TREVOR TOMS

DAS SPECTRUM BUCH

Programmieren in Maschinensprache
Spielprogramme
Viele Spectrum-Tips

ZX POWER

Hueber Software

Hueber Software

Hueber Software

Titel der englischen Originalausgabe
The Spectrum Pocket Book
Übersetzung: T. Westermayer
Übersetzung der Programme: T. Westermayer jun.
Umschlagfoto: Tony Stone Association, London
Umschlagentwurf: Agentur Cooperation, München
Lektorat: Joe Hembus
Herstellung: R. Bergmann

1. Auflage 1984 3 2 1

Satz: Fotosatz Weihrauch, Würzburg
Druck: RMO-Druck, München
ISBN 3-19-008207-3

Inhalt

Mein Dank geht an:

Dave Whitley, der geduldig auf eine klare Nacht wartete;
Liz, die auf die Tippfehler achtet;
John, für ein paar gute Ideen;
und Carys, einfach fürs Dabeisein.

Einleitung

Wieder hat Sinclair Research sich selbst übertroffen - der ZX Spectrum stellt einen gewaltigen Fortschritt gegenüber den früheren Sinclair-Computern dar. Er ist nicht nur preiswerter als ein Original-ZX80 mit zusätzlichem Speicher, sondern besitzt auch Eigenschaften, die man sich noch vor zwei Jahren bei einem Computer dieser Größe und Preislage nicht einmal hätte träumen lassen. Die Technik hat gegenüber dem ZX81 bedeutsame Fortschritte gemacht, mit einer zuverlässigen, schnellen Kassetten-Schnittstelle, keinem Bildschirmzucken und einem Keyboard, das viel leichter zu gebrauchen ist.

Das Buch, das Sie in Händen halten, ist eine ähnliche Entwicklung - das ZX80 Pocket Book sollte eine vom Handbuch hinterlassene Lücke füllen, während das ZX81 Pocket Book noch mehr auf die Fähigkeiten der ZX-BASIC-Sprache einging. Dieses Buch ist ein weiterer Schritt nach vorn. Es befaßt sich mit einem viel größeren Gebiet und liefert Programme, die Spaß machen, die ernst gemeint sind, die einem etwas beibringen oder auch nur nützliche Instrumente sind.

Obwohl Spectrum-BASIC eine erweiterte Version von ZX81-BASIC ist, habe ich keines der Programme aus den früheren Bänden übernommen, weil die zusätzlichen Befehle zu einer ganz neuen Art von Programmpräsentation führen. Viele ZX81-Besitzer werden zweifellos zum Spectrum aufsteigen und in diesen Programmen nur neues Material vorfinden.

Das ist alles möglich geworden durch Aufpassen und Zuhören - ich habe viele Briefe bekommen, die von einer Art Programme mehr, von einer anderen weniger verlangen. Zu Anfang wünschen sich die meisten Leute Spaß und Spiele - sogar für diesen Zweck allein bietet der Spectrum einen großartigen Gegenwert für sein Geld - aber wenn die Neuheit der Spiele verblaßt, wird der Wunsch nach mehr Wissen größer. Wie arbeitet ein Computer? Was ist ein Mikrocomputer eigentlich genau? Wie nutze ich Maschinencode?

Die Antworten auf diese Fragen werden Türen öffnen für viele lange Abende, aber es verschafft ein enormes Gefühl der Befriedigung, einen Computer dabei zu beobachten, wie er mit seinem ganzen Können Programme fährt.

Auf diesen Seiten habe ich versucht, Ihnen Spaß und Spiele für sofortigen Genuß zu liefern, ein paar Nutzprogramme, um Ihr eigenes Programmieren ein bißchen zu vereinfachen, Winke und Tips, um Ihnen einige der Merkmale zu zeigen, die in den Sinclair-Büchern nicht erklärt werden, und einen Abschnitt über den Gebrauch von Maschinencode, um Ihnen auch für die Zukunft Befriedigung zu verschaffen.

Der Spectrum besitzt Einrichtungen für hochauflösende Grafik, Farbe und Ton, und das wiederum bedeutet, daß der Bedarf an verbesserter Präsentation der Programme steigt. Das führt zu einem kleinen Dilemma, weil größere Programme für Sie mehr Tipparbeit bedeuten. Um dem gerecht zu werden, habe ich sowohl kleine als auch große Programme aufgenommen, damit Sie je nach Laune die Menge Eintippen anpacken können, die gerade recht ist.

Mir ist stets bewußt gewesen, daß das Eintippen und Fahren von Spielen **nicht** Programmieren ist. Ich hoffe, Sie werden aus dem Buch lernen und ein paar von den Ideen als Grundlage für Ihre eigenen Programme zu verwenden. Wenn Sie sich mit Programmieren in Maschinencode befassen, wird der zweite Abschnitt des Buches viele nützliche Instrumente und Programmiertips bieten – manche können sogar verwendet werden, ohne daß man ihren Zweck verstehen muß.

Wie dieses Buch produziert worden ist

Wir stellen zuerst eine funktionierende Version jedes Programms her und testen es bis zur Zerstörung. Um dafür zu sorgen, daß der funktionierende Code vom Spectrum Sie mit hundertprozentiger Sicherheit erreicht, haben wir für den Spectrum ein Interface gebaut, mit dem er an unseren Textautomaten angeschlossen wird. Programme werden so übertragen und gelangen druckfrisch zu Ihnen.

Nehmen wir nun einmal an, daß Sie, wie das manchmal vorkommt, Ihr Programm eingeben, auf **RUN** drücken – und nichts passiert. Erstens: Keine Panik. Zweitens: Listen Sie mit dem

Drucker oder auf dem Bildschirm auf und vergleichen Sie sehr gründlich mit dem Buch; bitten Sie außerdem jemand, es zusammen mit Ihnen durchzugehen. Wenn dabei das Problem nicht erkennbar wird, lesen Sie das Kapitel über Debugging (Fehlersuche), das Ihnen vielleicht hilft, den Bereich festzustellen, wo das Problem auftritt. Falls alles umsonst ist, schreiben Sie an mich und schildern Sie mir das Problem. Wenn Sie eine Kassette Ihrer Version des Programms mitschicken, ist das nützlich, ebenso eine Drucker-Auflistung des Programms. Wer frankierte Briefumschläge mit Rückadresse beilegt, wird bevorzugt abgefertigt!

Programmpräsentation

Ich hoffe, Sie werden erkennen, daß die Programme auch Programmiermethoden veranschaulichen und Ihnen Anregungen liefern, Ihre eigenen zu schreiben, weil das der Punkt ist, wo ein Homecomputer ein Videospielgerät übertrifft. Natürlich ist es schön, gleichzeitig auch etwas Spaß an der Sache zu haben. Aus diesem Grund gibt es eine größere Anzahl fröhlicher Programme.

Alle Programme sind mit ausführlichen Anmerkungen ausgestattet, damit Sie bei jedem Schritt genau sehen können, was vorgeht. Meine Kommentare stehen neben den Programmlistings in folgender Weise:

```
1100 LET x = 97          (das ist mein Kommentar).
```

Sie können Sie lesen, während Sie das Programm eingeben oder nachher studieren, wenn Sie Gelegenheit gehabt haben, sich anzusehen, was das Programm leistet. Versuchen Sie nicht, die Kommentare mit in den Computer einzugeben. Beispiel: In der obigen Zeile ist das letzte Zeichen der Zeile, das einzugeben wäre, die Ziffer 7.

Kassetten

Die Programme sind auch auf Kassette erhältlich.

Das große Feuer von London

Wenn Sie je nach der Gelegenheit gesucht haben, die Uhr zurückzudrehen, hier ist Ihre große Chance. Nach aller Wahrscheinlichkeit wird der Strom der Zeit, was den Ausgang dieses Spiels betrifft, nicht anders fließen, aber es macht Spaß, den Brand löschen zu wollen, der halb London verwüstete.

Sie erhalten einen Stadtplanauszug der Pudding Lane nah an der Themse - der ideale Ort, um Ihren Eimer zu füllen! Sie gehen die Botolph Lane hinunter und hören jemand "Feuer!" schreien. Blitzschnell stürzen Sie mit dem Eimer in der Hand zur Themse, um Hilfe zu leisten. Wenn Sie nicht schnell genug sind, breitet sich der Brand aus - sind Sie aber zu schnell, kippt das ganze Wasser aus dem Eimer.

Sie können nur den Straßen folgen (hier und dort aber mal eine Ecke abkürzen) und über die Asche der Gebäude gehen, wo das Feuer gelöscht worden ist. Wenn Sie **großes** Pech haben, können Sie keines der brennenden Häuser erreichen und müssen einfach warten, bis das Feuer in Reichweite kommt.

Der Eimer kann nur gefüllt werden, wenn man direkt an der Themse steht, brennende Häuser sind nur zu löschen, wenn man vor ihnen steht. Ein (sehr großer) Eimer kann acht Brände löschen - der Inhaltmesser in der oberen linken Bildschirmecke zeigt genau an, wieviel Wasser der Eimer enthält.

Sie bewegen sich durch Drücken der Cursortasten (5, 6, 7 und 8), füllen den Eimer mit Taste 'f' und gießen das Wasser aus mit Taste 'g'.

Der einzige Haken: Sie haben 120 Sekunden Zeit, die Aufgabe zu erfüllen - die Holzhäuser fangen so schnell zu brennen an und Funken von brennenden Gebäuden werden vom kräftigen Wind weit getragen.

```
 1 REM  Great Fire of London
 2 REM ©1982 Phipps Associates
10 RESTORE : RANDOMIZE : CLEAR
```

```
 100 GO SUB 7000                          Anweisungen
1000 LET ls=limit                         Zeitlimit
1010 GO SUB 8000: GO SUB 8300             Plan zeichnen
1020 PRINT #f;AT 0,0;CHR$ 145;"
zeigt den 1.Brand"'"Fortsetzung:
 Taste druecken";
1030 IF NOT LEN INKEY$ THEN GO T          auf Taste warten
O 1030
1040 GO SUB 7800                          Meldungszeile frei
1100 POKE 23672,0: POKE 23673,0           Uhr in Gang setzen
2000 GO SUB 7500: IF s<1 OR NOT           Zeit überprüfen
a THEN GO TO 5000
2005 IF tm-s>4 THEN GO SUB 7800           Meldungen gelöscht
2010 LET m$=INKEY$: IF NOT LEN m          nach 4 sec.
$ THEN GO TO 2500                         auf Aktion prüfen
2015 LET dm=m-lm: LET lm=m                Zeit seit letzter
2020 LET dx=(yx>1)*(m$="5")*-1+(          Bewegung horizontal
yx<31)*(m$="8")
2030 LET dy=(yy<20)*(m$="6")+(yy          Bewegung vertikal
>1)*(m$="7")*-1
2040 LET c=ATTR (yy+dy,yx+dx)             Bildschirm prüfen
2050 IF c-INT (c/8)*8<>road THEN          Bewegung zulässig?
 GO TO 2500
2060 PRINT AT yy,yx; PAPER clr;           Eimer bewegen
INK clr; OVER t;y$
2070 LET yy=yy+dy: LET yx=yx+dx
2080 PRINT AT yy,yx; PAPER clr;
INK clr; OVER t;y$
2100 IF m$<>"f" THEN GO TO 2200           prüfen -füllen-
2110 LET b=water: GO SUB 7600: I          ist Wasser da?
F NOT v THEN GO TO 2500
2120 FOR y=0 TO 7: PRINT AT y,1;          Eimer füllen
 INK water; PAPER bg;CHR$ 144: N
EXT y
2130 LET w=8                              als voll setzen
2140 GO TO 2500                           fortsetzen
2200 IF m$<>"p" THEN GO TO 2300           prüfen -gießen-
2230 FOR w=w-1 TO 0 STEP -1               w Einheiten gießen
2235 PRINT AT 7-w,1; PAPER bg;"           Meßgerät
"                                         zurücksetzen
2240 LET b=fire: GO SUB 7600: IF          Brände löschen
 v THEN PRINT AT y,x; INK road;
PAPER fg;" ": LET m$="GLUG.": GO
 SUB 7700: LET q=q+1: LET a=a-1
2250 NEXT w: LET w=0
2260 GO TO 2500
2500 LET x=RND*30>dm: IF x AND w
 THEN LET w=w-1: PRINT AT 7-w,1;
```

```
 PAPER clr;" "
2600 LET r=(limit-s)/limit
2602 IF a=-1 THEN LET a=0: LET y
=ply: LET x=plx: GO TO 2700
2605 IF RND>EXP (a-4) THEN GO TO
 2000
2610 LET x=1+INT (plx-15*r+RND*3
0*r): LET y=1+INT (ply-r*10+RND*
20*r)
2620 IF x<0 OR y<0 OR x>31 OR y>
19 THEN GO TO 2000
2630 LET c=ATTR (y,x): IF c-INT
(c/8)*8<>fg THEN GO TO 2000
2700 PRINT AT y,x; INK fire; PAP
ER road;f$: LET m$="FEUER!!!": G
O SUB 7700
2720 LET a=a+1
2999 GO TO 2000
5000 REM end game
5002 FOR x=30 TO -30 STEP -1: BE
EP .005,x: NEXT x
5005 IF LEN INKEY$ THEN LET m$="
OK, OK, heb den Finger auf!": GO
 SUB 7700: GO TO 5005
5006 GO SUB 7800: PAUSE 50
5010 IF a>1 THEN GO TO 5100
5020 PRINT #f;AT 0,0; FLASH t;"B
RAVO!"; FLASH f;" Du hast alles
geloescht!"'"London ist gerettet
!"
5030 GO TO 5500
5100 PRINT #f;AT 0,0;"Du hast ";
q;" Feuer geloescht"'"aber ";a;"
 brennen lassen!"
5110 IF a<10 THEN PRINT #f;AT 2,
0;"London wird das ueberleben!"
5120 IF a>=10 AND a<=20 THEN PRI
NT #f;AT 2,0;"London ist voellig
 ruiniert!"
5130 IF a>20 THEN PRINT #f;AT 2,
0;"London? Nie gehoert davon."
5500 PAUSE 4e4
5510 LET m$="Noch ein Spiel?": G
O SUB 9000: IF CHR$ CODE a$="J"
THEN GO TO 1000
5520 GO TO 9999
7000 REM initialise
7010 DEF FN h(x)=INT (x/256)
```

Brandradius

Brand wird größer

neuer Brandort

Hausnummer brennt

Wiederholungsschleife

Lärm Ende

Meldungszeile frei

```
7020 DEF FN l(x)=x-256*FN h(x)
7030 LET t=1: LET f=0
7040 LET s=f
7050 GO SUB 8100
7060 LET limit=120
7100 PAPER 6: BORDER 6: INK 2: C
LS : GO SUB 7300
7110 PRINT CHR$ 144; INK f;"  Da
s Grossfeuer von London   "; INK
 2;CHR$ 144
7120 GO SUB 7300
7130 PRINT AT 4,0;" Du hast ";li
mit;" Sekunden Zeit, um"'"das Fe
uer zu loeschen, das in"'"Puddin
g Lane ausgebrochen ist."'"Aber
es greift um sich..."
7140 PRINT "Benutze zur Bewegung
 die Kursor-"'"Tasten, ""f"", um
 den Eimer zu"'"fuellen, und ""p
"", um ihn am"'"Brandherd auszul
eeren."
7150 PRINT "Ungluecklicherweise
schwappt"'"desto mehr Wasser aus
 dem Eimer,"'"je schneller Du la
eufst.Die"'"Anzeige oben links z
eigt Dir,"'"wieviel Wasser Du no
ch hast."
7160 PRINT "Du kannst den Eimer
nur fuellen,"'"wenn Du direkt an
 der Themse"'"stehst, und nur Fe
uer in Dei="'"ner unmittelbaren
Umgebung"'"loeschen."
7170 LET m$="Fortsetzung: Taste
druecken": GO SUB 9000
7180 CLS : PRINT "Ein Eimer kann
 bis zu acht"'"Feuer loeschen.Ac
hte darauf,"'"wieviel Wasser Du
noch hast."
7190 PRINT "Du kannst nur auf de
n Strassen"'"und der Asche der v
erbrannten"'"Haeuser laufen, dar
fst aber"'"Ecken schneiden."
7200 PRINT "Die Geschichte geht
ihren eige="'"nen Gang - aber hi
er hast Du"'"eine reelle Chance,
 sie zu"'"aendern..."
7210 GO SUB 9000
7220 RETURN
```

Grafik definieren
Zeitlimit 2 Minuten

```
7300 INK 2: PAPER 6:
7310 FOR x=1 TO 32: PRINT CHR$ 1
44;: NEXT x
7320 RETURN
7500 REM show time
7505 LET m=PEEK 23672+256*PEEK 2    Zähler für Ticken
3673
7510 LET s=limit-INT (m/50)         verbleib. Sekunden
7520 PRINT AT 21,28; INK clr; PA
PER fg; INVERSE t;s;" ";
7530 IF s<>ls THEN BEEP .10,-8:     jede Sekunde ein Ton
LET ls=s
7540 RETURN
7600 REM check for object
7605 GO SUB 7500                    Zeit prüfen
7610 FOR x=yx-1 TO yx+1: FOR y=y    Umgebung absuchen
y-1 TO yy+1
7620 IF y<>yy OR x<>yx THEN LET
c=ATTR (y,x): LET v=(b=(c-INT (c
/8)*8)): IF v THEN RETURN
7630 NEXT y: NEXT x: LET v=0        v=0 ist nicht gefunden
7640 RETURN
7700 REM messages                   zeig m$ auf Zeile 23
7710 GO SUB 7800                    Zeile 23 frei
7720 PRINT #f;AT 1,0;m$;            zeige m$ an
7730 LET tm=s                       Zeit bewahren
7740 RETURN
7800 REM clear message
7810 PRINT #f;AT 0,0,,,,
7820 RETURN
8000 REM draw map
8005 LET bg=4: LET fg=0: LET roa
d=6: LET water=5: LET clr=8: LET
 fire=2
8010 PAPER bg: INK fg: BORDER bg
: CLS
8020 INK water: FOR y=0 TO 23: P
LOT 0,y: DRAW 255,0: NEXT y
8030 INK road: PAPER bg
8040 FOR x=32 TO 47: PLOT x,0: D
RAW 0,175: NEXT x
8050 FOR y=24 TO 39: PLOT 0,y: D
RAW 255,0: NEXT y
8060 FOR y=120 TO 135: PLOT 255,
y: DRAW -208,0: NEXT y
8070 FOR x=168 TO 183: PLOT x,17
5: DRAW 0,-96: NEXT x
8080 FOR x=96 TO 111: PLOT x,40:
```

```
 DRAW 0,80: NEXT x
8090 FOR x=0 TO 15: PLOT x+240,4
0: DRAW -196-x,64+x: NEXT x: FOR
 y=0 TO 15: PLOT 48,y+104: DRAW
207,-64: NEXT y
8095 RETURN
8100 REM special characters
8105 DATA 3
8110 DATA "a",170,85,170,85,170,
85,170,85
8120 DATA "b",195,231,126,60,60,
126,231,195
8130 DATA "c",24,36,66,255,255,1
26,60,60
8200 READ n
8210 FOR x=1 TO n: READ c$: FOR
y=0 TO 7: READ r: POKE USR c$+y,
r: NEXT y: NEXT x
8220 RETURN
8300 REM tidy screen
8305 LET plx=11: LET ply=13
8310 PRINT AT ply,plx; INK 2;CHR
$ 145
8320 LET y$=CHR$ 146: LET f$=CHR
$ 144
8330 LET yx=22: LET yy=0
8340 PRINT AT yy,yx; OVER t;y$
8350 PRINT AT 21,23; INK clr; PA
PER fg; INVERSE t;"Time:";
8360 PRINT AT 0,0; INK fg;">";AT
 7,0;">"
8370 LET dm=f: LET lm=f: LET tm=
f: LET w=f: LET s=f: LET q=0: LE
T a=0
8380 LET a=-1
8390 RETURN
9000 REM input
9010 INPUT (m$);" "; LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
")): NEXT x
9030 RETURN
9999 PAPER 7: BORDER 7: INK 0: C
LS : STOP
```

Kleine Versuche

Ein Nachteil bei einem Computer mit 16K RAM ist der, daß alle Programme dazu neigen, ziemlich lang zu werden. Damit Ton und Bild eindrucksvoll werden, nimmt der Code einen Umfang an, den man einem normalen Menschen kaum zumuten kann.

Hier haben wir eine Auswahl kleinerer Programme für den Anfang. Sie rufen keine schönen Bildschirmdisplays oder wunderbare Sonaten hervor, aber jedes ist anders. Sie finden einige Spiele, einen Quiz, ein Lehrprogramm für Schüler und anderes mehr.

Ich habe die Bemerkungen auf ein Mindestmaß beschränkt, weil es nirgends kompliziert wird.

Zielscheibe

Sie laufen an einem stehenden Ziel vorbei und müssen versuchen, es zu treffen. So einfach ist das gar nicht! Sie können schießen mit der Taste "8". Sie erscheinen an der linken oberen Ecke, das Ziel befindet sich rechts. Drücken Sie nur ab, wenn Sie glauben, dem Ziel genau gegenüber zu sein.

```
  10 LET n=0: LET m=0
  20 LET t=INT (RND*22): LET x=I
NT (RND*6)+26
  30 PRINT AT t,x;"O"                    Ziel anzeigen
 100 LET n=n+1
 110 FOR y=0 TO 21
 120 IF y THEN PRINT AT y-1,0;"         ein Leerraum
"
 130 PRINT AT y,0;">";                  sich anzeigen
 140 IF INKEY$="8" THEN GO TO 20        feuern?
0
 150 NEXT y
 160 PRINT AT 21,0;" "                  ein Leerraum
 170 GO TO 100                          wiederholen
 200 FOR z=0 TO x: PRINT AT y,z;        schießen
">";: NEXT z
 210 IF y=t THEN GO TO 300              Ziel getroffen?
 220 FOR z=0 TO x: PRINT AT y,z;        leere Spur
" ";: NEXT z
 230 LET m=m+1                          Fehlschußzähler
 240 GO TO 100
```

```
 300 PRINT AT t,x;"*": PRINT AT
19,0;"Du hast es erwischt!"
 310 PRINT "Du brauchtest ";m;"
Schuss in ","den Runden"
 320 INPUT "Nochmal? "; LINE a$:
 IF CHR$ CODE a$="j" THEN RUN
```

Enigma-Variationen

Dieses Programm verwandelt jeden String in einen Geheimcode, den nur Sie entschlüsseln können. Der Schlüssel kann jede Zahl sein, die Sie wünschen. Sie müssen sie vorher eingeben. Dann können Sie mit dieser Chiffre Mitteilungen ver- oder entschlüsseln. Jeder Schlüssel erzeugt einen anderen Code, so daß Sie, wenn Gegner einen Code knacken, einfach einen anderen Schlüssel nehmen.

Das Programm wird aber dadurch so großartig, daß beim Verschlüsseln des Strings eine variable Zufallsmethode verwendet wird. Wenn Sie "AAAA" eingeben, erhalten Sie also nicht für jedes Vorkommen von "A" dieselben vier Buchstaben. Dadurch ist der Schlüssel kaum zu knacken.

```
  10 INPUT "Schluessel ";k
  20 RANDOMIZE k: LET s=0
  30 LET a$="Verschluesseln (V)
             Entschluesseln (E)"
: GO SUB 9000: IF Ende THEN GO T
O 9999
  40 IF CHR$ CODE a$="E" THEN LE
T s=1: PRINT "Wir entschluesseln
"
  50 LET w$="ABCDEFGHIJKLMNOPQRS
TUVWXYZ 0123456789$#.,?": LET w=
LEN w$
  60 LET a$="Text eingeben:": GO
 SUB 9000: IF Ende THEN GO TO 99
99
  70 LET a=LEN a$
  80 FOR n=1 TO a
  90 LET x$=a$(n)
 100 GO SUB 9300
 110 IF s=1 THEN GO TO 160
 120 LET y=(INT (RND*w))+1+x
```

```
 130 IF y>w THEN LET y=y-w: GO T
O 130
 140 PRINT w$(y);
 150 NEXT n: GO TO 60
 160 LET y=x-(INT (RND*w)+1)
 170 IF y<1 THEN LET y=y+w: GO T
O 170
 180 GO TO 140
9000 REM Zeileneingabe
9010 INPUT (a$+" ");  LINE a$: LE
T Ende=NOT LEN a$
9020 FOR x=1 TO LEN a$
9030 LET a$(x)=CHR$ (CODE a$(x)-
32*(a$(x)>="a" AND a$(x)<="z"))
9035 NEXT x
9040 RETURN
9300 FOR x=1 TO LEN w$-LEN x$+1:
 IF x$=w$(x TO x+LEN x$-1) THEN
RETURN
9320 NEXT x: LET x=w
9330 RETURN
```

Quizspiel

Haben Sie sich schon mal als berühmten Quizmaster gesehen? Mit diesem Programm können Sie anderen Leuten Fragen zum Beantworten aufgeben. Der Spectrum übernimmt die ganze Arbeit. So prüft er etwa, ob die Antwort richtig ist, und führt Buch über den Punktestand.

Das Programm enthält fünf Probefragen, damit Sie sehen, wie Sie Ihr eigenes Quizprogramm zusammenstellen können. Geben Sie die Zahl der Fragen in Zeile 1010 ein. Die Fragen und Antworten erscheinen dann in den folgenden Zeilen. Es spielt keine Rolle, wie die Antwort eingegeben wird, das Programm erzwingt stets Übereinstimmung in Großbuchstaben. Die Antwort braucht nur in einem String enthalten zu sein, weil das Programm die ganze Antwort nach dem richtigen Wort absucht. Den inneren Ablauf dieser Routine findet man im späteren Abschnitt 'Subroutinen' erklärt.

```
  10 REM quiz
  20 RESTORE 1000
  30 READ n
  40 DIM s(2)
 100 FOR q=1 TO n
 105 PRINT TAB 10;"*** QUIZ ***"
 110 READ q$,a$: PRINT AT 10,0;q
$
 120 LET u$=a$: GO SUB 9020: LET
 x$=a$
 130 LET a$="Antwort:": GO SUB 9
000
 140 LET w$=a$: GO SUB 9300
 150 LET s=(x<>0)+1
 160 LET s(s)=s(s)+1
 170 IF s=2 THEN PRINT AT 21,0;"
Richtig!": GO TO 200
 180 PRINT AT 20,0;"Falsch. Die
Antwort ist:"'TAB 4;u$
 200 LET a$="Druecke ENTER": GO
SUB 9000
 210 CLS
 220 NEXT q
 300 PRINT AT 10,0;"Von ";n;" Fr
agen,"
 310 PRINT " hast du ";s(2);" ri
chtig,"
 320 IF NOT s(1) THEN PRINT " Da
s ist nicht schlecht!"
 330 IF s(1) THEN PRINT " und ";
s(1);" falsch."
 340 STOP
1000 DATA 5: REM Nummer der Frag
en
1010 DATA "Wie heisst die Haupts
tadt von   Frankreich?","Paris"
1020 DATA "Wie viele Beine hat e
ine Spinne?","8"
1030 DATA "Wie hiess Shakespeare
 mit Vor-  namen? (keine Abkuerz
ungen!)","William"
1040 DATA "Wie ist der Name des
ersten     Sinclair-Computers?",
"MK14"
1050 DATA "Was ist des Name des
Mikro-Chip im Spectrum","Z80"
9000 REM Zeileneingabe
9010 INPUT (a$+" "); LINE a$: LE
```

Fragentabelle
Zahl der Fragen
Punkte

Frage anzeigen

richtige Antwort
umwandeln

Antwort prüfen
ist sie richtig?
Punktestand
ergänzen

```
T ende=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
" AND a$(x)<="z")): NEXT x
9030 RETURN
9300 REM x=instr(w$,x$)
9310 FOR x=1 TO LEN w$-LEN x$+1:
 IF x$=w$(x TO x+LEN x$-1) THEN
RETURN
9320 NEXT x: LET x=0
9330 RETURN
```

Reaktionstester

Wenn Sie den Verdacht haben, daß jemand alt und zittrig wird – dieses Programm kann es entweder bestätigen und widerlegen. Setzen Sie das ahnungslose Opfer, Verzeihung – die Person vor dieses Programm, um festzustellen, wie lange sie braucht, um auf die Aufforderung zu reagieren, eine Taste zu drücken. Eine Reaktionszeit von 0.25 Sekunden ist ungefähr Durchschnitt, länger als 0.5 ist mehr als schwach, und man sollte die Person dann nicht an ein Lenkrad lassen!

```
  10 PRINT TAB 8;"Reaktionsteste
r"
  20 PRINT AT 20,0;"Druecke eine
 Taste wenn du       ""JETZT"" si
ehst"
 100 PAUSE 100+INT (RND*250)
 110 IF LEN INKEY$ THEN GO TO 20
0                                  auf Schwindler
                                   prüfen
 120 PRINT AT 10,13;"** JETZT **
"
 130 POKE 23673,0: POKE 23672,0    Uhr in Gang setzen
 140 IF NOT LEN INKEY$ THEN GO T   auf Taste warten
O 140
 150 LET t=PEEK 23672+256*PEEK 2   neue Zeit holen
3673
 160 LET t=t/50                    in Sek. umrechnen
 170 PRINT OVER 0;AT 21,0;"Du br
auchtest ";t;" Sekunden."
 175 IF LEN INKEY$ THEN GO TO 17
5
```

```
 180 INPUT "Druecke ENTER...";  L
INE a$
 190 CLS : GO TO 10
 200 PRINT AT 18,14; INVERSE 1;"
 BETRUEGER "; INVERSE 0
 210 PRINT AT 21,0,,
 220 GO TO 180
```

Anagramme

Von diesem Programm werden Kreuzworträtselliebhaber begeistert sein - es erzeugt Anagramme von jedem eingegebenen Wort. Die angezeigten Ergebnisse dienen eigentlich dazu, Sie anzuregen, daß Sie Wörter sehen, die man bilden kann, weil ein Wort mit 7 Buchstaben 5040 verschiedene Permutationen möglich macht, was viel zu viel ist, als daß der Normalmensch sich hinsetzen und das ansehen könnte. Wenn nach ein, zwei Bildschirmen voll sich kein Wort anbietet, spricht etwas dafür, daß Sie als Grundlage für Ihre Anagramme die falschen Wörter verwendet haben.

```
  10 INPUT "Wort eingeben "; LIN
E a$
  20 IF LEN a$=0 THEN STOP          stop ohne Input
  30 DIM n(LEN a$): LET n=LEN a$
  40 GO SUB 100                     Anagramm erzeugen
  50 FOR z=1 TO n: PRINT a$(n(z)
);: NEXT z
  60 PRINT ': GO TO 40
 100 REM n verschiedene Zahlen e
rzeugen
 110 LET c=1: FOR w=1 TO n
 120 LET x=INT (RND*n)+1
 130 IF c=1 THEN GO TO 170
 140 FOR a=1 TO (c-1)
 150 IF x=n(a) THEN GO TO 120
 160 NEXT a
 170 LET n(w)=x: LET c=c+1
 180 NEXT w
 190 RETURN
```

Morsen üben

Pfadfinder, Achtung! Hier ist alles, was ihr braucht, um richtig Morsen zu lernen. Wenn ihr eine Taste drückt, wird euch der Morsecode 'vorgespielt'. Ihr könnt das auf Kassette aufnehmen und einem Freund (Feind?) schicken.

Ein nützlicher Hinweis: Wenn ihr den Ton lauter hören wollt, dann zieht den Stecker aus der Buchse 'EAR' an eurem Kassettenrecorder und drückt auf 'PLAY' (zusammen mit PAUSE, wenn euer Recorder das hat). Der Ton kommt dann aus dem Lautsprecher des Recorders und ist viel lauter als der eingebaute Pieper. Wenn ihr an die 100-Watt-Stereoanlage anschließen könnt, ist das freilich noch besser ...

```
   5 GO SUB 200                                  Morsefolge definieren
  10 LET w=CODE INKEY$                           Buchstaben holen
  20 IF w=0 THEN GO TO 10                        auf Zeichen warten
  25 PRINT CHR$ w;                               anzeigen...
  30 IF w>=48 AND w<=57 THEN LET
 w=w-48: GO TO 60
  40 IF w>=65 AND w<=91 THEN LET
 w=w-55: GO TO 60
  50 IF w>=97 AND w<=123 THEN LE
T w=w-87: GO TO 60
  55 GO TO 10
  60 LET s=m(w+1)                                Folge holen
  70 IF s=0 THEN GO TO 10                        alles fertig
  80 LET c=s-(INT (s/10)*10)                     Tonfolge
  90 IF c=1 THEN BEEP .2,0: GO T
O 110
 100 IF c=2 THEN BEEP .6,0
 110 LET s=INT (s/10): FOR x=1 T
O 12: NEXT x
 120 GO TO 70
 200 DIM m(36)
 210 DATA 22222,22221,22211,2211
1,21111,11111,11112,11122,11222,
12222
 220 DATA 21,1112,1212,112,1,121
1,122,1111,11,2221,212,1121,22,1
2,222,1221,2122,121,111,2,211,21
11,221,2112,2212,1122
 260 FOR x=1 TO 36: READ m(x): N
EXT x
 270 RETURN
```

Mathe-Lehrer

Sorgen wegen der Prüfung? Angst, daß es mit dem Rechnen nicht klappt? Keine Bange - dieses Programm prüft auf Teufelkommraus und sagt kein Wörtchen, auch wenn alle Antworten falsch sind.

Man wählt die höchstwertige Zahl in der Frage, damit kleine Kinder mit Zahlen unter 10 spielen können, während Erwachsene mit Zahlen bis zu einer Million zurechtkommen - oder?

Als nächstes wählt man die Art der Aufgabe, die man lösen möchte: Addition, Subtraktion, Multiplikation oder Division. Das Programm liefert nur Aufgaben, die als Lösungen ganze Zahlen verlangen, aber Sie können das sehr leicht verhindern, wenn Sie wollen.

Jede Prüfung legt zehn Fragen vor, dann erfährt man seinen Punktestand. Sie können jede Frage überspringen und brauchen dazu nur ENTER zu drücken, wenn Sie die Antwort eingeben sollen. Falls Sie einen Drucker haben, können Sie eine Abschrift der Prüfung erhalten, wenn sie abgeschlossen ist, weil alle Fragen zusammen mit Ihren und den richtigen Lösungen auf dem Bildschirm bleiben.

```
  10 PRINT TAB 8; INVERSE 1;"**
Mathepruefer **"; INVERSE 0
  20 INPUT "Was ist die groesste
 Zahl ";max
  30 DEF FN q$()=STR$ n1+" "+r$+
" "+STR$ n2
  40 DEF FN a()=VAL FN q$()
  50 DIM p$(3,2): LET p$(1)="X":
 LET p$(2)="OK": LET p$(3)="?"
 100 LET a$="Welche Regel willst
 du?(+-*/)": GO SUB 9000: IF end
 THEN STOP
 110 LET w$="+-*/": LET x$=a$: G
O SUB 9300
 120 LET Regel=x: IF x=0 THEN GO  Regelnummer
 TO 100                           festhalten
 130 LET r$=a$                    Operator festhalten
 140 LET t=0: DIM s(3)            Punktezählung
 200 IF t=10 THEN GO TO 600       Test vollständig?
 205 LET t=t+1                    Testzähler
```

```
 210 LET n1=INT (RND*max)+1: LET
 n2=INT (RND*max)+1
 220 IF Regel=1 OR Regel=3 THEN
GO TO 300                          addieren und
                                   multiplizieren
 230 IF n2>=n1 THEN GO TO 210
 240 IF Regel=2 THEN GO TO 300     subtrahieren
 245 LET n2=INT (RND*max/3)+1      Divisor finden
 250 LET n1=INT (n1/n2)*n2
 260 IF n2=n1 THEN GO TO 210
 300 LET a$="Wieviel ist "+FN q$   Frage stellen
()+"?"
 310 LET a=2: GO SUB 9000: IF en
d THEN GO TO 400
 320 GO SUB 9400: IF x THEN LET
Loesung=VAL a$( TO x)
 325 IF NOT x THEN GO TO 300       auf Richtigkeit prüfen
 330 LET a=(Loesung=FN a())        ist Antwort richtig?
 400 RESTORE 9900+a: READ m$       Antwortstring
 410 PRINT AT 20,0;m$              Antwort anzeigen
 500 PRINT AT t+1,0;FN q$();" =
";: IF a<>2 THEN PRINT Loesung;
 510 PRINT TAB 19;p$(a+1);
 520 IF a<>1 THEN PRINT " Loesun
g=";FN a()
 525 LET s(a+1)=s(a+1)+1           Punktestand ergänzen
 530 LET a$="Zur naechsten Frage
n ENTER      druecken": GO SUB 9
000
 535 OVER 0
 540 PRINT AT 20,0,,,,             Meldungen weg
 550 GO TO 200                     nächste Frage
 600 PRINT AT 16,0;"Von den ";t;
" Fragen,"
 610 IF s(3) THEN PRINT " hast d
u beantwortet ";t-s(3)
 620 PRINT " du hast ";s(2);" ri
chtig."
 630 IF s(1) THEN PRINT " Du has
t ";s(1);" falsch."
 640 IF NOT s(1) THEN PRINT " Da
s ist recht gut!"
 650 LET a$="Noch ein Versuch? (
J/N)": GO SUB 9000: IF CHR$ CODE
 a$="J" THEN RUN
 660 STOP
9000 REM Zeileneingabe
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
```

```
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
" AND a$(x)<="z")): NEXT x
9030 RETURN
9300 REM x=instr(w$,x$)
9310 FOR x=1 TO LEN w$-LEN x$+1:
 IF x$=w$(x TO x+LEN x$-1) THEN
RETURN
9320 NEXT x: LET x=0
9330 RETURN
9400 REM x=numptr(a$)
9410 FOR x=1 TO LEN a$: IF a$(x)
>="0" AND a$(x)<="9" THEN NEXT x
9420 LET x=x-1
9430 RETURN
9900 DATA "Falsch. Es muss heiss
en "+STR$ FN a()
9901 DATA "Richtig!"
9902 DATA "Zu schwer? Die Loesun
g ist "+STR$ FN a()
```

Benutzergrafik

Ich habe verschiedene Programme gesehen, mit denen man eigene Grafik definieren kann, aber alles, was ich wollte, hat noch keines geliefert. Die meisten enthalten die unverzichtbare Einrichtung, mit den Cursortasten auf einem Gitter herumzufahren und Punkte nach Wunsch zu setzen oder wegzunehmen, aber nur wenige sind so weit gegangen, einen DATA-Befehl mit aufzunehmen, um die Grafik in folgenden Programmen zu definieren oder zuzulassen, daß ganze Reihen und Spalten im Gitter umgewandelt werden.

Aber genug geredet - wie funktioniert das? Wenn das Programm geladen und gefahren wird, zeigt es den laufenden Vorrat an Benutzergrafik und fordert auf, ein Zeichen zu ändern. Wenn Sie in diesem Stadium 'nein' antworten, fragt das Programm, ob Sie die Grafik auf Kassette sichern wollen. Das kann nützlich sein, wenn Ihre Programme recht groß sind, weil man dann die Befehle nicht zu behalten braucht, mit denen die Grafikzeichen gesetzt werden.

Wenn Sie 'ja' antworten, fordert das Programm auf, den entsprechenden Buchstaben zu ändern. Das Muster des Zeichens wird dann in einem riesengroßen 8x8-Gitter auf dem Bildschirm angezeigt, über dem oberen linken Quadrat ein 'Fadenkreuz'-Cursor. Der untere Bildschirmteil zeigt ein Menü der Einzelzeichen-Befehle, die Sie in diesem Stadium eingeben können:

X = fertig	B = alles leer
R = Reihe umwandeln	S = Spalte umwandeln
Ø = setzen/löschen	5, 6, 7, 8 = bewegen
Q = belassen	V = verstecken/nicht verstecken
A = anzeigen	

Das Fadenkreuz wird mit den Cursortasten zu einem beliebigen Quadrat geführt, ein Quadrat durch Drücken von "0" von Schwarz zu Weiß umgewandelt (oder umgekehrt). Das Gitter können Sie mit "B" beseitigen, die Quadrate einer ganzen Reihe oder Spalte durch Drücken von "R" oder "C" umwandeln.

Sie können eine Anzeige durch "A" erhalten, aber zunächst wollen Sie das Fadenkreuz 'verstecken'. Dazu dient der "V"-Befehl. Erneuter Druck auf "V" zeigt das Fadenkreuz wieder an.

"E" (für "Ende") gibt Ihnen den alten Bildschirm zurück, ohne das Zeichen zu verändern, während "X" (für 'Exit') das Zeichen im Grafikvorrat sichert. In diesem Stadium sehen Sie dann auch die Form eines DATA-Befehls, der in einem Programm dazu benützt werden kann, das Grafikzeichen direkt zu laden. Ein Programm, mit dem das geht, sieht so aus:

```
8000 REM load graphic from DATA
8010 DATA 2   (Zahl der zu definierenden Zeichen einfügen
8020 DATA "A",1,2,3,4,5,6,7,8   wie im Programm genannt
8030 DATA "B",1,2,3,4,5,6,7,8   wie im Programm genannt
8040 READ n
8050 FOR x=1 TO n: READ c$
8060 FOR r=0 TO 7: READ y: POKE
USR c$+r,y: NEXT r
8070 NEXT x
```

Mit einem Programm wie dem folgenden erhalten Sie Gelegenheit, beim Entwerfen der Zeichen mutig zu werden - Sie brauchen es nicht zuerst auf Papier zu zeichnen, weil Sie es rascher und wirksamer auf den Bildschirm zeichnen können.

Wenn Sie das Programm eingegeben haben, sichern Sie es mit dem Befehl:

SAVE "TABLET" LINE 1

```
   1 REM   User Graphic Tablet
   2 REM @Phipps Associates 1982
   3 REM  V1.0 29th July '82
  20 GO SUB 2000                          Grafik auflisten
  30 LET a$="Willst Du ein Zeich
en aendern?": GO SUB 9000: IF CH
R$ CODE a$<>"J" THEN GO TO 500
  50 LET i$="FALSCHE EINGABE - "          Strings
: LET e$=""                               initialisieren
  60 LET a$=e$+"Welchen Buchstab          Buchstaben holen
en?": GO SUB 9000: IF NOT CODE a
$ THEN GO TO 20
  70 IF LEN a$<>1 THEN LET e$=i$          bestätigen
: GO TO 60
  90 IF CODE a$<CODE "A" OR CODE
 a$>CODE "U" THEN LET e$=i$: GO
TO 60
 100 GO SUB 1000                          Gitter zeichnen
 110 GO SUB 1100                          Position Fadenkreuz
 120 LET dx=0: LET dy=0: LET x=0
: LET y=0
 130 DIM c(8,8)                           Grafikgitter
 140 GO SUB 2400                          Muster zeichnen
 150 GO SUB 2600                          Optionen anzeigen
 200 GO SUB 1300: IF ok THEN GO
SUB 2800: GO SUB 2900
 210 GO TO 20                             Wiederholung
 500 LET a$="Willst Du ihn abspe
ichern?": GO SUB 9000: IF CHR$ C
ODE a$<>"Y" THEN GO TO 9999
 510 LET a$="Name des Bandes?":
GO SUB 9000: IF LEN a$ THEN SAVE
 a$CODE USR "a",21*8
 520 GO TO 9999
1000 REM draw grid
1010 CLS : LET sx=96: LET sy=136          Position auf Schirm
1020 FOR n=0 TO 8: PLOT sx,sy-n*
8: DRAW 64,0: NEXT n
1030 FOR n=0 TO 8: PLOT sx+n*8,s
y: DRAW 0,-64: NEXT n
1040 RETURN
1100 REM
```

```
1110 LET cx=sx+4: LET cy=sy-4
1200 REM draw cursor
1210 PLOT OVER 1,40,cy: DRAW OVE
R 1,160,0
1220 PLOT OVER 1,cx,160: DRAW OV
ER 1,0,-100
1230 RETURN
1300 REM move cursor
1305 IF INKEY$<>"" THEN GO TO 13
05                                warten auf keine Taste
1310 LET x$=INKEY$: IF NOT LEN x
$ THEN GO TO 1310
1315 LET x$=CHR$ (CODE x$-32*(CO  Tastencodewert finden
DE x$>96))
1320 LET w$="05678RCBXQHP": LET
dx=0: LET dy=0
1325 GO SUB 9300: IF NOT z THEN   ist Code gültig?
GO TO 1300
1330 GO TO 1340+(z-1)*10          zu Aktionstaste gehen
1340 GO SUB 2200: GO TO 1300      inneren Plan halten
1350 LET dx=(x>0)*-1: GO TO 1500  nach links bewegen
1360 LET dy=(y<7): GO TO 1500     nach unten bewegen
1370 LET dy=(y>0)*-1: GO TO 1500  nach oben bewegen
1380 LET dx=(x<7): GO TO 1500     nach rechts bewegen
1390 LET s=x: FOR x=0 TO 7: GO S
UB 2200: NEXT x: LET x=s: GO TO
1300
1400 LET s=y: FOR y=0 TO 7: GO S  Spalte umwandeln
UB 2200: NEXT y: LET y=s: GO TO
1300
1410 LET lx=x: LET ly=y: FOR y=0  Gitter löschen
 TO 7: FOR x=0 TO 7: IF c(y+1,x+
1) THEN PRINT AT 5+y,12+x; INVER
SE 1; OVER 1;" "
1415 NEXT x: NEXT y: LET x=lx: L
ET y=ly: DIM c(8,8): GO TO 1300
1420 GO SUB 1200: LET ok=1: RETU  fertig OK
RN
1430 LET ok=0: RETURN             Ende – nicht OK
1440 GO SUB 1200: GO TO 1300      Fadenkreuz verstecken
1450 COPY : GO TO 1300            Schirm anzeigen
1500 GO SUB 1200                  Fadenkreuz löschen
1510 LET x=x+dx: LET y=y+dy       Position aktualisieren
1520 LET cx=cx+dx*8: LET cy=cy-d
y*8
1530 GO SUB 1200                  Fadenkreuz neu zeichnen
1540 GO TO 1300
2000 REM Print current set
```

```
2010 CLS : PRINT "      G R A F
I K - S E T      "
2020 FOR x=0 TO 1
2030 PRINT AT 2,x*16;"alt      ne
u"
2040 PRINT AT 3,x*16;"___      ___
 "
2050 NEXT x
2060 FOR x=0 TO 1
2070 FOR c=0 TO 10
2080 IF x=1 AND c=10 THEN GO TO
2100
2090 PRINT AT 5+c,(x*16+1);CHR$
(65+x*11+c);AT 5+c,(x*16+9);CHR$
 (144+x*11+c);
2100 NEXT c
2110 NEXT x
2120 RETURN
2200 REM maintain internal pictu
re
2210 PRINT AT y+5,x+12; INVERSE
1; OVER 1;" "
2220 LET c(y+1,x+1)=(c(y+1,x+1)=
0)
2230 RETURN
2400 REM draw existing pattern
2410 FOR r=0 TO 7
2420 LET p=PEEK (USR a$+r)
2430 PRINT AT 5+r,12;
2440 GO SUB 2500
2450 NEXT r
2460 RETURN
2500 REM create array
2510 LET shr=128
2520 FOR a=0 TO 7
2530 LET z=INT (p/shr)
2540 LET p=p-z*shr
2550 LET shr=shr/2
2560 PRINT INVERSE (z<>0); OVER
1;" ";: LET c(r+1,a+1)=(z<>0)
2570 NEXT a
2580 RETURN
2600 REM help
2610 PRINT AT 0,0; PAPER 1; INK
6;"Gewaehler Buchstabe: ";a$
2620 PRINT AT 16,0;
2630 PRINT PAPER 1; INK 6;"X=fer
tig","B=alle weiss","R=Reihe aen
```

ein Leerraum

eine Reihe holen

Schirmposition

(Reihe erweitern und anzeigen

erforderliches Bit

8 Bits pro Reihe

Bitwert holen

nächste Bitposition

ein Leerraum

nächstes Bit

```
dern","C=Saeule aendern","0=Feld
 aendern","5,6,7,8=Kursor","Q=ei
nzeln lassen ","H=(un)sichtbar",
"P=Kopie drucken",,
2690 RETURN
2800 REM pack character
2805 PRINT AT 21,13; PAPER 2;" W
AIT "
2810 FOR r=0 TO 7                      8 Reihen
2820 LET b=0                           Reihenwert
2830 FOR x=0 TO 7: IF c(r+1,x+1)       Summe berechnen
 THEN LET b=b+2^(7-x)
2835 NEXT x
2840 POKE USR a$+r,b                   in RAM setzen
2850 NEXT r                            nächste Reihe
2860 RETURN
2900 REM create data statements
2905 PAPER 7: INK 0: BRIGHT 0
2910 FOR y=14 TO 21: PRINT AT y,
0;,,: NEXT y
2920 PRINT AT 15,0; PAPER 1; INK
 6;"Data Ausdruck Format:"
2930 PRINT '"DATA """;a$;""",";
2940 FOR r=0 TO 7                      8 Werte
2950 PRINT PEEK (USR a$+r);"," A       diese anzeigen
ND r<>7;
2960 NEXT r
2970 PRINT AT 21,0; PAPER 1; INK
 7;"Taste druecken (P=print)"
2980 LET x$=INKEY$: IF NOT LEN x
$ THEN GO TO 2980
2990 IF x$="P" OR x$="p" THEN CO
PY
2995 RETURN
9000 REM line input                    s. Subroutinen
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$
9030 LET a$(x)=CHR$ (CODE a$(x)-
32*(a$(x)>="a" AND a$(x)<="z"))
9040 NEXT x
9050 RETURN
9300 REM let z=instr(w$,a$)            s. Subroutinen
9310 FOR z=1 TO LEN w$-LEN x$+1:
 IF x$<>w$(z TO LEN x$+z-1) THEN
 NEXT z: LET z=0
9320 RETURN
9999 STOP
```

Muster

Dieser Teil bringt Ihnen eine Reihe kleiner Programme, die auf dem Bildschirm interessante Muster erzeugen. Das erste, was Nachbarn und Freunde immer fragen, ist: "Und was macht dann dieses ZX-Computerdings eigentlich?"

Daraufhin stürzen Sie zu Ihrem Stapel Bandkassetten, reißen das Programm 'hübsche Muster' heraus und fahren es. Programme dieser Art haben sonst nicht viel Bedeutung, außer, daß sie hübsche optische Effekte erzeugen.

Ich will mich mit Kommentaren nicht lange aufhalten. Geben Sie sie einfach ein und fahren Sie sie. Ich hoffe, Sie gefallen Ihnen, wenn schon sonst nichts dahinter ist.

```
   5 REM *** Pattern 1 ***
  10 OVER 1
  20 BORDER 0: PAPER 0: BRIGHT 1
  30 CLS
  40 LET w=30
 100 FOR L=0 TO 426+w
 105 INK INT (L/8)-(INT (INT (L/
8)/8)*8)
 110 IF L<427 THEN LET x=L: GO S
UB 8000: PLOT a1,b1: DRAW a2,b2
 120 IF L>=w THEN LET x=L-w: GO
SUB 8000: PLOT a1,b1: DRAW a2,b2
 125 IF LEN INKEY$ THEN GO TO 99
99
 130 NEXT L
 140 GO TO 100
8000 REM bestimme Zeilenposition
8010 LET a1=(x>171)*(x-171)
8020 LET b1=(171-x)*(x<172)
8030 LET a2=(x<256)*x+255*(x>255
)
8040 LET b2=172*(x<256)+(428-x)*
(x>255)
8050 LET a2=a2-a1: LET b2=b2-b1
8060 RETURN
9999 BRIGHT 0: BORDER 7: INK 0:
PAPER 7: STOP
```

```
   5 REM **** Pattern 2 ****
  10 CLS
 100 OVER 1
 110 LET p=INT (RND*8): LET i=IN
T (RND*8)
 120 IF p=i THEN GO TO 110
 130 PAPER p: INK i: BORDER p
 140 PRINT AT 0,0;: FOR x=0 TO 2
1: PRINT OVER 1;,,: NEXT x
 150 FOR x=23232 TO 23295: POKE
x,p*8+i: NEXT x
 200 LET x1=0: LET y1=0: LET h=1
76: LET w=256: GO SUB 8000
 300 LET x1=x1+INT (w/4): LET y1
=y1+INT (h/4)
 310 LET h=INT (h/2): LET w=INT
(w/2)
 315 IF h=0 AND w=0 THEN GO TO 4
00
 320 GO SUB 8000
 330 GO TO 300
 400 IF NOT LEN INKEY$ THEN GO T
O 100
 410 IF LEN INKEY$ THEN GO TO 41
0
 420 GO TO 9999
 999 STOP
8000 REM Diamant zeichnen
8005 LET h1=INT (h/2): LET w1=IN
T (w/2)
8010 FOR x=h1-1 TO 0 STEP -1
8020 PLOT x1,y1+h1+x: DRAW w1-1-
x,h1-1-x
8030 PLOT x1+w-1,y1+h1+x: DRAW x
-(w1-1),h1-1-x
8040 PLOT x1+w-1,y1+h1-1-x: DRAW
 x-(w1-1),x-(h1-1)
8050 PLOT x1,y1+h1-1-x: DRAW w1-
1-x,x-(h1-1)
8060 NEXT x
8070 RETURN
9999 BORDER 7: PAPER 7: INK 0: C
LS
```

```
  5 REM *** Pattern 3 ***
 10 BORDER 0: PAPER 0: CLS
 20 FOR r=2 TO 86 STEP 2
 30 CIRCLE PAPER INT (1*r/11);1
28,88,r
 40 NEXT r


  5 REM *** Pattern 4 ***
 10 FOR r=0 TO 999
 20 PLOT 128,88
 30 LET x=INT (RND*256): LET y=
INT (RND*176)
 40 DRAW OVER 1; INK (INT (RND*
8));x-128,y-88
 50 NEXT r


  5 REM *** Pattern 5 ***
 10 FOR x=0 TO 9999
 20 LET z=SIN (x*PI/10)*15
 30 PRINT INK (x-INT (x/8)*8);A
T 21,16-z;"*";AT 21,16+z;"*"
 40 RANDOMIZE USR 3582
 50 NEXT x
 60 GO TO 10
```

Winke und Tips

Dieses Kapitel vermittelt alle möglichen nützlichen Informationen. An einer oder mehreren Stellen sind sie in diesem Buch alle verwendet worden, aber hier habe ich mir die Zeit genommen, sie etwas genauer zu erklären.

Datenträger

Ohne den Microdrive, das Mini-Diskettenlaufwerk, harren wir erwartungsvoll der seltsamen Befehle auf der obersten Tastenreihe: OPEN #, CLOSE #, FORMAT, CAT und andere.

Sie können aber **jetzt** schon verwendet werden, um Ihnen ein paar recht unerwartete und leistungsstarke Programmierinstrumente liefern. Zuvor ein kurzer Blick in die inneren Abläufe des Spectrum-ROM (Nur-Lese- oder Festspeicher).

An den Spectrum kann man eine Reihe von Geräten anschließen - Drucker, Microdrive-Geräte, RS232-Schnittstelle - die von ROM als **Datenströme** betrachtet werden. Bevor Sie ein Gerät benützen können, muß es eine Stromnummer erhalten (Ø bis 15). Das wird erreicht mit dem Befehl OPEN #. Bestimmte Ströme werden als permanent offen angesehen - Keyboard, Bildschirm und Drucker. Sie werden gesetzt als Stromnummern Ø und 1 für die Tastatur, 2 für den Bildschirm und 3 für den Drucker.

Das Keyboard hat zwei Nummern, weil es (in den Begriffen der Programmierung) mit den beiden untersten Bildschirmzeilen verbunden ist.

Genug erst mal - wie können Sie das nutzen? Als erstes können Sie von und zu jedem der verfügbaren Geräte eingeben beziehungsweise anzeigen, vorausgesetzt, das Gerät ist dazu fähig. Das wird dadurch erreicht, daß in jedem INPUT- oder PRINT-Befehl ein #n-Parameter steht.

Beispiel:

```
PRINT #2; "abc"
```

zeigt auf dem Bildschirm den String "abc" an - wie zu erwarten. Aber wenn Sie jetzt versuchen:

```
PRINT #3; "abc"
```

möchte man es glauben? Die Meldung wird auf dem Drucker ausgedruckt! Das #3 im Befehl hat praktisch gesagt: 'Schicke alle weiteren Ausgaben von diesem Befehl zu Gerät 3', und, wie oben erwähnt, ist Gerät 3 der Drucker.

Wer braucht überhaupt LPRINT und LLIST?

Das Obige hat Ihnen vermutlich gezeigt, daß die Befehle LPRINT und LLIST überflüssig sind. Ihr Programm kann den Befehl enthalten:

PRINT #3; "irgendein Text"; TAB 10; "mehr Text"

und den Befehl LPRINT völlig umgehen. Ebenso können Sie schreiben:

LIST #3 oder LIST #3, Zeilennummer

Der große Vorteil dieses Winks ist, daß Ihre Programme für die Verwendung des Druckers wie des Bildschirms geschrieben werden können, ohne daß eine einzige Zeile geändert werden muß!

Die Zahl, die dem Zeichen # folgt, kann jeder numerische Ausdruck sein, so daß PRINT #x "abc" auf dem Bildschirm anzeigt, wenn x = 2, und auf dem Drucker druckt, wenn x = 3.

Sie können in Ihre Programme eine Zeile dieser Art einfügen:

```
10 LET dev = 2: REM auf Bildschirm umleiten
20 INPUT "Willst du den Drucker verwenden?": LINE a$
30 IF CHR$ CODE a$ = "Y" OR CHR$ CODE a$ = "y"
   THEN LET dev = 3
```

Fügen Sie von jetzt an in alle Ihre Printbefehle "#dev" ein – so einfach ist das.

Die obige Routine kann wirksamer verkürzt werden durch:

```
10 INPUT "Willst du den Drucker verwenden?"; LINE a$
20 LET dev = 2+ (CHR$ CODE a$ = "Y" OR CHR$
   CODE a4 = "y")
```

24 Zeilen auf dem Schirm benutzen

Diese Möglichkeit ergibt sich aus dem Vorigen. Man benützt das #-Zeichen bei Printbefehlen. Wie oben erwähnt, hat das Key-

board zwei Stromnummern, weil es nicht nur das Keyboard verwendet, sondern auch die beiden untersten Bildschirmzeilen.

Wenn Sie das folgende Programm fahren, bleibt es stehen, nachdem nur 22 Zeilen angezeigt worden sind:

```
10 FOR x = 1 TO 30: PRINT x; NEXT x
```

Geben Sie nun das nächste Programm ein und fahren Sie es:

```
10 FOR x = 1 TO 24
20 IF x < 23 THEN PRINT x
30 IF x > 22 THEN PRINT #1; AT x-23,0; x
40 NEXT x
50 PAUSE 4E4
```

Diesmal haben Sie alle 24 Zeilen angezeigt - und das mit einem BASIC-Programm!

Der Befehl PAUSE am Ende ist notwendig, weil BASIC, wenn das Programm fertig ist, auf den untersten beiden Zeilen die Meldung "0/OK, ..." bringt, wodurch prompt die beiden im Programm angezeigten Zeilen überschrieben werden.

Sie können damit experimentieren, in diesen Bildschirmbereich anzuzeigen, indem Sie bei Ihren Printbefehlen "#1" verwenden, weil man auf dem Schirm von unten nach oben anzeigen kann:

```
10 FOR x = 0 TO 22
20 PRINT #1; AT x, 0; x
30 NEXT x
40 PAUSE 4E4
```

In diesem Fall sollten Sie beachten, daß die maximale Zeilenzahl, die Sie anzeigen können, 22 beträgt, weil zwischen den oberen und unteren Displayteilen immer eine Zeile Abstand verlangt wird. Alles, was Sie hier getan haben, ist, die untersten zwei Zeilen auf 23 'auszuweiten'.

Sie können interessante Wirkungen erzielen, wenn Sie normales Anzeigen mit '#1'-Anzeigen verbinden.

Dateinamen

Obwohl **Sie** von jetzt an Ihre Programme unter gleichzeitiger Verwendung von Bildschirm und Drucker schreiben, tun das nicht alle Leute. Wie können Sie also deren Programme so verändern, daß Ergebnisse auf dem Drucker ausgedruckt werden,

ohne daß Sie das ganze Programm durchgehen und **PRINT**-Befehle verändern?.

Hier kommt der Befehl "OPEN #" zur Geltung. Sobald Sie ein Gerät öffnen, geben Sie an, um was für eine Art von Gerät es sich handelt - um ein Bildschirmgerät, einen Drucker oder ein Keyboard. Sie geben ferner die Gerätenummer (wie oben aufgeführt) an.

Wenn Sie also einen Befehl in ein Programm einfügen:

```
1 OPEN #2, "P"
```

verändern Sie Gerät 2 so, daß es ein Druckergerät wird. Von nun an werden alle Ausgaben im Programm zu Gerät 2 auf den Drukker gelenkt. Geschickt, nicht?

Probieren Sie dieses Programm aus:

```
10 OPEN #2, "s"
20 PRINT "abc"
30 PRINT TAB 10; "def"
40 PAUSE 4E4
```

Fahren Sie das, sehen Sie sich das Ergebnis an und verändern Sie Zeile 10 zu:

```
10 OPEN #2, "k"
```

Diesmal haben Sie Ausgabe zum 'Keyboard' oder vielmehr zu den untersten beiden Bildschirmzeilen gewählt.

Sie können auch schreiben:

```
10 OPEN #2, "p"
```

um Ausgaben zum Drucker zu schicken.

DATA-Befehle

Die Befehle READ, DATA, RESTORE sind eine wirksame Zugabe zum ZX-BASIC. Diese Befehle können vor allem im Verein mit der VAL-Funktion verwendet werden, um sehr kompaktes Programmieren zu erreichen. Dazu später mehr.

Bedenken Sie erstens das Problem, für ein Programm Daten zu liefern. In vielen Fällen sind die Daten zwar vorweg gesetzt und verändern sich von einem Lauf zum anderen nicht (Sie würden sonst LOAD verwenden), Sie könnten sich aber veranlaßt sehen, während der Entwicklung des Programms einzelne Posten

aus den DATA-Befehlen herauszunehmen oder solche einzufügen. Der Haken dabei ist der: Sie müssen das ganze Programm durchgehen und die FOR-Schleifen ändern, um die zusätzlichen Daten aufzunehmen.

Warum nicht vor Ihrem DATA-Befehl einen Zähler einsetzen, der dem Programm mitteilt, wie viele Posten es lesen soll? Auf diese Weise brauchen Sie das Programm eigentlich nie zu verändern, weil der Zähler durchgehend verwendet werden kann. Ein Beispiel:

```
10 DATA 6
20 DATA "Katze", "Hund", "Baum", "Jimmy", "Ball",
   "Ferien" ...
100 READ Zähler
110 DIM d4 (Zähler, 6)
120 FOR n = 1 TO Zähler
130 READ d$ (n)
140 NEXT n
```

Bedingungsprüfung

Ist Ihnen aufgefallen, daß ein Programm ganz rasch sehr groß werden kann, wenn es eine Folge von Bedingungen dieser Art gibt:

```
100 IF x = 23 AND y = 47 THEN GO TO 1000
110 IF x < 10 AND y < 10 THEN GO TO 1100
120 IF x < 10 AND y < 20 THEN GO TO 1200
130 IF x = 10 AND y <> 0 THEN GO TO 1300
...
```

DATA-Befehle können dazu benützt werden, die Größe dieses Programms zu verringern, weil Sie die Bedingungen als einen String in einen DATA-Befehl aufnehmen und dann VAL verwenden können, um das Ergebnis der Bedingung zu finden, wenn es gebraucht wird. Sehen Sie sich das an:

```
10 DATA 4
20 DATA "x = 23 AND y = 47", 1000, "x < 10 AND
   y < 10", 1100
30 DATA "x < 10 AND y < 20", 1100, "x = 10 AND
   y <> 0", 1200
...
```

```
100 READ n
110 FOR x = 1 TO n
120 READ c$, g: IF VAL c$ THEN GO TO g
130 NEXT n
```

Offenkundig würde Ihr kleines Programm nur dann profitieren, wenn eine große Zahl von Bedingungen vorhanden ist, aber die Methode kann bei vielerlei Programmen angewendet werden.

Sie können noch mehr daraus machen, wenn Sie in den DATA-Befehlen statt direkter Strings Stringvariablen verwenden ...

Wenn Sie dieses Verfahren in voller Anwendung sehen wollen, studieren Sie das Assembliererprogramm weiter hinten im Buch. In diesem Programm wird die Methode dazu benützt, unnötige Bedingungen zu vermeiden, was mühselig sein kann. Die Nebenwirkung hier ist die, daß das Programm nicht nur schneller läuft als sonst, es braucht auch weniger Speicherplatz.

Daten auf Band sichern

Die meisten Spiele brauchen die Kassette nicht, weil diese Art Programm nicht mit großen Datenmengen umgeht. Programme für Haushalt und Geschäft erfordern das Speichern von Daten schon eher, und hier kommen die Befehle LOAD und SAVE zu ihrem Recht.

Ein Programm kann in einem numerischen oder Stringarray enthaltene Ergebnisse speichern durch die Erklärung:

```
SAVE "Name" DATA x$()    (oder nur x() für numerisch)
```

Alles schön und gut, aber wenn ein nachfolgendes Programm dieses Array (Tabelle, Feld) verwenden muß, hat es von seiner Größe keine Ahnung! Offenkundig konnte das Programm, das den DIM-Befehl enthält, der das Array hervorbrachte, dafür sorgen, daß Fehlermeldung 3 (subscript wrong = falscher Index) nicht auftauchte, aber wie verhindern das andere Programme?

Das geht auf zweierlei Art. Erstens (und für mein Gefühl ist das am besten) kann man das erste Element des Arrays dazu verwenden, die Größe des Arrays aufzunehmen:

```
10 DIM a(100): LET a(1) = 100
```

Wenn mit LOAD wieder geladen, kann das Programm die Zahl der Posten im Array bestimmen durch die Angabe:

100 **FOR** n = 2 **TO** a(1)

Bei Zeichenarrays nehmen Sie:

10 **DIM** a$(100,4): **LET** a$(1) = **CHR$** 100

was, wenn wieder geladen, gesteuert wird durch:

100 **FOR** n = 2 **TO CODE** a$(1)

Übersteigt die Größe eines Zeichenarrays 255 Posten, dann müssen Sie eine Routine verwenden, mit der die Zahl in zwei Hälften geteilt werden kann - Sie könnten sogar schreiben:

10 **DIM** a$(100,6): **LET** a$(1) = **STR$** 100

Hier muß Ihre Stringlänge aber mindestens 3 Zeichen umfassen, weil **STR$** 100 einen String erzeugt, der drei Zeichen lang ist.

Nach dem neuerlichen Laden wird das bewertet durch:

100 **FOR** n = 2 **TO VAL** a$(1)

Die Wahl liegt aber letzten Endes bei Ihnen und wird vermutlich vom Programm bestimmt werden.

Die zweite Methode verwendet eine Kennzeichnung als Hinweis auf das Arrayende. In einem numerischen Array etwa könnten Sie als letzten Arraywert 'IE38' anfügen. Wenn es ins nächste Programm geladen wird, sorgt eine einfache Prüfung auf diesen Wert dafür, daß Sie nicht Fehlermeldung 3 erhalten.

Verwenden Sie bei Stringarrays ebenso einen String aus Copyrightsymbolen, um das Ende anzuzeigen.

Farben neu setzen

Viele Programme verlangen die Verwendung von Farbe, was zu Ärger führen kann, wenn Sie versuchen, das Programm zum Laufen zu bringen. Jedesmal, wenn das Programm anhält, bleiben die Farben für das Programmlisting in Kraft.

Ein Weg, dieses Problem zu umgehen, ist der, in jedes Programm eine Zeile einzufügen:

9999 **BORDER** 7: **PAPER** 7: **INK** 0: **CLS: STOP**

Wenn das Programm aus einem beliebigen Grund anhält, können Sie "GO TO 9999" eingeben, um den Bildschirm auf Normal zurückzusetzen, bevor Sie das Programm auflisten.

Der PAUSE-Befehl

In vielen Programmen bemerken Sie einen merkwürdig aussehenden PAUSE-Befehl:

PAUSE 4E4

Was bedeutet das? Nun, ein PAUSE-Befehl mißt einen Wert, der größer ist als 33000, veranlaßt den Spectrum, auf unbestimmt lange Zeit anzuhalten (im Englischen 'forever' = immer, was sich, richtig ausgesprochen, auch bei 4E4 - four E four - so anhört). Jedenfalls bleibt er stehen, bis man eine Taste drückt. Das ist eine einfache Art und Weise, sich zu merken, wie man ein Programm anhält.

Platz sparen

Der Spectrum ist zwar von Haus aus mit 16K ausgerüstet, aber es fällt trotzdem relativ leicht, ihn vollzufüllen. Programme, die Farben und Grafik voll nutzen, brauchen bald mehr als die 9K, die noch verfügbar sind, sobald die Schirmkarte berücksichtigt ist. Es gibt viele Methoden, die Größe Ihrer Programme zu verringern. Hier nur ein paar davon:

1. Ersetzen Sie alle numerischen Konstanten in Programmen durch ihre VAL-Entsprechung. Beispiel: Aus LET A = 5 würde LET A = VAL "5" werden, aus GO TO 1100 GO TO VAL "1100". Das scheint die Sache zwar nur noch zu verschlimmern, aber wenn Sie sich ansehen, wie der Spectrum Zahleneinzelheiten im Speicher unterbringt, wird Ihnen bald klar, daß Sie jedesmal, wenn Sie den Trick anwenden, 3 Bytes sparen. Multiplizieren Sie die Anzahl, die Sie ersetzen, mit 3, und Sie können sehen, daß das schon bei einem kurzen Programm sehr viel Speicherplatz spart.
2. Wenn ein Programm einen Bildschirm voller Regeln zeigt, dann tun Sie diese in ein getrenntes Programm, das später das Hauptprogramm lädt und fährt. Wenn die beiden Programme auf Band hintereinander kommen, ist es eine einfache Sache, eine Meldung anzuzeigen wie: "Band jetzt bitte anhalten" und "Band jetzt bitte starten".
3. Jedes Programm, das in DATA-Befehlen große Mengen vor-

gegebener Daten setzt, kann die Daten in einem Array zwischenspeichern. Die DATA-Befehle können später aus dem Programm gelöscht werden, das Array läßt sich unabhängig davon auf Band sicherstellen. Das Hauptprogramm kann bei Beginn mit 'LOAD "Name" DATA x()' geladen werden, unter Verwendung von Meldungen in der obengenannten Art.

Definierte Funktionen

Die Fähigkeit, eigene Funktionen zu definieren, kann bei der Herstellung kompakter Programme eine große Rolle spielen. Die BASIC-Sprache in ihrer Standardform läßt kein echtes 'strukturiertes Programmieren' zu, obschon ZX-BASIC so codiert werden kann, daß ein gewisses Maß an Strukturierung herauskommt. Definierte Funktionen machen diesen Traum eher zu einer Wirklichkeit, weil mit einem Einzelbuchstaben-Namen außerordentlich komplexe Funktionen gestaltet und aufgerufen werden können.

Es lohnt sich ins Gedächtnis zu rufen, daß die für die Definierung der Funktionen verwendeten Variablen als für die Funktion **lokal** behandelt werden. Das läßt sich am besten an einem Beispiel zeigen. Versuchen Sie es damit:

```
10 LET x = 10
20 DEF FN a(x) = x*2
30 PRINT FN a(3)
40 PRINT x
```

Obwohl Zeile 20 den Namen "x" verwendet hat, um die Funktion zu definieren, ist der Spectrum schlau genug zu erkennen, daß Sie die **Variable** x nicht verändern wollen. Bei der Ausführung von Zeile 40 wird also der Wert 10 angezeigt.

Ein kleiner Hinweis: Der Gebrauch definierter Funktionen in einem Programm ist offenkundig vorteilhaft, aber Sie sollten wissen, daß der Spectrum jedesmal, wenn die Funktion verwendet wird, das komplette Programm nach dem richtigen "DEF FN"-Befehl sucht, um festzustellen, was zu geschehen hat. Das bedeutet: Sie sollten alle Ihre definierten Funktionen an den Programmanfang setzen, damit das Programm so schnell wie möglich läuft, vor allem bei größeren Programmen.

Ich habe unten eine Handvoll nützlicher definierter Funktionen für Sie aufgeführt - die meisten haben sich beim Schreiben der Programme für dieses Buch als praktisch erwiesen:

DEF FN i(x) = **PEEK** x + 256* **PEEK** (x + 1)
Mit dieser Funktion können Sie den Inhalt eines 16 Bit-Werts aus dem Speicher laden. Ich benütze sie häufig dazu, den Wert einer Systemvariablen zu laden. Um den Wert von RAMTOP zu erhalten, schreiben Sie beispielsweise nur: **LET** ramtop = **FN** i (23732).

DEF FN p(x) = **PEEK** x − (256* (**PEEK** x > 127))
Wenn Sie den Wert eines Einzelbytes im Speicher feststellen wollen, müssen Sie darauf Rücksicht nehmen, daß er entweder einen vorzeichenlosen Wert von Null bis 244 oder einen Wert mit Vorzeichen zwischen −128 und +127 darstellen kann. Diese Funktion behandelt den Wert als vorzeichenbehaftet und gibt den Byteinhalt an Adresse x als einen Wert zwischen −128 und +127 bekannt. Beispiel: Hex FF kann betrachtet werden entweder als 255 (vorzeichenlos) oder als −1 (mit Vorzeichen). Wenn Byte 23681 FF hex enthält, dann liefert **FN** p (23681) den Wert −1.

DEF FN h (x) = **INT** (x/256)
Liefert das 'hochwertige' Byte einer Variablen, die einen 16 Bit-Wert enthält.

DEF FN L (x) = **FN** (h (x)* 256))
Liefert das 'niederwertige' Byte einer Variablen, die einen 16-Bit-Wert enthält. Beachten Sie hier die Verwendung einer definierten Funktion innerhalb einer anderen - diese Funktion verlangt "DEF FN (h (x))" oben.

Die beiden letzten werden in der Hauptsache dazu verwendet, Werte mit 'POKE' in Systemvariable einzugeben. Beispiel: Wenn Sie den Wert der Systemvariablen CHAR$ zu einer Zahl verändern wollen, die in Variable enthalten ist, würden Sie im Normalfall schreiben:

```
1100 POKE 23606, c-256* INT (c/256)
1110 POKE 23607, INT (c/256)
```

Siehe dazu Kapitel 25 des Sinclair-Handbuchs.

Wenn Sie die beiden Funktionen verwenden, können Sie nun aber schreiben:

```
1100 POKE 23606, FN L (c)
1110 POKE 23607, FN (H (c)
```

Die Uhr

Manche Spiele in diesem Buch nutzen die 'Uhr' im Spectrum. Diese Uhr ist in Wahrheit eine Systemvariable (FRAMES), die fortwährend jede Sekunde fünfzigmal tickt. Mit Maschinencode kann man diese Uhr abschalten (obwohl das unangenehme Nebenerscheinungen hervorruft), aber in BASIC bleibt sie zuverlässig.

Die Uhr besetzt eine Drei-Byte-Stelle der Systemvariablen, die in der Lage ist, 2^{24} (16 777 216) mal zu 'ticken'. Da die Uhr in der Sekunde fünfzigmal zählt, läuft das auf eine maximale Zeitzählung von fast vier Tagen hinaus!

Bei den meisten Spielen ist eine abgelaufene Zeitzählung von 21 Minuten mehr als ausreichend. In diesem Fall braucht man nur die beiden ersten Bytes des FRAMES-Zählers. Wenn Sie längere Abläufe zu messen haben als 21 Minuten, müssen Sie das dritte Byte berücksichtigen.

Damit die Uhr genutzt werden kann, muß sie vor Beginn eines Programms neu eingestellt werden. Das geschieht dadurch, daß man Null in die ersten zwei Bytes setzt (bei mehr als 21 Minuten in alle drei). Das geht so:

```
POKE 23673,0: POKE 23672,0
```

Um festzustellen, wie viele Sekunden vergangen sind, seit die Uhr neu gestellt wurde, verwenden Sie die folgenden Befehle:

```
LET vergangen = FN i (23672)/50
```

(Denken Sie daran, daß die Funktion FN i (x) oben definiert ist.)

Soll Ihr Programm, sagen wir, 200 Sekunden laufen, ist ein Test jetzt einfach:

```
IF vergangen > 200 THEN GO TO ...
```

Berücksichtigen Sie, daß die Uhr zwar weiterhin 'tickt', Ihr Programm die abgelaufene Zeit aber nur dann prüft, wenn Sie den oben gezeigten Befehl befolgen. Aus diesem Grund sollten Sie nicht schreiben:

IF vergangen = 200 **THEN GO TO** ...

weil das Programm zwischen Zeitprüfungen vielleicht länger als eine Sekunde braucht, was diesen 'Ist gleich'-Test unter bestimmten Bedingungen scheitern läßt. Fügen Sie stets eine Prüfung auf 'größer als' ein.

Wenn Ihr Programm jede Sekunde einen Ton erzeugen will, können Sie schreiben:

IF vergangen <> letzte Prüfung **THEN BEEP** .01,-8

In diesem Fall können Sie die Ungenauigkeit sogar hören, weil Ihr Programm zwischen Zeitprüfungen manchmal länger braucht, was dazu führt, daß die Töne in unregelmäßigen Abständen kommen.

Wie man Fehlermeldung 3 vermeidet

Während der ganzen Programme in diesem Band (und sogar früher in diesem Abschnitt) werden Sie oft eine Zeile dieser Art sehen:

IF CHR$ CODE a$ = "Y" **THEN** ...

Der Sinn dieser Zeile ist der, ein Problem im folgenden Programm zu vermeiden:

```
10 INPUT "Drücke ENTER"; a$
20 IF a$ (1) = "X" THEN GO TO 10
```

Wenn Sie es fahren, drücken Sie ENTER, wie verlangt. Sie werden in Zeile 20 sofort die Fehlermeldung 3 sehen. Der Grund: Die Bedingung a$ (1) kann nicht bewertet werden - a$ enthält keine Zeichen. Die Folge davon ist, daß ein Programm stets prüfen und dafür sorgen muß, daß ein String lang genug ist, bevor sein Inhalt geprüft wird.

Die vorangestellte Zeile umgeht das Problem jedoch, weil die **CODE**-Funktion sogar dann wirksam wird, wenn der String leer ist - sie liefert dann nur den Wert Null. **CHR$ CODE** a$ erbringt damit eines von zwei Ergebnissen, entweder das erste Zeichen von a$ oder das Zeichen mit dem Code Null (das ist ein unbenütztes Zeichen).

Da die **CODE**-Funktion auf das **erste** Zeichen des Strings wirkt, können Sie diese Methode dazu benützen, sich irgendein

Einzelzeichen des Strings anzusehen, ohne Fehlermeldung 3 hervorzurufen. Sehen Sie sich Folgendes an:

IF CHR$ CODE a$ (4 **TO**) = "X" **THEN** ...

Das funktioniert sogar dann, wenn a$ weniger als vier Zeichen enthält. Das Prinzip bleibt das gleiche und kann außerordentlich nützlich sein beim Schreiben von Programmen, die ein hohes Maß an kluger Zeileneingaben-Abtastung erfordern. Das bringt mich zum Thema:

Kluges Zeilenabtasten

Was das ist? Wenn Sie schon einmal die echten Abenteuer-Spiele ausprobiert haben, wird Ihnen wohl klar sein, daß Sie einen Befehl eingeben, den der Computer abtastet, bevor er nach Ihren Anweisungen handelt.

Für die Befehle, die Sie eingeben, bestehen fast unendlich viele Möglichkeiten. Also muß der Computer offenkundig die Eingaben prüfen und entscheiden, welche er befolgen kann und welche nicht.

Die erste Aufgabe, die er leistet, ist die, daß er die Zeile in ihre Bestandteile aufspaltet - in Wörter. Die untenstehende Routine besteht aus zwei Teilen. Erstens habe ich eine Subroutine eingefügt, die einen in der Variable A$ enthaltenen String abtastet, um alle Wortendungen zu finden. Diese Information ist in einem Array A () enthalten. Zweitens gibt es eine definierte Funktion FNs$ (), die Ihnen anschließend jedes gewünschte Wort aus dem String liefert.

Array A () muß auf die Höchstzahl von Wörtern gesetzt werden, die Sie in der Eingabezeile verwenden wollen.

```
9600 REM Zeilenabtastung
9610 LET Maxwörter = 10      (nach Wunsch verändern)
9620 DIM a (Maxwörter, 2)    (Wortzeiger)
9630 LET Wortanfang = 0:
     LET Wortnr = 1          (Zeiger 1. Wort)
9640 FOR x = 1 TO LEN a$     (Zeile abtasten)
9650 Wortanfang <> (a$ (x) <> " "
     AND a$ (x) <> "," THEN
     LET a (Wortnr, Wortanfang +1) = x-1 (Wortanf/
     ende
```

```
        LET Wortnr = Wortnr + Wortanfang:
        LET Wortanfang = NOT Wortanfang:
        (Anf/Ende-Umkehr.
        IF Wortnr > Maxwörter THEN RETURN
9660 NEXT x: LET a (Wortnr, 2) = x-1
        (Sicherungszeiger)
9670 RETURN
```

Die definierte Funktion ist:

```
9680 DEF FN s$ (x) = a$ (a(x, 1) + 1 TO a (x, 2)
```

Die Routine verlangt einige Erläuterungen. Die Variable 'Wortanfang' enthält den Wert 'falsch', sobald die Routine nach einer Wortendung (z. B. nach einem Leerraum oder einem Komma), und den Wert 'wahr' (1), wenn sie nach einem Wortanfang (nicht einem Leerraum oder einem Komma) sucht. Die Positionen im String A$ von Wortanfängen und -endungen werden festgehalten im Array A (x, 2). Zeile 9650 führt mehrere Aufgaben aus. Sie hält den laufenden Wortanfang/ende-Zeiger im Array A (x, 2) fest, aktualisiert die Wörterzählung, wenn ein neues Wort angefangen wird, und kehrt den Test dann um; erfolgte die letzte Suche also nach einem Wortanfang, wird die nächste nach einem Wortende erfolgen und umgekehrt.

Keine einfache Routine. Anfänger werden damit eine Weile beschäftigt sein.

Wenn String a$ also enthält 'Der Rat ist in der Tat', können Sie nach dem Lauf von Subroutine 9600 die folgenden Ergebnisse erhalten:

```
PRINT FN s$ (1) liefert "Der"    PRINT FN s$ (2) "Rat"
PRINT FB s$ (9) liefert ""       PRINT FN s$ (6) "Tat"
```

Diese Ergebnisse erleichtern die Einzelverarbeitung des Strings erheblich. Ein Programm kann dadurch, daß es eine Tabelle zulässiger Wörter abtastet, leicht auf das Vorhandensein entweder bestimmter Wörter oder Gruppen von Wörtern und Wendungen prüfen. Ihre Phantasie ist die einzige Grenze!

Burgmauern

Die Normannen sind da! Sie stehen auf der Mauerbrüstung, können hinunterblicken und sehen, wie die Leitern an den Mauern heraufkommen - aber was können Sie tun?

"Siedendes Pech!" denken Sie und schütten sofort Pech auf die Eindringlinge hinunter. Manchmal verfehlen Sie, manchmal hören Sie die Geräusche, wenn das Pech über die Opfer rinnt.

Es ist unausweichlich, daß sie früher oder später eindringen. Die Frage ist nur, wie lange Sie sie fernhalten können. Pech gibt es genug, vorausgesetzt, Sie füllen immer wieder auf, aber wenn Ihr Faß einmal leer ist - mehr brauchen sie nicht, um die Burg zu stürmen, während Sie hilflos hin- und herlaufen.

Sie brauchen sich nur mit zwei Reglern zu befassen: Taste "6" läßt Sie Pech hinuntergießen, mit Taste "8" können Sie das Faß wieder füllen. Das nimmt aber Zeit in Anspruch, und während dieser Pause steigen die Leitern weiter in die Höhe.

Am Anfang können Sie die Schwierigkeitsstufe des Spiels wählen - je schwieriger, desto mehr Leitern, aber um so besser die Punktezahl für jedes Opfer.

Stellen Sie das Programm sicher mit:

SAVE "CASTLE" **LINE** 1

```
    1 REM **** Castle walls ****
    2 REM ©Phipps Associates1982
    3 REM     By Trevor Toms
    5 RESTORE : RANDOMIZE : CLEAR

   10 GO SUB 8100: REM initialise
   20 GO SUB 8000: REM instructio
ns
   30 INPUT "Spielstaerke? (1-5)?
 ";lv: IF lv<1 OR lv>5 THEN GO T
O 30
   40 LET n1=lv+1          (Wahrscheinlichkeiten setzen
   50 LET lg=28/n1
   60 LET r1=.2+lv/10: REM freque
ncy of ladder growing
   70 LET r2=.1+lv/50: REM probab
ility of oil stopping ladder
```

```
  80 LET r3=(5-lv)*10: REM movem
ent speed
  90 LET g=1000: LET s=0: LET h$
=">HELP<"
 100 GO SUB 8200: REM draw castl
e
 105 GO SUB 8600
 110 LET lp=2: LET rp=29: LET di
r=1
 120 DIM l(nl,2): FOR x=1 TO nl:
 LET l(x,2)=21: NEXT x
 130 LET y=1: FOR x=2+lg/2 TO 29
 STEP lg: LET l(y,1)=INT (x+.5):
 LET y=y+1: NEXT x
 140 OVER 0
 200 FOR p=lp TO rp STEP dir
 210 PRINT AT 3,p; OVER 1; INK 8
; PAPER 8;CHR$ 145
 220 IF INKEY$="6" THEN GO SUB 8
300: REM pour oil
 221 LET f=(INKEY$="8")*(g<>0)
 225 IF RND>r1 THEN GO TO 500
 230 IF INKEY$<>"" THEN GO TO 23
0
 235 LET l=INT (RND*nl)+1
 240 LET a=l(l,2): LET b=l(l,1)
 250 PRINT AT a,b; PAPER 8;CHR$
144
 255 BEEP .1,21-a
 260 IF a=3 THEN GO TO 600
 270 LET l(l,2)=a-1
 500 IF f THEN IF RND>r2 THEN LE
T g=g+FN r(60+lv*20): GO SUB 860
0: GO TO 230
 510 PRINT AT 3,p; OVER 1; INK 8
; PAPER 8;CHR$ 145
 515 NEXT p
 520 LET dir=dir*-1: LET lp=(lp=
2)*27+2: LET rp=(rp=2)*27+2
 530 GO TO 200
 600 PRINT AT 21,0; OVER 0; PAPE
R 2; INK 7;"HILFE!SIE SIND OBEN!
!!!"
7000 REM game end
7010 REM b=ladder column,p=your
column
7015 PAPER 8: INK 0: OVER 0
7017 LET l$=""
```

Bewegungsgrenzen

Leitern-Grundposit.

Mann bewegen

Pech ausgießen

aufmehr Pech prüfen

Leitern steigen weiter

noch nachfüllen

Posit. nächster Mann
Richtung wechseln

o. je. Das Ende

```
7020 FOR x=21 TO 3 STEP -1
7025 LET t$=CHR$ 146: IF t$=l$ T
HEN LET t$=CHR$ 147
7030 IF x<>21 THEN PRINT AT x+1,
b;CHR$ 144
7040 PRINT AT x,b;t$
7045 BEEP .1,-1
7050 PAUSE 15
7055 LET l$=t$
7060 NEXT x
7065 OVER 1
7070 PRINT AT 3,b;CHR$ 145
7100 LET dir=SGN (p-b)
7105 IF NOT dir THEN PRINT AT 3,
b-3; PAPER 2; INK 7;">ZACK<": PR
INT AT 21,0;"Die Leiter hat Dich
 erwischt!": GO TO 7220
7110 FOR x=b TO p STEP dir
7120 IF x<>b THEN PRINT AT 3,x-d
ir; OVER 1;CHR$ 145
7130 PRINT AT 3,x; OVER 1;CHR$ 1
45
7135 BEEP .1,-10
7140 PAUSE 15
7150 NEXT x
7200 PRINT AT 3,p-3; PAPER 7; IN
K 0; FLASH 1; OVER 0;"MORD!!!"
7210 FOR x=52 TO -13 STEP -1: BE
EP .001,x: NEXT x
7220 INPUT "Noch ein Spiel? (J/N
)? ";y$: IF NOT LEN y$ THEN GO T
O 7220
7230 IF CODE y$=CODE "J" OR CODE
 y$=CODE "j" THEN GO TO 30
7240 GO TO 9999
8000 REM instructions
8010 GO SUB 8200: REM draw castl
e
8015 PAPER 8: OVER 1
8020 PRINT AT 3,16;CHR$ 145
8025 PRINT AT 5,7; INK 7; PAPER
2;"** STURMANGRIFF **"
8030 PRINT "Die schrecklichen No
rmannen wol=len Deine Festung st
uermen.Du   hast 1000 Gallonen h
eisses Oel  zum Hinunterschuette
n (Druck aufTaste >6<.Geh sparsa
m damit um!"
```

Normannenleiter

wohin bewegen

Bewegung zu Ihnen

Blutbad

```
8040 PRINT "Mit Druck auf Taste
>8< kannst  Du jederzeit neues O
el holen.   Aber Vorsicht: Die L
eitern wer= den dabei immer hoeh
er..."
8090 RETURN
8100 REM initialise
8105 DATA "a",66,255,66,66,66,25
5,66,66
8110 DATA "b",24,60,86,126,60,24
,255,255
8115 DATA "c",24,90,126,24,56,40
,4,0
8120 DATA "d",24,90,126,24,28,20
,20,0
8130 FOR c=1 TO 4: READ c$: FOR
r=0 TO 7: READ x: POKE USR c$+r,
x: NEXT r: NEXT c
8140 DEF FN r(x)=INT (RND*x)+1
8190 RETURN
8200 REM castle
8210 BORDER 6: PAPER 6: INK 0
8215 CLS
8220 PLOT 16,175: DRAW 0,-31: DR
AW 224,0: DRAW 0,31: DRAW 15,0:
DRAW 0,-175: DRAW -255,0: DRAW 0
,175: DRAW 16,0
8230 PAPER 5
8240 FOR y=0 TO 3: PRINT AT y,2;
: FOR x=2 TO 29: PRINT OVER 1;"
";: NEXT x: NEXT y
8250 PAPER 6
8260 RETURN
8300 REM pour oil
8305 IF NOT g THEN GO SUB 9000:
FOR z=1 TO 20: NEXT z: GO SUB 90
00: GO TO 8450
8310 FOR y=4 TO 21
8320 LET x$=SCREEN$ (y,p)
8330 IF NOT CODE x$ THEN IF RND<
r2 THEN GO TO 8360
8340 PRINT AT y,p; INVERSE 1;" "
8345 BEEP .01,-15: BEEP .01,-10
8350 NEXT y
8360 LET x=y-1
8370 FOR z=1 TO n1: IF l(z,1)=p
THEN LET s=s+1+(x-l(z,2))*lv: LE
T l(z,2)=x: FOR n=52 TO 26 STEP
```

ein Leerraum

Boden finden

ein Leerraum

Gluckergeräusche

Leiterhöhe verändern

```
-1: BEEP .001,n: NEXT n: GO TO 8
400
8380 NEXT z
8400 FOR y=4 TO x: PRINT AT y,p;
 OVER 1; INVERSE 1;" ": NEXT y
8405 PLOT 0,0: DRAW 255,0
8410 LET g=g-FN r(60+lv*20): IF
g<0 THEN LET g=0
8430 GO SUB 8600
8440 IF g THEN RETURN
8450 PRINT AT 21,0;"KEIN OEL MEH
R!"
8460 RETURN
8600 REM score line
8610 PRINT AT 0,9; PAPER 2; INK
7;"Punkte:";s;TAB 20; FLASH (g=0
);"Oel:";g;"  "
8620 RETURN
9000 REM help
9010 PRINT AT 2,p-2; OVER 1; PAP
ER 8; INK 2;h$
9020 RETURN
9999 PAPER 7: BORDER 7: INK 0: S
TOP
```

ein Leerraum

Burgfundament neu
Pechmenge
verringern

2 Leerräume

Programme für Fehlersuche

Sie haben also eben Ihr neues Meisterwerk fertiggeschrieben und halten den Atem an, während Sie RUN drücken ...

Und ... nichts. Das Programm sitzt mit leerem Bildschirm da oder unten am Schirm erscheint eine Fehlermeldung. Was nun? Wo fangen Sie an, den Grund dafür zu ermitteln, warum der Spectrum nicht mitspielen will?

Zu allererst: Notieren Sie die Fehlermeldung auf einem Stück Papier. Der Grund: Man vergißt nur zu leicht die Zeilennummer, die als Bestandteil der Meldung erschienen ist.

Wenn das Programm einfach 'steht', drücken Sie BREAK, um zu erfahren, wo genau das Programm fehlgelaufen ist.

Drücken Sie auf keinen Fall CLEAR, denn das löscht alle Variablen im Programm, die außerordentlich wichtig dabei sein können, Ihnen bei der Auffindung von Fehlern zu helfen.

Wenn Sie ein Neuling beim Programmieren sind, kann es manchmal schwerfallen, zu begreifen, wie ein Programm schieflaufen kann – schließlich hat es ja beim Eingeben keine Syntaxfehler gegeben!

Ich stelle Ihnen nun ein kleines Programm vor, das einige Fehler enthält. Sie geben das Programm ein, fahren es, und anhand des Textes hier werden wir (hoffentlich) feststellen, wo die Fehler auftreten, sie korrigieren und es noch einmal versuchen. Auf diese Weise sollten Sie die Hauptpunkte bei der Fehlersuche in Ihren eigenen Programmen kennenlernen.

Ich möchte ein Programm schreiben, das einen Mann mit Fallschirm auf dem Bildschirm von oben nach unten sinken läßt. Wenn er unten ankommt, soll der Spectrum "PLATSCH" anzeigen.

Anmerkungen:

1. Mann und Fallschirm können mit einer definierbaren Grafik hervorgebracht werden. Ich habe dafür den Buchstaben "A" (CHR$ 144) gewählt. Der Mann wird so aussehen:

Siehe das Programm 'Benutzergrafik-Tafel' wegen weiterer Einzelheiten dazu.

2. Die Subroutine für das Grafikzeichen ist eine 'Standard'-Subroutine - ein solches Beispiel wird in Kapitel 14 des Sinclair-Handbuchs gegeben.

Hier also mein erster Versuch, das Programm zu schreiben. Sie könnten es eingeben und mit RUN fahren, um zu sehen, was dabei herauskommt:

```
10 REM Fallschirm
20 DATA "A", 28, 62, 127,
   34, 42, 62, 28, 20          (Grafikanordnung)
30 READ c$                     (Zeichenbuchst. holen)
40 FOR r = 0 TO 8              (Zeichen laden)
50 READ x: POKE USR v$ + r, x
60 NEXT x
100 CLS                        (Schirm leer bereit)
110 FOR y = 0 TO 22            (auf jeder Zeile anz.)
120 PRINT AT y, 16;            (Mann und Fallsch.
    CHR$ 145                   anzeigen)
130 NEXT y                     (in jeder Zeile)
```

Als erstes erhalten Sie (wenn Sie es so eingegeben haben, wie es hier steht) "2 Variable not found 50:2". Offenkundig ist ein Fehler in Zeile 50 - aber welcher? Eine der Variablen in der Zeile ist nicht gefunden worden. Um festzustellen, welche, geben Sie ein (als direkte Befehle):

```
PRINT x
PRINT v$
PRINT r
```

Sie werden feststellen, daß das bei der ersten und dritten klappt, nicht aber bei der zweiten (PRINT v$). Das ist die fehlende Variable. Werfen Sie einen Blick auf Zeile 30 - da steht "READ c$". Da c$ das Grafikzeichen enthält, das wir zu setzen versuchen, ergibt die Verwendung von v$ in Zeile 50 offenkundig keinen Sinn! Verändern Sie Zeile 50 zu:

```
50 READ x: POKE USR c$ + x
```

Fahren Sie das Programm erneut. Was geschieht diesmal?

"1 NEXT without FOR, 60:1". Zurück zum Listing. Zeile 60 hat

"NEXT x", während in Zeile 50 steht "FOR **r** = 0 TO 8". Was ist richtig? Wenn Sie sich Zeile 50 ansehen, können Sie erkennen, daß die Variable x durch den READ-Befehl gesetzt wird, während die Variable r steuert, in welche Zeile das Grafikzeichen gesetzt wird. Zeile 60 ist also falsch, und Sie sollten sie verändern zu:

60 **NEXT** r

Wieder fahren! Was nun? "E Out auf DATA 50:1". Zeile 50 wird neunmal befolgt, weil Zeile 40 sagt "For R = 0 TO 8" – insgesamt neun Schleifen. Die DATA-Anweisung auf Zeile 20 enthält 9 Posten, aber in Zeile 30 gibt es ein zusätzliches "READ". Entweder enthält also Zeile 20 ungenügende Daten, oder Zeile 40 darf nur 8 Posten enthalten.

Wir alle wissen (?), daß für ein Grafikzeichen nur acht Datenposten erforderlich sind. Zeile 20 muß also in Ordnung sein. Verändern Sie Zeile 40 so:

40 **FOR** r = 0 **TO** 7

Noch einmal RUN. Mann! Sofort zeigen sich zwei Probleme. Erstens läuft ein ganzer Strom von Männern und Fallschirmen den ganzen Bildschirm hinunter. Zweitens ist die Meldung "5 Out of Screen 120:1" erschienen. Nehmen wir das zuerst.

Zeile 110 regelt, welche Zeile auf dem Bildschirm für die Anzeige verwendet werden soll. Probieren Sie also:

PRINT y

als direkten Befehl. Wenn alles so ist wie bei mir, erhalten Sie den Wert 22. Nun gibt es eine Zeile 22 nicht – sie sind von 0 oben am Bildschirm bis 21 unten numeriert – Zeilen 22 und 23 sind für den Spectrum reserviert. Schuldig ist Zeile 100 im Programm. Sie müßte lauten:

110 **FOR** y = 0 **TO** 21

Das zweite Problem war der Strom von Männern in der Bildschirmmitte. Hier habe ich bei meinem Programmentwurf einen Fehler gemacht. Mit der Anzeige jedes neuen Mannes sollte der alte gelöscht werden. Versuchen Sie, die folgende Zeile einzugeben:

115 **STOP**

Fahren Sie jetzt das Programm. Sie sehen "9 **STOP** statement, 115:1" unten am Bildschirm. Geben Sie **CONTINUE** als Befehl

ein. Diesmal sehen Sie den ersten Mann angezeigt, dann hält das Programm wieder an. Wenn Sie wieder CONTINUE drücken, sehen Sie den nächsten Mann angezeigt. Dieser Trick ist überaus nützlich, um ein Programm von Fehlern zu befreien. Erstens verlangsamt er das Programm und läßt Ihnen Zeit, sich den Bildschirm anzusehen, zweitens können Sie jede der Variablen anzeigen lassen, um nachzuprüfen, ob sie den richtigen Wert haben. Wenn Sie irgendeine Variable anzeigen lassen, müssen Sie Ihr Programm mit GO TO weiterfahren, weil CONTINUE versucht, den letzten direkten Befehl neu auszuführen, und der Spectrum einfach stehenbleibt, bis Sie BREAK drücken.

Außerdem wollen Sie vielleicht gar keine Posten auf dem Bildschirm anzeigen lassen, weil das all die schönen Dinge durcheinanderbringt, die Ihr Programm erzeugt hat. Sie können das durch das Folgende umgehen:

Statt PRINT y einzugeben,
... geben Sie ein **PRINT** #1; **AT** 0,0; y: **PAUSE** 4E4

Das führt dazu, daß der Wert von y auf den beiden untersten Bildschirmzeilen angezeigt wird, dann wartet der Computer darauf, daß Sie eine Taste drücken. Siehe das Kapitel 'Winke und Tips' weiter vorn.

Zurück zu unserem Programm. Entfernen Sie Zeile 115 und fügen Sie dem Programm folgende drei Zeilen an:

```
115 IF y > 0 THEN PRINT AT y-1, 16; "" (ein Leerraum)
125 PAUSE 12
140 PRINT AT 21, 14; "PLATSCH"
```

Die erste dieser Zeilen entfernt den 'Strom' von Männern den Bildschirm herunter. Beachten Sie, daß der Test "IF y > 0" erforderlich ist, weil das Programm sonst auf Zeile minus 1 nicht anzuzeigen versuchen würde.

Die zweite Zeile ist eine kurze Pause, um das Programm ein wenig zu verlangsamen, weil der Mann sonst wie ein Bleigewicht herunterfällt! Die dritte Zeile beendet das Programm mit der Meldung PLATSCH, die ich am Anfang schon erwähnt hatte.

Wenn Sie das Programm jetzt fahren, sollte es genau das leisten, was ich wollte. Sie sollten gesehen haben, wie ich die Fehler mit Hilfe von drei wichtigen Methoden aufgespürt habe:

1. Ungefähr wissen, was das Programm leisten soll, weil Sie sonst 'blind' kämpfen.
2. Direkte PRINT-Befehle (also ohne Zeilennummer) verwenden, um den Wert von Variablen zu prüfen, wenn ein Fehler auftritt. Aus dem PRINT-Befehl kann ein "PRINT #1; AT 0,0" werden, wenn Sie den Bildschirm nicht gestört haben wollen.
3. STOP-Befehle verwenden, damit Teile des Programms genauer überprüft werden können. Sie lassen sich entfernen, sobald Sie überzeugt sind, daß der Abschnitt richtig funktioniert.

Ich möchte abschließend hier noch zwei Dinge sagen: Schreiben Sie Ihre Programme auf, **bevor** Sie sie eingeben. So lassen sich beim Fahren des Programms Fehler leichter entdecken, weil Sie unrichtige Zeilen (solche, die zu Fehlermeldungen führen) mit Ihrer Niederschrift vergleichen können. Zweitens: Wenn Sie einen Drucker haben, **verwenden** Sie ihn! Machen Sie ein neues Listing von Ihrem Programm. Sie möchten nicht glauben, wie viele Fehler von Programmieren gemacht werden, die ein überholtes Listing vor sich haben.

Kurz vor Schluß gebe ich Ihnen ein Listing des Programms, wie es aussehen **sollte**, nachdem alle obenerwähnten Veränderungen eingefügt worden sind:

```
  10 REM Parachute
  20 DATA "A",28,62,127,34,42,62
,28,20
  30 READ c$
  40 FOR r=0 TO 8
  50 READ x: POKE USR v$+r,x
  60 NEXT r
 100 CLS
 110 FOR y=0 TO 22
 120 PRINT AT y,16;CHR$ 145
 130 NEXT y
```

Phantastisch!

Bei diesem Abschnitt sehen Sie wirklich rot! Hier finden Sie ein Programm, mit dem man auf dem Bildschirm Labyrinthe erzeugen kann. Jedes Labyrinth ist einzigartig, und es gibt vom Eingang zum Ausgang nur einen Weg.

Es ist ziemlich leicht, ein Labyrinth-Zeichenprogramm zu machen, das vielerlei Wege zuläßt, aber solche Labyrinthe sind nicht so befriedigend, weil man sie viel rascher lösen kann.

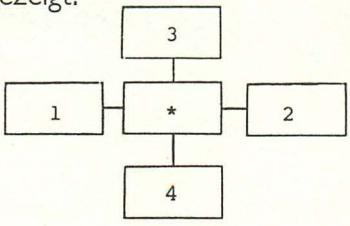
Das Programm braucht ungefähr zehn Minuten, um ein Labyrinth in einem Gitter 30 mal 20 zu zeichnen. Das ist nicht gerade schnell, aber es kommen unterhaltsame Ergebnisse heraus.

Wenn Sie einen Drucker haben, lohnt es, eine gedruckte Wiedergabe des Bildschirms herzustellen, damit Sie den Verbindungsweg auf Papier verfolgen können.

Das Labyrinth wird auf dem Bildschirm vor Ihren Augen gezeichnet, was ein sehr interessantes Schauspiel ist. Sie haben davon aber keinen Vorteil, weil Eingangs- und Ausgangspunkte erst dann angezeigt werden, wenn das ganze Labyrinth fertig ist.

Der ganze Algorithmus ist enthalten in einer Subroutine, die im Programm bei Zeile 7000 anfängt. Vielleicht wollen Sie diese Routine dazu verwenden, Ihre eigenen Spiele zu produzieren, die auf einem Labyrinth beruhen. Wenn ja, dann bleibt Ihnen, wenn Sie die verschiedenen vorkommenden PRINT-Befehle unterdrücken, das Zeichenarray m$ (), das ein Bild des geschaffenen Labyrinths enthält. Das Format dieses Labyrinths sieht so aus:

Jede Zeile auf dem Bildschirm entspricht einem der großen Strings im Array m$. Jede Zeile enthält eine Codenummer zwischen 0 und 15; sie stellt die Verbindungen zwischen benachbarten Zeilen dar. Die Zusammenhänge werden in der unten stehenden Tabelle gezeigt:



Code: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Verbunden * mit:

```
1    - y - y - y - y - y - y - y - y
2    - - y y - - y y - - y y - - y y
3    - - - - y y y y - - - - y y y y
4    - - - - - - - - y y y y y y y y
```

Sie werden diesen Algorithmus später verwendet sehen im Programm "3D-Labyrinth" - eine unglaubliche Reise durch ein kubisches Labyrinth.

Wenn das Programm gefahren wird, sehen Sie auf dem Bildschirm "Keimwert" angezeigt. Das ist die Zahl, mit welcher der RANDOMIZE-Befehl ausgesät wird. Sie kann dazu verwendet werden, dasselbe Labyrinth zweimal hervorzubringen. Wenn Sie das tun wollen, beginnen Sie das Programm mit:

LET es = Keimwert (Ihre Zahl)
GO TO 20

Das Programm sollte (wie üblich) gesichert werden durch:

SAVE "AMAZING" **LINE** 1

```
  1 REM ****  Amazing! ****
  2 REM ©1982 Phipps Associates
  5 RESTORE : RANDOMIZE : CLEAR
 : OVER 0
 10 LET rs=INT (RND*256)
 20 CLS : PRINT TAB 6;"**** La
byrinth ****"
 30 PRINT AT 2,0;"Typ: ";rs
 40 RANDOMIZE rs                    Zufallskeim setzen
 60 LET m$="Labyrinth-Breite? (
2-30):": GO SUB 9000: IF end THE
N GO TO 9999
 70 LET w=VAL a$: IF w<2 OR w>3
0 THEN GO TO 10
 80 LET m$="Labyrinth-Hoehe? (2
-20):": GO SUB 9000: IF end THEN
 GO TO 9999
 90 LET h=VAL a$: IF h<2 OR h>2
0 THEN GO TO 30
100 LET s=INT ((30-w)/2)            auf Schirm zentrieren
110 GO SUB 7000                     Labyrinthgenerator
200 GO TO 9999                      alles fertig
7000 REM create maze array m$
7010 CLS
7020 DATA x-1,y,x>1,x+1,y,x<w       Quadrate ringsum
7030 DATA x,y-1,y>1,x,y+1,y<h
```

```
7040 DATA 139,131,139,131,138,13
0,138,130,139,131,139,131,138,13
0,138,130
7050 DIM c(16): RESTORE 7040
7060 FOR n=1 TO 16: READ c(n): N
EXT n
7070 FOR x=1 TO w: PRINT AT 1,x+
s;CHR$ 139;AT h+1,x+s;CHR$ 131:
NEXT x
7080 FOR y=1 TO h: PRINT AT y,s+
1;CHR$ 139;AT y,w+s+1;CHR$ 138:
NEXT y
7090 PRINT AT y,x+s;CHR$ 130
7100 DIM m$(h,w): DIM r$(INT (w*
h*.67),2): DIM n$(4,2): DIM f(4,
3)
7110 LET r=0: LET f$=CHR$ r
7120 LET x=INT (RND*w)+1: LET y=
INT (RND*h)+1
7130 LET m$(y,x)=CHR$ 100: PRINT
 AT y,x+s;CHR$ 139
7200 GO SUB 7300
7210 IF NOT r THEN GO TO 7900
7220 GO SUB 7500
7230 LET x=a: LET y=b
7240 GO TO 7200
7300 REM fcreate
7310 RESTORE 7020: FOR n=1 TO 4:
 FOR m=1 TO 3: READ f(n,m): NEXT
 m: NEXT n
7320 FOR n=1 TO 4
7330 LET z=INT (RND*4)+1: IF f(z
,1)=-1 THEN GO TO 7330
7340 IF f(z,3) THEN GO SUB 7400
7350 LET f(z,1)=-1
7360 NEXT n
7370 RETURN
7400 REM addcell
7410 IF m$(f(z,2),f(z,1))<>" " T
HEN RETURN
7420 LET m$(f(z,2),f(z,1))=f$: L
ET r=r+1
7430 LET r$(r)=CHR$ f(z,1)+CHR$
f(z,2)
7440 RETURN
7500 REM connect
7510 LET x=CODE r$(r,1): LET y=C
ODE r$(r,2): LET r=r-1
```

Grafikcodes

Grafik lesen

Ränder zeichnen

unten rechts ausfüllen

Grenze erzeugen

keine Zellen mehr da
neue Zellen hinzufügen

das bei allen Zellen

Umgebungswerte holen

Umgebungszelle wählen

wenn zulässig, zu Grenze
als 'gewählt' markiert
für alle vier wiederholen

ein Leerraum

Grenze machen

Koordinaten behalten

```
7520 GO SUB 7600: IF NOT n THEN
STOP
7530 LET n=INT (RND*n)+1: LET a=
CODE n$(n,1): LET b=CODE n$(n,2)
7540 GO SUB 7700
7550 LET z=x: LET x=a: LET a=z
7560 LET z=y: LET y=b: LET b=z
7570 GO SUB 7700
7580 RETURN
7600 REM findf
7610 RESTORE 7020: LET n=0
7620 FOR z=1 TO 4
7630 READ a,b,c: IF NOT c THEN G
O TO 7660
7640 IF m$(b,a)=" " OR m$(b,a)=f
$ THEN GO TO 7660
7650 LET n=n+1: LET n$(n)=CHR$ a
+CHR$ b
7660 NEXT z
7670 RETURN
7700 LET z=(a-x<>0)+(a-x>0)+(b-y
<>0)*4+(b-y>0)*4
7710 LET z=CODE m$(y,x)+z: IF z>
100 THEN LET z=z-100
7715 LET m$(y,x)=CHR$ z
7720 PRINT AT y,x+s;CHR$ c(z+1)
7730 RETURN
7900 LET in=INT (RND*(h-1))+2: L
ET out=INT (RND*(h-1))+1
7910 LET m$(in,1)=CHR$ (CODE m$(
in,1)+1)
7920 PRINT AT in,s;">";CHR$ c(CO
DE m$(in,1)+1)
7940 PRINT AT out,s+w+1;CHR$ 130
7950 RETURN
9000 REM line input
9010 INPUT (m$+" "); LINE a$: LE
T end=NOT LEN a$
9030 RETURN
9999 STOP
```

Erzeugungsfehler

Zelle Eins anschließ

Zelle Zwei anschließ.

neuen Weg wählen

ein Leerraum

zwei Zellen anschließ.

Ein- u. Ausg. wählen

Umwandl. nicht nötig

Einlage

Dieses kleine Programm verwandelt Ihren Spectrum in einen Zeichenblock. Sie können die Cursortasten (5, 6, 7 und 8) dazu verwenden, um einen imaginären Bleistift auf dem Schirm herumzuführen und Bilder ganz nach Wunsch zu zeichnen.

So wie es dasteht, ist das Programm sehr einfach, und zwar bewußt, damit Sie ausreichend Möglichkeiten zu Verbesserungen haben - diagonal bewegen, Zeilen löschen, zu einer vorbestimmten Position gehen, um nur ein paar zu erwähnen.

Der 'Bleistift' beginnt immer in der linken unteren Bildschirmecke.

```
10 LET x = 0: LET y = 0
20 PLOT x, y
30 LET x$ = INKEY$: IF NOT LEN x$ THEN
   GO TO 30
40 LET x = x- (x > 0)* (x$ = "5") + (x < 255)*
   (x$ = "8")
50 LET y = y- (y > 0)* (x$ = "6") + (y < 175)'
   (x$ = "7")
60 GO TO 20
```

Reversi – ein Strategiespiel

Spielautomaten bieten Sofortspaß. Sie sind großartig, wenn Freunde dabei sind, weil man da seine Kräfte messen und viele Leute stundenlang amüsieren kann.

Viele Spiele dieser Art erfordern aber wenig geistige Anstrengung. Man muß nur die Knöpfe rasch genug drücken können, mehr ist nicht verlangt.

Hier deshalb etwas, um die kleinen grauen Zellen ein bißchen anzuregen. Reversi gibt es schon seit vielen Jahren, und die Herkunft liegt ziemlich im dunkeln, obschon es Nachweise für ein ganz ähnliches Spiel in Ungarn aus dem 19. Jahrhundert gibt.

Man kann das Brettspiel unter Markennamen kaufen. Vielleicht verstaubt irgendwo bei Ihnen ein Exemplar, weil wie bei den meisten Strategiespielen zwei oder mehr Spieler nötig sind und man keinen Partner findet.

Nun, hier ist ein Partner, der keine Kopfschmerzen hat, der es länger aushält als Sie, und der nicht beleidigt ist, wenn Sie gewinnen.

Das Spiel wird auf einem Brett von acht mal acht Feldern von zwei Personen gespielt. Hier spielen Sie gegen den Spectrum. Sie haben die weißen Figuren, der Spectrum die schwarzen.

Bei dem Spiel kommt es darauf an, daß Sie auf dem Brett mehr von Ihren Spielsteinen haben als der Spectrum.

Bei jedem Zug können Sie eine Figur setzen. Die einzige Regel beim Setzen ist die, daß die Figur mindestens einer der gegnerischen 'in die Flanke fallen' muß. Das bedeutet, daß Sie die Figur(en) des Gegners zwischen dem eben gesetzten und einer schon auf dem Brett stehenden einschließen müssen, und zwar in beliebiger Richtung, vertikal, horizontal oder diagonal. Ein Beispiel sehen Sie unten.

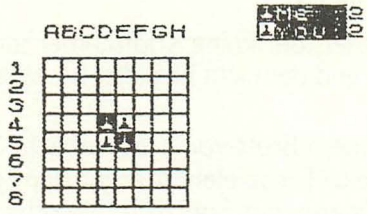
Alle Figuren des Gegners, die auf diese Weise eingeschlossen sind, werden durch solche von Ihrer eigenen Farbe ersetzt. Dann kommt der andere Spieler zum Zug.

Das Spiel ist zu Ende, wenn entweder alle Spielfelder besetzt sind oder kein Spieler mehr ziehen kann. Die Figuren werden gezählt. Gewonnen hat der Spieler mit den meisten eigenen Figuren auf dem Brett.

Für den Neuling klingt das außerordentlich einfach, aber die Strategie dieses Spiels ist erstaunlich kompliziert. Die in dieser Abart des Spiels eingebaute Strategie ist für höhere Ansprüche keine sehr schwierige, aber für die meisten Leute doch eine ganz ordentliche Herausforderung.

Das Brett wird auf der Bildschirmmitte angezeigt, oben und an der linken Seite mit Buchstaben beziehungsweise Ziffern als Hinweisen. Wenn Sie an der Reihe sind, geben Sie die Bezeichnung des Feldes ein, auf das Sie Ihre Figur setzen wollen. Das ist eine Bezeichnung aus zwei Zeichen, das erste ein Buchstabe A–H für die Spalte, das zweite eine Ziffer (1–8) für die Reihe. Wenn die Position nicht vorhanden ist, wird Ihnen das mitgeteilt.

Die Anfangsposition sieht so aus:



Sie können sich dafür entscheiden, als erster zu ziehen, oder das dem Spectrum überlassen. In der Anfangsposition können Sie eine Figur auf die Felder D3, C4, E6 oder F5 setzen. Das sind bei dieser Stufe die einzigen Felder, wo man Figuren des Gegners in die Flanke fallen kann.

Im Verlauf des Spiels nimmt die Zahl der Figuren, die Sie in die Zange nehmen können, entsprechend zu, und an der rechten oberen Bildschirmecke wird laufend die Zahl der eingesetzten Figuren jedes Spielers angezeigt.

Wenn Sie nicht ziehen können, geben Sie "x" ein. Der Spectrum prüft dann, ob Sie wirklich festsitzen, und empfiehlt Ihnen, wenn das zutrifft, einen Zug (nicht unbedingt den besten), damit Sie weitermachen können. Glauben Sie aber nicht, daß Ihnen das auf

die Siegesstraße verhilft! Ich kann Ihnen versichern, daß das nicht der Fall ist.

Sichern Sie das Programm mit:

SAVE "Reversi" **LINE** 1

```
    1 REM **** Reversi ****
    2 REM ©1982 Phipps Associates
    5 RESTORE : RANDOMIZE : CLEAR

   10 CLS : PRINT AT 0,7; INVERSE
  1;"**** Reversi ****"; INVERSE
0
   15 PRINT '" Version 2.0 v. 25.
 Oktober 1982"
   20 PRINT AT 10,8;"Spielfeld-Au
fbau"
   30 DEF FN y(p)=INT ((p-1)/8)
   40 DEF FN x(p)=(p-1)-FN y(p)*8
   50 DEF FN c$(p)=CHR$ (CODE "A"
+(p-2)-INT ((p-1)/10)*10)
   60 DEF FN r$(p)=CHR$ (CODE "0"
+INT ((p-1)/10))
1000 GO SUB 7200                        Brett etc. initialis.
1010 GO SUB 7300                        Brett zeichnen
1020 LET tc=0
1100 LET a$="Willst Du anfangen?
": GO SUB 9000
1110 LET t=(CHR$ CODE a$<>"J")
1200 LET t=NOT t: LET fc=0
1210 GO SUB 2000+t*1000                 Spielerwechsel
1220 IF cg=2 THEN GO TO 5000            wenn beide nicht
1400 LET b(p)=t+1: GO SUB 7500:         ziehen können...
GO SUB 7400                             Figuren setzen
1410 IF s(1)+s(2)=64 THEN GO TO         beide Fig. gesetzt
5100
1415 LET tc=tc+1
1420 GO TO 1200
2000 REM my move                        nächster Spieler
2005 PRINT AT 21,0;"Ich setze...
"
2010 FOR a=1 TO 64                      zuläss. Züge prüfen
2015 IF tc=0 THEN LET a=a+INT (R
ND*4)
2020 LET p=p(a): GO SUB 7000
2030 IF fc THEN GO TO 2100
2040 NEXT a
```

```
2050 BEEP .25,10: PRINT AT 21,0;
"Ich kann nicht setzen!"
2060 LET cg=cg+1: RETURN
2100 BEEP .25,10: PRINT AT 21,0;
"Ich setze ";FN c$(p);FN r$(p)
2110 LET cg=0
2120 RETURN
3000 REM your move
3010 LET a$="Du setzt (""x"" = k
ein Zug):": GO SUB 9000
3020 PRINT AT 21,0,,
3025 IF a$="X" THEN GO TO 3200
3030 IF LEN a$<>2 THEN GO TO 300
0
3040 LET x=CODE a$(1)-CODE "A":
IF x<0 OR x>7 THEN GO TO 3000
3050 LET y=CODE a$(2)-CODE "1":
IF y<0 OR y>7 THEN GO TO 3000
3060 LET p=(y+1)*10+x+2: GO SUB
7000
3070 IF fc=0 THEN PRINT AT 21,0;
"Unerlaubter Zug - Nochmal!": GO
 TO 3000
3080 LET cg=0
3100 RETURN
3200 REM cheat check
3205 PRINT AT 21,0;"Ich denke na
ch..."
3210 FOR a=60 TO 1 STEP -1: LET
p=p(a): GO SUB 7000: IF NOT fc T
HEN NEXT a: LET cg=cg+1: PRINT A
T 21,0,,: RETURN
3300 PRINT AT 21,0; FLASH 1;"BET
RUG!"; FLASH 0;" Du kannst auf "
;FN c$(p);FN r$(p)
3310 GO TO 3000
5000 REM game over
5010 PRINT AT 20,0;"Keiner kann
setzen."
5100 PRINT AT 21,0,,
5110 IF s(1)=s(2) THEN PRINT AT
21,0;"Wir haben gleich viel Punk
te.": GO TO 5200
5120 LET a$="Ich bin": IF s(2)>s
(1) THEN LET a$="Du bist"
5130 PRINT AT 21,0;a$;" Sieger."
5200 LET a$="Noch ein Spiel?": G
O SUB 9000: IF CHR$ CODE a$="J"
```

Flagge nicht zu beweg.

großes X
Zug bestätigen

prüf., ob hier richtig

prüfen, ob
zulässiger Zug

```
THEN RUN
5210 GO TO 9999
6999 STOP
7000 REM check legal move (p)
7001 REM t=0 is my move
7010 LET fc=0
7020 IF b(p) THEN RETURN
7030 FOR i=1 TO 8: LET mc=0
7040 LET xp=p+i(i): IF b(xp)<>(N
OT t)+1 THEN GO TO 7100
7050 LET mc=mc+1: LET t(fc+mc)=x
p: LET xp=xp+i(i): IF b(xp)=(NOT
 t)+1 THEN GO TO 7050
7060 IF b(xp)<>t+1 THEN LET mc=0
7100 LET fc=fc+mc
7110 NEXT i
7120 RETURN
7200 REM initialise
7210 DIM b(100): DIM i(8): DIM t
(30): DIM p(64): DIM s$(2,4)
7220 RESTORE 9900
7230 FOR x=1 TO 64: READ p: LET
p(p)=(FN y(x)+1)*10+FN x(x)+2: N
EXT x
7250 FOR x=0 TO 9: LET b(x+1)=9:
 LET b(91+x)=9: LET b(10*x+1)=9:
 LET b(10*x+10)=9: NEXT x
7260 LET b(45)=1: LET b(46)=2: L
ET b(55)=2: LET b(56)=1
7270 FOR x=1 TO 8: READ i(x): NE
XT x
7280 LET s$(1)="Ich:": LET s$(2)
="DU: "
7285 GO SUB 9500
7290 RETURN
7300 REM draw board
7305 CLS
7310 FOR x=0 TO 8: PLOT 96+x*8,1
44: DRAW 0,-64: NEXT x
7320 FOR y=0 TO 8: PLOT 96,144-(
y*8): DRAW 64,0: NEXT y
7330 PRINT AT 2,12; INK 4;"ABCDE
FGH"
7340 FOR y=1 TO 8: PRINT AT 3+y,
10; INK 4;y: NEXT y
7400 REM display positions
7410 DIM s(2)
7420 FOR y=0 TO 7: FOR x=0 TO 7:
```

Feld schon besetzt
Felder rundum prüf.

```
 LET z=b((y+1)*10+x+2): IF z THE
N PRINT AT y+4,x+12;CHR$ (143+z)
: LET s(z)=s(z)+1
7430 NEXT x: NEXT y
7440 FOR z=1 TO 2: PRINT AT z,24
; INK 2;CHR$ (z+143); INK 0; INV
ERSE 1;s$(z); INVERSE 0;s(z);" "
: NEXT z
7490 RETURN
7500 REM update board
7510 FOR f=1 TO fc: LET b(t(f))=
t+1: NEXT f
7520 RETURN
9000 REM line input
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
" AND a$(x)<="z")): NEXT x
9030 RETURN
9500 REM graphics
9510 DATA 2
9530 DATA "A",255,247,247,247,22
7,193,255,255
9540 DATA "B",128,136,136,136,15
6,190,128,255
9550 RESTORE 9510: READ n
9560 FOR x=1 TO n
9570 READ c$: FOR r=0 TO 7: READ
 b: POKE USR c$+r,b: NEXT r
9580 NEXT x
9590 RETURN
9900 REM board priorities
9910 DATA 1,49,5,17,18,6,50,2,51
,57,41,33,34,42,58,52,7,43,13,25
,26,14,44,8,19,35,27,61,62,28,36
,20,21,37,29,63,64,30,38,22,9,45
,15,31,32,16,46,10,53,59,47,39,4
0,48,60,54,3,55,11,23,24,12,56,4
9920 DATA -11,-10,-9,-1,1,9,10,1
1
```

ein Leerraum

3 D-Labyrinth

Dieses Programm führt wirklich das Höchste an Labyrinthen vor - ein Labyrinth in drei Dimensionen. Wie bei allen Labyrinthen geht es darum, wieder herauszukommen, aber zuerst müssen Sie das Ziel finden! Bei den üblichen Labyrinthen sieht man den Weg nach draußen, aber hier müssen Sie durch die Räume wandern, um ihn zu entdecken.

Der Blick auf jeden Raum ist so, als stünden Sie in der Nähe des Mittelpunkts und hätten die Rückwand vor sich. Oben am Bildschirm ist die Decke, links und rechts befinden sich Wände. Gänge werden gezeigt, wo sie vorhanden sind, und Sie haben die Wahl, welchen Sie betreten wollen.

Sie spielen in einem Würfel mit vier, fünf oder sechs Räumen pro Kante, je nach dem von Ihnen gewählten Schwierigkeitsgrad. Sie beginnen an einer Zufallsposition im Labyrinth und müssen dann den Weg nach draußen finden. Benützen Sie die Cursortasten, um sich nach links (5), rechts (8) oder geradeaus (7) zu bewegen. Sie können auch hinauf (U) und hinunter (D), wenn Tunnels das gestatten. Damit Sie hinter sich blicken können, müssen Sie jedoch "B" drücken, was Sie um eine halbe Drehung herumschiebt, ohne daß Sie den Raum verlassen.

Mit Ausnahme der vertikalen Bewegung verändert sich Ihr Blick auf das Labyrinth, wenn Sie sich nach rechts oder links oder ganz herumdrehen.

Bei Beginn des Spiels erhalten Sie mehrere Möglichkeiten, wie das Spiel ausgeführt werden kann. Sie können gegen die Zeit spielen - da haben Sie 240 Sekunden Zeit herauszufinden, was recht spannend wird. Außerdem können Sie dafür sorgen, daß die Lage der Räume neben der Ausgangsraum-Nummer angezeigt wird. Dadurch läßt sich viel leichter gewinnen. Für den Anfänger ist das also ideal. Aus diesem Grund wird das Zeitlimit auf 120 Sekunden verringert, wenn Sie gegen die Zeit spielen wollen und die Raumnummern angezeigt werden.

Es gibt mehrere strategische Methoden, einen Sieg zu erringen, aber darauf sollen Sie selbst kommen!

Bitte denken Sie daran, daß Sie Geduld haben müssen, während das Labyrinth hervorgebracht wird. Es kann mehrere Minuten dauern, wenn Sie die schwierigste Stufe wählen. Ist das Spiel zu

Ende, erhalten Sie die Möglichkeit, im selben Labyrinth noch einmal zu spielen, so daß Sie zwischen den Spielen nicht zu warten brauchen. Es wird ein neuer Ausgang bestimmt, so daß Sie sich nicht darauf verlassen können, den Weg zum alten Ausgang zu finden.

Wenn Sie für das Spiel mehr Zeit zulassen wollen, verändern Sie Zeile 1070. Die Variable "LIMIT" muß auf die gewünschte Anzahl von Sekunden gesetzt werden.

Stellen Sie das Programm wie gewohnt sicher mit:

SAVE "3D MAZE" **LINE** 1

```
   1 REM ******* 3D Maze *******
   2 REM ©1982 Phipps Associates
   5 RANDOMIZE : RESTORE
  10 LET f=0: BORDER f: CLS : GO Anfangsdisplay
 SUB 6500
  15 PLOT VAL "38",VAL "21": DRA
W f,VAL "60": DRAW VAL "32",f: D
RAW f,VAL "-60": CIRCLE VAL "42"
,VAL "51",VAL "2"
  20 PRINT AT VAL "6",VAL "8"; I
NK VAL "2"; INVERSE NOT f;"*** H
OEHLEN ***"
  25 LET a$="Willst Du die Spiel
anleitung?": GO SUB 9000: IF CHR
$ CODE a$="J" THEN GO SUB 9600
  30 LET a$="Welche Groesse? (1-
3)": GO SUB 9000: IF end THEN GO
 TO 9999
  40 LET x=VAL a$: IF x<VAL "1"
OR x>VAL "3" THEN GO TO 30
  50 LET h=x+VAL "3": LET w=h: L Würfelgröße setzen
ET v=h
 100 GO SUB 7000                  Labyrinth erzeugen
 200 FOR i=NOT f TO VAL "10": BE Ton bereit
EP .05,10: BEEP .05,15: NEXT i
 210 LET a$="Spiel gegen die Zei
t? (J/N)": GO SUB 9000: LET cloc
k=(CHR$ CODE a$="J")
 220 LET a$="Koordinaten-Anzeige
? (J/N)": GO SUB 9000: LET dc=(C
HR$ CODE a$="J")
1000 GO SUB 6800                  Ausgang wählen
1010 GO SUB 6900                  Startposition wählen
1020 DIM d(VAL "4",VAL "4"): DIM
 f(VAL "6"): DIM g(VAL "4",VAL "
```

```
6"): DIM d$(VAL "6"): DIM t$(VAL
 "6",VAL "3")
1030 RESTORE 2000: FOR i=VAL "1"
 TO VAL "4": FOR j=VAL "1" TO VA
L "4": READ d(i,j): NEXT j: FOR
j=VAL "1" TO VAL "6": READ g(i,j
): NEXT j: NEXT i
1035 FOR i=VAL "1" TO VAL "6": R
EAD t$(i): NEXT i
1040 LET d=VAL "1"
1050 LET d$="7B85UD"
1060 POKE 23673,f: POKE 23672,f
1070 LET lt=f: LET limit=240+(h-
4)*60
1080 IF dc THEN LET limit=(limit
/VAL "2")
1100 LET i=CODE m$(z,y,x)
1110 FOR p=5 TO f STEP -1
1120 LET f(p+1)=INT (i/2^p)
1130 LET i=i-f(p+1)*2^p
1140 GO SUB 6700: IF NOT t THEN
GO TO 5200
1150 NEXT p
1200 GO SUB 6500
1210 FOR p=1 TO 6: IF f(p) THEN
GO SUB g(d,p)
1220 GO SUB 6700: IF NOT t THEN
GO TO 5200
1230 NEXT p
1300 GO SUB 6700: IF NOT t THEN
GO TO 5200
1305 IF dc THEN PRINT #f;AT 1,f;
"Raum: ";z;y;x;TAB 19;"Ausgang i
n: ";iz;iy;ix;
1310 LET x$=INKEY$: IF NOT LEN x
$ THEN GO TO 1300
1320 IF CODE x$>96 THEN LET x$=C
HR$ (CODE x$-32)
1330 FOR p=1 TO 6: IF x$<>d$(p)
THEN NEXT p: GO TO 1300
1335 IF p=2 THEN LET d=d(d,p): G
O TO 1100
1340 IF p<5 THEN GO TO 1400
1350 LET j=VAL t$(p)
1360 IF NOT f(p) THEN GO TO 1300
1365 IF j=f OR j>v THEN GO TO 50
00
1370 LET z=j
```

Gänge zeichnen

zulässige Bewegungen

zuläss. Zeichen

Uhr laufen lassen

Zeitlimit setzen

Hälfte für leicht

Raumbeschreib. holen

Verbindungen bestimm.

verbliebene Zeit messen

Raum zeichnen

Zeit prüfen

Zeit prüfen

zulässige Taste?

umdrehen

horizont. Beweg.

neue vertik. Bewegung

prüf., ob Gang da

auf Ausgang prüfen

neuer Ort

```
1380 GO TO 1100                          neuen Raum zeichnen
1400 IF NOT f(d(d,p)) THEN GO TO         auf Gang prüfen
 1300
1410 LET j=VAL t$(d(d,p))                Koordinaten aktualis.
1420 IF j=0 OR j>v THEN GO TO 50         auf Ausgang prüfen
00
1440 IF t$(d(d,p),1)="x" THEN LE
T x=j: GO TO 1460
1450 LET y=j
1460 LET d=d(d,p)                        Perspektive verschieb.
1470 GO TO 1100                          neuen Raum zeichnen
2000 REM  b,f,r,l  Perspektive verschieben und zeichnen
2010 DATA VAL "1",VAL "2",VAL "3
",VAL "4",VAL "6000",VAL "6600",
VAL "6200",VAL "6100",VAL "6300"
,VAL "6400"
2020 DATA VAL "2",VAL "1",VAL "4
",VAL "3",VAL "6600",VAL "6000",
VAL "6100",VAL "6200",VAL "6300"
,VAL "6400"
2030 DATA VAL "3",VAL "4",VAL "2
",VAL "1",VAL "6100",VAL "6200",
VAL "6000",VAL "6600",VAL "6300"
,VAL "6400"
2040 DATA VAL "4",VAL "3",VAL "1
",VAL "2",VAL "6200",VAL "6100",
VAL "6600",VAL "6000",VAL "6300"
,VAL "6400"
2100 DATA "x-1","x+1","y-1","y+1
","z-1","z+1"
5000 FOR j=VAL "-20" TO VAL "20"         Gratulation
: BEEP VAL ".01",j: NEXT j
5010 CLS : PRINT AT VAL "10",VAL
 "10"; FLASH VAL "1";"DU BIST DR
AUSSEN!"
5020 IF NOT clock THEN GO TO 510
0
5030 PRINT ''"Uebrige Zeit:";t;"
 Sekunden"
5040 IF t<VAL "10" THEN PRINT "O
h, das war knapp."
5050 IF t>=VAL "10" AND t<=VAL "
30" THEN PRINT "Ganz schoen schn
ell..."
5060 IF t>=VAL "30" THEN PRINT "
Dafuer mach ich mir die Arbeit..
."
5100 LET a$="Noch ein Durchgang?
```

```
": GO SUB 9000: IF CHR$ CODE a$<
>"J" THEN GO TO 9999
5110 LET a$="Die selbe Hoehle? "
: GO SUB 9000: IF CHR$ CODE a$="
J" THEN LET d=-id: LET z=iz: LET
 y=iy: LET x=ix: GO SUB 6850: GO
 TO 200
5120 GO TO 10
5200 FOR x=VAL "20" TO VAL "-20"
 STEP VAL "-1": BEEP VAL ".01",x
: NEXT x
5210 CLS : PRINT AT VAL "10",VAL
 "14";"Schade!"''TAB VAL "4";"De
ine Zeit ist um."
5220 GO TO 5100
6000 PLOT VAL "96",VAL "67": DRA   Gang voraus
W f,VAL "42": DRAW VAL "64",f: D
RAW f,VAL "-42": DRAW VAL "-64",
f
6010 PLOT VAL "112",VAL "77": DR
AW f,VAL "21": DRAW VAL "32",f:
DRAW f,VAL "-21": DRAW VAL "-32"
,f
6020 PLOT VAL "96",VAL "67": DRA
W VAL "16",VAL "10": PLOT VAL "9
6",VAL "109": DRAW VAL "16",VAL
"-11": PLOT VAL "160",VAL "109":
 DRAW VAL "-16",VAL "-11": PLOT
VAL "160",VAL "67": DRAW VAL "-1
6",VAL "10"
6030 FOR j=VAL "77" TO VAL "98":
 PLOT VAL "112",j: DRAW VAL "32"
,f: NEXT j
6040 RETURN
6100 PLOT f,VAL "44": DRAW VAL "   Gang links
16",VAL "8": DRAW f,VAL "72": DR
AW VAL "-16",VAL "8"
6110 PLOT VAL "16",VAL "52": DRA
W VAL "-16",f: PLOT VAL "16",VAL
 "124": DRAW VAL "-16",f
6120 RETURN
6200 PLOT VAL "255",VAL "44": DR   Gang rechts
AW VAL "-16",VAL "8": DRAW f,VAL
 "72": DRAW VAL "16",VAL "8"
6210 PLOT VAL "239",VAL "52": DR
AW VAL "16",f: PLOT VAL "239",VA
L "124": DRAW VAL "16",f
6220 RETURN
```

```
6300 PLOT VAL "64",VAL "175": DR
AW VAL "8",VAL "-10": DRAW VAL "
112",f: DRAW VAL "8",VAL "10"
6310 PLOT VAL "72",VAL "165": DR
AW f,VAL "10": PLOT VAL "184",VA
L "165": DRAW f,VAL "10"
6320 RETURN
6400 PLOT VAL "64",f: DRAW VAL "
8",VAL "10": DRAW VAL "112",f: D
RAW VAL "8",VAL "-10"
6410 PLOT VAL "72",VAL "10": DRA
W f,VAL "-10": PLOT VAL "184",VA
L "10": DRAW f,VAL "-10"
6420 RETURN
6500 CLS
6510 PLOT f,VAL "175": DRAW VAL
"255",f: DRAW f,VAL "-175": DRAW
 VAL "-255",f: DRAW f,VAL "175"
6520 PLOT VAL "32",VAL "155": DR
AW VAL "192",f: DRAW f,VAL "-134
": DRAW VAL "-192",f: DRAW f,VAL
 "134"
6530 PLOT f,VAL "175": DRAW VAL
"32",VAL "-20": PLOT VAL "255",V
AL "175": DRAW VAL "-32",VAL "-2
0": PLOT VAL "255",f: DRAW VAL "
-32",VAL "21": PLOT f,f: DRAW VA
L "32",VAL "21"
6540 RETURN
6700 REM time
6710 IF NOT clock THEN LET t=NOT
 f: RETURN
6720 LET s=PEEK 23672+256*PEEK 2
3673
6730 LET t=limit-INT (s/50)
6740 PRINT #f;AT f,28;t;" "
6750 IF t<>lt THEN BEEP .1,-8: L
ET lt=t
6760 RETURN
6800 REM choose exit
6810 GO SUB 6900
6820 DATA x=1,1,x=4,2,y=1,4,y=4,
8,z=1,16,z=4,32
6830 RESTORE 6820
6840 FOR j=1 TO 6: READ c,d: IF
NOT c THEN NEXT j: GO TO 6800
6850 LET m$(z,y,x)=CHR$ (CODE m$
(z,y,x)+d)
```

Gang aufwärts

Gang abwärts

Raumumriß

keine Uhr nötig

abgelauf. Zeit holen

verblieb. Sekunden

ein Leerraum

jede Sekunde 1 Ton

```
6860 LET ix=x: LET iy=y: LET iz=
z: LET id=d
6870 RETURN
6900 LET x=INT (RND*w)+1: LET y=
INT (RND*h)+1: LET z=INT (RND*v)
+1
6910 RETURN
6999 STOP
7000 REM create maze                    s. Phantastisch!
7010 PRINT AT VAL "18",VAL "5";"
Bitte warte ";v-1;" Minuten"
7020 DATA x-1,y,z,x>1,x+1,y,z,x<
w
7030 DATA x,y-1,z,y>1,x,y+1,z,y<
h
7040 DATA x,y,z-1,z>1,x,y,z+1,z<
v
7100 DIM m$(v,h,w): DIM r$(INT (
v*h*w*VAL ".67"),VAL "3"): DIM n
$(VAL "6",VAL "3"): DIM f(VAL "6
",VAL "4")
7105 FOR z=1 TO v: FOR y=1 TO h:  sechs kleine z
 LET m$(z,y)="©©©©©©": NEXT y: N
EXT z
7110 LET r=f: LET f$=CHR$ r
7120 LET x=INT (RND*w)+1: LET y=
INT (RND*h)+1: LET z=INT (RND*v)
+1
7130 LET m$(z,y,x)=CHR$ VAL "100
"
7200 GO SUB 7300
7210 IF NOT r THEN RETURN
7220 GO SUB 7500
7230 LET x=a: LET y=b: LET z=c
7240 GO TO 7200
7300 RESTORE 7020: FOR n=1 TO 6:
 FOR m=1 TO 4: READ f(n,m): NEXT
 m: NEXT n
7310 FOR n=1 TO 6
7320 LET j=INT (RND*6)+1: IF f(j
,1)=-1 THEN GO TO 7320
7330 IF f(j,4) THEN GO SUB 7400
7340 LET f(j,1)=-1
7350 NEXT n
7360 RETURN
7400 IF m$(f(j,3),f(j,2),f(j,1))  kleines z
<>"©" THEN RETURN
7410 LET m$(f(j,3),f(j,2),f(j,1)
```

```
)=f$: LET r=r+1
7420 LET r$(r)=CHR$ f(j,1)+CHR$
f(j,2)+CHR$ f(j,3)
7430 RETURN
7500 LET x=CODE r$(r,1): LET y=C
ODE r$(r,2): LET z=CODE r$(r,3):
 LET r=r-1
7510 GO SUB 7600: IF NOT n THEN
STOP
7520 LET n=INT (RND*n)+1: LET a=
CODE n$(n,1): LET b=CODE n$(n,2)
: LET c=CODE n$(n,3)
7530 GO SUB 7700
7540 LET j=x: LET x=a: LET a=j
7550 LET j=y: LET y=b: LET b=j
7560 LET j=z: LET z=c: LET c=j
7570 GO SUB 7700
7580 RETURN
7600 RESTORE 7020: LET n=f
7610 FOR j=1 TO 6
7620 READ a,b,c,d
7630 IF d THEN IF m$(c,b,a)<>"@" kleines z
 AND m$(c,b,a)<>f$ THEN LET n=n+
1: LET n$(n)=CHR$ a+CHR$ b+CHR$
c
7640 NEXT j
7650 RETURN
7700 LET j=(a-x<>f)+(a-x>f)+(b-y
<>f)*4+(b-y>f)*4+(c-z<>f)*16+(c-
z>f)*16
7710 LET j=CODE m$(z,y,x)+j: IF
j>100 THEN LET j=j-100
7720 LET m$(z,y,x)=CHR$ j
7730 RETURN
8900 LET a$="Zur Fortsetzung ENT
ER druecken"
9000 REM line input
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
")): NEXT x
9030 RETURN
9600 REM instructions
9610 CLS : PRINT TAB 8;"*** HOEH
LEN ***"
9620 PRINT '"Du bist in einem zu
```

```
faellig ge"'"waehlten Raum des d
reidimen="'"sionalen Hoehlensyst
ems.Jeder"'"Raum hat bis zu 6 Au
sgaenge."'"Dein Blick ist jeweil
s von"'"der Mitte des Raumes in
die"'"Richtung, in die Du zuletz
t"'"gelaufen bist."
9630 PRINT "Hier siehst Du, mit
welchen Ta="'"sten Du Blick- und
 Bewegungs="'"richtung aendern k
annst:"
9640 PRINT "7 - vorwaerts"'"5 -
links"'"8 - rechts"'"U - nach ob
en"'"D - nach unten"'"B - 180 Gr
ad-Drehung"
9650 GO SUB 8900
9660 CLS
9670 PRINT "Du kannst waehlen, o
b Du gegen"'"die Zeit spielen mo
echtest,ob"'"Du jeweils die Raum
koordina="'"ten angezeigt haben
moechtest"'"und wie gross das Ho
ehlensy="'"stem sein soll."
9680 PRINT "ACHTUNG! Deine Blick
richtung    geht immer in die Ri
chtung, in  die Du zuletzt gelau
fen bist.   Dies gilt jedoch NIC
HT fuer Auf-und Abwaertsbewegung
en."
9690 PRINT '" VIEL GLUECK!"
9700 GO TO 8900
9999 STOP
```

Nützliche Subroutinen (Unterprogramme)

Das ist die Art von Abschnitt, nach der ich in jedem Buch suche, das ich mir kaufe! Die meisten Programme benötigen immer wieder dieselben Subroutinen, und trotzdem schreiben manche Leute sie jedesmal, vielleicht mit ein paar Veränderungen, ganz neu.

Fangen wir mit einer Allzweck-Eingabe-Subroutine an. Sie wandelt zusätzlich alle Eingaben in Großbuchstaben um. Sie mag zwar nicht dauernd nötig sein, kann aber bei allgemeiner Verwendung viel Programmplatz sparen, wenn ein Programm nur auf große (oder wenn Sie wollen, kleine) Buchstaben prüft. Nehmen Sie den Befehl:

```
200 IF x$ = "Y" OR x$ = "y" THEN GO TO 300
```

Tritt so etwas in einem Programm mehrmals auf, kann es ziemlich viel Platz und Laufzeit kosten.

Die folgende Subroutine verlangt, daß die Stringvariable a$ ein Anforderungszeichen enthält, das vor den Eingabedaten angezeigt wird. Beim Rücksprung enthält die Stringvariable a$ die eingegebenen Daten in Großbuchstaben, während die Variable "END" auf wahr gesetzt wird, falls ein leerer Inputstring eingegeben wurde. Das würde normalerweise behandelt werden als 'Ende des Laufs' oder 'verwende einen Vorgabewert' - siehe 'Ja/Nein-Input'.

```
9000 REM Zeileneingabe
9010 INPUT (a$ + ""); LINE a$:
     LET end = NOT LEN a$
9020 FOR x = 1 TO LEN a$
9030 LET a$ (x) = CHR$ (CODE a$ (x) -32*
     (a$ (x) > = "a" AND a$ (x) < = "z"))
9040 NEXT x
9050 RETURN
```

Ja/Nein-Input

Diese Art von Eingabe wird vermutlich häufiger verwendet als irgendeine andere, weil wegen der mangelnden 'Formen'-Fähig-

keit von BASIC Daten in der Regel zeilenweise eingegeben werden. Benutzer von größeren Systemen haben den Vorteil der Eingabe im 'Seitenmodus', wo der Computer einen ganzen Bildschirm voll Daten auf einmal schlucken kann.

ZX-BASIC hat in meinen Augen eine Einschränkung: Wenn man bei einer Stringvariablen einen Slicer verwendet, also aus dem String etwas herausnehmen will, und der Slicerwert zu hoch ist, erscheint Fehlermeldung 3. Mir wäre ein Nullstring (Länge Null) lieber, weil dann das Programmieren eleganter erfolgen könnte.

Das läßt sich mit ein bißchen Jonglieren umgehen (siehe 'Winke und Tips'), und normalerweise genügt es bei Ja/Nein-Input, sich das erste Zeichen anzusehen. Sie würden überhaupt nur auf "y" prüfen und alle anderen Antworten als "Nein" behandeln. Das nennt man einen **Vorgabewert.**

Sobald eine Frage mit einer Ja/Nein-Antwort gestellt werden soll, verwenden Sie folgende Zeilen:

```
100 LET m$ = "Ist das eine Frage?: GO SUB 9000
110 IF CHR$ CODE a$ = "Y" THEN GO TO ****
120 REM alles andere als Nein behandeln
```

Das verwendet die obige Subroutine, aber noch wichtiger dabei ist, daß Zeile 110 (oder was Sie sonst verwenden) nicht zu einem Fehler führt, wenn nichts eingegeben wird.

Umwandlung Dezimal/Hex

Sie werden diese Routine im Abschnitt Maschinencode dieses Bandes häufig verwendet sehen. Sie verlangt, daß in Variable H ein Wert gegeben wird und beim Rücksprung H$ den entsprechenden Wert in Hex enthält.

Sie können die Subroutine auf zweifache Art eingeben - **GO SUB** 9100, wobei H$ eine Hexzahl von vier Zeichen enthält, oder **GO SUB** 9120, wo es eine Hexzahl von zwei Zeichen hat. Mit zwei verschiedenen Eingaben können Sie 8- oder 16 Bit-Werte in Hex übertragen.

```
9100 REM let h$ = hex $ (h)
9110 LET h$ = "": GO TO 9130          (vier Leerräume)
9120 LET h$ = ""                      (zwei Leerräume)
9130 LET h1 = h: FOR x = LEN h$ TO 1 STEP -1:
```

```
        LET x 1 = h 1-INT (h1/16)*16:
        LET h$ (x) = CHR$ (x 1 + CODE "0" + 7*
        (x 1 > 9)):
        LET h 1 = INT (h1/16)
        NEXT x
9140 RETURN
```

Umwandlung Hex/Dezimal

Das ist natürlich die Umkehrung der obigen Subroutine. Sie verlangt eine Variabel H$, die eine Hexzahl in Großbuchstaben enthält. Beim Ausgang enthält die Variable H$ den Dezimalwert.

```
9200 REM let h = dez (h$)
9210 LET h = 0:
        FOR x = 1 TO LEN h$:
        LET h = h*16 + CODE h$ (x)-48-7* (h$ (x) > "9"):
        NEXT x
9220 RETURN
```

Stringerkennung

Sie werden ziemlich häufig prüfen müssen, ob ein String einen zweiten enthält. Ein Hauptbeispiel hier ist, wenn ein Programm den Benutzer bittet, eine Antwort auf eine Frage einzugeben:

Wie heißt die Hauptstadt von Niedersachsen?

Gibt der Spieler 'Hannover' ein, ist alles gut, weil das Programm vermutlich darauf prüft (und Großbuchstaben berücksichtigt). Aber wenn eingegeben wird "Hauptstadt ist Hannover" oder "Das ist Hannover", was hat man für Chancen?

Die folgende Subroutine erfordert zwei Variable - w$, die den zu suchenden String enthalten sollte, und x$, die den String enthalten sollte, der in w$ zu finden ist.

Beim Rücksprung wird die Variable X danach gesetzt, ob X$ in W$ gefunden ist oder nicht. Wenn nicht, erhält X den Wert Null. Wenn doch, enthält X die Position des Vorkommens von X$ innerhalb von W$.

Hier ein paar Beispiele, die W$, X$ und den Wert von X beim Rücksprung zeigen:

W$ enthält	X$ enthält	X beim Rücksprung
Hannover	Hannover	1
Es ist Hannover	Hannover	7
Ist es Bonn?	Hannover	0
Es ist Hannover	ist	4

```
9300 REM let x = instr (w$, x$)
9310 FOR x = 1 TO LEN w$-LEN x$ + 1:
     IF x$ = w$ (x TO x + LEN x$ -1)
     THEN RETURN
9320 NEXT x: LET x = 0
9330 RETURN
```

Numerische Umwandlung

Der Spectrum enthält einige überaus leistungsfähige Funktionen für die Prüfung von Datenwerten, aber es gibt keine Funktion, die einem Programm gestattet würde, zu prüfen, ob ein gegebener String eine zulässige Zahl enthält.

Programme erfordern möglicherweise die Verwendung von INPUT LINE a$, um einen String einzugeben, da die Verwendung von INPUT mit einer numerischen Variable zu sonderbaren Folgen führt, wenn versehentlich Buchstaben eingegeben werden. Probieren Sie:

```
10 INPUT x
```

Geben Sie, wenn das gefahren wird, Werte ein wie 10, 230, RAT, X und anderes, um zu sehen, was dann geschieht.

Die folgende Subroutine verlangt, daß die Variable A$ einen String enthält, der darauf geprüft werden soll, ob er eine zulässige Zahl ist. Beim Rücksprung enthält Variable X einen Zeiger zum letzten gültigen numerischen Zeichen. Er kann dann dazu benützt werden, ein angemessenes Vorgehen zu bestimmen. Beispiel:

A$ enthält	X bei Rücksprung
"1234"	4
"1X34"	1
"Rat"	0
"12 Baumweg"	2

Nach dieser Information könnte ein Programm Zeilen wie die folgenden enthalten:

```
1100 GO SUB 9400
1110 IF x = 0 THEN PRINT "Das ist keine Zahl!":
     GO TO ****
```

oder ...

```
1100 GO SUB 9400: IF NOT x THEN GO TO 1200
1110 LET Wert = VAL a$ (TO x)
```

Hier die Subroutine:

```
9400 REM let x = numptr (a$)
9410 FOR x = 1 TO LEN a$:
     IF a$ (x) > "0" AND a$ (x) < "9" THEN NEXT x
9420 LET x = x - 1
9430 RETURN
```

Zahlen anzeigen mit Randausgleich

Viele Programme, die Zahlen anzeigen oder ausdrucken, würden sehr davon profitieren, wenn die Werte in Spalten erschienen, nach dem Dezimalpunkt ausgerichtet. Die untenstehende Subroutine zeigt die Zahl in der Variablen V in Spalten an, wobei die rechte nach der Tabulatorposition in Variable C ausgerichtet wird. Verlangt der anzuzeigende Wert mehr Spalten als vorgesehen, wird die Zahl in der nächstfolgenden Zeile angezeigt.

In den meisten Fällen verlangt die Zahl zwei Dezimalstellen (bei Geldbeträgen), aber in Fällen, wo mehr gebraucht wird, können Sie die Variable N auf die beim Ausdrucken erforderliche Zahl von Dezimalstellen setzen.

Beim normalen Gebrauch würde N daher für den ganzen Lauf des Programms auf 2 gesetzt werden. Sie ändern die Subroutine vielleicht lieber ab, um die Variable N überhaupt nicht verwenden zu müssen - abgesehen davon, daß man durchgehend "n" durch "2" ersetzen kann, lassen sich die Zeilen 9510 und 9520 ersetzen durch die Einzelzeile 9510 **LET** z$ = "00".

```
9500 REM v in Spalten bei c mit n Dezimalstellen
9510 LET z$ = ""                    (String initialisieren)
9520 FOR z = 1 TO n:LET
     z$ = z$ + "0": NEXT z          (Dezimalstellen)
```

```
9530 LET x 1 = INT ABS v:          (Pfund und Pence)
     LET xp = INT ((ABS v - x 1)* 10 n)
9540 LET z$ = ("-" AND v < 0) + STR$ x 1 + "."
     (z$ (TO n - LEN STR$ xp) +
     (STR$ xp + z$)) (TO n)        (Bild erzeugen)
9550 PRINT TAB
     (c - LEN z$ + 1); z$;         (es anzeigen)
9560 RETURN
```

Nun ist es einfach, den Anfangsteil dieses Unterprogramms so zu erweitern, daß Anzeigemasken verwendet werden können. Statt die Variable n auf die Zahl von Dezimalstellen zu setzen, soll die Variable u$ auf eine 'Maske' der erwünschten Anzeige gesetzt werden. Die Maske sollte ein Zeichen "9" immer dort enthalten, wo eine Zahl angezeigt werden soll, und an der Position des Dezimalpunktes einen Punkt. So würde LET u$ = "999-99.99" die Routine veranlassen, den Wert in v mit zwei Dezimalstellen anzuzeigen. Die Zahl wird, falls sie zu groß ist, (von links her) gekürzt.

```
9500 REM v mit Maske u$ bei Spalte c anzeigen
9510 LET z$ = "": LET x 1 = 0
9515 FOR z = 1 TO LEN u$          (Dezimalstellen zählen)
9520 IF x 1 THEN LET z$ = z$ + "0"
9525 IF u$ (z) = "." THEN LET x 1 = 1
9530 NEXT z: LET x 2 = LEN z$
9535 LET x 1 = INT ABS v:
     LET xp = INT
     ((ABS v - x 1)* 10^1 x 2)     (ganze Zahl u. Brüche)
9540 LET z$ = ("-" AND v < 0)
     + STR$ x 1 + ("." + (z$
     (TO n - LEN STR$ xp) +
     (STR$ xp + z$)) (TO n)
     AND x 2)                      (Bild erzeugen)
9550 IF LEN z$ > LEN u$
     THEN LET z$ = z$ (LEN
     z$ - LEN u$ + 1 TO)           (notfalls kürzen)
9555 PRINT TAB (c - LEN
     z$ + 1); z$                   (Bild anzeigen)
9560 RETURN
```

Kontoführung

Viele Leute sehen im Spectrum mehr als eine Spielmaschine. Für sie kann er ihre Sammlungen verwalten, im Geschäft von Nutzen sein und sogar in begrenztem Maß als Textautomat dienen.

Das vorige Kapitel befaßte sich mit der Verwendung von Standardroutinen, und in diesem können Sie sehen, wie die meisten davon praktisch gut genutzt werden. Mit dem Programm können Sie über Ihre Ein- und Ausgaben Buch führen, ohne bei der Bank anfragen oder in der Tasche nachsehen zu müssen.

Die in diesem Programm genutzten Methoden sind dazu gedacht, Ihnen zu zeigen, wie Sie eigene Anwendungsmöglichkeiten finden und Abläufe nutzen können, die in der Computerindustrie schon seit vielen Jahren verwendet werden.

Eine dieser Methoden ist die 'Menüwahl'. Der Computer bietet eine Liste aller in diesem Fall verfügbaren Operationen und fordert den Benutzer auf, seine Wahl zu treffen. Sobald das Programm die Option erhalten hat, entscheidet es, was zu tun ist, um diesen Befehl auszuführen, wobei berücksichtigt wird, daß der Benutzer die Optionen vielleicht in einer normalerweise nicht zu erwartenden Reihenfolge wählt.

Eine andere Technik ist die der Datenverwaltung. In diesem Programm können Sie die Einzelheiten für alle Ihre laufenden Aufträge, festen Kosten, Eingänge und Abhebungen speichern. Was geschieht dann, wenn die Rate für die Hypothek kleiner wird? Sie müssen offenkundig den entsprechenden Auftrag ändern, damit das System mit Ihrem Bankkonto in Übereinstimmung bleibt. Mit den Datenverwaltungs-Routinen in diesem Programm können entweder die Werte oder der beschreibende Text mit einem Mindestaufwand an Mühe und Umständen verändert werden. Sie geben lediglich an, welche Zeile auf dem Bildschirm wie verändert werden soll; das Programm löscht die betreffenden Posten und erwartet Ihre korrigierten Daten als Ersatz für die alten. All das wird auf dem Bildschirm vorgenommen, so daß Sie mühelos erkennen können, was Sie im Verhältnis zu allen verfügbaren Posten ändern.

Sie werden sich auch über die Probleme des Datenzugangs im klaren sein. Er muß stets gründlich geprüft werden, um zu gewährleisten, daß keine störenden Elemente eingegeben werden.

Beispielsweise kann eine Monatszahl immer nur im Bereich 1-12 liegen - wer hat schon von einem Jahr mit 13 Monaten gehört?

Der Ausdruck des Programms ist ein Abschlußblatt, das die verfügbaren Barbeträge auf Ihrem Konto anzeigt. Das kann eine völlig zutreffende Wiedergabe Ihres Bankkontos sein oder auch nicht, weil Sie die Kosten unregelmäßig erscheinender Posten über eine Reihe von Monaten strecken können. Gas, Strom, Telefon und Mieten sind für eine solche Einrichtung die Hauptkandidaten, weil diese Rechnungen oft vierteljährlich oder auch jährlich erscheinen. Mit dem Programm können eingeplante Posten in regelmäßige Beträge aufgeteilt werden, die nach Ihrer Schätzung auf den gesamten Rechnungsbetrag hinauslaufen, sobald dieser anfällt.

Dieses Abschlußblatt muß alle Beträge in sauberen Kolonnen, ausgerichtet nach dem Dezimalpunkt, anzeigen. Die Subroutine, mit der das erreicht werden kann, wurde im vorigen Kapitel gezeigt. Hier können Sie sehen, wie sie das Aussehen eines Berichts verbessert.

Sehen wir uns an, wie Sie dieses Programm nutzen können.

Geben Sie das Programm ein und sichern Sie es:

SAVE "BUDGET" **LINE** 1

Zunächst werden Sie keine Einzelheiten auf Band haben, die vorgelegt werden können, so daß alles von Anfang an eingegeben werden muß. Diese Einzelheiten sind:

- Einzelheiten über Einkommensquellen
- Einzelheiten über Daueraufträge
- Berücksichtigung aller eingeplanten (oder über einen längeren Zeitraum verteilten) Rechnungen
- Ihr derzeitiger Kontostand

Im Idealfall sollten Sie am Ende eines Monats anfangen, wenn Ihr Konto den Tiefststand erreicht hat.

Wenn das Programm beginnt, verlangt es die laufende Monatszahl:

Eingeben laufende Monatszahl:

Damit werden die Namen der Bänder bestimmt, die während des Laufs verwendet werden sollten, und außerdem das Datum, das bei allen Berichten erscheinen soll.

Sie sollten diese Zahl (1 für Januar, 2 für Februar, usw.) eingeben. Das Programm prüft sie und liefert dann auf dem Bildschirm das Hauptmenü:

Kontoführung

Optionen:
1) Band vom Vormonat laden
2) Daueraufträge eingeben/ändern
3) eingeplante Posten eingeben/ändern
4) Einkommensdetails eingeben/ändern
5) Abhebungen eingeben
6) Abschluß anzeigen
7) Einzelheiten dieses Monats sicherstellen
8) Saldo
9) Ende

Ihre Wahl eingeben:

In diesem Stadium müssen Sie die Optionen 2, 4, 8 und möglicherweise 3 nehmen.

Die Optionen 2, 3 und 4 wirken auf ähnliche Weise. Wenn Posten vorhanden sind, werden diese auf dem Bildschirm für Sie angezeigt. Jede Zeile wird ungefähr so aussehen:

Zeilenbeschreibung	Wert
1 Hypothek	1580.00
2 Auto Rate	600.00
3 Fernseher Rate	80.00

Wollen Sie etwas verändern?

Sie sind eingeladen, hier beliebig zu verändern. Wenn Sie 'Ja' eingeben, fordert das Programm Sie auf, die richtige Zeilennummer einzugeben. Das ist die Zahl in der linken Spalte.

Wenn Sie die Zeilennummer eingeben, wird der entsprechende Eintrag auf dem Schirm gelöscht, und Sie erhalten die Aufforderung, den beschreibenden Text (höchstens 15 Zeichen) und den Wert einer Zahlung einzugeben. Die neuen Eingaben werden anstelle der alten auf dem Bildschirm eingefügt. Sie können auf dem Bildschirm so viele Zeilen verändern, wie Sie wollen, und das geht so weiter, bis Sie keine Zeilennummer mehr eingeben, sondern lediglich ENTER drücken.

Als nächstes wird Ihnen angeboten, neue Posten einzugeben. Ein 'Ja' veranlaßt das Programm, den beschreibenden Text und den Wert von Posten anzugeben. Das setzt sich fort, bis Sie ENTER drücken, sobald Sie zu einer neuen Textbeschreibung aufgefordert werden.

Keine Sorge, wenn Sie beim Schreiben Fehler machen – Sie können jederzeit dieselbe Option auf dem Bildschirmmenü erneut wählen und die Änderungs-Einrichtung nutzen, um alles abzuändern, was Ihnen nicht behagt.

Bei den eingeplanten Posten geht es etwas anders. Erstens wirken sie sich auf den konkreten Stand Ihres Bankkontos nicht aus, weil sie nur Vorkehrungen gegenüber künftigen Rechnungen sind. Innerhalb des Programms Ihrer Bilanz wird eine zweite Summierung vorgenommen, soweit sie die eingeplanten Beträge betrifft. Wenn Sie sie verwenden wollen, sollten Sie daran denken, daß die vorgetragene Gesamtsumme der eingeplanten Posten dem jeweils vorhandenen Betrag an frei verfügbarem Bargeld entspricht. Der Unterschied zwischen diesem Wert und dem tatsächlichen Kontostand ist der Gesamtbetrag erforderlichen Geldes – lassen Sie sich nicht verlocken, ihn auszugeben!

Gelegentlich wird Ihr tatsächlicher Abschluß unter dem eingeplanten liegen, weil bei Eingang jeder Rechnung der Betrag als Abhebung eingetragen wird und den tatsächlichen Abschlußbetrag verringert. Im nächsten Monat erfolgt aber die nächste Zahlung des eingeplanten Betrages und bringt mit der Zeit alles wieder ins Lot.

Wenn der Unterschied zwischen beiden Abschlüssen zu groß wird, können Sie das durch Verwendung von Option 8 auf dem Schirmmenü ändern. Sie können entweder den Abschluß selbst ändern (wie Sie es anfangs tun müssen, damit das System den derzeitigen Stand kennt) oder den Stand bei den eingeplanten Beträgen. Setzen Sie zunächst beide Beträge auf denselben Wert.

Abhebungen (Option 5) werden als Summenzahl eingesetzt, weil dieses Programm kein Interesse daran hat, einzelne Schecks unter die Lupe zu nehmen. Alle Zwischenzahlungen **auf** Ihr Konto sollten in diesem Stadium als Reaktion auf die Frage 'Zusätzliche Eingänge' ebenfalls eingegeben werden. Eine Null-Eingabe auf eine dieser Fragen wird als Nullwert behandelt.

Zuletzt können Sie das Abschlußblatt fahren. Das kann als Option zum Drucker geschickt werden ("Hardcopy?"). Alle Posten werden aufgeführt und zeigen die Postenart als Abkürzung von drei Zeichen an, EIN für Einkommen, D/A für Dauerauftrag und ZUK für eingeplante Beträge.

Bei Einkommensposten werden die Werte versetzt nach links angegeben, so daß Einkommen von ausgehendem Bargeld leicht unterschieden werden kann.

In diesem Stadium werden keine Gesamtbeträge fortgeschrieben, so daß etwaige Fehler in den Beschreibungen oder Werte auszubessern sind und ein neues Abschlußblatt erstellt werden kann.

Schließlich können Sie die Einzelheiten zur Vorbereitung auf den folgenden Monat auf einer Kassette sicherstellen. Wählen Sie Option 7 und legen Sie eine leere Kassette ein. Gespeichert werden zwei Arrays, V$ () und V (), so daß Sie zweimal die Meldung "Start tape, then press any key" sehen. Sie können das Band dazwischen weiterlaufen lassen.

Im folgenden Monat ist als erste Option 1 zu wählen, die Sie auffordert, das so gesicherte Band zu laden und Beschreibung, Werte und Abschlüsse einzugeben.

Bei Nutzung von Option 7 werden die Abschlußbeträge fortgeschrieben, die Bereiche Gesamtabhebungen und Zusatzeingänge auf Null zurückgestellt.

Bevor Sie das Programm verwenden, sollten Sie den Wert von Zeile 10 überprüfen. Sie enthält die Höchstzahl an Posten, die nach Ihrer Schätzung auf dem Abschlußblatt erscheinen werden. Beachten Sie, daß sie nicht erfordert, Details einzelner Schecks festzuhalten. Der Wert 64 dürfte normalerweise mehr als ausreichend sein, aber beim 16K-Spectrum können Sie ihn ohne Probleme auf rund 120 erhöhen.

```
    1 REM Kontofuehrung
    2 REM © 1982 Phipps Associate
s
   10 LET maxitems=64                    nach Wunsch ändern
   20 GO SUB 7000: REM Initialisi
eren
  100 LET a$="Laufende Monatszahl
  eingeben:": GO SUB 9000: IF end
  THEN GO TO 100
```

```
 110 GO SUB 9400: IF x THEN LET
Monat=VAL a$( TO x)
 120 IF Monat<1 OR Monat>12 THEN
 GO TO 100
 130 RESTORE 9910: FOR x=1 TO Mo
nat: READ n$: NEXT x
1000 REM Hauptmenue
1010 CLS
1020 PRINT TAB 9; INVERSE 1;"Kon
tofuehrung"; INVERSE 0
1030 PRINT '"Optionen:"''
1040 PRINT "(1) Band vom Vormona
t laden"
1050 PRINT "(2) Dauerauftraege e
ingeben/          aendern"
1060 PRINT "(3) eingeplante Post
en eingeben/      aendern"
1070 PRINT "(4) Einkommensdetail
s eingeben/       aendern"
1080 PRINT "(5) Abhebungen einge
ben"
1090 PRINT "(6) Abschluss anzeig
en"
1100 PRINT "(7) Einzelheiten die
ses Monats        sicherstellen"
1110 PRINT "(8) Saldo"
1120 PRINT "(9) Ende"
1200 LET a$="Gib deine Wahl ein:
": GO SUB 9000: IF end THEN GO T
O 1200
1210 GO SUB 9400: IF NOT x THEN
GO TO 1200
1220 LET Wahl=VAL a$( TO x)
1230 IF Wahl<1 OR Wahl>9 THEN GO
 TO 1200
1300 GO SUB 2000+(Wahl-1)*100
1310 GO TO 1000
2000 REM letzten Monat eingeben
2010 LET x=Monat-1: IF x=0 THEN
LET x=12
2020 LET f$="MONAT"+STR$ x
2030 LET a$="Band laden "+f$+CHR
$ 13+"Druecke ENTER, wenn fertig
."
2040 GO SUB 9000
2050 LOAD f$ DATA v$()
2060 LOAD f$ DATA v()
2070 RETURN
```

Monat Bestätigen

ablehnen, wenn ungültig

Monatsstring holen

Wahl bestätigen

entsprechende Routine

Bandname

Beschreibungen
Werte

```
2100 REM Dauerauftraege
2110 LET t=2: GO SUB 8000
2120 RETURN
2200 REM eingeplante Ausgaben
2210 LET t=3: GO SUB 8000
2220 RETURN
2300 REM Einkommen
2310 LET t=1: GO SUB 8000
2320 RETURN
2400 REM Abhebungen
2405 DIM w(2)                          Eink./Abhebungen
2410 LET a$="Abhebungen gesamt e
ingeben:": GO SUB 9000: IF end T
HEN RETURN
2420 LET w(1)=VAL a$
2430 LET a$="Zusatzeingaenge ein
geben:": GO SUB 9000: IF end THE
N RETURN
2440 LET w(2)=VAL a$
2450 RETURN
2500 REM Abschluss anzeigen
2510 LET a$="Hard copy?": GO SUB  s.'Winke u. Tips'
 9000: LET dev=2+(CHR$ CODE a$="
J")
2520 CLS
2530 PRINT #dev; INVERSE 1;"Ausg
leich"; INVERSE 0;" ";n$
2540 PRINT #dev: PRINT #dev; INV
ERSE 1;"Beschreibung eintippen";
TAB 27;"Wert"; INVERSE 0
2550 PRINT #dev
2560 GO SUB 6000: REM Meldung
2570 RETURN
2600 REM Diesen Monat sichern
2610 LET v(v(1)+2)=t(1): LET v(v  Summen aktualisieren
(1)+3)=t(2)
2620 SAVE "MONAT"+STR$ Monat DAT  Beschreibungen
A v$()
2630 SAVE "MONAT"+STR$ Monat DAT  Werte
A v()
2640 PRINT AT 21,0;"Band bezeich
nen als ""MONAT";Monat;""""
2650 DIM w(2): REM Monatsinforma
tion loeschen
2660 GO SUB 8900
2670 RETURN
2700 REM Ausgleich S/H anpassen
2710 CLS
```

```
2720 FOR t=1 TO 2
2730 PRINT "Laufend S/H ";
2740 PRINT "Tatsaechlich" AND t=
1;"Eingeplante Posten" AND t=2;
2750 LET c=31: LET v=v(v(1)+t+1)
: GO SUB 9500
2760 LET a$="Ist das richtig?":
GO SUB 9000: IF CHR$ CODE a$="J"
 THEN GO TO 2790
2770 LET a$="Neuen Kontostand ei
ngeben:": GO SUB 9000: IF end TH
EN GO TO 2790
2780 LET v(v(1)+t+1)=VAL a$: GO
TO 2750
2790 NEXT t
2795 RETURN
2800 REM Ende
2810 CLS : GO TO 9999
6000 REM Meldung
6010 LET t(1)=v(v(1)+2): LET t(2 Abschlüsse
)=v(v(1)+3)
6020 LET u$="99999.99": LET c=28 Maske anlegen
6030 LET x$="S/H": GO SUB 6500   Abschlüsse anzeigen
6100 FOR t=1 TO 3                Postenkategorien
6110 FOR n=2 TO v(1)+1
6120 IF t<>CODE v$(n) THEN GO TO richt. Typ finden
 6200
6130 PRINT #dev;s$(t);TAB 5;v$(n anzeigen
,2 TO );
6140 LET c=31: IF t=1 THEN LET c Soll/Haben
=28
6150 LET v=v(n): GO SUB 9500     Wert anzeigen
6155 PRINT #dev
6160 IF t<>1 THEN LET v=v*-1     Vorzeichen d. Typs
6170 LET t(2)=t(2)+v: IF t<>3 TH Abschluß
EN LET t(1)=t(1)+v
6200 NEXT n
6210 NEXT t
6220 FOR t=1 TO 2                neue Summen
6225 PRINT #dev;TAB 5;t$(t+3);
6230 LET c=28+3*(t=1)
6235 LET v=w(t): GO SUB 9500
6240 IF t=1 THEN LET v=v*-1
6250 LET t(1)=t(1)+v: LET t(2)=t
(2)+v
6260 NEXT t
6300 PRINT #dev
6310 LET x$="C/F": GO SUB 6500
```

```
6350 IF dev=2 THEN GO SUB 8900
6360 LET dev=2
6370 RETURN
6500 REM Gesamt
6510 FOR x=1 TO 2
6520 PRINT #dev;TAB 5;x$;" ";"Ta
tsaechlich" AND x=1;"eingeplante
Posten" AND x=2;
6530 LET c=28: LET v=t(x): GO SU
B 9500
6540 PRINT #dev
6550 NEXT x
6560 RETURN
7000 REM Initialisieren
7010 LET Monat=0
7020 DIM v$(maxitems+1,16): DIM
v(maxitems+3)
7030 RESTORE 9900: READ n
7040 DIM t$(n,15): DIM s$(n,3):
DIM r(22): DIM t(2)
7050 FOR x=1 TO n: READ t$(x):
EXT x
7055 FOR x=1 TO n: READ s$(x):
EXT x
7060 LET dev=2: REM Vorgabeschir
m
7070 DIM w(2)
7090 RETURN
8000 REM Einzelheiten festhalten
8010 LET i=0: LET r=0
8020 LET m$="ERGAENZEN": GO SUB
8800
8040 FOR n=2 TO v(1)+1
8050 IF CODE v$(n)<>t THEN GO TO
 8200
8060 LET i=i+1
8070 IF r+h=20 THEN GO SUB 8500:
 GO SUB 8800
8080 PRINT AT r+h,0;r+1;TAB 5;v$
(n,2 TO );
8090 LET c=31: LET u$="99999.99"
: LET v=v(n): GO SUB 9500
8100 LET r=r+1: LET r(r)=n
8200 NEXT n
8210 IF r THEN GO SUB 8500: REM
Ergaenzen
8220 LET m$="EINFUEGEN": GO SUB
8800
```

Pause, wenn auf Schirm

N Kategoriebeschreib.

N kurze Beschreibungen

Tabelle ablesen

richtiger Typ?

Postenzähler

Bildschirm voll

Posten anzeigen

nächste Reihe

neu einfügen

```
8230 LET a$="Neue einfuegen? ":
GO SUB 9000
8240 IF CHR$ CODE a$<>"J" THEN R
ETURN
8250 IF v(1)=maxitems+1 THEN LET
 a$="Tabelle voll. ENTER druecke
n": GO SUB 9000: RETURN
8260 LET Posten=v(1)+2: GO SUB 8
600: IF end THEN RETURN
8270 LET v(1)=v(1)+1
8280 GO TO 8250
8500 REM Bilschirm
8510 LET a$="Willst du veraender
n?": GO SUB 9000
8520 IF CHR$ CODE a$<>"J" THEN R
ETURN
8530 LET a$="Zeilennummer eingeb
en:": GO SUB 9000: IF end THEN R
ETURN
8540 GO SUB 9400: IF NOT x THEN
GO TO 8530
8550 LET y=VAL a$( TO x): IF y<1
 OR y>r THEN GO TO 8530
8560 LET Posten=r(y): PRINT AT h
+y-1,0;y,,AT h+y-1,5;
8570 GO SUB 8600: IF end THEN GO
 TO 8570
8580 GO TO 8530
8600 REM Posten aendern
8610 INPUT "Text eingeben:"; LIN
E a$: LET end=NOT LEN a$: IF end
 THEN RETURN
8620 LET v$(Posten)=CHR$ t+a$: P
RINT TAB 5;a$;
8630 LET a$="Wert eingeben:": GO
 SUB 9000
8640 LET v(Posten)=VAL a$
8650 LET c=31: LET u$="99999.99"
: LET v=v(Posten): GO SUB 9500
8660 RETURN
8800 REM Ueberschriften
8810 CLS : LET r=0
8820 PRINT INVERSE 1;t$(t); INVE
RSE 0;" ";n$;TAB 26, INVERSE 1;m
$; INVERSE 0
8830 PRINT : PRINT INVERSE 1;"Ze
ilenbeschreibung";TAB 27;"Wert";
 INVERSE 0
```

```
8840 LET h=4: PRINT AT h,0;
8850 RETURN
8900 REM ENTER druecken
8910 LET a$="Druecke ENTER zur F
ortsetzung"
9000 REM Zeileneingabe                    übliche Subroutinen
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
" AND a$(x)<="z")): NEXT x
9030 RETURN
9400 REM let x=numptr(a$)
9410 FOR x=1 TO LEN a$: IF a$(x)
>="0" AND a$(x)<="9" THEN NEXT x
9420 LET x=x-1
9430 RETURN
9500 REM Verwendung anzeigen
9510 LET z$="": LET x1=0
9515 FOR z=1 TO LEN u$
9520 IF x1 THEN LET z$=z$+"0"
9525 IF u$(z)="." THEN LET x1=1
9530 NEXT z: LET x2=LEN z$
9535 LET x1=INT ABS v: LET xp=IN
T ((ABS v-x1)*10^x2)
9540 LET z$=("-" AND v<0)+STR$ x
1+("."+(z$( TO x2-LEN STR$ xp)+(
STR$ xp+z$))( TO x2) AND x2)
9550 IF LEN z$>LEN u$ THEN LET z
$=z$(LEN z$-LEN u$+1 TO )
9555 PRINT #dev;TAB (c-LEN z$+1)
;z$;
9560 RETURN
9900 DATA 5                               Zahl der Beschreib.
9901 DATA "Einkommen"
9902 DATA "Dauerauftraege"
9903 DATA "Planungsdetails"
9904 DATA "Abhebungen"
9905 DATA "Zusatzeingaenge"
9906 DATA "EIN","DA","EP","ABH",          kurze
"ZE"                                      Beschreibungen
9910 DATA "Januar","Februar","Ma
erz","April","Mai","Juni"
9911 DATA "Juli","August","Septe
mber","Oktober","November","Deze
mber"
9999 STOP
```

Roboterjagd

Sie sitzen in der Falle in einem Gefängnisgelände, umgeben von einem elektrisch geladenen Zaun. Den Zaun berühren bedeutet sofortigen Tod - vermeiden Sie das um jeden Preis. Die Roboterwächter haben Sie bemerkt und sind schon unterwegs zu Ihnen - ein Entkommen ist nicht möglich!

Um von ihnen nicht erwischt zu werden, müssen Sie versuchen, Sie an die Zäune zu führen und sie außer Gefecht zu setzen. Das hat nur einen kleinen Haken - sobald ein Wächter ausgeschaltet ist, fällt auch der Hochspannungsmast aus, so daß die restlichen Wächter darüber hinweggehen können.

Das Spiel ist recht amüsant und kann auch von den jüngeren Familienmitgliedern leicht bewältigt werden.

Die Benutzergrafik wird entsprechend den verschiedenen Objekten im Spiel aufgestellt - Sie selbst, die Wächter und die Zäune. Verschiedene Codes dienen ferner dazu, einen unbeweglichen Wächter und Sie als unbewegliche Figur darzustellen!

Geräusche werden das ganze Spiel hindurch erzeugt, um Spannung hervorzurufen; die Töne werden im Lauf der Zeit schriller. Sobald etwas an einen Zaun gerät, hört man ein Schnalzen, am Ende der Partie wird eine passende Melodie gespielt, um anzuzeigen, ob Sie gewonnen oder verloren haben.

Jedes Spiel dauert nur ein, zwei Minuten, ideal für die Unterhaltung der Familie.

Sobald Sie das Programm eingegeben haben, sichern Sie es mit:

SAVE "ROBOTCHASE" **LINE** 1

```
  1 REM  **** Robot Chase ****
  2 REM © Phipps Associates '82
  5 RESTORE : RANDOMIZE : CLEAR

 10 GO SUB 8200          Sondergrafik definieren
 20 CLS
 30 PRINT TAB 12;"SPIONAGE"''"D
u bist in einer feindlichen    V
erteidigungsstellung, und un=  g
luecklicherweise sind die Wa=  c
hen auf Dich aufmerksam gewor= d
en.Deine einzige Chance, ihnen z
```

```
u entkommen, ist, sie bei der  V
erfolgung in die Elektro-Felderz
u locken."
  40 PRINT '"Elektrofelder--";CH
R$ 144''"Waechter--------";CHR$ 1
46''"Du-------------";CHR$ 145
  50 PRINT ''"Mit den Kursortast
en gehst Du inPfeilrichtung, mit
 den Tasten 4,R,U und 9 diagonal
."
  55 INPUT "Schwierigkeitsgrad?
(1-3)? ";d: IF d<1 OR d>3 THEN G
O TO 55
  60 PRINT ,,TAB 11;"VIEL GLUECK
"
  70 GO SUB 8300: PAUSE 25          Einleitung spielen
  90 LET y$=CHR$ 145: LET g$=CHR
$ 146: LET f$=CHR$ 144
 100 CLS : PRINT AT 10,8;"Spielf
eld-Aufbau..."
 110 LET ng=6+d
 115 DIM s(21,31): DIM b(ng,2)
 120 FOR x=1 TO ng                  Wächter und Sie setz.
 130 LET b(x,1)=FN r(30)
 140 LET b(x,2)=FN r(20)
 150 IF FN s(x) THEN GO TO 130
 160 LET i=146: IF x=1 THEN LET
i=145
 170 LET s(b(x,2),b(x,1))=i
 180 NEXT x
 200 FOR n=1 TO 55-FN r(d*10)       Zäune setzen
 210 LET x=FN r(30)
 220 LET y=FN r(20)
 230 IF s(y,x) THEN GO TO 210
 240 LET s(y,x)=144
 245 NEXT n
 250 BORDER 5: PAPER 6: GO SUB 9    Bildschirm zeichnen
000
 260 FOR y=1 TO 20: FOR x=1 TO 3
1: IF s(y,x) THEN PRINT AT y,x;C
HR$ s(y,x)
 270 NEXT x: NEXT y
 280 LET m=0
 300 LET m$=INKEY$                  Bewegung holen
 310 IF m$="" THEN GO TO 400
 320 LET x=(m$="5")*-1+(m$="8")+
(m$="4")*-1+(m$="r")*-1+(m$="u")
+(m$="9")
```

```
 330 LET y=(m$="6")+(m$="7")*-1+
(m$="4")*-1+(m$="r")+(m$="9")*-1
+(m$="u")
 360 LET g=1: GO SUB 8100                    sich selbst bewegen
 365 IF x<1 OR y<1 OR x>30 OR y>             prüfen, ob noch
20 THEN GO TO 9200                           auf dem Schirm
 370 IF FN s(1)=g THEN GO TO 910             Wächter
0                                            angestoßen?
 380 IF FN s(1)=144 THEN GO TO 9             ... oder einen Zaun?
200
 390 LET s(y,x)=145: PRINT AT y,             sich selbst bewegen
x;y$: BEEP .1,-10
 400 LET n=0: LET m=m+1
 410 FOR g=2 TO ng                           jetzt Wächter bewegen
 430 IF FN s(g)<>146 THEN GO TO              noch am Leben?
600
 440 LET n=n+1                               Rest Wächter zählen
 450 LET x=SGN (b(1,1)-b(g,1))               Ihre Richtung
 460 LET y=SGN (b(1,2)-b(g,2))
 465 PRINT AT b(g,2),b(g,1);" "              ein Leerraum
 470 GO SUB 8100                             Wächter bewegen
 490 IF FN s(g)=145 THEN GO TO 9             hat er Sie erwischt?
100
 500 IF FN s(g)=144 THEN FOR a=1             ist er an den Zaun
 TO 10: FOR b=0 TO 1: BEEP .01,b             geraten?
*5-5: NEXT a: NEXT b: LET s(y,x)
=147: LET t=0: GO TO 530
 510 LET s(y,x)=146                          neue Position
 520 LET t=2
 530 PRINT AT y,x; INK t;CHR$ s(
y,x)
 540 BEEP .1,m
 600 NEXT g
 610 IF n=0 THEN GO TO 9300                  noch Wächter da?
 620 GO TO 300                               dann weiter
 999 STOP
8100 LET s(b(g,2),b(g,1))=0: PRI             ein Leerraum
NT AT b(g,2),b(g,1);" "
8105 LET b(g,1)=b(g,1)+x: LET b(
g,2)=b(g,2)+y
8110 LET x=b(g,1): LET y=b(g,2)
8120 RETURN
8200 DATA "a",255,129,129,129,12             Grafik
9,129,129,255
8205 DATA "b",8,28,8,62,8,8,20,2
0
8210 DATA "c",0,0,24,60,126,219,
255,36
```

```
8215 DATA "d",66,34,68,66,24,60,
126,255
8220 DATA "e",0,32,33,39,36,252,
252,0
8230 FOR c=1 TO 5
8240 READ c$: FOR x=0 TO 7: READ
 r: POKE USR c$+x,r: NEXT x
8250 NEXT c
8260 DEF FN s(x)=s(b(x,2),b(x,1)
)
8270 DEF FN s$(x)=CHR$ FN s(x)
8280 DEF FN r(x)=INT (RND*x)+1
8290 RETURN
8300 REM nice tune
8310 FOR x=1 TO 3: BEEP .3,7: BE
EP .15,4: NEXT x: BEEP .6,0
8320 RETURN
8400 REM funeral march
8410 LET s=.6
8420 BEEP s,0: BEEP s*2/3,0: BEE
P s/3,0: BEEP s,0: BEEP s*2/3,3:
 BEEP s/3,2: BEEP s*2/3,2: BEEP
s/3,0: BEEP s*2/3,0: BEEP s/3,-1
: BEEP s,0
8430 RETURN
9000 CLS                                  Umzäunung zeichnen
9010 PLOT 0,175: DRAW 255,0: DRA
W 0,-175: DRAW -255,0: DRAW 0,17
5
9020 RETURN
9100 PRINT AT b(1,2),b(1,1); INK          Endspiel-Meldungen
 2;CHR$ 148
9105 PRINT AT 21,0;"Du bist gefa
ngen!"
9110 FOR x=0 TO 20: FOR y=0 TO 1
: BEEP .01,13+y*5: NEXT y: NEXT
x
9115 IF INKEY$<>"" THEN GO TO 91
15
9120 PAUSE 50: GO SUB 8400
9130 GO TO 9400
9200 PRINT AT b(1,2),b(1,1); FLA
SH 1; INK 2; PAPER 6;CHR$ 148
9205 PRINT AT 21,0; FLASH 1;"**
ZAP **"; FLASH 0;" Du bist im Fe
ld!"
9210 FOR x=39 TO 0 STEP -1: BEEP
 .005,x: NEXT x
```

```
9220 GO TO 9400
9300 PRINT AT 21,0; FLASH 1;"DU
ENTKOMMST"; FLASH 0;" SUPER-AGEN
T!"
9310 GO SUB 8300
9400 IF INKEY$<>"" THEN GO TO 94
00
9405 INPUT "Noch ein Spiel? ";z$
9410 IF LEN z$ THEN IF z$(1)="j"
 THEN GO TO 100
9999 BORDER 7: INK 0: PAPER 7: S
TOP
```

Abschnitt Maschinencode

Einleitung

Dieser letzte Teil des Buches (ein ziemlich großer Teil) befaßt sich dem gefürchteten 'Maschinencode'. Es gibt viele Gründe dafür, weshalb ich einen solchen Abschnitt aufgenommen habe. Sinclair hat nun drei verschiedene Computer auf den Markt gebracht - den ZX80, den ZX81 und als neuesten den Spectrum. Das ist alles im Verlauf von gut zwei Jahren geschehen und hat auf den Markt eine starke Auswirkung gehabt.

Ich habe den Fortschritt von Homecomputer-Besitzern während dieses Zeitraums verfolgt und war überaus erstaunt angesichts der Nachfrage nach immer mehr Wissen. Viele Besitzer kaufen die Geräte zwar nur dazu, um eine Vielzahl von Videospielen betreiben zu können, aber viele andere wollen aus ihrer Neuerwerbung das meiste herausholen. Die BASIC-Sprache findet 'auf Anhieb' Anklang, obwohl sie keineswegs übermäßig einfach ist. Trotzdem hat sie ihre Grenzen, was in der Hauptsache an zwei Gründen liegt. Erstens ist sie langsam. Das gilt für fast alle Mikrocomputer, obwohl der Spectrum tatsächlich noch langsamer ist als die meisten. Die Folge davon ist, daß viele gute Programme behindert werden durch eine langsame Reaktion, was den Spieler enttäuscht, der sie dann vermutlich als 'untauglich' (oder mit einem ähnlichen Ausdruck) abtut.

Zweitens kann BASIC nicht alle Möglichkeiten des Mikrocomputers ausschöpfen, weil BASIC entgegen manch anderer Meinung eben doch eine gewisse 'Normierung' aufweist.

Das heißt: Wenn Sie bei der Beschäftigung mit dem Computer über BASIC hinausgelangen wollen, müssen Sie sich einer anderen Sprache zuwenden. Mit der Zeit werden neue Sprachen zur Verfügung stehen - FORTH, Pascal, LISP, vielleicht sogar (Gott behüte) Cobol. Inzwischen ist der Maschinencode verfügbar. Er benützt nur einige Instrumente, an die man leicht herankommt.

Damit wollen wir uns hier abgeben. Sie finden in diesem Abschnitt an Werkzeug alles, was Sie brauchen - einen **vollstän-**

digen Assemblierer (ohne Spaß!), mit dem Sie übliche Maschinenbefehle in Assemblersprache (also in mnemotechnischer Form) eingeben können; einen Disassemblierer und ein Maschinencode-Überwachungsprogramm. Damit können Sie Ihre Programme testen, sobald sie geschrieben und in Assemblersprache übersetzt sind. (Falls Ihnen irgendeiner dieser Ausdrücke neu sein sollte – sie werden an den entsprechenden Stellen erklärt.)

Zusätzlich wird Ihnen gezeigt, wie Sie einige der Einrichtungen in Ihrem Spectrum-ROM nutzen können, in dem 'unveränderbaren' Programm, das sich im Inneren befindet. Sie können so BASIC-Programme eingeben und fahren.

Wem diese Begriffe neu sind, der wird sich zuerst in eine fremde Welt versetzt finden, aber ich lege großen Wert darauf, daß dieser Band nicht bloß für den Augenblick Spaß macht, sondern Sie ihn auch als Nachschlagewerk benützen können. So wie Sie aus vielen verschiedenen Quellen lernen, hoffe ich, daß Sie auch von einer anderen Warte aus zu diesen Seiten zurückkehren.

Wenn Sie mit dem Spectrum wirklich erst anfangen, dann klappen Sie das Buch jetzt nicht einfach zu. Eine große Anzahl der kleinen Programme, die nun folgen, wird für Sie nämlich sogar dann lehrreich und nützlich sein, wenn Sie nicht verstehen, wie sie funktionieren. Wenn Sie den Assemblierer fahren (siehe ZXASM weiter unten) und die Maschinencode-Routinen eingeben, die unten aufgeführt sind, können Sie sie auf Band sicherstellen, um sie in Ihre eigenen Programme mit aufzunehmen. Das heißt zwar, daß Sie einige Programme zunächst blind eintippen, aber vielleicht gewinnt Ihre Neugier doch die Oberhand ...

Fangen wir mit einem Blick auf den Spectrum-ROM an.

Spectrum-ROM

Der ROM (read only memory = Nur-Lese- oder Festspeicher) besetzt ein Viertel des Gesamtspeichers, der dem Spectrum direkt zugänglich ist – mit anderen Worten: Dem Systemprogramm des Spectrum sind 16K-Speicherplatz gewidmet. Das ist soviel, wie Sie beim 16K-Grundmodell überhaupt zur Verfügung haben! Wozu wird das alles verwendet?

Nun, erstens wirkt der Festspeicher als **Interpreter.** Er übersetzt Ihre BASIC-Programme in eine Form, die der Spectrum

(genauer, der Z80-Mikroprozessor) wirklich verstehen kann, um sie dann auszuführen. Auf diese Art können Sie Programme rasch schreiben und Programme anderer ohne Mühe verstehen, weil BASIC als eine Form von 'Englisch' leicht erkennbar ist.

Zweitens steuert er alle Übertragungen zwischen Z80-Prozessor und Tastatur, Fernsehschirm, Kassette und anderem Zusatzgerät. Diese Tätigkeiten nehmen eine beträchtlichen Teil des ROM in Anspruch.

Im allgemeinen neigen Programmierer in Maschinencode dazu, sich mehr für die zweite Art zu interessieren. Wenn Sie BASIC übersetzen wollten, könnten Sie ja ebensogut das Programm in BASIC schreiben und den ROM die Arbeit machen lassen! Offenkundig müssen manche MC-Programme (Abkürzung für Maschinencode) die Art und Weise nutzen, wie der ROM BASIC-Befehle behandelt, da ein Programm teilweise in BASIC und teilweise in Maschinencode geschrieben sein kann.

Was sind also die Hauptprobleme, vor denen man steht, wenn man ein Programm in Maschinencode schreiben möchte? Es sind folgende:

a) Die Befehle in Maschinencode verstehen
b) Wissen, wie man das Programm eingibt, speichert und fährt
c) Wie man die Einrichtungen des Spectrum (etwa die Tastatur, das Display etc.) benützt, ohne immer wieder zu BASIC zurückzukehren
d) Wie man Information von BASIC in Maschinencode und umgekehrt überträgt
e) Wie man das Programm testet, sobald es geschrieben und in Assemblersprache übertragen ist.

Die restlichen Abschnitte des Buches befassen sich mit den meisten dieser Probleme, aber das erste bedarf zunächst einer kleinen Erläuterung.

Die Befehle verstehen

Das ist, schlicht gesagt, eine Aufgabe, die dieses Buch überfordert. Man könnte einen Band von doppeltem Umfang darüber schreiben und würde vermutlich immer noch einige der größe-

ren Feinheiten auslassen, weil Programmieren in Assemblersprache ebensosehr eine Technik ist wie das Erlernen der Codes. Es gibt auf dem Markt einige Bücher zu dem Thema, aber damit geht es recht mühsam, weil man methodisch vorgehen muß, wenn man genau verstehen will, was sich abspielt. Mit ein bißchen Geduld findet sich in solchen Bänden aber alles, was man wissen muß. Sobald Sie sich auskennen, werden Sie erkennen, um wieviel leichter ein Assemblierer Ihnen das Leben macht.

Maschinencode von BASIC aus fahren

Die **USR**-Funktion ist der Ausgangspunkt für Ihre Routinen. Dadurch wird der Spectrum gezwungen, die Maschinencode-Routine zu fahren, deren Adresse folgt. Beispiel: Wenn Ihre Routine (oder irgendeine der folgenden) auf Adresse 32600 gesetzt werden würde, hätte Ihr Programm eine Zeile zu enthalten wie:

```
1100 LET x = USR 32600
```

Es gibt zwei weitere Überlegungen. Wenn Sie über Ihre Routine Information vermitteln wollen, läßt sie sich mit POKE in einen freien Bereich des Speichers setzen, damit die Routine sie wieder holen kann. Verfügbare Speicherbereiche sind:

Systemvariable-Byte	23681
Systemvariable-Bytes	23728 und 23729
Drucker-Pufferspeicher	256 Bytes ab 23296

Ihre Routine kann Informationen in diese Adressen eingeben (und mit der PEEK-Funktion herausholen) oder ein Einzelresultat in das Registerpaar BC setzen, bevor sie zu BASIC zurückkehrt. In diesem Fall würde die Variable "x" im obigen Beispiel zum Schluß der Routine auf diesen Wert gesetzt werden.

Die folgenden Abschnitte liefern viele Beispiele für funktionierende Routinen, die Sie eingeben und fahren können. Der später erscheinende Assemblierer ist der Schlüssel für die meisten dieser Programme, so daß ich Ihnen raten würde, zunächst ZXASM, den Spectrum-Assemblierer, zu studieren und einzugeben.

Spectrum-ROM nutzen

Da sitzen Sie also und codieren verzweifelt Ihr neues Programm 'Angriff aus dem All'. Plötzlich wird Ihnen klar, daß das Programm ein Tastaturzeichen annehmen muß, damit der Spieler sein Raumschiff bewegen kann. Wie wird das geleistet? Wie zeigen Sie Posten auf dem Schirm an? Oder rollen alles um ein Pixel nach links? (Versuchen Sie das einmal in BASIC!)

In allen folgenden Demonstrationsprogrammen habe ich Ihnen ein Listing des Programms vorgelegt, das direkt in den Assemblierer eingegeben werden kann. Vielleicht wäre das jetzt der richtige Augenblick, um sich einmal anzusehen, was er für Sie leisten kann, und ihn vielleicht ganz einzutippen!

Die Tastatur

Der Spectrum hat eine 'Uhr', wodurch fünfzigmal in der Sekunde die Systemvariable FRAMES inkrementiert (erhöht) wird. Das Wort 'Uhr' steht in Anführungszeichen, weil man sie ganz leicht aus dem Takt bringen kann.

Ein Teil der Spectrum-Schaltung erzwingt jede Fünfzigstelsekunde einen Aufruf an Adresse 56 im ROM. Dieser Bereich des ROM (mit dem Disassemblierer können Sie ihn sich ansehen) aktualisiert die Systemvariable FRAMES und tastet außerdem das Keyboard ab.

Das ist eine gute Nachricht für Leute, die an den ZX81 gewöhnt waren, wo das Abtasten auf mühsame Weise – nämlich durch Codieren – erfolgen mußte. Beim Spectrum brauchen Sie in einer Maschinencode-Routine lediglich eine Flagge zu testen, um festzustellen, ob eine Taste gedrückt worden ist. Falls eine Flagge gesetzt ist, können Sie den Code der Taste der Systemvariablen LASTK entnehmen (Adresse 23560). Die Flagge befindet sich in Bit 5 der Adresse 23611 (FLAGS).

Kapitel 26 des Sinclair-Handbuchs erwähnt, daß ein Maschinencode-Programm nicht das Register IY verwenden sollte. Der Grund dafür: Es wird im ganzen ROM dazu benützt, Information aus den Systemvariablen zu erhalten. Außerdem enthält es die Adresse der Systemvariablen ERRNR.

Falls Sie also sehen wollen, ob eine Taste gedrückt worden ist, kann Ihre Routine die folgenden Zeilen enthalten:

```
LASTK:  EQU 23560
        XOR A              ; A löschen
        BIT  5, (IY + 1)   ; Bit 5 Variable FLAGS prüfen
        JR   Z, NOKEY      ; Sprung, wenn keine Taste
                             drückt
        LD   A, (LASTK)    ; Code von Taste besorgen
        RES  5, (IY + 1)   ; Flagge neu setzen
NOKEY:
        ...                ; A enthält Null oder Tasten-
                             code
```

(Beachten: Wenn Sie den Assembler verwenden, können Sie die EQU-Anweisung dazu verwenden, sich auf alle ROM-Adressen zu beziehen. Mit diesem Befehl können Sie im ganzen Maschinencode-Programm Namen anstelle von Adressen verwenden und diese für sich vom Assemblierer einsetzen lassen. Genaueres bei ZXASM.)

Der von LASTK gelieferte Code berücksichtigt keine Befehls-Schlüsselwörter oder Funktionen in erweitertem Modus, wie das folgende Testprogramm beweist:

```
10 REM Test Keyboardabtastung
20 PRINT AT 10, 10; CHR$ PEEK 23560; ""
                                (zwei Leerräume)
30 GO TO 20
```

Drücken Sie beliebig Tasten, auch mit Shift. Aus diesem Programm ersehen Sie, daß LASTK sich nur dann verändert, wenn eine neue Taste gedrückt wird. Aus diesem Grund sollten Sie Bit 5 von FLAGS prüfen, bevor Sie den Inhalt laden.

Wenn es Sie überhaupt interessiert, ob eine Taste gedrückt **wird** oder nicht, ohne Rücksicht darauf, welche, können Sie den Wert der Adresse 23556 (KSTATE) testen. Sie enthält den Wert 255, wenn keine Taste gedrückt wird, und den 'Caps lock'-Wert der Taste, wenn sie gedrückt wird:

```
10 PRINT AT 1, 4; "Irgendeine Taste drücken"
20 PRINT AT 10, 10; CHR$ PEEK 23556: ""
                                (zwei Leerräume)
30 GO TO 20
```

Beachten Sie, daß das Schlüsselwort "COPY" den Codewert 255 hat.

Ihr Programm wird vielleicht auch auf BREAK prüfen wollen, weil diese Einrichtung innerhalb von Maschinencode nicht funktioniert, wenn sie vom Programm nicht ausdrücklich getestet wird.

Der Festspeicher enthält eine solche Routine in Adresse 1F54 (Hex). Wenn Sie diese Routine beim Rücksprung aufrufen, wird die Übertragungsflagge gesetzt, wenn BREAK **nicht** gedrückt, und gelöscht, wenn BREAK gedrückt wird. Beispiel:

```
CALL 1F54H      ; prüfen, ob BREAK gedrückt
RET  NC         ; Rücksprung, wenn BREAK
...             ; fortsetzen ...
```

Die obige Routine verlangt, daß gleichzeitig CAPS SHIFT und SPACE gedrückt werden. Manchmal kommt es Ihnen vielleicht nur darauf an, SPACE allein zu testen. Das geht mit:

```
LD   A, 7FH       ; Keyboardreihen-Adresse
IN   A, (ØFEH)    ; Reihenzustand feststellen
RRA               ; SPACE in Übertrag setzen
JP   NC, SPCKEY   ; Sprung, wenn SPACE gedrückt
...
```

Das Display

Das TV-Display ist mit Maschinencode schwerer zu bändigen, zum größten Teil wegen der Art, wie die Displaydatei im Speicher angelegt ist. In die Displaydatei ist mit PEEK und POKE nicht leicht hineinzukommen, aber wenn man ein bißchen genauer weiß, wie sie funktioniert, läßt sich das bewältigen.

Der Speicherbereich beginnt bei 16384 (4000 Hex) und besetzt 24x32x8 Bytes (= 6144 Bytes). Jede der 192 Pixelreihen am Bildschirm herunter erfordert 32 Bytes, so daß zumindest der Theorie nach die zweite Pixelreihe eines Zeichens 32 Pixel nach dem ersten erscheinen sollte. Falsch. ('Pixel' - siehe Register im Spectrum-Handbuch.)

Ein Zeichen besetzt ein Gitter von 8 mal 8 Pixel oder acht Pixelreihen. Jede der Pixelreihen wird **256** Bytes nach der vorherigen aufbewahrt. Wenn Sie also eine vertikale Linie von acht Punkten (Pixel) in der oberen linken Bildschirmecke zeichnen wollen, könnten Sie schreiben:

```
10 FOR x = 0 TO 7
20 POKE 16384 + x* 256,1
30 NEXT x
```

Der Bildschirm ist nun aufgeteilt in drei Abschnitte zu je acht Zeilen (drei Abschnitte von 64 Pixelreihen), und jeder dieser Abschnitte beginnt bei den Adressen 16384, 18432, 20480 - So beginnt Zeile 8 auf dem Bildschirm (beginnend bei Zeile 0) bei Adresse 18432, Zeile 16 bei 20480.

Innerhalb dieser Blöcke beginnt jeder Zeilenanfang 32 Bytes weiter unten. Hier einige Beispiele:

Zeile 2 - 16384 + 2*32 (Abschnitt 0 Anf. plus 2 Zeilen)
Zeile 17 - 20480 + 1*32 (Abschnitt 2 Anf. plus 1 Zeile)
Zeile 15 - 18432 + 7*32 (Abschnitt 1 Anf. plus 7 Zeilen)

Das läßt sich mit Maschinencode leicht berechnen, weil die Adresse jedes Bildschirmpunktes als eine 16 Bit-Darstellung berechnet werden kann durch:

010sspppprrrcccc

wobei ss die Abschnittszahl ist (0, 1 oder 2)
ppp die Pixelreihe innerhalb des Zeichens (0-7)
rrr die Zeilennummer im Abschnitt (0-7)
cccc die Spaltennummer (0-31)

Alle diese Werte sind natürlich binär. Das '010' am Anfang liefert den Wert 16384 als Beginn der Bildschirmkarte.

Die folgende Routine berechnet die Adresse jeder Pixelreihe aller Reihen und Spalten auf dem Bildschirm. Wenn Sie die Routine in Ihren eigenen Maschinencode-Programmen verwenden, setzen Sie Register A auf die Pixelreihennummer innerhalb des Zeichenblocks (0-7), Register B auf die Zeile am Schirm (0-23) und Register C auf die Spaltennummer (0-31). Beim Rücksprung enthalten die Register HL die Speicheradresse der Bildschirmkarte entsprechend dieser Position.

```
PCALC:
        PUSH  AF     ; Pixelreihe sichern
        LD    A, B   ; B enthält Zeile auf Schirm
        AND   18 H   ; entscheiden, in welchem Abschnitt
        OR    40 H   ; Bildschirmkartenadresse einfügen
        LD    H, A   ; in Register H sichern
        POP   AF     ; Pixelreihennummer wiederherstell.
```

```
ADD   A, H   ; Schirmversetzung hinzufügen
LD    H, A   ; in H ersetzen
LD    A, B   ; Zeile in Abschnitt bestimmen
AND   7      ; Abschnittnummer abdecken
RRCA         ; Zeile in oberste drei Bits setzen
RRCA
RRCA
ADD   A, C   ; Spaltennummer hinzufügen
LD    L, A   ; Ergebnis in Registerpaar HL
RET          ; alles fertig
```

In Anhang A finden Sie eine vollständige 'Bildschirm-Werkzeugtasche', mit dem Sie (mit einer Abart der obigen Subroutine) auf dem Bildschirm beliebig ein Zeichen anzeigen können; außerdem enthält es Routinen, um den Schirm nach links, rechts, oben und unten um jede gewünschte Pixelzahl abzurollen (bis zur Höchstzahl auf dem Schirm, versteht sich). Diese Scrolling-Subroutinen liefern gleichmäßige Bewegung, weil sie immer nur eine Pixelreihe bewegen, statt jeweils einen Block von acht Reihen.

Dieses Werkzeug ist zwar außerordentlich leistungsfähig und erleichtert gewiß das Schreiben vieler Programme, aber es gibt doch Umstände, unter denen Sie auf dem Schirm anzeigen und die Ausgabe mit dem ROM steuern wollen. Beispielsweise erweitert das Werkzeug Funktionen und Befehle nicht zu ihrer vollständigen Form, obwohl es die Regeln für Negativschrift und Benutzergrafik beachtet.

In diesem Fall können Sie jedes Zeichen anzeigen, wenn Sie eine der 'RST'-Einsprungadressen im ROM verwenden.

'RST'-Einsprungadressen sind acht Spezialadressen, die mit einer Einzelbyte-Anweisung statt dem üblichen 3-Byte-CALL-Befehl aufgerufen werden können. Der Befehl 'RST 10 H' ruft die Routine auf, die bei Adresse 10 H (16 dezimal) beginnt. Nebenbei: Die anderen verfügbaren RST-Einsprungadressen sind:

RST 0 h	Führt einen NEW-Befehl aus
RST 8 h	Löst die Fehlerroutine aus. Das Byte danach nennt den Fehlercode, 0-1 BH
RST 10 h	Ausgabezeichen an laufenden Strom
RST 18 h	Vom Interpreter verwendet
RST 20 h	Vom Interpreter verwendet
RST 28 h	Fließpunktrechner

RST 30 h Speicherverwaltungs-Routine
RST 38 h Keyboardabtastung und FRAMES-Uhrzähler

Mit der Routine 'RST 10 h' sollten Sie Register A mit dem Code des anzuzeigenden Zeichens laden, dann RST 10 h einschließen, worauf das Zeichen an der laufenden Bildschirmposition angezeigt wird.

Sie können die Position dadurch verändern, daß Sie eine Routine bei Adresse ØDD9H (3545 dezimal) benützen. Ich habe diese Routine in den folgenden Programmen 'PRNTAT' genannt. Register B sollte auf die Bildschirmreihe gesetzt werden, wo 24 die oberste Zeile und 1 die unterste darstellt, während Register C auf die Spaltennummer gesetzt werden muß, wo 32 die linke Kante und 2 die rechte Kante bezeichnet.

Bei Anwendung dieser Methode gibt es aber einen kleinen Haken. Der Spectrum hat die Fähigkeit, Daten auf einen von 16 verschiedenen Strömen auszugeben (siehe 'Winke und Tips'), und bevor Sie 'RST 10 h' verwenden, müssen Sie das geeignete Ausgabegerät wählen. Sobald das geschehen ist, brauchen Sie sich für den Rest Ihres Programms nicht mehr damit zu befassen, außer natürlich, Sie ändern auf einen anderen Wert ab. Der Bildschirm hat Gerät/Strom-Nummer 2, und eine Routine bei Adresse 1601 h wählt den laufenden Kanal. Hier ein vollständiges Beispiel:

```
LD    A, 2        ; Bildschirmausgabe wählen
CALL  1601H
LD    B, 24       ; Zeile 0 (siehe Hinweise oben)
LD    C, 33       ; Spalte 0
CALL  ØDD9H       ; Anzeigeposition setzen
LD    A, 41H      ; Code für 'A'
RST   10H         ; anzeigen
RET               ; zu BASIC zurückkehren
```

Nach Abschluß von 'RST 10 h' ist die Systemvariable SPOSN automatisch auf die nächste Position aktualisiert. Um also das Alphabet auf Zeile 10 des Bildschirms anzuzeigen, würde folgende Routine genügen:

```
LD    A, 2        ; Bildschirmausgabe wählen
CALL  1601 h
LD    B, 10       ; Zeile 14 (24−14 = 10)
LD    C, 30       ; Spalte 3 (33−3 = 30)
```

```
        CALL PRNTAT  ; Anzeigeposition setzen
                     (0DD9H)
        LD   A, 41H  ; Code für 'A'
LOOP:
        PUSH AF      ; Zeichen sichern
        RST  10H     ; Zeichen anzeigen
        POP  AF      ; wiederherstellen
        INC  A       ; nächster Buchstabe im Alphabet
        CP   5 BH    ; ist 'Z' schon vorbei?
        JR   C, LOOP ; nein, also neues Zeichen an-
                     zeigen
        RET          ; zu BASIC zurückkehren
```

Zwei nützliche Adressen:

CALL ØD6BH - löscht den Bildschirm (dezimal 3435)
CALL ØDFEH - rollt den Schirm nach oben (dezimal 3582)

Die zweite, also 'SCROLL', läßt die laufende Anzeigeposition unberührt, so daß Sie ganz neuartige Wirkungen erzielen können, wenn Sie auf einer Mittelzeile des Displays anzeigen, es um eine Zeile hinaufrollen und wieder anzeigen. Wenn Sie an einen ZX81 gewöhnt waren, fühlen Sie sich dabei dann vielleicht wieder wohl!

Sie können mit PLOT auf dem Bildschirm zeichnen, wenn Sie eine Routine bei Adresse 22E5H (dezimal 8933) verwenden. Diese Routine geht davon aus, daß Register B die Y-Koordinate (0-175) und Register C die X-Koordinate (Ø-255) enthält. Wenn Sie gezeichnete Punkte löschen wollen, können Sie eine von zwei Methoden verwenden.

Erstens können Sie einen Befehl 'SET Ø, (IY + 87)' einfügen. Das entspricht einem 'OVER 1' in BASIC, da alle nachfolgenden durch PLOT gezeichneten Punkte den Punkt an dieser Position auf dem Bildschirm **umkehren.**

Zweitens können Sie eine Anweisung 'SET 2, (IY + 87)' einfügen, die dafür sorgt, daß alle nachfolgenden Punkte nicht Ink-, sondern Paper-Punkte werden. Das entspricht '**INVERSE** 1' in BASIC.

Beide Methoden können jederzeit umgekehrt werden, wenn man das entsprechende Bit neu setzt, also RES Ø, (IY – 87) oder RES 2, (IY + 87). Ein vollständiges Beispiel für eine Routine, die PLOT verwendet, findet sich im Assemblierer-Abschnitt.

Noch eine nützliche Sache im Verhältnis zum Display - eine Subroutine, die eine 16-Bit-Zahl als vier Dezimalziffern anzeigt. Die Routine wird verwendet vom ROM, um Zeilennummern anzuzeigen, wenn Sie die Befehle LIST und LLIST verwenden, sie kann aber in Assembliererprogrammen gut genutzt werden. Die Routine beginnt bei Adresse 1A28H (6696 dezimal) und setzt voraus, daß die Register HL die Adresse einer 16-Bit-Zahl enthalten. Leider muß die 16-Bit-Zahl mit dem bedeutsamsten Bit zuerst gehalten werden, aber das ist nur ein kleiner Nachteil. Register E sollte einen von drei Werten enthalten - 32, wobei die Zahl mit vorangehenden Leerräumen angezeigt wird, 38, dann kommen vorher Nullen, oder 255, wo nicht-bedeutsame Nullen gar nicht angezeigt werden. Das folgende Programm zeigt die Zahlen Ø bis 9999 in der Bildschirmmitte an. Sie drehen sich im Kreis herum, bis Sie BREAK drücken:

```
SELDEV:   EQU 16Ø1H
PRNTAT:   EQU ØDD9H
PRTNUM:   EQU 1A28H
BRKTST:   EQU 1F54H
ENTER$HERE:
          LD    A, 2              ; Bildschirm wählen
          CALL  SELDEV
START$COUNT:
          LD    BC, Ø             ; Zähler initialisieren
NEXT$NUM:
          PUSH  BC                ; Zählung sichern
          LD    B, 24 - 1Ø        ; anzeigen auf Zeile 10
                                    (24 - 10 = 14)
          LD    C, 33 - 15        ; Spalte 15
                                    (33 - 15 = 18)
          CALL  PRNTAT            ; siehe oben
          POP   BC                ; Zählung wiederher-
                                    stellen
          LD    HL, COUNT + 1     ; Speicherbereichs-
                                    adresse
          LD    (HL), C           ; am wenigsten bedeut-
                                    sames Byte
          DEC   HL
          LD    (HL), B           ; bedeutsamstes Byte
```

```
        PUSH BC              ; Zählung sichern
        CALL PRTNUM          ; Routine bei 1A28H
                               im ROM
        CALL BRKTST          ; Testroutine unter-
                               brechen - 1F54H
        POP  BC              ; Zählung wiederher-
                               stellen
        RET  NC              ; falls BREAK, zu BASIC
                               zurückkehren
        INC  BC              ; nächster Wert
        LD   HL, 9999        ; Höchstwert
        XOR  A               ; Übertragsflagge löschen
        SBC  HL, BC          ; feststellen, ob Maxi-
                               mum erreicht
        JR   NC, NEXT$NUM    ; Sprung, um nächste
                               Zahl anzuzeigen
        JR   START$COUNT     ; Zähler neu setzen
COUNT:  DEFW 0               ; Datenbereich für Zähler
```

Dieses Programm kann so, wie es dasteht, in den Assembler eingegeben werden. Starten Sie sofort mit der Eingabe:

```
RANDOMIZE USR (PEEK 23730 + 256* PEEK 23731 + 1)
```

Für sich allein ist das Programm nicht besonders nützlich, aber als Hinweis darauf, wie **andere** Programme die bisher erwähnten Routinen beinhalten können, ist dieses Zählerprogramm überaus lehrreich.

Farbe

Nachdem Sie gesehen haben, wie man den Bildschirm für die Anzeige von Normaltext und -grafik verwendet, läßt sich Farbe leicht hinzufügen.

Wenn Sie 'RST 10 h' benützen, können Ihre Programme die Farbe dadurch verändern, daß sie 'Ink'- und 'Papier'-Steuerzeichen enthalten, gefolgt vom entsprechenden Farbencode - siehe Seiten 114/115 des Sinclair-Handbuchs. Die Farbencodeausgabe in dieser Weise setzt die Attribute aller folgenden Zeichen, da sie in den zeitweiligen Farbensystemvariablen ATTRT und MASKT gespeichert werden (wenn Sie durchsichtige Farben verwenden). Diese Systemvariablen werden stets für Ausgabe zum

Bildschirm verwendet und von den ständigen durch den Festspeicher neu geladen, der dazwischen BASIC-Befehle ausführt. Falls Sie das nicht selbst tun, brauchen Sie sich nur mit den zeitweiligen Feldern zu befassen.

Falls Sie vorhaben, Zeichen direkt in die Bildschirmkarte zu setzen, müssen Sie auch die Farbsteuerzeichen in die Attributkarte setzen, aber diese Karte wird als ein Byte für jede Position auf dem Bildschirm gehalten, beginnend bei Adresse 22528, so daß der Umgang damit nicht schwer ist.

Die Randfarbe (Border) wird dadurch geändert, daß man die Farbcodenummer zu Port EfH (254 dezimal) schickt:

```
BORDPORT: EQU ØEFH
    LD      A, (COLOUR)      ; Ihr eigener Datenbereich
    OUT     (BORDPORT), A    ; Randfarbe setzen
```

Wenn Sie die Systemvariable BORDER verwenden wollen, müssen Sie berücksichtigen, daß dieses Datenfeld auch die Attribute für den unteren Bildschirmteil enthält. Eine Routine, um den Rand auf die in der Systemvariablen BORDER bestimmte Farbe zu setzen, wäre:

```
BORDCR: EQU 23624            ; Systemvariable definieren
    LD      A, (BORDCR)      ; Variable holen
    AND     38H              ; Randfarbe isolieren
    RRCA                     ; in untere 3 Bits setzen
    RRCA
    RRCA
    OUT     (ØFEH), A        ; Randfarbe setzen
    RET
```

Ton

Gehen wir weiter zum Summer. Maschinencodeprogramme können diese Einrichtung sehr gut nützen und sind vielseitiger bei der Art der hervorgebrachten Töne, weil die Zeit für die dazwischen ausgeführten Befehle so kurz ist, daß der Ton praktisch nicht unterbrochen wird. Bei der Verwendung des BEEP-Befehls in BASIC kann man die kleinen Unterbrechungen im Ton hören, wenn andere Befehle befolgt werden.

Die ROM-Routine ist an Adresse 3B5H (= 949 dezimal) und

setzt voraus, daß die Registerpaare DE und HL vor Eingang gesetzt werden.

Das Registerpaar HL liefert die Frequenz des zu erzeugenden Tons. Ein Wert Null ist außerordentlich hoch, ein Wert – 1 (oder FFFF hex) außerordentlich niedrig. In der Praxis verlangt das mittlere C (BEEP n, Ø) in HL einen Wert von 666 h.

Das Registerpaar DE sollte auf die Ausgabelänge gesetzt werden, aber Vorsicht! Die Länge des Tons hängt auch von der Frequenz ab. Wenn Sie DE also beispielsweise auf 1ØØ h setzen, wird die Tondauer doppelt so lang, wenn die Frequenz halbiert wird. Das heißt, Sie müssen den Wert von DE abstimmen, damit eine vernünftige Tondauer herauskommt. Für das mittlere C, wo HL auf 666 h gesetzt ist, verlangt ein Ton von etwa einer Sekunde, daß DE auf 1ØØ h gesetzt wird.

Probieren Sie Folgendes aus:

```
BEEPER: EQU 3B5H
GLISSANDO:
    LD      HL, 1000H    ; Anfangsfrequenz
    LD      DE, 8        ; Verzögerung
SLIDE:
    PUSH    HL
    PUSH    DE           ; Daten sichern
    CALL    BEEPER       ; ein Geräusch erzeugen
    POP     DE           ; Register wiederherstellen
    POP     HL
    LD      BC, 8        ; Frequenzerhöhung
    XOR     A
    SBC     HL, BC       ; neue Frequenz berechnen
    JR      NC, SLIDE    ; Schleife, während HL nicht Null
    RET                  ; Rücksprung zu BASIC
```

Dieses kleine Programm liefert einen ziemlich gleichmäßig gleitenden Ton. Vielleicht können Sie es noch verbessern, um komplette Gleichmäßigkeit herzustellen. Auf jeden Fall ist es fast unmöglich, ihn mit BASIC so klar zu erzeugen, weil der Ton stets 'rauhe Kanten' hat. Manchmal können diese Kanten nützlich sein, aber wenn Sie verschiedene Werte für HL und DE ausprobieren, können Sie wirklich eine Vielzahl von Geräuschen hervorbringen.

Der Drucker

Ein Großteil des bisher zum Display Gesagten gilt auch für den Drucker. Innerhalb bestimmter Grenzen funktionieren sogar alle Bildschirmroutinen auf dem Drucker, wenn das gewählte Gerät einfach von 2 auf 3 abgeändert wird! Vor allem 'RST 10' gibt alle nachfolgenden Zeichen auf den Drucker aus. Versuchen Sie es noch einmal mit dem Programm zur Anzeige der Buchstaben des Alphabets, ändern Sie aber die erste Zeile ab zu 'LD A, 3', um den Drucker zu wählen:

```
    LD      A, 3        ; Druckerausgabe wählen
    CALL    1601 h
    LD      C, 30       ; Spalte 3 (33 - 3 = 30)
    CALL    PRNTAT      ; Ausdruckposition setzen (0DD9H)
    LD      A, 41H      ; Code für 'A'
LOOP:
    PUSH    AF          ; Zeichen sichern
    RST     10H         ; Zeichen drucken
    POP     AF          ; wiederherstellen
    INC     A           ; nächster Buchstabe im Alphabet
    CP      5 BH        ; geht das über 'Z' hinaus?
    JR      C, LOOP     ; nein, also neues Zeichen drucken
    LD      A0DH        ; 'ENTER', um Zeile zu drucken
    RST     10H
    RET                 ; Rücksprung zu BASIC
```

Vergleichen Sie diese Version mit der vorherigen - es gibt beinahe keinen Unterschied! Die Zeile 'LD B, 10', mit der die Zeilennummer gesetzt wird, ist verschwunden, weil man auf dem Drucker nicht in einer bestimmten Zeile drucken kann. In der Praxis fällt es nicht ins Gewicht, ob die Zeile vorhanden ist oder fehlt, weil der Festspeicher Register B nicht beachtet, wenn der Drucker verwendet wird.

Sie sollten daran denken, daß der Drucker nur dann eine Zeile druckt, wenn entweder ein 'ENTER'-Zeichen (hex d, dezimal 13) geschickt wird oder wenn Sie versuchen, 'tabulatorisch' zu einer Position zu gelangen, die schon überschritten ist. Aus diesem Grund sind vor dem Rücksprung zu BASIC die beiden zusätzlichen Zeilen eingefügt worden.

Beachten Sie, daß Sie 'Tab'-Positionen übermitteln können, indem Sie ein Hex-17-Zeichen eingeben (23 dezimal), gefolgt von einem Einzelzeichen mit der Tab-Position. Wenn diese zu groß ist, können Sie mit einer Fehlermeldung rechnen.

Manchmal ist die Methode, mit 'RST 10' zu drucken, unzureichend. Wenn Sie Grafik drucken wollen, gibt es ein kleines Problem.

Eine Routine bei Adresse ØECDH druckt den Inhalt des Drukkerpuffers 23296 (siehe Sinclair-Handbuch, Kapitel 24). Wenn Sie Ihre Grafik also in diesen Pufferspeicher stellen und CALL ØECDH (dezimal 3789) eingeben, wird sie ausgedruckt. Die Routine setzt den Puffer zusätzlich auf Nullen (entsprechend Leerstellen) zurück.

Der Druckpuffer hat darin sehr viel Ähnlichkeit mit dem Display, da er eine Bit-Karte von 32 Zeichen zu je acht Pixelreihen ist. In diesem Fall entspricht die Anordnung aber genau dem, was Sie erwarten - die zweite Pixelreihe eines Zeichens folgt 32 Bytes darunter.

Wenn Sie den Druckerpuffer benützen wollen, dann verwenden Sie zuallererst eine Routine bei Adresse ØEDFH (dezimal 3807), die den Puffer auf Nullen löscht.

Außerdem ist eine vollständige Kopie des Bildschirms durch Aufruf der Routine bei ØEACH (dezimal 3756) zu erhalten.

ZXASM – ein symbolischer Assemblierer

Wie schon versprochen, folgt hier ein vollständiger Assemblierer für den Spectrum. Diejenigen von Ihnen, die sich für Maschinencode begeistern, werden die Vorteile eines Assemblerprogramms erkennen – man braucht nicht mehr dauernd die Hexcode-Umwandlungstabellen zu wälzen! Wenn Sie mit Maschinencode erst angefangen haben, wird es durch die frühe Benutzung eines Assemblierers um vieles leichter, Ihre Programme zum Laufen zu bringen, weil die Gefahr falsch codierter Anweisungen erheblich verringert wird.

Was ist ein Assemblerprogramm nun genau? Wie Sie vermutlich wissen, verstehen Computer BASIC in Wirklichkeit gar nicht, sondern hören nur auf Zahlen, genannt 'Maschinencode', weil das der Code ist, den die Maschine erkennt. Leider können die Menschen sich nicht ohne große Mühe merken, was jede Zahl leistet. So sind Assemblierer entstanden. Dieses Programm verschafft Ihnen die Möglichkeit, anstelle von Zahlen einen **mnemotechnischen** Code zu verwenden, und da der mnemotechnische Code (kurz gesagt) eine grobe Beschreibung für den Ablauf der Anweisung bietet, ist er viel leichter anzuwenden.

Dieser Assemblierer gestattet vor allem den Gebrauch des kompletten mnemotischen Standardcodes von Zilog, einschließlich Symbole als Ersatz für Daten und Befehlsadressen. Darüber später mehr. Zulässig sind mehrere zusätzliche Anweisungen, so daß Sie beim Schreiben Ihrer Routinen besonders flexibel sind. Beispielsweise können Sie mit der ORG-Anweisung Ihre Routine an jeder gewünschten Adresse assemblieren – Sie können diese Adresse mitten im Assemblieren sogar ändern, ohne auf Probleme zu stoßen.

Wenn Sie Anfänger sind oder von Maschinencode nichts verstehen, können Sie sich bei anderen Büchern zum Thema weiteren Rat holen.

ZXASM (wie dieser Assemblierer heißt) ist allerdings ziemlich umfangreich, und die wahren Vorteile davon haben diejenigen unter Ihnen mit entweder einem Spectrum 48K oder einem Microdrive. Das liegt daran, daß die Anweisungen auf verschiedene Art eingegeben werden können:

1. Als direkte INPUT-Daten. Das hat den Vorteil der 'Unverzüglichkeit', der naheliegende Nachteil ist der, daß die Routine jedesmal von neuem eingegeben werden muß.
2. Enthalten in REM-Anweisungen. Hier kann die Routine natürlich entweder getrennt oder gemeinsam mit dem Assembliere sichergestellt werden, so daß sie nach Wunsch assembliert werden kann. Der Nachteil hier: Die ganze Routine muß zusammen mit dem Assembliere in den Speicher passen, und bei einer 16K-Maschine ist das nicht praktisch.
3. Als gespeicherte Daten in einer Datei. Besitzer von Microdrive-Geräten können die Anweisungen in einer Diskettendatei festhalten, die man wieder in den Assembliere einlesen kann. Der einzige Nachteil hierbei ist der, daß Sie einen Microdrive haben müssen! Aber im Ernst: Wenn Sie vorhaben, mit dem Spectrum in größerem Umfang zu programmieren, wäre das eine lohnende Geldanlage.

Das bedeutet nicht, daß der Assembler bei der 16K-Version nutzlos ist, weil mit einer oder zwei kleinen Ausnahmen der gesamte Maschinencode in diesem Buch mit dem Assembler in 16K entwickelt worden ist, während der Code direkt über das Keyboard eingegeben wurde. Wie in der Einleitung schon erwähnt, wird bei uns eine Maschinencodeversion dieses Assembliere auf Kassette erhältlich sein. Da sie weniger Speicherplatz braucht, sind alle Möglichkeiten verfügbar.

Zur Beachtung: Dieser Abschnitt enthält mehrere Beispiele von Routinen in Assemblersprache. Nicht alle davon sind voll funktionierende Programme, sondern dienen nur zur Belehrung. Befassen Sie sich in diesem Sinn damit und versuchen Sie nicht, sie einzugeben, um zu sehen, was dabei herauskommt, wenn Sie nicht ausdrücklich dazu aufgefordert werden.

Der Gebrauch von Symbolen

Eines der größten Probleme, vor denen einen Maschinencode-Programmierer steht, besteht darin, Adressen für Sprung- und Aufrufanweisungen zu berechnen, und sie dann neu zu berechnen, wenn später zusätzliche Befehle eingefügt werden müssen. Verwendet man statt dessen Symbole, dann berechnet der Assembliere alle diese Adressen für Sie. Hier ein Beispiel.

Die folgende Routine füllt den ganzen Bildschirm mit Inkpunkten:

```
LD      HL, 32768       ; lade Displaydatei-Adresse
LD      BC, 704         ; 24 Zeilen zu 32 Zeichen
LD      (HL), 255       ; Display ganz auf Ink setzen
INC     HL              ; nächste Reihe auf Bildschirm
DEC     BC              ; Zähler verringern
LD      A, B            ; prüfen, ob Zähler Null
OR      C
JR      NZ, – 8         ; nein - wiederholen
RET                     ; Rücksprung zu BASIC
```

Alles schön und gut, werden Sie sagen, aber wenn Sie jetzt zwischen 'INC HL' und 'DEC BC' neue Anweisungen einfügen wollen, müssen Sie zum Ausgleich auch den Befehl 'JR NZ, – 8' ändern. Sehen Sie sich das hier an

```
DFILE:  EQU 32768
LENGTH: EQU 704
BEGIN:
    LD      HL, DFILE       ; Displayadresse
    LD      BC, LENGTH      ; Zahl der Bytes
LOOP:
    LD      (HL), 255       ; auf 'Ink' setzen
    INC     HL              ; nächste Reihe
    DEC     BC              ; Zähler verringern
    LD      A, B            ; prüfen, ob Zähler = 0
    OR      C
    JR      NZ, LOOP        ; nein - wiederholen
    RET                     ; Rücksprung zu BASIC
```

Das erste, worauf man hier achten muß, sind die zusätzlichen EQU-Anweisungen. Was bewirken sie? Die Antwort ist einfach. Im Anschluß an einen EQU-Befehl wird der Assemblierer, sobald er (beispielsweise) den Namen DFILE sieht, (in diesem Fall) den Wert 32768 einsetzen. Damit bedeutet 'LD HL, DFILE' **genau** dasselbe wie 'LD HL, 32768'. Sie werden sich vielleicht fragen, was Ihnen das um Himmels willen bringen soll. Angenommen, Ihre Routine verwendet diesen Wert mehrmals, und Sie müssen ihn jetzt ändern. Statt das ganze Programm auf jedes Vorkommen von 'LD HL, 32768' abzusuchen, könnten Sie einen einzigen EQU-Befehl abändern und neu assemblieren.

Zweitens: Beachten Sie, daß vor der Zeile 'LD (HL), 255' das Symbol 'LOOP' aufgetaucht ist. Das heißt: Sobald der Assemblierer von jetzt an das Symbol 'LOOP' sieht, fügt er die **Adresse** des Befehls 'LD (HL), 255' ein. Alle Sprung- und Aufrufbefehle können jetzt diesen Namen anstelle der üblichen Adresse verwenden. Dadurch wird alles viel einfacher.

Sie können, wie oben gezeigt, ein Symbol auf zwei verschiedene Arten definieren. In jedem Fall **muß** nach dem Namen ein Doppelpunkt (:) kommen.

Der Name kann an beliebiger Stelle in der Routine erscheinen – es gibt bestimmte Einschränkungen, die unten aufgeführt sind, aber zulässig sind Bezugnahmen vorwärts wie rückwärts. Beispiel:

```
        OR      A
        JP      NZ, ALLDUN
        LD      (HL), A
ALLDUN: RET
```

Bei diesem Beispiel 'sieht' der Assemblierer den Befehl 'JP NZ, ALLDUN', **bevor** er weiß, wo ALLDUN überhaupt ist. Diese Art von Symbol wird als 'unaufgelöst' bezeichnet und wird in einer eigenen Tabelle festgehalten. Wenn alle Anweisungen assembliert sind, läuft er an der Liste unaufgelöster Symbole entlang, um festzustellen, wo sie vorgekommen sind, und setzt die richtigen Werte ein. Bleiben welche übrig, kommt es zu einer Fehlermeldung. Während der Assemblierung werden unaufgelöste Symbole mit einem Stern in Negativschrift bezeichnet.

Die ORG-Anweisung

Wenn Sie Ihre Assembleranweisungen eingeben, muß der Assemblierer wissen, wohin er den Maschinencode setzen soll. Er geht davon aus, daß Sie ihn über RAMTOP setzen wollen (siehe Systemvariable im Sinclair-Handbuch), aber das braucht nicht immer zuzutreffen. Sie wollen vielleicht zwei oder drei Routinen über RAMTOP haben und müssen dem Assemblierer deshalb sagen können, wohin der Code gehen soll. Das geschieht durch einen ORG-Befehl. Die Adresse nach dem ORG-Befehl wird für die gesamte nachfolgende Assemblierung verwendet. Ein Beispiel:

```
        ORG   32000          ; Code an diese Adresse
                               setzen
BWAIT:  CALL  1F54H          ; ROM-Routine für Prüfung
                               auf BREAK
        JR    NC, BWAIT      ; warten, bis BREAK ge-
                               drückt wird
        RET
        ORG   32100          ; weiterer Code an dieser
                               Adresse
        LD    A, (HL)
        OR    A
        JP    NZ, BWAIT
        RET
```

Die Routinen können in einem BASIC-Programm verwendet werden, wenn man die 'ORG'-Adresse in einer USR-Funktion angibt - in diesem Fall LET X = USR 32000.

Daten angeben

Ihre Maschinencode-Routinen brauchen vielleicht einige Datenbereiche, entweder Einzelbytes oder Bytepaare (Wörter), wo das niederwertige Byte vor dem hochwertigen kommt.

Dafür stehen zwei Befehle zur Verfügung:

DEFB um ein einzelnes Datenbyte zu definieren
DEFW um ein Standard-Wörterpaar zu definieren

Normalerweise würde vor dem Befehl ein Symbol verwendet werden, damit während der ganzen Routine namentlich auf die Adresse der Daten Bezug genommen werden kann.

Mit der folgenden Routine können Sie nach einer Koordinatentabelle, die bei der Symboladresse 'DATA' beginnt, auf dem Bildschirm Punkte zeichnen:

```
FF58            PLOT:   EQU   22E5H   ; ROM 'PLOT'-Routine
FF5B  210000    LD      HL, DATA      ; Punkte-Koordinatentabelle
FF5E  46        LD      B, (HL)       ; Zahl der Punkte
FF5F  23        INC     HL            ; Adresse der ersten Punkte-
                                        koordination
                PLLOOP:
FF60  C5        PUSH    BC            ; Register sichern
FF61  4E        LD      C, (HL)       ; X-Koordinate
```


RAM TOP 65368

```
FF5F 23      INC   HL
FF60 46      LD    B, (HL)     ; Y-Koordinate
FF61 23      INC   HL
FF62 E5      PUSH  HL          ; Zeiger zu Tabelle sichern
FF63 CD      CALL  PLOT F5 22  ; Punkt zeichnen
FF66 E1      POP   HL          ; Register wiederherstellen
FF67 C1      POP   BC
FF68 10F3    DJNZ  PLLOOP      ; für alle Punkte wiederholen
FF6A C9      RET               ; alles fertig
FF6B DATA:   03    DEFB 3      ; Zahl der folgenden Punkte
FF6C         0A    DEFB 10     ; X-Koordinate
FF6D         64    DEFB 100    ; Y-Koordinate
FF6E         0B    DEFB 11     ; X
FF6F         65    DEFB 101    ; Y
FF70         0C    DEFB 12     ; X
FF71         66    DEFB 102    ; 102
```

Sie können versuchen, die Routine genauso einzugeben, wie sie hier steht. Bevor Sie anfangen, sollten Sie aber die Adresse in RAMTOP verringern, indem Sie CLEAR 32599 als direkten Befehl eingeben. Nachdem die Routine (ohne Fehler!) assembliert worden ist, fahren Sie sie durch Eingabe von LET X = USR 32600.

Das selbstsuchende Lesen (Autoscan)

Das betrifft nur diejenigen von Ihnen, die mehr als 16K RAM verfügbar haben, weil es davon abhängt, daß die gesamte Assembler-Routine in REM-Anweisungen steht, was mit 16K einfach nicht geht.

Wenn Sie Autoscan wählen, sucht ZXASM das ganze Programm nach Ihren Assemblieranweisungen ab und assembliert sie automatisch. Dadurch sind große Routinen viel leichter zu entwikkeln; man kann sie getrennt aufbereiten und bearbeiten.

Schreiben Sie ein Programm, das ausschließlich aus REM-Anweisungen besteht, wo Ihre Befehle in diesen Anweisungen enthalten sind. Ein Anfangs-REM sollte lediglich eine offene Klammer enthalten - also etwa 10 REM.

ZXASM sucht danach und geht davon aus, daß alle nachfolgenden BASIC-Befehle REM-Anweisungen sind, die Ihren Code ent-

halten. Das setzt sich fort, bis eine REM-Anweisung gefunden wird, die nur eine Schlußklammer enthält - 999 REM.
Zwischen REM-Befehl und Klammer darf es keine Zwischenräume geben.
Außerdem sollten Sie nur Zeilennummern unter 1000 verwenden, weil ZXASM höhere Zeilennummern benützt. Wenn Sie mehr brauchen, könnten Sie Zeilennummern zwischen 3000 und 5000 nehmen, die dort nicht benützt werden.
Sobald Ihr kleines 'Programm' eingegeben ist, sichern Sie es für sich auf Kassette. Wenn Sie ZXASM fahren, werden Sie aufgefordert, den Namen Ihrer Routine auf Band zu nennen, dann wartet es darauf, daß Sie das Band starten. Dazu später 'ZXASM in Betrieb'. Inzwischen hier ein Beispiel für Sie:

```
 10 REM Berechnen, wieviel Speicherplatz gebraucht wird
 20 REM Keine ORG-Angabe, also wird der CODE
 30 REM automatisch über RAMTOP gesetzt.
 40 REM (
 50 REM PROG:  EQU 23635
 60 REM ELINE:  EQU 23641
 70 REM START: LD    HL, (ELINE)
 80 REM        LD,   DE, (PROG)
 90 REM        XOR,  A
100 REM        SBC   HL, DE
110 REM        LD    B, H
120 REM        LD    C, L
130 REM        RET
140 REM)
```

Sie brauchen es nicht schön säuberlich in Kolonnen zu setzen wie hier, aber lesbarer wird es dadurch schon. Der Assemblierer setzt für Ausdruckzwecke alles automatisch in Kolonnen.
Bei diesem Beispiel geben Sie das obige Programm so ein, wie es dasteht, dann sichern Sie es unter dem Namen – MEMO - RY.ASM. Später, wenn Sie die Betriebsanweisungen lesen, erfahren Sie, wie Sie das Programm wieder eingeben.
Denken Sie daran: Wenn Sie in eine REM-Anweisung einen Doppelpunkt setzen, geht der Spectrum davon aus, daß die als nächste gedrückte Taste ein Schlüsselwort sein wird (der Cursor verwandelt sich in ein 'K'). Um dem zu entgehen, drücken Sie erneut REM, geben den Rest der Assemblerzeile ein, gehen zurück und löschen das zusätzliche Schlüsselwort.

Maschinencode sicherstellen

ZXASM dient als Übersetzer - Assembleranweisungen werden in Maschinencode verwandelt. Ihre Routine kann, sobald sie assembliert ist, durch Verwendung der CODE-Option nach dem SAVE-Befehl gesichert werden. Beispielsweise sichert

SAVE "MEMORY" CODE 32500, 12

12 Bytes beginnend bei Adresse 32500. Vergessen Sie nicht, den Befehl CLEAR zu verwenden, bevor Sie diese Routine wieder laden. Der entsprechende LOAD-Befehl lautet dann:

LOAD "MEMORY" CODE

Da der wirksame Maschinencode gesichert worden ist, muß die Routine nicht immer wieder neu assembliert werden, wenn man sie in den Speicher laden will. Das ist offenkundig ein Vorteil, wenn Sie Programmkopien an Ihre Freunde weitergeben - Sie brauchen Ihnen nur den gesicherten Code zu geben, nicht das eigentliche Assembler-Ursprungsprogramm.

Anderes

Jede Zahl, die in eine Anweisung eingegeben werden soll, kann in einer der folgenden Formen eingegeben werden:

1. Als Dezimalzahl (z. B. 1234)
2. Als Hexzahl mit dem Zusatz 'H'. Beachten Sie, da die erste Ziffer eine Zahl zwischen 0 und 9 sein muß, sonst wird das Ganze als Symbolname behandelt. Z. B. steht 0FFFFH für – 1, 123H für 291 dezimal.
3. Als definiertes Symbol. Das erste Zeichen muß ein Buchstabe zwischen A und Z sein, und nur die ersten sechs Zeichen sind von Bedeutung. Dem Symbolnamen kann wahlweise ein positiver oder negativer Dezimal-Versetzungswert sein. Beispiele: DFILE, COLUMNUMBER, A 22 + 1, START – 6.
4. Als undefiniertes Symbol mit ähnlichen Aufbaubeschränkungen.

Anweisungen, die 8 Bit-Daten erfordern, relative Sprünge aber ausschließen (etwa ADD A, 23) können Daten vom Typ 4 (undefiniert) nicht verwenden. Ebenso können ORG, EQU, DEFB und DEFW Daten von Typ 4 nicht verwenden.

Obwohl JR- und DJNZ-Anweisungen eine relative Versetzung benötigen, wird diese vom Assemblierer berechnet. So sollten JR und DNJZ mit einer **absoluten** Adresse angegeben werden (wie bei einem JP-Befehl); die Arbeit macht dann der Assemblierer. Beispiel: JR NC, ØFF8ØH. Der Assemblierer berechnet nach der laufenden Assembleradresse die Versetzung, die erforderlich ist, um Adresse ØFF8ØH zu erreichen.

Zilog-Mnemotechnik enthält mehrere Widersprüchlichkeiten, die der Standardisierung wegen vom Assemblierer berücksichtigt worden sind. Vor allem benötigen die Anweisungen ADD, ADC und SBC **zwei** Operanden (etwa SBC A, 38), während SUB, AND, OR, XOR und CP nur einen benötigen (etwa SUB 38). Der EX-Befehl verwendet die Operanden EX DE, HL - das Umgekehrte wird nicht angenommen.

Die Anweisungen JP (HL), JP (IX) und JP (IY) sind logisch unrichtig, aber der Assemblierer nimmt sie nur in dieser Form an.

ZXASM verwandelt alle Eingaben in Großbuchstaben. Es spielt also keine Rolle, ob Sie Ihr Programm in Klein- oder Großbuchstaben oder sogar gemischt eingeben.

Sie können neben Ihren Assemblerbefehlen Kommentare anbringen, wenn Sie davor einen Strichpunkt setzen. ZXASM beachtet den Rest einer Zeile nach einem Strichpunkt nicht.

Wenn Sie Autoscan verwenden (falls Sie mehr als 16K RAM haben), können Sie in einer Zeile mehr als einen Befehl unterbringen, wenn Sie zwischen die Anweisungen Ausrufungszeichen setzen - etwa so:

```
1Ø REM (
2Ø REM ld hl, (23641)! ld de, (23635)
3Ø REM xor a! sbc hl, de! ld b, h! ld c, l
4Ø REM ret
5Ø REM)
```

Dieses Beispiel entspricht der Routine "MEMORY" vorhin.

Fehlermeldungen

Wir machen alle Fehler - bedaure! - die sich aus einer Vielzahl von Gründen ergeben können. Man kann sich bei einer Anweisung verschreiben oder eine Registerkombination verwenden,

die dem Befehl nicht entspricht; das sind nur zwei Beispiele. Wenn ZXASM in Ihrer Routine auf etwas stößt, das nicht in Ordnung ist, zeigt es an der betreffenden Zeile einen Fehler an und fordert Sie auf, neue einzugeben. (Wenn Sie Autoscan verwenden, fährt ZXASM einfach fort und überläßt Ihnen die Fehlerkorrektur nach dem Abschluß.)

Als Beispiel:

```
7FD8 LD TT, 200 ; wer kennt ein Registerpaar 'TT'?
```

ZXASM zeigt an:

```
7FD8 ERROR 9 LD TT, 200
```

und wartet auf die richtige Eingabe.

Hier eine vollständige Liste der verschiedenen Fehlercodenummern und ihrer Bedeutung:

Fehlernummer	Bedeutung
2	Symbol schon definiert
6	Zahl außer Bereich (−32767 bis 65535)
9	Ungültige Anweisung, unzulässiges Register/Registerpaare, falscher Operand.
98	Tabelle 'unaufgelöster' Symbole ist voll. Sie können die Grenze durch Veränderung der Variablen 'nu' in Zeile 1010 erhöhen, aber dann bleibt in der 16K-Version vielleicht nicht genug Speicherplatz. Wenn nicht, sollten Sie versuchen, Ihre Routine so umzubauen, daß weniger unaufgelöste Symbole vorkommen. Das ist beispielsweise dadurch zu erreichen, daß man DEFB- und DEFW-Befehle vorne an die Routine setzt statt untenhin.
99	Tabelle definierter Symbole ist voll. Sie können die Grenze erhöhen durch Veränderung der Variablen 'ns' in Zeile 1010, aber hier gelten ähnliche Einschränkungen wie bei Fehler 98. Wenn Sie die Zahl der Symbole nicht verringern können, lassen sich einige immer durch ihre konkreten Werte ersetzen.

Verbindung mit dem Disassemblierer

Wenn Sie die Symboleinrichtungen bei ZXASm genutzt haben, werden Sie diese Symbole vielleicht wieder verwenden wollen, wenn Sie die Disassemblierer- und Maschinencode-Überwachungsprogramme fahren. Sie könnten sie freilich wieder eingeben, wenn die Aufforderung 'Enter symbols?' erscheint, aber es gibt einen viel einfacheren Weg. Wenn Ihre Assemblierung fertig ist, nehmen Sie eine leere Kassette und:

SAVE "Symbole" DATA S$ ()

Array S$ () enthält alles, was der Disassemblierer braucht, um Ihnen ein genaues Listing Ihrer Routine zu liefern, Datenbezugnahmen, Sprung- und Aufrufbefehle in ihre Assemblerentsprechung zurückzuübersetzen. Wenn Sie den Disassemblierer fahren, sollten Sie sich an das Verfahren halten, eine Symboltabelle vom Band zu laden - siehe Betriebsanweisungen für den Disassemblierer.

ZXASM in Betrieb

Der Assembler wird im folgenden Listing in zwei Teilen gezeigt. Der Grund dafür: Er ist für einen 16K-Spectrum (bei dem nur 9K zur Verfügung stehen) einfach zu groß. Einige Bestandteile sind deshalb in ein eigenes Programm übernommen worden, das automatisch mit dem Hauptprogramm durch MERGE vermischt wird, wenn mehr als 16K festgestellt werden. Diese zusätzlichen Bestandteile umfassen a) Autoscan, wo Ihr Assemblercode in REM-Anweisungen eingegeben werden kann, und b) vergrößerte Symboltabellen (die 64 vereinbarte und unaufgelöste Symbole zulassen).

Beginnen Sie den Assemblierer mit:

LOAD "ZXASM"

Wenn er geladen ist, prüft er, ob mehr als 16K RAM zur Verfügung stehen. In diesem Fall versucht er automatisch ein Programm namens "ZXASM/48K" damit zu mischen. Sie sollten darauf achten, daß Ihr Assemblerband beide Programme hintereinander enthält.

Wenn er schließlich geladen ist, stellt er die Frage:

Hard copy?

Wird die Antwort "Ja" eingegeben, erscheinen alle Ausgaben, die sonst zum Bildschirm geschickt werden, auf dem Drucker. Das ist außerordentlich nützlich, wenn Sie die Autoscan-Einrichtung verwenden, weil Sie dann das Ganze fahren, sich selbst überlassen und eine Tasse Kaffee trinken gehen können!

Als nächstes werden Sie gefragt:

Auto scan?

Wenn die Antwort 'Nein' ist oder Sie nur 'ENTER' drücken, geht ZXASM davon aus, daß alle Eingaben vom Keyboard kommen. Er zeigt die laufende Assembleradresse (Vorgabe ist über RAMTOP) und wartet auf Ihre Anweisungen. In dem folgenden kleinen Beispiel werden alle Meldungen, die der Assemblierer zeigt, fett gedruckt angegeben:

LOAD "ZXASM"
Hard copy? nein
Auto scan? nein (nur, wenn Sie mehr als 16K haben)
7DE8 ld bc, 100
7DEB ret
7DEC (zum Abschluß nur 'ENTER' drücken)
OK

Ist die Antwort auf die Frage 'Autoscan' 'ja', werden Sie weiter gefragt:

Merge source from tape? (Mit Quellprogramm von Band mischen?)

Wenn Sie 'nein' antworten oder nur 'ENTER' drücken, geht ZXASM davon aus, daß Sie Ihren Assemblercode in REM-Anweisungen schon kombiniert haben und sucht sofort danach. Wenn Sie aber 'ja' antworten, fordert das Programm Sie auf, den Namen Ihres Assemblercodes auf Band einzugeben und wartet auf den Bandlauf. Ihr Assemblierer wird mit ZXASM gemischt, dann beginnt die Suche danach.

Nicht vergessen: Wenn Sie Autoscan verwenden, müssen Sie ein 'REM(' einfügen, um ZXASM zu zeigen, daß Ihr Code anfängt, und ein 'REM)', um das Ende zu bezeichnen. Wenn Sie das 'REM)' vergessen, versucht ZXASM sich selbst zu assemblieren und liefert Fehler.

Hier ein Laufbeispiel mit Autoscan:

LOAD "ZXASM"
Hard copy? ja

Auto scan? ja
Merge source from tape? ja
Enter tape name: MEMORY.ASM
START TAPE THEN PRESS A KEY
OK

Das Beste ist jetzt, einen Versuch zu wagen. Im Anfangsteil über Maschinencode sind einige Routinen aufgeführt, die Sie selbst eingeben können, nur um einmal zu sehen, wie das alles zusammenhängt. Außerdem sind das auch an sich ganz nützliche Programme.

Wenn Ihre Assemblierung fertig ist, zeigt ZXASM eine Liste aller von Ihnen verwendeten Symbole neben ihren jeweiligen Adressen. Falls einige Ihrer Symbole mit Bezugnahme nach vorn nicht aufgelöst werden können (weil Sie vergessen haben, das Symbol zu deklarieren), sehen Sie außerdem von diesen eine Liste mit der Assembleradresse, wo das unaufgelöste Symbol besteht. Unaufgelöste Symbole werden als Fehler eingestuft, in der abschließenden Fehlerzählung erscheinen sie auch.

Wenn ZXASM schließlich Fehler entdeckt hat, wird in einer Zeile am Ende die Gesamtfehlerzahl angegeben. Sie sollten sie prüfen und ausbessern und Ihr Programm neu assemblieren.

Hier ein vollständiges Beispiel mit der 'MEMORY'-Routine von vorhin, eingegeben mit einigen 'Probe'-Fehlern:

LOAD "ZXASM"
Hard copy? nein
Auto scan? nein
7F58 prog: equ 23635
7F58 elne: equ 23644
7F58 start
7F58 start:
7F58 ld hl, (eline)
7F5B ld d, (prog)
7F5B ld de, (prog)
7F5F xor a
7F6Ø sbc de, hl
7F6Ø sbc hl, de
7F62 ld b, h
7F63 ld c, l
7F64 ret
7F65

Hinweise für den Gebrauch von ZXASM

Angenommen, Sie geben ein Programm über die Tastatur ein und entdecken plötzlich, daß Sie eine Anweisung falsch eingegeben haben. So könnten Sie eingegeben haben:

```
LD      HL, 100
```

wo Sie in Wirklichkeit LD HL, 100H eingeben wollten. Das stellt kein Problem dar, weil Sie jetzt den ORG-Befehl verwenden können, um die Assembleradresse auf den falschen Befehl zurückzusetzen, ihn neu einzugeben, der Adresse den ursprünglichen Wert zurückzugeben und fortzufahren. Sehen Sie sich das einmal an:

```
7DE8 ld hl, 100      ; sollte LD HL, 100H sein
7DEB ld de, 700 h    ; noch nichts gemerkt!
7DEE ld bc, 32
7DF1 org 7 de, 8 h   ; ORG-Adresse zurücksetzen
7DE8 ld hl, 100 h    ; richtige Anweisung eingeben
7DEB org 7 df 1 h    ; ORG-Adresse wieder umstellen
7DF1 1 dir           ; normal fortfahren
...
...
```

Wenn Sie vergessen, ein Etikett vor eine Anweisung zu setzen, zu der Sie später zurückspringen wollen, können Sie ebenso leicht einen EQU-Befehl eingeben, der den Namen des Etiketts und die entsprechende Adresse angibt. Beispiel:

```
7E00 11 d b, 8
7E02 ld hl, 7 f00h
7E05 cp (hl)         ; das hätte ein Etikett gebraucht!
7E06 jr z, found
7E08 inc hl
```

Jetzt ist der Fehler bemerkt worden - die Zeile bei Adresse 7E05 hätte eingegeben werden **sollen** als 'LOOP: CP (HL)', damit die nächste Anweisung lauten konnte: 'DJNZ LOOP'. Das Problem läßt sich so lösen:

```
7E09 loop:   equ 7 e 05 h
7E09 djnz loop
...
...
```

Denken Sie daran, daß Sie Ihren Maschinencode mit LOAD auch aus einem BASIC-Programm übernehmen können, nicht nur durch einen direkten Befehl.

ZXASM-Listing

Wie vorhin schon erwähnt, besteht ZXASM aus zwei Teilen. Auf diese Weise kann für die Leute mit mehr als 16K RAM die zusätzliche Autoscan-Einrichtung eingefügt werden.

Geben Sie das erste Programm ein und sichern Sie es unter dem Namen "ZXASM". Das zweite Programm sollte unter dem Namen "ZXASM/48K" sichergestellt und unmittelbar hinter ZXASM auf demselben Band gespeichert werden.

Beachten Sie, daß ZXASM/48K mit ZXASM gemischt einige Zeilen überschreibt. Das ist Absicht - verändern Sie keine der Zeilennummern, wenn Sie die Programme eingeben.

Das Programm ist im Aufbau besonders kompliziert. Ich habe zwar neben dem Listing die üblichen Kommentare eingefügt, aber es könnte Ihnen trotzdem schwerfallen, dem allgemeinen Ablauf des Programms zu folgen. Das ist kein großes Wunder, weil ein Assemblierer eben ein ziemlich großes Programm ist. Dieses hier wurde, wo das möglich war, so gestutzt, daß für den kleineren Spectrum die Höchstzahl an Möglichkeiten herauskam.

```
1000 REM ZX Spectrum Assembler
1001 REM ©1982 Phipps Associates
1005 LET j=VAL "1": IF PEEK VAL
"23731">VAL "127" THEN GO SUB VA
L "7900"
1010 LET ns=10: LET nu=10
1020 DEF FN a(x$)=(x$>="A" AND x
$<="Z")
1030 DEF FN n(x$)=(x$>="0" AND x
$<="9")
1035 DEF FN s$(x)=a$(x(x,j)+j TO
 x(x,2))
1040 DEF FN h(x)=INT (x/256)
1050 DEF FN l(x)=x-256*FN h(x)
1060 DEF FN j()=(m$="JP" OR m$="
CALL" OR m$="RET" OR m$="JR")
1070 DEF FN v(x$)=CODE x$(7)+256
*CODE x$(8)
```

mit ZXASM/48K mischen

siehe Text

```
1080 DEF FN i(x)=PEEK x+256*PEEK
 (x+j)
1085 DEF FN r()=di-vdef*(add+2)
1090 DIM s$(ns+nu+j+j,VAL "8"):
DIM l$(VAL "6"): DIM t$(VAL "2")
1095 LET s$(j)=CHR$ NOT j
1100 LET u=NOT j: LET er=u: LET
ed=VAL "237"
1110 LET m$="Hard copy?": GO SUB
 9000: LET hc=j+j+(CHR$ CODE a$=
"J")
1120 REM (reserved)
1130 LET add=FN i(VAL "23730")+V                Vorgabeadresse holen
AL "1"
1140 CLS : PRINT #hc;"Addr Hex                  Negativschrift verw.
     Op   Operand          "''
1200 LET e=NOT j: LET vdef=j: LE
T dl=e: LET dd=e: LET ec=e: LET
l=e: LET w=e
1210 LET h=add: GO SUB 9100: LET
 m$=h$: GO SUB 7700: IF end THEN
 GO TO VAL "5000"
1220 LET m$=""
1230 DIM t(2): DIM x(4,2): LET t
=NOT j: LET p=j: LET t$="00"
1240 FOR x=j TO LEN a$                          Eingabezeile abtasten
1245 IF a$(x)=";" THEN GO TO 130                Kommentare ignorieren
0
1250 IF t<>(a$(x)<>" " AND a$(x)                Wortende bezeichnen
<>",") THEN LET x(p,t+j)=x-j: LE
T p=p+t: LET t=NOT t: IF p>4 THE
N GO TO 1310
1260 NEXT x
1300 IF x(p,j) OR p=j THEN LET x
(p,j+j)=x-j
1310 LET lx=(FN s$(j)(LEN FN s$(                auf Symbol prüfen
j))=":")
1320 LET m$=FN s$(j+lx)                         Befehl holen
1400 LET jump=FN j()                            ist Das JP/JR/CALL?
1410 FOR t=j TO j+j: LET x$=FN s                Opklassen bestimmen
$(j+t+lx): GO SUB 7400: NEXT t
1500 LET n$=t$+m$                               Abtastcode erzeugen
1510 LET l1=(LEN FN s$(lx+2)<>0)                Wieviele Operanden?
: LET l2=(LEN FN s$(lx+3)<>0): I
F l1 THEN GO TO 1600
1520 IF m$="" THEN GO TO 1900                   keine Anweisung
1530 FOR l=j TO j+j: LET i=VAL "                kein Optyp prüfen
2010+l*10": GO SUB 8300: IF NOT
```

```
x THEN NEXT l
1550 GO TO 1630
1600 LET i=VAL "2060+12*20": LET                1/2 Optypen prüfen
 l=NOT j
1610 GO SUB 8300: IF NOT x+12 TH
EN LET i=2070: GO SUB 8300
1615 IF x THEN GO TO 1650
1620 RESTORE VAL "2000+40*(12=0)
": GO SUB 8400: IF x THEN GO TO
1800
1630 IF NOT x THEN LET ec=VAL "9                ungültige Anweisung
"
1640 GO TO 1800
1650 LET n=x-j: RESTORE i+j: LET
 n$=t$: GO SUB 8400: IF NOT x TH
EN LET ec=VAL "9"
1800 IF ec THEN GO TO 1960                      nicht ausgeb. bei Fehl.
1820 IF m$="ORG" THEN LET add=di                auf Pseudo-Op prüfen
: LET h=di: GO SUB 9100
1830 IF e THEN POKE add,dd: IF e                prüf., obIX/IY-Klasse
=j+j THEN POKE add+j+j,dis                      Ausgabedistanz
1840 FOR x=0 TO l-j: READ x$: PO
KE add+x+(e<>0)+x*(e=2),VAL x$:
NEXT x
1900 IF NOT vdef THEN GO SUB 720                Ausgabe Maschinencode
0
1910 IF NOT lx OR ec THEN GO TO                 undefiniertes Symbol?
1960
1920 LET l$=FN s$(lx)( TO LEN FN                Auf Etikett prüfen
 s$(lx)-j): GO SUB 7600: IF v TH                schon vorhanden?
EN LET ec=2: GO TO 1960
1925 IF s=ns+j THEN LET ec=VAL "                volle Tabelle
99": GO TO 1960
1930 LET v=add: IF m$="EQU" THEN                neu ausgleichen
 LET v=di
1940 LET s$(s)=l$: LET s$(s,7 TO
 )=CHR$ FN l(v)+CHR$ FN h(v)
1950 LET s$(j)=CHR$ (s-j)                       Symbolzählung aktual.
1960 GO SUB 7300: IF ec THEN LET                Zeile anzeigen
 er=er+j: GO TO VAL "1200"
1970 LET add=add+l+e                            neue Eingabe abwarten
1990 GO TO 1200
2000 DATA "11LD","1","1","64+T(1
)*8+T(2)"
2001 DATA "13LD","vdef","2","t(1
)*8+6","dl"
2002 DATA "14LD","t(1)=7","3","5
8","dl","dh"
```

```
2003 DATA "15LD","(t(1)=7)*(t(2)
<2)","1","t(2)*16+10"
2004 DATA "23LD","t(1)<>6","3","
t(1)*16+1","dl","dh"
2005 DATA "24LD","t(1)=2","3","4
2","dl","dh"
2006 DATA "24LD","t(1)<4","4","e
d","75+t(1)*16","dl","dh"
2007 DATA "41LD","t(2)=7","3","5
0","dl","dh"
2008 DATA "42LD","t(2)=2","3","3
4","dl","dh"
2009 DATA "42LD","t(2)<4","4","e
d","67+t(2)*16","dl","dh"
2010 DATA "51LD","(t(2)=7)*(t(1)
<2)","1","t(1)*16+2"
2011 DATA "99","","2100"
2020 DATA "RET","201","NOP","0",
"RLCA","7","RRCA","15","RLA","23
","RRA","31","DAA","39","CPL","4
7","SCF","55","CCF","63","HALT",
"118","EXX","217","DI","243","EI
","251","99",""
2030 DATA "NEG","ED","68","RETN"
,"ED","69","RETI","ED","77","RRD
","ED","103","RLD","ED","111","L
DI","ED","160","LDIR","ED","176"
,"LDD","ED","168","LDDR","ED","1
84","CPD","ED","169","CPDR","ED"
,"185","CPI","ED","161","CPIR","
ED","177","99","",""
2040 DATA "30JR","ABS FN r()<128
","2","24","FN r()"
2041 DATA "10INC","1","1","4+T(1
)*8"
2042 DATA "20INC","T(1)<>4","1",
"3+T(1)*16"
2043 DATA "10DEC","1","1","5+T(1
)*8"
2044 DATA "20DEC","T(1)<>4","1",
"11+T(1)*16"
2045 DATA "30DJNZ","ABS FN r()<1
28","2","16","FN r()"
2046 DATA "60RET","1","1","192+T
(1)*8"
2047 DATA "20POP","T(1)<>3","1",
"193+T(1)*16-16*(T(1)=4)"
2048 DATA "20PUSH","T(1)<>3","1"
```

```
,"197+T(1)*16-16*(T(1)=4)"
2049 DATA "3ORST","vdef AND dl<=
56","1","199+INT (dl/8)*8"
2050 DATA "1OJP","t(1)=6","1","2
33"
2051 DATA "3OIM","vdef*(dl<3)*(d
l>=0)","2","ed","70+16*(dl=1)+24
*(dl=2)"
2052 DATA "3OJP","1","3","195","
dl","dh"
2053 DATA "3OCALL","1","3","205"
,"dl","dh"
2054 DATA "3ODEFB","vdef","1","d
l"
2055 DATA "3ODEFW","vdef","2","d
l","dh"
2056 DATA "3ODRG","vdef","0"
2057 DATA "3OEQU","vdef AND lx",
"0"
2059 DATA "99","","0"
2060 DATA "","","SUB","","AND","
XOR","OR","CP","99"
2061 DATA "10","1","1","128+n*8+
T(1)"
2062 DATA "30","vdef","2","198+r
*8","dl"
2063 DATA "99","","0"
2070 DATA "RLC","RRC","RL","RR",
"SLA","SRA","","SRL","99"
2071 DATA "10","1","2","203","n*
8+t(1)"
2072 DATA "99","","0"
2080 DATA "ADD","ADC","","SBC",
99"
2081 DATA "11","t(1)=7","1","128
+n*8+t(2)"
2082 DATA "13","t(1)=7","2","198
+n*8","dl"
2083 DATA "22","(t(1)=2)*(t(2)<>
4)*(n=0)","1","9+t(2)*16"
2084 DATA "22","(t(1)=2)*(t(2)<>
4)*(n<>0)","2","ed","78-n*4+t(2)
*16"
2085 DATA "99","","0"
2100 DATA "63JP","1","3","194+t(
1)*8","dl","dh"
2101 DATA "63JR","t(1)<4 AND ABS
 FN r()<128","2","32+t(1)*8","FN
 r()"
```

```
2102 DATA "22EX","(t(1)=4)*(t(2)
=4)","1","8"
2103 DATA "22EX","(t(1)=1)*(t(2)
=2)","1","235"
2104 DATA "52EX","(t(1)=3)*(t(2)
=2)","1","227"
2112 DATA "31BIT","vdef AND dl<8
","2","203","64+dl*8+t(2)"
2113 DATA "31RES","vdef AND dl<8
","2","203","128+dl*8+t(2)"
2114 DATA "31SET","vdef AND dl<8
","2","203","192+dl*8+t(2)"
2117 DATA "63CALL","1","3","196+
t(1)*8","dl","dh"
2120 DATA "14IN","vdef AND t(1)=
7","2","219","dl"
2121 DATA "41OUT","vdef AND t(2)
=7","2","211","dl"
2122 DATA "17IN","t(1)<>6","2","
ed","64+t(1)*8"
2123 DATA "71OUT","t(2)<>6","2",
"ed","65+t(2)*8"
2200 DATA "99","","0"                    Ende des Programms
5000 IF CODE s$(j) THEN PRINT #h         Symboltabelle
c''"Symbole:": FOR y=VAL "2" TO          aufführen
 VAL "CODE s$(1)+1": LET h=FN v(
s$(y)): GO SUB 9100: PRINT #hc;T
AB VAL "((y-2)*16+1)";s$(y, TO 6
);" ";h$;: NEXT y
5010 LET n=NOT j                         unaufgel. Tab. prüfen
5020 FOR t=ns+3 TO ns+2+u: LET a         vorhanden?
dd=FN v(s$(t)): LET c=PEEK add:
LET l$=s$(t): GO SUB 7600: IF v
THEN GO TO 5050
5030 IF NOT n THEN PRINT #hc''"U
nresolved:"
5040 LET n=n+j: LET h=add: GO SU         nein - anzeigen
B 9100: PRINT #hc;" ";l$;" ";h$:
 GO TO 5100
5050 RESTORE 9920: LET di=FN i(a         Typ bestimmen
dd+j): LET dl=FN l(di): LET di=V
AL "di-65536*(di>32767)"
5055 LET dl=VAL "dl-256*(dl>127)
"
5060 READ c$,a$,x$: IF NOT VAL c
$ THEN FOR x=j TO VAL x$: READ x
$: NEXT x: GO TO 5060
5070 FOR x=j TO VAL a$: READ y$:         Ausgabe umsetzen
```

```
 POKE FN v(s$(t))+VAL x$+x-j,VAL
  y$: NEXT x
5100 NEXT t
5200 IF er+n THEN PRINT #hc'er+n
;" Error(s)"
5210 GO TO 9999
7200 REM undefined
7210 LET u=u+j: IF u>nu THEN LET
  ec=VAL "98": RETURN
7220 LET s$(ns+u+j+j)=s$(ns+j+j,
  TO 6)+CHR$ FN l(add)+CHR$ FN h(
add)
7230 RETURN
7300 REM print
7310 IF ec THEN PRINT #hc;h$;" "
; INVERSE j;"ERROR ";ec; INVERSE
  NOT j;TAB 14;a$: RETURN
7320 IF lx THEN PRINT #hc;h$;TAB
  VAL "13";FN s$(j)
7330 IF m$="" THEN RETURN
7340 PRINT #hc;h$; INVERSE j;"*"
  AND (NOT vdef); INVERSE 0;TAB 5
;: FOR y=0 TO l+e-j: LET h=PEEK
(add+y): GO SUB 9120: PRINT #hc;
h$;: NEXT y
7350 PRINT #hc;TAB 14;m$;TAB 19;
FN s$(j+j+lx);"," AND 12;FN s$(3
+lx)
7360 RETURN
7400 REM check type
7405 LET x=LEN x$: IF NOT x THEN
  RETURN
7410 LET ix=(x$(j)="("): IF ix T
HEN LET x$=x$(2 TO LEN x$-j)
7415 GO SUB 7500
7420 RESTORE 9900+jump: READ n:
FOR x=0 TO n-j: READ c$: IF x$=c
$ THEN GO TO 7450
7425 NEXT x
7430 LET t$(t)=("3" AND (NOT ix)
)+("4" AND ix)
7440 GO SUB 8500: LET di=a+w: LE
T dh=FN h(di): LET dl=FN l(di):
RETURN
7450 LET t(t)=x: IF jump AND (NO
T ix) THEN LET t$(t)="6": RETURN

7460 LET t$(t)="1111111122222220
```

irgendwelche Fehler?

Detailzeile anzeigen

keine Mnemonik eingeg.

Operanden einstufen

```
70000005515011"(ix*15+x+j)
7470 IF x>=13 THEN LET e=j+ix-ju
mp: LET dd=VAL "dd+(dd=0)*(221+3
2*(x=14))": LET dis=w: LET x=10
7475 IF x>7 THEN LET t(t)=x-8
7480 IF ix AND t$(t)="1" THEN LE
T t(t)=6
7490 RETURN
7500 FOR x=j+j TO LEN x$: IF x$(
x)="+" OR x$(x)="-" THEN GO TO 7
550                                   Versetzung berechnen
7510 NEXT x
7520 LET w=NOT j: RETURN
7550 LET z$=x$( TO x-j): LET x$=
x$(x TO ): GO SUB 8900: LET x$=z
$
7560 LET w=x
7570 RETURN
7600 REM find symbol
7610 LET v=j                          als 'gefunden' setzen
7620 FOR s=2 TO CODE s$(j)+j: IF
 1$=s$(s, TO 6) THEN LET a=FN v(
s$(s)): RETURN
7630 NEXT s: LET v=0                  nicht gefunden
7640 RETURN
7700 GO TO 9000
7900 CLS : PRINT AT VAL "10",VAL
 "13";"WARTEN": MERGE "ZXASM/4
8K"
7910 RETURN
8300 REM Find op (1)
8310 RESTORE i: LET x=0
8320 READ x$: IF x$="99" THEN LE      Tabellen abtasten
T x=0: RETURN
8330 LET x=x+j: IF m$=x$ THEN RE      Rückspr., wenn
TURN                                  gefunden
8340 FOR y=j TO 1: READ x$: NEXT      zum nächsten
 y                                    Posten
8350 GO TO 8320
8400 REM Find op (2)
8410 READ x$,c$,y$: LET z=VAL y$      Tabellen abtasten
: IF x$="99" THEN GO TO 8450
8420 IF x$=n$ THEN LET x=VAL c$:      Rückspr., wenn
 IF x THEN LET l=z: RETURN            gefunden
8430 FOR y=j TO z: READ x$: NEXT      zum nächsten
 y                                    Posten
8440 GO TO 8410
8450 LET x=0: IF NOT z THEN RETU
```

```
RN
8460 RESTORE z: GO TO 8410
8500 REM Convert type                    Adresse bestimmen
8505 LET a=0
8510 GO SUB 8900: IF v THEN LET          prüfen, ob dezimal
a=x: LET ec=VAL "6*(x>65535 OR x
<-32768)": RETURN
8520 IF FN n(x$(j)) AND x$(LEN x         prüfen, ob Hex.
$)="H" THEN FOR x=j TO LEN x$-j:
 LET a=VAL "a*16+CODE x$(x)-48-7
*(x$(x)>""9"")": NEXT x: RETURN
8530 LET l$=x$: GO SUB 7600: IF          Symbole abtasten
v THEN RETURN
8560 LET ec=VAL "6*(FN a(x$(1))=
0)": IF ec THEN RETURN
8570 LET s$(ns+2)=x$: LET vdef=N         Flagge unaufgelöst
OT j
8580 RETURN
8900 REM vet numeric
8910 LET z=j+VAL "((x$(1)=""+"")
 OR (x$(1)=""-"")) AND LEN x$>1"
8920 FOR n=z TO LEN x$: IF FN n(
x$(n)) THEN NEXT n
8930 LET v=(n>LEN x$): IF v THEN
 LET x=VAL x$( TO n-j)
8940 RETURN
9000 REM kybrd
9010 INPUT (m$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=j TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
")): NEXT x
9030 RETURN
9100 REM h$=hex$(h)
9110 LET h$="    ": GO TO 9130           4 Leerräume
9120 LET h$="  "                         2 Leerräume
9130 LET h1=h: FOR x=LEN h$ TO j
 STEP -j: LET x1=h1-INT (h1/16)*
16: LET h$(x)=CHR$ (x1+CODE "0"+
7*(x1>9)): LET h1=INT (h1/16): N
EXT x
9140 RETURN
9900 DATA 15,"B","C","D","E","H"
,"L","M","A","BC","DE","HL","SP"
,"AF","IX","IY"
9901 DATA 15,"NZ","Z","NC","C","
PO","PE","P","M","","","HL","","
","IX","IY"
```

```
9920 DATA "(c-INT (c/8)*8)+INT (
c/64)=0","1","1","dl+FN l(a-add-
2)"
9921 DATA "c=ed OR c=221 OR c=25
3","2","2","FN l(a+di)","FN h(a+
di)"
9922 DATA "1","2","1","FN l(a+di
)","FN h(a+di)"
```

```
1002 REM ZXASM 48K extension
1010 LET ns=64: LET nu=64
1015 CLS : PRINT "*** ZX Spectr
um Assembler ***": PRINT '"RAM
 verfuegbar: ";(PEEK 23733-63)/4
;"K"'"RAMTOP  : ";: LET h=FN i(2
3730): GO SUB 9100: PRINT h;" ("
;h$;"H)"
1120 LET m$="Pruefen?": GO SUB 9
000: LET auto=(CHR$ CODE a$="J")
: IF auto THEN LET m$="HINZULADE
N?": GO SUB 9000: IF CHR$ CODE a
$="J" THEN INPUT "Programmname?:
"; LINE a$: CLS : PRINT AT 8,2;"
REKORDER STARTEN, TASTE DRUECKEN
": PAUSE 4e4: CLS : MERGE a$
1125 IF auto THEN GO SUB 7800
7700 IF NOT auto THEN GO TO 9000
7710 LET end=0: IF FN i(auto-1)=
10730 THEN LET end=1: RETURN
7720 LET a$=""
7730 IF PEEK auto=13 THEN GO TO
7760
7735 IF PEEK auto=33 THEN LET au
to=auto+1: GO TO 7770
7740 LET a$=a$+CHR$ PEEK auto
7750 LET auto=auto+1: GO TO 7730
7760 LET auto=auto+6
7770 IF NOT LEN a$ THEN GO TO 77
10
7780 GO TO 9020
7800 REM Auto start
7810 PRINT AT 10,10;"Suche..."
7820 LET x=FN i(23635): LET y=FN
 i(23627)
7830 IF x>=y THEN LET auto=0: RE
TURN
```

Tabellen erhöhen

Autoscan zulassen

auf nächste Zeile abt.

nach erst. Zeile abt.

```
7840 IF PEEK (x+4)=234 THEN GO T
O 7860
7850 LET x=x+4+FN i(x+2): GO TO
7830
7860 IF PEEK (x+5)<>40 THEN GO T
O 7850
7870 LET auto=x+6
7880 RETURN
7900 REM This allows reruns
```

ZXDISASM – Symbolischer Disassemblierer

Sie wollen also einen heimlichen Blick in das Sinclair-ROM werfen? Mit diesem Disassembler können Sie jedes Maschinencode-Programm in seine mnemotechnische Form zurückverwandeln. Aber das ist noch nicht alles – Sie können damit auch eine Tabelle von Namen (oder **Symbolen**) belegen, so daß jedesmal, wenn ein Sprung-, Aufruf- oder Datenbezugnahmebefehl vorkommt, der entsprechende Symbolname dafür eingesetzt wird. Dadurch läßt sich Ihr Disassembler-Listing viel leichter lesen. Sie brauchen im Kapitel 'Systemvariable' des Sinclair-Handbuchs nicht mehr herumzublättern!

Wenn Sie die Absicht haben, selbst Assemblerprogramme zu schreiben und dazu den Assemblierer in diesem Buch verwenden, wird die Symboltabellen-Einrichtung dieses Programms Ihnen dabei eine große Hilfe sein.

Das Programm läuft in einem normalen 16K-Spectrum, allerdings muß die Symboltabelle ein wenig verkleinert werden. In der Praxis wäre es klug, die Befehle und alle REM-Anweisungen bei einem 16K-Gerät wegzulassen, damit für Symbole und Ihre Routinen über RAMTOP möglichst viel Platz ist.

Zusätzlich finden Sie im Anschluß an diesen Abschnitt ein Maschinencode-Überwachungsprogramm. Damit können Sie sich den Speicher ansehen, Bytes verändern und über das Keyboard kleine Routinen in Hex eingeben. Diese beiden Programme sind so angelegt, daß sie mit dem MERGE-Befehl gemischt werden können, so daß Leute, die mehr als 16K haben, die Programme zu einem leistungsfähigen Debugginginstrument vereinigen können. Wenn Sie nur 16K RAM haben, müssen Sie die beiden Programme getrennt halten, aber nützlich bleibt das trotzdem. Der Umstieg von einer Aufgabe zur anderen geht nur etwas langsamer.

ZXDISASM in Betrieb

Laden Sie das Programm mit LOAD "ZXDISASM". Wenn es geladen ist, sehen Sie die Frage:

Hard copy?

Wenn Sie 'ja' antworten, werden alle Ausgaben an den Drucker geschickt, sonst erscheinen sie auf dem Bildschirm. (Sie können die Ergebnisse trotzdem drucken lassen, wenn Sie die Frage 'SCROLL?' jedesmal mit COPY beantworten.)

Die Antwort 'nein' oder Druck auf 'ENTER' schickt alle Ausgaben an den Fernsehbildschirm.

Als nächstes werden Sie gefragt:

Enter symbol names? (Symbolnamen eingeben?)

Wenn Sie mit 'nein' antworten oder 'ENTER' drücken, wird die Disassemblierung durch Eingabe der Startadresse fortgesetzt. Siehe unten.

Die Antwort 'ja' bietet Ihnen die Wahl, Ihre Symbolnamen auf zweierlei verschiedene Art einzugeben:

Load symbols from tape? (Symbole von Tabelle laden?)

Die Antwort 'ja' zwingt ZXDISASM, den Namen der Banddatei zu verlangen, zu warten, bis Sie das Band laufen lassen, und die entsprechende Symboltabelle hineinzumischen.

Wenn Sie 'nein' antworten, fordert ZXDISASM Sie auf, die Namen der Symbole über das Keyboard einzugeben. Nach Eingabe eines Symbolnamens werden Sie aufgefordert, die entsprechende Adresse des Symbols einzugeben. Adressen können eingegeben werden als Dezimalzahl oder als Hexzahl (mit angehängtem 'H' - Beispiele: 1234H, 1AE3H). Das setzt sich fort, bis Sie nach der Anforderung eines Symbolnamens 'ENTER' drücken. Dann werden Sie gefragt:

Save symbols on tape? (Symbole auf Band sichern?)

Wenn Sie 'ja' antworten, erhalten Sie die Aufforderung, den Bandnamen einzugeben, den ZXDISASM verwenden soll, dann wird die Tabelle gesichert. Anschließend wird die Disassemblierung fortgesetzt. Die gesicherte Tabelle kann später mit der Antwort 'ja' auf die vorherige Frage in ZXDISASM geladen werden. Wenn Sie den Assemblierer verwenden, können Sie außerdem die Symboltabelle verwenden, die von diesem Programm ausgegeben wird - siehe den Abschnitt 'ZXASM - ein symbolischer Assemblierer'.

In diesem Stadium wird die Symboltabelle in Adressenfolge sortiert. Das erlaubt schnelleren Zugang zu den entsprechenden Namen.

Schließlich muß ZXDISASM wissen, wo im Speicher Sie mit der Disassemblierung beginnen wollen. Die Frage lautet:

Enter start adress:

Wenn Sie nur 'ENTER' drücken, schließt ZXDISASM ab. Im anderen Fall ist eine der folgenden Eingaben möglich:

a) Eine Dezimalzahl.
b) Eine Hexzahl mit angehängtem 'H' (etwa 22FFH oder 5DH). Beachten Sie, daß die erste Ziffer eine Zahl sein muß, etwa 0FF5AH.
c) Ein Symbolname. In diesem Fall wird die entsprechende Adresse verwendet.

ZXDISASM Listing

Geben Sie das Programm so ein, wie es unten aufgeführt ist, und sichern Sie es mit dem Namen "ZXDISASM".

Wenn Sie mehr als 16K RAM haben, sollten Sie Zeile 10 abändern, um die Höchstgröße der zugelassenen Symboltabelle zu ändern. Die Normalgröße beträgt 20, und Sie sollten das auf 64 erhöhen.

Wenn Sie nur 16K haben, geben Sie die Zeilen 7100–7180 nicht ein (die Befehle) und lassen Sie auch die Zeilen weg, die nur REM-Anweisungen enthalten. Das Programm paßt auch dann hinein, wenn Sie die Befehle und REM-Anweisungen belassen, aber der über RAMTOP verfügbare Platz für große Programme in Maschinencode verringert sich.

```
    1 REM Spectrum Disassembler
    2 REM @1982 Phipps Associates
   10 LET ns=10: DIM s$(ns+1,8):
LET s$(1)=CHR$ 0
   20 GO TO 7500          prüfen, ob ZXMCMON gemischt
   30 DEF FN s(x)=CODE s$(x,7)+25
6*CODE s$(x,8)
  100 GO SUB 7000                    Anweisungen
  110 LET a$="ENTER zur Fortsetzu
ng": GO SUB 9000
  120 IF CODE s$(1) THEN GO TO 20    übergehen, wenn
0                                    Symbole geladen
  130 LET a$="Eingabe der Symbole
? (J/N)": GO SUB 9000: IF FN y()
 THEN GO SUB 7200: GO SUB 8300
```

```
 200 LET a$="Hard copy (J/N)?":
GO SUB 9000: LET hc=VAL "2"+FN y
()
 210 CLS
 300 IF LEN INKEY$ THEN GO TO 30                keine Taste gedrückt?
0
 305 LET a$="Startaddresse einge
ben:": GO SUB 9000: IF end THEN
GO TO 20
 310 GO SUB 8500: IF a<0 THEN GO                in Adresse umwandeln
 TO 305
 320 IF LEN INKEY$ THEN GO TO 32                keine Taste gedrückt?
0
 330 PRINT #hc;"Addr Hex          Op            Negativschrift nehmen
   Operand/Notiz"''
 340 FOR x=2 TO CODE s$(1)+1: LE                findet Symbolanfang
T xa=FN s(x): IF a>xa THEN NEXT
x: LET xa=10e30
 350 LET xs=x
 400 REM main loop
 405 GO SUB 8000                                Vorgaben neu setzen
 410 IF LEN INKEY$ THEN GO TO 30                prüf., ob Halt erforderl.
0
 415 IF a>=xa THEN PRINT #hc;TAB                notfalls Etikett
 13;s$(xs, TO 6);":": LET xs=xs+                anzeigen
1: LET xa=1e31: IF xs<=CODE s$(1
)+1 THEN LET xa=FN s(xs): GO TO
415
 420 LET c=PEEK a: IF c=DD OR c=                Opcode holen –
FD THEN GO SUB 8100                             prüfen auf IX/IY
 425 LET p$="BCDE"+e$+"AF": LET                 Registernamen
q$="BCDE"+e$+"SP": LET r$="BCDEH                setzen
LMA"
 430 IF c=CB THEN GO SUB 5000: G                Bit-Codes
O TO 500
 440 IF c=ED THEN GO SUB 6000: G                erweitertes Codes
O TO 500
 450 GO SUB 8200                                Opcode teilen
 460 GO SUB 1000+g*1000                         zu relevant. Abschnitt
                                                 gehen
 500 REM print
 510 LET h=a: GO SUB 9100: PRINT                Adresse anzeigen
 #hc;h$;" ";
 520 FOR n=1 TO l: LET h=PEEK (a                Hex. anzeigen
+n-1): GO SUB 9120: PRINT #hc;h$
;: NEXT n
 530 PRINT #hc;TAB 14;j$;TAB 19;                Opcode anzeigen
k$;"," AND l$<>"";l$;" " AND n$<
>"";n$
```

```
 900 LET a=a+1
 910 GO TO 400
1000 REM group 00
1010 GO TO VAL "10201100120013001400140016001700"(t2*4+1 TO t2*4+4)
1020 LET i=9900: GO SUB 7300: IF t1<2 THEN RETURN
1050 LET l=l+1: LET h=a+2+FN p(a+1): GO SUB 9100: GO SUB 8450: LET n$=h$
1060 RETURN
1100 IF t3 THEN LET j$="ADD": LET k$=e$: LET l$=FN p$(q$): RETURN
1120 LET j$="LD": LET k$=FN p$(q$): LET h=a+1+e: GO SUB 8400: LET l$=h$: RETURN
1200 LET i=9905: GO SUB 7300: IF t1<4 THEN RETURN
1220 LET h=a+1+e: GO SUB 8400
1230 IF t3 THEN LET l$=VAL$ l$: RETURN
1240 LET k$=VAL$ k$
1250 RETURN
1300 LET j$=("DEC" AND t3)+("INC" AND (NOT t3)): LET k$=FN p$(q$): RETURN
1400 LET j$=("DEC" AND t2=5)+("INC" AND t2=4)
1420 LET k$=FN r$(t1): LET l=l+e
1430 RETURN
1600 LET j$="LD": LET k$=FN r$(t1): LET h=a+1+e*2: GO SUB 8600: LET l=l+e: LET l$=h$: RETURN
1700 LET i=9910: GO SUB 7300
1710 RETURN
2000 REM group 01
2010 IF t1=6 AND t2=6 THEN LET j$="HALT": RETURN
2020 LET j$="LD"
2030 LET k$=FN r$(t1): LET l$=FN r$(t2)
2040 LET l=l+e
2050 RETURN
3000 REM group 10
3010 LET k$=FN r$(t2): LET l=l+e
```

Adresse aktualisieren
nächster Befehl

```
3100 LET i=9915: GO SUB 7300
3105 IF t1<4 AND t1<>2 THEN LET
l$=k$: LET k$="A"
3110 RETURN
4000 REM group 11
4010 GO TO 4100+t2*100
4020 LET j$="??": RETURN
4100 LET j$="RET": GO SUB 7400:
RETURN
4200 IF t3 THEN LET i=9920: GO S
UB 7300: RETURN
4210 LET j$="POP": LET k$=FN p$(
p$): RETURN
4300 LET j$="JP": GO SUB 7400: L
ET h=a+1: GO SUB 8400: LET l$=h$
: RETURN
4400 IF NOT t1 THEN LET h=a+1: G
O SUB 8400
4410 IF t1=2 OR t1=3 THEN LET h=
PEEK (a+1): GO SUB 9120: LET l=l
+1
4420 LET i=9925: GO SUB 7300: RE
TURN
4500 LET j$="CALL": GO SUB 7400:
 LET h=a+1: GO SUB 8400: LET l$=
h$: RETURN
4600 IF NOT t3 THEN LET j$="PUSH
": LET k$=FN p$(p$): RETURN
4610 IF t1<>1 THEN GO TO 4020
4620 LET j$="CALL": LET h=a+1: G
O SUB 8400: LET k$=h$: RETURN
4700 LET h=PEEK (a+1): GO SUB 91
20: LET k$=h$: LET l=l+1: GO TO
3100
4800 LET j$="RST": LET h=t1*8: G
O SUB 9120: LET k$=h$
4810 RETURN
5000 REM CB group
5010 LET l=l+1+e: LET c=PEEK (a+
1+e*2): GO SUB 8200
5100 IF NOT g THEN LET i=9930: G
O SUB 7300: LET k$=FN r$(t2): RE
TURN
5110 LET j$="BITRESSET"((g-1)*3+
1 TO (g-1)*3+3)
5120 LET k$=CHR$ (t1+CODE "0")
5130 LET l$=FN r$(t2)
```

```
5140 RETURN
6000 REM ED group
6010 LET l=l+1: LET c=PEEK (a+1)
: GO SUB 8200
6020 IF g=2 THEN GO TO 6300
6030 IF g<>1 THEN LET j$="??": R
ETURN
6040 GO TO VAL "6050606060706100
6200621062206230"(t2*4+1 TO t2*4
+4)
6050 LET j$="IN": LET l$="(C)":
LET k$=FN r$(t1): RETURN
6060 LET j$="OUT": LET k$="(C)":
 LET l$=FN r$(t1): RETURN
6070 LET j$=("ADC" AND t3)+("SBC
" AND (NOT t3)): LET k$=e$: LET
l$=FN p$(q$): RETURN
6100 LET j$="LD": LET h=a+2: GO
SUB 8400: LET h$="("+h$+")"
6120 LET k$=FN p$(q$): LET l$=h$
6130 IF NOT t3 THEN LET l$=k$: L
ET k$=h$
6140 RETURN
6200 LET j$="NEG": RETURN
6210 LET j$=("RETI" AND t1)+("RE
TN" AND (NOT t1)): RETURN
6220 LET j$="IM": LET k$=CHR$ (t
1+CODE "0"): RETURN
6230 LET i=9935: GO SUB 7300
6240 RETURN
6300 RESTORE 9940+t2*10
6310 FOR x=1 TO t1-3: READ j$: N
EXT x
6320 RETURN
7000 REM initialise
7005 DEF FN r$(x)=(r$(x+1) AND r
$(x+1)<>"M")+(FN i$(a+1+e) AND r
$(x+1)="M")
7010 DEF FN p(x)=PEEK x-(256*(PE
EK x>127))
7015 DEF FN b$(x)=("+" AND FN p(
x)>=0)+STR$ FN p(x)
7020 DEF FN i$(x)="("+e$+(FN b$(
x) AND e)+")"
7025 DEF FN p$(x$)=x$(rp*2+1 TO
rp*2+2)
7030 DEF FN y()=CHR$ CODE a$="J"
```

def. Funktionen

```
7035 DEF FN n(x$)=(x$>="0" AND x
$<="9")
7040 LET p$="BCDEHLAF": LET q$="
BCDEHLSP": LET r$="BCDEHLMA"
7045 LET DD=VAL "221": LET FD=VA
L "253"
7050 LET ED=VAL "237": LET CB=VA
L "203"
7070 LET ramtop=1+PEEK 23730+256
*PEEK 23731
7100 REM instructions
7110 CLS : PRINT "*** ZX Spectru
m Disassembler ***"
7120 PRINT '"Version 2.2 27. Aug
ust 1982"
7130 PRINT '"Alle Werte werden i
n Hexadezimalangezeigt, ausser S
onderzeichen,die in dezimal darg
estellt wer- den (z.B. JR +19 od
er              LD (IY-8),FF)."
7140 PRINT '"Die Eingabe kann in
 ""hex"" (z.B. 43a2h) oder ""dez
imal"" oder in  einem System va
riabler Bezeich- nungen eingegeb
en werden: so wieRAMTOP."
7190 RETURN
7200 REM enter symbol table
7210 LET a$="Laden der Symbol vo
n Band?": GO SUB 9000: IF FN y()
 THEN LET a$="Name:": GO SUB 900
0: LOAD a$ DATA s$(): RETURN
7220 FOR s=2 TO ns+1
7230 LET a$="Symbolnamen Eingebe
n:": GO SUB 9000: IF end THEN LE
T s=s-2: GO TO 7280
7240 LET s$(s, TO 6)=a$
7250 LET a$="Adresse Eingeben:":
 GO SUB 9000: IF end THEN GO TO
7250
7255 GO SUB 8500: IF a<0 THEN GO
 TO 7250
7260 LET s$(s,7 TO )=CHR$ (a-INT
 (a/256)*256)+CHR$ (INT (a/256))
7270 PRINT s$(s, TO 6),a
7275 NEXT s: LET s=ns
7280 LET s$(1)=CHR$ s
7290 LET a$="Speichern der Symbo
le auf Band?": GO SUB 9000: IF F
```

Registernamen

Spezial-Opcodes

derzeitiger RAMTOP

von Keyboard laden

Bestätigung anzeigen

Zählung beibehalten

```
N y() THEN LET a$="Name:": GO SU
B 9000: SAVE a$ DATA s$()
7295 RETURN
7300 REM create operand
7310 RESTORE i: READ n
7320 FOR x=1 TO t1+1
7330 READ j$: IF n=3 THEN READ k
$,l$
7340 NEXT x
7390 RETURN
7400 REM condition codes          verw. bei JP/JR/CALL/RET
7410 RESTORE 9980: FOR x=1 TO t1
+1: READ k$: NEXT x
7420 RETURN
7500 REM Machine Monitor          ZXMCMON-Schnittstelle
7501 GO TO 100: REM not present
8000 REM Reset defaults
8010 LET e$="HL"                  geänd. durch IX/I-Codes
8020 LET e=0: LET l=1
8030 LET J$="": LET K$="": LET l
$="": LET h$="": LET n$=""
8040 RETURN
8100 REM DD/FD opcode             Indexregister-Codes
8110 LET e$=("IX" AND c=DD)+("IY
" AND c=FD)
8120 LET e=1: LET l=l+1
8130 LET c=PEEK (a+1)
8140 RETURN
8200 REM Split opcode             Gruppencode etc. bestimmen
8210 LET g=INT (c/64): LET t1=IN
T (c/8)-g*8: LET t2=c-t1*8-g*64
8220 LET rp=INT (t1/2): LET t3=(
rp*2<>t1)
8230 RETURN
8300 REM Sort symbols
8310 PRINT AT 21,0;"Sortiere..."
8320 FOR x=2 TO CODE s$(1)+1: LE
T z=1: FOR y=2 TO CODE s$(1)-x+2
8330 IF FN s(y)>FN s(y+1) THEN L
ET x$=s$(y): LET s$(y)=s$(y+1):
LET s$(y+1)=x$: LET z=0
8340 NEXT y: IF z THEN RETURN
8350 NEXT x
8360 RETURN
8400 REM Get address in h$        16 Bit-Daten holen
8410 LET h=PEEK h+256*PEEK (h+1)
: GO SUB 9100
8420 LET l=l+2
```

```
8450 IF NOT CODE s$(1) THEN RETU
RN
8460 FOR x=2 TO CODE s$(1)+1
8470 IF h>FN s(x) THEN NEXT x: R
ETURN
8480 IF h=FN s(x) THEN LET h$=s$
(x, TO 6)
8490 RETURN
8500 REM Convert a$ to address
8510 IF a$(LEN a$)="H" AND FN n(
a$(1)) THEN LET a=0: FOR x=1 TO
LEN a$-1: LET a=a*16+CODE a$(x)-
CODE "0"-7*(a$(x)>"9"): NEXT x:
RETURN
8520 FOR x=2 TO CODE s$(1)+1: IF
 a$=s$(x, TO LEN a$*(LEN a$<7))
THEN LET a=CODE s$(x,7)+256*CODE
 s$(x,8): RETURN
8530 NEXT x
8540 LET a=-1: FOR x=1 TO LEN a$
: IF FN n(a$(x)) THEN NEXT x: LE
T a=VAL a$
8550 RETURN
8600 REM Byte value
8610 LET h=PEEK h: GO SUB 9120:
LET l=l+1
8620 RETURN
9000 REM line input
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
")): NEXT x
9030 RETURN
9100 REM h$=hex$(h)
9110 LET h$="    ": GO TO 9130
9120 LET h$="  ": REM byte only
9130 LET h1=h: FOR x=LEN h$ TO 1
 STEP -1: LET x1=h1-INT (h1/16)*
16: LET h$(x)=CHR$ (x1+48+7*(x1>
9)): LET h1=INT (h1/16): NEXT x
9140 RETURN
9900 DATA 3,"NOP","","","EX","AF
","AF","DJNZ",FN b$(a+1),"","JR"
,FN B$(a+1),"","JR","NZ",FN B$(a
+1),"JR","Z",FN B$(a+1),"JR","NC
",FN B$(a+1),"JR","C",FN B$(a+1)
9905 DATA 3,"LD","(BC)","A","LD"
```

Symboltabelle prüfen

über Adresse hinaus?

Symbolnamen holen

ist es Hex.?

ist es ein Symbol?

als dezimal behandeln

8 Bit-Daten holen

```
,"A","(BC)","LD","(DE)","A","LD"
,"A","(DE)","LD","""(""+H$+"")""
","HL","LD","HL","""(""+H$+"")""
","LD","""(""+h$+"")""","A","LD"
,"A","""(""+h$+"")"""
9910 DATA 1,"RLCA","RRCA","RLA",
"RRA","DAA","CPL","SCF","CCF"
9915 DATA 1,"ADD","ADC","SUB","S
BC","AND","XOR","OR","CP"
9920 DATA 3,"??","","","RET","",
"","??","","","EXX","","","??","
","","JP","("+E$+")","","??","",
"","LD","SP",E$
9925 DATA 3,"JP",H$,"","??","","
","OUT","("+H$+")","A","IN","A",
"("+H$+")","EX",E$,"(SP)","EX","
DE",E$,"DI","","","EI","",""
9930 DATA 1,"RLC","RRC","RL","RR
","SLA","SRA","??","SRL"
9935 DATA 3,"LD","I","A","LD","R
","A","LD","A","I","LD","A","R",
"RRD","","","RLD","",""
9940 DATA "LDI","LDD","LDIR","LD
DR"
9950 DATA "CPI","CPD","CPIR","CP
DR"
9960 DATA "INI","IND","INIR","IN
DR"
9970 DATA "OUTI","OUTD","OTIR","
OTDR"
9980 DATA "NZ","Z","NC","C","PO"
,"PE","P","M"
9999 STOP
```

ZXCMON – Maschinencode-Überwachungsprogramm

Wie im vorigen Abschnitt versprochen, hier ein Maschinencode-Überwachungsprogramm (Monitor) für den Spectrum. Damit können Sie Speicherbereiche in Hex anzeigen, einzelne Bytes anzeigen und über das Keyboard kleine Maschinencode-Routinen in Hex eingeben.

Wenn Sie die Absicht haben, Maschinencode-Programmierung in großem Maßstab zu betreiben, wäre es offenkundig klüger, den Assemblierer zu verwenden – das geht viel einfacher! Bei bestimmten Gelegenheiten ergibt es jedoch mehr Sinn, einfach ein paar Befehle über das Keyboard 'einzutippen', während Sie etwas testen.

Das Programm soll voll und ganz in Übereinstimmung mit dem Disassemblierer verwendet werden. Alle Zeilennummern sind so erarbeitet, daß das kombinierte Programm ein kombiniertes Bildschirmmenü einschließt, außerdem kann jedes Programm das Vorhandensein des anderen prüfen. Aus diesem Grund empfiehlt sich, keine der Zeilennummern im Programm zu ändern, weil die Kombination sonst nicht richtig funktioniert.

ZXMCMON in Betrieb

Wenn das Programm läuft, wird Ihnen ein Menü mit folgenden Möglichkeiten geboten:

(1) Speicher anzeigen
(2) Speicher ändern
(3) Maschinencode in Hex eingeben
(4) Speicher disassemblieren

Die letze Option erscheint nur, wenn das Programm mit ZXDISASM kombiniert worden ist.

Sie geben die entsprechende Nummer ein und drücken 'ENTER'.

Wenn keine Nummer eingegeben wird, schließt ZXMCMON ab.

Die Wahlmöglichkeiten der Reihe nach:

1. **Speicher anzeigen**

Sie sind aufgefordert, eine Adresse einzugeben, die in Dezimal oder Hex sein kann (die üblichen Regeln – der Zahl ein 'H' anfügen). Wenn keine Adresse eingegeben wird, erscheint noch einmal das Hauptmenü. Der Speicher wird in Hex-Bytepaaren angezeigt, pro Zeile acht Bytes mit der Adresse des ersten Bytes am Zeilenanfang. Das setzt sich fort, bis Sie irgendeine Taste drücken. ZXMCMON fordert dann eine neue Adresse an. Eine Druckerkopie erhält man mit dem Befehl COPY, mit CONT geht es weiter.

2. **Speicher ändern**

Anfangs werden Sie aufgefordert, die Startadresse einzugeben. Wird keine eingegeben, erscheint das Hauptmenü. Sonst zeigt ZXMCMON die laufende Adresse und den Inhalt des Bytes an dieser Adresse. Beispiel:

7FD8 21-

Das heißt, die Adresse 7FD8 (hex) enthält den Wert 21 (hex). Sie haben jetzt drei Möglichkeiten. Wenn Sie ein Hexzahlenpaar eingeben und 'ENTER' drücken, ersetzt dieser Wert den vorher gezeigten. Dann wird der Wert in der nächsten Adresse gezeigt. Wenn Sie nur 'ENTER' drücken, bleibt der Wert unberührt, und der Wert der nächsten Adresse erscheint. Geben Sie aber einen Punkt ein, ".", so fordert ZXMCMON zur Eingabe einer neuen Adresse auf. Hier ein Beispiel:

3	**Enter start adress:** 3260Ø	
	7FD8 ØØ-	(das belassen)
	7FD9 21-ØC	(21 zu ØC verändert)
	7FDA FF-FE	(FF zu FE verändert)
	7FDB FF-.	(neue Adresse anford.)
	Enter start address:	(etc., etc. ...)

3. **Eingabe Hex-Maschinencode**

Wie vorher schon erwähnt, ist das lediglich ein 'Notbehelf', damit während dem Testen von Routinen kleine Veränderungen vorgenommen werden können. Dann werden Sie aufgefordert, eine Startadresse einzugeben, wird 'ENTER' allein gedrückt, erhalten Sie wieder das Hauptmenü. Sonst kann die Adresse (wie oben) in Dezimal oder Hex sein. Als nächstes können Sie einen String von Hexzeichenpaaren eingeben. Sie werden in Binärzahlen umgewandelt und an der gewünschten Adresse eingefügt.

Das Display zeigt die laufende Adresse. Nachdem dieser umgewandelt und eingegeben ist, verlangt ZXMCMON einen weiteren String. Das setzt sich fort, bis Sie nur 'ENTER' drücken. Der String, den Sie eingeben, kann jede beliebige Länge haben (beachten Sie aber, daß alle entdeckten Fehler dazu führen, daß der **Rest** des Strings nicht beachtet wird, also Vorsicht!). Fehler können hervorgerufen werden durch a) eine ungerade Zahl eingegebener Zeichen - Sie müssen Hexzeichen-**Paare** eingeben - dann bleibt der ganze String unbeachtet, und b) ein unzulässiges Hexpaar (beispielsweise ist 9G nicht Hex), worauf der **Rest** des Strings unbeachtet bleibt. Behalten Sie die Adresse im Auge.

4. **Speicher disassemblieren**

Im vorigen Abschnitt ist diese Möglichkeit ausführlich besprochen worden. Wenn ZXDISASM abschließt, springt das Programm automatisch zurück zum Maschinencode-Monitor-Menü.

ZXMCMON Listing

Hier gibt es nicht mehr viel zu sagen, außer, daß Leute, die dieses Programm mit dem Disassemblierer kombinieren wollen, zuerst den Disassembler (ZXDISASM) laden sollten, um dann mit MERGE "ZXMCMON" hineinzumischen und die Kombination auf einem eigenen Band zu sichern. Wegen dieser Eigenschaft sehen manche Befehle im Programmlisting vielleicht ein wenig merkwürdig aus, aber keine Sorge, das fügt sich alles zusammen.

```
7500 REM Machine Code Monitor
7501 REM ©1982 Phipps Associates
7505 LET e$="": GO SUB 8000: LET Prüfung auf ZXDISASM
 d=(LEN e$<>0)
7510 LET ramtop=PEEK 23730+256*P Ramtop-Wert setzen
EEK 23731+1: IF NOT d THEN LET s
$=CHR$ 0
7520 CLS : PRINT TAB 10;"Monitor
 Menue"''" (1) Ausdruck 'HEX' au
f Display"'" (2) Eingabe Masch.-
Code 'HEX'"'" (3) Aenderungen"
7530 IF d THEN PRINT " (4) Disas
semblieren"
```

```
7540 PRINT ''"'ENTER' allein bee
ndet das        Programm"
7550 LET a$="Eingabe der Wahl:":
 GO SUB 9000: IF end THEN GO TO
9999
7560 IF VAL a$<1 OR VAL a$>3+d T
HEN GO TO 7550
7570 CLS : GO TO VAL "7600770078
000100"((VAL a$-1)*4+1 TO (VAL a
$-1)*4+4)
7600 REM              Display memory
7610 IF LEN INKEY$ THEN GO TO 76
10
7620 GO SUB 8700: IF end THEN GO
 TO 7500
7630 FOR a=a TO a+65535 STEP 8:
LET h=a: GO SUB 9100: PRINT h$;"
 ";
7640 FOR b=a TO a+7: LET h=PEEK
b: GO SUB 9120: PRINT h$;" ";: N
EXT b
7650 PRINT : IF LEN INKEY$ THEN
GO TO 7600
7660 NEXT a: GO TO 7500
7700 REM        Enter machine code
7705 IF LEN INKEY$ THEN GO TO 77
05
7710 GO SUB 8700: IF end THEN GO
 TO 7500
7715 PRINT AT 10,0;"Adresse: ";a
;" (";: LET h=a: GO SUB 9100: PR
INT h$;")"
7720 LET a$="Eingabe in Hex:": G
O SUB 9000: IF end THEN GO TO 75
00
7725 PRINT AT 21,0,,
7730 IF LEN a$<>INT (LEN a$/2)*2
 THEN PRINT AT 21,0; FLASH 1;"UN
GUELTIGE LAENGE"; FLASH 0;"-wied
erholen": GO TO 7720
7735 FOR x=1 TO LEN a$-1 STEP 2:
 GO SUB 7900
7740 IF x1*16+x2>255 THEN PRINT
AT 21,0; FLASH 1;"EINGABE FEHLER
"; FLASH 0;"-wiederholen": GO TO
 7715
7745 POKE a,x1*16+x2
7750 LET a=a+1
```

zu Routine springen

ein Leerraum

8 Bytes je Reihe

auf Tastendruck achten

klären, ob keine Taste gedrückt ist

Adresse holen

Hexstring holen

Fehlerzeile löschen

in Binär umwandeln

Code in RAM setzen

nächste Adresse

```
7755 NEXT x
7760 GO TO 7715                          neuen String abwarten
7800 REM                  Alter memory
7810 IF LEN INKEY$ THEN GO TO 78         warten, ob kein
10                                       Tastendruck
7820 GO SUB 8700: IF end THEN GO         Adresse holen
 TO 7500
7830 LET h=a: GO SUB 9100: LET x
$=h$: LET h=PEEK a: GO SUB 9120
7840 LET a$=x$+" "+h$+"-": GO SU
B 9000: IF end THEN GO TO 7880
7850 IF a$="." THEN GO TO 7800           prüfen, ob fertig
7860 IF LEN a$>2 THEN LET a$=a$(         Eingabestring prüfen
 TO 2)
7870 LET x=1: GO SUB 7900: POKE          neuen Wert einfügen
a,x1*16+x2
7880 PRINT x$;" ";h$;"-";a$              Bestätigung anzeigen
7890 LET a=a+1: GO TO 7830               nächste Adresse
7900 REM Convert a$(x) to binary
7910 LET x1=CODE a$(x)-CODE "0"-
7*(a$(x)>"9")
7920 LET x2=CODE a$(x+1)-CODE "0
"-7*(a$(x+1)>"9")
7930 RETURN
8090 RETURN : REM no disassembl          Prüfung auf ZXDISASM
r present
8500 REM      Create true address
8510 IF a$(LEN a$)="H" THEN LET
a=0: FOR x=1 TO LEN a$-1: LET a=
a*16+CODE a$(x)-CODE "0"-7*(a$(x
)>"9"): NEXT x: RETURN
8520 FOR x=2 TO CODE s$(1)+1: IF
 a$=s$(x, TO LEN a$*(LEN a$<7))
THEN LET a=CODE s$(x,7)+256*CODE
 s$(x,8): RETURN
8530 NEXT x
8540 LET a=VAL a$
8550 RETURN
8700 REM                 Start address
8710 LET a$="Startadresse eingeb
en:": GO SUB 9000: IF end THEN R
ETURN
8720 GO SUB 8500                         in Wert umwandeln
8730 RETURN
9000 REM                    Line input
9010 INPUT (a$+" "); LINE a$: LE
T end=NOT LEN a$
9020 FOR x=1 TO LEN a$: LET a$(x
```

```
)=CHR$ (CODE a$(x)-32*(a$(x)>="a
")): NEXT x
9030 RETURN
9100 REM                    h$=hex$(h)  4 Leerräume
9110 LET h$="     ": GO TO 9130    2 Leerräume
9120 LET h$="  "
9130 LET h1=h: FOR x=LEN h$ TO 1
 STEP -1: LET x1=h1-INT (h1/16)*
16: LET h$(x)=CHR$ (x1+CODE "0"+
7*(x1>9)): LET h1=INT (h1/16): N
EXT x
9140 RETURN
9999 STOP
```

Bildschirm-Werkzeug

Dieser Anhang enthält das vollständige Assembler-Quellprogramm für eine Bildschirm-Werkzeugtasche zum Gebrauch mit dem Spectrum. Die Werkzeugtasche verschafft Ihnen die Möglichkeit, an jeder Stelle auf dem Bildschirm anzuzeigen, nach links, nach rechts, nach oben und unten abzurollen. Die Scrolling-Routinen sind darin etwas Besonderes, daß sie jeweils eine **Pixel**reihe statt einer ganzen Zeichenreihe abrollen. Das erweckt den Eindruck völlig gleichmäßiger Bewegung, etwas, das Sie mit dem Abrollen ganzer Zeichenreihen nicht erreichen können.

Die Routine kann im RAM überall gesetzt werden, weil sie sich nicht auf absolute Adressen stützt. Das bedeutet, daß die Routine bei einem 16K-Spectrum ebenso läuft wie bei einem mit 48K. Damit das möglich ist, mag ein Teil des Codes in der Subroutine mit Beginn Symboladresse "PCALC:" allzu kompliziert aussehen.

Bevor Sie die Routine laden, sollten Sie die Adresse von RAMTOP heruntersetzen, um genügend Raum dafür zu schaffen. Bei einem Spectrum 48K könnten Sie CLEAR 64999 nehmen (womit die Routine bei 65000 beginnt), während beim Spectrum 16K CLEAR 32599 in Frage kommt. In diesem Fall beginnt die Routine bei 32600.

Es gibt zwei Hauptwege, die Routine einzugeben:

1. Den Assemblierer verwenden. Geben Sie das Assembler-Quellprogramm (nicht die Spalten mit den Überschriften

"Addr" oder "Hex") entweder direkt in ZXASM ein oder durch Verwendung von Autoscan. Einzelheiten siehe bei ZXASM.

2. Eingabe des Hexcodes. Verwenden Sie das Programm ZXMCMON und geben Sie die Hexcodierung aus der Spalte mit der Überschrift "Hex" ein. Wenn Sie diese Methode verwenden, prüfen Sie in verschiedenen Abständen, daß die auf dem Bildschirm gezeigte laufende Adresse mit der unten im Listing stehenden übereinstimmt.

In beiden Fällen sichern Sie, wenn die Routine schließlich über RAMTOP geladen ist, auf Band mit dem Befehl:

SAVE "TOOLKIT" CODE xxxx, 256

Dabei steht xxx für die Adresse, wo Sie laden wollen - entweder 32600 oder 65000.

Im Gebrauch funktioniert die Routine so:

1. Ein Zeichen auf dem Bildschirm anzeigen:

```
POKE 23681, CODE "x"   (Code des anzuzeigenden
                              Zeichens)
POKE 23728, col        (Spaltennummer Bildschirm)
POKE 32729, row        (Reihennummer Bildschirm)
RANDOMIZE USR xxxx     (xxxx = 65000 oder 32600)
```

2. Abrollen

```
POKE 23681, n          (n für Zahl der abzurollenden
                          Pixelreihen. 0 < n < 256)
RANDOMIZE USR
(xxxx +2)              (nach links abrollen)
RANDOMIZE USR
(xxxx +4)              (nach unten abrollen)
RANDOMIZE USR
(xxxx +6)              (nach oben abrollen)
RANDOMIZE USR
(xxxx +8)              (nach rechts abrollen)
```

```
10 REM Tool Kit Vorführung
20 PPOLE 23681, CODE "*"
30 FOR y = 8 TO 13: FOR x = 13 TO 17: POKE
   23728, x: POKE 23729, y: RANDOMIZE USR 32600:
   NEXT x: NEXT y
40 FOR x = 1   TO 16: POKE 23681, x
```

```
 50 RANDOMIZE USR 65008
 60 RANDOMIZE USR 65006
 70 RANDOMIZE USR 65002
 80 RANDOMIZE USR 65004
 90 NEXT x
100 POKE 23681, 160: RANDOMIZE USR 65008
110 STOP
```

```
ADDR HEX          OP    OPERAND
                  CHARS:  EQU 23606
                  UDG:    EQU 23675
                  SPARE1: EQU 23681
                  SPARE2: EQU 23728
                  PFLAG:  EQU 23697
                  PCLOAD: EQU 52H
EA60 1808         JR    PRNTCH
EA62 183F         JR    SCRLL
EA64 185A         JR    SCRLUP
EA66 185A         JR    SCRLDN
EA68 185A         JR    SCRLRT
                  PRNTCH:
EA6A 3A815C       LD    A,(SPARE1)
EA6D ED4BB05C     LD    BC,(SPARE2)
                  XPRNT:
EA71 FE80         CP    80H
EA73 3807         JR    C,NOTUG
EA75 D690         SUB   90H
EA77 2A7B5C       LD    HL,(UDG)
EA7A 1806         JR    PRNTAL
                  NOTUG:
EA7C 2A365C       LD    HL,(CHARS)
EA7F 24           INC   H
EA80 D620         SUB   20H
                  PRNTAL:
EA82 EB           EX    DE,HL
EA83 6F           LD    L,A
EA84 2600         LD    H,0
EA86 29           ADD   HL,HL
EA87 29           ADD   HL,HL
EA88 29           ADD   HL,HL
EA89 19           ADD   HL,DE
EA8A AF           XOR   A
EA8B FE           DI
EA8C CD5200       CALL  PCLOAD
EA8F 184C         JR    PCALC
EA91 3A915C       LD    A,(PFLAG)
EA94 4F           LD    C,A
EA95 0608         LD    B,8
                  PRNTLP:
EA97 7E           LD    A,(HL)
EA98 CB59         BIT   3,C
EA9A 2801         JR    Z,NOTINV
EA9C 2F           CPL
                  NOTINV:
EA9D 12           LD    (DE),A
EA9E 23           INC   HL
EA9F 14           INC   D
EAA0 10F5         DJNZ  PRNTLP
EAA2 C9           RET
                  SCRLL:
EAA3 3A815C       LD    A,(SPARE1)
EAA6 4F           LD    C,A
                  LCROW:
EAA7 21FF3F       LD    HL,3FFFH
EAAA 06C0         LD    B,192
                  LSCRN:
EAAC 112000       LD    DE,32
EAAF 19           ADD   HL,DE
EAB0 E5           PUSH  HL
EAB1 AF           XOR   A
                  LROW:
EAB2 7E           LD    A,(HL)
EAB3 17           RLA
EAB4 77           LD    (HL),A
EAB5 2B           DEC   HL
EAB6 1D           DEC   E
EAB7 20F9         JR    NZ,LROW
EAB9 E1           POP   HL
EABA 10F0         DJNZ  LSCRN
EABC 0D           DEC   C
EABD 20E8         JR    NZ,LCROW
EABF C9           RET
                  SCRLUP:
```

```
ADDR HEX        OP    OPERAND
EACØ 1837       JR    SCRLU2
                SCRLDN:
EAC2 1864       JR    SCRLD2
                SCRLRT:
EAC4 3A815C     LD    A,(SPARE1)
EAC7 4F         LD    C,A
                RCROW:
EAC8 210040     LD    HL,4000H
EACB 08C0       LD    B,192
                RSCRN:
EACD 1E20       LD    E,32
EACF AF         XOR   A
                RROW:
EAD0 7E         LD    A,(HL)
EAD1 1F         RRA
EAD2 77         LD    (HL),A
EAD3 23         INC   HL
EAD4 1D         DEC   E
EAD5 20F9       JR    NZ,RROW
EAD7 10F4       DJNZ  RSCRN
EAD9 0D         DEC   C
EADA 20EC       JR    NZ,RCROW
EADC C9         RET
                PCALC:
EADD 3B         DEC   SP
EADE 3B         DEC   SP
EADF E3         EX    (SP),HL
EAE0 23         INC   HL
EAE1 23         INC   HL
EAE2 E3         EX    (SP),HL
EAE3 FB         EI
EAE4 F5         PUSH  AF
EAE5 F5         PUSH  AF
EAE6 78         LD    A,B
EAE7 E618       AND   18H
EAE9 F640       OR    40H
EAEB 57         LD    D,A
EAEC F1         POP   AF
EAED 82         ADD   D
EAEE 57         LD    D,A
EAEF 78         LD    A,B
EAF0 E607       AND   7
EAF2 0F         RRCA
EAF3 0F         RRCA
EAF4 0F         RRCA
EAF5 81         ADD   C
EAF6 5F         LD    E,A
EAF7 F1         POP   AF
EAF8 C9         RET
                SCRLU2:
EAF9 3A815C     LD    A,(SPARE1)
EAFC 47         LD    B,A
                NXTSU:
EAFD C5         PUSH  BC
EAFE AF         XOR   A
EAFF 47         LD    B,A
EB00 4F         LD    C,A
                NXTLU:
EB01 F3         DI
EB02 CD5200     CALL  PCLOAD
EB05 18D6       JR    PCALC
EB07 3C         INC   A
EB08 FE08       CP    8
EB0A 3807       JR    C,SROW
EB0C 04         INC   B
EB0D 78         LD    A,B
EB0E FE18       CP    24
EB10 3012       JR    NC,UPDONE
EB12 AF         XOR   A
                SROW:
EB13 EB         EX    DE,HL
EB14 F3         DI
EB15 CD5200     CALL  PCLOAD
EB18 18C3       JR    PCALC
```

```
ADDR HEX          OP    OPERAND
EB1A EB           EX    DE,HL
EB1B C5           PUSH  BC
EB1C Ø12ØØØ       LD    BC,32
EB1F EDBØ         LDIR
EB21 C1           POP   BC
EB22 18DD         JR    NXTLU
                  UPDONE:
EB24 C1           POP   BC
EB25 1ØD6         DJNZ  NXTSU
EB27 C9           RET
                  SCRLD2:
EB28 3A815C       LD    A,(SPARE1)
EB2B 47           LD    B,A
                  NXTSD:
EB2C C5           PUSH  BC
EB2D 3EØ7         LD    A,7
EB2F Ø617         LD    B,23
EB31 ØEØØ         LD    C,Ø
                  NXTLD:
EB33 F3           DI
EB34 CD52ØØ       CALL  PCLOAD
EB37 18A4         JR    PCALC
EB39 EB           EX    DE,HL
EB3A 3D           DEC   A
EB3B FEFF         CP    ØFFH
EB3D 2ØØ8         JR    NZ,SDROW
EB3F Ø5           DEC   B
EB4Ø 78           LD    A,B
EB41 FEFF         CP    ØFFH
EB43 2812         JR    Z,DNDONE
EB45 3EØ7         LD    A,7
                  SDROW:
EB47 F3           DI
EB48 CD52ØØ       CALL  PCLOAD
EB4B 189Ø         JR    PCALC
EB4D EB           EX    DE,HL
EB4E C5           PUSH  BC
```

```
ADDR HEX          OP    OPERAND
EB4F Ø12ØØØ       LD    BC,32
EB52 EDBØ         LDIR
EB54 C1           POP   BC
EB55 18DC         JR    NXTLD
                  DNDONE:
EB57 C1           POP   BC
EB58 1ØD2         DJNZ  NXTSD
EB5A C9           RET

Symbols:
CHARS   5C36      UDG     5C7B
SPARE1  5C81      SPARE2  5CBØ
PFLAG   5C91      PCLOAD  ØØ52
MENU    EA6Ø      PRNTCH  EA6A
XPRNT   EA71      NOTUG   EA7C
PRNTAL  EA82      PRNTLP  EA97
NOTINV  EA9D      SCRLL   EAA3
LCROW   EAA7      LSCRN   EAAC
LROW    EAB2      SCRLUP  EACØ
SCRLDN  EAC2      SCRLRT  EAC4
RCROW   EACØ      RSCRN   EACD
RROW    EADØ      PCALC   EADD
SCRLU2  EAF9      NXTSU   EAFD
NXTLU   EBØ1      SROW    EB13
UPDONE  EB24      SCRLD2  EB28
NXTSD   EB2C      NXTLD   EB33
SDROW   EB47      DNDONE  EB57
```

ZX-SPECTRUM-POWER

„Wir haben von jedem Programm eine funktionierende Version hergestellt und diese bis zur völligen Zerstörung getestet." So beschreibt Trevor Toms das System der radikalen Gründlichkeit und Vollständigkeit, mit der er sein SPECTRUM-BUCH angelegt hat, ein definitives Handbuch zum SINCLAIR SPECTRUM, ein Werk, das den ganzen Spaß und Nutzen dieses Micro-Computers erschließt und auch für erfahrene SPECTRUM-Benutzer keine Lücke offen läßt. Toms: „Manche der hier gebotenen Instrumente und Programme können sogar verwendet werden, ohne daß man ihren Zweck verstehen muß!"

DAS SPECTRUM-BUCH gliedert sich in einen Teil mit Spaß-, Spiel- und Nutzprogrammen in BASIC, wobei es neben den einfachen, schnell zu realisierenden Programmen auch große, zeitaufwendige Programme gibt, und in einen MASCHINENCODE-Teil, der dem SPECTRUM-Benutzer klarmacht, welche Zukunft er mit seinem Gerät erleben kann: hier wird es im Umgang mit ZXASM (Symbolischer Assembliere-rer), ZXDISAM (Symbolischer Disassemblierer) und ZXMCMON (Maschinencode-Monitor) wirklich aufregend. Ein hochwillkommener Service-Teil BILDSCHIRM-WERKZEUGE enthält das vollständige Assembler-Quellprogramm für eine Bildschirm-Werkzeugtasche zum Gebrauch mit dem Spectrum.

Mit Recht darf der Autor feststellen, daß der Fortschritt von vorangegangenen ZX-Handbüchern zum SPECTRUM-BUCH so gewaltig ist wie der vom ZX 81 zum ZX SPECTRUM.

ZX POWER

ISBN 3-19-008207-3 Hueber Software