Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Heft 68

computer kurs

Heft 68

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

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

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

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

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

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

Abonnement

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

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

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

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

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

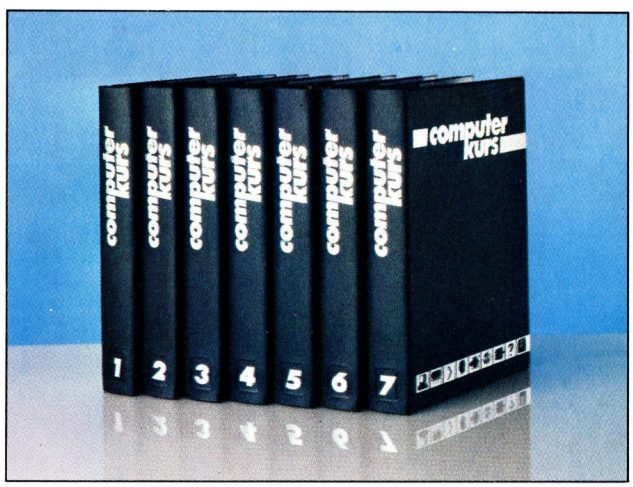
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

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

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

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

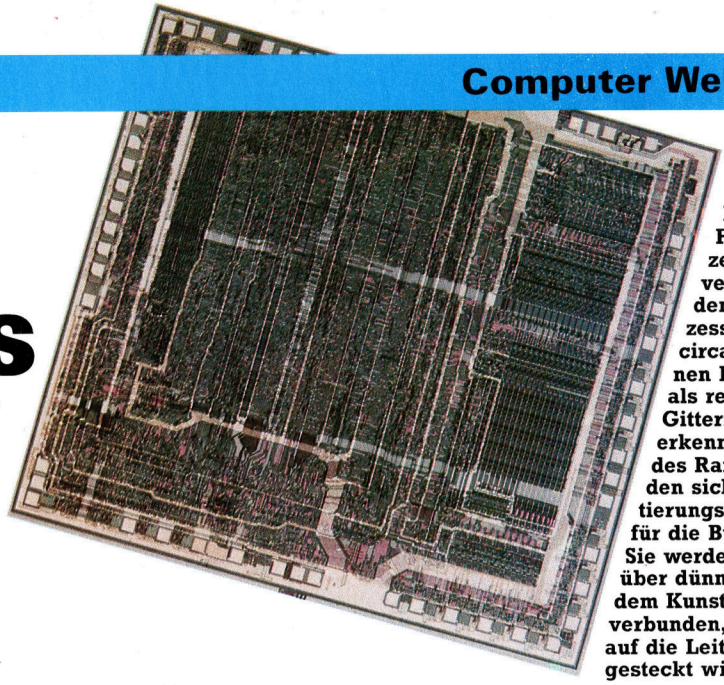


© APSIF, Copenhagen, 1982, 1983; © Orbis Publishing Ltd., 1982, 1983; © Marshall Cavendish Ltd., 1984, 1985, 1986;

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

Chip-Tourismus

Das komplizierte Zusammenwirken der verschiedenen Komponenten eines Microcomputers ist nicht ganz leicht zu verstehen. Eine Platinen-„Rundreise" soll die Funktion der einzelnen Bauteile und ihr Wechselspiel erklären.



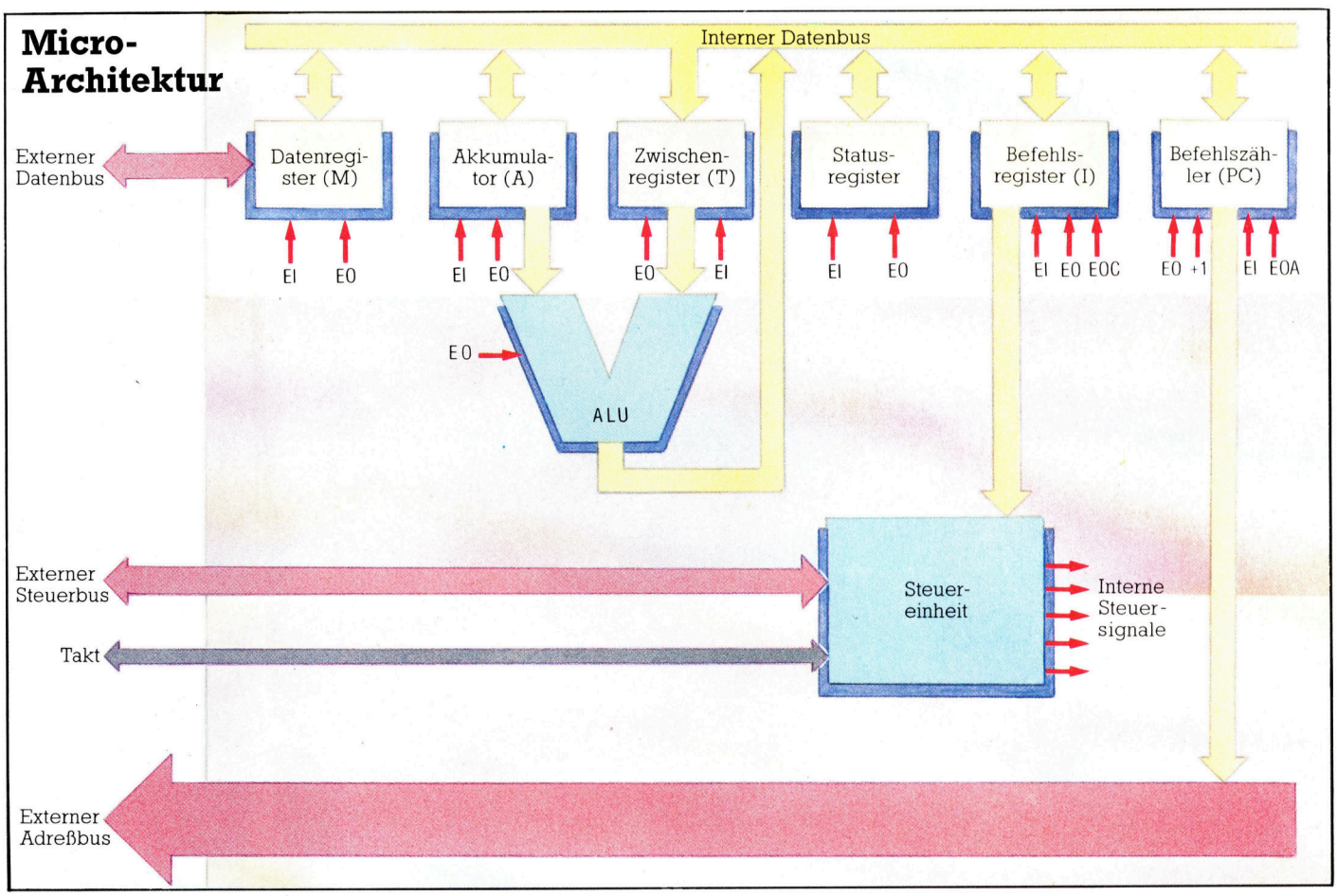
Diese Fotografie zeigt stark vergrößert den Zilog-Prozessor Z80. Die circa 25 internen Register sind als rechtwinklige Gittermuster zu erkennen. Längs des Randes befinden sich 40 Kontaktierungsflächen für die Busanschlüsse. Sie werden später über dünne Drähte mit dem Kunststoffgehäuse verbunden, das dann auf die Leiterplatte gesteckt wird.

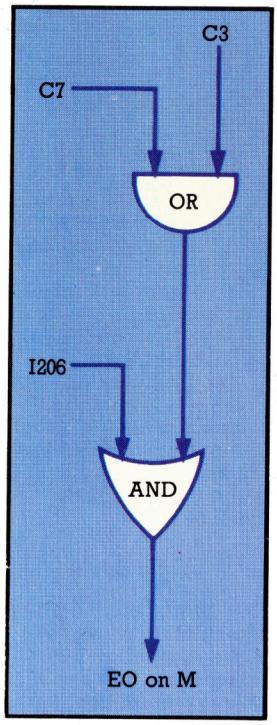
Unter dem Mikroskop sind auf einem Microprozessorchip typische Strukturen zu erkennen: Die internen Register erscheinen als charakteristische Gitterfelder, die Logikgatter als unregelmäßig gemusterte Bereiche, und die quadratischen Metallplättchen am Rande sind für die Kontaktierung, also die Verbindung mit der Außenwelt zuständig. Außerdem fallen Gruppen paralleler Leitungen auf, die der internen Kommunikation zwischen den Funktionselementen dienen.

Jede CPU-Anweisung setzt eine ganze Kette von elektronischen Elementarschritten in Gang, die jenseits der Software-Zuständigkeit ablaufen. Zur Illustration wollen wir beschreiben, wie auf einen ADD-Befehl hin die Addition einer Zahl zum Akkumulatorinhalt mit Rückschreiben der Summe abläuft.

Das untenstehende Diagramm zeigt in vereinfachter Form die typische CPU-Architektur. Alle Register kommunizieren über einen internen Datenbus und hängen außerdem zur Synchronisation des Datenverkehrs über Steuerleitungen an einer zentralen Steuerlogik.

Die Abbildung veranschaulicht den Aufbau eines typischen 8-Bit-Prozessors. Die Funktionsgruppen kommunizieren miteinander über interne Datenbus- und Steuerleitungen. Über Steuersignale und den externen Daten- und Adreßbus hat die CPU Verbindung mit dem Arbeitsspeicher und der Peripherie.





Für den ADD-Befehl muß in den Schritten 3 und 7 des Befehlszyklus das Steuersignal „Enable Output" (EO) an das Datenregister (M) gesendet werden. Diese Gatter verknüpfen die Leitungen C2 und C3 des Takt-Decoders so mit dem Befehlsdekoder-Ausgang I206, daß zum richtigen Zeitpunkt das EO-Signal am M-Register ansteht.

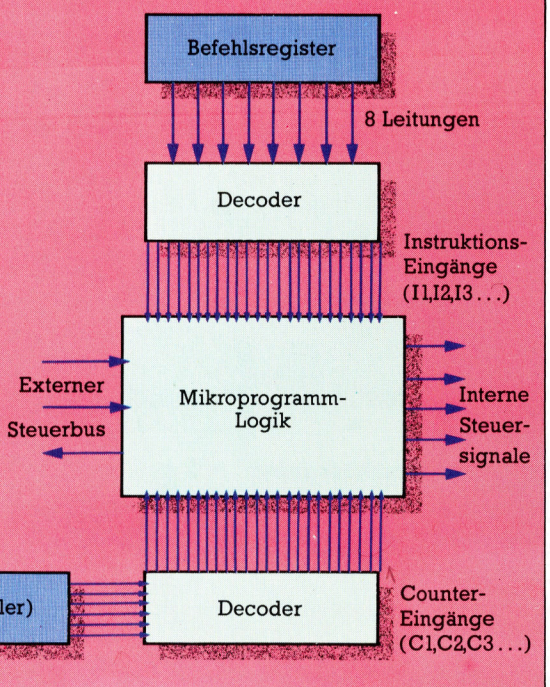
Der erwähnte ADD-Befehl steht vorerst in mehreren aufeinanderfolgenden Bytes innerhalb eines längeren Programms im Speicher des Rechners. Das erste Byte enthält dabei den Operationsteil, das zweite die zu addierende Zahl. Der Prozessor ruft nun zunächst das Byte mit dem Opcode aus dem Arbeitsspeicher ins Befehlsregister ab („Fetch"-Zyklus) und führt dann die Anweisung aus („Execute"-Zyklus). Die Adresse des Befehlsbytes steht schon vorher im Befehlszähler (Programm Counter = PC) bereit. Beim Fetch-Execute-Zyklus ist von besonderem Interesse, wie die Steuereinheit den Datenaustausch zwischen den Registern organisiert – das jeweilige Register muß im richtigen Moment ein Freigabe(Enable)-Signal bekommen. Für diese Signale sind folgende Abkürzungen zweckmäßig:

EI (= Enable Input) gibt ein Register für das Beschreiben vom internen Datenbus aus frei,
EO (= Enable Output) veranlaßt das Auslesen eines Registers auf den Datenbus,
EOC (= EO to Control Unit) bewirkt die Ausgabe des Registerinhalts an die Steuerlogik,
EOA (= EO to Address Bus) das Auslesen des Registers auf den Adreßbus.
Der Fetch-Zyklus umfaßt vier Schritte:

Schritt	Steuersignal	Wirkung
1	EOA an PC	Kopieren des Befehlszählers(PC) auf den Adreßbus
2	+1 an PC	Erhöhen des Befehlszählers
3	EO an M	Kopieren des Datenregisters (M)
	EI an I	ins Befehlsregister(I)
4	EOC an I	Kopieren des Befehlsregisters in die Steuerlogik

Zu Befehl!

Die Steuereinheit verfügt über ein Zählregister, das die Impulse des externen Taktgebers auflaufen läßt, und über je einen Decoder für die Entschlüsselung der Bitmuster im Taktzähler und im Befehlsregister. In der Microprogramm-Logik sind für jeden Befehl zur Ausführung in Elementarschritten individuelle Gatterschaltungen vorgesehen, die während des Fetch-Execute-Zyklus in der richtigen Reihenfolge die nötigen Steuersignale bereitstellen.



Mit dem vierten Schritt ist endlich der ADD-Befehl in die Steuerlogik gebracht, und der Befehlszähler enthält die Adresse des Bytes mit der Konstanten, die addiert werden soll. Vor der Erklärung des Execute-Zyklus ist noch ein Blick auf die Steuereinheit angebracht.

Geordnetes Hin und Her

Sie enthält ein Register für die Aufnahme des aktuellen Opcodes und einen Decoder, der entsprechend dem 8-Bit-Muster des Befehls eine von 256 Ausgangsleitungen aktiviert – wenn nicht alle möglichen Opcodes genutzt werden, sind es weniger als 256 Leitungen. Darüber werden nun in der „Microprogrammlogik" individuelle Gatterschaltungen angewählt, durch die im Zusammenspiel mit einem Taktzähler-Register (Counter) und einem weiteren Decorder die verschiedenen Steuersignale bereitgestellt werden. Dabei läuft hardwaremäßig ein „Microprogramm" ab; in unserem Beispiel liefern die Taktschritte während der fünf Taktschritte des Execute-Zyklus folgende Signale:

Schritt	Steuersignal	Wirkung
5	EOA an PC	Einlesen der Additions-
	EI an M	konstanten in M
6	+1 an PC	PC für das nächste Befehlsbyte inkrementieren
7	EO an M	Konstanteneingabe in das
	EI an T	ALU-Zwischenregister (T)
8	ADD an ALU	Addieren der Konstanten zum
	EO an T u. A	Akku-Inhalt in der ALU
9	EO an ALU	Ergebnisübernahme in den
	EI an A	Akkumulator

Während der Fetch-Zyklus für alle Befehle gleich bleibt, muß der Execute-Zyklus genau auf die jeweilige Anweisung zugeschnitten sein – hier auf das Laden der Konstanten und ihre Addition zum Akku-Inhalt. Deshalb erfordert jeder Opcode ein individuelles Microprogramm, das heißt eine eigene Gatterschaltung. Wie damit die Steuersignale erzeugt werden, läßt sich am besten an einem konkreten Beispiel erklären: Das Ausgabesignal für das Datenregister (EO an M) muß bei Schritt 3 und 7 des Befehlszyklus aktiviert werden. Der Opcode für die Konstantenaddition ist 1100, 1110, dezimal also 206. Sie brauchen nun nur die Leitungen C3 und C7 vom Taktregister-Dekoder durch ein OR-Gatter zu verknüpfen und das Ergebnis mit der Leitung I206 vom Befehlsdekoder zu ANDen – schon erhält man die gewünschten EO-Signale. Das M-Register muß natürlich auch noch durch andere Microprogramme ansprechbar sein, aber das läßt sich einfach durch OR-Verknüpfungen sämtlicher EO-Signale erreichen.

Näheres über diese „Microprogrammierung" werden wir Ihnen berichten.

Auf die Plätze . . .

Nachdem die allgemeinen Regeln und Listings für die E/A-Routinen von Go ausgearbeitet sind, müssen nun noch einige kleinere Routinen integriert werden, die die Datenstruktur organisieren.

Für ein Go-Programm werden einige allgemeine Routinen benötigt. Dazu zählen Prozeduren zur Aktualisierung der Spielbrettdarstellung, eine Zugkontrolle usw. Anstelle vieler spezieller Routinen implementieren wir hierfür eine allgemeine Routine, die vielfältig eingesetzt werden kann. Anschließend werden die Bedeutungen der Ergebnisse bewertet.

Die Haupt-Prozedur für diese Aufgabe heißt in der Acorn-B-Version PROCsearch. Ihr werden zwei Parameter übergeben: die Position jedes Steines in einer Gruppe, die wir bewerten wollen (P%), und die Farbe der Gruppe (C%). Die Prozedur addiert dann 1 zu CSTN% (Steinanzahl einer Gruppe) und untersucht die vier angrenzenden Steine. Für jeden dieser Steine erfolgt derselbe Vorgang, vorausgesetzt er wurde noch nicht gezählt. Als Struktur der Such-Routine ergibt sich:

```
SEARCH (von-Position, für-Farbe)
  IF von-Position ist auf dem Brett
    AND von-Position enthält einen Stein von
        für-Farbe
    AND wir haben von-Position noch nicht
        gezählt
  THEN
      markiere von-Position, um erneutes
      Zählen zu verhindern; addiere 1 zu
      CSTN% (unsere Zählvariable)
    SEARCH (nördlich von von-Position,
            für-Farbe)
    SEARCH (östlich von von-Position,
            für-Farbe)
    SEARCH (südlich von von-Position,
            für-Farbe)
    SEARCH (westlich von von-Position,
            für-Farbe)
  ENDIF
END SEARCH
```

Beachten Sie, daß die rekursiven Aufrufe mit vier separaten Anweisungen gehandhabt werden. Durch dieses Verfahren lassen sich Probleme bei verschachtelten Rekursionen sehr leicht vermeiden.

Wenn die Routine sich selbst aufruft, wird der „von-Position"-Parameter durch die neue von-Position ersetzt (in einer der vier Richtungen), und die Routine läuft weiter. Die neue von-Position kann die Such-Routine auch rekursiv aufrufen, wobei wiederum eine neue von-Position-Variable angelegt wird. Irgendwann wird eine der IF-Konditionen FALSCH sein, so daß dieser bestimmte Aufruf abgebrochen wird, ohne über die THEN-Anweisungen einen rekursiven Aufruf auszuführen. Immer wenn die Rekursion an einen solchen Punkt kommt, wird der vorherige Wert der von-Position wiederhergestellt.

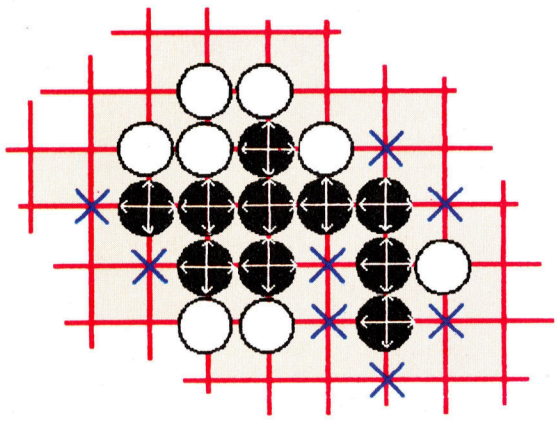
Diese „Stack"-Datenstruktur kann man sich wie einen Plattenstapel vorstellen. Wenn die Routine rekursiv aufgerufen wird, wird eine neue Platte auf den Stapel gelegt. Beim Abbruch dieses Arbeitsvorgangs muß dieselbe Platte jedoch wieder vom Stapel genommen werden (und der gleiche Variablenwert wird wiederhergestellt). Diese Vorgehensweise ist bekannt als LIFO (Last In First Out)-Struktur.

Simulierte Prozeduren

Im Gegensatz zum Acorn-B-BASIC, das Rekursion durch PROCedure-Parameterübergabe und lokale Variablen unterstützt, verfügen der C64, Spectrum und Schneider CPC 464/664 nicht über diese Möglichkeiten. Entsprechend finden Sie in den jeweiligen Routinen die Array-Bezeichnungen SK%() oder s(), in denen die benötigten Werte zur weiteren Verwendung abgelegt werden.

Dies geschieht durch einen Zeiger (beispielsweise mit der Bezeichnung STACK%), der die oberste Position auf dem Stack markiert. Jedesmal, wenn etwas auf dem Stack abgelegt wird, erhöht sich der Zeiger. Die jeweilige Variable wird dann auf den obersten Wert des Stack gesetzt.

Ein Nebeneffekt der Suchroutine ist, daß die Rekursion immer am Rand der bearbeiteten Gruppe endet. Wenn diese Ränder erreicht



Steine bewerten

Die Such-Routine, die zur Ermittlung des Status von Gruppen auf dem Spielbrett verwendet wird, enthält rekursive Prozeduren und sucht von jedem Stein in der Gruppe in alle vier Richtungen. Die Routine bricht ab, wenn ein Freifeld oder ein gegnerischer Stein erreicht wird.

werden, kann man überprüfen, ob es sich um die Ecke des Spielbrettes oder um einen gegnerischen Stein handelt. Ist keine dieser Bedingungen wahr, muß es sich um ein Freifeld handeln. Konsequenterweise zählt der zweite Teil der Routine die mit der Gruppe verbundenen Freifelder. Auch diese Positionen müssen markiert werden, da sie sonst zweimal gezählt würden.

Zum Zählen einer Gruppe wird immer die Routine PROCcount aufgerufen, die wiederum PROCsearch aktiviert. Hierbei werden die Zählvariablen initialisiert und die Markierungen nach Ablauf der Such-Routine gelöscht.

Dazu wird die Routine PROCclear verwendet, die mittels des logischen Operators AND die entsprechenden Bytes mit dem Wert 3 verknüpft. Da die Farben in den beiden niederwertigen Bits jedes Brett-Bytes, die Stein- und Freifeld-Markierungen aber im dritten und vierten Bit gespeichert werden, ist der Aufruf von PROCclear(3) ideal:

Byte:	A	B	C	D	E	F	G	H
Maske:	0	0	0	0	0	0	1	1
Ergebnis:	0	0	0	0	0	0	G	H

Die Spectrum Version sieht etwas anders aus, da der AND-Befehl im Spectrum-BASIC nur für WAHR/FALSCH-Ergebnisse in Ausdrücken wie „IF X<3 AND Y=3 THEN . . .“ verwendet werden kann. Daher müssen wir eine normale arithmetische Subtraktion durchführen, um Bit 2 (der Spectrum Version) dieser Routine zu löschen.

Allroundroutine

Nachdem diese allgemeine Löschroutine definiert ist, kann sie auch zum Löschen des Bretts vor Spielbeginn verwendet werden, indem sie mit einem 0-Parameter aufgerufen wird (siehe Zeile 1400).

Jetzt ist nur noch eine Spiel-Prozedur nötig, bevor wir die Routinen zur Ausführung von Spielzügen schreiben können. Dabei handelt es sich um den Programmteil, der bei Programmstart das Spielbrett mit den entsprechenden Handicap-Steinen erstellt. Bei Go darf der schwächere Spieler (der immer Schwarz hat) beim ersten Zug zwischen zwei und neun Steinen auf das Brett setzen. Diese Steine gehören auf spezielle Positionen (siehe Handicap-Routine Zeilen 600 bis 750). Wenn Sie den bereits bekannten ersten Programmteil einmal gestartet haben, wird Ihnen aufgefallen sein, daß Sie am Ende der „title screen“-Routine zur Eingabe einer Handicap-Zahl aufgefordert werden. Durch Hinzufügen von Zeile 1580 und der Routine von Zeile 1630 bis 1690 werden die Handicap-Steine nun korrekt gesetzt. In den DATA-Anweisungen zwischen den Zeilen 670 und 740 werden die richtigen Brett-Positionen angegeben, wobei jede Anweisung einer bestimmten Anzahl von Handicap-Steinen entspricht.

Zweites Modul

Acorn B

```
  270 PROCread_handicaps
  590 :
  600 DEF PROCread_handicaps
  610 LOCAL L%
  620 DIM hncp% 43
  630 RESTORE 670
  640 FOR L% = 0 TO 43
  650   READ hncp%?L%
  660   NEXT
  670 DATA &44,&CC
  680 DATA &44,&CC,&4C
  690 DATA &44,&CC,&4C,&C4
  700 DATA &44,&CC,&4C,&C4,&88
  710 DATA &44,&CC,&4C,&C4,&84,&8C
  720 DATA &44,&CC,&4C,&C4,&84,&8C,&88
  730 DATA &44,&CC,&4C,&C4,&84,&8C,&48,&
C8
  740 DATA &44,&CC,&4C,&C4,&84,&8C,&48,&
C8,&88
  750 ENDPROC
  760 :
  770 REM*******************************
 1400 PROCclear(0)
 1580 PROChandicap(hand%)
 1620 :
 1630 DEF PROChandicap(hand%)
 1640 LOCAL L%,P%,Q%
 1650 Q%=INT((hand%-2)/2*(hand%+1)+0.5)
 1660 FOR L%=Q% TO Q%+hand%-1
 1670   P%=hncp%?L% : board%?P%=black%
 1680   NEXT
 1690 ENDPROC
 1700 :
 1710 REM*******************************
 4030 :
 4040 DEF PROCcount(P%,C%)
 4050 clib%=0 : cstn%=0
 4070 PROCsearch(P%,C%)
 4080 PROCclear(colour%)
 4090 ENDPROC
 4100 :
 4110 REM*******************************
 4120 :
 4130 DEF PROCsearch(P%,C%)
 4140 IF (P% AND 240)=0 OR (P% AND 15)=0
 THEN ENDPROC
 4150 IF (board%?P% AND colour%)=0 THEN
4250
 4160 IF (board%?P% AND C%)=0 THEN ENDPR
OC
 4170 IF (board%?P% AND marker%)>0 THEN
ENDPROC
 4180 board%?P%=C%+marker%
 4190 cstn%=cstn%+1
 4200 PROCsearch(P%+dir%(1),C%)
 4210 PROCsearch(P%+dir%(2),C%)
 4220 PROCsearch(P%+dir%(3),C%)
 4230 PROCsearch(P%+dir%(4),C%)
 4240 ENDPROC
 4250 IF (board%?P% AND liberty%)>0 THEN
ENDPROC
 4260 board%?P%=liberty%
 4270 clib%=clib%+1
 4290 ENDPROC
 4300 :
 4310 REM*******************************
 4320 :
 4330 DEF PROCclear(M%)
 4340 LOCAL L%
 4350 FOR L%=0 TO 255
 4360   board%?L%=board%?L% AND M%
 4370   NEXT
 4380 ENDPROC
 4390 :
 4400 REM*******************************
```

Commodore 64

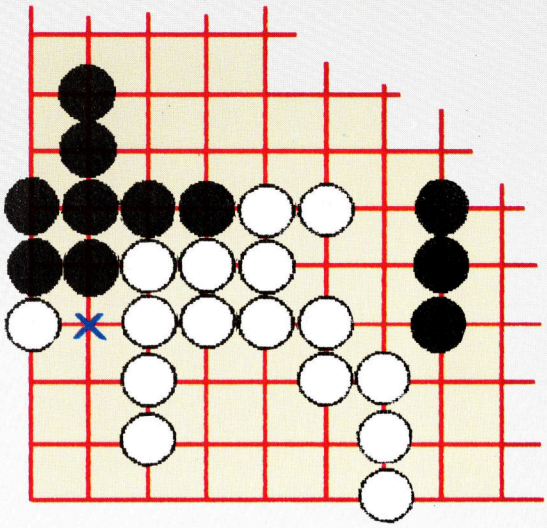
```
320 GOSUB 600
590 :
600 REM READ-HANDICAPS ROUTINE
620 HNCP=BOARD+512
640 FOR L=0 TO 43
650 READ H%:POKE HNCP+L,H%
660 NEXT
670 DATA  68,204
680 DATA  68,204, 76
690 DATA  68,204, 76,196
700 DATA  68,204, 76,196,136
710 DATA  68,204, 76,196,132,140
720 DATA  68,204, 76,196,132,140,136
730 DATA  68,204, 76,196,132,140, 72,200
740 DATA  68,204, 76,196,132,140, 72,200
,136
750 RETURN
760 :
770 REM**********************************
1400 MSK%=0:GOSUB 4330
1580 GOSUB 1630
1620 :
1630 REM HANDICAP ROUTINE
1650 Q%=INT((HND%-2)/2*(HND%+1)+0.5)
1660 FOR L=Q% TO Q%+HND%-1
1670 P%=PEEK(HNCP+L):POKE BOARD+P%,BLACK%
1680 NEXT
1690 RETURN
1700 :
1710 REM**********************************
4030 :
4040 REM COUNT ROUTINE
4050 CLIB%=0:CSTN%=0
4070 SP%=CP%:SC%=CC%:GOSUB 4130
4080 MSK%=COLOUR%:GOSUB 4330
4090 RETURN O THEN RETURN
4150 IF (PEEK(BOARD+SP%) AND COLOUR%)=0
GOTO 4250
4160 IF (PEEK(BOARD+SP%) AND SC%)=0 THEN
 RETURN
4170 IF (PEEK(BOARD+SP%) AND MARKER%)>0
THEN RETURN
4100 :
4110 REM**********************************
4120 :
4130 REM SEARCH ROUTINE
4140 IF (SP% AND 240)=0 OR (SP% AND 15)=
4180 POKE BOARD+SP%,SC%+MARKER%
4190 CSTN%=CSTN%+1
4195 SK%(STACK%)=SP%:STACK%=STACK%+1
4200 SP%=SK%(STACK%-1)+DIR%(1):GOSUB 413
0
4210 SP%=SK%(STACK%-1)+DIR%(2):GOSUB 413
0
4220 SP%=SK%(STACK%-1)+DIR%(3):GOSUB 413
0
4230 SP%=SK%(STACK%-1)+DIR%(4):GOSUB 413
0
4235 STACK%=STACK%-1:SP%=SK%(STACK%)
4240 RETURN
4250 IF (PEEK(BOARD+SP%) AND LIBERTY%)>0
 THEN RETURN
4260 POKE BOARD+SP%,LIBERTY%
4270 CLIB%=CLIB%+1
4290 RETURN
4300 :
4310 REM**********************************
4320 :
4330 REM CLEAR ROUTINE
4350 FOR L=0 TO 255
4360 POKE BOARD+L,PEEK(BOARD+L) AND MSK%
4370 NEXT
4380 RETURN
4390 :
4400 REM**********************************
```

Schneider CPC 464/664

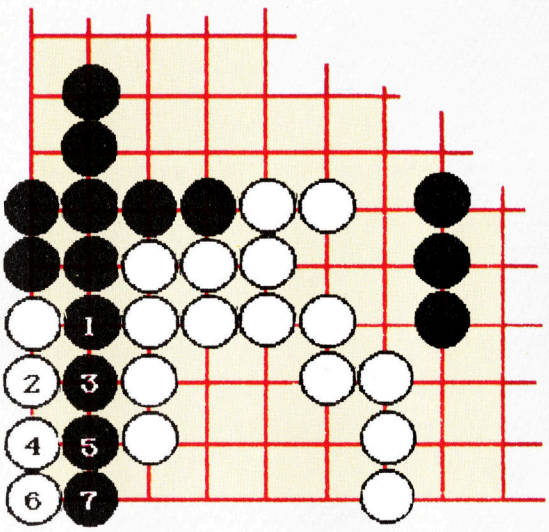
```
270 GOSUB 600:REM read handicaps
590 :
600 REM read handicaps routine
620 hncp=board+&200
630 RESTORE 670
640 FOR I%=o TO 43
650 READ h%:POKE(hncp+I%),h%
660 NEXT I%
670 DATA &44,&cc
680 DATA &44,&cc,&4c
690 DATA &44,&cc,&4c,&c4
700 DATA &44,&cc,&4c,&c4,&88
710 DATA &44,&cc,&4c,&c4,&84,&8c
720 DATA &44,&cc,&4c,&c4,&84,&8c,&88
730 DATA &44,&cc,&4c,&c4,&84,&8c,&48,&c8
740 DATA &44,&cc,&4c,&c4,&84,&8c,&48,&c8
,&88
750 RETURN
760 :
770 REM **********************
1400 mask%=0:GOSUB 4330:REM clear routine
1580 GOSUB 1630:REM hahdicap routine
1620 :
1630 REM handicap routine
1650 q%=INT((hand%-2)/2*(hand%+1)+0.5)
1660 FOR I%=q% TO q%+hand%-1
1670 LET p%=PEEK(hncp+I%):POKE(board+p%)
,black%
1680 NEXT I%
1690 RETURN
1700 :
1710 REM **********************
4030 :
4040 REM count routine
4050 clib%=0:cstn%=0
4060 cloc%(1)=0:cloc%(2)=0
4070 sp%=cp%:sc%=cc%:GOSUB 4130:REM search
4080 mask%=colour%:GOSUB 4330:REM clear
4090 RETURN
4110 REM **********************
4120 :
4130 REM search routine
4140 IF (sp% AND 240)=0 OR (sp% AND 15)=
0 THEN RETURN
4150 IF (PEEK(board+sp%)AND colour%)=0 T
HEN 4250
4160 IF (PEEK(board+sp%)AND sc%)=0 THEN
RETURN
4170 IF (PEEK(board+sp%)AND marker%)>0 T
HEN RETURN
4180 POKE(board+sp%),sc%+marker%
4190 cstn%=cstn%+1
4195 s(stack%)=sp%:stack%=stack%+1
4200 sp%=s(stack%-1)+dir%(1):GOSUB 4130
4210 sp%=s(stack%-1)+dir%(2):GOSUB 4130
4220 sp%=s(stack%-1)+dir%(3):GOSUB 4130
4230 sp%=s(stack%-1)+dir%(4):GOSUB 4130
4235 stack%=stack%-1:sp%=s(stack%)
4240 RETURN
4250 IF (PEEK(board+sp%) AND liberty%)>0
 THEN RETURN
4260 POKE(board+sp%),liberty%
4270 clib%=clib%+1
4290 RETURN
4300 :
4310 REM ********************
4320 :
4330 REM clear routine
4350 FOR I%=0 TO 255
4360 POKE(board+I%),(PEEK(board+I%) AND
mask%)
4370 NEXT I%
4380 RETURN
4390 :
4400 REM ********************
```

Sinclair Spectrum

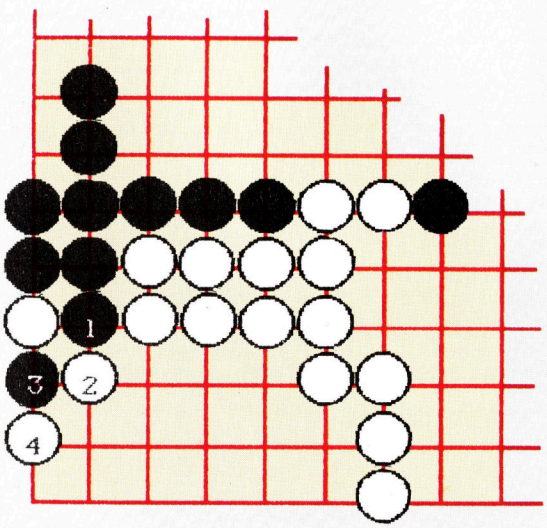
```
 270 GO SUB 600
 590:
 600 REM read-handicaps routine
 620 LET hncp=board+512
 630 RESTORE 670
 640 FOR l=0 TO 43
 650 READ h: POKE hncp+l,h
 660 NEXT l
 670 DATA  68,204
 680 DATA  68,204, 76
 690 DATA  68,204, 76,196
 700 DATA  68,204, 76,196,136
 710 DATA  68,204, 76,196,132,140
 720 DATA  68,204, 76,196,132,14
0,136
 730 DATA  68,204, 76,196,132,14
0, 72,200
 740 DATA  68,204, 76,196,132,14
0, 72,200,136
 750 RETURN
 760:
 770 REM ************************
1400 LET mask=0: GO SUB 4330
1580 GO SUB 1630
1620:
1630 REM handicap routine
1650 LET q=INT ((hand-2)/2*(hand
+1)+0.5)
1660 FOR l=q TO q+hand-1
1670 LET p=PEEK (hncp+l): POKE b
oard+p,black
1680 NEXT l
1690 RETURN
1700:
1710 REM ************************
4030:
4040 REM count routine
4050 LET clib=0: LET cstn=0
4070 LET sp=cp: LET sc=cc: GO SU
B 4130
4080 LET mask=colour: GO SUB 4330
4090 RETURN
4100:
4110 REM ************************
4120:
4130 REM search routine
4140 IF INT (sp/16)=0 OR sp-16*I
NT (sp/16)=0 THEN  RETURN
4150 IF PEEK (board+sp)=liberty
OR PEEK (board+sp)=0 THEN  GO TO
 4250
4160 IF PEEK (board+sp)=colour-s
c THEN  RETURN
4170 IF PEEK (board+sp)>colour T
HEN  RETURN
4180 POKE board+sp,sc+marker
4190 LET cstn=cstn+1
4195 LET s(stack)=sp: LET stack=
stack+1
4200 LET sp=s(stack-1)+d(1): GO
SUB 4130
4230 LET sp=s(stack-1)+d(4): GO
SUB 4130
4235 LET stack=stack-1: LET sp=s
(stack)
4240 RETURN
4250 IF PEEK (board+sp)>colour T
HEN  RETURN
4260 POKE board+sp,liberty
4270 LET clib=clib+1
4290 RETURN
4300:
4310 REM ************************
4320:
4330 REM clear routine
4350 FOR l=0 TO 255
4360 IF PEEK (board+l)>mask THEN
  POKE (board+l),PEEK (board+l)-
mask-1: GO TO 4360
4370 NEXT l
4380 RETURN
4390:
4400 REM ************************
```



Raumvorteil
In dieser Situation hat Weiß die untere linke Ecke des Spielbrettes eingeschlossen. Trotzdem ist das Gebiet noch nicht vollständig sicher. Das Schlüsselfeld ist mit einem Kreuz markiert.



Wettlauf am Rand
Wird Schwarz gestattet, auf dieses Schlüsselfeld zu setzen, so wird dieser Zug als „Atari" auf den weißen Stein in der Ecke des Spielbrettes bezeichnet. Weiß kann versuchen, in die Ecke zu „laufen", kann jedoch den Angriff durch Schwarz nicht verhindern.



Alternatives Spiel
Anstatt in die Ecke zu laufen, sollte Weiß den Stein in der Ecke vergessen und lieber den Vorstoß von Schwarz in das eigene Gebiet stoppen. Die entsprechende Spielweise sehen Sie in der Abbildung links.

Geborgte Zeit

Im zweiten Teil über die Grafiktalente des Commodore 64 sehen wir, wie der VIC-II Chip für seine Bildschirmanzeige Zeit vom 6510 „borgt". Das Beispielprogramm „Splitscreen" teilt den Bildschirm und aktiviert zwei unterschiedliche Grafikmodi gleichzeitig.

Um Bildschirmdaten überhaupt anzeigen zu können, muß der VIC-II Informationen aus dem Speicher holen. Ihm stehen dazu einige Adreßleitungen und alle Leitungen des Datenbusses zur Verfügung. Den Zugang zu ROM und RAM (die er sich mit dem 6510 teilt) verschafft sich der VIC-II mit der Technik des „direkten Speicherzugriffs" (DMA – direct memory access). Im Idealfall würde nun der VIC-II die Adreß- und Datenleitungen nur nutzen, wenn der 6510 sie nicht belegt. Die Abläufe der beiden Prozessoren wären dann „transparent". Doch leider ist der 6510 ein recht einfacher Prozessor mit nur wenigen Registern.

Der VIC-II muß aber, besonders bei mehreren Sprites, viele Daten lesen und hat nicht genug Zeit für eine transparente Arbeitsweise. Mit einer speziellen Steuerleitung (BA – Bus available) sendet er dem 6510 ein Wartesignal und verschafft sich so ausreichend Zeit zum Lesen der Bildschirmdaten.

Wenn der Video-Chip Zeit von dem 6510 benötigt, setzt er die BA-Leitung auf „low". Der 6510 kann dann gerade noch den aktuellen Befehl beenden, bevor der VIC-II die AEC-Leitung (Adress Enable Control) herunterschaltet und die Bustreiber des 6510 deaktiviert. Dieser Vorgang kommt einer „Betäubung" des 6510

„Splitscreen" für den Commodore 64

Ladeprogramm in BASIC

```
900 REM****************************
902 REM**     SPLITSCREEN 64      **
903 REM**      BASIC LOADER       **
910 REM** NOTE-IN THIS VERSION **
920 REM** HIRES UNDER BASIC SO **
930 REM**    CAN'T USE PEEK IN   **
940 REM**     PLOT SUBROUTINE    **
950 REM****************************
960 :
1000 REM SPLIT SCREEN LOAD AND TEST**
1010 DATA76,50,192,49,234,173,25,208
1020 DATA240,37,169,255,160,21,141,25
1030 DATA208,173,0,221,73,2,141,0,221
1040 DATA173,17,208,73,32,141,17,208,41
1050 DATA32,240,4,169,121,160,255,141
1060 DATA18,208,140,24,208,108,3,192
1070 DATA173,14,220,41,254,141,14,220
1080 DATA173,20,3,141,3,192,173,21,3
1090 DATA141,4,192,169,5,141,20,3,169
1100 DATA192,141,21,3,173,17,208,41,95
1110 DATA141,17,208,169,0,141,18,208
1120 DATA169,255,141,25,208,169,1,141
1130 DATA26,208,165,1,41,254,133,1,169
1140 DATA0,133,247,169,160,133,248,160
1150 DATA0,162,28,169,0,145,247,200,208
1160 DATA251,230,248,202,208,246,169
1170 DATA188,133,248,160,0,162,4,169
1180 DATA183,145,247,200,208,251,230
1190 DATA248,202,208,246,165,1,9,1,133
1200 DATA1,96,255
1210 DATA20633:REM*CHECKSUM*
1220 CC=0:FORI=0TO160
1230 READX:POKE49152+I,X:REM INSERT CODE
1240 CC=CC+X:NEXT
1250 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1260 REM***TEST SPLIT SCREEN***
1270 SYS49152:REM SPLIT SCREEN
1280 CM = 40960:REM START HIRES
1290 REM** PLOT VERT  AXES **
1300 X=160:FORY=0TO70
1310 GOSUB1400:REM COMPUTE ADDRESS
1320 POKEM+R,2↑(7-C):REM PLOT POINT
1330 NEXT
1340 REM** PLOT HORIZ AXES **
1350 Y=37:FORX=0TO319
1360 GOSUB1400:REM COMPUTE ADDRESS
1370 POKEM+R,255:REM PLOT SEGMENT
1380 NEXT
1390 END
1400 REM** COMPUTE S/R **
1410 U=INT(Y/8):V=INT(X/8)
1420 R=YAND7:C=XAND7
1430 M=CM + (40*U+V)*8
1440 RETURN
```

Assemblerprogramm

```
        ;++++++++++++++++++++++++++
        ;++++++++++++++++++++++++++
        ;++                      ++
        ;++   SPLITSCREEN 64     ++
        ;++                      ++
        ;++++++++++++++++++++++++++
        ;++++++++++++++++++++++++++

        ;
         *= $C000
         JMP START
        ;
         MEM1 = $F7
         MEM2 = $F9
         MEM3 = $FB
         SCN = $FD
         CINV = $314     ;IRQ VECTOR
         TEMPIRQ *=*+2
        ;
NEWIRQ =*                ;NEW IRQ WEDGE START
        ;
         LDA $D019       ;LOOK AT THE INTERRUPT STATUS
         BEQ NOTVIC      ;WAS THE IRQ FROM VIC?
         LDA #$FF
         LDY #$15
         STA $D019       ;CLEAR INT LATCH.
         LDA $DD00
         EOR #$02
         STA $DD00       ;CHANGE BANKS
         LDA $D011
         EOR #$20        ;CHANGE BIT MAP MODE BIT
         STA $D011
         AND #$20        ;TEST BMM BIT
         BEQ LSCN        ;LOWER PART OF SCREEN
         LDA #$79        ;SET RASTER TO LINE 22
         LDY #$FF
LSCN
         STA $D012       ;SET NEW RASTER COMPARE
         STY $D018
NOTVIC
         JMP (TEMPIRQ)
        ;
START =*                 ;INITIALISE IRQ WEDGE
        ;
         LDA $DC0E
         AND #$FE        ;DISABLE SCAN TIMER A
         STA $DC0E       ;CIA #1
         LDA CINV
         STA TEMPIRQ     ;
         LDA CINV+1      ;CHANGE IRQ VECTOR
         STA TEMPIRQ+1   ;
         LDA #<NEWIRQ
         STA CINV
         LDA #>NEWIRQ
         STA CINV+1
         LDA $D011
         AND #$5F        ;RESET BMM MODE
         STA $D011
         LDA #$00
         STA $D012
         LDA #$FF
         STA $D019       ;RESET IRQ LATCHES
         LDA #$01
         STA $D01A       ;ENABLE RASTER COMPARE
         LDA $01
         AND #$FE
         STA $01         ;BANK OUT BASIC ROM
         LDA #$00
         STA MEM1        ;SET UP 0 PAGE PTRS TO POINT
         LDA #$A0        ;TO START OF HIRES AREA BEHIND
         STA MEM1+1      ;BASIC ROM
         LDY #$00
         LDX #$1C
         LDA #$00        ;CLEAR HIRES
DDD
         STA (MEM1),Y
         INY
         BNE DDD
         INC MEM1+1
         DEX
         BNE DDD
         LDA #$BC
         STA MEM1+1
         LDY #$00
         LDX #$04
         LDA #$B7        ;SET COLOUR
DDE
         STA (MEM1),Y
         INY
         BNE DDE
         INC MEM1+1
         DEX
         BNE DDE
         LDA $01
         ORA #$01
         STA $01         ;RESTORE BASIC ROM
         RTS             ;BACK TO BASIC
```

gleich, der überhaupt nicht merkt, daß er nicht mehr im Spiel ist.

Der VIC-II „stiehlt" dem 6510 eine ganze Menge Zeit. So müssen beispielsweise nach Darstellung jeder achten Rasterzeile die Pointeradressen der Zeichen (die aus acht Pixelzeilen bestehen) geholt werden. Da in einer Bildschirmzeile 40 Zeichen Platz haben, werden auch 40 Pointeradressen gebraucht. Beim PAL-System erneuert der Video-Chip etwa 25mal pro Sekunde jede zweite Rasterlinie des 625zeiligen Fernsehschirms. Das amerikanische NTSC-System arbeitet mit 524 Zeilen, die pro Sekunde 30mal erneuert werden. Diese Abläufe verlangsamen die Arbeit des 6510 um 15 bis 20 Prozent.

Unterschiedlich aufgelöst

Die DMAs scheinen zunächst keinen Einfluß auf die internen Maschinenvorgänge zu haben, doch zeigt sich, daß das vorübergehende „Abschalten" des 6510 bei der Ein- und Ausgabe Probleme verursacht – beispielsweise beim Zugriff auf Cassettendateien. Dabei kann es sogar nötig werden, den Bildschirm während der E/A (mit POKE 53265,11) zu löschen und danach (mit POKE 53265,27) wieder anzuschalten. Ob der unmittelbare Speicherzugriff des VIC-II während der E/A Schwierigkeiten verursacht, hängt im wesentlichen vom externen Gerät und der Datenübertragungsmethode ab. Beim Diskettenzugriff braucht der Bildschirm nicht gelöscht zu werden.

Der Commodore 64 kann – umsichtig programmiert – zwei Grafikmodi mit hoher und niedriger Auflösung gleichzeitig auf dem Bildschirm darstellen. Damit lassen sich im oberen Teil beispielsweise Diagramme oder Grafik darstellen, während im unteren Teil Text erscheint oder Eingaben vorgenommen werden.

Unser Splitscreen-Programm richtet hinter dem ROM des BASIC-Interpreters einen Bildschirmspeicher für die hohe Auflösung ein und zeigt dessen Inhalt im oberen Bildschirmdrittel an. Der untere Teil des Schirms bleibt im normalen Textmodus.

In einem früheren Artikel wurde bereits gezeigt, wie die RAM-Vektoren des Commodore 64 eingesetzt werden. Über den Inhalt der Zwei-Byte-Pointer hatten wir den Ablauf von Betriebssystemroutinen auf unseren Programmcode geleitet. Mit dieser Technik lenken wir nun die IRQ-Routine um.

Es gibt mehrere Möglichkeiten, IRQs für den 6510 auszulösen. Im Normalfall setzt einer der CIA-Timer jede Sechzigstelsekunde ein Bit des Interrupt Flag Registers (IFR) bei Adresse 53273 ($D019). Damit wird eine IRQ-Bearbeitungsroutine ausgelöst, die unter anderem über die Matrix der Tastaturschalter abfragt, ob eine Taste gedrückt wurde. Bei bestimmten Ereignissen werden die IFR-Bits auch vom VIC-II Chip gesetzt – beispielsweise beim Zusammenstoß zweier Sprites oder wenn der Rasterzähler einen vorbestimmten Wert erreicht. Ein weiteres Register, das „Interrupt Enable Register" (IER) bei Adresse 53274 ($D01A), dient als „Schalter" für die IRQ-Leitung des 6510. Wenn im IER bestimmte Bits gesetzt werden, lösen die gesetzten Bits des IFR im 6510 ein Interrupt aus.

Hier der Aufbau von „Splitscreen":
1) Wenn am oberen Bildschirmrand ein Raster-Interrupt eintritt, wird der Bit-Map-Modus eingeschaltet und ein Raster-Interrupt auf die Bildschirmmitte gesetzt. Ein RTI gibt die Steuerung an das OS zurück.
2) Das Interrupt in der Bildschirmmitte schaltet wieder in die niedrige Auflösung, setzt den nächsten Taster-Interrupt auf den oberen Bildschirmrand und geht anschließend ins OS.

Bei diesem Aufbau wird der Teil des normalen Bildschirmspeichers, der dem oberen Bildschirmdrittel entspricht, für die Farbinformation des hohen Auflösungsmodus eingesetzt, während die restlichen zwei Drittel die normalen Zeichen enthalten.

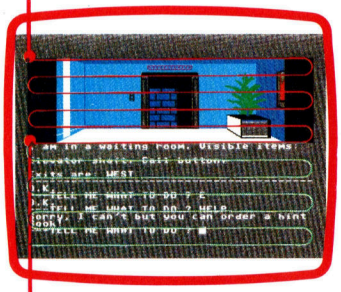
Das Programm hat jedoch einen Haken. Raster-Interrupts treten alle Fünfzigstelsekunden ein und müssen sofort bearbeitet werden, da der Rasterstrahl sonst weiterwandert und falsche Daten anzeigt. Im Normalfall würde die IRQ-Bearbeitungsroutine jede Sechzigstelsekunde ausgelöst. Da die Routine jedoch recht lang ist, würde unser Programmkeil nur mit Verzögerung ausgeführt, wenn die Routine unmittelbar vor dem Raster-Interrupt aufgerufen wird. Diese Situation muß unbedingt vermieden werden, da die Grenzlinie zwischen den beiden Auflösungsarten sonst flackert.

Beschleunigtes BASIC

Wir lösen das Problem, indem wir die Ausführung der IRQ-Bearbeitungsroutine unmittelbar hinter den Programmkeil verlegen. Der Sechzigstelsekunden-Timer wird ausgeschaltet und so die normale IRQ-Auslösung verhindert. Sofort nach Ausführung unseres Programmkeils springen wir dann auf die IRQ-Bearbeitungsroutine. Dieser Vorgang sychronisiert die beiden Codes. Zwar wird nun die Tastatur seltener abgefragt, doch läuft dafür das BASIC auch merklich schneller.

Da der Bildschirmspeicher für hohe Auflösung nun hinter dem ROM des BASIC-Interpreters liegt, wird das PLOTten in BASIC komplizierter: Die Sprache kann den RAM-Bereich hinter dem BASIC-ROM nicht ansprechen und hat so keinen Zugang zu AND- und OR-Befehlen, die unter anderem für das Setzen der Pixel nötig sind. POKE umgeht jedoch das BASIC-ROM, spricht den dahinterliegenden RAM-Bereich unmittelbar an und macht so das PLOTten vom BASIC aus erst möglich. Bei reinen Maschinencodeprogrammen kann das BASIC-ROM völlig abgeschaltet werden.

Der erste Raster-Interrupt schaltet auf hohe Bildschirmauflösung.



Der zweite Raster-Interrupt veranlaßt das Zurückschalten in den Textmodus.

Viele Abenteuerspiele für den C 64 arbeiten mit Raster-Interrupts, um hochauflösende Bilder und Text gleichzeitig darzustellen. Der von Scott Adams entwickelte „Spiderman" zeigt im oberen Bildschirmdrittel die Spielszene, während der Rest der Anzeige die Beschreibung im normalen Textmodus enthält. Der Rasterlauf wurde so programmiert, daß er bei jedem Abtasten des Bildschirms zwei Interrupts auslöst: einen am oberen Bildschirmrand und den anderen etwa in Höhe des ersten Bildschirmdrittels. Bei jedem Raster-Interrupt wird eine Routine aufgerufen, die zwischen den beiden Darstellungsmodi umschaltet.

C wie Computer

Programmiersprachen lassen sich in zwei Hauptgruppen unterteilen: Hochsprachen wie BASIC oder PASCAL und maschinennahe Sprachen. Die Sprache C überbrückt die Kluft zwischen beiden Gruppen.

In den letzten Jahren wurde mit viel Energie an Programmiersprachen gearbeitet, die menschliche Denkvorgänge imitieren und den Programmierer von den Eingrenzungen der Hardware freihalten sollten. Trotz aller Anstrengungen ist aber immer noch viel Programmierarbeit nötig, wenn Hardware direkt gesteuert werden muß, damit kleine und relativ langsame Maschinen maximale Leistung erzielen können.

Anpassungen dieser Art werden üblicherweise in der Assemblersprache vorgenommen. Gelegentlich finden auch Sprachen Anwendung (zum Beispiel FORTRAN), die auf eine bestimmte Hardware abgestimmt sind. Assemblersprachen erreichen zwar (geschickt programmiert) maximale Wirkung, sind aber nur auf einen Maschinentyp ausgerichtet. Es bestand daher schon seit langem der Bedarf für eine einfache Programmiersprache auf relativ niedriger Ebene, die auf einem breiten Maschinenspektrum läuft und die Strukturen und Vorteile der Hochsprachen mit der direkten Hardwaresteuerung verbindet.

BCPL war die erste Sprache, die diesen Anforderungen entsprach. Sie wurde von Martin Richards an der Cambridge University entwikkelt. Als Ken Thompson und seine Mitarbeiter in den Bell Laboratorien am Betriebssystem Unix arbeiteten, suchten sie dafür eine Hochsprache und übernahmen viele Konzepte von BCPL in eine Sprache namens B. Kernighan und Ritchie entwickelten aus B schließlich C und veröffentlichten die Sprachdefinition 1978 in dem Buch „The C Programming Language", das als Standard für C-Anwendungen gilt – bis letztlich ein neuer internationaler Standard entwickelt wird.

C und Unix standen immer in enger Verbindung. Von den 13 000 Codezeilen, aus denen die Kernroutinen von Unix bestehen, wurden nur etwa 800 Zeilen in Assembler entwickelt, der Rest entstand in C. Durch ihren geringen Befehlsumfang ist C leicht erlernbar. Sie vereinfacht außerdem die Erstellung von Compilern und läßt sich damit leicht auf viele verschiedene Maschinentypen anpassen. Da C sehr maschinennah ist, ist auch ihr Code kompakt und schnell. Es wurden damit nicht nur Unix und Unix-Anwendungen geschrieben, sondern auch größere Teile von MS-DOS und CP/M erstellt.

C-Compiler gibt es für fast alle Micros und auch für viele Maschinen der Groß-EDV. Softwarehäuser können unter C erstellte Programme leicht auf völlig verschiedenartige Geräte wie den Apple Macintosh und IBM PC übertragen. In dieser Serie halten wir uns an die Version von Kernighan und Ritchie und setzen – beispielsweise bei vorgefertigten Funktionen – das Betriebssystem Unix voraus.

C-Programme werden aus anwenderdefinierten Funktionen aufgebaut, die eine Reihe von Argumenten, Werten oder Pointer auf Werte annehmen können und Werte oder Pointer zurückliefern (die weiterverarbeitet oder ignoriert werden können). Jede Funktion wird über ihren Namen aufgerufen.

Die Funktion „main"

Alle C-Programme beginnen mit der Ausführung der Funktion „main". Die äußerste Programmebene muß daher eine Definition dieser Funktion sein. Das folgende Programm ist zwar sehr einfach, aber dennoch vollständig:

```
main()
{
    printf("hallo Welt/n");
}
```

Es zeigt einige wichtige Eigenschaften der Sprache. Die Definition einer Funktion besteht aus einem Funktionsnamen, gefolgt von einer in Klammern stehenden Argumentenliste. In diesem Beispiel hat main zwar keine Argumente, muß aber Klammern enthalten.

Der Hauptteil der Funktionsdefinition besteht aus einer Folge von Anweisungen (in diesem Fall gibt es nur eine), die von Klammern ([]) umschlossen sind und mit einem Semikolon enden. Die Klammern ähneln dem „Begin . . .End" von PASCAL. Sie umschließen eine beliebige Folge von Befehlen oder Deklarationen, die nun wie einzelne Befehle eingesetzt werden können. Der Gebrauch des Semikolons ist jedoch einfacher als PASCAL – jede vollständige Anweisung wird durch ein Semikolon abgeschlossen. Unser Programm enthält nur eine einzige Anweisung – die Standardfunktion printf.

Die Ein- und Ausgabe wurde in C nicht fest definiert, da sie von Gerät zu Gerät stark variieren kann. Statt dessen liefert jede Anwendung einen eigenen Satz von E/A-Funktionen, die dem Sprachstandard entsprechen müssen. Der von printf zurückgelieferte Wert hat keine

Hisoft-C läuft auf Schneider und Spectrum Computern. Die Version hält sich eng an den Standard, verfügt jedoch nicht über das Fließkommaformat. Hisoft hat jedoch eine Fließkommaversion angekündigt, die in Kürze herauskommen soll. Der einzige andere Nachteil ist das Handbuch. Es ist zwar umfassend angelegt, aber so unübersichtlich, daß es sich für Anfänger nicht eignet. Wenn Sie jedoch schon Erfahrungen mit der Sprache haben oder über ein anderes Lehrbuch verfügen, ist dieses Paket sehr brauchbar.

Escape-Zeichen

/n Neue Zeile
/t Horizontales Tab
/b Rückwärtsschritt
/r Return
/f Seitenvorschub
/' Gibt ein Apostroph aus (wird als Zeichenbegrenzung eingesetzt)
/" Gibt ein Anführungszeichen aus (wird als Stringbegrenzung eingesetzt)
/d d ist eine Oktalstelle; gibt das Zeichen mit dem entsprechenden ASCII-Code aus
/xh h ist eine Hexadezimalstelle; gibt das Zeichen mit entsprechendem ASCII-Code aus
/e Escape
/v Vertikales Tab

Formatumwandlung

Zeichen	Argumenttyp	Bedeutung
d	int	Dezimalformat
o	int	Oktalformat
x	int	Hexadezimalformat
u	int	Dezimalformat ohne Vorzeichen
c	char/int	Ausgabe als einzelnes Zeichen
s	*char	Ausgabe als String
e	float/double	Mantisse-Exponent-Format
f	float/double	Normales Dezimalformat
g	float/double	das kürzeste Format von d, e oder f
%		Ausgabe des %-Zeichens

Bedeutung und wird ignoriert. printf kann eine beliebige Anzahl von Argumenten haben: numerische, Zeichenvariablen oder Strings (siehe oben). Auch gibt printf nicht automatisch ein Return aus, sondern nur auf Anweisung. C hat dafür spezielle Stuersymbole, die mit dem „Backslash" (\) beginnen. Das Symbol für Return ist \n (siehe Tabelle der Escape-Steuerzeichen).

Das folgende Programm erzeugt das gleiche Ergebnis wie unser erstes Beispiel:

```
main()
{
    printf("hallo");
    printf("Welt");
    printf("/n");
}
```

C arbeitet mit Typen – das heißt, der Typ jeder Variable muß deklariert werden. Folgende Typen sind ständig vorhanden:

char – Zeichen, die ein Byte belegen
short – kurze Ganzzahlen
int – normale Ganzzahlen
long – ganze Ganzzahlen
float – reelle Zahlen im Fließkommaformat
double – Zahlen im Fließkommaformat mit doppelter Genauigkeit

Variablendeklarationen bestehen aus einem Typennamen, gefolgt von einer Liste sämtlicher Variablennamen dieses Typs. Alle Variablen müssen deklariert werden, wobei sich die Deklarationen an einer beliebigen Stelle des Programms befinden können. Variablennamen dürfen beliebig lang sein, jedoch werden nur die ersten acht Stellen verarbeitet.

Wie auch in anderen Sprachen dürfen Variablennamen in C keine Leerzeichen enthalten, wohl aber das Zeichen für Unterstreichung (Underline). Die Namen dürfen weiterhin nicht mit einer Zahl oder einem Underline beginnen. Groß- und Kleinbuchstaben werden unterschieden: A ist eine andere Variable als a. Die Bytezahl der numerischen Typen ist nicht festgelegt und kann je nach Anwendung variieren. Eine Ganzzahl vom Typ „short" hat normalerweise acht Bits, „int" 16 Bits und „long" 32 Bits.

Zuordnungen und mathematischen Abläufe folgen den üblichen Regeln, wobei das Gleichheitszeichen (=) die Rolle des Zuordnungsoperators spielt. Mit % eingesetzt, ergeben die arithmetischen Operatoren +, –, * und / den Modulowert:

```
int x,y,z
. . .
z=x%y
```

Ausdrücke und Zuordnungen können beliebig viele numerische Typen und „char" enthalten, wobei char hierbei als Ein-Byte-Ganzzahl (mit dem ASCII-Code eines Zeichens) angesehen wird. C führt alle Typenumwandlungen von „niedrigen" in „höhere" Typen automatisch aus. Hierbei ist interessant, daß normalerweise alle Operationen im Fließkommaformat in doppelter Genauigkeit ausgeführt werden, selbst wenn alle Operanden nur vom Typ „float" sind. Typenänderungen lassen sich aber auch forcieren, indem der neue Typ in Klammern vor dem Variablennamen aufgeführt wird:

```
int n;
float f;
. . .
f=sqrt((double)n);
```

übernimmt eine Kopie des Wertes n in doppelter Genauigkeit, da die Quadratwurzelfunktion als Argument den Type double erwartet. Der Inhalt von n bleibt dabei unverändert.

Mit ++ und –– besitzt C zwei weitere praktische Zusatzoperationen für die In- und Dekrementierung. ++a inkrementiert den Wert a und, ––a dekrementiert ihn, bevor der aktuelle Befehl ausgeführt wird. Bei a++ und a–– geschieht die In- und Dekrementierung erst nach Beendigung des aktuellen Befehls.

```
int a,b
. . .
b=1;
a=b++; /*ergibt 1 in a und 2 in b*/
```

während

```
b=1
a=++b;/* die 2 sowohl in a und b ergibt */
```

Da diese Zuordnung als Funktion behandelt wird, liefert sie auch ein Ergebnis zurück: den zugeordneten Wert. Dieser Wert läßt sich nun in einem anderen Ausdruck wie

```
x=y=z;
```

weiterverarbeiten. Das Beispiel ordnet x und y den Wert von z zu.

Wir hatten bereits erwähnt, daß Ein- und Ausgabe von Standardfunktionen ausgeführt werden, die je nach Anwendung variieren können. Da es jedoch schwierig ist, Programme ohne ein E/A-Format zu schreiben, gehen wir am Ende dieser Folge noch kurz auf die beiden Hauptfunktionen der Terminal-E/A ein. Eine kennen wir bereits: printf formatiert Werte der verschiedensten Typen und sendet sie an das Standard-Ausgabegerät (normalerweise den Bildschirm). Das vollständige Format sieht so aus:

```
printf(control-string, arg1, arg2, . . . . .);
```

Argumente können Strings oder Variablen der grundlegenden Datentypen sein.

Der „Control-string" kann normale Zeichenfolgen enthalten, die ohne Veränderung ausgegeben werden. Er hat jedoch auch spezielle Formatierungsangaben für jedes der Argumente und legt auch die Anzahl der Argumente fest. Formatierungsangaben beginnen mit % und enden mit einem der Umwandlungszeichen (siehe Tabelle). Zwischen diesen beiden Zeichen bestimmt das Zeichen „–" die Linksbündigkeit des Ausgabefeldes, eine Zahl jedoch die Feldbreite einer Zahlenausgabe. Ein Punkt gefolgt von einer weiteren Zahl legt (bei einem String) die Anzahl der anzuzeigenden Zeichen fest oder (bei Zahlen vom Typ float oder double precision) die Zahl der Nachkommastellen.

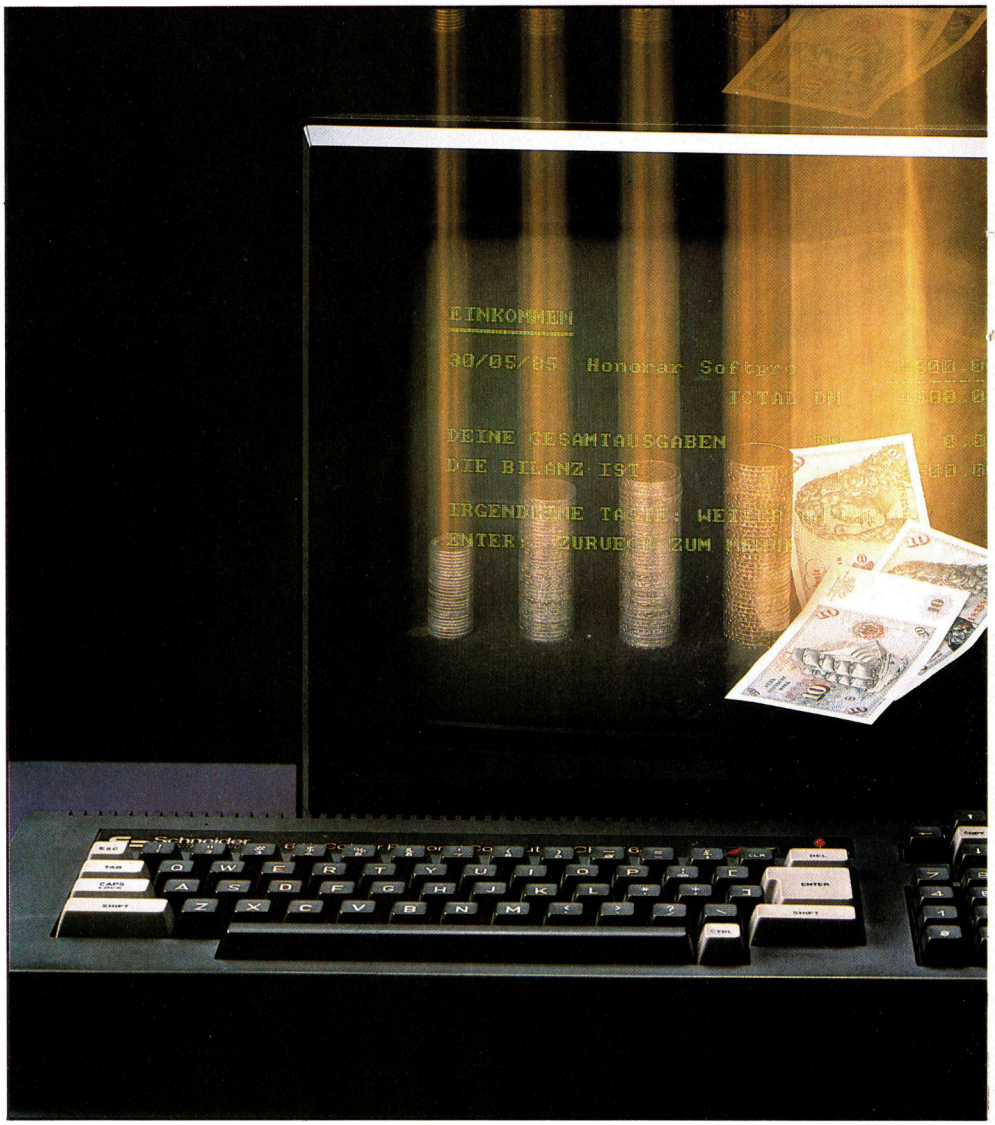
Computerhilfe in Geldfragen

Computer haben viele nützliche Seiten. In regelmäßiger Folge wollen wir Sie mit einigen davon bekannt machen und Ihnen zeigen, wie Sie sie nutzen. Den Anfang macht ein Kostenkalkulator für die meist gängigen Computersysteme, der Sie bei der Haushaltsbuchführung unterstützen kann und überdies eine große Hilfe für die Erstellung Ihrer Steuererklärung darstellt.

Wo bleibt nur das ganze Haushaltsgeld? Die Übersicht über Einnahmen und Ausgaben zu behalten, ist ein Problem, das jeder nur allzu gut kennt. Doch wie ein Geschäftscomputer kann auch Ihr Microrechner Finanzdaten speichern und verarbeiten. Hier ist ein einfaches Programm, das Ihnen gute Übersicht über Einnahmen und Ausgaben ermöglicht.

Der „Kostenkalkulator" wurde geschrieben, um diese Aufgabe etwas leichter zu machen. Das Programm läuft auf den Atari-Modellen 400, 800, 600XL, 800XL und 130XE, auf dem Sinclair Spectrum, Commodore 128 und 64 sowie Schneider CPC 464. Auf dem VC20 muß die Bildschirmausgabe der kürzeren Zeile angepaßt werden.

Um eine Bilanz zu aktualisieren, füttern Sie einmal im Monat (oder wann immer Sie Zeit haben) den Rechner mit Einzelposten ihres Einkommens, etwa Gehalt oder Taschengeld und den Ausgaben anhand von Kontoauszügen und Scheckbüchern. So gibt der Rechner jederzeit einen Überblick darüber, wann und wo Geld ausgegeben wurde und wie Einnahmen zu Ausgaben stehen.



Ausgaberubriken

Das Programm ist ziemlich lang. Ist es aber einmal eingegeben und auf Cassette gespeichert, kann es immer wieder in den Rechner geladen werden und steht somit ständig zur Verfügung.

Das Programm stellt eine Spalte für das Einkommen und sieben für die Ausgaben unter verschiedenen Rubriken zur Verfügung. Die Rubriken können den persönlichen Bedürfnissen angepaßt und verändert werden: Alles, was Sie dazu tun müssen, ist, den Wortlaut in den DATA-Zeilen des Programms zu ändern, bevor es gestartet wird. Das Einkommen muß die letzte Eingabe sein, und Sie müssen acht Spalten festlegen. Sonst kann es nicht so einfach funktionieren.

Das Programm wird in zwei Schritten abgespeichert: zunächst das Programm selbst, dann alle Informationen, mit denen es beim letzten Benutzen „gefüttert" wurde. Das bedeutet, daß Sie zwei Programmnamen benötigen, einen für jeden der zwei Teile.

Um das Programm zu speichern, folgen Sie der normalen „Speicher-Prozedur" für den Computer so, wie sie im Handbuch beschrieben ist.

Um das Programm zu laden, folgen Sie der üblichen Ladeprozedur, die Sie von eigenen Programmen kennen.

Anweisungen für das Laden und Speichern der DATAs werden etwas später gegeben. Wenn das Programm startet, bietet das Hauptmenü immerhin sieben Auswahlmöglichkeiten:

1. Daten eingeben
2. Daten abfragen
3. Daten speichern
4. Daten laden
5. Drucken ja/nein
6. Daten ändern
7. Programm beenden

Um einen Eintrag zu machen, sollten Sie die Taste 1 betätigen, wenn das Hauptprogramm erscheint. Nicht ENTER oder RETURN in diesem Stadium drücken. Der Computer wird der Reihe nach die verschiedenen Einzelinformationen abfragen. Er erkundigt sich nach Datum, Buchungstext (hier kann eingetragen werden, was gekauft wurde oder welcher Art das Einkommen ist), Betrag, Rubrik (eine der Rubriken, die in die DATA-Zeilen eingetragen wurden). Am Schluß jeder in dieser Reihenfolge eingegebenen Informationen dient die ENTER- oder RETURN-Taste zur Bestätigung.

Wenn Sie die Einträge beendet haben, warten Sie, bis der Computer nach neuen Daten fragt. ENTER oder RETURN führen Sie schnell zurück zum Hauptmenü.

Um dann einen oder mehrere Einträge anzusehen, wählen Sie Menüpunkt 2, wenn das Hauptmenü erscheint. Auf keinen Fall die ENTER- oder RETURN-Taste drücken.

Bilanzieren

Der Computer zeigt eine Anzahl von verschiedenen Rubriken: sieben Ausgabeposten, einen Einkommensposten. Um eine Rubrik zu wählen, ist die passende Nummer (wiederum nicht ENTER oder RETURN) einzugeben. Der Computer wird alle Posten auswerfen, die er in dieser Kategorie gespeichert hat, mit dem Gesamtbetrag am Ende.

Auf dem Bildschirm des Spectrum wird die Frage „scroll?" angezeigt, wenn der Bildschirm zu klein für alle gleichzeitig anzuzeigenden Einträge ist. Nicht die Taste N in diesem Stadium des Programms betätigen; es muß der Reihenfolge nach durch die Liste gehen.

Sobald Sie fertig sind, können Sie mit ENTER oder RETURN zum Hauptmenü zurückkehren. Wenn Rubrik 8 gewählt wurde, wird nicht nur das genaue Gesamteinkommen, sondern auch die Posten sämtlicher Ausgabe-Rubriken plus einer Bilanz des Einkommens und der Ausgaben angezeigt.

Wenn Sie Taste 6 drücken, um einen Eintrag zu ändern, wird der Computer eine Liste von allen gemachten Einträgen anzeigen, unabhängig von der entsprechenden angewählten Rubrik.

Sobald nach einer Änderung ENTER oder RETURN gegeben wird, bringt Sie der Computer automatisch ins Hauptmenü zurück. Um eine zweite Änderung zu machen, sollten Sie nochmals Punkt 6 anwählen.

Speichern

Wenn der Menüpunkt 5 (ohne ENTER oder RETURN) im Hauptprogramm angesprochen wird, fragt der Computer mit J/N, ob der Drucker eingesetzt werden soll. Mit J kann man zum Hauptmenü zurückkehren. Alle Informationen, die normalerweise der Bildschirm anzeigt, werden statt dessen ausgedruckt, wenn der Menüpunkt 2 angewählt ist. Wollen Sie nichts mehr ausdrucken, kehren Sie zu Menüpunkt 5 zurück und schalten den Drucker aus.

Seien Sie vorsichtig, wenn Sie J drücken und der Drucker nicht angeschlossen ist. Der Spectrum und der Atari werden den Befehl ignorieren, aber auf jedem der anderen Computer können Sie leicht Daten verlieren.

Die wichtigsten Schritte zum Speichern und Laden sehen so aus: Um die Finanzdaten zu speichern, tippen Sie Menüpunkt 3 (ohne ENTER oder RETURN). Jetzt geben Sie einen Namen für die Datei, z. B. „Gelddatei", ein. Dann ist nur noch RETURN oder ENTER und die "RECORD"-Taste auf dem Cassettenrecorder zu betätigen. Sind alle Daten gespeichert, können Sie zum Hauptmenü zurückkehren, von wo Menüpunkt 7 angewählt werden kann, um nunmehr das Programm zu beenden.

Um die Informationen, die zuvor gespeichert wurden, in das System zu laden, wählen Sie Punkt 4 im Hauptmenü. Jetzt den Dateinamen eingeben, RETURN oder ENTER und anschließend die „Play"-Taste auf dem Cassettenrecorder drücken, und schon ist Ihre persönliche Finanzübersicht abrufbereit.

So sind Sie ständig auf dem Laufenden, was den Verbleib Ihres Geldes angeht.

Mit dieser Haushaltsbuchführung kann Ihr Heimcomputer eine sinnvolle Aufgabe übernehmen und erweist sich so als nützlicher Helfer.

Schneider

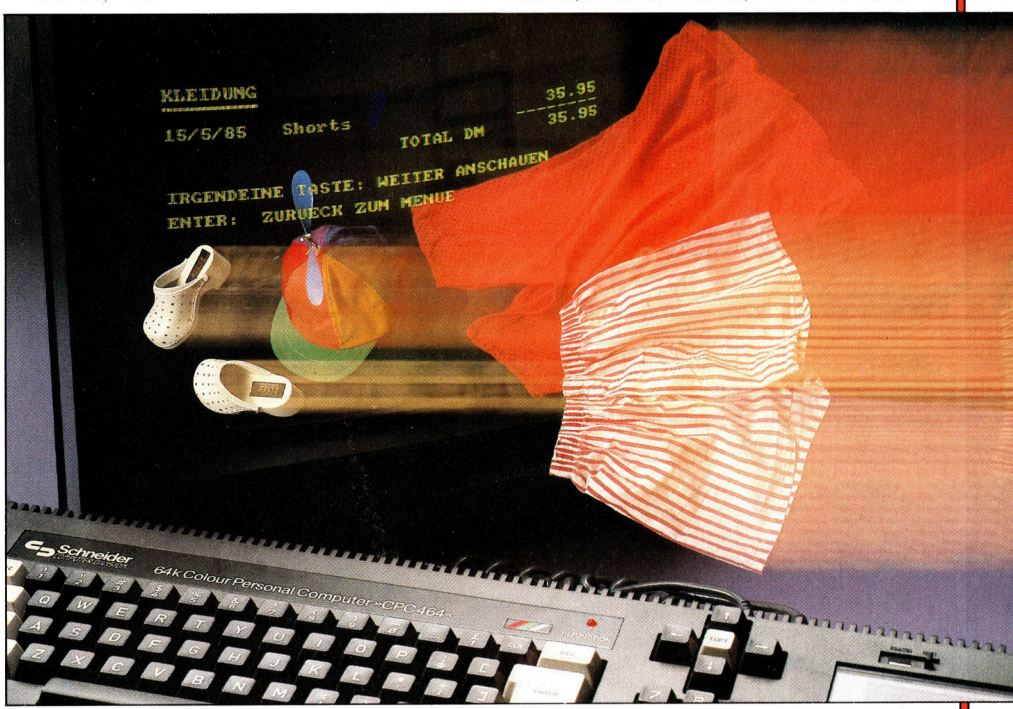
```
10 INK 0,0: INK 1,24: BORDER 0: ZONE 10:
   MODE 1
20 n=0: w=0: pay=0: spent=0:
   DIM a$(300),a(300),d$(300),k$(7)
50 FOR t=0 TO 7: READ k$(t): NEXT
60 GOSUB 660 ' HAUPTMENUE
70 ON a GOSUB 160,560,800,850,920,970,
   90
80 IF a<>7 THEN 60
90 PRINT: PRINT "Bist Du sicher ?"
100 b$=INKEY$: IF b$="" THEN 100 ELSE
   IF LOWER$(b$)="j" THEN CLS: END
110 GOTO 60
160 '*** EINGABE ***
170 z=6: CLS
180 IF n>299 THEN PRINT: PRINT:
   PRINT "□S□P□E□I□C□H□E□R
   □□□V□O□L□L□□!": SOUND 1,
   30,50,15: CALL &BB18: RETURN
190 PRINT: PRINT "Mit RETURN im DATUM-
   Feld zum MENUE"
200 PRINT: PRINT "DATUM□□□□□□
   BUCHUNGSTEXT□□□□□BETRAG
   □□□RUBR"
220 LOCATE 1,z: INPUT "",d$(n+1): IF
   d$(n+1)="" THEN 350
230 LOCATE 11,z: INPUT "",a$(n+1):
   LOCATE 27,z: INPUT "",a(n+1): LOCATE
   36,z: INPUT "",ca$
240 d$(n+1)=LEFT$(d$(n+1),8): a$
   (n+1)=LEFT$(a$(n+1),16)
250 GOTO 270
260 LOCATE 36,z: PRINT "□□□□□□":
   LOCATE 36,z: INPUT "",ca$
270 x=0: FOR t=0 TO 7: IF INSTR(k$(t),
   UPPER$(ca$))=1 THEN x=x+1:y=t
280 NEXT
290 IF x<>1 THEN 260
300 a$(n+1)=CHR$(y)+a$(n+1)
310 IF y=1 THEN pay=pay+a(n+1) ELSE
```


```
    spent=spent+a(n+1)
320 z=z+1: n=n+1
330 IF z>19 THEN z=0: CLS
340 GOTO 220
350 RETURN
360 ' *** RUBRIKEN ZEIGEN ***
370 CLS: sum=0
380 PRINT #w: PRINT #w,k$(c): PRINT
    #w,STRING$(LEN(k$(c)),154)
400 FOR t=1 TO n
410 IF n=0 THEN 460
420 s$=RIGHT$(a$(t),1)
430 IF ASC(LEFT$(a$(t),1))<>c THEN 460
440 PRINT #w: PRINT #w,d$(t);TAB(11);
    RIGHT$(a$(t),LEN(a$(t))-1);TAB(31);
    USING "#######.##";a(t)
450 sum=sum+a(t)
460 NEXT
470 PRINT #w,TAB(32);"---------
    -"
480 PRINT #w,TAB(21);"TOTAL DM",USING
    "########.##";sum
490 IF c<>7 THEN 520
500 PRINT #w: PRINT #w: PRINT
    #w,"DEINE GESAMTAUSGABEN DM ",
    USING "########.##";spent
510 PRINT #w: PRINT #w,"DIE BILANZ
    IST";SPC(12);"DM□",USING "###
    ########.##";pay-spent
520 w=0
540 PRINT: PRINT: PRINT "IRGENDEINE
    TASTE: WEITER ANSCHAUEN": PRINT:
    PRINT "ENTER: ZURUECK ZUM MENUE"
550 b$=INKEY$: IF b$="" THEN 550 ELSE
    g=ASC(b$): RETURN
560 ' *** ANSCHAUEN ***
570 CLS: PRINT
580 FOR t=0 TO 7: PRINT: PRINT TAB(10);
    DEC$((t+1,"#");"□□";k$(t): NEXT
590 PRINT: PRINT: PRINT "WELCHE RUBRIK
    NUMMER ?"
600 b$=INKEY$+" ": c=ASC(b$)-49
610 IF c=-36 THEN RETURN
620 IF c<0 OR c>7 THEN 600
630 GOSUB 360 'RUBRIKEN ZEIGEN
640 IF g=13 THEN 650 ELSE 570
650 RETURN
660 ' *** MENUE ***
670 CLS
680 LOCATE 10,2: PRINT "HAUPT MENUE"
690 LOCATE 10,5: PRINT "1 - Daten
    eingeben"
700 LOCATE 10,7: PRINT "2 - "Daten
    abfragen"
710 LOCATE 10,9: PRINT "3 - Daten
    speichern"
720 LOCATE 10,11: PRINT "4 - Daten
    laden"
730 LOCATE 10,13: PRINT "5 - Drucker
    ja/nein"
740 LOCATE 10,15: PRINT "6 - Daten
    aendern"
750 LOCATE 10,17: PRINT "7 - Programm
    beenden"
760 LOCATE 10,20: PRINT "Deine Wahl ?"
770 b$=INKEY$+"□": a=ASC(b$)-48
780 IF a<1 OR a>7 THEN 770
790 RETURN
800 ' *** SAVE ***
810 INPUT "Name: ",d$
815 IF d$="" THEN RETURN
817 PRINT "REC und PLAY - dann eine Taste
    druecken"
820 CALL &BB18: PRINT "Daten werden
    abgespeichert"
830 OPENOUT "!"+d$: PRINT #9,n
840 FOR t=1 TO n: WRITE #9,d$(t),a$(t),
    a(t): NEXT: CLOSEOUT: RETURN
850 ' *** LOAD ***
860 PRINT: INPUT "Name: ",d$
870 PRINT "PLAY druecken - Daten werden
    geladen"
880 OPENIN "!"+d$: INPUT #9,n
890 FOR t=1 TO n: INPUT #9,d$(t),a$(t),
    a(t)
900 IF ASC(a$(t))=7 THEN pay=pay+a(t)
    ELSE spent=spent+a(t)
910 NEXT: CLOSEIN: RETURN
920 ' *** DRUCKEN ***
930 PRINT: PRINT "Printer ? (j\n)"
940 b$=INKEY$: IF b$="" THEN 940 ELSE
    IF LOWER$(b$)="j" THEN w=8: GOTO
    960
950 IF LOWER$(b$)<>"n" THEN 940 ELSE
    w=0
960 RETURN
970 ' *** AENDERN ***
980 CLS: t=1
990 IF n=0 THEN RETURN
1000 WHILE (b$<>" " AND
    b$<>CHR$(13))
1010 CLS: PRINT: PRINT: PRINT "EINTRAG
    NUMMER";STR$(t),k$(ASC(a$(t))): PRINT
1015 PRINT d$(t),RIGHT$(a$(t),LEN(a$(t))
    -1);TAB(25);"DM";TAB(31);USING "#
    ######.##";a(t): PRINT:
    PRINT
1020 PRINT: PRINT: PRINT "V(orwaerts)
    Z(urueck)": PRINT: PRINT"Aendern mit
    Leertaste"
1025 PRINT: PRINT "Mit RETURN zurueck zum
    Menue"
1030 b$=INKEY$: IF b$="" THEN 1030
1040 IF LOWER$(b$)="z" THEN t=t-1: IF
    t<1 THEN t=1
1050 IF LOWER$(b$)="v" THEN t=t+1:
    IF t>n THEN t=n
1060 WEND
1070 IF b$=CHR$(13) THEN RETURN
1080 e=t: PRINT: PRINT "DATEN
    AENDERN !"
1090 ca$=CHR$(ASC(a$(e)))
1100 IF ASC(a$(e))=7 THEN pay=pay-
    a(e) ELSE spent=spent-a(e)
1110 INPUT "DATUM□",q$: IF q$<>""
    THEN d$(e)=q$
1120 INPUT "BUCHUNGSTEXT□ ",q$: IF
    q$<>"" THEN a$(e)=q$ ELSE a$(e)=
```

```
   RIGHT$(a$(e),LEN(a$(e))-1)
1130 INPUT "BETRAG□",q$: IF q$<>""
   THEN a(e)=VAL(q$)
1140 INPUT "RUBRIK ",q$: IF q$<>""
   THEN ca$=q$
1150 GOTO 1170
1160 INPUT "Neue RUBRIK ",ca$
1170 x=0: FOR t=0 TO 7
1180 IF INSTR(k$(t),UPPER$(ca$))=1 THEN
   x=x+1:y=t
1190 NEXT
1200 IF x<>1 THEN 1160
1210 a$(e)=CHR$(y)+a$(e)
1230 PRINT "AENDERUNG BEENDET": CALL
   &BB18
1240 RETURN
1250 DATA HAUSHALT,UNTERHALTUNG,
   MIETEN und STEUERN,KLEIDUNG,AUTO,
   FERIEN,VERSCHIEDENES,EINKOMMEN
```

Spectrum

```
50 LET mn=200: IF PEEK 23733=127
   THEN LET mn=100
100 DIM c$(8,16): DIM a(mn): DIM
   a$(mn,23)
110 LET u=0: LET v=1
120 FOR n=v TO 8: READ c$(n):
   NEXT n
130 POKE 23658,8
140 LET k$=".00": FOR n=v TO 7:
   LET k$=k$+CHR$ 8: NEXT n
190 LET p=2: LET tt=u: LET cr=u
200 CLS : PRINT BRIGHT v; PAPER
   2; INK 6;AT 2,6;
   "□H□A□U□P□T
   □M□E□N□U□E"
210 PRINT BRIGHT v;PAPER 7;AT 5,6;
   "□1:-□DATEN EINGEBEN□"; AT 7,6;
   "□2:-□DATEN ABFRAGEN□";AT 9,6;
   "□3:-□SAVE AUF BAND□□";AT 11,
   6;"□4:-□LADEN VON BAND□";AT
   13,6;"□5:-□DRUCKER J/N
   □□□□";AT 15,6;"□6:-□EINTRAG
   AENDERN□";AT 17,6;"□7:-
   □PROGRAMM STOP□□"
220 PRINT AT 21,6;INK 3;FLASH v;PAPER 2;
   INK 7;"□-□WAEHLE OPTION.□-□"
230 IF INKEY$="" THEN GOTO 230
240 LET z$=INKEY$: IF z$<"1" OR
   z$>"7" THEN GOTO 230
250 CLS : GOSUB 1000*VAL z$
260 GOTO 200
1000 LET c=u
1005 LET c=c+v: IF c=mn+v THEN
   RETURN
1006 IF a$(c,v)="□" THEN GOTO
   1010
1007 GOTO 1005
1010 PRINT AT u,u;BRIGHT v;PAPER 2;INK 7;
   "□□DATUM□□□□□TEXT□□□
   □□□BETRAG□RUB"
1015 IF c=mn+v THEN RETURN
1020 INPUT"DATUM?□□□□□□";LINE
   a$(c,2 TO 9):IF a$(c,2)="□" THEN
   RETURN
1030 PRINT TAB u;a$(c,2 TO 9)
1040 INPUT "ARTIKEL?□□□□"; LINE
   a$(c,10 TO 23): IF a$(c,10)="□" THEN
   GOTO 1040
1050 PRINT TAB 9;a$(c,10 TO 21);
1060 INPUT "BETRAG□";a(c): IF
   a(c)=u THEN GOTO 1060
1070 LET vv=a(c)*100: LET v$=STR$
   vv: PRINT TAB 27-LEN v$;a(c);
1080 INPUT "RUBRIK□□□□"; LINE f$:
   IF f$="" THEN GOTO 1080
1090 FOR n=v TO 8: IF f$=c$(n,v
   TO LEN f$) THEN GOTO 1130
1100 NEXT n: GOTO 1080
1130 IF n=8 THEN LET cr=cr+a(c)
1140 IF n<>8 THEN LET tt=tt+a(c)
1150 PRINT TAB 29;c$(n,v TO 3)
1160 LET a$(c,v)=CHR$ (48+n)
1200 LET c=c+v: GOTO 1015
2000 FOR n=v TO 8: PRINT PAPER v;
   INK 7;AT n*2,6;"□";n;":-□";
   c$(n): NEXT n
2010 PRINT FLASH v; INK 2;AT 19,3;
   "□Waehle Rubrik (1 bis 8)□"
2020 IF INKEY$="" THEN GOTO 2020
2030 LET z$=INKEY$: IF z$<"1" OR
   z$>"8" THEN GOTO 2020
2040 LET t=u: LET c=u
2050 CLS : PRINT #p; PAPER 6;
   BRIGHT v;TAB 10;c$(VAL z$);
   TAB 31; "□"
2055 LET c=c+v: IF c=mn THEN GOTO
   2500
2060 IF a$(c,v)="□" THEN GOTO 2500
2070 IF a$(c,v)<>z$ THEN GOTO 2055
2080 PRINT #p;a$(c,2 TO 9); TAB 10;
   a$(c,10 TO 23);
2090 LET am=a(c)*100: LET n$=STR$
   am: PRINT #p;TAB 29;k$;TAB 31-
   LEN n$;a(c)
2100 LET t=t+a(c)
2110 GOTO 2055
2500 PRINT #p;TAB 25;
   "--------": LET tx=t*100:
   LET n$=STR$ tx: PRINT #p;TAB
   12;"TOTAL:-□";TAB 29;k$;TAB
   31-LEN n$;t
2510 IF z$<>"8" THEN GOTO 2590
2520 LET tz=tt*100: LET n$=STR$
   tz: PRINT '#p;"□□□
   GESAMTAUSGABEN:-□□";TAB 29;k$;
   TAB 31-LEN n$;tt
2530 LET ba=(t-tt)*100: LET n$=
   STR$ ba: PRINT '#p;TAB 10;
   "□BILANZ:-□";TAB 29;k$;TAB 31-
   LEN n$;ba/100
2590 PRINT PAPER 2; INK 7'
   "□□□□Weiter mit
   Tastendruck□□□□□"
2600 PAUSE u: IF PEEK 23560=13
   THEN RETURN
2610 CLS : GOTO 2000
3000 GOSUB 8000: IF re=v THEN
   RETURN
3010 PRINT PAPER 6;AT 10,u;"□GIB EINEN
   DATEINAMEN EIN□": INPUT LINE w$: IF
   LEN w$>10 OR LEN w$<v THEN GOTO
```



```
3010
3020 CLS : SAVE w$ DATA a(): SAVE
   w$ DATA a$(): RETURN
4000 GOSUB 8000: IF re=v THEN
   RETURN
4010 PRINT BRIGHT v;AT 10,u;"Name der zu
   speichernden Daten?": INPUT LINE w$: IF
   LEN w$>10 THEN GOTO 4010
4020 PRINT PAPER 3; INK 7;AT 10,u;
   "Cassette einlegen,""Play druecken□□"
4030 LOAD w$ DATA a(): LOAD w$
   DATA a$()
4040 LET cr=u: LET tt=u: FOR n=v
   TO mn: IF a$(n,v)="8" THEN LET
   cr=cr+a(n)
4050 IF a$(n,1)<>"8" THEN LET
   tt=tt+a(n)
4060 NEXT n: RETURN
5000 PRINT BRIGHT v;AT 10,u; "□Wollen
   Sie einen Ausdruck J/N?□"
5010 PAUSE u: IF INKEY$="" THEN
   GOTO 5010
5020 LET z$=INKEY$
5030 IF z$="N" THEN LET p=2:
   RETURN
5040 IF z$="J" THEN LET p=3:
   RETURN
5050 GOTO 5010
6000 LET c=v: IF a(c)=u THEN RETURN
6010 PRINT AT u,u; BRIGHT v;
   PAPER (VAL a$(c,v))-v; INK 9;
   "□Nummer□";c,c$(VAL a$(c,v))
6015 PRINT PAPER 2; INK 7;"'DATUM
   □□□□□□□BUCHUNGSTEXT
   □□□BETRAG": PRINT'a$(c,2 TO 9),
   TAB 10;a$ (c, 10 TO 23);
6020 LET am=a(c)*100: LET n$=STR$
   am: PRINT TAB 29;k$;TAB 31-LEN
   n$;a(c)
6030 PRINT PAPER 3; INK 7;AT 20,u;
   "□A□-□Vorwaerts□□□Q□-□
   Rueckwaerts□EDIT□-□
   Aufzeichnungsaenderung□□"
6040 PAUSE u
6050 IF INKEY$="Q" AND c>v THEN
   LET c=c-v: GOTO 6010
6060 IF INKEY$="A" AND c<>mn THEN
   LET c=c+v
6070 IF a(c)=u THEN LET c=c-v
6080 IF PEEK 23560=7 THEN GOTO 6100
6090 GOTO 6010
6100 INPUT BRIGHT v; "Neue Eingabe:
   Datum□"; LINE a$(c,2 TO 9): IF
   a$(c,2)="□" THEN GOTO 6100
6110 PRINT AT 5,u;a$(c,2 TO 9)
6120 INPUT BRIGHT v;"Neuer
   Buchungstext□"; LINE a$(c,10 TO 23): IF
   a$ (c,10)="□" THEN GOTO 6120
6130 PRINT AT 5,10;a$(c,10 TO 23)
6135 IF a$(c,v)="8" THEN LET
   cr=cr-a(c)
6136 IF a$(c,v)<>"8" THEN LET
   tt=tt-a(c)
6140 INPUT BRIGHT v;"Neuer Betrag:
   □□□□□";a(c): IF a(c)=u THEN
   GOTO 6140
6150 LET am=a(c)*100: LET n$=STR$
   am: PRINT AT 5,29;k$;TAB 31-LEN
   n$;a(c)
6160 INPUT BRIGHT v;"Neue
   Kategorie:□"; LINE f$: IF f$=""
   THEN GOTO 6160
6170 FOR n=v TO 8: IF f$=c$(n,v
   TO LEN f$) THEN GOTO 6190
6180 NEXT n: GOTO 6160
6190 LET a$(c,v)=CHR$ (48+n)
6200 IF n=8 THEN LET cr=cr+a(c)
6210 IF n<8 THEN LET tt=tt+a(c)
6220 RETURN
7000 GOSUB 8000: IF re=v
   THEN RETURN
7010 RANDOMIZE USR u
8000 PRINT PAPER 4;AT 10,9;"□Bist du
   sicher?□"
8010 PAUSE u: LET re=u: IF INKEY$
   <>"J" THEN LET re=v
8020 RETURN
9000 DATA "HAUSHALT","FREIZEIT",
   "MIETE UND STEUERN","KLEIDUNG",
   "AUTOKOSTEN","FERIEN","SONSTIGES",
   "EINKOMMEN"
```

Commodore

```
10 DIM DS(1,400), D(1,400): CO=0: POKE
   53280,0: POKE 53281,0
20 DIM N(8), RS(8): FOR I=1 TO 8: READ
   RS(I): NEXT
30 DATA HAUSHALT, FREIZEIT, MIETE UND
   STEUERN, KLEIDUNG, AUTO
40 DATA URLAUB, VERSCHIEDENES,
   EINKOMMEN
50 CO=0: CRS=CHR$(13): DWS="■■
   ■□DEINE□WAHL□?□"
99 REM****** HAUPTOROGRAMM ************
100 PRINT "■■■■■"TAB(10)"■□
   □□□HAUPTMENUE□□□□■"
102 PRINT TAB(10) "■1. □DATEN
   EINGEBEN"
104 PRINT TAB(10) "■2.□DATEN
   ABFRAGEN"
106 PRINT TAB(10) "■3.□DATEN
   SPEICHERN"
108 PRINT TAB(10) "■4.□DATEN LADEN"
110 PRINT TAB(10) "■5.□DRUCKEN
   JA/NEIN"
112 PRINT TAB(10) "■6.□DATEN
   AENDERN"
114 PRINT TAB(10) "■7.□PROGRAMM
   BEENDEN": PRINT TAB(12) DWS
120 GET AS:K=VAL(A$): IF K<1 OR K>7
   THEN 120
130 ON K GOTO 800,200,300,400,500,600,
   700
199 REM****** 2. DATEN ABFRAGEN *******
200 GOSUB 950: IF AS=CR$ THEN 100
210 IF DF=1 THEN OPEN 4,4: CMD 4
220 GOSUB 1000: PRINT "■ ------
   ----------
   ----------
   ----GES.:□◆DM";
230 A=n(R): TA=12: GOSUB 1230:
   PRINT"■ ----------
   ----------
   ----------"
240 IF R<>8 THEN 260
250 PRINT TAB(7) "■GESAMTAUSGABEN
   □□:◆DM";: A=FR: TA=12: GOSYB
   1230: A=N(R)-FR
260 IF R=8 THEN PRINT TAB(15) "■
   ■BILANZ□□:◆DM";: GOSUB 1230
270 IF DF=1 THEN PRINT#4,CR$: CLOSE 4
280 GOSUB 900: IF A$=CR$ THEN 100
290 GOTO 200
299 REM****** 3. DATEM SPEICHERN ******
300 GOSUB 450: OPEN 1,1,1,A$:
   PRINT#1,CO: FOR I=1 TO CO
310 PRINT#1,D$(0,I)+CR$+D$((1,I)+
   CR$+STR$(D(0,I))+CR$+STR$(D(1,I))
320 NEXT I: CLOSE 1: GOTO 100
399 REM****** 4. DATEN LADEN **********
400 GOSUB 450: OPEN 1,1,0,A$:
   INPUT#1,CO: FOR I=1 TO CO
410 INPUT#1,D$(0,I),D$(1,I), D(0,I),D(1,I):
   NEXT I: CLOSE 1: GOTO 100
449 REM****** FILENAMEN EINGABE
450 INPUT "■■■■ GIB DEN
   DATEINAMEN EIN: ◆";A$: RETURN
499 REM****** 5. DRUCKEROPTION ********
500 PRINT "■■ MOECHTEST DU EINEN
   AUSDRUCK (J/N) ?"
510 GET A$: IF A$<>"J" AND
   A$<>"N" THEN 510
520 DF=0: IF A$="J" THEN DF=1
530 RETURN
599 REM****** 6. DATEN AENDERN ********
600 IF CO=0 THEN 100
610 CQ=1: GOTO 1360
699 REM****** 7. ENDE *****************
700 PRINT "■■■*** BIST DU SICHER
   (J/N) ? ***"
710 WAIT 198m 255: GET A$: IF A$<>"J"
   THEN 100
720 PRINT "■■": END
799 REM****** 1. DATEN EINGEBEN ********
800 IF CO=400 THEN PRINT TAB(10)
   "■■□SPEICHER VOLL !! ": GOSUB
   900: GOTO 100
810 CO=CO+1: D$(0,CO)="": C1=CO:
   PRINT"■■■■"
820 IF K<>6 THEN PRINT "□□MIT
```


```
   'RETURN' IM DATUM-FELD ZUM
   MENUE"
830 INPUT "■■■■ DATUM:◆";D$(Ø,
   C1): IF D$(Ø,C1)="" THEN
   CO=CO-1: GOTO 1ØØ
84Ø INPUT "■■ TEXT:◆";D$(1,C1)
85Ø INPUT "■■ BETRAG:◆"D(Ø,C1):
   GOSUB95Ø: D(1,C1)=R: IF
   A$=CHR$(13) THEN 6Ø
86Ø IF K=6 THEN 136Ø
87Ø GOTO 8ØØ
899 REM***** WARTEN
9ØØ PRINT "■■↑WEITER MIT TASTE;
   MIT ¡RETURN¡ ZUM MENUE";
91Ø WAIT 198,225: GET A$: RETURN
949 REM***** RUBRIK ANGEBEN
95Ø PRINT"◆■■■■□□□■□
   R□U□B□R□I□K□"
96Ø FOR I=1 TO 8: PRINT
   "□□□■";I;"▮▮.□"+R$(I): NEXT:
   PRINT "□□□"+DW$
97Ø WAIT 198,255: GET A$: R=VAL(A$)
98Ø IF (R<1 OR R>8) AND A$<>CR$
   THEN 97Ø
999 IF K=1 AND A$=CR$ THEN 97Ø
995 RETURN
999 REM***** TABELLE AUSGEBEN
1ØØØ C1=Ø: PRINT "◆■": IF K=6 THEN
   PRINT TAB(12) "■EINTRAG NR.:"CQ
1Ø1Ø PRINT TAB(19-LEN(R$(R))/2) R$(R):
   N(R)=Ø
1Ø2Ø PRINT "■ - - - - - - - - - - -
   - - - - - - - - - - - - - - - -
   - - - - - - - - - - - - - - - -"
1Ø3Ø PRINT
   "□□DATUM□□◫
   □□□□□□□ TEXT □□
   □□□□□◫□□BETRAG"
1Ø4Ø PRINT "- - - - - - - - - - - -
   - - - - - - - - - - - - - - - -
   - - - - - - - - - - - - - - - -
   ◆":SC=Ø
1Ø5Ø IF K=6 THEN C1=CQ: GOSUB 12ØØ:
   GOTO 13ØØ
1Ø6Ø C1=C1+1: IF D(1,C1)=R THEN
   GOSUB 12ØØ: IF DF=Ø THEN
   SC=SC+1
1Ø7Ø IF SC>=1Ø THEN SC=Ø: GOSUB
   128Ø: PRINT "■■■■■■"
1Ø8Ø IF C1=4ØØ OR D(Ø,C1)=Ø THEN
   FR= Ø: GOTO 11ØØ
1Ø9Ø GOTO 1Ø5Ø
1Ø99 REM***** AUSGABEN AUFSUMMIEREN
11ØØ FOR I=1 TO CO: IF D(1,I)<>8 THEN
   FR=FR+D(Ø,I)
111Ø NEXT I: RETURN
1199 REM***** 3 SPALTEN AUSGEBEN
12ØØ PRINT LEFT$(D$(Ø,C1)+"□□□□
   □□□□□□",9+"■◫◆";
1211 PRINT LEFT$(D$(1,C1)+"□□□□
   □□□□□□□□□□□□□
   □",18)+"■◫◆DM";
122Ø A=D(Ø,C1): TA=8: N(R)=n(R)+A
123Ø A$=STR$(A): IF A-INT(A)=Ø THEN
   A$=A$+".ØØ"
124Ø IF MID$(A3, LEN(A$)-1,1)="."
   THEN A$=A$+"Ø"
126Ø PRINT RIGHT$("□□□□□□□□□
   □□□□□□□□□□□□"+A$,TA):
   RETURN
1279 REM***** 14 ZEILEN LOESCHEN
128Ø GOSUB 9ØØ: IF A$=CR$ THEN 1ØØ
129Ø PRINT "■■■■■■■": FOR I=1
   TO 14
1295 PRINT "□□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□";:
   NEXT I: RETURN
1299 REM***** Wahl BEI (6)
13ØØ PRINT "■↑ (,) - ZURUECK",
   "(.)-VORWAERTS"+CR$+"■
   (LEERTASTE) - AENDERN"
131Ø WAIT 198,255: GET A$: IF A$=CR$
   THEN 1ØØ
132Ø IF A$="□" THEN 82Ø
133Ø IF A$="." THEN CQ=CQ+1: GOTO
   136Ø
134Ø IF AS<>"," THEN 131Ø
135Ø CQ=CQ-1: IF CQ<1 THEN CQ=CO
136Ø IF CQ<CO THEN CQ=1
137Ø R=D(1,CQ): GOTO 1ØØØ
```

Variablenbeschreibung:
A$—Hilfs- und Eingabevariable
CO—Anzahl an eingegebenen Datensätzen
D(Ø,i)—Betrag für Datensatz i.Ø<i<4ØØ
D(1,i)—Nummer der Rubrik
D$(Ø,i)—Datum für Datensatz i.Ø<i<4ØØ
D$(1,i)—Text für jeweilige Eingabe
K—Nummer des Menüpunkts
N(8)—Summe der Einzahlungen je Rubrik
R—Nummer der gewählten Rubrik
R$(8)—Name der jeweiligen Rubrik

Atari

```
1ØØ OPEN #5,4,Ø,"K":POKE 82,1
11Ø GOSUB 168Ø
12Ø PRINT CHR$(125);L$;" Hauptmenue":
   PRINT :POKE 71Ø,192:CH=Ø
13Ø PRINT L$;"1...Daten eingeben"
```

```
140 PRINT L$;"2...Daten abfragen"
    150 PRINT L$;"3...Daten speichern"
160 PRINT L$;"4...Daten laden"
170 PRINT L$;"5...Drucken Ja/Nein"
180 PRINT L$;"6...Daten aendern"
190 PRINT L$;"7...Programm beenden"
200 PRINT :PRINT "Deine Wahl:":
    S=1:GOSUB 430
210 GET #5,T:T=T-48:ON T GOTO 470,
    650,970,1070,1160,1230,1650:GOTO 210
220 POSITION 2,18:FOR T=1 TO 4:PRINT
    CHR$(156);:NEXT T:RETURN
230 GOSUB 220:PRINT "Deine Eingabe ";:
    INPUT W$:RETURN
240 X=PEEK(85):Y=PEEK(84):FOR Q=1
    TO 9:PRINT CHR$(156);:NEXT Q:
    RESTORE 1800
250 POSITION 1,Y:FOR Q=1 TO 8:READ W$:
    PRINT Q;" - ";W$:NEXT Q:PRINT
260 PRINT "Welche Rubrik ?";
270 GET #5,K:KA=K-48:IF KA<1 OR
    KA>8 THEN 270
280 POSITION 1,Y:FOR Q=1 TO 10:PRINT
    CHR$(156);:NEXT Q:POSITION X,Y
290 RESTORE 1800:FOR Q=1 TO KA:READ
    W$:NEXT Q:RETURN
300 GOSUB 220:PRINT "FEHLER - ";PEEK
    (195);" IN ZEILE ";PEEK(186)+256*PEEK
    (187)
310 S=11:GOSUB 420:GOTO 440
320 X$="□□□□□□□.00":BX=INT
    (BX*100+0.5)/100
330 X$(6)=STR$(BX-INT(BX))
340 L$=STR$(INT(BX)):X$(7-LEN(L$),
    7)=L$:X$(10)="":GOTO 410
350 X=PEEK(85)-1:Y=PEEK(84):GOSUB
    220:PRINT "Deine Eingabe ?";:POSITION
    X,Y:I=0
360 ? CHR$(31);:IF PEEK(93)=0 THEN
    X=X+1:POKE 85,X:IF X<17 THEN 360
370 PRINT CHR$(31);:Q=(PEEK(93)=0)
380 IF I AND Q THEN POKE 85,X
390 I=Q:ON PEEK(764)=255 GOTO 370
400 POSITION 15,18:INPUT W$:RETURN
410 L$="□":L$(10)="□":L$(2)=L$:
    RETURN
420 FOR L=1 TO 2:GOSUB 430:NEXT L
430 FOR T=15 TO 0 STEP -0.6:SOUND
    0,0+S*10,10,T:SOUND 1,1+S*10,10,T:
    NEXT T:RETURN
440 PRINT "Weiter mit RETURN";:GET
    #5,T:GOTO 120
450 POSITION 4,8:PRINT "Eingeben:":PRINT
    "Geraet:Dateiname":INPUT W$:IF LEN
    (W$) THEN RETURN
460 POP :GOTO 120
470 IF LEN(DAT$)+60>SP THEN GOSUB
    220:PRINT "Speicher voll":S=25:GOSUB
    420:GOTO 120
480 PRINT CHR$(125):PRINT "Eingabe":
    R$="□":R$(50)=R$:R$(2)=R$
490 POSITION 2,4:PRINT "Datum
    □□□□□□:"
500 GOSUB 230:IF W$="" THEN 120
510 R$(2,9)=W$:POSITION 14,4:
    PRINT R$(2,9)
520 POSITION 2,6:PRINT "Text
    □□□□□□□:"
530 GOSUB 230:R$(17)=W$:POSITION
    14,6:? R$(17)
540 POSITION 2,8:PRINT "Betrag
    □□□□□:"
550 GOSUB 230:TRAP 550:BX=VAL(W$):
    TRAP 32767
560 IF BX>9999.99 OR BX<0 THEN 550
570 GOSUB 320:R$(10,16)=X$(3)
580 POSITION 14,8:PRINT R$(10,16)
590 POSITION 2,10:GOSUB 240:PRINT
    "Rubrik□□□□:";W$:R$(1,1)=CHR$
    (KA)
600 GOSUB 220:PRINT "Ist die Eingabe
    richtig? (J/N)";:S=1:GOSUB 430: GET
    #5,K
620 IF K<>74 AND K<>106 THEN
    GOSUB 220:PRINT "DATEN NICHT
    GESPEICHERT!":S=25:GOSUB 420:GOTO
    470
620 DAT$(LEN(DAT$)+1)=CHR$(LEN
    (R$)):DAT$(LEN(DAT$)+1)=R$
630 IF KA=8 THEN EIN=EIN+BX:GOTO
    470
640 AUS=AUS+BX:GOTO 470
650 IF DAT$="" THEN GOSUB 220:
    PRINT "Keine Daten vorhanden!":S=25:
    GOSUB 420:GOTO 120
660 PRINT CHR$(125):GOSUB 240
670 W$="E":IF D THEN W$="P":TRAP
    1210
680 CLOSE #1:OPEN #1,8,0,W$:TRAP
    32767:POKE 710,192
690 N=1:SUM=0
700 RESTORE 1800:FOR I=1 TO KA:READ
    W$:NEXT I
    710 PRINT #1:PRINT #1;W$:W$(1,1)=
    "-":W$(2)=W$:PRINT #1;W$:PRINT
    #1
720 W$="□":W$(100)="□":W$(2)=
    W$:W$="Datum":W$(11)="Text":W$
    (32+25*D)="Betrag":PRINT #1;W$
730 W$(1,1)="-":W$(2)=W$:PRINT #1;
    W$
740 TRAP 860:X=ASC(DAT$(N))+1:KS=
    ASC(DAT$(N+1)):TRAP 32767
750 IF KS<>KA THEN 840
760 R$=DAT$(N+1,N+X-1)
770 IF NOT D AND PEEK(84)>18 THEN
    POSITION 1,4:PRINT
    CHR$(156);:POSITION 1,18
780 W$="□":W$(100)=W$:W$(2)=W$
790 W$=R$(2,9):W$(11,29+25*D)=R$
    (17)
800 W$(32+25*D)=R$(10,16):PRINT
    #1;W$
810 SUM=SUM+VAL(R$(10,16))
```

```
820 IF PEEK(764)=255 THEN 840
830 GET #5,K:ON (K=27 OR K=155)
    GOTO 1200:GET #5,K
840 N=N+X
850 GOTO 740
860 IF PEEK(195)<>5 THEN CLOSE
    #1:GOTO 300
870 IF NOT SUM THEN PRINT #1;" KEINE
    DATEN!":GOTO 950
880 W$(1,1)="=":W$(2)=W$:PRINT
    #1;W$
890 W$(1,1)="□":W$(2)=W$:W$(23+
    25*D)="Summe:"
900 BX=SUM:GOSUB 320:W$(30+25*
    D)=X$
910 PRINT #1;W$:PRINT #1:PRINT #1
920 IF KA<>8 THEN 950
    930 BX=AUS:GOSUB 320:PRINT #1;
    "□□□Deine Gesamtausgaben□□□
    DM ";X$
940 BX=EIN-AUS:GOSUB 320:PRINT #1;
    "□□□Die Bilanz ist□□□□□□□□
    □□DM ";X$
950 CLOSE #1:IF NOT D THEN PRINT :GOTO
    440
960 GOTO 120
970 POKE 710,16:IF DAT$="" THEN 650
980 PRINT CHR$(125):PRINT L$;"Daten
    speichern"
990 GOSUB 450:IF W$(1,1)="D" THEN
    TRAP 1010:CLOSE #1:OPEN #1,4,0,W$:
    TRAP 32767
1000 PRINT "Datei ueberschreiben?
    (J/N)":S=25:GOSUB 430:GET #5,K:IF
    K<>74 AND K<>106 THEN 990
1010 TRAP 1060:CLOSE #1:OPEN
    #1,8,0,W$:T=1
1020 PRINT "Schreibe Datei ";CHR$(34);W$
    CHR$(34)
1030 PRINT #1;EIN:PRINT #1;AUS
1040 PRINT #1;DAT$:CLOSE #1
1050 TRAP 32767:GOTO 120
1060 PRINT :PRINT :GOSUB 420:CLOSE
    #1:PRINT "ERROR- ";PEEK(195);
    " beim Speichern ";W$:GOTO 440
1070 POKE 710,16:PRINT CHR$(125):PRINT
    L$;"Daten laden":S=20:GOSUB 430
1080 GOSUB 450:TRAP 1150:CLOSE
    #1:OPEN #1,4,0,W$
1090 PRINT :PRINT "Lade ";W$:POKE
    710,210
1100 INPUT #1,EIN:INPUT #1,AUS
1110  L=USR(ADR(ML$),16,7,ADR(DAT$),
    SP):IF NOT L THEN POKE 195,163:
    GOTO 1150
1120 IF L=SP THEN TRAP 1130:GET
    #1,A:TRAP 1070:GOTO 1140
1130 DAT$(L)="":S=9:GOSUB 430:
    GOTO 120
1140 GOSUB 220:PRINT "Speicher voll!":?
    "Datei unvollstaendig geladen!":S=25:
    GOSUB 420:GOTO 440
1150 POSITION 2,20:PRINT "ERROR- ";
    PEEK(195);" beim Laden ";W$:S=25:
    GOSUB 420:GOTO 440
1160 GOSUB 220:PRINT "Drucken ? (J/N) ";:
    GOSUB 430
1170 GET #5,K:D=(K=74)+(K=106)
1180 GOSUB 220:IF NOT D THEN PRINT
    "Drucker ausgeschaltet":GOSUB 420:GOTO
    120
1190 TRAP 1210:CLOSE #1:OPEN
    #1,8,0,"P":CLOSE #1
1200 GOSUB 220:PRINT "Drucker
    eingeschaltet":GOSUB 420: GOTO 120
1210 GOSUB 220:PRINT "... und wo ist der
    Drucker???":S=27:GOSUB 420
1220 FOR T=1 TO 100:NEXT T:D=0:
    GOTO 1180
1230 IF DAT$="" THEN GOSUB 220:PRINT
    "Keine Daten vorhanden!":S=25:GOSUB
    420:GOTO 120
1240 N=1:DS=0:DS1=N
1250 TRAP 1390:OFF=ASC(DAT$(N))+1:
    TRAP 32767
1260 R$=DAT$(N+1,N+OFF-1):DS=
    DS+1:IF DS<DS1 THEN N=N+OFF:
    GOTO 1250
1270 PRINT CHR$(125):PRINT "□Datensatz
    □";DS:POSITION 2,4:PRINT "Datum□□
    □□□:□";R$(2,9)
1280 POSITION 2,6:PRINT "Text□□□□□□
    □:□";:W$=R$(17):IF LEN(W$)>20
    THEN W$(21)=""
1290 ? W$
1300 POSITION 2,8:PRINT "Betrag□□□□:
    □";R$(10,16)
1310 POSITION 2,10:PRINT "Rubrik□□□
    □:□";
1320 KA=ASC(DAT$(N+1)):GOSUB 290:
    PRINT W$
1330 GOSUB 220:PRINT "V(orwaerts)
    Z(urueck) A(endern)":PRINT " [RETURN]
    - Menue"
1340 GET #5,K:IF K=155 THEN 120
1350 IF K=65 THEN 1410
1360 DS1=DS1+(K=86)-(K=90):IF
    DS1<1 THEN DS1=1
1370 ON DS=DS1 GOTO 1340:N=N+OFF:
    IF DS1<DS THEN N=1:DS=0
1380 GOTO 1250
1390 IF PEEK(195)<>5 THEN 300
1400 N=N-OFF:DS1=DS1-1:GOTO 1340
1410 RL=LEN(R$):POSITION 14,4:GOSUB
    350
1420 IF W$<>"" THEN R$(2,9)=L$:R$
    (2,9)=W$:POSITION 14,4:? R$(2,9)
1430 POSITION 14,6:GOSUB 350:IF W$=""
    THEN 1460
1440 IF LEN(W$)>30 THEN W$(31)=""
1450 R$(17)=W$:POSITION 14,6:PRINT L$;
    L$:POSITION 14,6:PRINT R$(17)
1460 POSITION 14,8:GOSUB 350:IF W$=""
    THEN 1500
1470 TRAP 1460:BX=VAL(W$):TRAP
    32767:IF BX<0 OR BX>9999.99 THEN
    1460
1480 GOSUB 320:CH=VAL(R$(10,16)):R$
    (10,16)=X$(3):BX=VAL(W$):POSITION
    14,8:PRINT R$(10,16)
1490 GOSUB 220:? R$:? W$:GET #5,K
1500 GOSUB 220:PRINT "Rubrik aendern
    (J/N) ";:GET #5,K:IF K<>74 AND
    K<>106 THEN 1520
1510 POSITION 2,10:GOSUB 240:R$(1,1)=
    CHR$(KA):PRINT "Rubrik□□□□□:□";
    W$
1520 IF NOT CH THEN 1550
1530 KA=ASC(R$):W$=R$(10,16):IF
    KA=8 THEN EIN=EIN-CH+BX:
    GOTO 1550
1540 AUS=AUS-CH+BX
1550 IF LEN(R$)=RL THEN DAT$(N+1,
    N+OFF-1)=R$:GOTO 120
1560 DAT$(N,N)=CHR$(LEN(R$))
1570 IF LEN(R$)<RL THEN DAT$(N+1,N+
    LEN(R$))=R$:DAT$(N+LEN(R$)+1)=
    DAT$(N+OFF):GOTO 120
1580 GOSUB 220:? "MOMENT BITTE..."
1590 X=LEN(R$)-RL:Y=LEN(DAT$):IF
    X+Y>=SP THEN GOSUB 220:?
    "SPEICHER VOLL!":S=25:GOSUB 420:
    GOTO 120
1600 FOR I=Y TO N+OFF STEP -1
1610 DAT$(I+X,I+X)=DAT$(I,I)
1620 NEXT I
1630 DAT$(N+1,N+LEN(R$))=R$
1640 GOTO 120
1650 PRINT CHR$(125):POKE 710,16:PRINT
    " M(enue) / B(ASIC) ?";:GET$5,T:IF
    T<>66 THEN 120
1660 POKE 82,2:GRAPHICS 0:END
1670 REM Initialisierung
1680 DIM W$(100),ML$(57),R$(50),L$(10),
    X$(10)
1690 SP=FRE(0)-100:DIM DAT$(SP)
1700 RESTORE 1720:FOR I=1 TO 57:READ
    P:ML$(I)=CHR$(P):NEXT I
1710 GOTO 410
1720 DATA 104,104,104,170,104,104,157
1730 DATA 66,3,104,157,69,3,104,157
1740 DATA 68,3,104,157,73,3,104,157
1750 DATA 72,3,32,86,228,48,11,189,72
1760 DATA 3,133,212,189,73,3,133,213
1770 DATA 96,169,0,133,212,133,213
1780 DATA 189,67,3,201,136,240,232
1790 DATA 133,195,96
1800 DATA Haushalt,Unterhaltung,Mieten und
    Steuern,Kleidung,Auto,Ferien,Verschiedenes,
    Einkommen
```

Handelsbilanz

In diesem Artikel verabschieden wir uns von der Neuen Welt und segeln nach Hause, um die neu erworbenen Waren zu verkaufen.

Vor ihrer Heimreise besucht ein Rivale des Häuptlings heimlich Ihr Schiff. Der Rebell will Waffen kaufen, um den Häuptling zu unterwerfen, und bietet 30 Perlen für jede Waffe. Wenn Sie dem Handel zustimmen, machen Sie einen erheblichen Gewinn. Doch sollte der Häuptling bei einem Scheitern des Putsches feststellen, daß Sie die Waffen verkauft haben, wird er sich rächen.

Selbstverständlich können Sie das Angebot ablehnen. In diesem Fall erweist sich der Häuptling dankbar und stattet Sie mit Vorräten für die Heimfahrt aus, einschließlich weiterer 50 Perlen als Belohnung. Obwohl der Putsch Aussicht auf Erfolg hat, sind die Folgen bei einem Mißlingen schrecklich. Der Häuptling gibt den Befehl, Ihr Schiff zu vernichten. Harte Zeiten kommen auf Sie zu.

Können Sie diese Katastrophe verhindern, setzt das Schiff Segel Richtung Heimat. Die Mannschaft ist in guter Verfassung, und der Wind steht günstig, so daß die Reise nur acht Wochen dauert. Wenn das Schiff den Hafen erreicht, können die Waren verkauft, die Mannschaft bezahlt und, mit Glück, Gewinne erzielt werden. Reicht das Geld nicht zum Bezahlen der Mannschaft aus, können Sie das Schiff verkaufen. Schließlich erstellt das Programm einen Bericht über die Fahrt.

Das Hauptprogramm ruft in Zeile 893 eine Unterroutine (Zeile 10300) auf, die das Angebot zum Waffenhandel beinhaltet. Zuerst überprüft das Programm, ob überhaupt Waffen an Bord sind. Hierzu wird in Zeile 10305 das zweite Element des Vorrats-Arrays, OA(2), untersucht, in dem die Anzahl der Waffen gespeichert ist. Ist der Wert 0, erfolgt der Rücksprung zum Hauptprogramm. Sind Waffen verfügbar, macht der Rebell sein Angebot. Sie werden nun durch Zeile 10326 aufgefordert, Y oder N einzugeben.

Wenn Sie sich für den Waffenhandel entscheiden, verzweigt die Routine zu Zeile 10400. Es besteht eine 75prozentige Chance, daß das Programm zu Zeile 10450 verzweigt (Scheitern des Aufstands). Wenn Sie wollen, können Sie diese Chancen entsprechend Ihren Ansichten über Fair play ändern.

Ist die Revolte vereitelt, holt sich der Häuptling die gehandelten Waren wieder vom Schiff und setzt es in Brand. Andernfalls wird der Programmlauf mit Zeile 10429 fortgesetzt, und der neu ernannte Häuptling versorgt das Schiff vor der Abfahrt mit Proviant.

Die Gleichung in Zeile 10429 berechnet die Anzahl an Perlen, die Sie für die Waffen erhalten. Hierzu wird die Waffenanzahl OA(2) mit 30 multipliziert. Der ermittelte Wert wird zum ersten Element von AO() addiert. In Zeile 10430 werden die Waffen aus dem Vorrats-Array gelöscht. Für den Rest des Programms ist das nicht unbedingt notwendig, doch es erleichtert eine Erweiterung des Spiels, etwa für Zwischenfälle auf der Heimreise.

Perlen als Belohnung

Wenn Sie das Angebot des Rebellen nicht annehmen, belohnt Sie der amtierende Häuptling vielleicht mit 50 Perlen. Darüber entscheidet der Zufall in Zeile 10345. Die letzte Unterroutine beginnt bei Zeile 10500 und handhabt die Heimreise und den Warenverkauf.

In Zeile 10512 wird die Variable WW angelegt, in der die Gehaltsabrechnung für die Heimreise gespeichert wird. Der Wert wird in der Schleife ab Zeile 10514 berechnet. Die Formel multipliziert die Anzahl jedes Mannschaftstypes, CC(T), mit dem jeweils entsprechenden Wochenlohn WG(T). Der Schleifenzähler T gewährleistet, daß dieser Vorgang für jeden Mannschaftstyp wiederholt wird. Anschließend wird der Gesamtwochenlohn berechnet. Zur Kalkulation des Gesamtlohnes für die Heimreise wird dieser neue Gesamtwert mit acht multipliziert.

Die Marktpreise der erhandelten Güter werden durch das Array V2(3) angegeben und in Zeile 62 DIMensioniert. Die einzelnen Preise werden bei Programmstart per Zufall bestimmt. Mittels der Zeilen 10532 bis 10536 werden die

BASIC-Dialekte

Spectrum:
Ersetzen Sie im gesamten Programm AO() durch E(), V1() durch B() und V2() durch D(). Ändern Sie außerdem:

```
10310 CLS:GO SUB 9200
10328 INPUT I$:LET I$=I$( TO 1)
10354 LET I$=INKEY$:IF I$="" THEN GO TO
   10354
10400 CLS:GO SUB 9200
10501 CLS:GO SUB 9200
10547 LET I$=INKEY$:IF I$="" THEN GO TO
   10547
```

Acorn B:
Ändern Sie das Programm wie folgt:

```
10310 CLS:GO SUB 9200
10354 I$=GET$
10400 CLS:GOSUB 9200
10501 CLS:GOSUB 9200
10547 I$=GET$
```

Modul Dreizehn: Die Reise endet

Ergänzungen am Hauptprogramm

```
893 GOSUB10300
894 GOSUB10500
```

Putsch-Routine

```
10300 REM REVOLUTION
10305 IFOA(2)=0THENRETURN
10310 PRINTCHR$(147):GOSUB9200
10315 S$="DURING THE NIGHT A RIVAL OF THE*":GOSUB9
100
10316 S$="CHIEF VISITS THE SHIP IN SECRET*":GOSUB9
100
10317 PRINT:GOSUB9200
10318 S$="HE WANTS TO BUY YOUR GUNS FOR*":GOSUB910
0
10320 S$="A REVOLUTION*":GOSUB9100
10322 PRINT:GOSUB9200
10324 S$="HE OFFERS YOU 30 PEARLS PER GUN*":GOSUB9
100
10326 S$="DO YOU SELL HIM THE GUNS?(Y/N)*":GOSUB91
00
10328 INPUTI$:I$=LEFT$(I$,1)
10330 IFI$<>"N"ANDI$<>"Y"THEN10328
10332 IFI$="Y"THEN10400
10334 PRINT:GOSUB9200
10336 S$="THE CHIEF FINDS OUT AND IS*":GOSUB9100
10338 S$="PLEASED WITH YOU*":GOSUB9100
10340 S$="HE GIVES YOU FREE PROVISIONS*":GOSUB9100
10342 S$="FOR THE JOURNEY BACK*":GOSUB9100
10344 GOSUB9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="AND 50 PEARLS!!*":GOSUB9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB9200
10352 S$=K$:GOSUB9100
10354 GETI$:IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147):GOSUB9200
10405 IFRND(1)<.75THEN10450
10410 S$="THE REVOLUTION IS SUCCESSFUL*":GOSUB9100
10412 PRINT:GOSUB9200
10415 S$="THE NEW CHIEF REWARDS YOU WITH*":GOSUB91
00
10420 S$="FREE PROVISIONS FOR THE RETURN*":GOSUB91
00
10425 S$="JOURNEY.*":GOSUB9100
10429 AO(1)=AO(1)+(OA(2)*30):REM ADD PEARLS
10430 OA(2)=0
10431 GOTO10350
10450 S$="THE REVOLUTION FAILS!!*":GOSUB9100
10452 PRINT:GOSUB9200
10455 S$="THE OLD CHIEF IS ANGRY WITH YOU*":GOSUB9
100
10457 S$="HE BURNS YOUR SHIP AND STEALS*":GOSUB910
0
10458 S$="EVERYTHING!!*":GOSUB9100
10459 PRINT:GOSUB9200
10460 S$="   GAME OVER!!*":GOSUB9100
10462 END
10464 GOTO10462
```

Ende-der-Reise Routine

```
10500 REM END OF VOYAGE
10501 PRINTCHR$(147):GOSUB9200
10505 S$="WITH A STRONG CREW AND GOOD*":GOSUB9100
10507 S$="WINDS THE JOURNEY BACK GOES*":GOSUB9100
10508 S$="WELL AND TAKES ONLY 8 WEEKS*":GOSUB9100
10512 WW=0
10514 FORT=1TO5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB9200
10520 S$="WAGE BILL FOR RETURN VOYAGE = *":GOSUB91
00
10522 PRINTWW;"GOLD PIECES"
10524 PRINT:GOSUB9200
10526 S$="WHEN YOU GET BACK*":GOSUB9100
10528 S$="YOU SELL YOUR TRADING GOODS*":GOSUB9100
10530 S$="THE GOODS ARE NOW WORTH:*":GOSUB9100
10532 PRINT"PEARLS - ";V2(1);"GOLD PIECES"
10534 PRINT"CARVINGS - ";V2(2);"GOLD PIECES"
10536 PRINT"SPICES - ";V2(3);"GOLD PIECES"
10538 PRINT:S$="YOU GET A TOTAL OF*":GOSUB9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINTX;"GOLD PIECES"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="YOU NOW HAVE:*":GOSUB9100
10552 PRINTMO+X;"GOLD PIECES"
10555 PRINT:GOSUB9200
10556 S$="THE WAGE BILL FOR THE VOYAGE IS*":GOSUB9
100
10557 PRINTWT+WW;"GOLD PIECES"
10559 PRINT:GOSUB9200
10560 S$="YOU END THE GAME WITH:*":GOSUB9100
10562 Z=MO+X-WT-WW
10565 PRINTZ;"GOLD PIECES"
10566 PRINT:GOSUB9200
10567 PRINT:S$="YOUR RATING IS:*":GOSUB9100:PRINT
10568 IF Z>3200 THEN S$="ARCH CAPITALIST*":GOSUB 9
100:END
10569 IF Z>2500 THEN S$="MASTER TRADER*"GOSUB9100:
END
10570 IF Z>2000 THEN S$="MERCHANT CLASS III*":GOSU
B9100:END
10571 IF Z>1000 THEN S$="A PROPER JONAH*":GOSUB 91
00:END
10572 S$="MORE OF A DUCK THAN A DRAKE*"
10573 GOSUB9100:END
```

in diesem Array gespeicherten Werte für jede gehandelte Ware dargestellt.

Der Gesamterlös an Gold wird in Zeile 10540 errechnet und in X gespeichert. Die verwendete Formel multipliziert die Menge jedes Handelsgutes, verzeichnet in AO(), mit dem aktuellen in V2() gespeicherten Marktpreis. Die Einzelerlöse aus Perlen, Schnitzereien und Gewürzen werden addiert, und anschließend wird das Gesamtergebnis dargestellt.

Es ist durchaus möglich, daß das Startkapital von 2000 Goldstücken bei Reisebeginn nicht vollständig ausgegeben wurde. Jegliches Restguthaben ist in der Variablen MO gespeichert und am Ende zu dem durch Handel erhaltenen Gold addiert.

Die Lohnabrechnung für die komplette Fahrt findet in Zeile 10557 statt, indem die Löhne für die Hinreise (WI) mit den Löhnen für die Heimreise addiert werden. Danach errechnet das Programm den Handelsprofit, indem es die Gesamtlöhne vom Kapital subtrahiert. Abschließend wird angegeben, wie erfolgreich Sie waren, indem eine Beurteilung, basierend auf der Höhe des auf der Reise erzielten Gewinns, ermittelt wird.

MIDI-Rock

Wir starten ein neues Projekt im Selbstbaukurs: Entwurf und Konstruktion eines MIDI-Interface für den Acorn B oder Commodore 64. Zum Auftakt erläutern wir das MIDI-Konzept und seine Anforderungen an die Hardware.

MIDI ist die Abkürzung für Musical Instrument Digital Interface (Digitale Schnittstelle für Musikinstrumente). MIDI-Schnittstellen haben festgelegte Eigenschaften, die eine Verbindung der nach diesem Standard ausgerüsteten Geräte erlauben. Der Datenaustausch geht digital vor sich. Also kann auch ein Heimcomputer in ein MIDI-System integriert werden, sowohl zur Steuerung von elektronischen Instrumenten als auch zum Speichern von Daten. Ein MIDI-Interface kann leicht auf einer Zusatzplatine aufgebaut werden; der materielle Aufwand hält sich in Grenzen.

Seit Einführung der MIDI-Spezifikationen im August 1983 gab es über die Leistungsfähigkeit von MIDI eine Vielzahl von Mißverständnissen, nicht zuletzt deshalb, weil viele Musiker elektronischen Instrumenten eher mit Desinteresse begegneten. Und die musikalischen Möglichkeiten wurden zu Beginn wohl auch ein wenig nachlässig und ungenau dargestellt.

In unserem Kurs wollen wir die Eigenschaften von MIDI in den Begriffen des Programmierers erklären, ohne daß der Computer den Musikern unter Ihnen den Spaß verderben soll. Sie erhalten die gleichen Grundinformationen, die auch ein Entwicklungsingenieur für die Gestaltung eines MIDI-kompatiblen Gerätes zur Verfügung hat.

Zuerst einmal benötigen wir eine ungefähre Vorstellung vom Ablauf der Datenübertragung mit MIDI. Dazu betrachten wir die verschiedenen Teile eines elektronischen Musiksystems als Peripherie eines Computers. Diese Einteilung ist bei manchen Geräten nicht sofort zu erkennen, trotzdem besteht beispielsweise ein mehrstimmiger moderner Synthesizer aus zwei Komponenten in einem Gehäuse: Tastatur (für die Eingabe) und Tonerzeugungsschaltung (Ausgabeeinheit). Normalerweise sind Ein- und Ausgabeteil direkt gekoppelt – sobald Sie eine Taste drücken, wird auch der gewünschte Ton erzeugt.

Diesen Aufbau kann man mit der Arbeitsweise einer modernen elektronischen Schreibmaschine vergleichen: Auch hier sind Tastatur und Schreibwerk so zusammengeschaltet, daß nach einem Tastendruck sofort ein Zeichen gedruckt wird. Dennoch sind die Einzelkomponenten leicht als unabhängige Teile eines zentralen Computers zu erkennen, der über die Tastatur eingegebene Daten zuerst verarbeitet, bevor er sie an die Ausgabeeinheit weiterleitet. Diese Arbeitsweise haben wir bereits unter der Bezeichnung „Textverarbeitungssystem" kennengelernt.

Bei einem Synthesizer arbeiten die internen Schnittstellen des Gerätes zwischen Tastatur und Klangerzeugungseinheit bereits digital. Die Tastatur dient dabei zur Steuerung des Synthesizers, ähnlich wie die Tastatur der Schreibmaschine das Druckwerk steuert. Diese Verbindung läßt sich aber leicht auflösen. Zwischen beiden Einheiten vermittelt dann der Computer, und aus dem Syntheziser ist ein vielseitig einsetzbares Musik-„Verarbeitungssystem" geworden.

Die „Musik" ist in diesem Fall nicht mehr ein Muster von Luftschwingungen, wie es bei der Wiedergabe über einen Plattenspieler produziert wird. Sie wird vielmehr als eine Reihe von „Ereignissen" verarbeitet. Die Eingabe der Ereignisse erfolgt über die Tastatur, danach werden sie zur Ausgabeeinheit weitergeleitet. Erst die Schaltungen zur Klangerzeugung sind es, die diese „Ereignisse" in elektronische Signale umsetzen und sie über Verstärker/Lautsprecher zu Tönen umformen. Damit kann MIDI aber auch andere Steuereinheiten als nur Tastaturen nutzen. Auch ein Saiten- oder Blasinstrument könnte diesen Zweck erfüllen, sofern es über eine Ausgabeeinheit nach MIDI-Spezifikationen verfügt.

Die gute alte Zeit

Ein gutes Beispiel für ein bekanntes, wenn auch etwas grobes Musik-Verarbeitungssystem stammt aus einer Zeit, in der Computer noch nicht einmal Zukunftsmusik waren: Das Pianola ist ein Klavier, das auf einer Papierrolle eingestanzte Löcher in Musik verwandeln kann. Beim Transport des Papiers durch einen Fühlermechanismus wird das Lochmuster abgetastet, wobei jede Lochposition einer ganz bestimmten Note entspricht. Die Hämmer des Klaviers schlagen exakt nach den Anweisungen dieses papierenen Datenträgers die entsprechenden Saiten an.

Der Abstand eines Loches von der Kante des Papierstreifens definiert die Klanghöhe, und die Länge des Loches gibt an, wie lange der Ton gehalten werden soll. Beim Pianola ist auch eine Bearbeitung von Musikstücken mög-

lich: Ein falsch gestanztes Loch kann überklebt und durch ein neues ersetzt werden. Von der Toncodierung mit dieser Technik führt ein direkter Weg zu den Lochkartensystemen der ersten Computer.

Einen modernen Musikprozessor kann man sich als verbesserte Version der alten Pianola-Technik vorstellen. Das Papier ist dabei durch den Speicher des Computers und magnetische Aufzeichnungsgeräte ersetzt. Die Bearbeitung der Musik besteht heute in der Veränderung der Speicherinhalte des Computers durch ein entsprechendes Programm.

Bevor wir uns damit eingehender beschäftigen, müssen wir ein paar Grundlagen von MIDI genauer kennenlernen. Dazu gehören die Methode der Datenübertragung und das Datenformat. Diese beiden Faktoren entsprechen den Hard- und Softwareeigenschaften der Schnittstelle.

Zuerst müssen wir wissen, wie ein einzelnes Byte übertragen wird. MIDI arbeitet als asynchrones, serielles Interface mit einer Übertragungsrate von 31,25 KBaud. Das bedeutet, daß nur jeweils ein Bit übermittelt werden kann und der Sender mit dem Empfänger nicht über gemeinsame Taktimpulse synchronisiert ist. Nachteil dieser Technik ist die begrenzte Übertragungsgeschwindigkeit. Andererseits besteht die Verbindung der Geräte aus nur zwei Leitungen, was den Verdrahtungs- und Steckeraufwand gering hält und dadurch preiswerter ist.

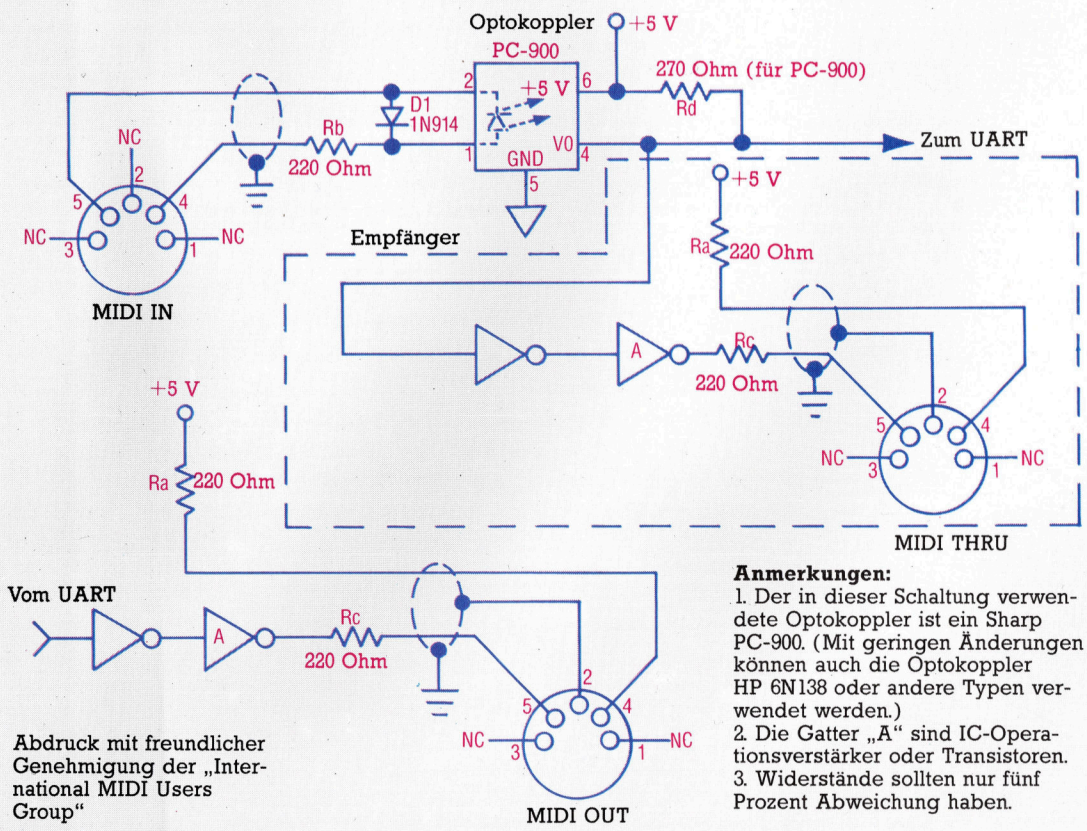
MIDI schreibt die Verwendung von fünfpoligen Standard-180-DIN-Steckern vor (Option: professionelle 3polige XLR-Stecker). Dadurch bleiben die Kosten für das Interface niedrig, so daß sein Einsatz bereits für relativ preiswerte Instrumente lohnt.

Hauptkritikpunkt bei MIDI ist die geringe Übertragungsgeschwindigkeit, die bei einem größeren System durchaus ins Gewicht fällt. Andererseits bleibt die Übertragungsrate dadurch auch in einem Rahmen, der die Verarbeitung mit Heimcomputern möglich macht.

MIDI-Daten werden in Gruppen von zehn Bit gesendet – je acht Bit plus ein Start- und ein Stopbit. Im Ruhezustand liegt die Übertragungsleitung auf +5 Volt (High). Der Beginn einer Übertragung wird dadurch signalisiert, daß die Leitung für eine 1-Bit-Periode auf 0 Volt geht (Startbit). Danach werden die acht Datenbits gesendet – MSB (Most Significant Bit) zuerst, als Abschluß eine Periode von einer Bitlänge mit hohem Level (High). Für die Übertragung eines Byte werden also zehn Bitperioden

MIDI-Standard-Hardware

Das Interface besteht aus Sende- und Empfangsteil. Beide haben eigene Buchsen, die mit MIDI IN (Empfang) und MIDI OUT (Senden) bezeichnet sind. Gelegentlich findet man auch noch einen dritten, mit MIDI THRU bezeichneten Anschluß, der eine exakte Kopie der an MIDI IN empfangenen Daten liefert. Dadurch können Empfangsgeräte hintereinandergeschaltet werden, ohne daß am Sender mehrere Ausgänge nötig sind.



Anmerkungen:
1. Der in dieser Schaltung verwendete Optokoppler ist ein Sharp PC-900. (Mit geringen Änderungen können auch die Optokoppler HP 6N138 oder andere Typen verwendet werden.)
2. Die Gatter „A" sind IC-Operationsverstärker oder Transistoren.
3. Widerstände sollten nur fünf Prozent Abweichung haben.

Abdruck mit freundlicher Genehmigung der „International MIDI Users Group"

gebraucht, eine Zeit von 10 Bit/31,25 KBaud = 320 Microsekunden. Wie wichtig dieser Wert für MIDI-Programmierer ist, werden wir später in dieser Folge noch erfahren.

Ist das Interface inaktiv, geht die Datenleitung auf High. Der Empfänger überwacht die Leitung bis zum Auftreten des ersten Low-Levels – also des Übertragungs-Startbits. Daran erkennt der Empfänger, daß ein Datenbyte gesendet werden soll und startet einen Zähler, der bis eineinhalb Perioden nach dem Startbit läuft. Nach Verstreichen dieser Zeit befindet sich der Empfänger in der Mitte der Periode mit dem MSB-Bit und kann exakt ermitteln, ob es Null oder Eins ist. Die nächsten Bits werden durch Weiterzählen um je eine Periode und jeweiliges Ermitteln des Low- oder High-Wertes bestimmt.

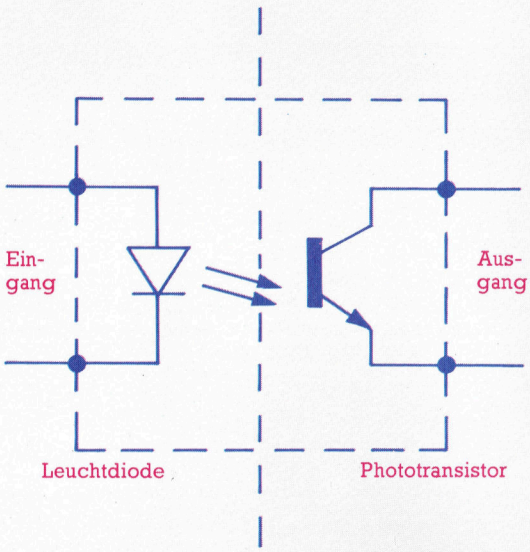
Zum Abschluß wird das Stopbit auf High-Level geprüft, um festzustellen, ob die Datengruppe korrekt vom Start- und Stopbit eingefaßt war. Bei dieser Übertragungsmethode müssen die Taktimpulse von Sender und Empfänger nicht exakt synchronisiert sein. Trotzdem dürfen die Unterschiede zwischen den Taktzeiten nur gering sein. MIDI erlaubt eine Abweichung von maximal einem Prozent.

Wenn man den seriellen Aufbau bedenkt, ist das MIDI-Interface eigentlich relativ schnell (RS232 läuft mit einer Maximalgeschwindigkeit von 19,2 KBaud). Die Verbindungsleitungen dürfen daher nicht länger als 15 Meter sein, um die Anstiegs- und Abfallzeiten kurz zu halten.

Zu diesem Zeitpunkt muß auf die Grenzen von MIDI hingewiesen werden, speziell die Übertragungsverzögerungen zwischen einzelnen Instrumenten. Im allgemeinen besteht die Übertragung eines Ton-EIN/AUS-Befehls aus drei Bytes, dauert also 3*320 Mikrosekunden = 0,96 Millisekunden. Für den Fall eines vielstimmigen Instrumentes, bei dem etwa zwanzig solche Befehle gleichzeitig benötigt werden, bedeutet das: Der letzte Befehl erreicht das Gerät etwa 20 Millisekunden später als beabsichtigt.

Viele im Echtzeitverfahren arbeitende Sequenzer haben Zeitauflösungen bis hinunter zu drei bis vier Millisekunden und liegen damit unter der Wahrnehmungsschwelle. Eine Verzögerung von 20 Millisekunden ist jedoch für das menschliche Ohr bereits erkennbar.

Optokoppler

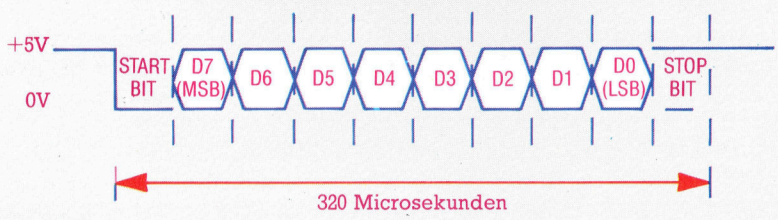


Die zweite Anforderung an eine MIDI-Schnittstelle besteht darin, daß die Eingangsleitung vom Empfangsgerät elektrisch isoliert sein soll, um Schleifen im System zu vermeiden. Solche Schleifen machen sich durch ein störendes Brummen im Bereich der Netzfrequenz unangenehm bemerkbar (50 Hz). Dieses Problem kann mit einem sogenannten „Optokoppler" bewältigt werden, ein Bauteil, bei dem Daten nicht elektrisch, sondern in Form von Licht übertragen werden können.

Die beiden Eingangsleitungen des Optokopplers sind mit einer Leuchtdiode verbunden. Eine LED sendet bei einer positiv angelegten Spannung Licht aus. Die Optokoppler-LED ist von außen nicht sichtbar, sondern beleuchtet einen benachbarten Phototransistor, der bei Lichteinfall leitend wird. Das logische Signal am Eingang des Optokopplers (Low oder High) wird dadurch unverändert auf den Ausgang übertragen, ohne daß eine elektrische Verbindung besteht.

Für MIDI ist festgelegt, daß die Koppler-LED zum Aufleuchten nur einen Strom von weniger als 5mA verwenden darf. Die Anstiegs- und Abfallzeiten am Ausgang sollten unter zwei Microsekunden liegen.

Serielle Datenübertragung bei MIDI



Bilderrahmen

In den nächsten beiden Artikeln sehen wir uns einige Fließkommaroutinen des Commodore 64 genauer an. Wir erstellen damit eine dreidimensionale Figur in hoher Auflösung.

Die Fließkommaroutinen des BASIC-Interpreters sind nicht sonderlich gut dokumentiert. So fehlt beispielsweise die Tabelle der Einsprungadressen, und einzelne Routinen lassen sich nur durch mühsames Experimentieren finden. Wir führen unsere Untersuchung daher anhand einer praktischen Aufgabe durch: Wir rotieren deshalb eine dreidimensionale Rahmenfigur auf dem hochauflösenden Bildschirm.

Die hier eingesetzten mathematischen Techniken lassen sich natürlich auch für andere Routinen einsetzen – beispielsweise bei der Matrixmultiplikation. Bedenken Sie aber, daß mathematische Berechnungen, die wegen der schnell aufeinanderfolgenden Bildschirmdarstellungen in Echtzeit programmiert werden, nicht unbedingt beste Programmiertechnik sind. Es ist eleganter, erst alle Bildschirmdaten zu berechnen und dann die Bewegungsfolge auszulösen.

BASIC-Variablen sind im Speicher über den BASIC-Programmen untergebracht. Sie befinden sich in einer Variablentabelle, auf deren Anfang die Speicherstellen 45 und 46 (dezimal) zeigen. Die Anfangsadresse der Variablentabelle wird so gefunden:

PEEK(45)+256*PEEK(46)

Unsere Tabelle zeigt verschiedene Variablenpointer mit ihrem normalen Inhalt.

Beachten Sie, daß dynamische Strings im Speicher von oben nach unten abgelegt werden, während Arrays über den anderen Variablen der Variablentabelle gespeichert sind. Taucht während der Programmausführung eine neue Variable auf, verschiebt das Betriebssystem den Arraybereich um die Anzahl Bytes nach oben, die die Variable benötigt.

Die Speicherung von Strings ist komplizierter. Ganzzahlenvariablen (außerhalb von Arrays) und Fließkommavariablen belegen je sieben Bytes, Stringvariablen jedoch bis zu 255 Bytes. Das Betriebssystem speichert daher nur die Stringlänge und einen Pointer auf die Anfangsadresse. Wenn in einem BASIC-Programm ein String definiert wird (etwa A$="ABCD"), dann zeigt der Pointer auf das erste Byte des Strings im BASIC-Programmbereich. Ein String dieser Art heißt „statisch". Wenn das Programm einen Stringwert ändert, wird der „dynamisch". Die Werte dynamischer Strings werden von der Speicherobergrenze an abwärts angelegt und die Pointer der Variablentabelle entsprechend verändert. Auf diese Weise hat jeder Eintrag die konstante Länge von sieben Bytes.

Pointer	Funktion	Normaler Inhalt
43/44	Anfang von BASIC	2049
45/46	Anfang der Variablen	je nach Programmlänge
47/48	Anfang der Arrays	je nach Variablenzahl
49/50	Ende der Arrays + 1	je nach Arrayzahl/-größe
51/52	Stringuntergrenze	je nach Stringzahl/-länge
55/56	Speicherobergrenze	40960

Die Adresse einer Variablen in der Variablentabelle läßt sich mit ein wenig Detektivarbeit leicht finden. Nach Aufruf der Variablen wird ihre Adresse in einen Pointer der Zero Page geladen (dezimal 71 und 72), der dort untersucht werden kann.

Das folgende Programm zeigt das Speicherformat der normalen Variablen. Die erste Routine holt einen String aus dem Speicher und sieht sich mit der gerade beschriebenen Technik den Inhalt der Adressen 71 und 72 an. Die Werte werden dann in zwei freien Bytes des Cassettenbuffers zur späteren Berechnung der Stringadresse gespeichert.

```
1000 REM**FIND A STRING IN MEMORY**
1010 X$="ABCDEF"
1020 REM**MAKE X$ CURRENT VARIABLE**
1030 X$=X$+""
1040 REM**SAVE POINTER TO VAR TABLE**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**ADDRESS IN VAR TABLE**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**LOOK AT ENTRY IN VAR TABLE**
1090 LS=PEEK(ADR):REM LENGTH OF STRING
1100 SA=PEEK(ADR+1)+256*PEEK(ADR+2)
1110 REM SA IS START ADDR OF STRING
1120 REM**NOW READ STRING**
1130 FORI=SATOSA+LS
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINTVAR$
```

Normalerweise benötigen Ganzzahlenvariablen (zum Beispiel X%) weniger Speicher und beschleunigen außerdem mathematische Abläufe. Beim C 64 ist das jedoch nicht der Fall. Wenn das Commodore-BASIC Ganzzahlberechnungen ausführt, wandelt es die Werte erst in das Fließkommaformat um und ruft dann die Fließkommaroutinen auf. Auf diese Weise belegen Ganzzahlvariablen, die eigentlich mit zwei Bytes auskommen, sieben Speicherbytes (falls sie nicht in Arrays abgelegt sind). Bei der Weiterverarbeitung werden die zusätzlichen Bytes jedoch ignoriert. So können Sie eine Ganzzahl im Speicher finden:

```
1000 REM**FIND AN INTEGER IN MEMORY**
1010 X%=3456
1020 REM**MAKE X% CURRENT VARIABLE**
1030 X%=X%
1040 REM**SAVE POINTER TO VAR TABLE**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**ADDRESS IN VAR TABLE**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**LOOK AT ENTRY IN VAR TABLE**
1090 LO=PEEK(ADR+1):HI=PEEK(ADR)
1100 REM**COMPUTE RESULT**
1110 SIGNBIT=(HIAND128)/128
1120 VAR=LO+256*(HIAND127)-32768*SIGNBIT
1130 PRINT VAR
```

Mit dieser Technik können Sie natürlich auch Fließkommavariablen suchen. Es gibt jedoch eine weit praktischere Methode.

Das folgende Programm berechnet mit DEF FN die Adresse (in Speicherstelle 71 und 72) der aktuellen BASIC-Variablen. Da X eine Funktionsvariable ist, gibt die erzeugte Adresse die Speicherstelle des ersten Bytes von X in der BASIC-Variablentabelle wieder. Beim Aufruf von FN wird diese Adresse der Variablen ADD zugeordnet. Beachten Sie, daß die Übergabe von 0 (oder eines anderen Wertes) mit dem FN-Befehl den Wert von X in der Variablentabelle oder die von der Funktion berechnete Adresse nicht verändert.

Der Commodore 64 arbeitet mit drei Variablentypen, die mit verschiedenen Formaten im Speicher Platz finden. Ganzzahlvariablen werden in zwei Bytes als Zweierkomplement gespeichert. Fließkommazahlen verlangen fünf Bytes für Mantisse, Vorzeichenbit und Exponenten. Stringvariablen sind in einem anderen Bereich des Speichers untergebracht und belegen ein Byte pro Zeichen. Die Stringdaten befinden sich jedoch nicht in der Variablentabelle. Dort geben statt dessen drei Bytes die Zeichenzahl des Strings an, während eine 16-Bit-Adresse auf den Anfang des Strings im Speicher zeigt.

```
1000 REM**FIND A FPVAR IN MEMORY**
1010 DEF FNADR(X)=PEEK(71)+256*PEEK(72)
1020 ADD=FNADR(0):REM ALWAYS RETURNS ADDRESS OF X
1030 X=-3.14159
1040 REM**CONVERT FROM BASE 2 TO DECIMAL
1050 POWERTWO=2^(PEEK(ADD)-129)
1060 SIGN=(-1)^((PEEK(ADD+1)AND128)/128)
1070 REM*FRACTION PART IS 31 BITS WIDE*
1080 D1=PEEK(ADD+1)AND127:REM 7 BITS
1090 D2=PEEK(ADD+2):REM 8 BITS
1100 D3=PEEK(ADD+3):D4=PEEK(ADD+4)
1110 REM**GULP!**
1120 FRACT=2^(-7)*D1+2^(-15)*D2+2^(-23)*D3+2^
     (-31)*D4
1130 MANT=1+FRACT
1140 VAR=SIGN*POWERTWO*MANT
1150 PRINTVAR
```

Die Ausführung eines DIM-Befehls reserviert im Speicher Platz für ein Array. Der reservierte Speicher enthält einen Kopf und die Anzahl der für die Arrayelemente abgestellten Bytes. Das Format von Arrayelementen unterscheidet sich von allen anderen Variablentypen. Unser Diagramm zeigt die möglichen Variationen. Wenn Sie Arrayelemente vom Maschinencode aus ansprechen wollen, müssen Sie diese Unterschiede kennen.

Sehen wir uns nun die Fließkommaarithmetik genauer an. Bei der Ausführung von Fließkommaberechnungen speichert der BASIC-Interpreter seine Ergebnisse in zwei „Fließkommaakkumulatoren" – FAC und ARG –, die das gleiche Format haben wie die im Speicher abgelegten Variablen. FAC befindet sich bei den Adressen $61 bis $65 (dezimal 97 bis 101) und ARG bei $69 bis $6D (dezimal 105 bis 109). Aus Gründen der Einfachheit werden wir auch weiterhin mit den Interpreterroutinen arbeiten, die die Zahlen zwischen FAC und dem entsprechenden Speicher übertragen.

Array-Variablen

Ganzzahlvariablen

Hi-Byte der Ganzzahl	Lo-Byte der Ganzzahl
Byte 0	Byte 1

Fließkommavariablen

Binärexponent Plus 128	Vorzeichenbit AND Mantisse 1 (7 Bits)	Mantisse 2	Mantisse 3	Mantisse 4
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4

String-Variablen

Anzahl der Zeichen	Lo-Byte der Anfangsadresse des Strings	Hi-Byte der Anfangsadresse des Strings
Byte 0	Byte 1	Byte 2

Hier sind die Routinen

Wir verwenden folgende Interpreterroutinen:

- MOVFM (CALL Adresse $BBA2):

MOVFM lädt den Inhalt von FAC mit der im Speicher abgelegten Fließkommavariablen. Symbolisch wird dieser Vorgang durch : ← M dargestellt. Für den Aufruf wird der Akkumulator mit dem Lo-Byte der Anfangsadresse der Speichervariablen geladen und das Y-Register mit dem Hi-Byte.

- MOVMF (CALL Adresse $BBD4):

Diese Routine legt den Inhalt von FAC in sieben Speicherbytes ab (M ← F). Für den Aufruf wird das X-Register mit dem Lo-Byte und Y mit dem Hi-Byte des Anfangsbytes der Variable im Speicher geladen.

- FMULT (CALL Adresse $BA28):

Diese Routine multipliziert den Inhalt von FAC mit einer zweiten Speichervariable und legt das Ergebnis in FAC ab. Die erste Variable wird mit MOVFM in FAC geladen. Wir zeigen auf die zweite Variable, indem wir vor Aufruf der Routine den Akkumulator mit dem Lo-Byte und Y mit dem Hi-Byte der Anfangsbytes laden. Falls nötig, können wir das Ergebnis mit MOVMF wieder speichern.

- FADD (CALL Adresse $B867):

Diese Additionsroutine führt FAC=MEM+FAC aus. Für den Aufruf wird der Akkumulator mit dem Lo-Byte von MEM und Y mit dem Hi-Byte von MEM geladen.

- FSUB (CALL Adresse $B850):

Diese Subtraktionsroutine führt FAC=MEM–FAC aus. Für den Aufruf wird der Akkumulator mit dem Lo-Byte von MEM und Y mit dem Hi-Byte von MEM geladen.

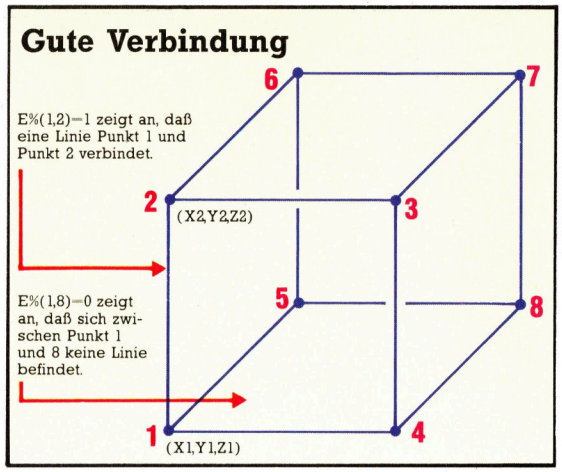
Diese Interpreterroutinen bauen wir in die erste Phase unseres Grafikprogramms ein.

Das Grafikprogramm arbeitet mit einer Umrißfigur, die durch eine Reihe von Punkten und eine Matrix von Eckverbindungen festgelegt wird. Die Punkte haben die Koordinaten X(I),Y(I),Z(I) – wobei I die Werte 1 bis NP (Anzahl der Punkte) durchlaufen kann. Die Matrix der Eckverbindungen ist E%(I,J) – wobei I und J die Zahlen 1 bis NP durchlaufen können. E%(I,J) wird als Eins definiert, wenn Punkt I mit

Punkt J verbunden ist. Anderenfalls ist diese Variable Null. Natürlich wird dabei der Speicher nicht optimal eingesetzt (wir arbeiten mit zwei Bytes, obwohl eines ausreichen würde). Unsere Methode hat jedoch den Vorteil, daß sich die zu verbindenden Punkte leichter angeben lassen. Außerdem kann der Wert von NP im praktischen Einsatz nie bemerkenswert hoch werden.

Um die Rotationsroutine im Maschinencode von BASIC aus ansprechen zu können, geben wir die Basisadressen der Arrays mit POKE in den Speicher. Wir beginnen mit Element 1 und müssen daher die Adresse von X(1),Y(1),Z(1) und E%(1,1) finden. Weiterhin müssen wir den Maschinencode mit mehreren Angaben versorgen: NP sowie COS und SIN des Rotationswinkels um die Achse Z.

Bei der Entwicklung eines komplexen Maschinencodeprogramms lohnt es sich, in kleinen Schritten vorzugehen, da sonst später viel Zeit für eine eventuelle Fehlersuche nötig ist. Wir haben das Projekt daher in drei Phasen geplant. Zunächst definieren wir die Algorithmen und schreiben ein Testprogramm in BASIC (BASIC-Rotationsprogramm). Es rotiert einen Würfel um die Z-Achse und bildet das Ergebnis auf den Ebenen X und Y ab.



Das Programm ROTSUB rotiert einen durchsichtigen Würfel in der hochauflösenden Grafik des Commodore 64. Die Linien der Figur werden mit vier Arrays definiert. Drei davon halten die X-, Y- und Z-Koordinaten jedes einzelnen Punktes. Das vierte Array – E%(,) – definiert, welche der Punkte durch Linien miteinander verbunden werden.

Unser Ziel ist es, dieses BASIC-Listing in Maschinencode zu übersetzen. Wir beschreiben zunächst jedoch eine Version der Subroutine bei Zeile 1800 im Maschinencode: das Programm I-ROTSUB. Der dritte Kasten enthält ein Programm, das I-ROTSUB testet.

In der nächsten Folge vervollständigen wir das Projekt über die Fließkommaroutinen und die dreidimensionale Figur in hoher Auflösung und drucken den restlichen Assemblercode und ein BASIC-Ladeprogramm ab.

Das Programm I-ROTSUB

Assemblieren und laden Sie das folgende Assemblerlisting und speichern Sie den Code unter I-ROT.HEX.

```
;++++++++++++++++++++++++++
;++     I-ROTSUB 64      ++
;++                      ++
;++++++++++++++++++++++++++
;++USES ARRAYS DEFINED    ++
;++FROM BASIC ADDRESSES   ++
;++MUST BE POKED FIRST    ++
;++++++++++++++++++++++++++
;
;PLTSUB = $C183
;XLO    = $C103
;XHI    = $C104
;YLO    = $C105
;
*       = $C544
;
;++++ ROTSUB  VARIABLES ++++
;
; VARIABLES CALLED FROM BASIC
XBASLO  *=*+1 ; POKE50500,X(0)LO
XBASHI  *=*+1 ; POKE50501,X(0)HI
YBASLO  *=*+1 ; POKE50502,Y(0)LO
YBASHI  *=*+1 ; POKE50503,Y(0)HI
NP      *=*+1 ; POKE50504,NP
; VARIABLES USED BY M/C
XILO    *=*+1
XIHI    *=*+1
YILO    *=*+1
YIHI    *=*+1
CSLO    *=*+1
CSHI    *=*+1
SNLO    *=*+1
SNHI    *=*+1
MEM1    *=*+5 ;FLOATING POINT VAR
MEM2    *=*+5 ;FLOATING POINT VAR
FAC      =$0061
ARG      =$0069
;
;INTERPRETER ARITHMETIC CALLS
;
FMULT    =$BA28 ; FAC=FAC*ARG
FADDT    =$B86A ; FAC=FAC+ARG
FSUB     =$B850 ; FAC=MEM-FAC
FADD     =$B867 ; FAC=FAC+MEM
MOVFM    =$BBA2 ; FAC=MEMORY
MOVMF    =$BBD4 ; MEMORY=FAC
;
;++++ SAVE REGISTERS++
;
        PHA
        TXA
        PHA
        TYA
        PHA
;
;++++ INITIALISE VARIABLES ++++
;
        LDA XBASLO
        STA XILO
        LDA XBASHI
        STA XIHI
        LDA YBASLO
        STA YILO
        LDA YBASHI
        STA YIHI
;
;++++PERFORM  MEM1=X(I)*CS-Y(I)*SN
;
START
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = X(I)*CS
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1=X(I)*CS
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = Y(I)*SN
        LDA #<MEM1
        LDY #>MEM1
        JSR FSUB    ; FAC = MEM1-FAC
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF   ; MEM1= FAC
;
;++++PERFORM MEM2=Y(I)*CN+X(I)*SN
;
        LDA YILO
        LDY YIHI
        JSR MOVFM   ; FAC = Y(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT   ; FAC = Y(I)*CS
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= Y(I)*CS
        LDA XILO
        LDY XIHI
        JSR MOVFM   ; FAC = X(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT   ; FAC = X(I)*SN
        LDA #<MEM2
        LDY #>MEM2
        JSR FADD    ; FAC = MEM2+FAC
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF   ; MEM2= FAC
;
;++++PERFORM X(I) =MEM1:Y(I)=MEM2
;
        LDA #<MEM1
        LDY #>MEM1
        JSR MOVFM   ; FAC =MEM1
        LDX XILO
        LDY XIHI
        JSR MOVMF   ; X(I)=FAC
        LDA #<MEM2
        LDY #>MEM2
        JSR MOVFM   ; FAC =MEM2
        LDX YILO
        LDY YIHI
        JSR MOVMF   ; Y(I)=FAC
;
;++++TEST END LOOP
;
        DEC NP
        BEQ EXIT
;
;++++INCREMENT ARRAY POINTERS
;
        LDA #$05
        CLC
        ADC XILO
        STA XILO
        BCC XNOHI
        INC XIHI
XNOHI
        LDA #$05
        CLC
        ADC YILO
        STA YILO
        BCC YNOHI
        INC YIHI
YNOHI
        JMP START
;
;++++ PULL REGISTERS OFF STACK ++++
;
EXIT
        PLA
        TAY
        PLA
        TAX
        PLA
;
        RTS
.END
```

Rotationsprogramm in BASIC

```
1000 REM** BASIC ROTATING CUBE**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 REM**DIMENSION ARRAYS**
1040 NP=8:REM NUMBER OF POINTS
1050 DIM X(NP),Y(NP),Z(NP)
1060 DIM ED(NP,NP):REM EDGE CONNECTIONS
1070 REM**INITIALISE ARRAYS**
1080 REM--CUBE COORDINATE DATA
1090 DATA  75, 75, 75:REM----------/1
1100 DATA -75, 75, 75:REM TOP FOUR /2
1110 DATA -75,-75, 75:REM POINTS   /3
1120 DATA  75,-75, 75:REM----------/4
1130 DATA  75, 75,-75:REM----------/5
1140 DATA -75, 75,-75:REM BOT FOUR /6
1150 DATA -75,-75,-75:REM POINTS   /7
1160 DATA  75,-75,-75:REM----------/8
1170 REM**ROTATE CUBE ABOUT X-AXIS π/4
1180 FORI=1TONP
1190 READX(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM**ROTATE CUBE ABOUT Z-AXIS π/4
1240 FORI=1TONP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM--EDGE CONNECTION DATA--
1290 E(1,2)=1:REM 1 CONNECTED TO 2
1300 E(2,3)=1:E(3,4)=1:E(4,1)=1
1310 E(5,6)=1:REM BOT SQUARE
1320 E(6,7)=1:E(7,8)=1:E(8,5)=5
1330 E(5,1)=1:REM TOP TO BOT EDGES
1340 E(6,2)=1:E(7,3)=1:E(8,4)=1
1350 REM**SYMMETRISE E(I,J)**
1360 FORI=1TONP:FORJ=1TONP
1370 IFE(I,J)=1THENE(J,I)=1
1380 NEXT:NEXT
1390 REM#########################
1400 REM##PLOT ROTATING CUBE##
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2*π STEP SA
1430 GOSUB1800:REM ROTATE THRU SA
1440 GOSUB1590:REM INIT/CLEAR SCREEN
1450 REM--PLOT CUBE--
1460 FORI=1TONP
1470 FORJ=1TOI
1480 IF E(I,J)=0THEN1510:REM NOT JOINED
1490 GOSUB1630:REM COMPUTE PROJECTION
1500 GOSUB1670:REM JOIN POINTS
1510 NEXT:NEXT
1520 REM-------------
1530 NEXT A:REM NEXT ANGLE
1540 REM#########################
1550 REM**WAIT**
1560 GETA$:IFA$=""THEN1560
1570 GOSUB1760:REM RESET SCREEN
1580 END
1590 REM**SETUP HIRES**
1600 POKE49408,1:POKE49409,1
1610 POKE49410,1:SYS49422
1620 RETURN
1630 REM**COMPUTE PROJECTION ON HIRES**
1640 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1660 RETURN
1670 REM**LINESUB**
1680 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1690 MHI=INT(X1%/256):MLO=X1%-256*MHI
1700 NHI=INT(X2%/256):NLO=X2%-256*NHI
1710 POKE49920,MLO:POKE49921,MHI
1720 POKE49922,NLO:POKE49923,NHI
1730 POKE49924,Y1%:POKE49925,Y2%
1740 SYS49934:REM LINESUB
1750 RETURN
1760 REM**RESET SCREEN**
1770 POKE49408,0:SYS49422
1780 PRINTCHR$(147)
1790 RETURN
1800 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1810 FORI=1TONP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

Testprogramm für I-ROTSUB

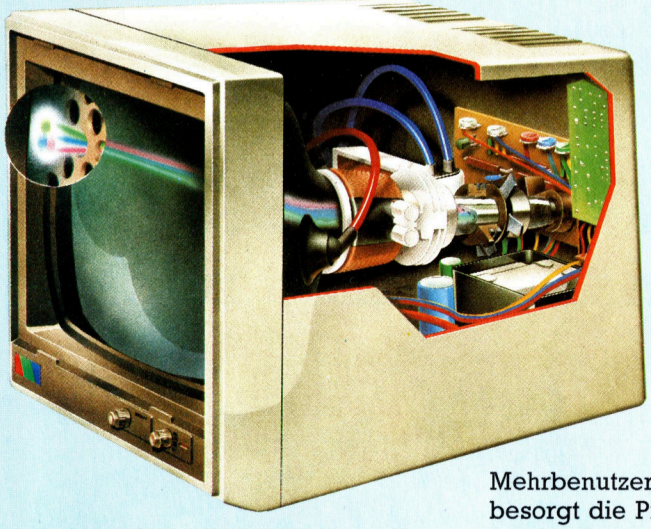
Geben Sie das folgende BASIC-Programm ein und speichern Sie es als TEST I-ROT. Achten Sie darauf, daß zwischen der Ausführung von Zeile 1880 bis 2000 und dem Aufruf von SYS50523 keine Variable definiert wird, da sonst die Gefahr des Programmabsturzes bestünde.

```
1000 REM** TEST I-ROT **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"I-ROT.HEX",8,1
1040 REM**DIMENSION ARRAYS**
1050 NP=8:REM NUMBER OF POINTS
1060 DIM X(NP),Y(NP),Z(NP)
1070 DIM ED(NP,NP):REM EDGE CONNECTIONS
1080 REM**INITIALISE ARRAYS**
1090 REM--CUBE COORDINATE DATA
1100 DATA  75, 75, 75:REM----------/1
1110 DATA -75, 75, 75:REM TOP FOUR /2
1120 DATA -75,-75, 75:REM POINTS   /3
1130 DATA  75,-75, 75:REM----------/4
1140 DATA  75, 75,-75:REM----------/5
1150 DATA -75, 75,-75:REM BOT FOUR /6
1160 DATA -75,-75,-75:REM POINTS   /7
1170 DATA  75,-75,-75:REM----------/8
1180 REM**ROTATE SPACE ABOUT X-AXIS π/4
1190 FORI=1TONP
1200 READX(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM**ROTATE SPACE ABOUT Z-AXIS π/4
1250 FORI=1TONP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM--EDGE CONNECTION DATA--
1300 E(1,2)=1:REM 1 CONNECTED TO 2
1310 E(2,3)=1:E(3,4)=1:E(4,1)=1
1320 E(5,6)=1:REM BOT SQUARE
1330 E(6,7)=1:E(7,8)=1:E(8,5)=5
1340 E(5,1)=1:REM TOP TO BOT EDGES
1350 E(6,2)=1:E(7,3)=1:E(8,4)=1
1360 REM**SYMMETRISE E(I,J)**
1370 FORI=1TONP:FORJ=1TONP
1380 IFE(I,J)=1THENE(J,I)=1
1390 NEXT:NEXT
1400 REM#########################
1410 REM##PLOT ROTATING CUBE##
1420 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB1870:REM ROTATE THRU SA
1450 GOSUB1600:REM INIT/CLEAR SCREEN
1460 REM--PLOT CUBE--
1470 FORI=1TONP
1480 FORJ=1TOI
1490 IF E(I,J)=0THEN1520:REM NOT JOINED
1500 GOSUB1640:REM COMPUTE PROJECTION
1510 GOSUB1680:REM JOIN POINTS
1520 NEXT:NEXT
1530 REM-------------
1540 NEXT A:REM NEXT ANGLE
1550 REM#########################
1560 REM**WAIT**
1570 GETA$:IFA$=""THEN1570
1580 GOSUB1770:REM RESET SCREEN
1590 END
1600 REM**SETUP HIRES**
1610 POKE49408,1:POKE49409,1
1620 POKE49410,1:SYS49422
1630 RETURN
1640 REM**COMPUTE PROJECTION ON HIRES**
1650 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM**LINESUB**
1690 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1700 MHI=INT(X1%/256):MLO=X1%-256*MHI
1710 NHI=INT(X2%/256):NLO=X2%-256*NHI
1720 POKE49920,MLO:POKE49921,MHI
1730 POKE49922,NLO:POKE49923,NHI
1740 POKE49924,Y1%:POKE49925,Y2%
1750 SYS49934:REM LINESUB
1760 RETURN
1770 REM**RESET SCREEN**
1780 POKE49408,0:SYS49422
1790 PRINTCHR$(147)
1800 RETURN
1810 REM**ROTATE CUBE ABOUT Z-AXIS/SA
1820 FORI=1TONP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM**ROTATE ABOUT Z-AXIS/SA
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 POKE50504,NP:REM NUMBER OF POINTS
1950 CS=CS:REM MAKE CS CURRENT VAR
1960 POKE50509,PEEK(71)
1970 POKE50510,PEEK(72)
1980 SN=SN:REM MAKE SN CURRENT VAR
1990 POKE50511,PEEK(71)
2000 POKE50512,PEEK(72)
2010 SYS50523
2020 RETURN
```

Fachwörter von A bis Z

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.

Monitor = Monitor
Als Hardwarebegriff wird die Bezeichnung Monitor für ein Datensichtgerät verwendet, dessen Bildröhre der eines normalen Fernsehapparates entspricht. Die Elektronik ist jedoch speziell für die direkte Wiedergabe des Computer-Bildsignals ausgelegt. Daher bietet der Monitor eine erheblich höhere Bildqualität als ein „zweckentfremdeter" Fernsehempfänger, bei dem die Information ja erst einem hochfrequenten Träger aufmoduliert werden muß, um das übliche Antennensignal zu simulieren.



Monochrom-Monitore haben meist einen grün oder bernsteinfarben leuchtenden Schirm, der die Augen auch bei längerem Arbeiten am Bildschirm nicht übermäßig belastet. Farbmonitore gibt es in zwei Ausführungen: Bei den RGB-Monitoren werden die Farbanteile Rot, Grün und Blau getrennt übertragen, während die Composite-Video-Monitore mit einem einzigen Mischsignal arbeiten.

Farbmonitore sind entweder für ein „Composite Video" (=FBAS)-Signal oder für RGB-Signale eingerichtet. Beim RGB-Monitor werden die Farbanteile Rot, Grün und Blau über getrennte Leistungsadern und eine mehrpolige DIN- oder auch Euro-AV-Buchse zugeführt. Ein Composite-Video-Monitor empfängt statt dessen nur ein einziges Mischsignal, das im Gerät zur Gewinnung der Farbanteile decodiert werden muß. Der Anschluß erfolgt gewöhnlich über eine BNC-Buchse, wie sie bei Videorecordern gängig ist. Die Bildqualität liegt bei den RGB-Monitoren durchweg eine Stufe höher als bei den Composite-Video-Geräten.

Softwaremäßig versteht man unter „Monitor" ein Systemprogramm, das Steuerungs- und Überwachungsaufgaben wahrnimmt, vorzugsweise beim Abarbeiten von „Jobs" im Mehrbenutzerbetrieb. Der Monitor besorgt die Prioritätssteuerung, die Zuteilung von CPU- und Peripheriezeiten an Simultanprogramme und koordiniert den Datenaustausch. Wo der Monitor für die Organisation des Gesamtsystems zuständig ist, wird er auch als „Supervisor" oder Hauptsteuerprogramm bezeichnet.

Motherboard = Hauptplatine
Das Motherboard ist die Hauptplatine eines Computers. Darauf befinden sich alle wesentlichen Komponenten, etwa die CPU, ein Teil des Arbeitsspeichers und die Ein-/Ausgabesteuerung. Das Motherboard trägt oftmals eine ganze Reihe von Steckleisten für Zusatzkarten.

Als in der Frühzeit der Heimcomputer die Beschäftigung mit Micros noch eine Domäne der Elektronikbastler war, versuchten die Freaks, sich gegenseitig durch Einbau von Zusatzkarten zu übertrumpfen. Mancher Apple platzte buchstäblich aus den Nähten, weil man ihn einfach zu sehr „vollgestopft" hatte. Zwar gibt es das auch heute noch, aber das große Geschäft mit den Zusatzkarten spielt sich im kommerziellen Bereich ab, wo Personal Computer mit anwenderspezifischen Sonderfunktionen ausgestattet werden sollen.

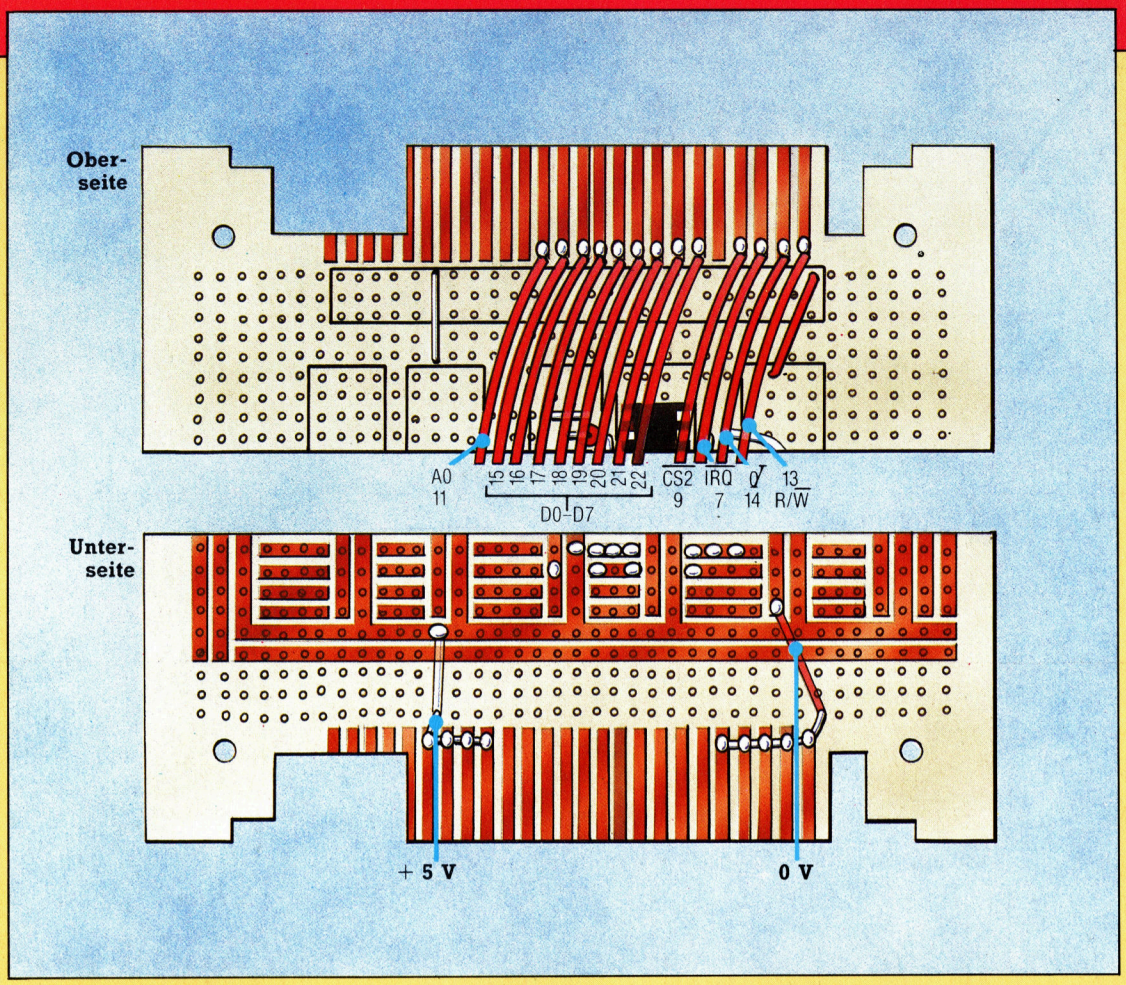
Mouse = Maus
Die Maus ist ein Eingabegerät, mit dem sich durch Hin- und Herfahren auf der Tischplatte der Cursor steuern läßt. Druckknöpfe auf der Maus ermöglichen das Auslösen definierter Rechneraktionen, wenn sich der Cursor über einem bestimmten Piktogramm oder Menüdetail befindet.

Eine solche Maus hat auf der Unterseite eine bewegliche Kugel, deren Rotation sich über Reibrollen auf zwei Codierscheiben überträgt, die ihrerseits durch Lichtschranken abgefragt werden. So wird die Bewegung der Maus in eine elektrische Impulsfolge umgesetzt und an den Rechner übermittelt.

Noch vor zwei Jahren galt die Maussteuerung in Fachkreisen als reine Spielerei. Durch den Erfolg des Apple Macintosh erkannte die Industrie die Vorteile dieses Eingabegeräts und begann, für immer mehr Rechner sogenannte „Maus-Pakete" zu produzieren.

Etwas ganz anderes sind die „intelligenten" Robotermäuse, die sich in einem unbekannten Labyrinth zurechtfinden. Sie verfügen über ein Sensorsystem für das Erkennen von Hindernissen und sind entweder mit einer eigenen Steuerlogik ausgestattet oder sie hängen an einem Rechner, der das „Denken" für sie übernimmt. Die Maus fährt durch das Labyrinth, bis sie irgendwo anstößt, und versucht dann, den Ausgang zu finden.

Bildnachweise
1877: Paul Bryant
1878–1879, 1882: Caroline Clayton
1883: Dimension Graphics
1885: Marcus Wilson-Smith
1888–1895: Rolf Seiffe
1896, 1897: Ian McKinnell
1898, 1899, U3: Kevin Jones



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

computer kurs

Klänge vom Chip

Wichtig für Musikfreunde: Wir beschreiben den Aufbau eines MIDI-Interface für den Commodore C 64 und den Acorn B.

Dog and Bucket

Das vollständige Listing dieses Spiels mit genauen Erläuterungen.

Das hohe C

Die Programmiersprache C weist erstaunliche Parallelen zu PASCAL auf, etwa bei Schleifenstrukturen.

Fürs Heimkino

Pioneers PX-7 verfügt über „audiovisuelle" Schnittstellen, so daß unter anderem auch Bildplatten angesteuert werden können.