fadenkreuz sollte exakt 27 mm vom Befestigungsloch entfernt sein.

Das Visier wird am freien Ende des Unterarms mit einer M5-Schraube und drei Muttern angebracht. Als Anfangshöhe auf 32 mm einstellen – durch die Mutternbefestigung können Sie den Abstand später leicht ändern, wenn Sie mit unterschiedlich starken Vorlagen arbeiten wollen.

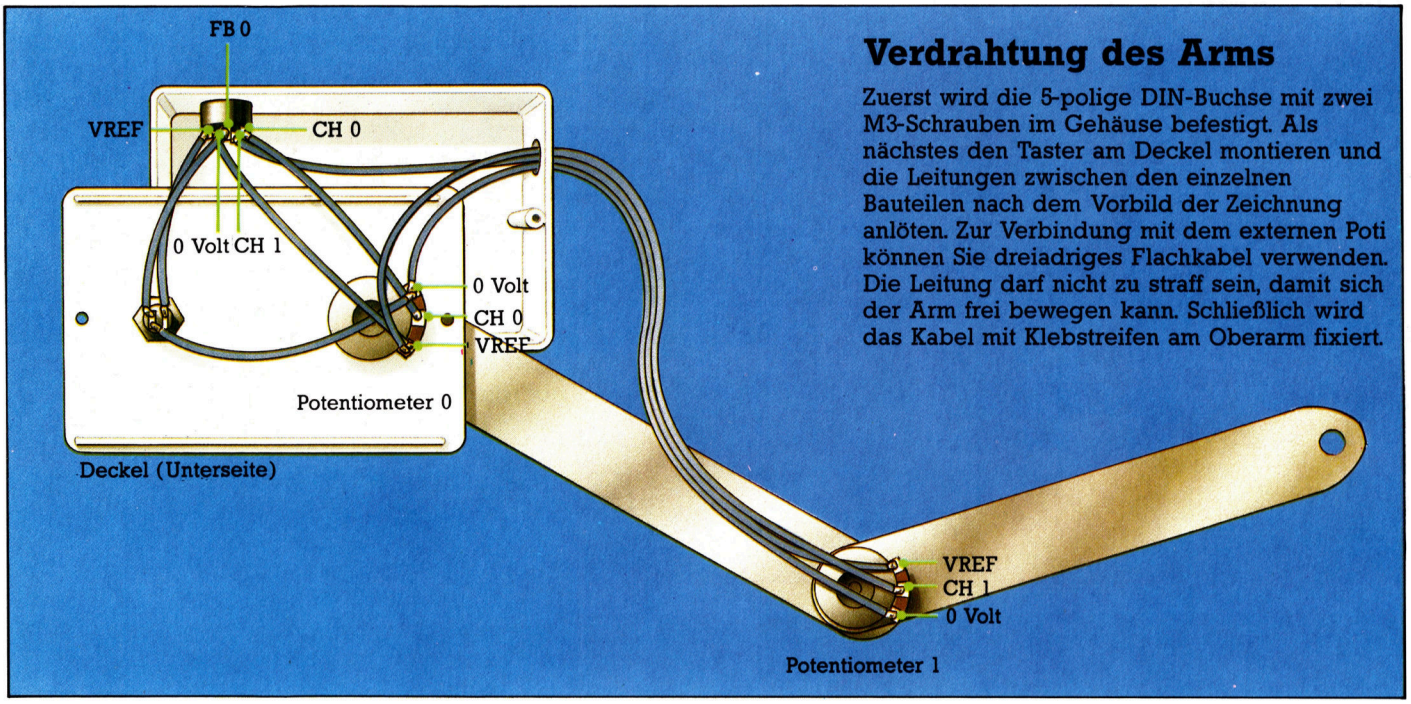






Schaltbild

Der Taster ist mit dem FB0-Kanal des Analogeingangs verbunden, der das Feuerknopfsignal des Joysticks liest. Das Steuerprogramm kann diesen Anschluß jederzeit abfragen. Die bei A/D-Wandlung notwendige Referenzspannung (VREF) liegt über den Widerstandsbahnen der beiden Potentiometer. Die Mittelanschlüsse (Schleifer) der Potis sind mit den ADC-Kanälen 0 und 1 verbunden. Der hier abgegriffene Spannungswert wird in Digitalformat umgewandelt.

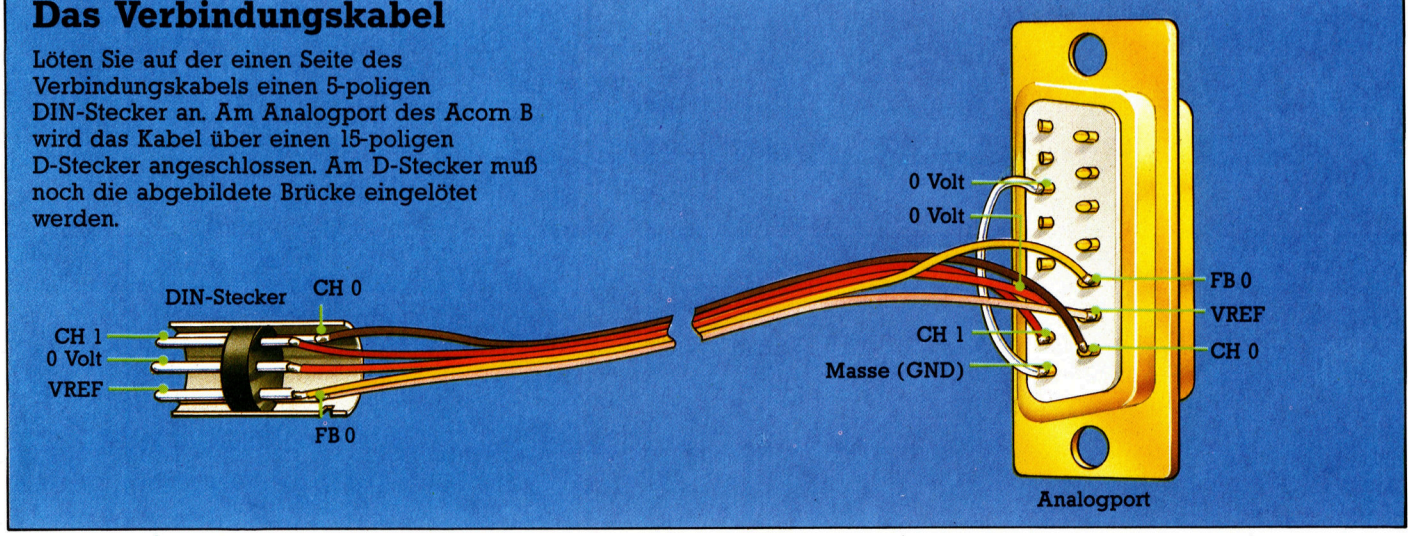


Verdrahtung des Arms

Zuerst wird die 5-polige DIN-Buchse mit zwei M3-Schrauben im Gehäuse befestigt. Als nächstes den Taster am Deckel montieren und die Leitungen zwischen den einzelnen Bauteilen nach dem Vorbild der Zeichnung anlöten. Zur Verbindung mit dem externen Poti können Sie dreiadriges Flachkabel verwenden. Die Leitung darf nicht zu straff sein, damit sich der Arm frei bewegen kann. Schließlich wird das Kabel mit Klebstreifen am Oberarm fixiert.

Das Verbindungskabel

Löten Sie auf der einen Seite des Verbindungskabels einen 5-poligen DIN-Stecker an. Am Analogport des Acorn B wird das Kabel über einen 15-poligen D-Stecker angeschlossen. Am D-Stecker muß noch die abgebildete Brücke eingelötet werden.



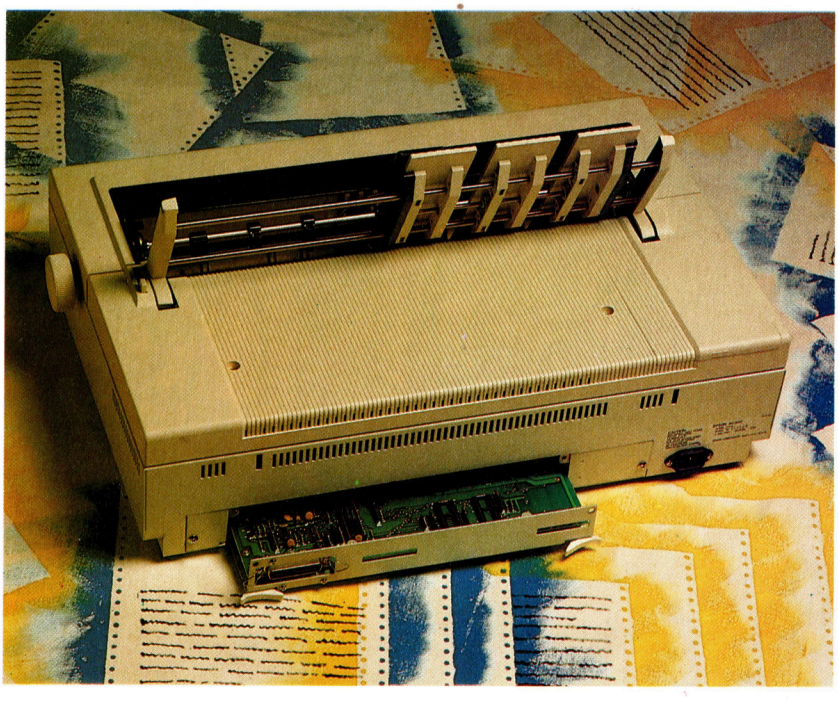
Der Jet-Set

Als einer der führenden Matrixdrucker-Hersteller für Microcomputer hat sich die Firma Epson etabliert. Ein aktuelles Produkt aus diesem Haus ist der Tintenstrahldrucker SQ-2000.

Der SQ-2000 ist ein hochwertiger Tintenstrahldrucker mit einer großen Auswahl an Schriftarten. Er hat eine neu entwickelte Patronen-Tintenversorgung mit automatischem Reinigungssystem.

An der Rückseite des SQ-2000 ist der Steckplatz für ein Centronics-, RS232C- oder IEEE-Interfacemodul vorgesehen. Daneben können auch noch ein 32K-Buffer und ROMs mit Schriftarten eingesetzt werden.

In den letzten Jahren sind die Preise bei hochwertigen Matrixdruckern so stark gefallen, daß sich viele Heimcomputerbesitzer derartige Geräte leisten können. Eine ähnliche Tendenz ist neuerdings auch bei den Tintenstrahldruckern zu beobachten. Der hier vorgestellte SQ-2000 von Epson ist zwar noch immer relativ teuer, jedoch hat die Firma ebenfalls ein Modell der unteren Preisklasse angekündigt – die Tintenstrahldrucker gelangen allgemein in die Reichweite des ernsthaften Heimcomputerbenutzers.



Wie die vertrauten FX- und MX-Typen von Epson präsentiert sich auch der SQ-2000 in einem gefälligen, cremefarbenen Kunststoffgehäuse. Die Bedientasten, der Netzschalter und die Kammer für die Tintenpatrone sind gut zugänglich. Da der SQ-2000 vorzugsweise fürs kleine Büro, etwa die Buchführung, gedacht ist, verfügt er über einen extra breiten Wagen zur Aufnahme von großformatigem Papier. Auch sonst war Epson um Vielseitigkeit bemüht: Zu der serienmäßigen Friktions-Walzenführung gibt es gegen Aufpreis eine Cassette für Einzelblatteinzug sowie einen Traktoraufsatz.

Das Rechnerinterface ist im Grundpreis enthalten, aber noch nicht eingebaut; rückseitig ist ein Steckplatz dafür vorbereitet. Dort läßt sich nach den jeweiligen Erfordernissen ein paralleles Centronics, ein serielles RS232C- oder ein IEEE 488-Modul einschieben, das dann nur noch mit ein paar Schrauben zu sichern ist. Die Schnittstellenmodule enthalten einen 2-KByte-Buffer zum Zwischenspeichern der Daten vom Rechner; der SQ-2000 besitzt einen weiteren Buffer für das gleichzeitige Decodieren und Drucken der Daten aus dem Eingabespeicher. Außerdem ist noch ein Speicher für das Laden eines selbstdefinierten Zeichensatzes vorgesehen.

Pluspunkte vereint

Tintenstrahldrucker bieten gegenüber den Matrixdruckern und den Typenradgeräten viele Vorteile – sie verbinden die Schriftqualität der Typenradmodelle mit der Geschwindigkeit und Flexibilität der Matrixdrucker und arbeiten dazu noch äußerst geräuscharm. Das Tintenstrahlverfahren hat aber seine spezifischen Probleme, deren befriedigende Lösung erst in letzter Zeit gelungen ist: Die Tintenkanäle verstopften gerne, und beim Umgang mit älteren Geräten blieben die Finger selten sauber. Epson behauptet, mit dem neuen Tintensystem seien die Schwierigkeiten endgültig behoben. Die Tinte kommt beim SQ-2000 aus einer hermetisch geschlossenen Cassette,

die drei getrennte Kammern für die Spezialtinte, eine Reinigungsflüssigkeit und für Spülrückstände enthält. Der Tintenvorrat soll für drei Millionen Zeichen ausreichen.

Die erhöhte Zuverlässigkeit des neuen Epson-Systems wird dadurch erreicht, daß die Tintenkanäle beim Ein- und Ausschalten des Geräts automatisch gespült werden, außerdem periodisch während des Betriebs. Die Säuberungsaktion kann auch von Hand vorn am Gerät ausgelöst werden. Das Bedienfeld enthält außerdem die vertrauten Tasten ‚On Line', ‚Line Feed' und ‚Form Feed', ferner einen ‚Sheet feed'-Knopf für den Einzug von Einzelblättern. Drei grüne Kontrollampen zeigen an, ob das Gerät eingeschaltet ist, sich im On-Line- oder Off-Line-Betrieb befindet und ob Empfangsbereitschaft für Daten vom Rechner besteht.

Typisch Ink Jet

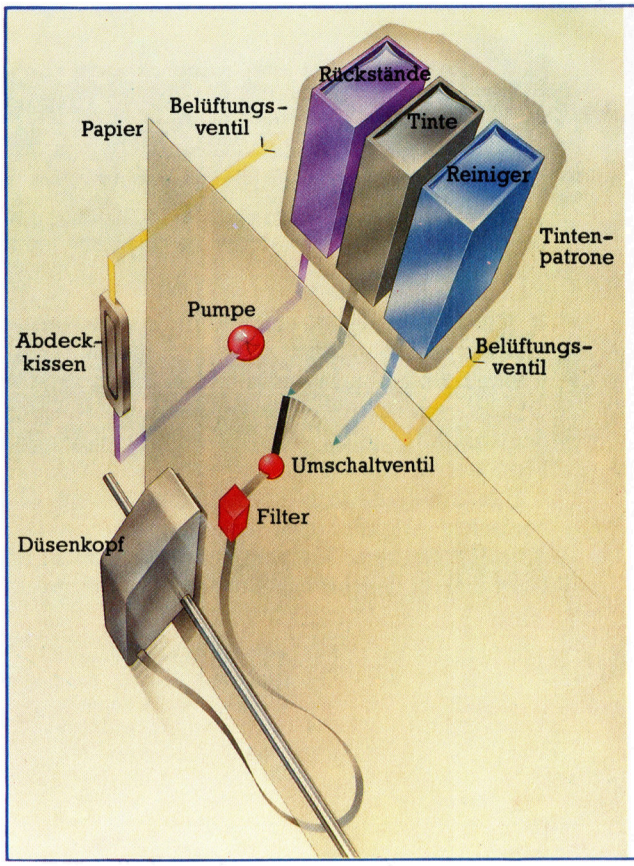
Besonders eindrucksvoll ist beim SQ-2000 die Vielfalt der Schriftarten. Die Grundeinstellungen sind ‚Draft' = Konzept und ‚NLQ'=Near Letter Quality =(Fast) Briefqualität. Bei Draft ist die Druckgeschwindigkeit höher als bei NLQ, auf Kosten der Schönheit. In beiden Betriebsarten können die Typen Pica, Elite und Roman gewählt werden, und zwar nach Wunsch gedehnt, komprimiert, kursiv, unterstrichen, fett, proportional und natürlich auch hoch- und tiefgestellt (Super/Subscript).

Wie ein Matrixdrucker erzeugt auch der Ink Jet jedes Zeichen als Punktmosaik, daher die Freiheit der Programmierung. Weil beim Ink Jet die einzelnen Punkte aber ineinanderfließen, erscheint das Schriftbild wesentlich eleganter als beim Matrixdrucker. Der SQ-2000 arbeitet mit einem rechtwinkligen Punktraster, das im Draft-Mode 15 × 24 Punkte enthält, im NLQ-Mode dagegen 29 × 24 Punkte, was die Schriftqualität zur Freude sämtlicher Brieffreunde und Geschäftspartner, die inzwischen längst wissen, daß wir einen Computer besitzen, deutlich erhöht. Matrixdrucker sind wirklich nicht mehr chic.

Im Druckkopf sind 24 Tintendüsen in zwei vertikalen Zwölferreihen angeordnet. Die beiden Reihen liegen leicht höhenversetzt nebeneinander, so daß benachbarte Punkte überlappen und die Mosaikstruktur nicht mehr ins Auge fällt. Wie mit einem Matrixdrucker lassen sich auch mit dem SQ-2000 im Bitmustermodus unmittelbar hochaufgelöste Bildschirmgrafiken wiedergeben. Die Druckgeschwindigkeit beträgt im Draft-Modusflotte 176 Zeichen/s, im NLQ-Modus aber auch noch beachtliche 105 Zeichen/s.

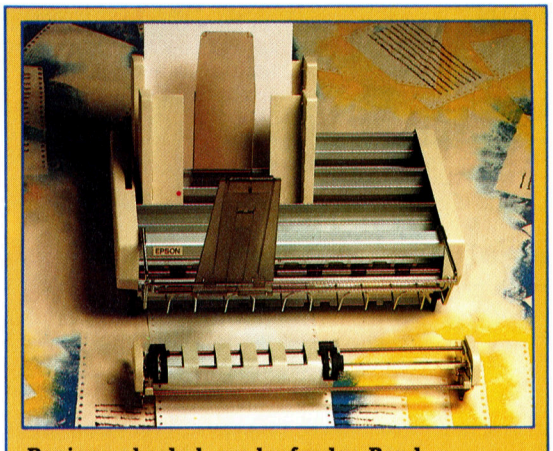
Neben dem Papierführungszubehör gibt es noch weitere Extras, unter anderem einen 32-KByte-Eingabebuffer zur Entlastung des Rechners. Die CPU lädt den Buffer sekundenschnell und ist dann für andere Arbeiten verfügbar, während der Bufferinhalt ausgedruckt wird. Außer den Standardschriftarten gibt es für das Gerät eine Anzahl von ROMs mit anderen Typen.

Wer seinen Microcomputer professionell einsetzt, ist mit diesem Drucker hervorragend bedient. Für den Geldbeutel der meisten Heimcomputerbenutzer liegt das Gerät preislich wohl zu hoch, aber auch bei diesem Jet wird die attraktive Economy-Klasse nicht lange auf sich warten lassen.



Säuberungsmaßnahmen

Das neue Tintenversorgungssystem von Epson arbeitet mit drei getrennten Tanks für Tinte, Reinigungsflüssigkeit und Rückstände, die bei der automatischen Säuberung anfallen. Über ein Umschaltventil können die Tintenkanäle in regelmäßigen Abständen durchgespült werden. Zu diesem Zweck fährt der Düsenkopf ganz nach links vor ein Abdeckkissen, durch das die Spülflüssigkeit mit Hilfe einer Pumpe in den Schmutzbehälter gesaugt wird.



Papiernachschub am laufenden Band

Standardmäßig ist der SQ-2000 für Einzelblatteinzug eingerichtet, auf Wunsch sind auch eine Traktorführung und ein Blattmagazin erhältlich. Aus den beiden getrennt anwählbaren Magazincassetten können automatisch einzelne Blätter eingeführt werden, etwa für Briefe oder Berichte mit individuellem Kopfaufdruck.

EPSON SQ-2000

ABMESSUNGEN
595 × 383 × 165 mm

SCHNITTSTELLEN
Centronics-Parallel-, RS232C- und IEEE488-Interface

GESCHWINDIGKEIT
Im Draft-Mode 176 Zeichen/s, im NLQ-Mode 105 Zeichen/s

PAPIERFÜHRUNG
Friktionswalze oder Traktor

WAGENBREITE
min. 140, max. 406 mm

DOKUMENTATION
Gute, kurzgefaßte Anleitung mit Ausführungsbeispielen für die Definition eigener Zeichen und den Grafik-Ausdruck.

STÄRKEN
Vereint die Flexibilität und Geschwindigkeit von Matrixdruckern mit der Schriftqualität von Typenradgeräten.

SCHWÄCHEN
Es ist unbedingt auf die richtige Papiersorte zu achten, sonst wird das Druckbild unsauber.

Katzenjammer

Bei der Programmierung unseres Spiels mit interaktiven Figuren legen wir nun die einzelnen Elemente des Spielablaufs fest.

Unser Ablaufdiagramm zeigt den gesamten Entscheidungsprozeß der Figurensteuerroutine. Es enthält nicht nur den Baum für die Objektsteuerung, sondern auch die Module für Spielablauf, „Objektbewußtsein" und die Wechselwirkungen mit anderen Figuren.

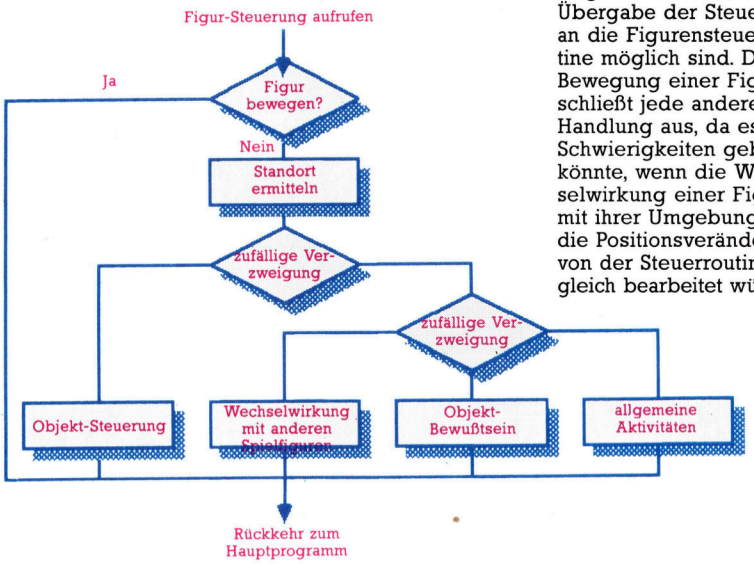
Der Spielaufbau ist bekannt: In einem bestimmten Stadium des Spiels ‚ißt' eine der Spielfiguren eine Fleischpastete und stirbt an Lebensmittelvergiftung. Das Spiel endet, wenn eine andere Figur beweisen kann, daß der Barkeeper Fred daran schuld war, da er für seine Pasteten Katzenfutter verwendet. Zur Lösung der Aufgabe muß die Figur die Konservenbüchse finden, sich das Opfer ansehen und die richtigen Schlüsse ziehen.

Das Programm speichert die notwendigen Informationen in vier Flags. Das erste kennen wir bereits: Die numerische Variable g wird in Zeile 5220 aktualisiert und enthält die Nummer der Spielfigur, die Fleischpastete gegessen hat. Zwei weitere Flags zeichnen den Tod einer Figur auf und geben Auskunft darüber, ob eine Spielfigur das Opfer gefunden hat. Das vierte Flag ist ein Array, dessen DIMension der Zahl der Spielfiguren entspricht und das die Initialisierung jedes Element zunächst auf Null stellt. Sobald eine Figur das Opfer entdeckt hat, wird das entsprechende Element auf 255 gesetzt. Das vollständige Listing drucken wir in Heft 69.

Über die Baumstruktur des Spielablaufs wird zunächst geprüft, ob die Figur nicht schon gestorben ist. Ist das der Fall, setzt die Routine das „Tod"-Flag auf die Figurennummer. Da die Routine nur ein Opfer akzeptiert, springt das Programm danach sofort zurück.

Ist das „Tod"-Flag jedoch noch nicht gesetzt, prüft die Routine, ob der Akteur gerade die Fleischpastete gegessen hat. Wenn der Wert von g mit der aktuellen Figurennummer übereinstimmt, wird weiterhin getestet, ob hier ein Todesfall vorliegt (das „Tod"-Flag ist dann größer als Null). Ist der Tod in keinem Fall einge-

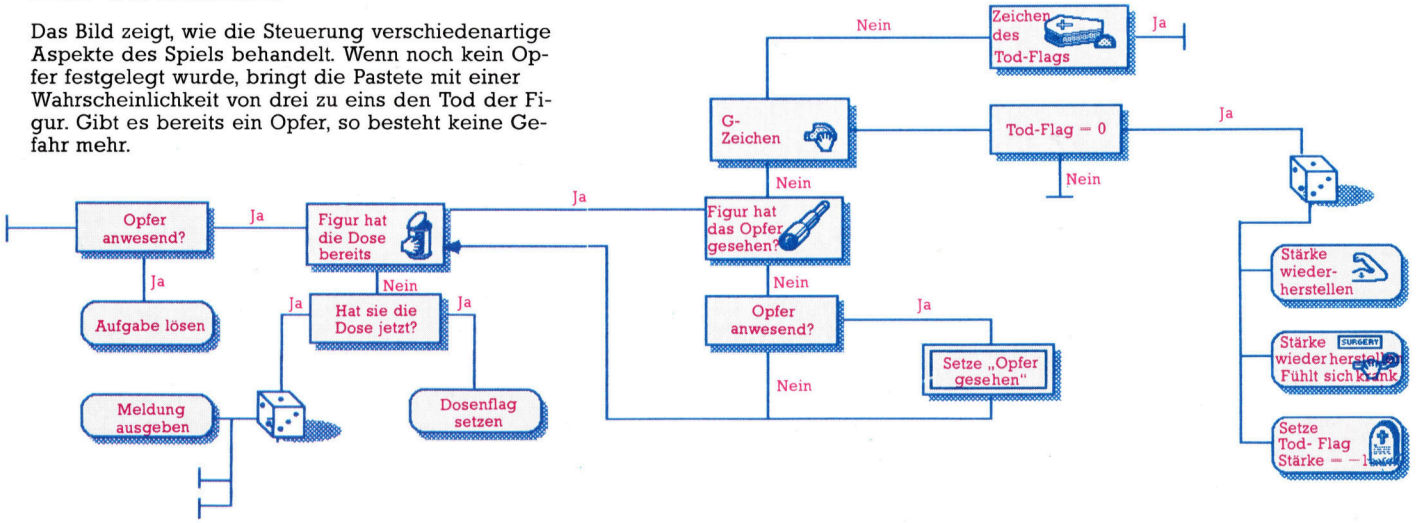
Verteilte Rollen



Das Ablaufdiagramm zeigt die Abläufe, die bei Übergabe der Steuerung an die Figurensteuerroutine möglich sind. Die Bewegung einer Figur schließt jede andere Handlung aus, da es Schwierigkeiten geben könnte, wenn die Wechselwirkung einer Figur mit ihrer Umgebung und die Positionsveränderung von der Steuerroutine zugleich bearbeitet würden.

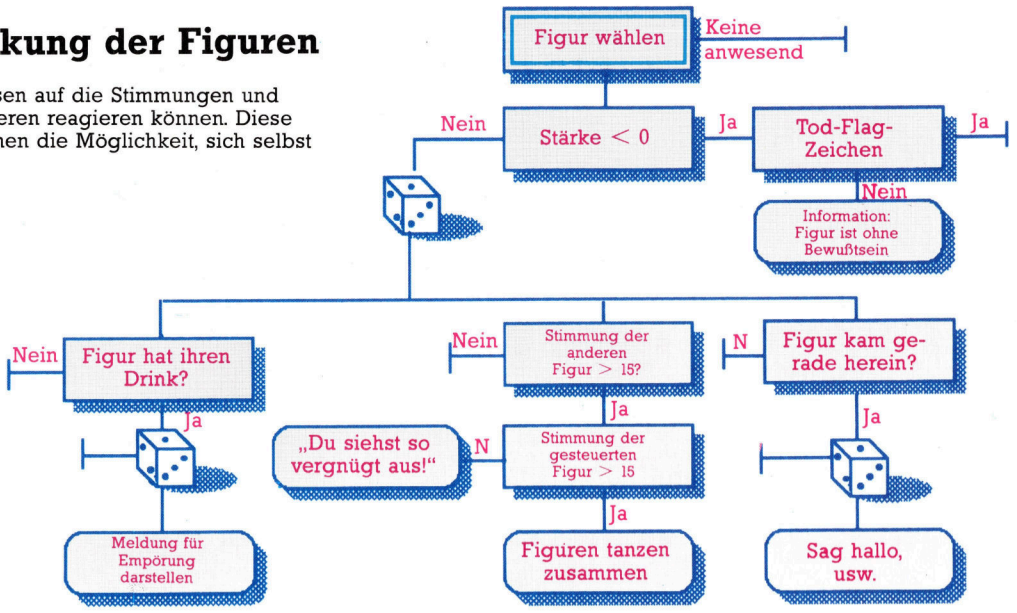
Ein Überblick

Das Bild zeigt, wie die Steuerung verschiedenartige Aspekte des Spiels behandelt. Wenn noch kein Opfer festgelegt wurde, bringt die Pastete mit einer Wahrscheinlichkeit von drei zu eins den Tod der Figur. Gibt es bereits ein Opfer, so besteht keine Gefahr mehr.



Wechselwirkung der Figuren

Unsere Figuren müssen auf die Stimmungen und Handlungen der anderen reagieren können. Diese Baumstruktur gibt ihnen die Möglichkeit, sich selbst auszudrücken.

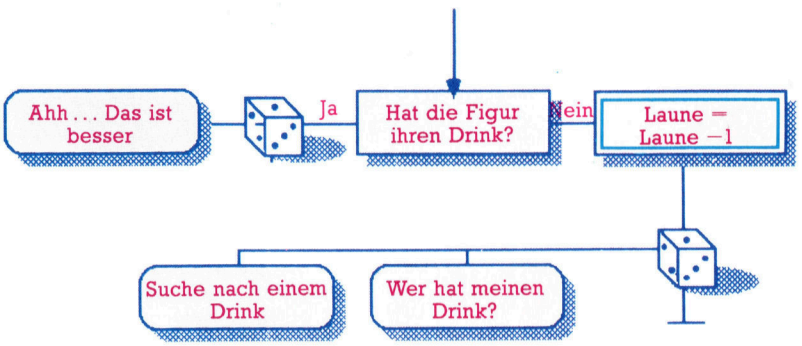


treten, verzweigt das Programm auf Zufallsbasis zu einer von drei Endsituationen: Bei den ersten beiden verursacht die Fleischpastete Magenschmerzen, wobei die Stärke der Figur, die die Zeile 5230 um 10 Punkte verringert hatte, wieder auf ihren ursprünglichen Wert gesetzt wird. Im dritten Fall wird die Stärke weiter (falls nötig) auf −1 verringert und das „Tod"-Flag auf diese Figurennummer gesetzt.

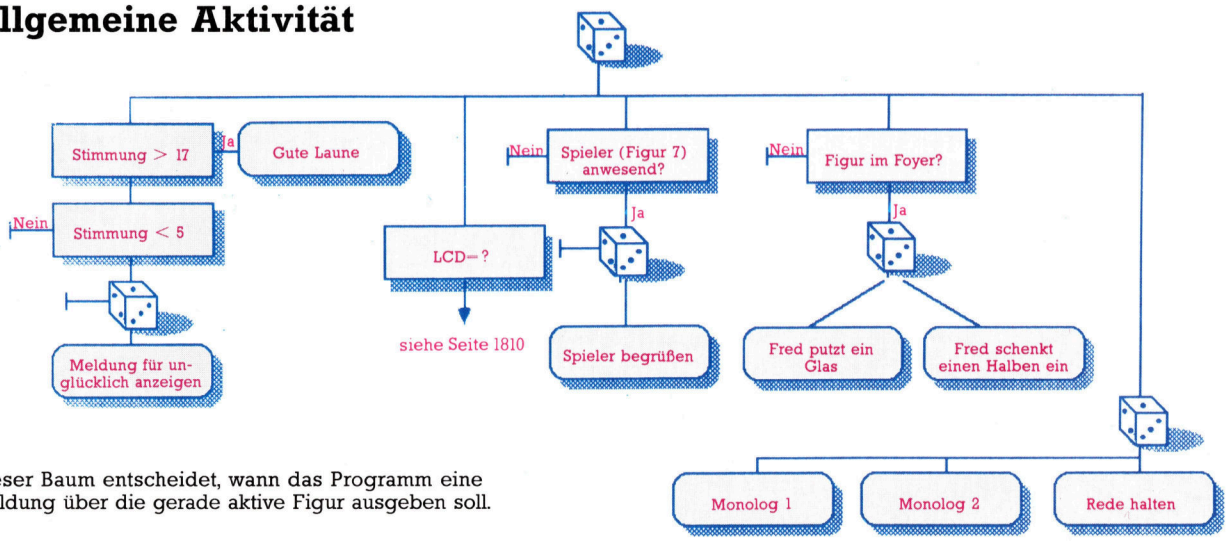
Wenn die Spielfigur nicht von der Pastete kostet, wird geprüft, ob die Bedingungen des Spielablaufs erfüllt sind. Über die verschiedenen Wege der Baumstruktur können Sie die Abläufe nachvollziehen. Beachten Sie, wie durch die zufällige Verzweigung in drei Richtungen entweder eine Meldung erscheint oder das Programm über die Endpunkte (links unten im Diagramm) in die Baumstruktur zurückgeht, um weitere Tests durchzuführen.

Objekt-Bewußtsein

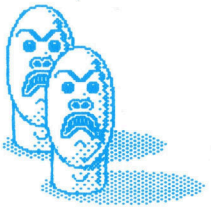
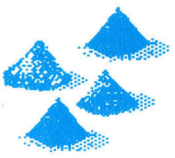
Hier sind weitere objekt-orientierte Routinen für die Baumstruktur der Figurenbewegung. Die Stimmung des Darstellers wird um 1 dekrementiert, wenn er nicht seinen eigenen Drink hat.



Allgemeine Aktivität



Dieser Baum entscheidet, wann das Programm eine Meldung über die gerade aktive Figur ausgeben soll.

	(1)	(2)	(3)	
T$()	 Pearls	 Carvings	 Spices	Beschreibung der Waren, die der Häuptling anbietet.
V1()	2gp	2gp	1gp	Marktwert der Waren bei Abfahrt
V2()	?	?	?	Marktwerkt der Waren bei Rückkehr
EQ(,)  1 Bag of Salt	0.5	0.5	1	
 1 Bale of Cloth	5	5	10	
 1 Jewel	3	3	6	Tauschwert der Waren
 1 Knife	2	2	4	

Die Neue Welt

Das Schiff hat nun die Neue Welt erreicht. Es sollte tunlichst vermieden werden, sich mit den Eingeborenen zu bekriegen, um erfolgreich Handel zu treiben.

Nach der glücklichen Ankunft in der Neuen Welt und dem Versuch, mit den in der Heimat eingekauften Waren Handel zu treiben, endet die eigentliche Fahrt. Sie können auf Gewinn nach Ihrer Rückkehr durch den Verkauf der fremden Handelsgüter hoffen. Doch die Gefahren sind längst nicht vorbei – noch kennen Sie die Eingeborenen nicht, die freundlich oder feindlich sein können. Deshalb müssen Sie vorsichtig sein und alles vermeiden, was zu einem Konflikt zwischen Mannschaft und Eingeborenen führen könnte.

Wenn die Hauptschleife abgebrochen wird, ruft Zeile 891 die Unterroutine ab Zeile 10000 auf, die Ihre Ankunft in der Neuen Welt beinhaltet. Sobald sich das Schiff der Küste nähert, werden Ihnen bewaffnete Eingeborene entgegenrudern, um das Schiff zu inspizieren. Sie müssen nun entscheiden, ob Sie das Risiko eines Angriffs auf Ihr Schiff in Kauf nehmen, oder aber das Feuer eröffnen und so einen Krieg provozieren.

Zeile 10015 überprüft das zweite Element des Vorrats-Arrays OA(2), das die Anzahl der Waffen an Bord enthält. Sind Waffen vorhanden, müssen Sie entscheiden, wie Sie sich verhalten wollen. Zeile 10022 wartet auf Ihre Antwort. Tippen Sie „Yes" und setzen Ihre Waffen ein, werden viele Eingeborene getötet. Es ist klar, daß die Überlebenden sich rächen und in der Nacht zurückkehren, um das Schiff in Brand zu stecken – womit das Spiel beendet wäre. Unnötige Aggression zahlt sich nicht aus.

Gebrauchen Sie die Waffen nicht, gibt es für die Eingeborenen natürlich keinen Grund für einen solchen Rückschlag, und Zeile 10026 verzweigt zu Zeile 10050. In diesem Abschnitt betreten die Eingeborenen friedlich das Schiff und begrüßen die Mannschaft und führen sie zum Indianerdorf. Dort lernen Sie den Häuptling kennen, der bereits mit Besuchern aus der Alten Welt Kontakt hatte. Am nächsten

Tag kann der Handel beginnen. Zuvor müssen jedoch noch einige Initialisierungen durchgeführt werden.

Die Preise für Perlen, Gewürze und Schnitzereien wurden beim Verlassen des Heimathafens bestimmt, doch haben sie sich inzwischen geändert.

Zum Handeln werden mehrere neue Arrays angelegt. In Zeile 60 wird T$() DIMensioniert, das die Bezeichnung der drei angebotenen Waren enthält: Perlen, Gewürze und Schnitzereien. Zeile 61 DIMensioniert V1(), das den Marktpreis der drei Güter beim Verlassen des Heimathafens beinhaltet.

Perlen und Schnitzereien wurden für jeweils zwei Goldstücke, Gewürze für ein Goldstück pro Gramm verkauft. In Zeile 62 wird V2() DIMensioniert, das die Preise der Waren bei Rückkehr enthält. Da der Marktpreis der Güter variiert, wird der endgültige Preis noch durch ein Zufallselement beeinflußt. Der Wert von V2(1) für Perlen wird in Zeile 62 auf zwei oder zweieinhalb Goldstücke gesetzt. Ähnlich wird in derselben Zeile der Preis für Schnitzereien und Gewürze festgelegt.

Fairer Tausch

Beim Handeln werden Waren nicht gekauft oder verkauft, sondern gegen entsprechende Waren getauscht. Der Häuptling bietet dem Kapitän für jedes Warenangebot eine bestimmte Menge an Perlen, Schnitzereien und Gewürzen als Gegenwert an.

In den Zeilen 64 bis 68 wird ein zweidimensionales Array DIMensioniert, dem dann die Werte der Tauschraten für jedes Handelsgut zugewiesen werden. Der erste Unterbereich entspricht den vier Gütern, mit denen Sie handeln: Salz, Stoffe, Messer und Juwelen. Der zweite Unterbereich repräsentiert die drei Waren der Eingeborenen, wobei die Tauschrate in drei separaten Programmzeilen für alle Güter bestimmt wird.

In Zeile 64 wird die Tauschrate für einen Beutel Salz, das erste Element des ersten Unterbereichs, bestimmt. Perlen sind das erste Element im zweiten Unterbereich, und EQ(1,1) wird ein Wert zugewiesen, der das Tauschverhältnis von Salz zu Perlen festlegt. Wird EQ(1,1) auf 0,5 gesetzt, entspricht die Tauschrate einer halben Perle für einen Beutel Salz. Ähnlich wird EQ(1,2) ein Wert für das Verhältnis von Element 1 im Unterbereich 2 zugewiesen – Salz und Schnitzereien. Wird EQ(1,2) auf 0,5 gesetzt, erhält man für zwei Beutel Salz eine Schnitzerei. EQ(1,3)=1 bestimmt die Rate für ein Gramm amerikanischer Gewürze gegen einen Beutel mit Salz.

Die Tauschrate für das zweite Element im ersten Unterbereich – ein Ballen Stoff – wird in Zeile 66 bestimmt. Ein Ballen ist fünf Perlen, fünf Schnitzereien oder zehn Gramm Gewürze wert. Zeile 67 befaßt sich mit dem dritten Element des ersten Unterbereichs, Juwelen, die jeweils drei Perlen, drei Schnitzereien oder sechs Gramm Gewürze wert sind. Messer, das vierte Element der Angebotspalette, werden in Zeile 68 gehandelt.

In Zeile 69 wird ein Array AO(3) DIMensioniert, um die Mengen an Perlen, Schnitzereien und Gewürzen zu speichern, die man beim Handeln erwirbt.

Die Programmteile, die diese Arrays verwenden, entwickeln wir im nächsten Artikel.

Modul 11: Die Ankunft

Dimensionierung der Handels-Arrays

```
60 DIMT$(3):T$(1)="PEARLS":T$(2)="CARVINGS":T$
(3)="SPICES"
61 DIMV1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIMV2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+
(INT(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIMAO(3)
```

Ergänzungen am Hauptprogramm

```
890 REM ARRIVAL AT NEWWORLD
891 GOSUB10000
```

Ankunfts-Unterroutine

```
10000 REM ARRIVAL AT NEWWORLD
10001 PRINTCHR$(147):GOSUB9200
10005 S$="YOU ARRIVE AT THE NEW WORLD*":GOSUB9100
10006 PRINT:GOSUB9200
10007 S$="AS YOU APPROACH THE SHORE*":GOSUB9100
10009 S$="NATIVES PADDLE OUT TO MEET YOU*":GOSUB91
00
10010 PRINT:GOSUB9200
10015 IFOA(2)=0THEN10050
10017 S$="THEY LOOK FIERCE AND ARE ARMED!!*":GOSUB
9100
10018 PRINT:GOSUB9200
10020 S$="DO YOU OPEN FIRE? (Y/N)*":GOSUB9100
10022 INPUTI$:I$=LEFT$(I$,1)
10024 IFI$<>"N"ANDI$<>"Y"THEN10022
10026 IFI$="N"THEN10050
10028 PRINT:GOSUB9200
10030 S$="MANY NATIVES ARE KILLED*":GOSUB9100
10032 S$="BUT DURING THE NIGHT*":GOSUB9100
10034 S$="OTHERS RETURN*":GOSUB9100
10036 S$="AND BURN YOUR SHIP!!!!*":GOSUB9100
10038 PRINT:PRINT:GOSUB9200
10040 S$="        GAME OVER*":GOSUB9100
10042 END
10044 GOTO10042
10050 S$="THEY TAKE YOU TO MEET THEIR*":GOSUB9100
10052 S$="CHIEF. HE HAS MET YOUR RACE*":GOSUB9100
10054 S$="BEFORE AND IS FRIENDLY.*":GOSUB9100
10056 GOSUB9200
10058 S$="THE CREW ARE FED AND RESTED*":GOSUB9100
10060 PRINT:GOSUB9200
10062 S$="THE NEXT DAY TRADING BEGINS*":GOSUB9100
10064 PRINT:GOSUB9200
10066 S$=K$:GOSUB9100
10068 GETI$:IFI$=""THEN10068
10069 RETURN
```

BASIC-Dialekte

Spectrum: Ersetzen Sie V1() durch B(), V2() durch D(), EQ(,) durch Q(,) und AO() durch E() im gesamten Programm. Führen Sie außerdem folgende Änderungen durch:

```
60     DIM T$(3,9)
10001 CLS:GO SUB 9200
10022 INPUT I$:LET I$=I$( TO 1)
10068 LET I$=INKEY$:IF I$= "" THEN GO TO 10068
```

Acorn B:
Ändern Sie das Programm wie folgt:

```
10001 CLS:GOSUB 9200
10068 I$=GET$
```

Computer-Casino

Wir stellen Ihnen einige Wege zur Chancenerhöhung beim Glücksspiel vor, bei denen Künstliche Intelligenz zum Tragen kommt.

Das Casino zieht ganz unterschiedliche Typen von Spielern an – Profis am Spieltisch sind eine exklusive Minderheit. Obwohl der Veranstalter bei jedem Spiel einen sicheren Gewinn macht, gibt es einige Methoden, den Profit des Spielers zu steigern. Leider beruhen die meisten Systeme auf einem festen Schema beim Setzen, das von den Angestellten des Casinos schnell erkannt wird. Bei intensiver Anwendung dieser Methoden wird der Spieler meist höflich, aber bestimmt aus dem Saal gebeten.

Sie sollten äußerst mißtrauisch werden, wenn Ihnen jemand von einem angeblich „todsicheren" Spielsystem vorschwärmt. Dennoch gibt es Verfahren, die unter speziellen Bedingungen tatsächlich funktionieren. Eines davon erfand Mitte des 18. Jahrhunderts Jean Le Rond d'Alembert. Für sein System müssen die Gewinnchancen mindestens 1:1 oder besser stehen. Das ist zwar selten, kann bei einem guten Vorhersageprogramm jedoch eintreten. Als Roulette noch ohne Bevorzugung des Casinos gespielt wurde, hat d'Alembert mit diesem Verfahren jedenfalls ein hübsches Sümmchen verdient.

Heutzutage kann man ohne die Hilfe Fortunas beim Roulette kaum noch reich werden. Spiele und Wetten aber, bei denen ein Vorhersageprogramm Chancen über 50 Prozent prophezeit, können mit d'Alemberts System profitabel werden.

- Zuerst fünf Einheiten setzen (etwa 5 Mark)
- Nach jedem Gewinn den Einsatz um eine Einheit vermindern.
- Nach jedem Verlust den Einsatz um eine Einheit erhöhen.
- Bei Verlust aller fünf Einheiten wieder erneut mit fünf Einheiten beginnen.

Die komplexe Arbeitsweise der Methode läßt sich am Computer gut testen. Unser Programm zeigt, was Sie damit erreichen können. Die RND-Funktion im BASIC stellt die Zufallszahlen für das Testprogramm bereit.

Durch Veränderung der Wahrscheinlichkeit eines Treffers kann man die Leistung des Systems für unterschiedliche Chancen prüfen. Auch selbstentwickelte Strategien können Sie auf diese Weise ohne finanzielle Verluste ausprobieren. Der große Vorteil der Simulation liegt darin, daß man der „Spielhölle" fernbleiben kann – die Brieftasche mit dem geliebten Inhalt gerät also nicht in Gefahr.

Gewinne maximieren

Das Wesentliche der d'Alembert-Methode ist die Maximierung der zu erwartenden Gewinne. Wenn die Chancen allerdings gegen Sie stehen, maximiert sie auch die Verluste!

Beim Blackjack gibt es im Gegensatz dazu eine Methode, mit der Sie auf Kosten des Casinos verdienen können: Auf ein gutes Blatt wird hoch gewettet, bei schlechten Karten setzt man nur den vorgeschriebenen Minimaleinsatz. Allerdings muß man gute von schlechten Blättern unterscheiden können. Dazu ist es nötig, sich an die bereits ausgespielten Karten zu erinnern – ein „Counter" werden, wie es im Fachjargon heißt.

Die Grundlage aller Blackjack-Systeme geht auf Edward Thorp zurück, der in den frühen 60er Jahren in Las Vegas sein Glück machte und die von ihm angewandte Technik im Buch „Beat the Dealer" auch veröffentlichte. Vorsichtig eingesetzt kann sein System Ihre Chancen verbessern, es gibt aber auch zwei Nachteile:
1. Die Gewinnmarge liegt im Bereich von nur 0,5 Prozent vom Umsatz
2. Casinoangestellte sind dazu angehalten, auf „Counter" zu achten.

Beides zusammen macht Probleme: Der Systemspieler kann leicht erkannt werden, und verfeinerte Systeme erfordern Zurückhaltung, die wiederum den Gewinn reduziert. Auf die Dauer ist ein Gewinn von vier Mark bei einem Einsatz von 1000 Mark nicht gerade üppig.

Hochinteressant sind die mit Hilfe der Künst-

lichen Intelligenz in den letzten zehn Jahren entstandenen Expertensysteme. Sie haben gewisse Ähnlichkeit mit einem menschlichen Experten, etwa die Fähigkeit des Umgangs mit unsicheren Einflüssen und ungenau definierten Situationen, die durch den Einsatz von Logik oder durch die Begründung mit Wahrscheinlichkeiten gemeistert werden.

Bei der Entwicklung eines Expertensystems, beispielsweise für die Entdeckung von Erz-Lagerstätten oder zur Diagnose von Krankheiten, konsultiert der Programmierer zuerst einen Fachmann. Sein Wissen wird in Programmform gebracht und für die Computerverarbeitung aufbereitet. Dieser Vorgang läßt sich unverändert auf die Entwicklung eines Pferdewett-Systems übertragen.

Dem Programmierer Stephen F. Smith von der Carnegie-Mellon Universität in Pittsburgh ist es gelungen, einem LS-1 genannten Lernprogramm das dort beliebte Kartenspiel Draw Poker beizubringen.

Draw Poker wird mit zwei oder mehr Teilnehmern gespielt, die jeweils fünf Karten erhalten. Die Spieler können dann entweder tauschen, einsetzen oder passen. Beim Tausch kann jeder Spieler bis zu drei Karten ablegen und sie gegen drei verdeckt gegebene neue ersetzen. Durch Passen wird eine Runde beendet. Der Einsatz, den jeder Spieler auf seine Karten macht, wandert in einen gemeinsamen ‚Pot'. Will einer der Spieler ‚Sehen', werden alle Karten gezeigt. Den Pot gewinnt der Spieler mit den besten Karten.

LS-1 ist ein Lernsystem, daß mit einem sogenannten „genetischen" Algorithmus arbeitet. Die Hauptschleife sieht so aus:

1. Erzeuge eine zufällige Anfangspopulation von Regeln.
2. Bewerte alle Regeln. Zeige sie auf dem Bildschirm, wenn ihre durchschnittliche Erfolgsquote hoch genug ist.
3. Berechne andernfalls die Erfolgsquoten der einzelnen Regeln mit der Formel p=e/E, wobei e die Einzel- und E die Gesamttrefferquote aller Regeln ist.
4. Erzeuge die nächste Generation von Regeln durch Auswahl nach den in Schritt 3 berechneten Erfolgswahrscheinlichkeiten und durch Anwendung spezieller „genetischer" Operatoren. Danach weiter mit Schritt 2.

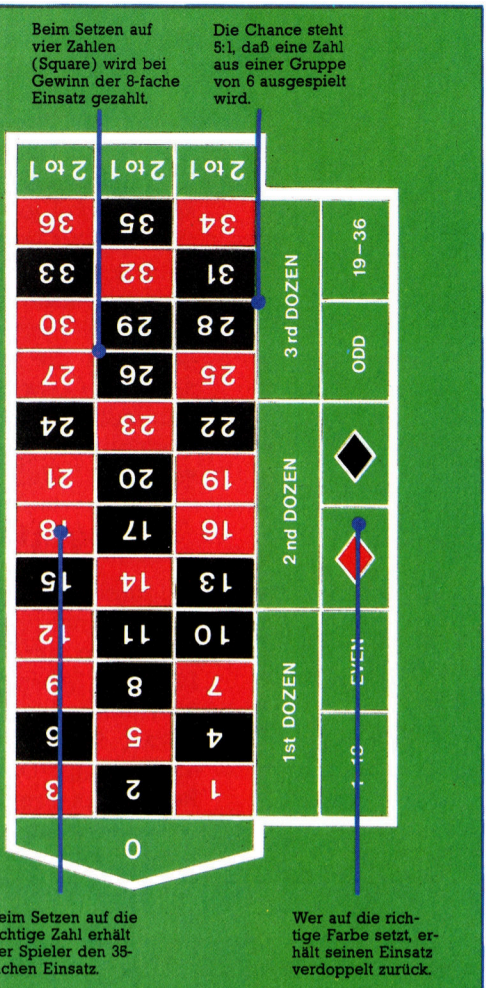
Jedes Durchlaufen dieser Schleife entspricht einer Regel-Generation.

Inversion und Mutation

Mit den „genetischen Operatoren" sind die „Inversion", die „Mutation" und das „Crossover" gemeint. Crossover ist ein Vorgang, bei dem Informationen aus zwei verwandten Strukturen zu einer neuen Struktur verbunden werden. Die neue Struktur, sozusagen das „Kind", wird als neuer Prüfling dem weiteren Test unterzogen. Die Inversion ordnet die Informationen

Nichts geht mehr . . .

Beim Roulette kann man besonders leicht erkennen, daß die Chancen immer zugunsten des Casinos stehen. Das Feld der amerikanischen Roulette-Version gibt die nebenstehende Zeichnung wieder. Die Spieler können ihre Chips entweder auf einzelne Zahlen, Zahlengruppen oder auf Farben setzen. Die Wahrscheinlichkeit für das Auftreten einer bestimmten Zahl beträgt 36:1. Ausgezahlt wird bei einem Volltreffern aber nur der 35-fache Einsatz, das Casino macht also im Schnitt 2,7 Prozent Gewinn. Bei den einfachen Chancen (Gerade/Ungerade, Rot/Schwarz, 1-18/19–36) behält das amerikanische Casino bei Zero den halben Einsatz. Der Profit sinkt dabei auf nur 1,35 Prozent.



einer Regel neu an, und die Mutation sorgt für zufällige Regeländerungen.

Beim Draw Poker waren die Leistungen des Programms beeindruckend. LS-1 wurde mit einem herkömmlichen Pokerprogramm getestet – ein menschlicher Gegner hätte die 40 000 Testrunden wohl auch kaum durchgestanden. Bei jeder Wett-Entscheidung gab es vier Möglichkeiten: hoher Einsatz, niedriger Einsatz, neue Karten oder Passen. Nach einer Anfangsphase, in der LS-1 nur über eine Reihe von zufällig erzeugten Regeln verfügte, steigerte sich die Leistung so sehr, daß das gegnerische Computerprogramm verbessert werden mußte. Die neue Version war ein härterer Gegner, aber LS-1 gewann schließlich in neun von zehn Fällen.

Expertensystem auf Trab

Ein weiteres Pilotprojekt, ebenfalls mit „genetischem" Ansatz, wurde am Beispiel von Rennergebnissen der Sommersaison 1982 durchgeführt. Die Daten bezogen sich auf 153 Pferde in 51 Hindernisrennen. Die Basisdaten wurden geteilt: 99 Ergebnisse dienten ausschließlich als Trainings-Datensatz für die Erzeugung der Regeln. An den restlichen 54 Ergebnissen wurden die gefundenen Regeln dann überprüft. Jedes Pferd wurde mit einem System von 18 Variablen bewertet, etwa

SPEED	Schnelligkeit des Pferdes
RATING	Schnellste, zweitschnellstes, drittschnellstes Pferd usw.
FC	Position in den Pferdewetten
LAST	Plazierung im letzten Rennen
LASTONE	Plazierung im vorletzten Rennen
LASTTWO	Plazierung im vor-vorletzten Rennen
DAYS	Tage seit dem letzten Start
SPOTFORM	Tagesform nach Bewertung des „Daily Mirror"
GOING	Zustand der Rennbahn
WT	Jockeygewicht

Mit Ergebnissen von der Rennbahn kam das Programm zu Regeln wie etwa (SPEED >60) und (WT>(SPEED*2,18)), mit denen wahrscheinliche Gewinner von den Verlierern getrennt wurden. Diese Regeln werden zur Erzeugung von einer Art „Fingerabdruck" genutzt: Falls etwa Regel 1 und 3 zutreffen, Regel 2 und 4 jedoch nicht, ergibt sich der binäre „Fingerabdruck" 0101 (dezimal 5). Die Indexzahl 5 führt dann zur Position 5 eines Verzeichnisses, in dem Informationen über dieses spezielle Zusammentreffen bestimmter Regeln gespeichert sind.

Bei der Konfrontation mit unbekannten Daten zeigte sich, daß die Computer-Regeln in 80 Prozent der Fälle zu richtigen Aussagen führten. Natürlich kann man schon in 72% aller Fälle Recht behalten, indem man einfach „Nein" sagt – schließlich verliert die Mehrzahl der Pferde. Es zeigt sich aber, daß die Regeln ein sehr wirkungsvolles Filtersystem sind. Das Programm sagte nur vier Gewinner voraus – alle haben auch in Wirklichkeit gewonnen. Das kann als gutes Zeichen gewertet werden, denn Selektivität ist die Grundlage des wissenschaftlichen Ansatzes beim Wetten. Korrekte Vorhersagen für jedes Rennen sind nicht zu erwarten: Viel besser ist es, nur sichere Sieger in einem schwachen Feld zu erkennen, und dann zuzuschlagen. Das System hatte aber noch einen zweiten Erfolg: Unter den vorhergesagten dreizehn besten Pferden fanden sich zehn tatsächliche Sieger und nur drei Verlierer.

Eine Falschaussage

Unter den vom Programm als Verlierer bezeichneten 41 Pferden gab es nur fünf Sieger, in der als „klare Außenseiter" eingeschätzten Gruppe tauchte sogar nur eine einzige Falschaussage auf – ein klarer Beweis für die guten Leistungen des Systems beim Erkennen chancenloser Teilnehmer.

Man sieht, daß die Künstliche Intelligenz mit ihren Lernsystemen auch beim Pferderennen neue Wege eröffnen kann. Die umfangreiche Literatur zum Thema verspricht jedoch noch eine Vielzahl bisher ungenutzter Möglichkeiten. Lernfähige Algorithmen, Mustererkennungsmethoden und vieles mehr lassen auf interessante Entwicklungen hoffen.

EINSATZ-SIMULATOR

```
 10 REM ******************************
 15 REM ** D'ALEMBERT'S METHOD:     **
 20 REM ******************************
 75 @%=4
100 PRINT'"Staking simulator:"
101 REPEAT
110   PRINT
120   PRINT "Please give probability of success ";
122   INPUT PS
125   UNTIL PS>0 AND PS<=1
130 INPUT "How many trials ", T
140 N=0
150 GAIN=0: LOSS=0
155 LAST=0: ST=-99
160 W=0
190 REM -- Main Loop:
200 REPEAT
202   N=N+1
210   GOSUB 1000 : REM compute stake
212   PRINT N;"  ";
220   IF RND(1) <= PS THEN S=1 ELSE S=0
230   IF S THEN GOSUB 1500 ELSE GOSUB 1600
240   LAST=S
250   UNTIL N>=T
300 PRINT: PRINT "Total won = ";GAIN-LOSS
330 PRINT "Proportion of successes: ";W/T
360 END
999 :
1000 REM -- Stake Decision Routine:
1010 REM can be altered for experiments:
1020 IF LAST THEN ST=ST-1 ELSE ST=ST+1
1030 IF ST<1 THEN ST=5
1040 RETURN
1050 :
1500 REM -- Success Routine:
1510 GAIN=GAIN+ST
1520 PRINT "Won  ";ST;TAB(15);"Winnings = ";GAIN-LOSS
1525 W=W+1
1550 RETURN
1560 :
1600 REM -- Failure Routine:
1610 LOSS=LOSS+ST
1620 PRINT "Lost ";ST;TAB(15);"Winnings = ";GAIN-LOSS
1650 RETURN
1660 :
```

Tor zur Außenwelt

In dieser zweiteiligen Artikelfolge untersuchen wir das Ein- und Ausgabesystem des Commodore 64. Wir sehen uns dabei zunächst die Peripherieschnittstellen des Computers an.

Unser umseitiges Photo zeigt die Ausgänge des Commodore 64. Neben den Buchsen für Cassette, Audio/Video und TV stehen dem Programmierer drei E/A-Schnittstellen zur Verfügung. Von links nach rechts sind das die Erweiterungsschnittstelle (auch Cartridge-Port genannt), der serielle Ausgang (serieller Bus) und der User Port.

● **Der serielle Ausgang:** Über diese sechspolige DIN-Buchse werden serielle Commodoredrucker und das Diskettenlaufwerk 1541 angeschlossen. Bei jeder Gerätenummer (Device-Number) außer 2 bezieht sich der OPEN-Befehl immer auf diesen Ausgang. OPEN2,8,2,„DATEINAME" spricht beispielsweise das Diskettenlaufwerk über den seriellen Ausgang an und eröffnet dort eine Datei. Die serielle Schnittstelle sollte möglichst nur mit Commodoregeräten und auch nur über BASIC oder die E/A-Routinen eingesetzt werden.

Es gibt jedoch auch Schnittstellenkarten, die die seriellen Signale dieses Ausgangs in parallele IEEE-Daten umwandeln. Damit lassen sich etwa Peripheriegeräte des PET (beispielsweise die Diskettenstation 4040) bei entsprechendem Bedarf einsetzen.

● **Der User Port:** Diese flache 24polige Steckleiste kann für serielle wie auch für parallele Datenübertragung eingesetzt werden. Zum Beispiel läßt sich darüber ein Drucker außerhalb des Commodorestandards an den C 64 anschließen. Der Drucker wird dabei als RS232-Gerät gesteuert.

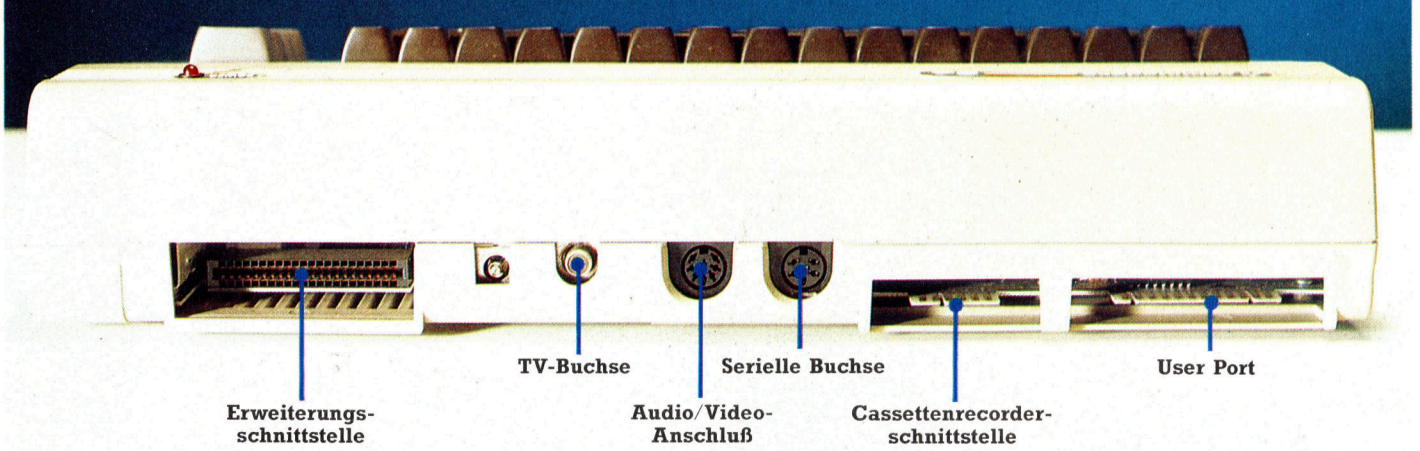
Die Anwenderschnittstelle kann auch für die parallele Acht-Bit-Datenübertragung eingesetzt werden. Da das OS keine Treiberroutinen für die parallele Datenübertragung besitzt, müssen Sie sich jedoch Ihre eigenen Treiber (Maschinencodeprogramme, die Peripheriegeräte steuern) schreiben. In der nächsten Folge stellen wir ein Programm namens „Parawedge" – ein interruptgesteuertes, paralleles Datenübertragungsprogramm – vor. Das Programm kann für die Übermittlung eines Speicherblockes eingesetzt werden (z. B. ein BASIC-Programm von einem Commodore 64 zum anderen).

● **Die Erweiterungssteckleiste:** Diese 44-polige Steckleiste bietet Zugang zu allen Systembussen und den Daten- und Adreßleitungen des Commodore 64. Außer Spielcartridges werden hier auch die parallelen IEEE-Cartridges angeschlossen, mit denen der C 64 Peripheriegeräte des PET betreiben kann. Über diese Steckleiste werden auch Geräte oder Software angeschlossen, die die Steuerung der Maschine fast vollständig übernehmen.

E/A-Befehle

Die OS-Routinen für Ein- und Ausgabe (im Englischen „Kernel" genannt) sind Maschinencodeprogramme, die von E/A-Befehlen des BASIC (OPEN, CLOSE, GET#, PRINT# etc.) aufgerufen werden. Auf dem Commodore sind diese Routinen dem Maschinencodeprogram-

Die E/A-Schnittstellen des Commodore 64



mierer leicht zugänglich. Ein kurzes Maschinencodeprogramm wird Ihnen zeigen, wie LOAD gesteuert wird.

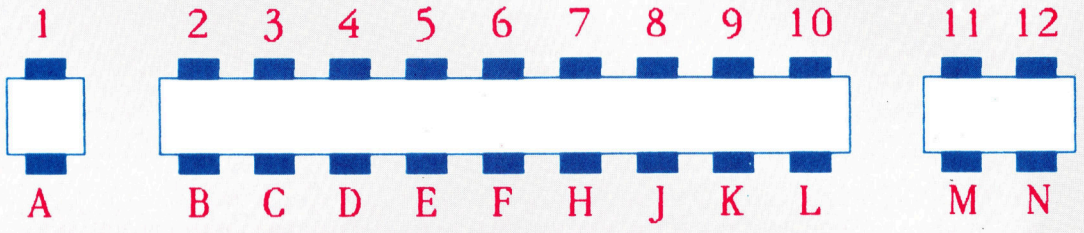
Wir haben weiterhin untersucht, welche Möglichkeiten dem C 64 über die RS232-Schnittstelle zur Verfügung stehen, und wie damit ein Modem betrieben wird.

Auf die Schnelle

Die Systemroutinen lassen sich jedoch nicht für alle E/A-Vorgänge einsetzen. Für Geräte, die nahe beeinander stehen, benötigt man Parallelverbindungen mit hohen Übertragsgeschwindigkeiten statt der seriellen Kommunikation, die eher für weiter entfernte Geräte geeignet ist. Da der C 64 keine Systemroutinen für die Steuerung des User Ports besitzt, müssen Sie die beiden 6526 CIAs (Complex Interface Adaptors) selbst programmieren. In der nächsten Folge zeigen wir anhand einer Assemblerroutine für den Acht-Bit parallelen Datentransfer, wie Programme dieser Art aussehen. Die Routine ermöglicht es dem Computer, über den User Port Daten zu lesen oder zu senden, während gleichzeitig ein BASIC-Programm abläuft. Dieser Ablauf wird durch die Interruptsteuerung des Schreib-/Lesecodes möglich.

Ein Beispiel für die IRQ-Anforderung – den „Programmkeil" – haben wir schon früher gezeigt. Hier arbeiten wir mit NMIs, da die Flagleitung (die zweite Interruptleitung des 6510) des User Ports zur Erzeugung eines (nichtmaskierbaren) NMI-Interrupts eingesetzt werden kann.

User Port



Oben

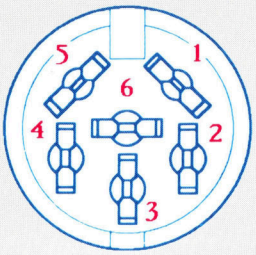
Kontakt	Typ	Allgemeine Bedeutung
1	GND	Erde
2	-	+5 Volt mit 100 mA Maximum
3	-	System RESET
4	CNT 1	CIA # 1 serieller Ausgang – Zähler
5	SP 1	CIA # 1 serieller Ausgang
6	CNT 2	CIA # 2 serieller Ausgang – Zähler
7	SP 2	CIA # 2 serieller Ausgang
8	PC 2	Handshakeleitung von CIA #2
9	-	Diese Leitung ist mit ATN des seriellen Ausgangs verbunden
10	-	9V AC +Phase
11	-	9V AC –Phase
12	-	Erde

Anmerkungen:
1) 3-Leitungen = Xon/Xoff Protokoll
X-Leitungen = CTS/RTS Protokoll
2) (*) Diese Leitungen können nicht nur über die RS232 E/A-Routinen des OS adressiert werden, sondern lassen sich auch direkt dem User Port zuordnen. Sie werden für die Steuerung von externen Geräten, wie Bufferboxen oder dem Bodenroboter unserer Bastelserie, eingesetzt.

Unten

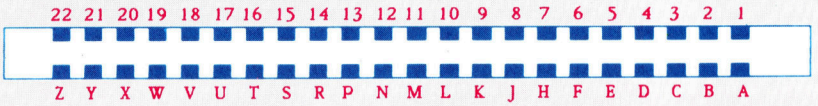
Kanal	Typ	RS232 Funktion (*)	Eingesetzt in 3-/x-Ltg.
A	GND	Erde	Beide
B	FLAG2	Empfangsdaten -ein	Beide
C	PB0	Empfangsdaten -ein	Beide
D	PB1	Sendeteil schalten (RTS)-aus	Beide (hoch in 3–1)
E	PB2	Sendegerät bereit (DTR)-aus	Beide (hoch in 3–1)
F	PB3	akustisches Signal (RI)-ein	–
H	PB4	Signal empfangen (DCD)-ein	Nur X-Leitung
J	PB5	Nicht eingesetzt	–
K	PB6	sendebereit (CTS)-ein	Nur X-Leitung
L	PB7	Empfangsgerät bereit (DSR) -ein	Nur X-Leitung
M	PA2	Sendedaten -aus	Beide
N	GND	Masse	Beide

Serielle Buchse



Kanal	Typ	Bemerkung
1	SERIAL SRQIN	Sendeanforderung
2	GND	
3	SERIAL ATN I/O	"Attention"-Signal für Geräte am Port
4	SERIAL CLK I/O	Taktgeber für seriellen Ausgang
5	SERIAL DATA I/O	Übertragungsltg. für ein Datenbit
6	RESET	Leitung für Hardware-RESET

Erweiterungs-schnittstelle



Oben

Kanal	Typ	Bemerkung
1	GND	
2	+5v	
3	+5v	
4	IRQ	Interruptanforderung
5	R/W	Lesen/Schreiben EIN
6	Dot clock	
7	I/O 1	
8	GAME	
9	EXROM	
10	I/O 2	
11	ROML	
12	BA	
13	DMA	
14	D7	
15	D6	
16	D5	
17	D4	8 Datenleitungen
18	D3	
19	D2	
20	D1	
21	D0	
22	GND	

Unten

Kanal	Typ	Bemerkung
A	GND	
B	ROMH	
C	RESET	Hardware-RESET
D	NMI	Nicht-mask. Interrupt
E	S 02	
F	A15	
H	A14	
J	A13	
K	A12	
L	A11	
M	A10	
N	A9	
P	A8	16 Adreßleitungen
R	A7	
S	A6	
T	A5	
U	A4	
V	A3	
W	A2	
X	A1	
Y	A0	
Z	GND	

Die E/A-Routinen in Aktion

Der Ein-/Ausgabeteil des OS besteht aus einer Reihe von Subroutinen, die vom BASIC oder vom Maschinencode aus aufgerufen werden können. Das E/A-ROM reicht von $E000 bis $FFFF, doch werden die eigentlichen Routinen über die Pointer einer „Sprungtabelle" aufgerufen, die sich an der Obergrenze des Speichers befindet. Durch die Sprungtabelle läßt sich der Code auch auf anderen Commodoregeräten einsetzen, da sich die Adresse der Tabelle nie ändert – auch wenn die E/A-Routinen umgeschrieben werden.

Um E/A-Routinen einsetzen zu können, brauchen Sie weitere Informationen, da die Routinen nur dann funktionieren, wenn alle Mechanismen berücksichtigt sind.

Der folgende Code ist ein typisches Beispiel für den Einsatz der E/A-Routine LOAD bei der Ausführung eines verschiebbaren LOAD. Speichern Sie zunächst die ASC-Codes (Commodores ASCII-Version) des Dateinamens in aufeinanderfolgende Adressen und rufen Sie dann diesen Code auf:

```
LDA #$01     Logische Dateinummer (lfn)
LDX #$08     Devicenummer (Diskette)
LDY #$00     Sekundäradresse (#$00 ergibt
             verschiebbares Laden)
JSR $FFBA    E/A-Routine SLFS (lfn und Sekun-
             däradresse anlegen)
LDA #$0A     Länge Dateiname (z. B. dez. 10)
LDX PLO      Lo-byte Zeiger auf die Startadresse
             des Dateinamens
LDY PHI      Hi-byte Zeiger auf die Startadresse
             des Dateinamens
JSR #FFBD    E/A-Routine SETNAM (Dateiname
             anlegen)
LDA #$00     Laden = #$00 / Verif. = #$01
LDX DLO      Zieladresse (Anfang) – lo-byte
LDY DHI      Zieladresse (Anfang) – hi-byte
JSR $FFD5    E/A-Routine LOAD
```

Der Einsatz der RS232-Schnittstelle

Beim Einsatz von OS-Routinen zur Steuerung der RS232-Schnittstelle sollten Sie folgende Punkte beachten:

● Der Commodore 64 arbeitet mit Spannungen von 0V bis 5V, der RS232-Standard jedoch mit -12V bis 12V. Außer bei der Kommunikation mit einem Commodore 64 muß die Spannung daher umgewandelt werden. Commodore bietet für diese Aufgabe eine RS232-Cartridge an.

● Da die ASC-Codes des Commodore sich von denen des ASCII-Standards unterscheiden, benötigen Sie für das Senden und Empfangen je ein Umwandlungsarray.

● Bei jedem Eröffnen (OPEN) eines RS232-Kanals löscht das OS mit CLR automatisch die eingesetzten Register. Dabei gehen die Variablenwerte des laufenden BASIC-Programms verloren, so daß GOSUB-Befehle Fehlermeldungen auslösen, wenn sie auf RETURN treffen. Der Löschvorgang wird von den E/A-Routinen ausgelöst, die an der Speicherobergrenze zwei Buffer zu je 256 Bytes reservieren.

● Bei langen BASIC-Programmen und/oder bei einer großen Anzahl von Stringzuordnungen muß früher oder später der Speicher bereinigt werden. Dabei können eingehende Daten verlorengehen.

Der Befehl für die Eröffnung eines RS232-Kanals von BASIC aus lautet:

```
OPEN2,2,3,CHR$(CTRL)+CHR$(COMM)
```

CTRL und COMM sind die „Control-" und „Command"-Bytes, die die Informationen zur Kanaleröffnung enthalten. Beachten Sie, daß CTRL und COMM keine Variablen, sondern zwei echte Bytes (oder PEEKs von zwei zuvor angelegten Speicherstellen) sein müssen, da jedes Bit der CTRL- und COMM-Bytes eine bestimmte Funktion hat (siehe Tabellen).

Beispielsweise wird die Eröffnung eines RS232-Kanals mit einem Stop-Bit, einem 7-Bit-Wort, 300 Baud (CTRL Byte gesamt = 38), gerader Parität, Voll-Duplex und drei Steuerleitungen (COMM Byte gesamt = 96) mit folgendem Befehl durchgeführt:

```
OPEN2,2,3,CHR$(38)+CHR$(96)
```

Bei der Festlegung der Baudrate müssen folgende Faktoren berücksichtigt werden: Für das Senden von Daten ist die Baudrate nicht kritisch, da die Geräte normalerweise geringe Geschwindigkeitsschwankungen tolerieren. Unter BASIC können Zeichen durchaus mit 2400 Baud über die RS232-Leitung gesandt werden. Dabei beträgt die Übertragungsrate der Zeichen 2400 Baud, doch sind die Pausen zwischen den einzelnen Bytes oft weitaus länger, so daß die tatsächliche Geschwindigkeit niedriger ist.

Beim Empfang von Daten ist die Situation jedoch grundlegend anders. In diesem Fall hat ein BASIC-Programm selbst bei 300 Baud kaum Zeit, sich ein Byte aus dem Eingabebuffer zu nehmen und zum Bildschirm zu schicken. Für den Empfang von Daten müssen Sie daher den Datenfluß „steuern".

1) Einlesen einer geringen Anzahl Bytes (je höher die Baudrate, desto kleiner die Zahl) mit GET#2,A$. Bearbeiten Sie diese Bytes schnell, oder speichern Sie sie zur späteren Verarbeitung in einem Array.

2) Stoppen des anderen Gerätes mit dem Befehl PRINT#2,CHR$(17).

3) Einlesen von weiteren Bytes, bis der eigentliche RS232-Buffer leer ist. Ausführliche Bearbeitung aller empfangenen Bytes, während das System prüft, ob Zeicheneingaben oder EOF-Signale, die etwa das Dateiende anzeigen, empfangen wurden.

4) Anforderung weiterer Daten von dem Sendegerät mit PRINT#2,CHR$(19) und zurück zu Schritt eins.

Ist ein RS232-Kanal einmal eröffnet, werden Bytes auf die übliche Weise mit PRINT# oder GET# (nicht aber INPUT#) gesendet oder empfangen. Sie sollten auch das Statusbyte ST testen, das für alle E/A-Programme mit ST=0 oder ST=8 anzeigt, daß das System fehlerfrei arbeitet.

CTRL-Byte

Bit 7	6	5	4	3	2	1	0	Funktion
—	—	—	X	0	0	0	1	50 Baud
—	—	—	X	0	0	1	0	75 Baud
—	—	—	X	0	0	1	1	110 Baud
—	—	—	X	0	1	0	0	134,5 Baud
—	—	—	X	0	1	0	1	150 Baud
—	—	—	X	0	1	1	0	300 Baud
—	—	—	X	0	1	1	1	1200 Baud
—	—	—	X	1	0	0	0	1800 Baud
—	—	—	X	1	0	1	0	2400 Baud
—	0	0	X	—	—	—	—	8-Bit data
—	0	1	X	—	—	—	—	7-Bit data
—	1	0	X	—	—	—	—	6-Bit data
—	1	1	X	—	—	—	—	5-Bit data
0	—	—	X	—	—	—	—	1 Stop-Bit
1	—	—	X	—	—	—	—	2 Stop-Bits

X = Ist egal

COMM-Byte

Bit 7	6	5	4	3	2	1	0	Funktion
—	—	—	—	X	X	X	0	Protokoll mit 3 Leitungen
—	—	—	—	X	X	X	1	Protokoll mit X Leitungen
—	—	—	0	X	X	X	—	Vollduplex
—	—	—	1	X	X	X	—	Halbduplex
—	—	0	—	X	X	X	—	Parity AUS
0	0	1	—	X	X	X	—	ungerade Parität
0	1	1	—	X	X	X	—	gerade Parität
1	0	1	—	X	X	X	—	Mark send p-check AUS
1	1	1	—	X	X	X	—	Space send p-check AUS

Fachwörter von A bis Z

Matrix = Matrix
Eine Matrix ist ein mehrdimensionales System von mathematischen Elementen, die im Computer nach einem bestimmten Schema als Array abgespeichert werden. Jeder Dimension der Matrix wird ein eigener Index zugeordnet, und das einzelne Feldelement ist dann durch einen Satz von Indexwerten eindeutig definiert. Bei einer zweidimensionalen Matrix läßt sich beispielsweise die Variable x als Zeilen- und y als Spaltenindex verwenden; das Wertepaar (2, 3) würde das Element im Kreuzungspunkt der 2. Zeile und 3. Spalte bezeichnen.

Memory Hierarchy = Speicherhierarchie
Jeder Speichertyp benötigt eine andere Zugriffszeit. Deshalb wird die Speicherung der Daten „hierarchisch" organisiert: Die Daten nehmen je nach Zugriffszeit für die Programmierung einen unterschiedlichen Rang ein. Die am schnellsten benötigten Daten stehen in der Hierarchie obenan, und die akut benötigten Daten müssen vom System jeweils rechtzeitig aus den unteren Ebenen hochgeschoben werden.

Die Spitze der Hierarchie bilden bei einem Microcomputer die CPU-Register, die aber insgesamt nur wenige Byte aufnehmen. Danach kommt der Teil des RAM, der zur aktuellen „Zero Page" gehört, gefolgt von den übrigen RAM-Bytes. Darunter rangieren die Daten des gegenwärtig benutzten Datenträgers. Zur untersten Ebene werden schließlich die Disketten und Cassetten gerechnet, die nicht im Abspielgerät stecken.

Die Zugriffszeit wird dabei von oben nach unten immer größer; sie reicht von Mikrosekunden-Bruchteilen für die CPU-Register bis zu Minuten für Cassetten, bei denen erst noch Einlegen und Spulen erforderlich ist. Zum Ausgleich weisen die Speicher auf den untersten Hierarchiestufen die größte Kapazität auf, und zugleich sinken die Kosten der Datenspeicherung. Während Sie bei den CPU-Registern mit etlichen Mark pro Byte rechnen müssen, erhalten Sie für dieses Geld bis zu ein Megabyte an Disketten- oder Cassettenkapazität.

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.

Microprocessor = Microprozessor
Ein Microprozessor ist ein hochintegrierter Baustein, der die CPU des Microcomputers enthält. Der Prozessor umfaßt mindestens das Rechenwerk (ALU) für die Ausführung arithmetisch/logischer Operationen und das Steuerwerk mit Befehlszähler sowie Stack- und Adreßregistern. Im Steuerwerk ist auch der Befehlssatz des Prozessors gespeichert, also die Liste der CPU-Kommandos, und außerdem ist es für die gesamte Ein/Ausgabesteuerung zuständig, also für den Informationsfluß zwischen der CPU, dem Arbeitsspeicher und den anderen angeschlossenen Systemkomponenten.

Der erste bekanntere Microprozessor war der Intel 4004, der aufgrund einer Idee von Edward Hoff Ende der sechziger Jahre entwickelt und ab 1971 gefertigt wurde. Seither sind Hunderte verschiedenster Prozessortypen entworfen und in Millionenauflage produziert worden – die erfolgreichsten waren der 6502 vom MOS Technology, der Zilog Z80 und der Intel 8088. Heute liegt das Hauptgewicht der Entwicklung auf preisgünstigen 32-Bit-Microprozessoren, die sich mit ihren Vorfahren kaum noch vergleichen lassen.

MIDI = MIDI
MIDI ist eine Abkürzung für „Musical Instrument Digital Interface" (Digitale Instrumentenschnittstelle) – ein international anerkannter Standard, der die Kopplung von Computern und elektronischen Musikinstrumenten unterschiedlicher Hersteller ermöglicht. Im MIDI-Verbund kann eine Vielzahl von Instrumenten und Bandgeräten durch einen Steuerrechner synchronisiert werden.

Der zentrale Computer weist jedem Instrument im MIDI-Netz eine Kanaladresse zu. Möchte er irgendwann eine ‚Message' (Mitteilung) an ein bestimmtes Instrument senden, übergibt er sie zunächst dem MIDI-Steuerinterface. Dort wird das Message-Byte mit einem Start- und einem Stopbit versehen, und die zehn Bit werden dann seriell über die MIDI-Leitung geschickt. Wo die Empfängeradresse mit dem Adreßteil der Message übereinstimmt, tritt das MIDI-Interface in Aktion, um den mitgegebenen Befehl zu entschlüsseln und auszuführen; andernfalls wird die Anweisung ignoriert.

Auf dem Markt erscheinen immer mehr MIDI-Schnittstellen, die sich mit den verschiedensten Microcomputern steuern lassen. Zwar verlangt die Mindestspezifikation nur einen MIDI-IN- und einen MIDI-OUT-Anschluß, aber viele Hersteller gehen darüber hinaus. Das hier abgebildete Interface MP401 von Roland bietet eine ganze Reihe zusätzlicher Steuerports.

Bildnachweise
1821: Marcus Wilson-Smith
1823, 1825, 1840, 1841: Ian McKinnell
1827, 1831, 1834, 1835, 1843: Kevin Jones
1836: Crispin Thomas
1837: Epson
1838, 1838: Caroline Clayton
1842: Imagebank

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

computer kurs

Modellentwurf
Ein Programmpaket, mit dem auch Laien sich ein Expertensystem nach Maß bauen können.

Fairer Tausch
Da Sie nun die Neue Welt erreicht haben, können Sie mit dem Handel beginnen. Gewürze und Perlen bringen Gewinn ein.

Gestatten, Dr. Logo
Logo ist eine der besten Sprachen für den Einstieg in die Welt der Programmierung. Wir sehen uns die Schneider-Version an.

Fast MSX-Standard
Die amerikanischen Spectravideo-Computer 318 und 328 erfüllen einige Anforderungen des MSX-Standard.