Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Heft 69

Pioneers MSX-Computer

Das WIMP-System

Selbstbau: MIDI-Schnittstelle

Dreidimensionale Grafik

computer kurs

Heft 69

Inhalt

Computer Welt

Schreiben und Lesen 1905
Die Steuerleitungen eines Microprozessors

Tips für die Praxis

Klänge vom Chip 1907
MIDI für Acorn B und C64

Soundcheck 1922
Die komplette Schaltung wird geprüft

Software

Dog and Bucket 1910
Das vollständige Listing

Fenster zur Welt 1920
Das anwenderfreundliche WIMP-Konzept

Sprache

Das hohe C 1914
Bemerkenswerte Parallelen zu Pascal

Hardware

Fürs Heimkino 1917
Computer-Laserdisc-Kombination im Test

BASIC 69

Die Neue Welt 1 1926
Die Spectrum-Version unseres Adventures

Bits und Bytes

Grafik mit Dreh 1929
Fließkommaroutinen zum Routieren

Fachwörter von A—Z

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

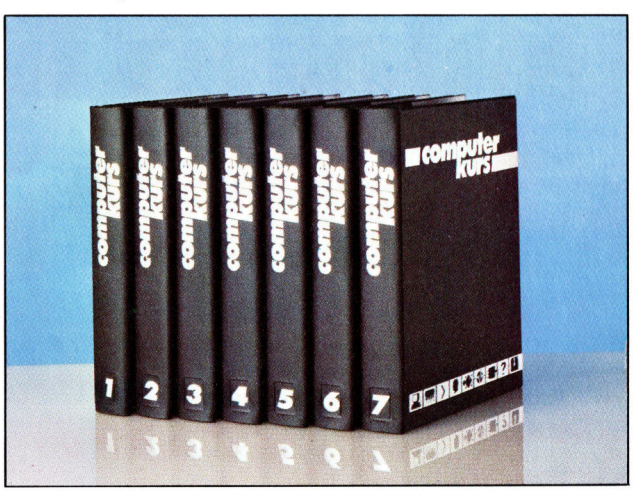
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

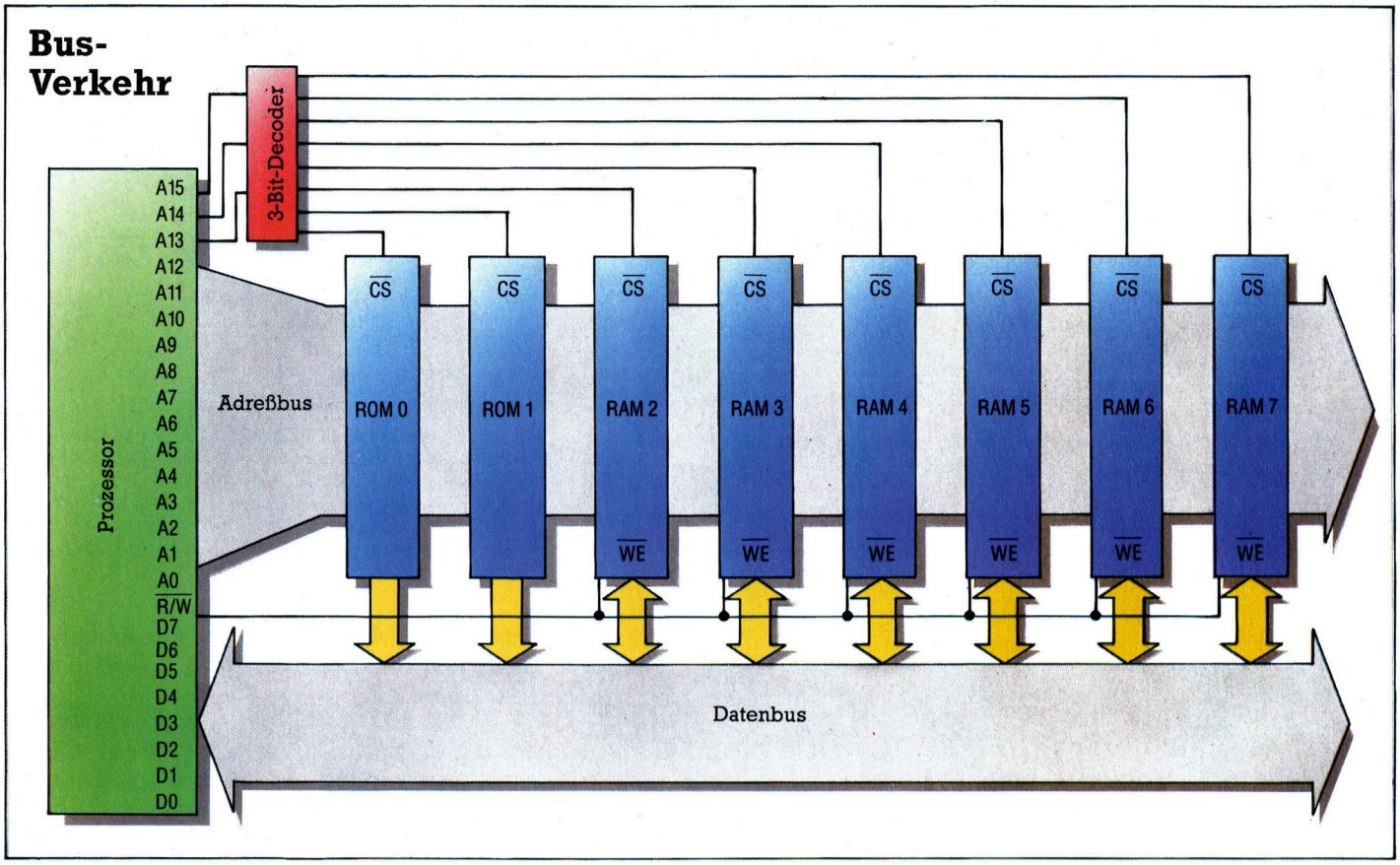
Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Peter Aldick, Holger Neuhaus, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

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



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

Schreiben und Lesen



Nachdem im vorigen Heft erklärt wurde, wie der Befehlszyklus innerhalb der CPU durch die Steuerlogik abgewickelt wird, geht es jetzt um die Speicherbausteine, und zwar vor allem um die Schreib- und Lesevorgänge.

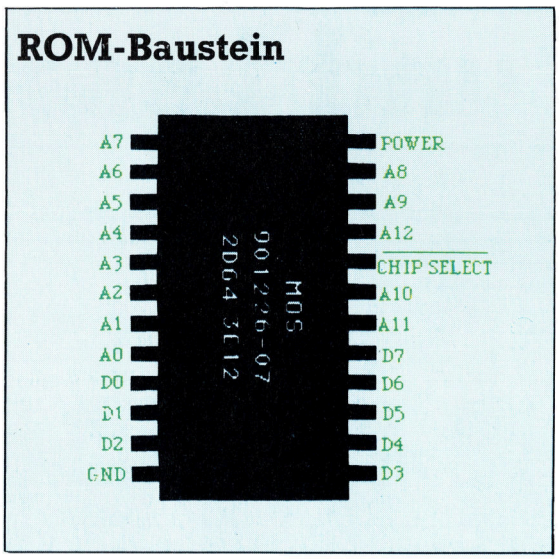
Das Blockdiagramm zeigt eine typische Speicherstruktur für einen 8-Bit-Prozessor mit 16 K ROM und 48 K RAM (statisch). Der Schreib/Lese-Ausgang R/W des Prozessors ist mit den Schreibfreigabe (WE)-Anschlüssen sämtlicher RAM-Chips verbunden. Wenn die WE-Leitung durch die CPU auf Null gesetzt wird, erfolgt die Datenübernahme in das RAM-Byte, das gerade adressiert ist. Zur Auswahl eines bestimmten RAM- oder ROM-Bausteins werden die drei höchsten Adreßbits durch einen Decoder auf eine von acht Chip-Select-Leitungen umgesetzt.

Den Unterschied zwischen einem RAM und einem ROM kennt fast jeder, der überhaupt mal etwas von Hardware gehört hat. Während ein RAM nicht nur ausgelesen, sondern auch beschrieben werden kann, ist der ROM-Inhalt durch den Herstellungsprozeß fixiert. Da bei den RAMs zwischen „statischen" und „dynamischen" zu unterscheiden ist, vorweg noch eine kurze Bemerkung dazu: Bei den statischen RAMs sind die eigentlichen Speicherzellen J-K-Flipflops, die jeweils für ein Bit zuständig sind. Ein Flipflop hält einen definierten Ausgangswert (1 oder 0) fest, bis er durch einen neuen Eingangsimpuls „umgekippt" wird. Für jedes Speicherbyte sind acht derartige Flipflops vonnöten – ein statischer RAM-Baustein enthält also viele Tausende davon.

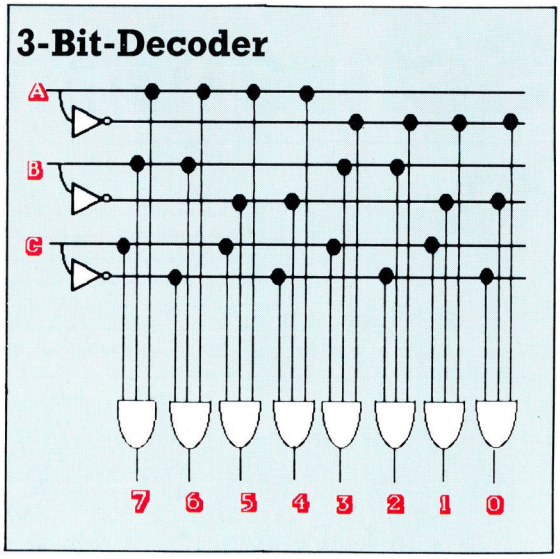
Bei einem dynamischen RAM dagegen sind die Datenbits in Form elektrischer Ladungen gespeichert. Das erfordert zwar viel weniger Platz, so daß der einzelne Chip eine größere Kapazität aufweist, aber dafür muß ein dynamisches RAM periodisch „aufgefrischt" werden, weil die Ladung infolge von Leckströmen sehr schnell abfließt.

Die nächste Seite zeigt die Pinbelegung des Prozessorbausteins 6510, der vom 6502 abstammt, und daneben ein typisches 8-KByte-ROM. Der Prozessor hat vier Gruppen von Anschlüssen, nämlich einen 16-Bit-Adreßbus (A0-A15), einen 8-Bit-Datenbus (D0-D7) und eine Anzahl von Steuerleitungen. Als Besonderheit kommt beim 6510 noch ein 6-Bit-Port hinzu, über den unter anderem zusätzliche RAM- oder ROM-Bänke in den 64K-Adreßraum geschaltet werden können.

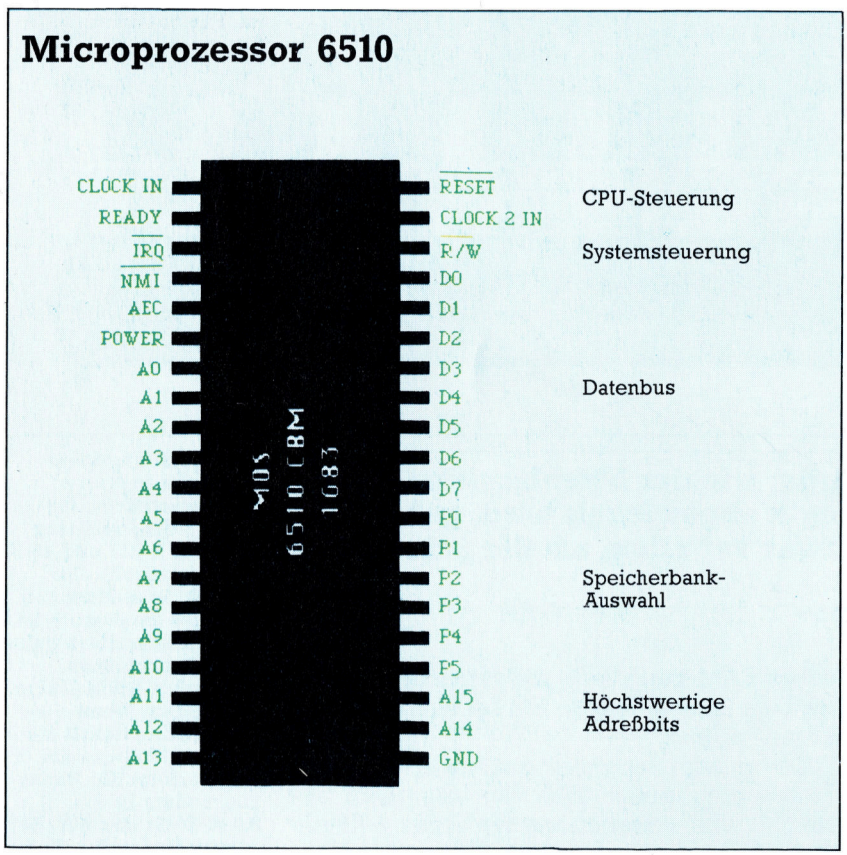
Das daneben abgebildete 8K-ROM hat 13 Adreßleitungen (A0-A12) für das Anwählen der 8192 Bytes, acht Datenleitungen und einen Chip-Select-Anschluß. Statische RAMs weisen eine analoge Pinbelegung auf, haben aber zusätzlich noch einen „Write Enable"-Pin. Der Schlüssel zur Adressierung solcher RAMs und ROMs ist die Chip-Select-Leitung.



Das hier gezeigte 24-polige 8K-ROM weist eine typische Pinbelegung auf: 13 Eingänge werden zu Adressierung der 8192 Byte benötigt, acht Pins beansprucht der Datenbus, zwei die Spannungsversorgung (Power und Ground=GND), und der Chip-Select-Anschluß dient zur Anwahl dieses speziellen ROMs.



Ein Decoder ist eine Digitalschaltung zur Entschlüsselung von Binärzahlen. Jedem Bitmuster am Eingang ist genau eine Ausgangsleitung zugeordnet, der gezeigte 3-Bit-Decoder muß acht Ausgänge haben. Ausgang 0 wird durch den Eingangswert ABC=000 aktiviert, Ausgang 1 durch 001 usw. bis zum Ausgang 7 für ABC=111.



Wie der abgebildete 6510 haben die meisten 8-Bit-Prozessoren 40 „Beinchen", nämlich 16 für den Adreßbus, acht für den Datenbus, den Rest für Takt- und Steuersignale (einschließlich Interrupt-Eingänge und Schreibfreigabe) sowie zur Spannungsversorgung. Der 6510 ist eine verbesserte 6502-Version. Der Fortschritt besteht wesentlich in einem zusätzlichen Datenregister für die Cassetten-Ein/Ausgabe.

Ein 8-Bit-Prozessor kann über den 16-Bit-Adreßbus bis zu 64 KByte direkt ansprechen (2^{16}=65536=64 K). Es sind also acht separate RAM- oder ROM-Chips mit je acht KByte adressierbar. Dabei werden die Adreßleitungen A0-A12 für die Ansteuerung des einzelnen Bytes innerhalb eines Bausteins beansprucht, während die Leitungen A13-A15 für die Chip-Auswahl zur Verfügung stehen. Als Beispiel ist ein typisches Speicherkonzept mit zwei 8K-ROMs und sechs 8K-RAMs abgebildet. Die drei obersten Bits des Adreßbus werden für die Chip-Auswahl auf einen 3-Bit-Decoder mit acht Ausgängen gegeben, während die unteren 13 Adreßbits bei allen Chips gleichzeitig anstehen. In Aktion tritt aber immer nur der durch das Chip-Select-Signal jeweils angewählte Baustein.

Bei den RAMs wird über die ROM-Anschlüsse hinaus noch ein „Read/Write"-Signal zur Unterscheidung von Lese- und Schreibvorgang benötigt. Deshalb hat jedes RAM noch einen Write-Enable-Pin, der über Low-Pegel aktiviert wird und deshalb die Bezeichnung $\overline{WE}$ trägt; er ist mit dem Read/Write-Kontakt R/$\overline{W}$ der CPU zu verbinden. Beim 6510 liegt die R/$\overline{W}$-Leitung normalerweise auf High und wird nur zum Schreiben auf Low heruntergesetzt.

Gewußt, wo

Wenn der Prozessor etwa den Lesebefehl ausführen soll „Akkumulator mit dem Inhalt von Speicherplatz $C000 laden", legt er zunächst die hexadezimale Adresse $C000 auf den Adreßbus, während die R/$\overline{W}$-Leitung auf High bleibt. Die drei höchsten Adreßbits haben in diesem Fall den Wert 110, das heißt, RAM 6 in unserem Blockdiagramm wird durch den 3-Bit-Decoder angewählt. Die übrigen 13 Adreßbits sind alle Null, so daß das unterste Speicherbyte von RAM 6 angesprochen ist. Die Information gelangt über den Datenbus in CPU und Akkumulator.

Für eine Write-Anweisung, etwa „Akku-Inhalt unter $C000 abspeichern", erfolgt genau der gleiche Adressiervorgang, aber außerdem wird die R/$\overline{W}$-Leitung und damit der Schreibfreigabe-Anschluß $\overline{WE}$ heruntergezogen. Der Akku-Inhalt steht bereits auf dem Datenbus an und wird mit dem Freigabesignal in das erste Byte von RAM 6 kopiert.

Klänge vom Chip

Im ersten Kursabschnitt haben wir die technischen Eigenschaften einer MIDI-Schnittstelle beschrieben. Nach diesen Vorschriften soll jetzt eine MIDI-Schaltung zum Anschluß am Commodore 64 oder Acorn B entstehen.

Die Hauptkomponente des Interface ist der sogenannte ACIA (Asynchronous Communications Interface Adaptor). Der ACIA wandelt Parallel-Daten in die von MIDI verlangte serielle Form und umgekehrt. Üblicherweise ist die gesamte Schaltung in einem IC integriert. Wir verwenden dazu den Chip MC68B50, der in vielen Heimcomputern die serielle RS232-Schnittstelle und den Datenverkehr mit einem Cassettenrekorder steuert.

Neben dem ACIA benötigen wir einen Interface-Taktgeber, einen Optokoppler sowie geeignete Ein- und Ausgangsbuchsen. Die Besonderheiten für die Port-Anschlüsse des Commodore 64 bzw. Acorn B geben wir im nächsten Abschnitt an.

Die innere Struktur des Gerätes bilden mehrere Acht-Bit Datenregister. Die Register-SELect Leitung (RSEL) verbindet jeweils ein bestimmtes Register mit dem Computer-Datenbus. Für den Aufbau der MIDI-Schaltung brauchen Sie die Funktion des ACIA zwar nicht genau zu kennen, einige Basisinformationen helfen jedoch bei der Programmierung:

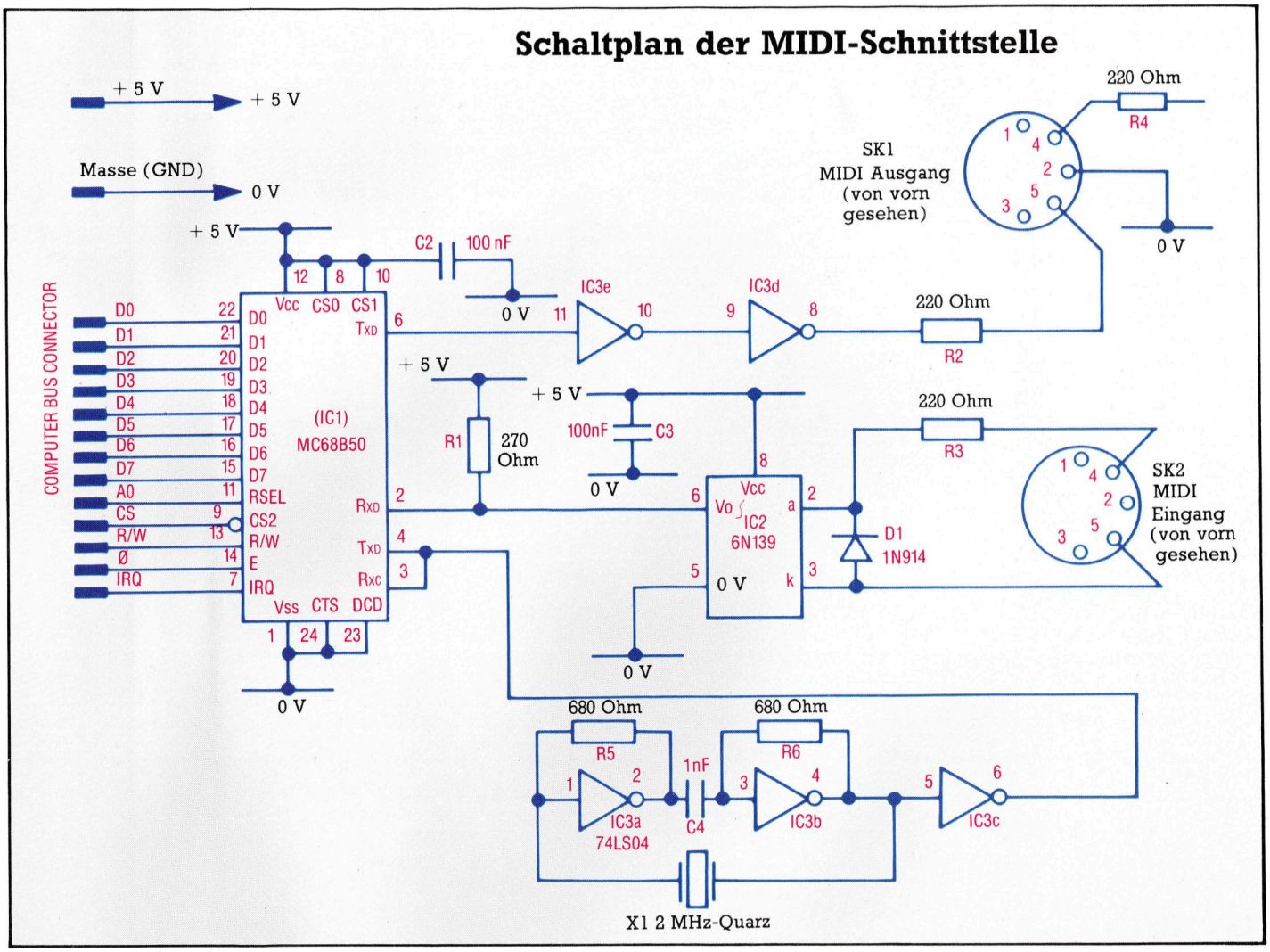
- **Das Status-Register** besteht aus acht Bits, die den aktuellen Zustand des ACIA-Chips angeben. Sie können durch eine READ-Anweisung ausgelesen werden, wenn die RSEL-Leitung auf Low (logisch Null) liegt.
- **Das Kontrollregister** enthält acht Bit für die Steuerung der ACIA. In das Kontrollregister wird mit RSEL-Leitung auf Low geschrieben.

Der Optokoppler 6N139 sorgt für die elektrische Isolation der Eingangsleitung. Der aus dem 2MHz-Quarz mit IC3a und IC3b aufgebaute Taktgeber erzeugt seine Impulse unabhängig von den Taktzyklen des Computers.

IC3c, d und e übernehmen Bufferfunktionen für die Übertragung und bei der Erzeugung der Taktfrequenz.

Die beiden „Chip Select" Leitungen CS0 und CS1 sind mit der Versorgungsspannung verbunden und liegen damit ständig auf High, so daß CS2 der einzige aktive Chip- (bzw. Device Enable-) Eingang ist. Für bestimmte Adressen wird diese Leitung Low, so daß über den Datenbus Ein- und Ausgabeoperationen mit den internen Registern der ACIA vorgenommen werden können.

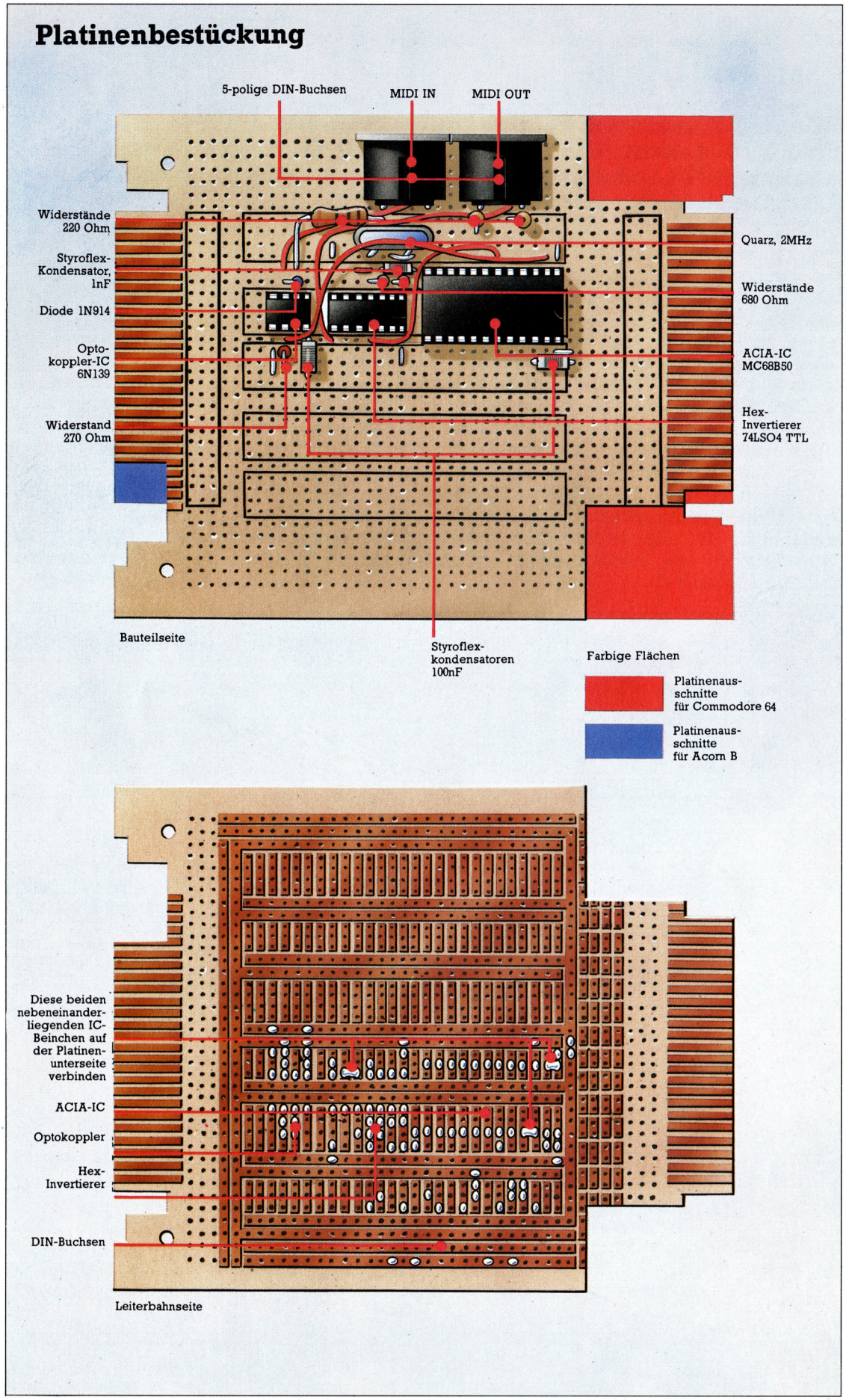
Schaltplan der MIDI-Schnittstelle



Die Bauteile der MIDI-Schnittstelle sollten auf einer solchen Einschubplatine aufgebaut werden. An beiden Platinenkanten liegen doppelseitige Kontaktstreifen. Die rechte Kontaktseite paßt in den Commodore 64, die linke dient zum Anschluß am Acorn B. Vor der Bestückung müssen Ausschnitte an den Seiten gemacht werden. Natürlich brauchen Sie die Ausschnitte nur auf der Ihrem Rechner entsprechenden Seite zu machen. Dafür können Sie entweder ein scharfes Messer oder eine kleine Metallsäge benutzen. Alle Verbindungsleitungen werden auf der Platinenoberseite mit einadriger Wire-wrap-Litze verdrahtet, bis auf die Verbindung der drei nebeneinanderliegenden Paare von IC-Beinchen. Sie werden einfach auf der Unterseite zusammengelötet.

Zuerst die passiven Bauteile – Widerstände, Kondenstoren, DIN-Buchsen und IC-Sockel einlöten. Danach kommen die Verbindungsleitungen und der Schwingquarz. Achten Sie beim Einsetzen der Diode darauf, daß der farbige Markierungsring von oben gesehen rechts liegt. Die kurze Verbindung neben dem Kondensator C3 nicht vergessen! Zum Schluß werden die ICs in die entsprechenden Sockel gesteckt, wobei Sie auf die Position der Markierungen achten müssen.

Platinenbestückung



● **Das TSR** (Transmit Shift Register = Sendeumsetzer) nimmt die zur Übertragung nötige Umwandlung eines Datenbytes in die serielle Form vor. In das Register wird ein Byte aus dem TDR (Transmit Data Register = Datenübertragungsregister) immer dann eingelesen, wenn das TDR voll und die Übertragung des vorhergehenden Bytes abgeschlossen ist. Dabei werden die acht Datenbits auch durch Start- und Stopbits ergänzt. Das TSR ist ein internes Register und kann nicht direkt über den Datenbus angesprochen werden.

● **Das TDR** bildet einen Buffer zwischen Systembus und TSR. Das zu übertragende Byte wird von dem ACIA mit auf High liegender SELect-Leitung in das TDR geschrieben. Dabei wird Bit 1 des Statusregisters gelöscht. Mit gelöschtem Statusbit sollte das TDR nie geladen werden, weil sonst Daten verlorengehen.

● **Das RSR** (Receive Shift Register = Empfangsumsetzer) wandelt serielle Daten in Parallelform um. Ein vollständig empfangenes Byte wird ins RDR geladen, wobei das Statusbit 0 gesetzt wird. Falls dieses Bit schon vorher gesetzt war, wird zusätzlich das Statusbit 5 gesetzt, um anzuzeigen, daß das vorige Byte noch nicht von der CPU verarbeitet ist und überschrieben wurde.

Fehlen Start- oder Stopbits, wird Statusbit 4 gesetzt. Das RSR kann wie das TSR nicht direkt angesprochen werden.

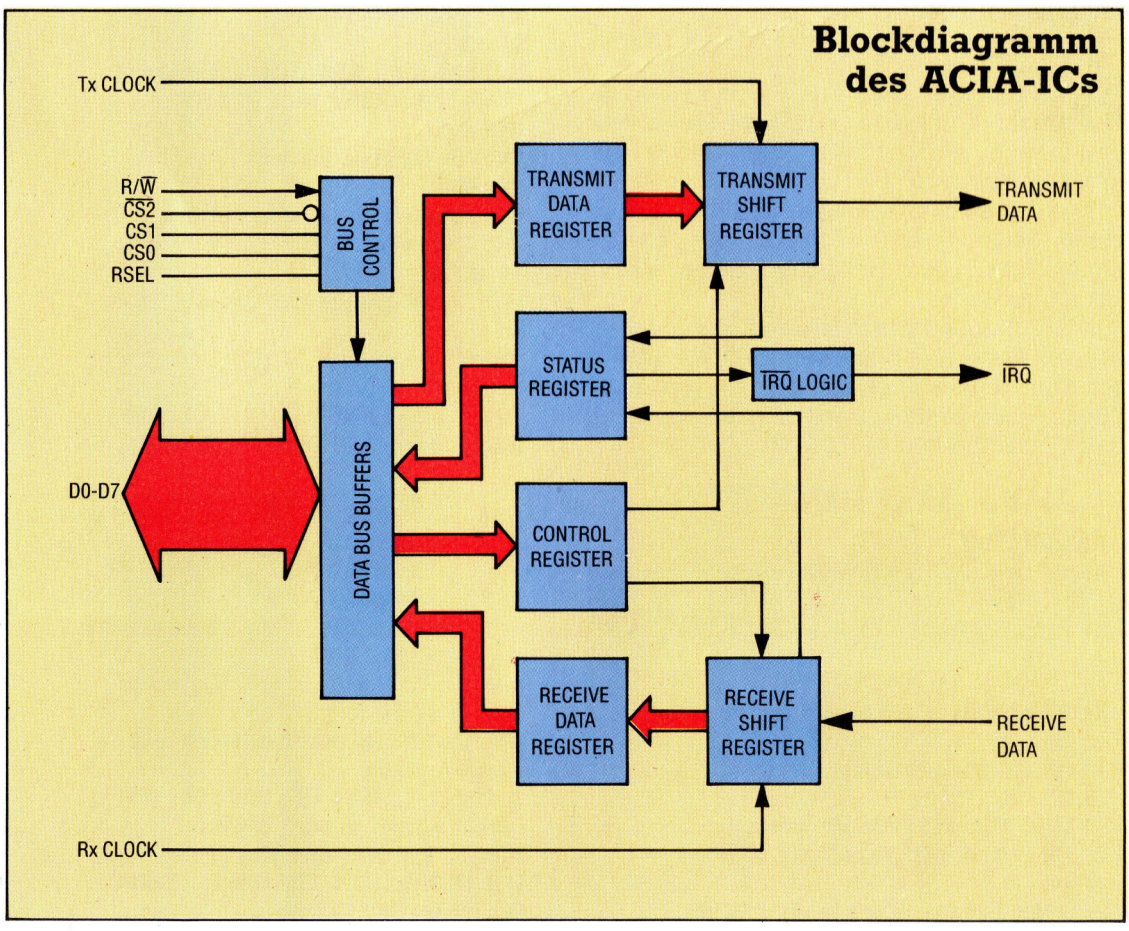
● **Das RDR** (Receive Data Register = Empfangsdatenregister) wird nach jedem Empfang eines vollständigen Byte durch das RSR geladen. Es enthält also das jeweils letzte Datenbyte. Bei auf High liegender RSEL-Leitung kann dieses Byte ausgelesen werden.

Liste der Bauteile

Die Bauteile erhalten Sie im gut sortierten Elektronik-Fachgeschäft. Eventuell Vergleichstypen verwenden oder den Fachhändler bitten, bei Mütron, Bornstraße 22, 2800 Bremen 1, zu bestellen.

Anzahl	Bez. im Plan	Bauteil
1	C4	Kondensator (Styroflex, 1nF)
2	C2/C3	Kondensator (Styroflex, 100nF)
1	R1	Widerstand, 270 Ohm
3	R2/R3/R4	Widerstand, 220 Ohm
2	R5/R6	Widerstand, 680 Ohm
1		Rolle Wire-wrap-Litze
2	SK1/SK2	5polige DIN-Buchsen (180°)
1	IC1	MC68B50 ACIA-Chip
1	IC2	6N130 Optokoppler
1	IC3	74LS04 TTL Sechsfach-Invertierer
1		24poliger IC-Sockel
1		8poliger IC-Sockel
1		14poliger IC-Sockel
1	X1	Schwingquarz, 2,0 MHz
1	D1	Diode, 1N914

Zur Montage verwenden Sie eine Einschubplatine. Sicherheitshalber beim Einkauf die Abbildung der Platine mitnehmen!



Wir zeigen Ihnen das vollständige Listing unseres interaktiven Programms für den Schneider CPC. Außerdem finden Sie Änderungshinweise für den Acorn B, Sinclair Spectrum und Commodore 64.

Das komplette Listing ist in drei Farben gedruckt – schwarz gehaltene Zeilen wurden bereits veröffentlicht, grüne Zeilen wurden neu hinzugefügt.

In einigen Fällen, in denen bereits veröffentlichte Zeilen geändert wurden, um für die neuen Teile Platz zu schaffen, wurden die Zeilen rot gedruckt.

Nach Programmstart können Sie jederzeit mit der Taste Null den Figuren-Editor aktivieren. Drücken Sie dagegen 1, 2 oder 3, wird der Spieler entsprechend in das Wohnzimmer, den Salon oder die Küche versetzt.

„DOG AND BUCKET"-Listing

Initialisierung
Diese Zeilen definieren die Variablen und lesen Daten in die verschiedenen Arrays.

```
10 REM *welcome to the Dog and Bucket*
20 REM
30 REM .....initialise.....
40 REM
45 GOSUB 4030: REM clear the screen
50 DIM l$(3,5),b$(12,4),c$(7,11),d$(11)
60 r=1
70  PRINT "Default values (y/n)?": GOSUB
 4110: IF i$="y" OR i$="Y" GOTO 90
80  GOSUB 2350: GOTO 100
90  FOR n=1 TO 7: READ c$(n,1): FOR d=2
TO 11: READ c$(n,d): NEXT d: NEXT n
100 FOR n=1 TO 3: READ l$(n,1): FOR e=2
TO 5: READ l$(n,e): NEXT e: NEXT n
110 FOR n=1 TO 12: READ b$(n,1): FOR d=2
 TO 4: READ b$(n,d): NEXT d: NEXT n
120 FOR n=2 TO 11: READ d$(n): NEXT n
130 DEF FNb(y,z)=VAL(b$(y,z))
140 DEF FNc(y,z)=VAL(c$(y,z))
150 DEF FNm$(c$,d)=STR$((VAL(c$))-d)
160 DEF FNi$=b$(VAL(c$(c,3)),1)
180 REM setup trees
190 DIM t(5,40,4),k(3,30),c(35),s(6),h(6
): z=0
200 REM object tree
210 FOR n=1 TO 21: REM 21 choice nodes
220 READ k(1,n),t(1,n,2),t(1,n,1): NEXT
n
230 REM plot tree
240 FOR n=1 TO 22: FOR s=1 TO 4: READ t(
2,n,s): NEXT s: READ a$: NEXT n
250 REM character interaction tree
260 FOR n=1 TO 22: FOR s=1 TO 4: READ t(
3,n,s): NEXT s: READ a$: NEXT n
270 REM general activity tree
280 FOR n=1 TO 39: FOR s=1 TO 4: READ t(
4,n,s): NEXT s: READ a$: NEXT n
290 REM object awareness tree
300 FOR n=1 TO 13: FOR s=1 TO 4: READ t(
5,n,s): NEXT s: READ a$: NEXT n
500 REM
510 REM test program loop
520 REM
530 GOSUB 2100: GOSUB 2150: GOSUB 2240:
PRINT: PRINT: GOSUB 1000: GOTO 530
1000 REM
1010 REM character handler
1020 REM
1030 REM check to see if key pressed
1040 GOSUB 4260: IF i$()"" THEN GOSUB 20
40: RETURN
1050     REM process each character in t
urn
1060 FOR c=1 TO 6: IF c$(c,9)="7" THEN c
$(c,9)="0"
1070 IF z=c THEN GOTO 1500
1080 IF FNc(c,10))0 THEN c$(c,10)=FNm$(c
```

Figuren-Modul
Die Zeilen 1000 bis 1500 überprüfen jedes Zeichen und entscheiden, ob es bearbeitet oder verschoben werden soll.

```
$(c,10),1): GOTO 1500
1090     REM flag=0 so reset flag and pr
ocess character
1100 RESTORE: FOR n=1 TO c*10+c-1: READ
c$(c,10): NEXT n
1110 IF FNc(c,10)=0 THEN GOTO 1500: REM
default value=0 so don't process
1120     REM check move flag
1130 IF FNc(c,11))0 THEN c$(c,11)=FNm$(c
$(c,11),1): GOTO 1190
```

BASIC-Dialekte
Einige Besonderheiten sind bereits im letzten Heft erwähnt worden. Zusätzlich sollten jetzt noch folgende Punkte ergänzt werden:

Spectrum:

```
4350 v=INT(RND*a):
    RETURN
4540 RESTORE 9920: FOR
    e=1 TO t(t,n,4): READ h:
    NEXT e: GOSUB h
4570 RESTORE 9930: FOR
    e=1 TO t(t,n,3): READ h:
    NEXT e: GOSUB h:
    RETURN
5470 RESTORE 9910: FOR
    e=1 TO (t(t,n,1)+1):
    READ h: NEXT e: GOTO h
9910 DATA 5480,5490,
    5500,5520,5530,5540,
    5550
9920 DATA 3930,3940,3995
9930 DATA 3030,3060,
    3070,3080,3100,3120,
    3140,3150,3160,3170,
    3180,3190,3200,3210,
    3220,3230,3240,3250,
    3270
```

Acorn B:

```
4350 v=RND(a): RETURN
```

Commodore 64:
Löschen Sie alle Zeilen ab Nummer 7000 aufwärts und ändern Sie das Programm:

```
190 DIM t(5,40,4), k(3,30),
c(35), s(6),
h(6),i$(23)
310 FOR n=1 TO 23: READ
i$(n): NEXT n
4680 m$=i$(t(t,n,4)):
RETURN
```

```
1140      REM move flag=0 so reset flag a
nd move character
1150 RESTORE: FOR n=1 TO c*11: READ c$(c
,11): NEXT n
1160 IF c$(c,11)="0" THEN GOTO 1180
1170 GOSUB 2840: GOTO 1500
1180      REM sort through the trees
1190 GOSUB 2400: REM initialise conditio
ns
1200 FOR t=2 TO 5: GOSUB 2430: GOSUB 546
0: NEXT t
1210 IF FNc(c,4)>0 THEN GOSUB 5000: REM
object manipulation tree
1500 NEXT c: GOTO 1030: REM do next char
acter - when all finished check for key
press/do again
```

Übergeordnete Routinen
Diese Routinen überprüfen Figuren und Objekte, geben ihre Details aus und führen allgemeine Funktionen aus.

```
2000 REM
2010 REM input routine
2020 REM
2030 GOSUB 4110
2040 IF (ASC(i$)<48) OR (ASC(i$)>51) THE
N RETURN
2050 IF i$="0" THEN GOSUB 2320: RETURN
2060 r=VAL(i$): c$(7,2)=i$: RETURN
2070 REM
2080 REM location print
2090 REM
2100 PRINT l$(r,1)
2110 RETURN
2120 REM
2130 REM print visible objects
2140 REM
2150 PRINT "You can see: ";
2160 p=0: FOR b=1 TO 12: IF VAL(b$(b,2))
<>r GOTO 2190
2170 p=p+1: IF p>1 THEN PRINT ", ";
2180 PRINT b$(b,1);
2190 NEXT b
2200 IF p=0 THEN PRINT "nothing";
2210 PRINT: RETURN
2220 REM
2230 REM print visible characters
2240 REM
2250 p=0: FOR c=1 TO 6: IF VAL(c$(c,2))<
>r GOTO 2290
2260 p=p+1: IF p=1 THEN PRINT "You are i
n the company of: ";: GOTO 2280
2270 PRINT ", ";
2280 PRINT c$(c,1);
2290 NEXT c
2300 IF p=0 THEN PRINT "There's no-one h
ere."
2310 RETURN
2320 REM
2330 REM initialise chars subroutine
2340 REM
2350 PRINT: h=c: RESTORE: FOR c=1 TO 7:
READ c$(c,1): GOSUB 4070: PRINT c$(c,1);
" - ";: PRINT "Set this char?": GOSUB 41
10
2360 IF (i$<>"y") AND (i$<>"Y") THEN FOR
 n=2 TO 11: READ n$: NEXT n: GOTO 2390
2370 FOR d=2 TO 11: READ s$: PRINT d$(d)
;: INPUT i$: IF i$<>"" THEN c$(c,d)=i$
2380 NEXT d
2390 NEXT c: c=h: RETURN
2400 REM
2410 REM conditions
2420 REM
2430 h=FNc(c,8): i=FNc(c,3): j=FNc(c,6):
 c(1)=ABS(i>0): c(2)=ABS((FNb(j,2)=FNc(c
,2)) AND (q=1)): c(3)=ABS(b$(i,3)="y"):
c(4)=ABS(i=j): c(5)=ABS(b$(i,4)="y")
2440 c(6)=ABS(i=3): c(7)=ABS(FNc(c,5))5)
: c(8)=ABS(FNc(c,5))2): c(9)=ABS(VAL(c$(
c,9))=1): c(10)=ABS(FNc(x,3)=0): c(11)=A
BS(FNc(h,2)=FNc(c,2)): c(12)=255
2450 c(13)=ABS(z=c): c(14)=ABS(g=c): c(1
5)=ABS(z=0): c(16)=ABS(s(c)=255): c(17)=
ABS(FNc(z,2)=FNc(c,2)): c(18)=ABS(h(c)=2
55): c(19)=ABS(i=2)
2460 c(20)=ABS(FNc(x,4)<0): c(21)=ABS(z=
x): c(22)=ABS(FNc(x,3)=FNc(c,6)): c(23)=
ABS(FNc(x,5))15): c(24)=ABS(FNc(x,9)=7):
 c(25)=ABS(FNc(c,5))15): c(26)=ABS(FNc(c
,5))17)
2470 c(27)=FNc(c,9): c(28)=ABS(FNc(c,2)=
r): c(29)=ABS(FNc(c,2)=1): c(30)=ABS(FNc
(c,5)<5)
2500 RETURN
2510 REM
2520 REM check location for objects
2530 REM
2540 f=0: REM set 'found flag' to zero
2550 FOR b=1 TO 12
2560 IF FNb(b,2)=FNc(c,2) THEN f=1: b=12
2570 NEXT b
2580 IF f=1 THEN n=3: GOTO 2600
2590 n=39
2600 RETURN
2610 REM
2620 REM check for presence of owner of
object: if present, set x to character n
umber: jump back into tree
2630 REM
2640 f=0:x=0
2650 FOR m=1 TO 6
2660 IF (FNc(m,2)=FNc(c,2)) AND (FNc(m,6
)=FNc(c,3)) THEN f=1: x=m: GOSUB 2430: m
=6
2670 NEXT m
2680 IF f=1 THEN n=15: GOTO 2700
2690 n=39
2700 RETURN
2710 REM
2720 REM select object at random from ch
aracter's location
2730 REM
2740 b=0
2750 FOR s=1 TO 12
2760 IF FNb(s,2)<>FNc(c,2) THEN GOTO 278
0
2770 GOSUB 4180: IF q=1 THEN b=s: s=12
2780 NEXT s
2790 IF b=0 THEN GOTO 2750
2800 RETURN
2810 REM
2820 REM move a character
2830 REM
2840 IF FNc(c,4)<1 THEN RETURN: REM too
weak to move
2850 y=0: f=0: FOR w=2 TO 5: IF l$(VAL(c
$(c,2)),w)="0" THEN GOTO 2910
2860 GOSUB 4180: IF q=1 THEN f=1: c$(c,9
)="7": GOTO 2880
2870 GOTO 2910
2880 IF FNc(c,2)=r THEN PRINT c$(c,1);"
leaves the room...";: y=1
2890 c$(c,2)=l$(VAL(c$(c,2)),w): w=5: IF
 FNc(c,2)=r THEN PRINT c$(c,1);" enters
the room...";: y=1
2900 IF y=1 THEN y=c: GOSUB 2250: c=y: P
RINT: PRINT: REM update characters prese
nt message
2910 NEXT w: IF f=0 GOTO 2850
2920 RETURN
```

Aktions-Tabellen
Während der Ausführung der Bäume werden diese Routinen aufgerufen.

```
3000 REM
3010 REM action table
3020 REM
3030 FOR n=1 TO 3: GOSUB 4090: NEXT n: m
$=c$(c,1)+" takes another look at the bo
dy, and suddenly realises the hideous tr
uth. "+CHR$(34)+"The pasty's made of cat
-food"+CHR$(34)+" ": GOSUB 4630
3040 GOSUB 4390: m$=m$+"yells, and in a
mad rush the assembled company scramble
over the bar and attack Fred the Barman,
 who pleads with them to no avail to spa
re his miserable life.": GOSUB 4630: GOS
UB 4720
3050 FOR n=1 TO 2000: NEXT n: GOSUB 4720
: m$="...and the next day Fred the Barma
n is nowhere to be seen.  However, a num
ber  of oddly shaped pasties are availab
le to feed the ever-hungry clientele of
the Dog and Bucket...": GOSUB 4630: GOSU
B 4720: END
3060 h(c)=255: GOSUB 4680: m$=c$(c,1)+m$
: GOSUB 4630: GOSUB 4720: RETURN
3070 z=c: GOSUB 4680: n$=c$(c,1)+m$: GOS
UB 4300: m$=n$+m$+"stomach, and immediat
ely expires.  The other characters are t
oo involved with their drinks to notice.
..": GOSUB 4630: GOSUB 4720: g=0: c$(c,4
)="-1": RETURN
3080 c$(c,4)="10": g=0: IF t(t,n,4)>0 TH
EN GOSUB 4680: m$=c$(c,1)+m$: GOSUB 4630
: GOSUB 4720
3090 RETURN
3100 a=20: GOSUB 4350: IF v<>5 THEN RETU
RN
3110 GOSUB 4680: GOSUB 4630: GOSUB 4720:
 RETURN
3120 a=2: GOSUB 4350: IF v<>0 THEN RETURN
```

```
3130 m$=c$(c,1)+" tries to wake "+c$(x,1
)+" without success...": GOSUB 4630: GOS
UB 4720: RETURN
3140 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" says "+c$(c,1)+" to "+c$(x,1): GOSUB
4630: GOSUB 4720: RETURN
3150 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+", "+c$(c,1)+" asks "+c$(x,1): GOSUB 46
30: GOSUB 4720: RETURN
3160 GOSUB 4680: m$=c$(c,1)+" and "+c$(x
,1)+m$: GOSUB 4630: GOSUB 4720: c$(c,5)=
"10": RETURN
3170 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" says "+c$(c,1)+" to "+c$(x,1): GOSUB
4630: GOSUB 4720: RETURN
3180 GOSUB 4680: m$=c$(c,1)+m$: GOSUB 46
30: GOSUB 4720: c$(c,5)="5": RETURN
3190 GOSUB 4680: m$=c$(c,1)+m$: GOSUB 46
30: GOSUB 4720: RETURN
3200 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" says "+c$(c,1)+" dejectedly...": GOSU
B 4630: GOSUB 4720: c$(c,5)="10": RETURN
3210 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" says "+c$(c,1): GOSUB 4630: GOSUB 472
0: c$(c,9)="3": RETURN
3220 x=FNc(c,8): m$=c$(c,1)+" and "+c$(x
,1)+" are deep in conversation.": GOSUB
4630: GOSUB 4720: GOSUB 4220: x=0: RETUR
N
3230 x=FNc(c,3): IF x)0 THEN GOSUB 4680:
 m$=c$(c,1)+" drunkenly examines "+b$(x,
1)+m$: GOSUB 4630: GOSUB 4720: GOSUB 422
0: x=0
3235 RETURN
3240 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" yelps "+c$(c,1): c$(c,5)="4": GOSUB 4
630: GOSUB 4720: GOSUB 4220: RETURN
3250 GOSUB 4680: m$=CHR$(34)+m$+CHR$(34)
+" howls "+c$(c,1)+", looking at the ":
GOSUB 4630: IF FNc(c,3)=3 THEN m$="pasty
.": GOSUB 4630: GOSUB 4720: GOSUB 4220:
RETURN
3260 m$="sandwich.": GOSUB 4630: GOSUB 4
720: GOSUB 4220: RETURN
3270 GOSUB 4680: m$=c$(c,1)+m$: GOSUB 46
30: GOSUB 4300: GOSUB 4630: m$="drink":
GOSUB 4630: GOSUB 4720: RETURN
3900 REM
3910 REM gosub table
3920 REM
3930 s(c)=255: m$=c$(c,1)+" kneels down
beside the prostrate body of "+c$(z,1)+"
. The horrible truth slowly dawns, but
the others seem too drunk to pay any imm
ediate attention...": GOSUB 4630: GOSUB
4720: RETURN
3940 f=0:x=0: FOR y=1 TO 6
3950 IF y=c THEN GOTO 3970
3960 IF FNc(y,2)=FNc(c,2) THEN f=1: GOSU
B 4180: IF q=1 THEN x=y: GOSUB 2430: y=6
3970 NEXT y: IF f=0 THEN t(3,1,4)=2: RET
URN
3980 IF x=0 THEN GOTO 3940
3990 RETURN
3995 c$(c,5)=FNm$(c$(c,5),1): RETURN
```

Niedere Routinen
Diese Routinen geben Daten auf dem Bildschirm aus und kontrollieren etwa die Eingabe von Zeichen über die Tastatur.

```
4000 REM
4010 REM low level system subroutines
4020 REM
4030 REM clear the screen
4040 REM
4050 CLS: RETURN
4060 REM
4070 REM beep
4080 REM
4090 PRINT CHR$(7);: RETURN
4100 REM
4110 REM get a character from the keyboa
rd
4120 REM
4130 i$=INKEY$: IF i$="" GOTO 4130
4140 RETURN
4150 REM
4160 REM random number routine
4170 REM
4180 q=INT(RND(1)*2)+1: RETURN
4190 REM
4200 REM zero character codes
4210 REM
4220 c$(c,8)="0": c$(c,9)="0": RETURN
4230 REM
4240 REM test to see if key pressed
4250 REM
4260 i$=INKEY$: RETURN
4270 REM
4280 REM print his/her
4290 REM
4300 IF c$(c,7)="f" THEN m$="her ": RETU
RN
4310 m$="his ": RETURN
4320 REM
4330 REM variable random number routine
4340 REM
4350 v=INT(RND(2)*a): RETURN
4360 REM
4370 REM print he/she
4380 REM
4390 IF c$(c,7)="f" THEN m$="she ": RETU
RN
4400 m$="he ": RETURN
4500 REM
4510 REM jumpblock
4520 REM
4530 REM re-entrant nodes
4540 ON t(t,n,4) GOSUB 3930,3940,3995
4550 RETURN
4560 REM action nodes
4570 ON t(t,n,3) GOSUB 3030,3060,3070,30
80,3100,3120,3140,3150,3160,3170,3180,31
90,3200,3210,3220,3230,3240,3250,3270: R
ETURN
4600 REM
4610 REM print messages if player presen
t
4620 REM
4630 IF FNc(c,2)=r THEN PRINT m$;
4640 m$="": RETURN
4650 REM
4660 REM select a message from data stat
ement
4670 REM
4680 RESTORE 7030: FOR m=1 TO t(t,n,4):
READ m$: NEXT m: RETURN
4690 REM
4700 REM print a blank line
4710 REM
4720 IF FNc(c,2)=r THEN PRINT: PRINT
4730 RETURN
```

Verarbeitung der Baumstrukturen
Die Zeilen 5000 bis 5999 sortieren die unterschiedlichen „Bäume". Hier sind auch Routinen zur Objekt-Manipulation integriert.

```
5000 REM object tree routines
5010 p=0: REM zero print flag
5020 IF FNc(c,2)=r THEN p=1
5030 n=1: REM start at node 1
5040 IF n)21 GOTO 5070
5050 k=c(k(1,n))+1: IF k(1,n)=12 THEN GO
SUB 4180: k=q
5060 n=t(1,n,k): GOTO 5040
5070 IF n)=24 GOTO 5090
5080 ON (n-21) GOSUB 2540,2640: GOTO 504
0
5090 ON (n-23) GOTO 5100,5130,5160,5180,
5210,5240,5260,5270,5280,5300,5310,5330,
5340,5360,5370,5430
5100 GOSUB 2740: c$(c,3)=STR$(b)
5110 IF p=1 THEN PRINT c$(c,1);" picks u
p";b$(b,1): PRINT
5120 b$(b,2)="0": c$(c,9)="4":  RETURN
5130 c$(c,3)=c$(c,6)
5140 IF p=1 THEN PRINT c$(c,1);" picks u
p";FNi$: PRINT
5150 b$(VAL(c$(c,3)),2)="0": c$(c,9)="4"
:  RETURN
5160 IF p=1 THEN PRINT c$(c,1);" takes a
 sip from";FNi$: PRINT
5170 c$(c,4)=FNm$(c$(c,4),-1): RETURN
5180 GOSUB 4180: IF (p=1) AND (q=1) THEN
 PRINT c$(c,1);" is eating the sandwich.
": PRINT
5190 c$(c,4)=FNm$(c$(c,4),-2): c$(c,9)="
6": GOSUB 4180: IF q=1 THEN GOSUB 4220
5200 RETURN
5210 IF p=1 THEN PRINT c$(c,1);" takes a
 tentative bite of the pasty, groans, an
d drops it on the floor.": PRINT
5220 g=c: REM set pasty eaten flag
5230 c$(c,3)="0": c$(c,4)=FNm$(c$(c,4),1
0): b$(3,2)=c$(c,2): RETURN
5240 IF p=1 THEN PRINT c$(c,1);" puts do
wn";FNi$: PRINT
5250 b$(VAL(c$(c,3)),2)=c$(c,2): c$(c,3)
="0": RETURN
5260 c$(c,5)=FNm$(c$(c,5),-1): RETURN
5270 GOSUB 5240: RETURN
5280 IF p=1 THEN PRINT c$(c,1);" throws"
;b$(VAL(c$(c,3)),1);" at ";c$(x,1): PRIN
T
```

```
5290 c$(x,4)=FNm$(c$(x,4),1): b$(VAL(c$(
c,3)),2)=c$(c,2): c$(x,8)=STR$(c): c$(x,
9)="5": c$(c,3)="0": RETURN
5300 GOSUB 4220: RETURN
5310 IF p=1 THEN PRINT "I think I've got
 your drink, says ";c$(c,1);" to ";c$(x,
1): PRINT
5320 c$(c,8)=STR$(x): c$(c,9)="2": RETUR
N
5330 c$(c,4)=FNm$(c$(c,4),2): RETURN
5340 IF p=1 THEN PRINT c$(c,1);" gives";
FNi$;" to ";c$(x,1): PRINT
5350 c$(x,3)=c$(c,3): c$(c,3)="0": c$(c,
8)=STR$(c): c$(c,9)="1": RETURN
5360 GOSUB 4220: RETURN
5370 IF p=0 GOTO 5420
5380 IF p=1 THEN PRINT c$(c,1);" is drun
kenly thanking ";c$(VAL(c$(c,8)),1);" fo
r returning ";
5390 IF p=1 AND c$(c,7)="f" THEN PRINT "
her ";: GOTO 5410
5400 PRINT "his ";
5410 PRINT "drink": PRINT
5420 GOSUB 4220: RETURN
5430 RETURN
5440 REM
5450 REM sort trees
5460 n=1
5470 ON (t(t,n,1)+1) GOTO 5480,5490,5500
,5520,5530,5540,5550
5480 k=c(t(t,n,2))+1: n=t(t,n,2+k): GOTO
 5470
5490 GOSUB 4530 : n=t(t,n,3): GOTO 5470
5500 GOSUB 4570
5510 RETURN
5520 GOSUB 4680: GOSUB 4630: GOSUB 4720
5530 RETURN
5540 a=t(t,n,4): GOSUB 4350: n=t(t,n,3)+
v: GOTO 5470
5550 k=c(t(t,n,2)): n=t(t,n,3)+k: GOTO 5
470
```

Programmdaten
Hier stehen die Daten für die Arrays sowie die von verschiedenen Modulen verwendeten Mitteilungen.

```
6000 REM
6010 REM character data
6020 REM
6030 DATA "Toby Belcher","2","7","10","1
0","7","m","0","0","1","4","Fiona Frappe
","1","8","30","10","8","f","0","0","1",
"5","Steve Swig","1","9","8","10","9","m
","0","0","1","6","Sally Short","2","0",
"20","10","10","f","0","0","1","5"
6040 DATA "Rupert Beer","2","11","10","6
","11","m","0","0","1","6","Molly Mixer"
,"1","12","15","6","12","f","0","0","1",
"5","you","1","0","255","255","0","m","0
","0","0","0"
6050 REM
6060 REM location data
6070 REM
6080 DATA "You are in the lounge of the
Dog and    Bucket. Several shady charact
ers are    gathered together in a corner
 playing   dominoes.  Behind the counter
, Fred the Barman looks his usual cheery
 self.     Exits lead east.","0","0","2"
,"0"
6090 DATA "Behold the saloon of the Dog
and Bucket,which looks as if it could do
 with      extensive redecoration. The f
loor       appears to have been regularl
y hosed    down with beer slops.  Doors
lead west  and south.","0","3","0","1"
6100 DATA "Ugh! This is the kitchen, whe
re the     famous Dog and Bucket Cornish
 Pasties   are prepared for an ever-hung
ry         clientele. You notice a numbe
r of empty cat-food tins, which is stran
ge because there are no cats.","2","0","
0","0"
6110 REM
6120 REM object data (for b$(12,4))
6130 REM
6140 DATA " a glass of beer","2","n","y"
," an empty tin of catfood","3","n","n",
" a Dog and Bucket cornish pasty","1","y
","n"," a bar-stool","2","n","n"," an as
htray","1","n","n"
6150 DATA " a stale ham sandwich","2","y
","n"," a pint of bitter","0","n","y","
a creme de menthe","0","n","y"," a whisk
y and water","0","n","y"," a neat vodka"
,"2","n","y"," a pint of lager","0","n",
"y"," a gin and ginger ale","0","n","y"
6160 REM
6170 REM character attribute data (for d
$(11))
6180 REM
6190 DATA "Location","Strength","Invento
ry","Mood","Own object","Sex","Last char
 (lch)","Last command code (lcd)","Handl
e frequency","Move frequency"
6200 REM
6210 REM object tree data
6220 REM
6230 DATA 1,2,22,12,5,4,2,7,6,3,9,8,4,11
,10,12,39,24,12,6,25,5,12,39,6,13,27,12,
23,29,12,30,14,12,26,39,12,28,39,12,31,1
7,7,18,16,8,39,19,9,21,39,10,36,20,12,33
,32,12,35,34,11,38,37
6240 REM
6250 REM plot tree data
6260 REM
6270 DATA 0,13,2,22," ",0,14,5,3," ",0,1
5,21,4," ",5,0,18,3," ",0,16,6,7," ",0,1
7,7,11," ",0,18,9,8," ",0,17,12,13," ",0
,19,10,17," ",5,0,14,3," ",1,0,7,1," ",4
,0,0,0," ",2,0,1,0," ",2,0,5,1," ",4,0,0
,0," ",4,0,0,0," ",2,0,2,4," ",2,0,3,2,"
 ",2,0,4,3
6280 DATA " ",2,0,4,0," ",4,0,0,0," ",4,
0,0,0," "
6300 REM interaction tree
6320 DATA 1,0,3,2," ",4,0,0,0," ",0,20,6
,4," ",0,21,7,5," ",4,0,0,0," ",5,0,8,3,
" ",2,0,6,0," ",0,22,11,14," ",0,23,12,1
7," ",0,24,13,18," ",4,0,0,0," ",4,0,0,0
," ",4,0,0,0," ",5,0,15,2," ",4,0,0,0,"
",2,0,7,5," ",0,25,22,21," "
6330 DATA 5,0,19,2," ",4,0,0,0," ",2,0,8
,6," ",2,0,9,7," ",2,0,10,8," "
6340 REM general activity tree
6350 DATA 5,0,2,5," ",0,26,11,7," ",6,27
,24,7," ",0,28,8,12," ",0,29,9,15," ",5,
0,21,3," ",2,0,11,9," ",4,0,0,0," ",4,0,
0,0," ",4,0,0,0," ",0,30,10,18," ",5,0,1
3,2," ",2,0,12,10," ",4,0,0,0," ",5,0,16
,2," ",3,0,0,11," ",3,0,0,12," "
6360 DATA 5,0,19,2," ",4,0,0,0," ",2,0,1
3,13," ",2,0,14,14," ",2,0,14,15," ",2,0
,14,16," ",4,0,0,0," ",0,11,31,32," ",4,
0,0,0," ",5,0,33,3," ",0,11,36,37," ",5,
0,38,2," ",2,0,14,17," "
6370 DATA 4,0,0,0," ",2,0,15,0," ",4,0,0
,0," ",2,0,16,18," ",4,0,0,0," ",4,0,0,0
," ",2,0,17,19," ",4,0,0,0," ",2,0,18,20
," "
6380 REM object awareness tree
6390 DATA 0,4,8,2," ",5,0,3,5," ",4,0,0,
0," ",4,0,0,0," ",4,0,0,0," ",4,0,0,0,"
",2,0,14,21," ",1,0,9,3," ",5,0,10,4," "
,2,0,19,22," ",2,0,14,23," ",4,0,0,0," "
,4,0,0,0," "
7000 REM
7010 REM message data
7020 REM
7030 DATA "A strange smell fills the air
...could it be the odour of Catty-Kit A
La Carte?"," suddenly collapses on the f
loor, clutching "," looks rather ill, an
d warns the others not to touch the past
y."
7040 DATA " examines the tin carefully,
 and assumes a thoughtfull expression."," What are you doing with my drink?","Wher
e have you been?"
7050 DATA ", in a drunken fit, jump onto
 the table and dance together."
7060 DATA "You look cheerful"," clambers
 drunkenly onto the bar, tries to dance,
 and falls off again."
7070 DATA ", in a drunken stupor, sudden
ly looks up and peers through the VDU sc
reen at you....","Fred the Barman pulls
another pint...","Fred is busy cleaning
glasses"
7080 DATA "It's a dog's life...","I come
 not to bury Caesar...","Let me tell you
 the story of my life...","Barman, fill
my glass!"
7090 DATA "Hi there, everyone...","  as i
f it held the secret of life.","What did
 you do that for!!","I think I'm going t
o be sick!"
7100 DATA "Ahh...This drink is sublime..
."," is desperately hunting for ","Who's
 got my drink?"
```

Das hohe C

Wir untersuchen die Steuerstrukturen von C und beginnen mit den logischen Operatoren. Bei den Schleifenanweisungen finden wir bemerkenswerte Parallelen zu PASCAL.

Die meisten Vergleichsoperatoren von C entsprechen denen anderer Sprachen: <, <=, >, >=. Beachten Sie, daß das Symbol für gleich == ist und für ungleich !=. Die logischen Operatoren sind && für AND und || für OR. Auch die hierarchische Abfolge der Operatoren entspricht der sonst üblichen Übereinkunft. Komplizierte logische Ausdrücke lassen sich daher ähnlich wie in BASIC anlegen. Die Werte von Zuordnungen kann man in logischen Bedingungen testen. So ist beispielsweise über die Standardfunktion getchar(), die das nächste Zeichen aus dem Eingabekanal holt, eine Bedingung aufzubauen:

```
char count(max line length-1 && (c=
   getchar())) !=/n' && c != EOF
```

Dabei wird zunächst die maximale Zeichenzahl überprüft, dann (falls die Zahl nicht erreicht ist) das nächste Zeichen geholt und schließlich getestet, ob das Zeilen- oder Dateiende vorliegt – alles in einem einzigen logischen Ausdruck.

Auch logische Ausdrücke können Werte haben: FALSE ist NULL und TRUE ist Eins (jeder Wert, der nicht Null ist, wird als TRUE interpretiert). Damit lassen sich logische Ausdrücke auch für Berechnungen einsetzen. Einfache numerische Werte können ohne Operator getestet werden. Eine Ganzzahl ungleich Null verwandelt der Negationsoperator ! in Null, eine Ganzzahl gleich Null in Eins.

```
if(!int wert)
```

entspricht

```
if(int wert==0)
```

Außer den normalen logischen Operatoren besitzt C eine Reihe von Bitoperatoren, die mit jedem Typ von integer oder char Bitmanipulationen vornehmen können, die im allgemeinen nur in Assembler zur Verfügung stehen:

&	bitweise AND
|	bitweise inklusiv OR
^	bitweise exklusiv OR
<<n	verschiebt die Ganzzahl n um n Bits nach links
>>n	verschiebt die Ganzzahl n um n Bits nach rechts
~	Einerkomplement

Der &-Operator wird oft für die Ausmaskierung bestimmer Bits eingesetzt, damit ein oder mehrere Bits des Wortes getestet werden können. Der folgende Befehl prüft, ob das Bit 3 eines bestimmten Bytes auf 1 steht:

```
if(c&0x08)
```

Beachten Sie, daß die Schreibweise 0x vor Hexadezimalkonstanten steht und Zahlen, die mit Null beginnen, vom Computer als Oktalzahl verarbeitet werden.

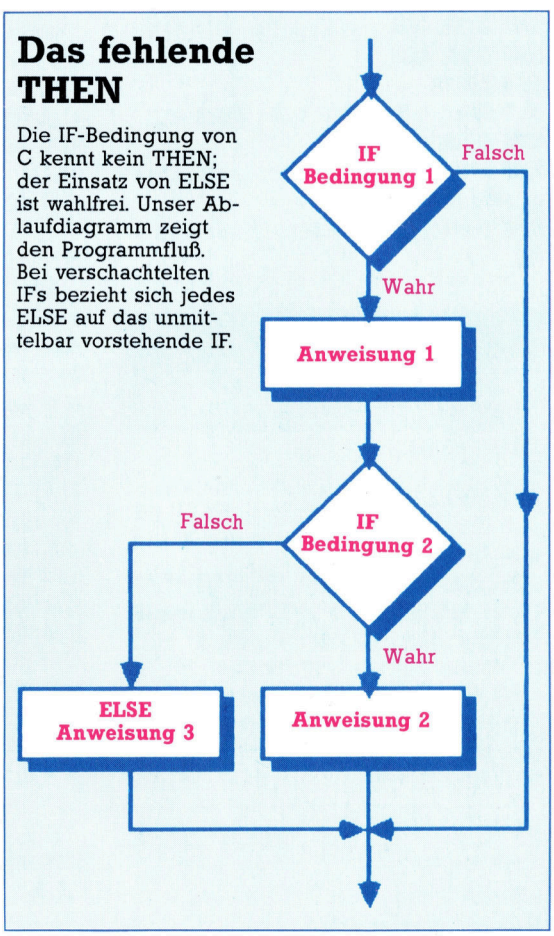
Verzweigungen lassen sich in C mit den üblichen IF-Abfragen ausführen:

```
if(ausdruck)
   anweisung 1
else
   anweisung 2
```

Dabei wird normalerweise ein logischer Aus-

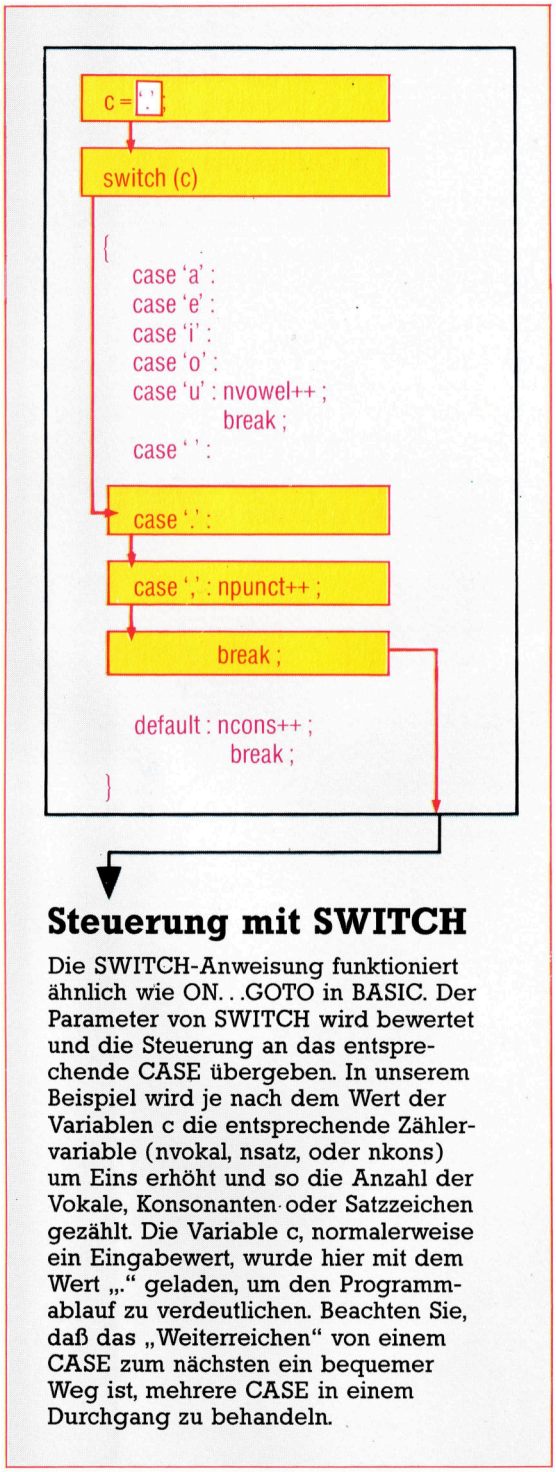
Das fehlende THEN

Die IF-Bedingung von C kennt kein THEN; der Einsatz von ELSE ist wahlfrei. Unser Ablaufdiagramm zeigt den Programmfluß. Bei verschachtelten IFs bezieht sich jedes ELSE auf das unmittelbar vorstehende IF.



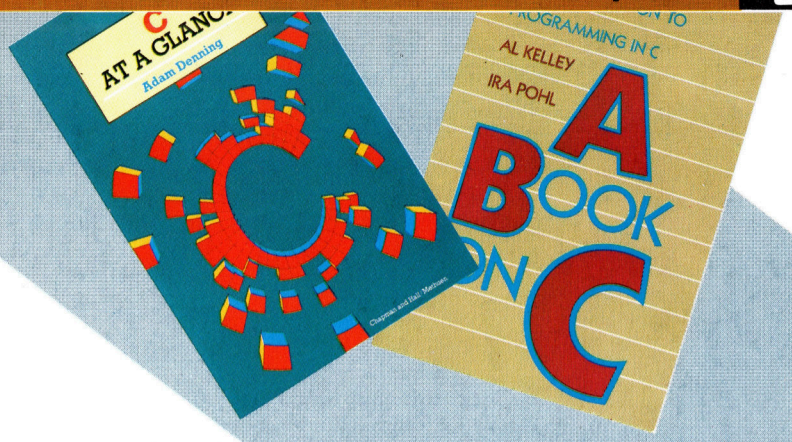
druck verwandt, doch ist jeder Ausdruck möglich, der eine Ganzzahl liefert. Beachten Sie, daß kein then aufgeführt ist. Die Verwendung des else-Teils ist wahlfrei.

Anweisungen können einfaches Format haben, zusammengesetzte Anweisungen müssen in Klammern {} eingeschlossen werden. Wie in vielen anderen Sprachen können verschachtelte ifs Probleme verursachen. Hier gilt aber ebenfalls, daß jedes else dem unmittelbar davor auftretenden if zugeordnet wird. Die Regel läßt sich jedoch ganz einfach mit Klammern umgehen.



Steuerung mit SWITCH

Die SWITCH-Anweisung funktioniert ähnlich wie ON...GOTO in BASIC. Der Parameter von SWITCH wird bewertet und die Steuerung an das entsprechende CASE übergeben. In unserem Beispiel wird je nach dem Wert der Variablen c die entsprechende Zählervariable (nvokal, nsatz, oder nkons) um Eins erhöht und so die Anzahl der Vokale, Konsonanten oder Satzzeichen gezählt. Die Variable c, normalerweise ein Eingabewert, wurde hier mit dem Wert „." geladen, um den Programmablauf zu verdeutlichen. Beachten Sie, daß das „Weiterreichen" von einem CASE zum nächsten ein bequemer Weg ist, mehrere CASE in einem Durchgang zu behandeln.



C – Auf einen Blick: Dieses Buch bietet einen tieferen Einblick in C, als der Titel vermuten läßt. BASIC-Kenntnisse werden vorausgesetzt, und C wird so behandelt, wie es auf den meisten Heimcomputern erscheint. (ISBN 0-412-27140-0).

Das ABC von C: „A Book On C" bietet eine ausführliche Behandlung der Sprache und geht auch auf beigeordnete Themen ein – speziell auf das eng mit C verbundene Betriebssystem UNIX. (ISBN 0-8053-6860-4).

Wahlmöglichkeiten mit mehr als zwei Alternativen werden von der Anweisung switch gesteuert, die der PASCAL-Funktion case entspricht.

```
switch(ganzzahlausdruck)
{
case value 1:anweisung 1;
case value 2:anweisung 2;
...
default: default anweisung;
}
```

Die Werte von case müssen unterschiedliche Konstante sein. Die Angabe von default ist wahlfrei. Da die Case-Werte Labels sind, springt die Programmausführung nicht automatisch auf die Endanweisung, wenn die Ausführung eines Case beendet ist, sondern spricht das nächste Case an.

break kann das jedoch verhindern. Die Steuerung wird dann an die Anweisung übergeben, die auf switch folgt. (siehe Kasten).

Es gibt drei Arten von Schleifen. Am häufigsten ist die while-Anweisung:

```
while(bedingung) anweisung;
```

Zusammengesetzte Anweisungen werden mit Klammern umschlossen.

Da Bedingungen beliebige Ganzzahlausdrücke sein können, ist es eine weit verbreitete (wenn auch keine gute) Programmierpraxis, den Schleifeninhalt soweit wie möglich als Bedingung anzulegen. Die folgenden Beispiele zeigen zwei Wege, eine bestimmte Schleife zu schreiben. Dabei werden Ziffern, die als alphanumerische Zeichen eingegeben werden, in Ganzzahlen umgewandelt. Das erste nichtnumerische Zeichen beendet schließlich die Routine.

```
int num;
char c;
num=0;
c=getchar();
while(c)='0' && c(='9')
{
   num=num*10+c-'0';
   c=getchar();
{
```

Achten Sie darauf, wie die mathematische Berechnung die Zeichen c und '0' einsetzt. Der ASCII-Code wird automatisch in den Typ int umgewandelt, damit eine Rechnung überhaupt stattfinden kann. Die zweite Methode verlegt die Zuordnung zu c in die Schleifenbedingung:

```
num=0;
while((c=getchar()) >= '0' && c <='9')
  num=num*10+c-'0';
```

Da hier nur ein einziger Befehl wiederholt wird, sind keine Klammern nötig.

Die for-Anweisung ist die zweite Methode, Schleifen anzulegen. Sie ähnelt dem FOR. . .NEXT von BASIC und enthält eine Initialisierung, eine Endanweisung (oder Fortsetzung) und eine Schrittanweisung, die bei jedem Schleifendurchlauf aktiviert wird. In den meisten Sprachen wird die Initialisierung mit dem Wert einer Ganzzahlvariablen durchgeführt. Die Endbedingung wird von dem letzten Wert bestimmt, den diese Variable erreichen kann. Die Schrittfunktion addiert einfach einen Ganzzahlenwert auf die Variable.

```
for(expl;exp2;exp3)
  anweisung;
```

expl enthält die Initialisierung und läuft vor der ersten Ausführung der Anweisung ab. Solange die Bedingung wahr (nicht-Null) ist, werden die Fortsetzungsbedingung exp2 und die Schleife wiederholt. exp3 wird am Ende jeder Anweisung ausgeführt. Die BASIC-Schleife:

```
FOR I=1 TO N
  Anweisung(en)
NEXT I
```

entspricht in C:

```
for(i=1;i<=n;i++)
  anweisung;
```

Unser Beispielprogramm zählt und druckt alle Primzahlen zwischen 2 und 1000. Interessant ist besonders der kompakte Code, der durch die Vorinkrementierung mit ++ und das Format der FOR-Schleife erzeugt wird.

Primzahlen

```
/* Das Programm zeigt alle Primzahlen unter
1000 */
main()
{
   int prime count=0, limit=1000, divisor,
   number to test;
/* Beachten Sie die Initialisierung der Variablen
prime count und limit in der Deklarationsanweisung */
   for (number to test=2, number to test<
   limit, ++ number to test)
/* Es folgt die Hauptschleife, die alle Zahlen
zwischen 2 und 1000 testet: */
   {
   divisor=1;
/* Mit %(mod)Operator Divisoren finden */
   while (number to test %++divisor !=0);
/* Die Inkrementierung wird vor dem Test ausgeführt */
/* Die Zahl ist Primzahl, wenn der Divisor den
Wert von number to test erreicht */
   if (divisor==number to test)
/* 1 zum Zähler addieren */
   {
   ++print count;
/* Zehn Primzahlen auf einer Zeile anzeigen */
   printf("%6d",number to test);
   if (prime count % 10==0)
   printf("/n";
   }
}
printf("/n/nes gibt %d Primzahlen unter %d/n",
   prime count,limit);
}
```

Flexible Sache

Um zeigen zu können, wie flexibel die for-Anweisung ist (und C überhaupt), codieren wir die WHILE-Schleife für die Umwandlung eines Eingabestrings in Ganzzahlen mit for:

```
for(num=0;(c=getchar()) >='0' && c<='9';
   num=num*10+c-'0');
```

Beachten Sie, daß der gesamte Schleifeninhalt im Inneren der Bedingung liegt. Obwohl dadurch kompakter und schneller Code entsteht, ist er doch recht unübersichtlich.

C kann Schleifen noch auf eine dritte Art bilden. Sie entspricht dem repeat. . .until von PASCAL. Hier wird der Test im Gegensatz zu den anderen beiden Abläufen erst am Ende der Anweisung durchgeführt:

```
do
  anweisung
while(bedingung);
```

Zwei Zusatzanweisungen gestalten die Schleifensteuerung einfacher als in anderen Sprachen. Wenn die Programmausführung inmitten einer Schleife ein break findet, verzweigt sie sofort auf den unmittelbar hinter dem Schleifenende liegenden Befehl. Das folgende Programmbeispiel liest eine Zeichenfolge ein, die mit einem Return beendet wird, springt aber auch bei der Eingabe von Control C (ASCII-Code 3) aus der Schleife:

```
while((c=getchar())!='/n')
{
  if c==3
    break
  else
    continue /* c weiter bearbeiten */;
}
```

Die Anweisung funktioniert ähnlich, aktiviert aber die nächste Schleife automatisch.

Obwohl mit break und continue alle GOTO-Befehle überflüssig wurden, gibt es in C ein goto. Es wird mit einem Mechanismus der Namensgebung eingesetzt und sollte nur in Spezialfällen verwandt werden.

Fürs Heimkino

Im Unterschied zu den meisten andern MSX-Rechnern verfügt der PX-7 von Pioneer über zahlreiche „audiovisuelle" Schnittstellen.

Der Personal Computer PX-7 von Pioneer erlaubt es, die diversen Möglichkeiten des Speichermediums Laserdisk in Verbindung mit dem abgebildeten Laserplattenspieler LD-700 bzw. dem aufwendigeren LD-1100 zu einem erschwinglichen Preis in die Praxis umzusetzen. Der PX-7 ist ein hochentwickelter MSX-Rechner, der LD-700 ein preisgünstiges Laufwerk für 30 cm-Philips-Bildplatten.

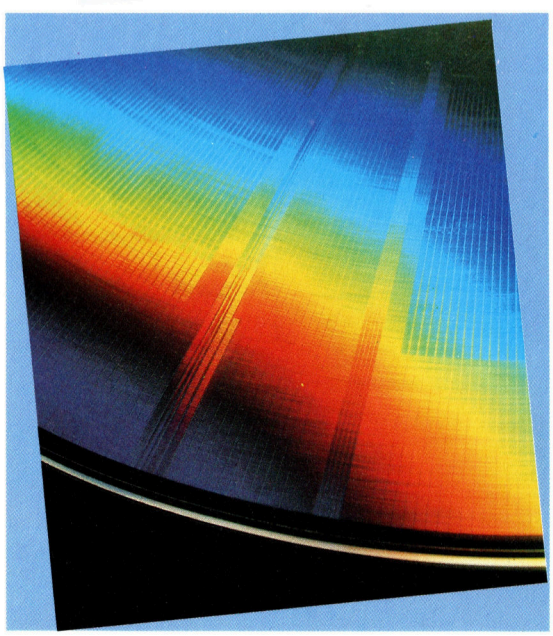
Ungewöhnlich für einen MSX-Computer ist beim PX-7 schon die separate Tastatur. Der Rechner selbst kann in unmittelbarer Nähe von Videorecorder, Bildplattenspieler, Fernseher und HiFi-Anlage untergebracht werden, zumal er auch rein äußerlich eher einem HiFi-Baustein als einem Computer ähnelt. Die Tastatur ist mit dem Rechner über ein 1,50 m langes Kabel verbunden. Die Rechnerfrontplatte enthält eine Kopfhörerbuchse, einen Lautstärkeregler und einen Mischregler zur Einstellung des Pegelverhältnisses zwischen Computer-Sound und Signalen aus externer Quelle, etwa dem Laserplattenspieler. Außerdem gibt es einen Audio-Video-Überbrückungsschalter, der den Rechner deaktiviert und die Audio- und Videogeräte direkt mit einer angeschlossenen HiFi- und Fernsehanlage verbindet.

Der PX-7-Computer und der Bildplattenspieler LD-700 von Pioneer sind aufeinander abgestimmt. Der PX-7 kann mit Ergänzungsbefehlen zum Standard-MSX-BASIC für die Steuerung des Plattenspielers programmiert werden, so daß die Auswahl von Einzelbildern oder Bildfolgen vom Rechner her möglich ist. Außerdem können Computer-Sound und -Grafik mit dem Signal von der Laserplatte gemixt werden.

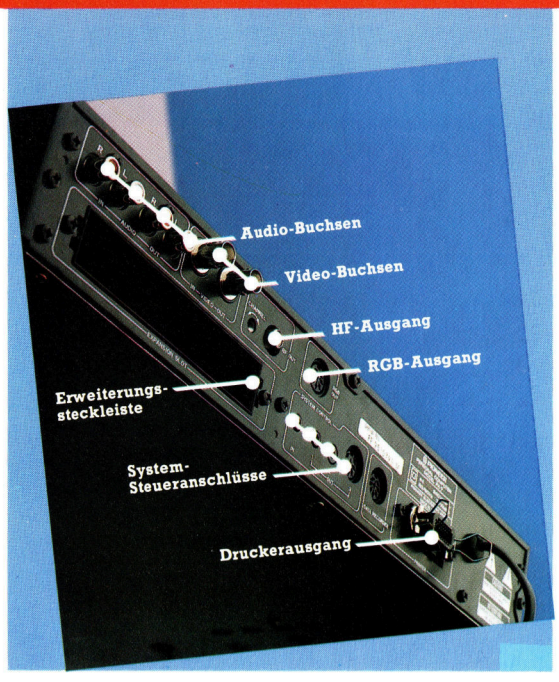
Das Innenleben

In dem Gehäuse steckt ein normaler 32K-MSX-Computer, der mit MSX-Software und -Peripherie arbeiten kann. Überraschenderweise benutzt der PX-7 trotz Laserplattentechnologie noch Cassetten für die Programm- und Datenspeicherung. Wer tiefer in die Möglichkeiten des Rechners einsteigen will, findet aber rückseitig einen zweiten Steckmodul-Anschluß, über den auch Diskettenlaufwerke und anderes Zubehör zu betreiben sind.

Weiterhin gibt es Schnittstellen für Fernseher, Composite-Video- und RGB-Monitor, Cassettenrecorder, zwei Joysticks, Centronics-Drucker und die erwähnten beiden Modul-Steckplätze sowie HiFi-Stereo-Anschlüsse und ein „System Control Interface". Letzteres ist eine universelle Schnittstelle für den Betrieb von Bildplattenspielern, Videorecordern usw. Wenn Neuentwicklungen auf den Markt kom-



Bei CAV-Laserplatten (Constant Angular Velocity = konstante Drehzahl) ist wie bei Magnetplatten ein Direktzugriff auf Einzelinformationen möglich. Das Abtastsystem lokalisiert die Plattensektoren anhand der hier erkennbaren radialen Teilung.



Der PX-7 zeichnet sich vor allem durch ein großzügiges Sortiment von Anschlüssen aus, mit denen über Standardkabel ein ganzes Arsenal von HiFi- und Videogeräten gesteuert werden kann.

men, sind sie über dieses Interface und die zugehörige Software leicht in das System einzubeziehen. Mehrere Geräte können zugleich angeschlossen sein: Jedes bekommt einen eigenen Identifizierungscode, so daß über Rechnerbefehle individuelles Ansprechen überaus bequem möglich ist.

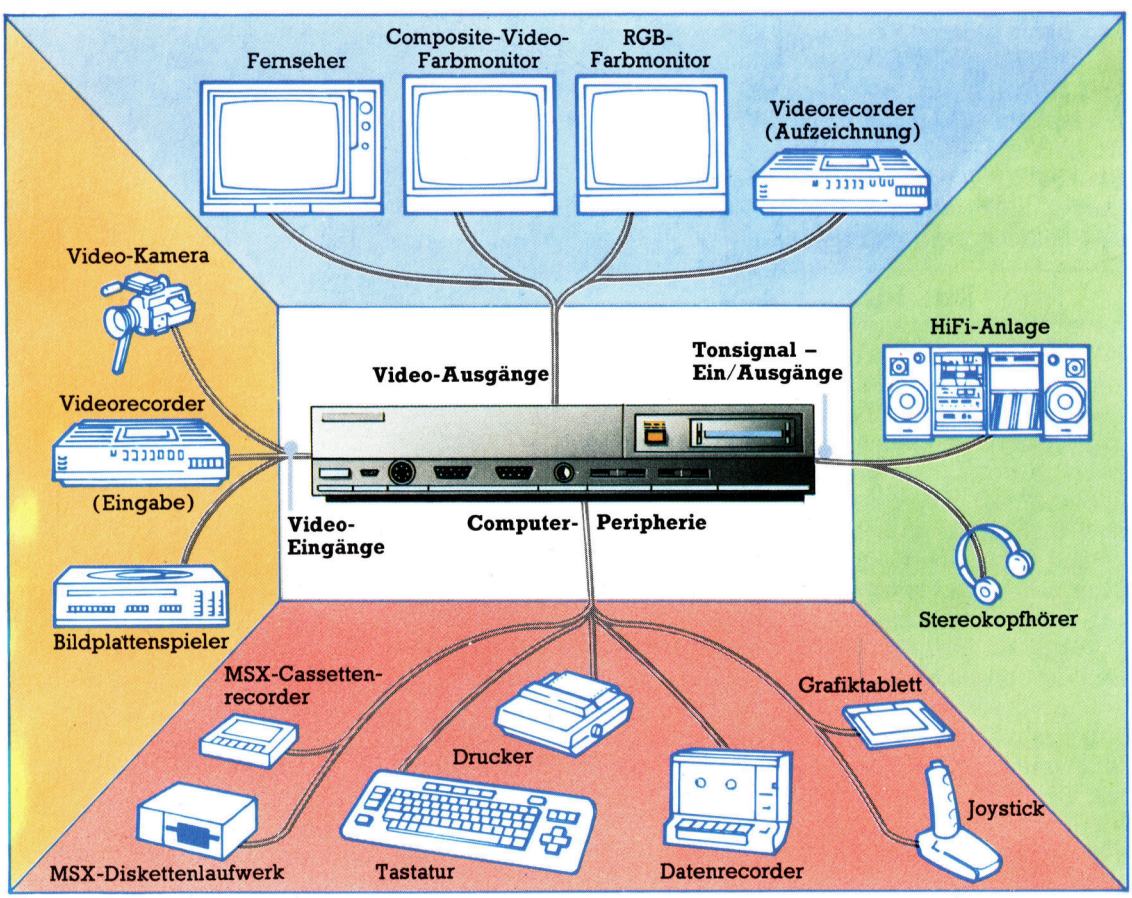
Die Schnittstellensteuerung wird durch ROM-gespeicherte Ergänzungsbefehle zum MSX-BASIC sehr erleichtert. Nach Einschalten des Rechners können Sie zwischen Standard-MSX-BASIC und P-BASIC (enthält zusätzliche Befehle für die Gerätesteuerung) wählen. Die Zusatzkommandos sind als CALL-Befehle implementiert, so daß das MSX-BASIC selbst weiterhin normal läuft. Die meisten der 16 neuen Befehle betreffen die Steuerung der Video- und Audio-Ports des Computers. Auf dem Bildschirm lassen sich entweder das Signal vom Rechner, das einer andern angeschlossenen Quelle (beispielsweise eines Laserplattenspielers) oder eine Überlagerung von beidem darstellen. Vier Sondertasten am Bedienpult erlauben rasches Umschalten zwischen Rechner, Videogerät und Überlagerung.

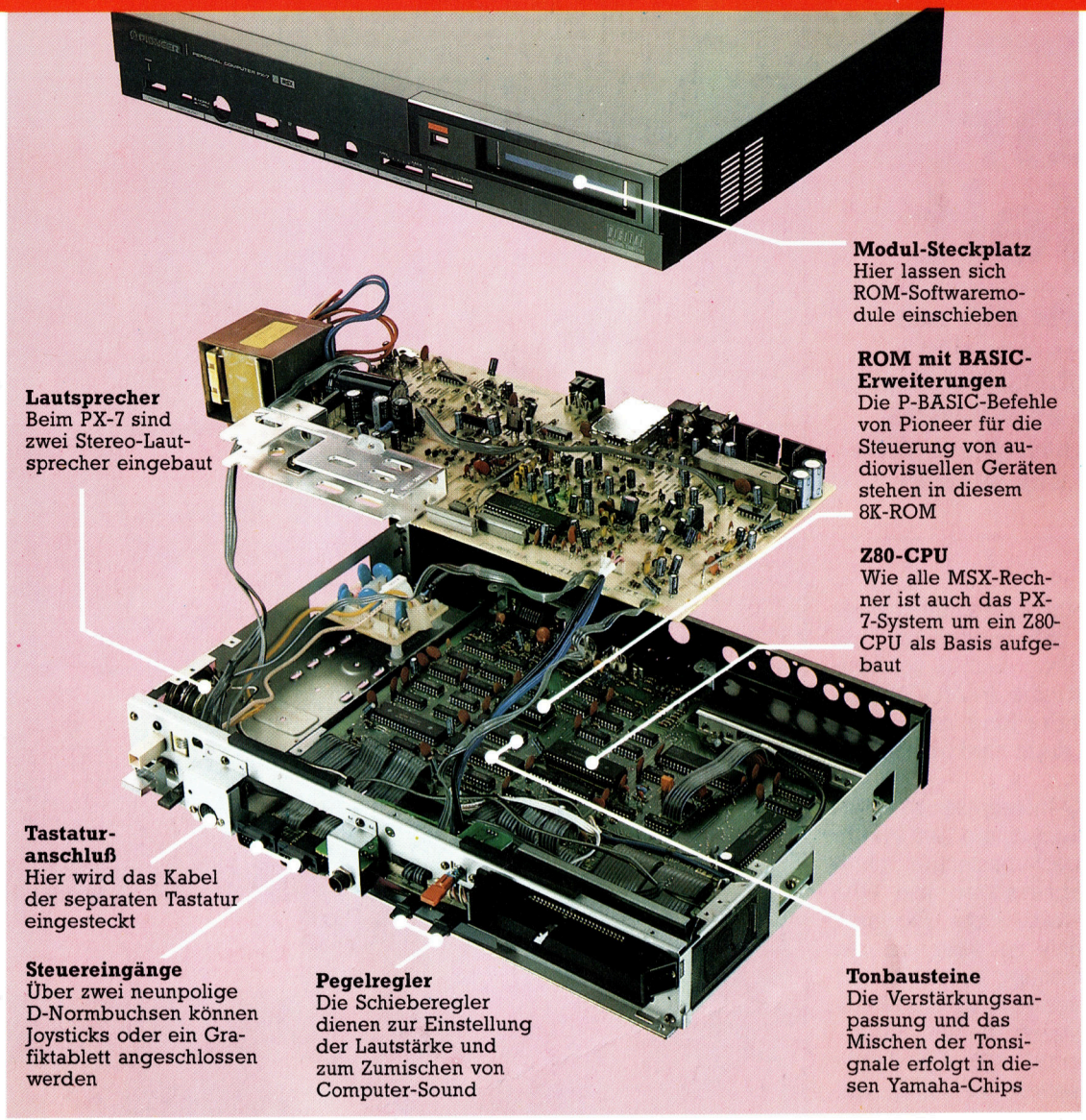
Mehr als MSX-Standard

Die Sound-Befehle ermöglichen die Stummschaltung eines oder beider Stereokanäle und die Balance-Einstellung. Weitere Ergänzungen zum MSX-BASIC bilden diverse Befehle zum Abspeichern von Bildern auf Diskette. Am wichtigsten ist aber das CALL REMOTE-Kommando, über das man jedem an der Systemsteuerung hängenden Gerät individuelle Anweisungen erteilen kann.

Für den Laserplattenspieler gibt es etliche Sonderbefehle, z. B. bewirkt CALL SEARCH (0,F,2000) das Aufsuchen des 2000. Einzelbildes (Frame) auf der Platte. Über CALL FRAME wird beim Erscheinen eines bestimmten Einzelbilds (oder eines Kapitelanfangs) ein vorher festgelegtes Unterprogramm aufgerufen.

Eins der ursprünglichen Ziele bei der Schaffung des MSX-Standards bestand darin, den Computer zum Kontrollzentrum für die ganze häusliche Unterhaltungselektronik einschließlich sämtlicher Ton- und Videokomponenten zu machen. Der PX-7 von Pioneer ist wohl der erste MSX-Rechner, der dafür wirklich geeignet ist. Die Erweiterung des standardmäßigen MSX-BASIC zum „P-BASIC" erlaubt die softwaremäßige Steuerung von Audio- und Videogeräten. Außerdem ist die Mischmöglichkeit mit Ton und Grafik vom Computer vorgesehen.





Das erlaubt die direkte Einbindung des Laserplattensystems in ein Programm.

Der Bildplattenspieler LD-700 ist ein Standardgerät und daher unabhängig vom Rechner erhältlich und verwendbar. Eine CAV-Platte von 30 cm Durchmesser enthält bis zu 54 000 Bilder (Laufzeit 36 min.). Neben dem Abspielen ganzer Filme sind Standbild, Zeitlupe und Zeitraffer möglich, und zwar in wesentlich höherer Qualität als bei Band-Videogeräten. Die mitgelieferte Fernbedienung erlaubt Kapitel- und Einzelbildsuchlauf – ein Kapitel ist eine zusammengehörige Folge von Einzelbildern.

Nicht nur Stereo

Jede Platte verfügt über eine doppelte Tonspur und ermöglicht daher die zweisprachige Tonaufzeichnung; bei einigen Platten trägt die zweite Spur aber auch schon ein fertiges Computerprogramm.

Die Hardware von Pioneer ist solide und ausgereift. Der Rechner erwärmt sich im Regal über dem Bildplattenspieler allerdings erheblich. Als Zubehör wird das Grafiktablett PXTB-7 mit dem Softwarepaket „Video Art" auf Cassette angeboten, zum Erstellen von Programmen für die Einblendung von Computergrafik in Laserplattenbilder. Die Software ist leider enttäuschend langsam.

Interaktive Videoprogramme für den Gebrauch im Wohnzimmer, Büro oder Klassenraum sind mit dem PX-7 jedoch sehr einfach zu schreiben und zu betreiben. Die Programmerstellung ist dank der integrierten BASIC-Erweiterungen fast trivial. Die Geschwindigkeit ist zudem unkritisch, weil das Abspielgerät zum Aufsuchen eines vorgegebenen Einzelbilds ohnehin rund zwei Sekunden Zeit benötigt.

Die Kombination PX-7/LD-700 ist keine Sparversion, sondern sie bietet alle Möglichkeiten für den interaktiven Video-Betrieb. Sie können Bildplatten-Computerspiele schreiben, oder auch gewaltige Bilddaten organisieren. Das System arbeitet auch mit CPE(Computer-Programm Encoded)-Platten, die auf einer der Tonspuren ein Rechnerprogramm enthalten.

Wie stets bei jungen Technologien dauert es einige Zeit, bis ein umfassendes Softwareangebot für die neue Hardware verfügbar ist. Dennoch ist der PX-7 einer der interessantesten MSX-Rechner.

Pioneer PX-7

ABMESSUNGEN:
420 × 323 × 70 mm

SPEICHER:
32K Benutzer-RAM, ausbaubar auf 64K; 16K Video-RAM; 40K ROM

ZENTRALEINHEIT:
Z80-CPU mit 4-MHz-Takt

BILDSCHIRMDARSTELLUNG:
24 Zeilen zu 40 Zeichen; 16 Farben, höchste Grafikauflösung 256 × 192 Punkte; bis zu 32 Sprites (Video-Chip 9929)

SOUND:
Dreistimmig (Tongenerator 8910), Stereo-Ein/Ausgänge

SCHNITTSTELLEN:
Composite Video, Antenne, RGB, Centronics-Drucker, 2 Joysticks, 2 Modulsteckplätze, Cassettenrecorder, Audio- und Video-Ein/Ausgänge, Kopfhörer, Systemsteuerung

VERFÜGBARE SPRACHEN:
MSX-BASIC (32K) und P-BASIC (8K)

DOKUMENTATION:
Mitgeliefert werden kleine Anleitungsbücher, wie bei HiFi-Anlagen üblich. Die Dokumentation ist vollständig und enthält eine Menge technischer Details.

STÄRKEN:
Der PX-7 ist wohl das höchstentwickelte MSX-System und bedeutet in Verbindung mit dem Bildplattenspieler einen echten Fortschritt in der Unterhaltungselektronik. Die Komponenten sind solide und ansprechend gebaut.

SCHWÄCHEN:
Der MSX-Standard hat sich bisher nicht auf breiter Front durchsetzen können und wenig Unterstützung durch die Zulieferbranche gefunden. Daher sind auch beim PX-7 die Möglichkeiten nur begrenzt nutzbar.

Fenster zur Welt

Wir untersuchen, wie sich seit den siebziger Jahren aus den Ursprüngen von SMALLTALK eine Betriebsumgebung entwickelte, die die neue Computergeneration radikal beeinflußt: das WIMP-Systemkonzept.

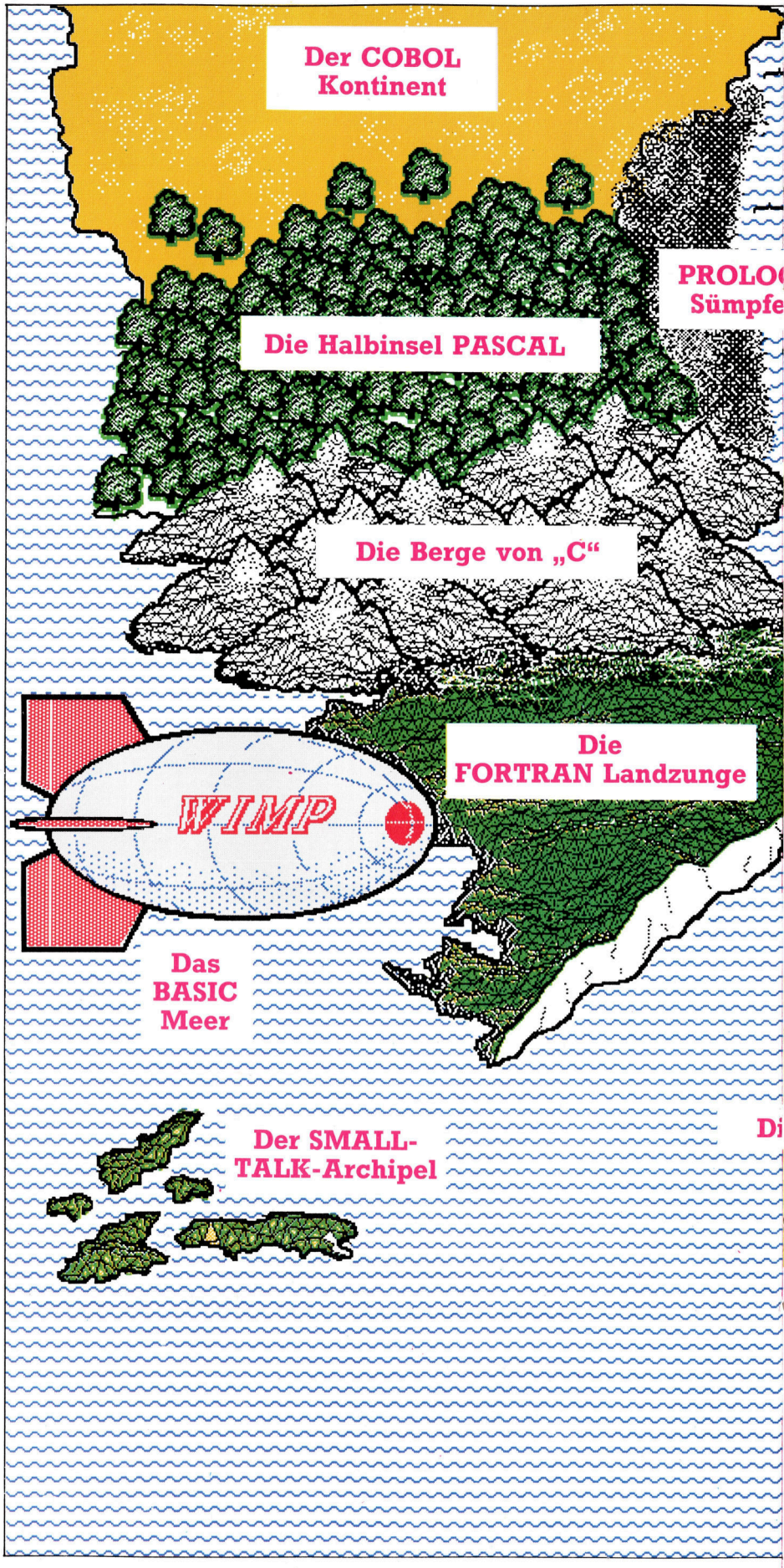
Anfänger werden beim Einschalten ihrer Personal-Computer üblicherweise mit einer von zwei Arten von Programmen konfrontiert. Wenn in das Gerät eine BASIC-Version eingebaut ist, kann leicht der Eindruck entstehen, daß Computer nicht mehr sind als BASIC-Maschinen.

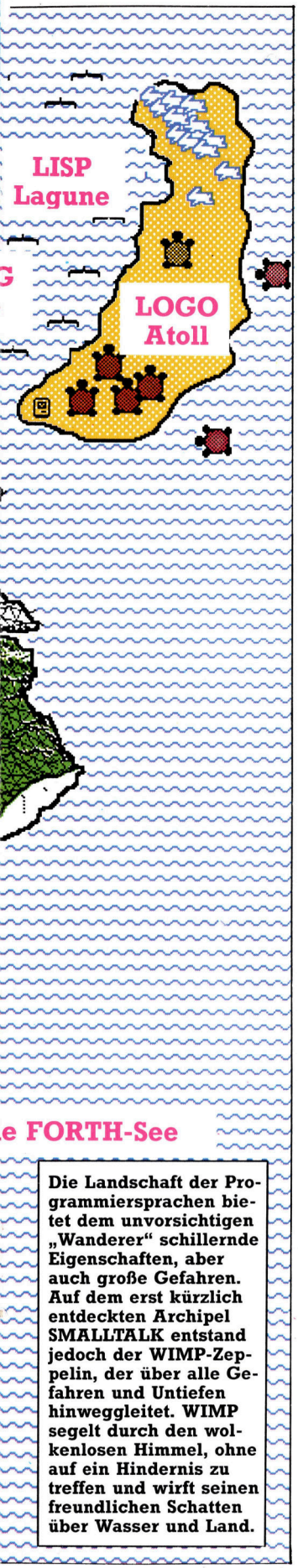
Die integrierten BASIC-Interpreter sind jedoch oft leichter zu bedienen, als der zweite Programmtyp – das Betriebssystem ohne Sprachumgebung. Betriebssysteme wie CP/M, MS-DOS, UNIX etc. haben die Eigenart, eher bedienerunfreundlich zu sein, da sie noch aus den Tagen stammen, in denen der Computeranwender mit den technischen Feinheiten seiner Maschinen vertraut sein mußte.

Inzwischen haben Geräte wie der Apple Macintosh die Kommunikation zwischen Anwender und Computer radikal verändert. Diese Systeme arbeiten mit Fenstern (Windows), Symbolen (Icons) und Mäusen und werden daher kurz WIMP-Systeme genannt. Die WIMP-Technologie ist jedoch nur ein einziger Aspekt des Forschungsbereiches der „objektorientierten Programmierung". Da die Forschungsergebnisse weitreichende Auswirkungen haben, beschäftigen wir uns hier ausführlich mit der Entwicklung auf diesem überaus interessanten Gebiet.

Ohne Zweifel ist SMALLTALK – ein Projekt des Xerox Forschungsteams im Palo Alto Research Centre (PARC) – bisher die am stärksten objektorientierte Sprache. Bei den Ursprüngen von SMALLTALK treffen wir auf ein ungewöhnliches Projekt – das Dynabook von Alan Kay.

Es mutet seltsam an, daß eine Firma, die ihr Geld hauptsächlich mit der Papierverarbeitung verdient, großes Interesse am Konzept einer „papierlosen" elektronischen Büroumgebung zeigt. Die Xerox Company of America hatte jedoch schon in den frühen siebziger Jahren zukünftige Tendenzen vorausgeahnt und wurde daher – vielleicht mehr als andere Firmen – zu einem der wichtigsten Vorreiter bei der Entwicklung von WIMP-Systemen.





Die Landschaft der Programmiersprachen bietet dem unvorsichtigen „Wanderer" schillernde Eigenschaften, aber auch große Gefahren. Auf dem erst kürzlich entdeckten Archipel SMALLTALK entstand jedoch der WIMP-Zeppelin, der über alle Gefahren und Untiefen hinweggleitet. WIMP segelt durch den wolkenlosen Himmel, ohne auf ein Hindernis zu treffen und wirft seinen freundlichen Schatten über Wasser und Land.

Bei der großen Konkurrenz von IBM, DEC und anderen war Xerox klar, daß ihr der späte Eintritt in den Computermarkt nur gelingen würde, wenn sie sich von den anderen Anwendern deutlich abhöbe und so einen Marktanteil erobern könnte. Xerox gründete daher PARC, versah Forscher und Techniker mit großzügigen Mitteln und gab ihnen die weitgesteckte Aufgabe, die Gebiete der Büro-Automatisierung, der Künstlichen Intelligenz (KI), der Schnittstelle Mensch-Maschine (MMI) und Computersysteme allgemein zu erforschen.

Den Anstoß für SMALLTALK gab das futuristische Dynabook – auch „Informationssystem für Alle" genannt. Es wurde von dem Studenten Alan Kay entwickelt, der davon ausging, daß bald jeder Bürger einen kleinen, batteriebetriebenen Computer (etwa in Buchgröße) besitzen würde, mit dem er auf den gesamten Wissensschatz zugreifen könnte.

Das hochtechnisierte, handliche Dynabook sollte einen hochauflösenden Bildschirm haben, Ein- und Ausgabegeräte, Möglichkeiten der Video- und Audiokommunikation und schließlich – über ein Netzwerk von Radiosatelliten – Zugriff zu einer großen umfassenden Datenbank. Leider waren die technischen Probleme der Miniaturisierung zu groß und die Geldsummen für die Verwirklichung dieser Vision zu hoch. Das Dynabook-Forschungsteam wurde aufgelöst. Viele der Konzepte und Ideen hatten jedoch bleibenden Einfluß auf Forschung und Entwicklung und fanden schließlich Eingang in die Maschinen der modernsten Technologie.

Dezentralisierte Rechner

Eines der wichtigsten Konzepte entstand schon in einem frühen Stadium: die Idee des persönlichen Arbeitsgerätes. Bis dahin bedeutete Arbeit am Computer das Teilhaben an den Kapazitäten einer Maschine der Groß-EDV oder einer Reihe von Mini-Computern. Die Datenverarbeitung lief indirekt ab, wobei „Batch-Läufe" der Normalfall waren.

Der größte Unterschied zwischen dieser Umgebung und der PARC-Philosophie war die Umschlagzeit. Da es bei herkömmlichen Systemen oft Stunden dauerte, bis Ergebnisse vorlagen, war das Dynabook-Konzept mit seiner Forderung nach unmittelbaren Resultaten im Jahre 1972 revolutionär. Weiterhin bedeutete der Einsatz von mausähnlichen Eingabegeräten, Symbolen und Windows ein radikales Umdenken in der Datenübermittlung. In der Folge entstanden hochgradig flexible Grafikroutinen, die fest ins ROM der Hardware integriert waren und eine seltsame Sprache, von den PARC-Mitarbeitern SMALLTALK genannt.

Im Zentrum der SMALLTALK Forschung stand die Entwicklung eines umfassenden Hard- und Softwaremodells, dessen Grundlage die Kommunikation zwischen „Objekten" mit bestimmten festgelegten Eigenschaften war.

Bekanntlich enthält ein Computersystem verschiedene Funktionsmodule: Tastatur, Bildschirm, Drucker, etc. Jede dieser Hardwarekomponenten verfügt über genau festgelegte Fähigkeiten und kann nur bestimmte Aufgaben erledigen. Jedes Objekt enthält weiterhin Daten (über seine Eigenschaften) und „weiß", wie eine begrenzte Zahl von Abläufen ausgeführt wird. So „weiß" der Bildschirm, wie er an ihn gesandte Daten anzeigen soll und kann den Cursor wieder an den linken Rand stellen, wenn der rechte Rand überschritten ist.

Orientierung am Objekt

Genau besehen ist ein Computer ein komplexes, ineinander verwobenes Netzwerk aus Hard- und Software, das zum Teil aus Elektronik und zum Teil aus Softwaresteuerung besteht. Ein Softwareingenieur, der jede Komponente eines Computersystems als Objekt ansieht und ihre Eigenschaften und Algorithmen exakt definiert, kann ein System aus einem Netzwerk miteinander kommunizierender Objekte aufbauen. Statt mit der herkömmlichen Technik der strukturierten Programmierung eine Prozedur mit Namen aufzurufen, die dann (als Parameter übergebene) Daten verarbeitet, werden Programm-Objekte definiert, die nach bestimmten Klassen oder Kategorien aufgeteilt sind und Methoden zur Bearbeitung bestimmter Datentypen besitzen. Die Datenargumente werden als „Meldung" übergeben.

In SMALLTALK wurde dieses objektorientierte Konzept zwar am konsequentesten durchgeführt, doch erfordert das System große Speicher- und Verarbeitungskapazitäten. Kürzlich entstand – besonders in den USA – großes Interesse an MODULA-2. Die Sprache ist von PASCAL abgeleitet und wurde von Professor Nikolaus Wirth entwickelt, dem Erfinder von PASCAL. Sie soll die Entwicklung großer Systeme erleichtern, bei denen Programmiererteams voneinander unabhängig aufeinander abgestimmte „Module" (daher der Name) erstellen. Dabei sind Daten, Datentypen und Prozeduren in jedem Modul automatisch getrennt – es sei denn, sie werden speziell in andere Module „im- oder exportiert". Dieses Konzept entspricht der Idee von SMALLTALK, mit der Ausnahme, daß Meldungsübergabe und Datenargumente über PASCAL-ähnliche Prozeduren und Parameterlisten ablaufen. Mit dieser sicheren und einfachen Methode lassen sich große Systeme zusammensetzen, ohne daß der ungehinderte Zugriff auf alle Systemkomponenten Probleme hervorrufen könnte.

Obwohl es Maschinen wie den Macintosh erst seit kurzem gibt, setzen viele Anwender die Direktheit und Bedienerfreundlichkeit der WIMP-Systeme bereits als selbstverständlich voraus. Der Einfluß von SMALLTALK auf die Hardwareentwicklung war daher beträchtlich.

Soundcheck

Unser MIDI-Schnittstellenprojekt geht in die zweite Bauphase. Wir wollen zeigen, wie die Funktion der kompletten Schaltung von BASIC aus geprüft werden kann.

Der ACIA-Baustein, das Herz unserer MIDI-Schnittstelle, ist mit den im Commodore 64 bzw. Acorn B verwendeten Prozessoren 6502 und 6510 kompatibel. Die Daten-, Adreß- und Steuerleitungen der Prozessoren können also direkt an den ACIA angeschlossen werden.

Die fertige Platine ist so aufgebaut, daß sie sich in den Erweiterungsanschluß des Commodore 64 stecken läßt. Die Acorn-B-Version wird dagegen mit 40poligem Flachkabel und passenden Steckern am Tube-Port des Rechners angeschlossen. Die dafür nötigen Arbeiten werden wir für die beiden Computer getrennt erläutern. Nach dem Platinenaufbau soll die Schaltung geprüft werden.

Wenn Geräte am Datenbus angeschlossen sind, die weniger Adreßleitungen als die CPU selbst besitzen, muß zum Ansprechen dieser Peripheriebausteine eine Adreßdecodierung vorgenommen werden. Daher haben solche Geräte einen „Chip Select" (Chipwahl) Anschluß, mit dem das Gerät auf den Datenbus geschaltet wird. Die CPU kann dann über die vorhandenen Adreßleitungen auf die Register des Gerätes zugreifen.

Die „Chip Select"-Signale für verschiedene Peripheriegeräte werden durch Decodierung der oberen Adreßleitungen erzeugt. Die einzelnen Bitkombinationen auf diesen Leitungen erlauben es, mehrere Peripheriegeräte getrennt anzusprechen. Die internen Register der einzelnen Geräte erscheinen damit in der Memory-Map der CPU und können dann wie reguläre Speicherplätze des Computers bearbeitet werden. Allerdings ist dabei Vorsicht nötig – verwendete Adressen dürfen keine Speicherplätze belegen, die vom Betriebssystem des Computers gebraucht werden.

Der Erweiterungsport des Commodore 64 verfügt über zwei mit I/O1 und I/O2 bezeichnete Ausgänge, die beim Ansprechen der Speicherpages $DE bzw. $DF auf Low gehen. Wenn der CS2-Anschluß des ACIA-Chips nun mit I/O1 verbunden ist, können wir damit den ACIA in Page $DE einbinden. Die Adreßleitungen A1 bis A7 sind nicht angeschlossen, also kann auf die internen Register des Chips mit jeder Adresse im Bereich zwischen $DE00 und $DEFF zugegriffen werden. Durch den direkten Anschluß von A0 am Register-Wahlanschluß des ACIA (RSEL) erreichen alle geraden Adressen die Sende/Empfangsdatenregister. Die Adressen $DE00 und $DE01 bieten sich für diesen Zweck geradezu an.

Am Acorn B gibt es zwei verschiedene Ports für den Anschluß – über den Tube-Port können wir den Datenbus direkt mit 2 MHz ansprechen. Für die Adressen $FEE0 bis &FEFF hat der Acorn eine spezielle Decodierleitung NTUBE. Nachteilig ist dabei, daß der Acorn B beim Einschalten durch Lesen bestimmter Tube-Adressen prüft, ob irgend welche Peripherie vorhanden ist. Ist irgendetwas anderes als ein zweiter Prozessor angeschlossen, scheint der Acorn B „abzustürzen".

Den Rechner überlistet

Professionell würde man dieses Problem dadurch umgehen, daß eine der oberen Adreßleitungen mit CS0 auf dem Chip verbunden würde. Als Alternative kann aber auch NTUBE mit CS2 verbunden und die Platine erst bei eingeschaltetem Rechner eingesetzt werden. Die mit NPGFC und NPGFD bezeichneten Pins dienen dazu, externe Geräte in die Seiten &FC und &FD zu „mappen". Sie entsprechen den Commodore-64-Anschlüssen I/O1 und I/O2.

Die Benutzung des 1MHz-Busses hat allerdings auch Nachteile: Die Verlangsamung des Systemtaktes auf 1MHz führt zu Störungen auf den Decodier-Leitungen, die mit einer Hilfsschaltung beseitigt werden müssen. Außerdem gibt es am 1MHz-Bus keine 5-Volt-Versorgung – man muß die 5 Volt am User Port oder der „AUX"-Stromversorgung abzweigen.

Bit für Bit

Falls Sie Ihr MIDI-Interface selbst programmieren möchten, müssen Sie die Funktion der vier ACIA-Register kennen.

Ein Sende-Interrupt tritt auf, wenn das Kontrollregisterbit 6 gelöscht und Bit 5 sowie das TDR (Transmit Data Register)-Statusbit gesetzt sind. Durch Schreiben von Daten zum TDR wird der Interrupt gelöscht.

Ein Empfangs-Interrupt erfolgt, wenn Kontrollbit 7 und Statusbit 0 gesetzt sind. Lesen des RDR (Receive Data Register) löscht den Interrupt (falls nicht auch Statusbit 5 gesetzt ist, welches durch Auslesen des Statusregisters vor dem RDR gelöscht wird). Auch Statusbit 2 kann einen Interrupt erzeugen, es wird bei unserer Anwendung allerdings nicht benutzt. Für Senden und Empfangen sind zwei verschiedene Taktgeber vorgesehen, die im

Register im ACIA

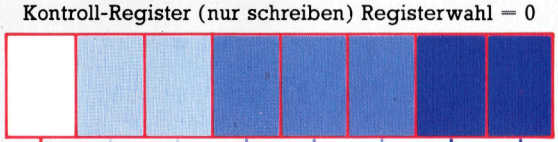
Empfangs/Sende-Datenregister:
Registerwahl = 1

Empfangs – Leseregister
Senden – in Register schreiben

Statusregister (nur lesen) Registerwahl = 0

- Empfangs-Datenregister voll
- Sende-Datenregister leer
- Daten werden empfangen
- Sendebereit (CTS)
- Start/Stopbitfehler
- Empfänger-Überlauf
- Paritätsfehler (*)
- Interrupt-Anforderung

Kontroll-Register (nur schreiben) Registerwahl = 0



Wahl der Teilungsrate
(Taktzyklen)
00 Teilen durch 1 (*)
01 Teilen durch 16 (*)
10 Teilen durch 64
11 Reset

Wahl des Datenformats

	Daten-bits	Stop-bits	Parität
000	7	2	gerade (*)
001	7	2	ungerade (*)
010	7	1	gerade (*)
011	7	1	ungerade (*)
100	8	2	keine (*)
101	8	1	keine
110	8	1	gerade (*)
111	8	1	ungerade (*)

Interruptsteuerung senden
00 ausgeschaltet (*)
01 eingeschaltet
10 ausgeschaltet (*)
11 ausgeschaltet (*)

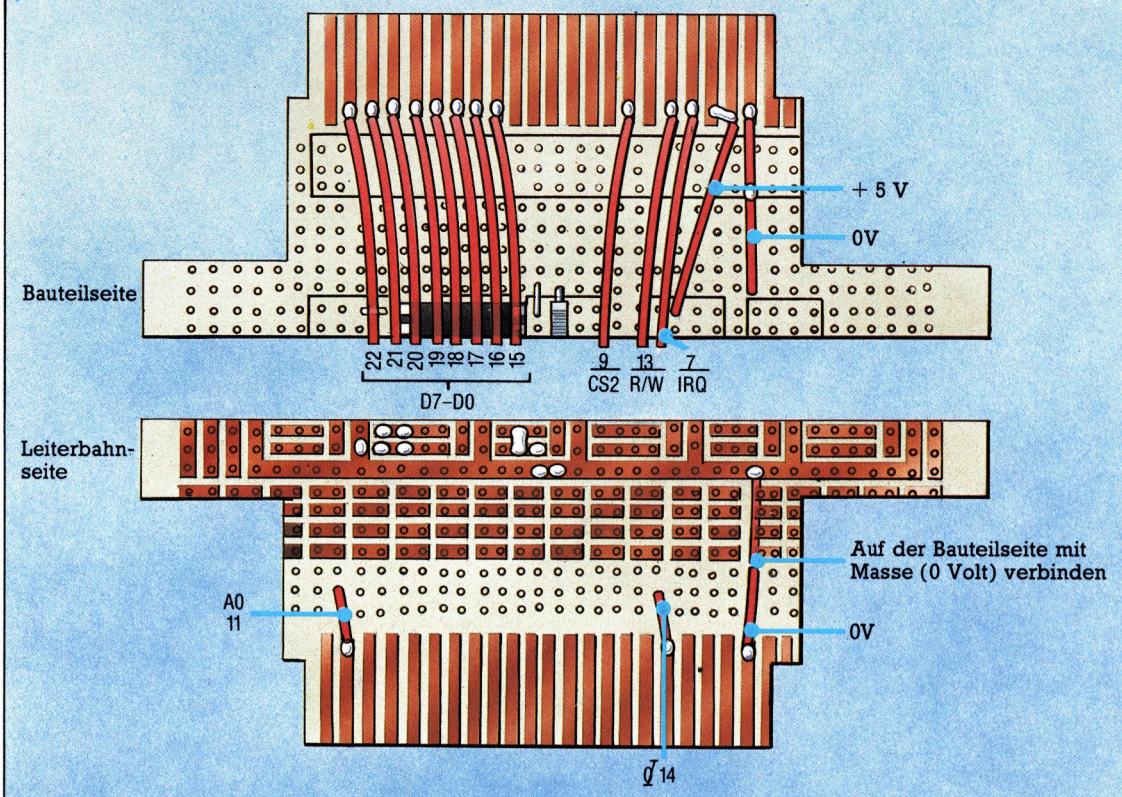
Interruptsteuerung empfangen
0 ausgeschaltet
1 eingeschaltet

(*) = im MIDI-Projekt nicht einsetzbar

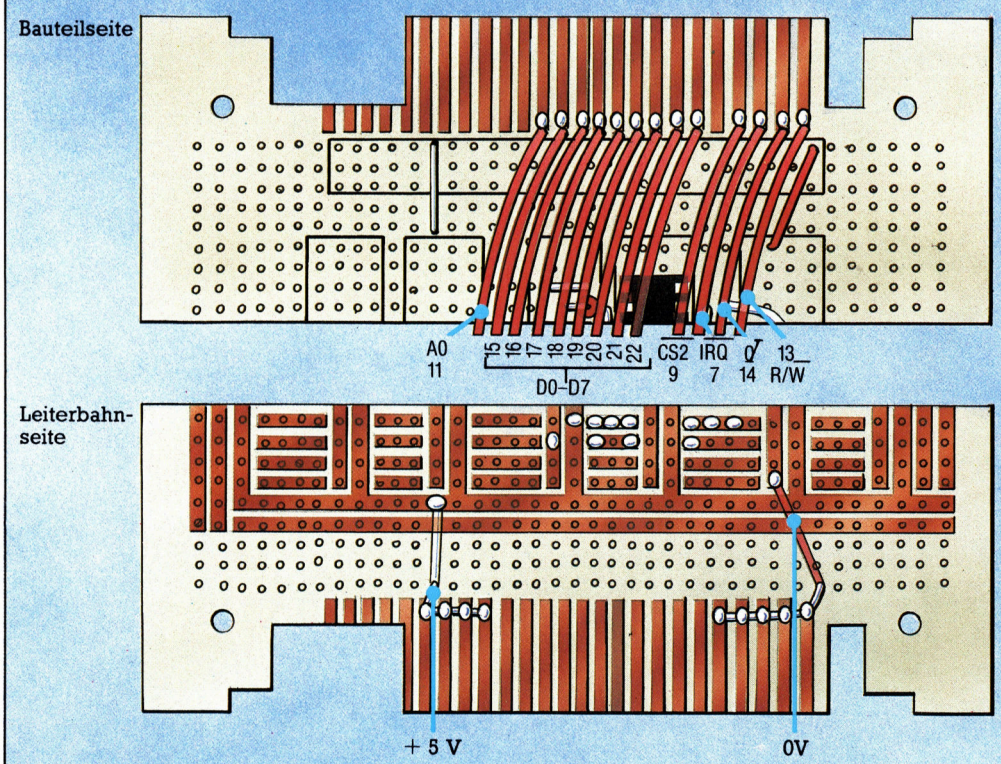
Normalfall jedoch verbunden sind. Der Taktimpuls für die Baudrate wird direkt aus dem Takteingang erzeugt oder durch Teilung durch 16 oder 64, je nach Werten der Kontrollbits 0 und 1, vermindert.

Bit 2, 3 und 4 des Kontrollregisters bestimmen die Anzahl der Stop- und Datenbits und ob die Parität gerade oder ungerade ist bzw. übergangen wird. Im MIDI-Standard müssen Bit 2 und 4 gesetzt und Bit 3 gelöscht sein (acht Datenbits, keine Parität, ein Stopbit). Bit 5 und 6 im Kontrollregister bestimmen die Übertragungsart. Bei MIDI muß Bit 6 gelöscht sein, Bit 5 erzeugt dann die Sende-Interrupts.

Commodore 64



Acorn B



Kontakt-belegung der Platine

Diese Abbildung zeigt, welche Pins am ACIA-IC mit welchen Anschlüssen an einem der beiden Computer verbunden werden müssen. Die Leitungen an den Kontaktstreifen sind mit Nummern versehen, die den Nummern der IC-Anschlüsse entsprechen. Welche Zahl zu welchem Beinchen gehört, finden Sie so heraus: Halten Sie das IC mit der Markierung nach oben und den Anschlüssen von sich fort. Anschluß 1 liegt dann auf der linken Seite neben der Chip-Markierung. Von dort werden die Anschlüsse abwärts bis zur 12 numeriert, Pin 13 liegt auf der rechten Seite Pin 12 gegenüber. Von Pin 13 zählen Sie wieder aufwärts bis 24, dem Anschluß rechts neben der Markierung.

Für die Verbindungen wird Wire-wrap-Litze verwendet. Ist alles fertig aufgebaut, können Sie die Commodore-Version der Platine mit der Bauteilseite nach oben in den Erweiterungsanschluß des Rechners stecken und mit dem Testprogramm prüfen. Für die Acorn-Version muß zuerst ein Verbindungskabel aus einem Meter 40poligem Flachkabel mit einem 40poligen Pfostenfeldstecker und einem 2 × 20poligen Platinenstecker aufgebaut werden.

ACIA-Platinentest

Wenn die Platine richtig angeschlossen ist, können wir die Schaltung auf ihre Funktion prüfen. Sollte die Platine ihren Dienst versagen, müssen Sie mit einem Multimeter oder auch durch genaue Inspektion nach Verdrahtungsfehlern suchen.

1. Falls der Computer bei angeschlossener Platine nicht normal arbeitet:
- Testen, ob die Spannung zwischen Masseleitung und +5-Volt-Leitung wirklich 5 Volt beträgt. Platine nach fehlerhaften Verbindungen zwischen einzelnen Leiterbahnen aufs genaueste absuchen.
- Platine vom Computer trennen und mit dem Multimeter auf Kurzschlüsse zwischen den Bus-Anschlüssen prüfen.

2. Wenn der Rechner normal arbeitet, die MIDI IN und die MIDI OUT Buchsen mit einem 5poligen DIN-Kabel (Hifi-Anlage) verbinden und die folgenden Befehle eingeben (Acorn B in eckigen Klammern):
POKE 56832,3 [?&FEE0=3]

Dieser Befehl schreibt eine 3 in das ACIA-Kontrollregister und erzeugt damit einen Reset. Als nächstes eingeben:
POKE 56832,22 [?&FEE0=&16]

Damit wird der Wert $16 in das Kontrollregister geschrieben, was den ACIA folgendermaßen konfiguriert:

- Empfangs- und Sendeinterrupts abgeschaltet (wir können sie zu diesem Zeitpunkt noch nicht weiterverarbeiten).

- Format für Empfangs- und Sendedaten: Acht Bit plus ein Stopbit ohne Parität und Paritätsprüfung.

- Baudrate auf Taktzyklus an Pin 3 und 4, geteilt durch 64. (2MHz/64= 31,25 KHz, die für MIDI festgelegte Übertragungsrate).

Der ACIA sollte jetzt sende- und empfangsbereit sein. Prüfen Sie das durch Auslesen des Statusregisters mit:
PRINT PEEK (56832) [PRINT?&FEE0]

Dabei sollte der Wert 2 angezeigt werden. Das bedeutet, Sende-Datenregister (Bit 1 gesetzt) und Empfangs-Datenregister (Bit 2 gelöscht) sind beide leer. Da noch keine Daten empfangen wurden und Interrupts abgeschaltet sind, sollten die restlichen Statusbits 2 bis 7 gelöscht sein.

3. Nun wird ein Byte vom Senderegister durch das Verbindungskabel zum Empfangsregister geschickt:
POKE 56833,X [?&FEE1=X]
wobei X eine Zahl zwischen 0 und 255 ist. Der Wert gelangt ins Senderegister.

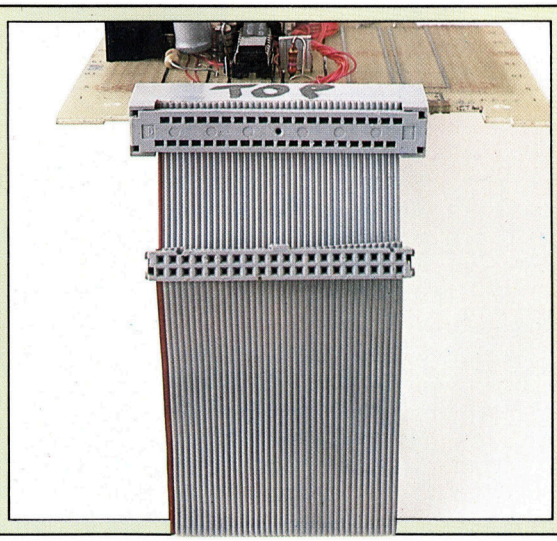
4. Bis zum Eintippen des nächsten Befehls sollte das Byte empfangen worden sein. Also das Statusregister wieder auslesen:
PRINT PEEK (56832) [PRINT?&FEE0]

Diesmal sollte 3 angezeigt werden. Bit 1 wurde sofort nach dem letzten Befehl gelöscht (Senderegister voll), kurz danach aber wieder gesetzt, um anzuzeigen, daß das Senderegister frei für das nächste Byte ist. Bit 0 wurde gesetzt und gibt an, daß das Byte empfangen wurde und aus dem Datenregister gelesen werden kann. Einen möglichen Fehler gibt es dabei: Falls Bit 0 nicht gesetzt ist, gibt es wahrscheinlich eine offene Leitung im Übertragungsweg, die den Empfangseingang auf hohem Potential hält.

5. Wenn der Empfang eines Byte gesichert ist, können Sie durch Auslesen des Datenregisters prüfen, ob der gesendete Wert korrekt angekommen ist:
PRINT PEEK (56833) [PRINT?&FE1)
Dieser Befehl sollte zur Anzeige des vorher von Ihnen eingegebenen X-Wertes führen.

Flachband-Anschluß

Die MIDI-Schnittstellenplatine kann am Acorn B mit einem 40poligen Flachbandkabel, einem 40poligen Platinenstecker und einem 40poligen Pfostenfeldverbinder angeschlossen werden. Der Pfostenfeldverbinder paßt direkt in den Tube-Port auf der Unterseite des Acorn B. Bei der Montage ist genau auf die Orientierung der Stecker zu achten. Legen Sie dazu das Kabel flach aus und montieren Sie den Platinenstecker so, daß Pin 1 (am Gehäuse markiert) unten liegt. Die Oberseite des Steckergehäuses beschriften, damit es nicht falsch eingesetzt wird. Der Pfostenfeldverbinder wird am anderen Ende des Kabels mit dem rechteckigen Vorsprung nach oben montiert.



Die Neue Welt 1

In den letzten Artikeln unseres Programmprojekts zeigen wir die Programm-Versionen für den Spectrum, Acorn B und Schneider CPC. Zunächst stellen wir die Spectrum-Version vor.

Das Programm wurde auf einem Commodore 64 geschrieben. Die Probleme bei der Anpassung an den Spectrum liegen in zwei Hauptbereichen: Erstens gestattet der Spectrum nur Variablennamen für Arrays oder FOR. . .NEXT-Schleifen, die aus einem Buchstaben bestehen; daher haben wir eine Umwandlungstabelle erstellt. Zweitens ist das String-Handling beim Spectrum anders, das heißt, LEFT$, RIGHT$ und MID$ sind nicht verfügbar, obwohl jeweils entsprechende Äquivalente vorhanden sind. Spectrum-Besitzer sollten daher die bei jedem Modul angegebenen Besonderheiten beachten. Zusätzlich sollte PRINT CHR$(147) durch CLS ersetzt werden, und Routinen, die auf Tastendruck warten, wie

```
<Zeilennr.> GET I$:IF I$="" THEN
<Zeilennr.>
```

sollten gegen

```
<Zeilennr.> LET I$= INKEY$:IF I$=""
THEN GO TO<Zeilennr.>
```

ausgetauscht werden.

Umwandlungstabelle für Spectrum-Variablen

Microsoft Äquivalent	Zweck	Spectrum
TS(,)	Mannschaftstyp/-stärke	(T(,)
WG()	Wochenlöhne	W()
CC()	Mannschaftstypen-Zähler	Y()
PA()	erhaltene Vorräte	A()
PC()	Vorrats-Preise	C()
PN()	benötigte Vorräte	N()
OC()	Preise für Handelswaren	O()
OA()	Menge an Handelswaren	G()
HR()	Flag für halbe Rationen	H()
RR()	Flag für Ereignisse	R()
AO()	Menge gehandelter Waren	E()
EQ(,)	Tauschwerte	Q(,)
V1()	Werte bei der Abfahrt	B()
V2()	Werte bei der Rückkehr	D()
S1	Zähler für Stärkereduzierung	S
S3	Zähler für Bildschirmausgabe	S
S4	Zähler für kurze Verzögerung	J
S5	Zähler für lange Verzögerung	S

```
1 REM ****************
2 REM ** NEW WORLD **
3 REM **  TRADING  **
4 REM **    GAME   **
5 REM ****************
6 :
```

Der erste Abschnitt des Programms initialisiert Variablen und Arrays.

```
9 K$="  PRESS ANY KEY TO CONTINUE"
10 DIM TS(16,2): REM CREW TYPE/STRENGTH
11 CN=0: REM NO. OF CREW
12 MO=2000: REM START MONEY
13 DIMWG(5):WG(1)=10:WG(2)=25:WG(3)=15:WG(4)=20:WG
(5)=15: REM WAGES
14 WT=0: REM WEEKLY WAGE BILL
15 CM=16:REM MAX CREW
16 DIMC$(5):C$(1)="SAILOR":C$(2)="DOCTOR":C$(3)="M
ECHANIC"
17 C$(4)="NAVIGATOR":C$(5)="COOK"
18 DIMCC(5): REM COUNT OF EACH CREW TYPE
20 DIMU$(4):U$(1)="KILO":U$(2)="KILO":U$(3)="KILO"
:U$(4)="BARREL"
21 DIMP$(4):P$(1)="VEG":P$(2)="FRUIT":P$(3)="MEAT"
:P$(4)="WATER"
22 DIMPA(4)
23 DIMPC(4):PC(1)=.5:PC(2)=1:PC(3)=2:PC(4)=.5
24 DIMPN(4):PN(1)=2:PN(2)=1:PN(3)=1:PN(4)=.5
30 DIMOA(6)
31 DIMO$(6)
32 O$(1)="BOTTLE OF MEDICINE":O$(2)="GUN":O$(3)="B
AG OF SALT"
33 O$(4)="BALE OF CLOTH":O$(5)="KNIFE":O$(6)="JEWEL"
34 DIMOC(6)
35 OC(1)=1:OC(2)=5:OC(3)=.2
36 OC(4)=2:OC(5)=.5:OC(6)=1
40 JL=8:REM LENGTH OF VOYAGE
41 EW=0: REM EXTRA WEEKS
42 DIMRR(16)
43 REM INDICATORS TO SHOW IF RANDOM EVENT(N) ALREA
DY OCCURED
44 RC=0
45 REM COUNT OF RANDOM EVENTS SO FAR
46 RM=13
47 G$="N":REM GOOD WEATHER INDICATOR FOR USE IN MU
TINY FACTOR
48 A$="N":B$="N"
49 DIMM(6):REM INDS TO SHOW IF MAJOR EVENTS HAVE B
EEN DONE
60 DIMT$(3):T$(1)="PEARLS":T$(2)="CARVINGS":T$(3)=
"SPICES"
61 DIMV1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIMV2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+(INT
(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIMAO(3)
80 PRINTCHR$(147):S$="  NEW WORLD TRADING GAME*":G
OSUB9100:PRINT
81 GOSUB9200
82 S$="YOU ARE THE CAPTAIN OF A SHIP*" :GOSUB9100:
PRINT
83 S$="SAILING TO THE NEW WORLD. THE*":GOSUB9100:P
RINT
84 S$="JOURNEY IS SAID TO TAKE EIGHT*":GOSUB9100:P
RINT
85 S$="WEEKS,BUT MAY TAKE LONGER. YOU*":GOSUB9100:
PRINT
86 S$="MUST HIRE A CREW,PAY THEM, BUY*":GOSUB9100:
PRINT
87 S$="PROVISIONS,EQUIPMENT AND GOODS*":GOSUB9100:
PRINT
88 S$="FOR TRADING. YOU HAVE 2000 GOLD*":GOSUB9100
:PRINT
89 S$="PIECES TO SPEND.*":GOSUB9100:PRINT:GOSUB9200

90 PRINT:S$="          GOOD LUCK!*":GOSUB9100:GOSUB
9200:PRINT
91 S$=K$:GOSUB9100
94 GETI$:IFI$=""THEN94
95 GOSUB9200
```

Dieser Abschnitt ruft die Unterroutinen auf, die dem Spieler das Anheuern einer Mannschaft, den Einkauf von Vorräten und Handelsgütern ermöglichen.

```
500 GOSUB1000
550 GOSUB2000
600 GOSUB3000
610 PRINTCHR$(147)
615 S$="YOU ARE NOW READY TO START*":GOSUB9100
625 S$="THE VOYAGE.*":GOSUB9100
630 GOSUB9200
635 PRINT:S$="YOU HAVE THE FOLLOWING CREW:*":GOSUB
9100
```

```
640 GOSUB9200
645 FORT=1TO5
650 IFCC(T)=0THEN670
655 PRINTCC(T);
660 PRINTC$(T);
662 IF CC(T)=1THENPRINT" ";:GOTO668
664 PRINT"S"
668 GOSUB9200
670 NEXT
674 GOSUB9200
675 PRINT:S$="AND THE FOLLOWING PROVISIONS:*":GOSU
B9100
680 GOSUB9200
685 FORT=1TO4
690 IFPA(T)=0THEN710
695 PRINTPA(T);U$(T);"S OF ";
700 PRINTP$(T)
708 GOSUB9200
710 NEXT
715 GOSUB9200
720 PRINT:S$="YOU HAVE ALSO GOT:*":GOSUB9100
725 GOSUB9200
730 IFOA(1)=0THEN740
733 IFOA(1)=1THENS$="BOTTLE OF MEDICINE *":GOTO735
734 S$="BOTTLES OF MEDICINE*"
735 PRINTOA(1);:GOSUB9100
736 GOSUB9200
740 IFOA(2)=0THEN750
743 IFOA(2)=1THENS$="GUN*":GOTO745
744 S$="GUNS*"
745 PRINTOA(2);:GOSUB9100
746 GOSUB9200
750 IFOA(3)=0THEN760
753 IFOA(3)=1THENS$="BAG OF SALT*":GOTO755
754 S$="BAGS OF SALT*"
755 PRINTOA(3);:GOSUB9100
756 GOSUB9200
760 IFOA(4)=0THEN770
763 IFOA(4)=1THENS$="BALE OF CLOTH*":GOTO765
764 S$="BALES OF CLOTH*"
765 PRINTOA(4);:GOSUB9100
766 GOSUB9200
770 IFOA(5)=0THEN780
773 IFOA(5)=1THENS$="KNIFE*":GOTO775
774 S$="KNIVES*"
775 PRINTOA(5);:GOSUB9100
776 GOSUB9200
780 IFOA(6)=0THEN790
783 IFOA(6)=1THENS$="JEWEL*":GOTO785
784 S$="JEWELS*"
785 PRINTOA(6);:GOSUB9100
786 GOSUB9200
790 GOSUB9200
792 PRINT:PRINT"YOU HAVE ";MO;" GOLD PIECES LEFT"
796 GOSUB9200
797 S$="PRESS ANY KEY TO START VOYAGE*"
798 GOSUB9100
799 GETI$:IF I$=""THEN799
800 WT=0 :REM ZEROISE WAGE TOTAL
801 H$="N" :REM HALF RATION INDICATOR
802 DIMHR(4):HR(1)=1:HR(2)=1:HR(3)=1:HR(4)=1
```

Hier beginnt die Hauptschleife für die Fahrt. Die Wochen werden mit WK gezählt.

```
820 WK=1
825 GOSUB4000: REM CREW STATUS REPORT
830 GOSUB4200:REM PROVISIONS REPORT
835 GOSUB4300:REM OTHER GOODS REPORT
840 GOSUB9200:PRINTCHR$(147)
842 PRINT:PRINT:PRINT
843 S$="IT IS ESTIMATED THAT THE VOYAGE*":GOSUB9100
844 PRINT"WILL TAKE A FURTHER";INT(JL-WK+1);"WEEKS"
845 GOSUB9200
846 PRINT:S$=K$:GOSUB9100
847 GETI$:IFI$=""THEN847
850 GOSUB5000:REM CHECK WAGE BILL
855 GOSUB5100: REM ISSUE RATIONS
860 GOSUB5500
861 REM GO TO GENERATE RANDOM EVENTS
870 GOSUB6500:REM GOTO MAJOR CONTINGENCY
875 IFHR(3)=.5ANDRND(1)<.5THENPRINTCHR$(147):GOSUB
6050
878 REM ALBATROSS IF SHORT OF MEAT
879 GOSUB7200
880 GOSUB 5300: REM END-OF-WEEK REPORT
889 WK=WK+1:IFWK<=JLTHEN825
890 REM ARRIVAL AT NEWWORLD
891 GOSUB10000
892 GOSUB10070
893 GOSUB10300
894 GOSUB10500
999 END
```

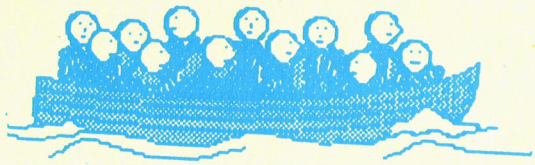
Hier endet das Hauptprogramm. Der Rest des Programms besteht aus Unterroutinen, die vom Hauptteil aufgerufen werden.

```
1000 PRINTCHR$(147):PRINT" STAGE 1 - HIRING CREW"
1010 PRINT" ----------------------"
1012 PRINT
1015 GOSUB9200
1020 PRINT:PRINT"CREW TYPES AVAILABLE:"
1025 GOSUB9200
1030 PRINT
1040 PRINT"TYPE DESCRIPTION WAGES PER WEEK"
1050 PRINT"---- ----------- --------------"
1060 PRINT"   1   SAILOR        10 GOLD PCS"
1070 PRINT"   2   DOCTOR        25 GOLD PCS"
1080 PRINT"   3   MECHANIC      15 GOLD PCS"
1090 PRINT"   4   NAVIGATOR     20 GOLD PCS"
1100 PRINT"   5   COOK          15 GOLD PCS"
1105 GOSUB9200
1110 PRINT:PRINT:PRINT
1120 S$="ENTER CREW TYPE REQUIRED(1-5)*":GOSUB9100
1122 S$="OR 'F' TO FINISH HIRING*":GOSUB9100:PRINT
:INPUTI$
1125 CT=VAL(I$)
1128 IFLEFT$(I$,1)="F"THENPRINT:PRINT"END OF CREW
HIRE.":GOSUB9200:GOTO1310
1130 IFCT>0ANDCT<6THEN1150
1139 PRINT:PRINT
1140 PRINTI$;:S$=" IS NOT A CREW TYPE*":GOSUB9100
1142 GOSUB9200
1145 S$="PLEASE ENTER AGAIN"
1146 GOSUB9100
1147 GOTO1300
1150 PRINT:PRINT
1155 CN=CN+1:REM CREW HIRED SO FAR
1156 TS(CN,1)=AT:REM CREW TYPE
1157 TS(CN,2)=100:REM STARTING STRENGTH
1158 WT=WT+WG(CT):REM TOTAL WAGES
1159 CC(CT)=CC(CT)+1:REM CREW TYPE COUNT
1160 S$="CREW SO FAR:"
1170 FORT=1TO5
1180 PRINTS$;CC(T);" ";C$(T);
1185 IFCC(T)>1ORCC(T)=0THENPRINT"S":GOTO1189
1186 PRINT" "
1189 S$="                "
1190 NEXT
1195 PRINT: PRINT"TOTAL WEEKLY WAGE BILL ";WT
1200 IFCN=CM-1THENPRINT:S$="ONLY ONE MORE CREW*":G
OSUB9100:GOTO1295
1202 IFCN=CMTHENPRINT:S$="    SHIP NOW FULL!!":GOS
UB9100:GOTO1310
1295 REM
1300 GOTO1015
1310 PRINT: S$=K$:GOSUB9100:PRINT:   GOSUB9200
1320 GETI$:IFI$=""THEN1320
1999 RETURN
2000 PRINTCHR$(147)
2010 S$="    STAGE 2 - PROVISIONING*"
2015 GOSUB9100
2020 S$="    -------   ------------*"
2025 GOSUB9100
2030 GOSUB9200:PRINT
2040 PRINT"YOU'VE HIRED A CREW OF ";CN;"."
2045 GOSUB9200:GOSUB9200
2050 FORT=1TO4
2055 PRINT
2060 PRINT"EACH CREW MEMBER WILL NEED "
2070 PRINT"AT LEAST ";PN(T);" ";U$(T);
2075 IFPN(T)=1THENPRINT" ";:GOTO2085
2080 PRINT"S";
2085 PRINT" OF ";P$(T)
2086 PRINT"AT ";PC(T);" GOLD PCS PER ";U$(T)
2090 PRINT"FOR EACH WEEK OF THE JOURNEY."
2095 GOSUB9200:PRINT:GOSUB9200
2100 PRINT"HOW MANY ";U$(T);"S OF ";P$(T)
2110 S$="DO YOU WANT TO BUY*":GOSUB9100
2120 PRINT
2130 INPUTI$
2140 PA(T)=VAL(I$):GOSUB9200
2150 IFPA(T)>((CN*8*PN(T))-1) THEN2260
2160 IFPA(T)=0THENPRINT"IF YOU DON'T BUY ANY":GOTO
2180
2170 PRINT"IF YOU ONLY BUY ";PA(T);U$(T);
2175 IFPA(T)=1THENPRINT" OF":GOTO2180
2176 PRINT"S OF"
2180 PRINTP$(T);",":GOSUB9200
2190 PRINT"SOMEONE MIGHT GET ";
2200 S$="HUNGRY"
2210 IFT=4THENS$="THIRSTY"
2220 PRINTS$;"!":GOSUB9200
2230 S$= "DO YOU WANT TO TRY AGAIN":GOSUB9100
2240 INPUTP$:P$=LEFT$(P$,1)
2242 IFP$<>"Y"ANDP$<>"N"THEN2230
2245 IFP$="N"THEN2400
2250 PA(T)=0:T=T-1:GOTO2410
2260 IFPA(T)*PC(T)>MOTHEN2270
2265 GOTO2400
2270 S$="YOU DON'T HAVE ENOUGH MONEY FOR":GOSUB9100
2280 PRINTPA(T)
2290 PRINTU$(T);"S OF ";P$(T):GOSUB9200
2300 S$="PLEASE TRY AGAIN*":GOSUB9100:PA(T)=0:T=T-
1:GOTO2410
2400 MO=MO-(PA(T)*PC(T))
2410 PRINT:S$="PROVISIONS SO FAR:*":GOSUB9100
2412 GOSUB9200
2415 FORTT=1TO4
2420 PRINTPA(TT);U$(TT);
2430 IFPA(TT)=1THENPRINT"  OF ";:GOTO2440
2435 PRINT"S OF ";
2440 PRINTP$(TT)
2450 GOSUB9200
```

```
2460 NEXT
2480 PRINT"MONEY LEFT = ";MO;" GOLD PIECES"
2485 GOSUB9200:GOSUB9200
2490 NEXTT
2500 GOSUB9201:PRINT:S$="END OF PROVISIONING*":GOS
UB9100:GOSUB9200
2510 PRINT: S$=K$:GOSUB9100:PRINT:   GOSUB9200
2520 GETI$:IF ?-""THEN2520
2999 RETURN
3000 PRINTCHR$(147): REM STAGE 3
3002 GOSUB9200
3005 PRINT"      STAGE 3 - OTHER GOODS"
3010 PRINT"      -------   -----------"
3020 GOSUB9200
3025 PRINT
3030 S$="THERE ARE OTHER THINGS THAT MAY*":GOSUB9100
3035 S$="BE USEFUL ON THE VOYAGE. FOR  *":GOSUB9100
3040 S$="EXAMPLE MEDICINE AND TRADING  *":GOSUB9100
3046 GOSUB9200
3050 S$="YOU MAY ALSO NEED WEAPONS.*":GOSUB9100
3055 GOSUB9200:GOSUB9200
3060 FORT=1TO6
3065 PRINT
3070 PRINT"A ";O$(T);
3075 S$=" COSTS*":GOSUB9100
3080 PRINTOC(T);
3081 PRINT" GOLD PIECE";
3085 IFOC(T)=1THENPRINT" ":GOTO3090
3086 PRINT"S"
3090 GOSUB9200
3095 S$="WOULD YOU LIKE TO BUY (Y OR N)*":GOSUB9100
3110 INPUTP$:P$=LEFT$(P$,1)
3115 IFP$<>"Y"ANDP$<>"N"THEN3095
3120 IFP$="N"THEN3175
3125 GOSUB9200
3130 S$="HOW MANY DO YOU WANT*":GOSUB9100
3135 INPUTI$
3140 TT=VAL(I$)
3145 IFOC(T)*TT>MOTHEN3150
3147 GOTO3160
3150 S$="YOU DON'T HAVE ENOUGH MONEY*":GOSUB9100
3152 GOSUB9200
3154 S$="PLEASE ENTER AGAIN  *":GOSUB9100
3155 GOSUB9200:GOTO3130
3160 MO=MO-(OC(T)*TT)
3165 OA(T)=TT
3170 GOSUB9200
3175 PRINT
3176 PRINT"MONEY LEFT = ";MO
3200 GOSUB9200:NEXT T
3205 GOSUB9200:PRINT:PRINT
3210 S$="END OF STAGE 3*":GOSUB9100
3220 GOSUB9200:PRINT
3230 S$=K$:GOSUB9100
3240 GETI$:IFI$=""THEN3240
3999 RETURN
```

Die folgenden Unterroutinen analysieren den aktuellen Status von Schiff und Mannschaft und geben einen Wochenbericht.

```
4000 REM CREW STATUS REPORT
4010 PRINTCHR$(147)
4020 S$="          CAPTAINS LOG*":GOSUB9100
4025 S$="          ------------*":GOSUB9100
4030 GOSUB9200
4035 PRINT"AT THE START OF WEEK";WK
4040 S$="THE STATE OF THE CREW IS:*":GOSUB9100
4045 GOSUB9200:PRINT
4055 PRINT
4060 FORT=1TO16
4070 IFTS(T,1)=0THEN4110
4075 PRINTC$(TS(T,1));"    (";
4078 IFTS(T,2)=-999THENS$="DEAD !!!!!!)*":GOTO4099
4080 IFTS(T,2)>75THENS$="VERY HEALTHY)*":GOTO4099
4085 IFTS(T,2)>50THENS$="HEALTHY)*":GOTO4099
4095 IFTS(T,2)>25THENS$="SICK ! )*":GOTO4099
4098 S$="VERY SICK !! )*"
4099 GOSUB9100:GOSUB9200
4110 NEXT
4115 GOSUB9200:PRINT
4119 WW=0
4120 FORT=1TO5
4130 WW=WW+(CC(T)*WG(T))
4135 NEXT
4140 S$="WAGE BILL FOR THE WEEK*":GOSUB9100
4145 PRINTWW;"GOLD PIECES"
4150 GOSUB9200
4155 WT=WT+WW
4160 S$="TOTAL WAGES FOR VOYAGE SO FAR:*":GOSUB9100
4165 PRINTWT;"GOLD PIECES"
4170 GOSUB9200
4175 PRINT"MONEY LEFT = ";MO;"GOLD PIECES"
4180 PRINT:S$=K$:GOSUB9100
4190 GETI$:IFI$=""THEN4190
4199 RETURN
4200 REM PROVISIONS REPORT
4205 PRINTCHR$(147)
4206 PRINT"AT THE START OF WEEK";WK:GOSUB9200
4210 S$="YOU HAVE THE FOLLOWING*":GOSUB9100
4215 S$="PROVISIONS LEFT:*":GOSUB9100
4220 PRINT:GOSUB9200
4225 FORT=1TO4
4226 IFPA(T)=0ORPA(T)=-999THEN4240
4230 PRINTPA(T);U$(T);"S OF ";P$(T)
4232 X=PA(T)/(CN*PN(T))
4235 PRINT"(ENOUGH FOR";INT(X);" WEEKS)"
4239 GOSUB9200
4240 NEXT
4290 PRINT:S$=K$:GOSUB9100
4295 GETI$:IFI$=""THEN4295
4299 RETURN
4300 REM OTHER GOODS REPORT
4305 PRINTCHR$(147)
4306 PRINT"AT THE START OF WEEK";WK:GOSUB9200
4310 S$="YOU ALSO HAVE:*":GOSUB9100
4320 PRINT:GOSUB9200
4322 IFOA(1)=0THEN4332
4325 PRINTOA(1);:S$="BOTTLES OF MEDICINE*":GOSUB9100
4330 GOSUB9200
4332 IFOA(2)=0THEN4342
4335 PRINTOA(2);:S$="GUNS*":GOSUB9100
4340 GOSUB9200
4342 IFOA(3)=0THEN4352
4345 PRINTOA(3);:S$="BAGS OF SALT*":GOSUB9100
4350 GOSUB9200
4352 IFOA(4)=0THEN4362
4355 PRINTOA(4);:S$="BALES OF CLOTH*":GOSUB9100
4360 GOSUB9200
4362 IFOA(5)=0THEN4372
4365 PRINTOA(5);:S$="KNIVES*":GOSUB9100
4370 GOSUB9200
4372 IFOA(6)=0THEN4380
4375 PRINTOA(6);:S$="JEWELS*":GOSUB9100
4380 GOSUB9200:PRINT
4382 PRINT"YOU HAVE";MO;:S$=" GOLD PIECES LEFT*":G
OSUB9100
4384 GOSUB9200
4397 PRINT:S$=K$:GOSUB9100
4398 GETI$:IFI$=""THEN4398
4399 RETURN
5000 REM CHECK WAGE BILL
5005 IFWT>MOTHEN5010
5008 GOTO5099
5010 PRINTCHR$(147)
5020 PRINT:PRINT:PRINT
5025 S$="THE CREW HAVE HEARD A RUMOUR*":GOSUB9100
5030 S$="THAT YOU DON'T HAVE ENOUGH*":GOSUB9100
5035 S$="GOLD TO PAY THEM AT THE END*":GOSUB9100
5040 S$="OF THE VOYAGE.*":GOSUB9100
5045 GOSUB9200:PRINT
5050 S$="THEY ARE GETTING ANGRY !!*":GOSUB9100
5055 GOSUB9200:PRINT
5060 S$="LET'S HOPE YOU MANAGE TO MAKE*":GOSUB9100
5065 S$="A TRADING PROFIT!*":GOSUB9100
5066 GOSUB9200
5070 PRINT:S$=K$:GOSUB9100
5080 GETI$:IFI$=""THEN5080
5099 RETURN
5100 REM ISSUE RATIONS
5103 PRINTCHR$(147)
5105 S$="       ISSUING RATIONS*":GOSUB9100
5106 S$="       ---------------*":GOSUB9100
5107 GOSUB9200:PRINT"WEEK";WK:PRINT
5108 H$="N"
5110 FORT=1TO4
5112 HR(T)=1
5115 IFPA(T)>0THEN5180
5120 PRINT"NO ";P$(T);" LEFT!!!":GOSUB9200
5130 S$="THE CREW IS GETTING WEAKER !!*":GOSUB9100
5135 WF=10:GOSUB9300
5139 GOTO5290
5180 X=(PN(T)*CN)*(JL-WK+1)
5185 IFPA(T)<XTHEN5200
5190 GOTO5270
5200 PRINT"RUNNING SHORT OF ";P$(T)
5205 GOSUB9200
5210 S$="DO YOU WANT TO PUT THE CREW ON*":GOSUB9100
5215 PRINT"HALF RATIONS OF ";P$(T)
5220 INPUTI$:I$=LEFT$(I$,1)
5221 IFI$<>"Y"ANDI$<>"N"THEN5220
5225 IFI$="N"THEN5270
5230 HR(T)=.5:H$="Y"
5240 WF=5:GOSUB9300
5250 S$="THE CREW IS GETTING WEAKER!*":GOSUB9100
5270 X=PN(T)*HR(T)*CN
5272 IFX>PA(T)THENX=PA(T)
5275 PA(T)=PA(T)-X
5280 IFPA(T)=0THENPA(T)=-999
5285 PRINTX;U$(T);"S OF ";P$(T);" ISSUED"
5290 PRINT:GOSUB9200:NEXT
5295 PRINT:S$=K$:GOSUB9100
5298 GETI$:IFI$=""THEN5298
5299 RETURN
```



Grafik mit Dreh

Im ersten Artikel über die dreidimensionale Grafik des Commodore 64 hatten wir einen Teil des Grafikprogramms in Maschinensprache übertragen. Wir übersetzen nun den Rest des BASIC-Listings und stellen weitere Fließkommaroutinen vor.

Die Programmkombination Test I-Rot und I-Rot-Hex läuft schon recht schnell, wird aber durch das Abfragen der Matrix E%(I,J) stark verlangsamt. Die Matrix legt fest, welche Punkte der Figur miteinander verbunden sind und gibt uns damit die Information, welche Linien gezeichnet werden sollen. Um höhere Ausführungsgeschwindigkeiten zu erreichen, müssen wir daher auch den Rest des BASIC-Programms in Maschinencode übersetzen.

Für die mathematischen Berechnungen in den Zeilen 1640 und 1650

```
X1%=X(I)+159:Y1%=199-(Z(I)+100)
X2%=X(J)+159:Y2%=199-(Z(J)+100)
```

setzen wir weitere Interpreterroutinen ein. Beide BASIC-Zeilen nehmen eine Fließkommavariable, beispielsweise X(I), addieren (im Fließkommaformat) 159, interpretieren das Ergebnis als Ganzzahl und speichern es im Zwei-Byte-Format als X1%. Wir ersetzen sie mit

- FLPINT (CALL Adresse $B1AA):

FLPINT nimmt den Ganzzahlenwert der in FAC gespeicherten Zahl und gibt das Ergebnis (falls es zwischen −32767 und 32767 liegt) an die Register Y und A im Lo-Hi-Format zurück. Beachten Sie, daß im Gegensatz zu den meisten anderen Interpreterroutinen hier das Format Lo-Hi eingesetzt ist.

- SNGFT (CALL Adresse $B3A2):

SNGFT nimmt eine Ganzzahl im Ein-Byte-Format aus dem Y-Register und legt sie im Fließkommaformat in FAC ab.

SNGFT wird in der Subroutine SETUP des Assemblerlistings (Zeile 5150) eingesetzt. Wenn beispielsweise der Dezimalwert 159 in das Indexregister Y gesetzt wird, wandelt ein Aufruf von SNGFT diesen Wert um und legt das Ergebnis in FAC ab. MOVFM überträgt den Inhalt von FAC dann in MEM1.

Für die Umwandlung des BASIC-Programms in Maschinencode müssen wir nun festlegen, welche Arrayelemente den rotierenden Umriß definieren sollen. Dabei kann die Berechnung der Arraypointer recht schwierig werden. Die Koordinatenarrays X(I), Y(I) und Z(I) machen keine Probleme, da wir in jedem Array nur jeweils fünf Bytes auf den Pointer addieren müssen, um die Adresse des nächsten Elements zu erhalten.

Das zweidimensionale Array E%(I,J) ist jedoch völlig anders aufgebaut. Es hat folgendes Format:

```
E%(0,0)E%(1,0)E%(2,0) . . .E%(NP,0)
E%(0,1)E%(1,1)E%(2,1) . . .E%(NP,1)
etc. . . .
```

Das Array besteht aus Speicherblöcken, die je 2*(NP−1) lang sind, wobei jeder Block den Werten des zweiten definierten Elements entspricht und jedes Element (des Ganzzahlarrays) genau zwei Bytes lang ist.

Da wir den BASIC-Code genau in den Maschinencode übertragen wollen, lauten die I,J Schleifen zum Durchsuchen von E%(I,J):

```
FOR I=1 TO NP
FOR J = 1 TO I
```

Nun geht's rund

Eine Pointerveränderung, die im Maschinencode dem BASIC-Befehl NEXT I entspricht, verkompliziert sich jedoch, wenn es möglich sein soll, alle Elemente mit der Definition Null zu überspringen. Für die Rotation der Figur werden die Elemente von E%(I,J) daher in folgender Reihenfolge angesprochen:

```
E%(1,1)
E%(1,2)E%(2,2)
E%(1,3)E%(2,3)E%(3,3)
E%(1,4))E%(2,4)E%(3,4)E%(4,4) etc.
```

Ein schneller Überschlag zeigt, daß 2*(NP+1) bei jeder Inkrementierung von I auf den Pointer addiert werden muß. Im Maschinencode des 6502 wird dafür am besten die indirekte Adressierung eingesetzt. Hier der Code:

```
LDY JINDEX
LDA (ZPTEMP),Y
```

ZPTEMP ist ein Zwei-Byte Pointer auf die Zero Page. JINDEX steuert J. Bei jeder Inkrementierung von J muß nun auch ZPTEMP erhöht werden. Durch die Inkrementierung von ZPTEMP und Y erhöht sich das „Netto"-Offset bei jeder Inkrementierung von J um zwei Bytes. ZPTEMP wird daher bei jeder Wiederholung der I-Schleife um (2*NP+1)−(I−1) inkrementiert. (I-1) muß von der Blocklänge abgezogen werden, da ZPTEMP bei der (gerade beendeten) J-Schleife bereits (I−1)mal inkrementiert wurde. Wenn das Offset über die Formel errechnet wird, zeigt ZPTEMP nach jeder Erhöhung von I auf das richtige Byte.

Es wäre praktisch, wenn eine Interpreterroutine die Variable E%(I,J) für uns finden könnte. Tatsächlich gibt es eine derartige Routine. Sie ist jedoch extrem kompliziert und langsam.

Routinen zum Rotieren

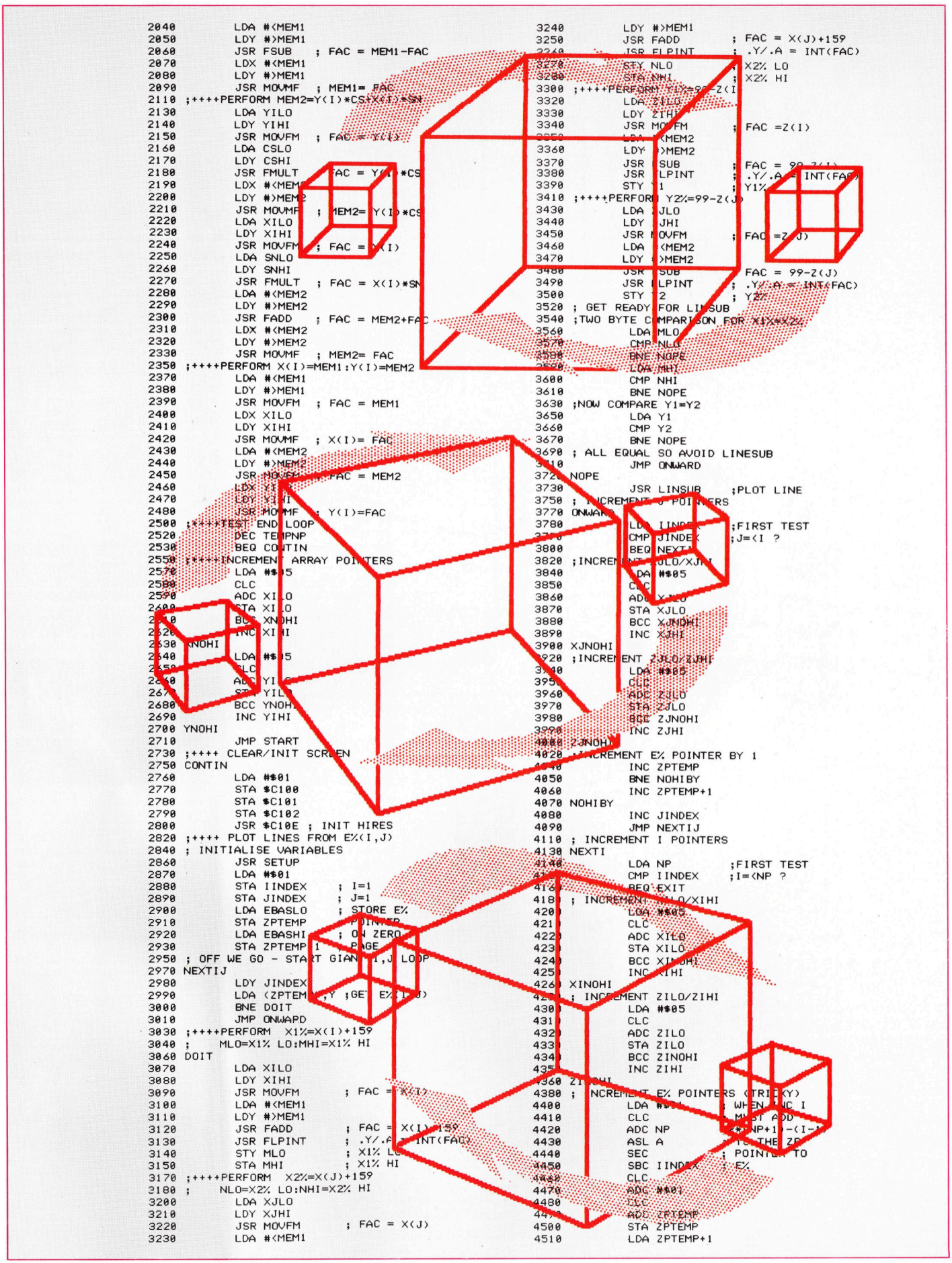
Das folgende Maschinencodeprogramm rotiert auf dem hochauflösenden Bildschirm die im Array E%(I,J) definierte Umrißfigur. Wir haben damit eine Maschinencodeversion der BASIC-Schleife, die die Arrays durchsucht und so die rotierende Figur erzeugt. Das Ladeprogramm enthält den Quellentext in BASIC DATA-Befehlen. Zusammen mit dem BASIC-Testprogramm legt es die Maschinencodevariablen an und ruft die Routine auf. Das Testprogramm arbeitet entweder mit der assemblierten Version des Quellencode – II-ROT.HEX oder mit dem vom Ladeprogramm erzeugten Code. Wenn Sie mit dem Ladeprogramm arbeiten, sollten Sie allerdings die Zeile 1030 des Testprogramms löschen.

BASIC-Ladeprogramm

```
1000 REM**  INSERT M/C II-ROT.HEX   **
1010 REM*****************************
1020 DATA72,138,72,152,72,32,101,199
1030 DATA173,83,197,172,84,197,32,162
1040 DATA187,173,75,197,172,76,197,32
1050 DATA40,186,162,94,160,197,32,212
1060 DATA187,173,87,197,172,88,197,32
1070 DATA162,187,173,77,197,172,78,197
1080 DATA32,40,186,169,94,160,197,32,80
1090 DATA184,162,94,160,197,32,212,187
1100 DATA173,87,197,172,88,197,32,162
1110 DATA187,173,75,197,172,76,197,32
1120 DATA40,186,162,99,160,197,32,212
1130 DATA187,173,83,197,172,84,197,32
1140 DATA162,187,173,77,197,172,78,197
1150 DATA32,40,186,169,99,160,197,32
1160 DATA103,184,162,99,160,197,32,212
1170 DATA187,169,94,160,197,32,162,187
1180 DATA174,83,197,172,84,197,32,212
1190 DATA187,169,99,160,197,32,162,187
1200 DATA174,87,197,172,88,197,32,212
1210 DATA187,208,93,197,240,31,169,5,24
1220 DATA109,83,197,141,83,197,144,3
1230 DATA238,84,197,169,5,24,109,87,197
1240 DATA141,87,197,144,3,238,88,197,76
1250 DATA112,197,169,1,141,0,193,141,1
1260 DATA193,141,2,193,32,14,193,32,101
1270 DATA199,169,1,141,81,197,141,82
1280 DATA197,173,79,197,133,253,173,80
1290 DATA197,133,254,172,82,197,177,253
1300 DATA208,3,76,207,198,173,83,197
1310 DATA172,84,197,32,162,187,169,94
1320 DATA160,197,32,103,184,32,170,177
1330 DATA140,0,195,141,1,195,173,85,197
1340 DATA172,86,197,32,162,187,169,94
1350 DATA160,197,32,103,184,32,170,177
1360 DATA140,2,195,141,3,195,173,89,197
1370 DATA172,90,197,32,162,187,169,99
1380 DATA160,197,32,80,184,32,170,177
1390 DATA140,4,195,173,91,197,172,92
1400 DATA197,32,162,187,169,99,160,187
1410 DATA32,80,184,32,170,177,140,5,195
1420 DATA173,0,195,205,2,195,208,19,173
1430 DATA1,195,205,3,195,208,11,173,4
1440 DATA195,205,5,195,208,3,76,207,198
1450 DATA32,14,195,173,81,197,205,82
1460 DATA197,240,40,169,5,24,109,85,197
1470 DATA141,85,197,144,3,238,86,197
1480 DATA169,5,24,109,91,197,141,91,197
1490 DATA144,3,238,92,197,230,253,208,2
1500 DATA230,254,238,82,197,76,73,198
1510 DATA173,74,197,205,81,197,240,88
1520 DATA169,5,24,109,83,197,141,89,197
1530 DATA144,3,238,84,197,169,5,24,109
1540 DATA89,197,141,89,197,144,3,238,90
1550 DATA197,169,1,24,109,74,197,10,56
1560 DATA237,81,197,24,105,11,24,101,253
1570 DATA133,253,165,254,105,0,133,254
1580 DATA173,68,197,141,85,197,173,69
1590 DATA197,141,86,197,173,72,197,141
1600 DATA91,197,173,73,197,141,92,197
1610 DATA169,1,141,82,197,238,81,197,76
1620 DATA73,198,104,168,104,170,104,96
1630 DATA173,68,197,141,83,197,141,85
1640 DATA197,173,69,197,141,84,197,141
1650 DATA86,197,173,70,197,141,87,197
1660 DATA173,71,197,141,88,197,173,74
1670 DATA197,141,93,197,173,72,197,141
1680 DATA89,197,141,91,197,173,73,197
1690 DATA141,90,197,141,92,197,160,159
1700 DATA32,162,179,162,94,160,197,32
1710 DATA212,187,160,99,32,162,179,162
1720 DATA99,160,197,32,212,187,96
1730 DATA79160:REM*CHECKSUM*
1740 FORI=50536TO51123
1750 READX:POKEI,X:CC=CC+X
1760 NEXT
1770 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1780 PRINT"II-ROT.HEX INSTALLED OK"
```

Assembler-Listing

```
1010 ;++++++++++++++++++++++++++
1020 ;++      ROTSUB 64         ++
1030 ;++                        ++
1040 ;++++++++++++++++++++++++++
1140 ;
1150 *       = $C544
1190 ; VARIABLES CALLED FROM BASIC
1200 ;
1210 XBASLO  *=*+1  ; POKE50500,X(1)LO
1220 XBASHI  *=*+1  ; POKE50501,X(1)HI
1230 YBASLO  *=*+1  ; POKE50502,Y(1)LO
1240 YBASHI  *=*+1  ; POKE50503,Y(1)HI
1250 ZBASLO  *=*+1  ; POKE50504,Z(1)LO
1260 ZBASHI  *=*+1  ; POKE50505,Z(1)HI
1270 NP      *=*+1  ; POKE50506,NP
1280 CSLO    *=*+1  ; POKE50507,CSLO
1290 CSHI    *=*+1  ; POKE50508,CSHI
1300 SNLO    *=*+1  ; POKE50509,SNLO
1310 SNHI    *=*+1  ; POKE50510,SNHI
1320 EBASLO  *=*+1  ; POKE50511,E%(1,1)LO
1330 EBASHI  *=*+1  ; POKE50512,E%(1,1)HI
1340 ;
1350 ; VARIABLES USED BY M/C
1370 IINDEX  *=*+1
1380 JINDEX  *=*+1
1390 XILO    *=*+1
1400 XIHI    *=*+1
1410 XJLO    *=*+1
1420 XJHI    *=*+1
1430 YILO    *=*+1
1440 YIHI    *=*+1
1450 ZILO    *=*+1
1460 ZIHI    *=*+1
1470 ZJLO    *=*+1
1480 ZJHI    *=*+1
1490 TEMPNP  *=*+1
1500 MEM1    *=*+5 ;FLOATING POINT VAR
1510 MEM2    *=*+5 ;FLOATING POINT VAR
1520 ZPTEMP  =$FD ;FREE ZERO PAGE LOC
1530 ;
1540 ;INTERPRETER ARITHMETIC CALLS
1560 FMULT   =$BA28 ; FAC=FAC*MEM
1570 FSUB    =$B850 ; FAC=MEM-FAC
1580 FADD    =$B867 ; FAC=FAC+MEM
1590 MOVFM   =$BBA2 ; FAC=MEM
1600 MOVMF   =$BBD4 ; MEM=FAC
1610 FLPINT  =$B1AA ; .Y/.A=INT(FAC) N.B. HI/LO ORDER !!
1620 SNGFT   =$B3A2 ; FAC= .Y
1630 ;
1640 ;OTHER MACHINE CODE ROUTINE/VARS
1660 LINSUB  =$C30E
1670 MLO     =$C300
1680 MHI     =$C301
1690 NLO     =$C302
1700 NHI     =$C303
1710 Y1      =$C304
1720 Y2      =$C305
1740 ;++++ SAVE REGISTERS++
1760         PHA
1770         TXA
1780         PHA
1790         TYA
1800         PHA
1820 ;++++ INITIALISE VARIABLES ++++
1840         JSR SETUP
1860 ;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
1880 START
1890         LDA XILO
1900         LDY XIHI
1910         JSR MOVFM   ; FAC = X(I)
1920         LDA CSLO
1930         LDY CSHI
1940         JSR FMULT   ; FAC = X(I)*CS
1950         LDX #<MEM1
1960         LDY #>MEM1
1970         JSR MOVMF   ; MEM1=X(I)*CS
1980         LDA YILO
1990         LDY YIHI
2000         JSR MOVFM   ; FAC = Y(I)
2010         LDA SNLO
2020         LDY SNHI
2030         JSR FMULT   ; FAC = Y(I)*SN
```



```
2040          LDA #<MEM1
2050          LDY #>MEM1
2060          JSR FSUB    ; FAC = MEM1-FAC
2070          LDX #<MEM1
2080          LDY #>MEM1
2090          JSR MOVMF   ; MEM1= FAC
2110 ;++++PERFORM MEM2=Y(I)*CS+X(I)*SN
2130          LDA YILO
2140          LDY YIHI
2150          JSR MOVFM   ; FAC = Y(I)
2160          LDA CSLO
2170          LDY CSHI
2180          JSR FMULT   ; FAC = Y(I)*CS
2190          LDX #<MEM2
2200          LDY #>MEM2
2210          JSR MOVMF   ; MEM2= Y(I)*CS
2220          LDA XILO
2230          LDY XIHI
2240          JSR MOVFM   ; FAC = X(I)
2250          LDA SNLO
2260          LDY SNHI
2270          JSR FMULT   ; FAC = X(I)*SN
2280          LDA #<MEM2
2290          LDY #>MEM2
2300          JSR FADD    ; FAC = MEM2+FAC
2310          LDX #<MEM2
2320          LDY #>MEM2
2330          JSR MOVMF   ; MEM2= FAC
2350 ;++++PERFORM X(I)=MEM1:Y(I)=MEM2
2370          LDA #<MEM1
2380          LDY #>MEM1
2390          JSR MOVFM   ; FAC = MEM1
2400          LDX XILO
2410          LDY XIHI
2420          JSR MOVMF   ; X(I)= FAC
2430          LDA #<MEM2
2440          LDY #>MEM2
2450          JSR MOVFM   ; FAC = MEM2
2460          LDX YILO
2470          LDY YIHI
2480          JSR MOVMF   ; Y(I)=FAC
2500 ;++++TEST END LOOP
2520          DEC TEMPNP
2530          BEQ CONTIN
2550 ;++++INCREMENT ARRAY POINTERS
2570          LDA #$05
2580          CLC
2590          ADC XILO
2600          STA XILO
2610          BCC XNOHI
2620          INC XIHI
2630 XNOHI
2640          LDA #$05
2650          CLC
2660          ADC YILO
2670          STA YILO
2680          BCC YNOHI
2690          INC YIHI
2700 YNOHI
2710          JMP START
2730 ;++++ CLEAR/INIT SCREEN
2750 CONTIN
2760          LDA #$01
2770          STA $C100
2780          STA $C101
2790          STA $C102
2800          JSR $C10E ; INIT HIRES
2820 ;++++ PLOT LINES FROM E%(I,J)
2840 ; INITIALISE VARIABLES
2860          JSR SETUP
2870          LDA #$01
2880          STA IINDEX     ; I=1
2890          STA JINDEX     ; J=1
2900          LDA EBASLO     ; STORE E%
2910          STA ZPTEMP     POINTER
2920          LDA EBASHI     ; ON ZERO
2930          STA ZPTEMP+1   ; PAGE
2950 ; OFF WE GO - START GIANT I,J LOOP
2970 NEXTIJ
2980          LDY JINDEX
2990          LDA (ZPTEMP),Y ;GET E%(I,J)
3000          BNE DOIT
3010          JMP ONWARD
3030 ;++++PERFORM  X1%=X(I)+159
3040 ;    MLO=X1% LO:MHI=X1% HI
3060 DOIT
3070          LDA XILO
3080          LDY XIHI
3090          JSR MOVFM       ; FAC = X(I)
3100          LDA #<MEM1
3110          LDY #>MEM1
3120          JSR FADD        ; FAC = X(I)+159
3130          JSR FLPINT      ; .Y/.A = INT(FAC)
3140          STY MLO         ; X1% LO
3150          STA MHI         ; X1% HI
3170 ;++++PERFORM  X2%=X(J)+159
3180 ;    NLO=X2% LO:NHI=X2% HI
3200          LDA XJLO
3210          LDY XJHI
3220          JSR MOVFM       ; FAC = X(J)
3230          LDA #<MEM1
3240          LDY #>MEM1
3250          JSR FADD        ; FAC = X(J)+159
3260          JSR FLPINT      ; .Y/.A = INT(FAC)
3270          STY NLO         ; X2% LO
3280          STA NHI         ; X2% HI
3300 ;++++PERFORM Y1%=99-Z(I)
3320          LDA ZILO
3330          LDY ZIHI
3340          JSR MOVFM       ; FAC =Z(I)
3350          LDA #<MEM2
3360          LDY #>MEM2
3370          JSR FSUB        ; FAC = 99-Z(I)
3380          JSR FLPINT      ; .Y/.A = INT(FAC)
3390          STY Y1          ; Y1%
3410 ;++++PERFORM Y2%=99-Z(J)
3430          LDA ZJLO
3440          LDY ZJHI
3450          JSR MOVFM       ; FAC =Z(J)
3460          LDA #<MEM2
3470          LDY #>MEM2
3480          JSR FSUB        ; FAC = 99-Z(J)
3490          JSR FLPINT      ; .Y/.A = INT(FAC)
3500          STY Y2          ; Y2%
3520 ; GET READY FOR LINSUB
3540 ;TWO BYTE COMPARISON FOR X1%=X2%
3560          LDA MLO
3570          CMP NLO
3580          BNE NOPE
3590          LDA MHI
3600          CMP NHI
3610          BNE NOPE
3630 ;NOW COMPARE Y1=Y2
3650          LDA Y1
3660          CMP Y2
3670          BNE NOPE
3690 ; ALL EQUAL SO AVOID LINESUB
3710          JMP ONWARD
3720 NOPE
3730          JSR LINSUB     ;PLOT LINE
3750 ; INCREMENT J POINTERS
3770 ONWARD
3780          LDA IINDEX     ;FIRST TEST
3790          CMP JINDEX     ;J=<I ?
3800          BEQ NEXTI
3820 ;INCREMENT XJLO/XJHI
3840          LDA #$05
3850          CLC
3860          ADC XJLO
3870          STA XJLO
3880          BCC XJNOHI
3890          INC XJHI
3900 XJNOHI
3920 ;INCREMENT ZJLO/ZJHI
3940          LDA #$05
3950          CLC
3960          ADC ZJLO
3970          STA ZJLO
3980          BCC ZJNOHI
3990          INC ZJHI
4000 ZJNOHI
4020 ;INCREMENT E% POINTER BY 1
4040          INC ZPTEMP
4050          BNE NOHIBY
4060          INC ZPTEMP+1
4070 NOHIBY
4080          INC JINDEX
4090          JMP NEXTIJ
4110 ; INCREMENT I POINTERS
4130 NEXTI
4140          LDA NP         ;FIRST TEST
4150          CMP IINDEX     ;I=<NP ?
4160          BEQ EXIT
4180 ; INCREMENT XILO/XIHI
4200          LDA #$05
4210          CLC
4220          ADC XILO
4230          STA XILO
4240          BCC XINOHI
4250          INC XIHI
4260 XINOHI
4270 ; INCREMENT ZILO/ZIHI
4300          LDA #$05
4310          CLC
4320          ADC ZILO
4330          STA ZILO
4340          BCC ZINOHI
4350          INC ZIHI
4360 ZINOHI
4380 ; INCREMENT E% POINTERS (TRICKY)
4400          LDA #$01       ; WHEN INC I
4410          CLC            ; MUST ADD
4420          ADC NP         ; (NP+1)-(I-1)
4430          ASL A          ; TO THE ZP
4440          SEC            ; POINTER TO
4450          SBC IINDEX     ; E%
4460          CLC
4470          ADC #$01
4480          CLC
4490          ADC ZPTEMP
4500          STA ZPTEMP
4510          LDA ZPTEMP+1
```

```
4520          ADC #$00
4530          STA ZPTEMP+1
4550 ; REINITIALISE XJLO/XJHI
4570          LDA XBASLO
4580          STA XJLO
4590          LDA XBASHI
4600          STA XJHI
4620 ; REINITIALISE ZJLO/ZJHI
4640          LDA ZBASLO
4650          STA ZJLO
4660          LDA ZBASHI
4670          STA ZJHI
4690 ; REINITIALISE JINDEX
4710          LDA #$01
4720          STA JINDEX
4740 ; INCREMENT IINDEX
4760          INC IINDEX
4770          JMP NEXTIJ
4790 ; GIANT LOOP ENDS HERE
4800 ;
4810 ;++++ PULL REGISTERS ++++
4820 ;
4830 EXIT
4840          PLA
4850          TAY
4860          PLA
4870          TAX
4880          PLA
4890          RTS
4930 ;++++ SUBROUTINE
4940 ;
4950 SETUP
4960          LDA XBASLO
4970          STA XILO
4980          STA XJLO
4990          LDA XBASHI
5000          STA XIHI
5010          STA XJHI
5020          LDA YBASLO
5030          STA YILO
5040          LDA YBASHI
5050          STA YIHI
5060          LDA NP
5070          STA TEMPNP
5080          LDA ZBASLO
5090          STA ZILO
5100          STA ZJLO
5110          LDA ZBASHI
5120          STA ZIHI
5130          STA ZJHI
5140          LDY #159
5150          JSR SNGFT     ; FAC = 159
5160          LDX #<MEM1
5170          LDY #>MEM1
5180          JSR MOVMF     ; MEM1=159 IN FPT
5190          LDY #99
5200          JSR SNGFT     ; FAC = 99
5210          LDX #<MEM2
5220          LDY #>MEM2
5230          JSR MOVMF     ; MEM2=99 IN FTP
5240          RTS
```

Testprogramm in BASIC

```
1000 REM**PLOT ROTATING CUBES**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"II-ROT.HEX",8,1
1040 PRINTCHR$(147)"WAIT 17 SECS"
1050 REM**DIMENSION ARRAYS**
1060 NP=24:REM NUMBER OF POINTS
1070 DIM X(NP),Y(NP),Z(NP)
1080 DIM E%(NP,NP):REM EDGE CONNECTIONS
1090 REM**INITIALISE ARRAYS**
1100 REM--INITIAL CUBE COORDINATE DATA
1110 DATA  50, 50, 50:REM----------/1
1120 DATA -50, 50, 50:REM TOP FOUR /2
1130 DATA -50,-50, 50:REM POINTS   /3
1140 DATA  50,-50, 50:REM----------/4
1150 DATA  50, 50,-50:REM----------/5
1160 DATA -50, 50,-50:REM BOT FOUR /6
1170 DATA -50,-50,-50:REM POINTS   /7
1180 DATA  50,-50,-50:REM----------/8
```

Der folgende Programmteil liest die zuvor definierten Daten in die Arrays X(), Y() und Z(). Beachten Sie, daß die Anfangswerte der kleinen Würfel um ein Drittel verringert wurden.

```
1190 REM**READ IN DATA FOR 3-CUBES**
1200 FORI=1TO8:REM CENTRE BIG CUBE
1210 READX(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FORI=9TO16:REM RIGHT SMALL CUBE
1250 READX,Y,Z
1260 X(I)=.3*X+120:Y(I)=.3*Y:Z(I)=.3*Z
1270 NEXT
1280 RESTORE
1290 FORI=17TO24:REM LEFT SMALL CUBE
1300 READX,Y,Z
1310 X(I)=.3*X-120:Y(I)=.3*Y:Z(I)=.3*Z
1320 NEXT
```

Der nächste Abschnitt rotiert die Koordinaten der Punkte, so daß die drei Würfel unter einem Winkel von 45 Grad erscheinen.

```
1330 REM**ROTATE SPACE ABOUT X-AXIS π/4
1340 FORI=1TONP
1350 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1360 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1370 NEXT
1380 REM**ROTATE SPACE ABOUT Z-AXIS π/4
1390 FORI=1TONP
1400 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1410 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1420 NEXT
```

Der folgende Programmteil legt über das Array E%() die Punkte der drei Würfel fest, die miteinander verbunden werden.

```
1430 REM**EDGE CONNECTION DATA**
1440 REM-MIDDLE BIG CUBE-
1450 E%(1,2)=1:REM 1 CONNECTED TO 2
1460 E%(2,3)=1:E%(3,4)=1:E%(4,1)=1
1470 E%(5,6)=1:REM BOT SQUARE
1480 E%(6,7)=1:E%(7,8)=1:E%(8,5)=1
1490 E%(5,1)=1:REM TOP TO BOT EDGES
1500 E%(6,2)=1:E%(7,3)=1:E%(8,4)=1
1510 REM-----------------
1520 REM-RIGHT SMALL CUBE
1530 FORI=9TO16:FORJ=9TO16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT:NEXT
1560 REM-LEFT SMALL CUBE
1570 FORI=17TO24:FORJ=17TO24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT:NEXT
1600 REM**SYMMETRISE E%(I,J)**
1610 FORI=1TONP:FORJ=1TONP
1620 IFE%(I,J)<>0THENE%(J,I)=1
1630 NEXT:NEXT
```

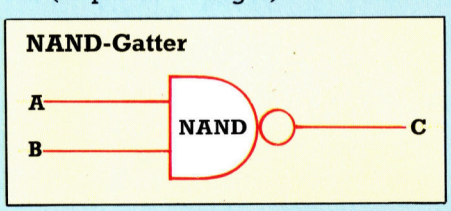
Das Programm ruft den Maschinencode auf, der die Rotationswerte berechnet und neue Figuren zeichnet. Die Maschinenroutine wird im Inneren der Schleife mehrfach aufgerufen, so daß sich die Figuren um 360 Grad drehen.

```
1640 REM##########################
1650 REM##PLOT ROTATING CUBE##
1660 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1670 GOSUB1790:REM INITIALISE
1680 FOR A=0 TO 2*π STEP SA
1690 SYS50536:REM ROTATE
1700 NEXTA:REM NEXT ANGLE
1710 GETA$:IFA$=""THEN1710
1720 REM##########################
1730 GOSUB1750:REM RESET SCREEN
1740 END
1750 REM**RESET SCREEN**
1760 POKE49408,0:SYS49422
1770 PRINTCHR$(147)
1780 RETURN
1790 REM**INITIALISE ROTSUB**
1800 REM**************************
1810 REM**N.B. DO NOT DEFINE NEW**
1820 REM**VARIABLES BETWEEN THIS**
1830 REM** SUBROUTINE & CALLING **
1840 REM**  MACHINE CODE ROTATE.**
1850 REM** AS THIS WOULD CHANGE **
1860 REM** BASE ADDRESS ARRAYS  **
1870 REM**************************
1880 X(1)=X(1):REM X(1) CURRENT VAR
1890 POKE50500,PEEK(71):REM X(1)LO
1900 POKE50501,PEEK(72):REM X(1)HI
1910 Y(1)=Y(1):REM Y(1) CURRENT VAR
1920 POKE50502,PEEK(71):REM Y(1)LO
1930 POKE50503,PEEK(72):REM Y(1)HI
1940 Z(1)=Z(1):REM Z(1) CURRENT VAR
1950 POKE50504,PEEK(71)
1960 POKE50505,PEEK(72)
1970 POKE50506,NP:REM NUMBER OF POINTS
1980 CS=CS:REM MAKE CS CURRENT VAR
1990 POKE50507,PEEK(71)
2000 POKE50508,PEEK(72)
2010 SN=SN:REM MAKE SN CURRENT VAR
2020 POKE50509,PEEK(71)
2030 POKE50510,PEEK(72)
2040 E%(1,1)=E%(1,1):REM E% CURRENT VAR
2050 POKE50511,PEEK(71)
2060 POKE50512,PEEK(72)
2070 RETURN
```

Fachwörter von A bis Z

Name = Name
Namen dienen im Computerbereich zum Identifizieren unterschiedlicher Daten. Namen können in Programmier- und Systemsprachen für Dateien, Prozeduren, Variable, Tabellenfelder und vieles mehr verwendet werden, und zwar nicht nur als Gedächtnishilfe: Der Programmierer braucht sich keine weiteren Gedanken um den Speicherort einer Variablen oder Datei zu machen. Das System legt automatisch Listen mit der Zuordnung von Namen und Adressen an. Wenn der Rechner aus Platzgründen Dateien im RAM oder auf Diskette hin- und herschiebt, ist der Zugriff über die Dateinamen trotzdem stets gesichert, ohne daß dem Programmierer die Verlagerung überhaupt bewußt wird.

NAND = NAND
NAND ist die Abkürzung von NOT AND und bewirkt als Boolesche Verknüpfung eine UNDung mit Negation des Ergebnisses. Die NAND-Funktion liefert daher umgekehrt wie das AND das Resultat FALSE, wenn alle Eingangsgrößen gleichzeitig den Wert TRUE haben. Elektronisch wird die Verknüpfung durch ein NAND-Gatter realisiert, dessen Ausgang genau dann auf 1 liegt, wenn alle Eingänge den Wert 0 führen (in positiver Logik).



Das Schaltzeichen für den NAND-Gatter entsteht aus dem AND-Symbol, indem ausgangsseitig zur Kennzeichnung der Negation ein kleiner Kreis angefügt wird.

Negative Logic = Negative Logik
Wie zu erwarten, stehen bei negativer Logik alle Konventionen auf dem Kopf: Die binäre 1 bedeutet FALSE und die 0 bedeutet TRUE. Die Einführung hat rein technische Gründe – viele Schaltungen lassen sich dadurch einfacher gestalten. Negative Logik ist z. B. bei Microprozessoren für das Schreib/Lese-Signal verbreitet, bei dem die 0 Freigabe und die

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.

1 Sperren bewirkt. Der Begriff wird auch verwendet, wenn bei Booleschen Operationen durch Komplementbildung die ursprüngliche Funktion ins genaue Gegenteil verkehrt wird.

Nesting = Verschachtelung
Die Verschachtelung ist ein wichtiges Prinzip bei der Programmierung von Mehrfach-Schleifen. In BASIC muß jede innere FOR-Schleife vollständig in der äußeren Schleife eingebettet sein. Eine Verletzung dieser Grundregel löst bestenfalls eine Fehlermeldung aus, meist aber einen Programmabsturz. Wenn der Rechner auf das Schlüsselwort FOR trifft, schiebt er die zugehörige Adresse auf den Stack. Stößt er dann auf die folgende NEXT-Anweisung, holt er sich die FOR-Adresse wieder und verzweigt zum Schleifenanfang. Wenn Schleifen nicht korrekt geschachtelt sind, lädt der Computer die falsche Rücksprungadresse vom Stack und setzt die Programmbearbeitung an einer falschen Stelle fort.

Die Schachtelung wird vor allem akut, wenn wiederholt gleichartige Operationen auszuführen sind, wie etwa in dem folgenden Programm:

```
10 FOR A=1 TO 12
20 FOR B=1 TO 12
30 PRINT A;"x";B;"="; A*B
40 NEXT B
50 NEXT A
```

Die Ausführung beginnt mit der A-Schleife, indem die Adresse der ersten FOR-Anweisung auf den Stack gelegt und irgendwo im Speicher die Variable A auf 1 gesetzt wird. Das gleiche Arbeitsprinzip gilt für die B-Schleife. Da der Stack nach dem FIFO-Prinzip organisiert ist, lädt der Rechner dann in Zeile 40 gemäß der NEXT-Anweisung als oberste Stack-Adresse wieder die des FOR von Zeile 20 und setzt das Programm dort mit der Erhöhung der Variablen B fort. Nach zwölfmaligem Durchlaufen der B-Schleife und anschließendem Rücksprung auf Zeile 20 erreicht B den Wert 13, das heißt, der Schleifenendwert ist überschritten. Daraufhin geht der Rechner auf Zeile 50 über, wo er wieder ein NEXT findet, sich daraufhin vom Stack die Einsprungsadresse der A-Schleife greift usw.

Verschachtelte Schleifen

```
5 REM Korrekte Schachtelung
10 FOR X=0 TO 6
20 FOR Y=0 TO X
30 PRINT X;":";"Y";"";
40 NEXT Y
50 PRINT
60 NEXT X

5 REM Fehlerhafte Schachtelung
10 FOR X=0 TO 6
20 FOR Y=0 TO X
30 PRINT X;":";Y;"";
40 NEXT X
50 PRINT
60 NEXT Y
```

Bei korrekter Schachtelung muß die innere Schleife ganz in die äußere eingebettet sein. Beim oberen Programm liegt die Y-Schleife vollständig innerhalb der X-Schleife, und es gibt keine Probleme. Im unteren Programm ragt dagegen die eine Schleife einseitig über die andere hinaus, und daher sind bei der Ausführung Fehler zu erwarten.

Viele BASIC-Interpreter verlangen den Namen der Laufvariablen in der NEXT-Anweisung nicht. Sie springen ohne Überprüfung auf die FOR-Anweisung zurück, die gerade im Stack ansteht. Eine Fehlschachtelung wäre geradezu fatal.

Bildnachweise
1905, 1907–1910, 1918,
1923–1925: Kevin Jones
1906, 1914: Caroline Clayton
1917, 1919: Chris Stevens
1920, 1921, 1928: Ian McKinnell
U3: Liz Dixon

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

computer kurs Heft 70

Wer selbst Programme schreibt und sie auch mal anderen zugänglich macht, möchte sein geistiges Eigentum sichern – mit einem Kopierschutz. In unserem nächsten PROGRAMMIER-SERVICE zeigen wir, wie man seine Software sichern kann.

Stein auf Stein

Es geht weiter mit dem Go-Spiel. Beschrieben wird das Modul, das zum Setzen der Steine notwendig ist. Außerdem geht es um die Aktualisierung der Spielbrett-Darstellung.

Starkes Echo

Für den Commodore 64 und den Acorn B gibt es das Musikpaket „Echo“, das den Soundchip im Rechner über eine Klaviertastatur anspricht. Das System ist leicht einzusetzen.