80-Zeichen-Grafik für den C 128

Der neue Basic-Interpreter des Commodore 128 enthält 26 Befehle, mit deren Hilfe man einfach und schnell auch komplizierte Grafik-Programme erstellen kann. Leider unterstützen diese Befehle nur die Programmierung des VIC-Chips mit der schon vom C 64 her bekannten Auflösung. Der zusätzlich eingebaute 80-Zeichen-Chip »VDC« wird nur zum Aufbau eines 80-Zeichen-Text-Bildschirms genutzt. Wer auch diesen Bildschirm für grafische Darstellungen benutzen will, der muß seine eigenen Routinen schreiben. Doch gibt es – wie der vorliegende Artikel zeigt – eine recht einfache Lösung für dieses Problem. Das abgedruckte Basic-Programm (Listing 1) baut ein Maschinenprogramm für den »User-RAM-Bereich« von $1300 bis $1bff auf. Nach dem Initialisieren dieses Maschinenprogramms mit »SYS DEC ("1303")« stehen die Befehle GRAFIK, BOX, CIRCLE, DRAW und PAINT auch für den VDC-Chip zur Verfügung. Die Befehle LOCATE, SCALE und SCNCLR und die Funktionen RCLR, RDOT und RGR können ohne Einschränkung im 640x200-Grafik-Modus benutzt werden. Beide Bildschirme können gleichzeitig im HiRes-Modus arbeiten, beim Aufruf eines Grafikbefehls prüft der Interpreter selbständig, welcher Video-Chip gerade angesprochen ist.

So programmiert man den 80-Zeichen-Chip

Der 8563-Video-Chip ist ein erweiterter »Abkömmling« des viel verwendeten 6845-CRT-Controllers. Er verfügt (im C 128) über 16 KByte eigenen Bildschirm-RAM und ist nur über zwei Register unter den Adressen 54784 ($d600) und 54785 ($d601) mit den anderen Bausteinen des Computers verbunden. Alle Register und der gesamte Video-RAM-Bereich müssen über diese beiden Adressen angesprochen werden. Das »Einblenden« des Video-RAMs in den Adressenbereich der CPU ist nicht vorgesehen. Will man einen Wert in eines der 36

Auf dieses Grafikpaket für eine 640 x 200-Punkte-Auflösung auf dem 80-Zeichen-Bildschirm haben C 128-Besitzer gewartet! Alle Grafikbefehle des Basic 7.0 stehen damit für den neuen Modus zur Verfügung.

Register des 8563 übertragen, so muß man zunächst die Nummer des anzusprechenden Registers in die Speicherstelle 54784 und dann den zu übertragenden Wert in die Speicherstelle 54785 schreiben. Genauso geht es beim Lesen eines Registerinhaltes: »POKE 54784, Registernummer: PRINT PEEK (54785): REM Registerinhalt«.

Die HiRes-Grafik-Register

Für die Grafikprogrammierung sind von den 36 Registern des VDC vor allem die Register 18, 19, 31, 25 und 26 interessant. Der Inhalt von Register 26 bestimmt (im Grafikmodus!) Vorder- und Hintergrundfarbe. Über Bit 0 bis 3 kann man 16 verschiedene Hintergrundfarben, über Bit 4 bis 7 16 verschiedene Vordergrundfarben anwählen.

Ein Beispiel: die Befehlsfolge »POKE 54784,26:POKE 54785, (3*16 + 2)« bewirkt, daß im HiRes-Modus eine rote Zeichenfarbe auf weißem Grund erscheint. Im Textmodus wird mit diesem Befehl nur die Hintergrundfarbe verändert. Register 25 ist ein mehrfach belegtes Register. Über die Bits 0 bis 3 kann man den Bildschirm um maximal 16 Pixels horizontal nach links verschieben. Mit Bit 4 kann man zwischen einfacher (0) und doppelter (1) Pixelgröße wählen. Mit den Bits 5 bis 7 kann man zwischen den drei Betriebsarten Text (Bit 6 = 1), Semigrafik (Bit 5 = 1) und Grafik (Bit 7 = 1) wählen.

Für uns ist hier nur interessant, daß man durch Beschreiben von Register 25 mit dem Wert 128 den HiRes-Modus wählen und durch Beschreiben des Registers mit dem Wert 64 wieder in den Text-Modus zurückkehren kann. Alle anderen Bits sind standardmäßig mit Null besetzt. Über die Register 18,19 und 31 wird das Video-RAM angesprochen. Man besetzt in der beschriebenen Weise zunächst Register 18 mit dem High-Byte und Register 19 mit dem Low-Byte der gewünschten Bildschirmspeicheradresse vor.

Durch Schreiben in Register 31 kann man dann einen Wert in die gewählte Speicherstelle schreiben, durch Auslesen von Register 31 erhält man den Inhalt dieser Speicherstelle. Aber Vorsicht! Nach jeder Schreib- oder Leseoperation wird der RAM-Zeiger in Register 18/19 automatisch einen Schritt erhöht! Will man also einen Wert aus dem Video-RAM auslesen und anschließend (verändert) wieder zurückschreiben, so muß man die Zeiger 18/19 zweimal setzen. Das erscheint zwar sehr umständlich, doch hat diese Arbeitsweise einen überzeugenden Vorteil: 10mal »PEEK (Reg. 31)« ergibt die 10 Bytes von Zeiger 18/19 bis Zeiger 18/19 + 9. Der Zeiger 18/19 muß nur einmal gesetzt werden. Das Auslesen oder Vorbesetzen größerer Speicherbereiche wird so erheblich beschleunigt. Man überlege sich einmal, um wieviel langsamer die Eingabeschleife des Bildschirmeditors würde, wenn Register 18 und 19 auf jedes Zeichen einzeln gesetzt werden müßten.

Organisation des Grafik-Bildschirms

Für jedes Pixel (Grafikpunkt) wird ein Bit benötigt. Das macht acht Pixel pro Byte oder 80 Byte für eine waagerechte Bildzeile bei einer horizontalen Auflösung von 640 Pixels. In vertikaler Richtung erreicht der VDC 200 Zeilen Auflösung, das heißt insgesamt werden 80x200 = 1600 Byte Video-RAM von einem Bild belegt. Ein Farb-RAM ist nicht vorgesehen. Der VDC kann hochauflösende Grafik nur in einer einzigen Farbe darstellen, ein entscheidender Nachteil des VDC gegenüber dem VIC. Auch in der Adressierung der einzelnen Pixels unterscheiden sich VDC und VIC.

```
20170 data 16e5, 20, 46, 17
20180 data 16fa, 20, 46, 17
20190 data 1740, 4c, b5, 16
20200 data 1869, 4c, 1d, 1a
20210 data 1672, 20, 32, 9e
20220 data 1781, 20, 32, 9e
20230 data 1893, 20, 32, 9e
20240 data 1853, ea, ea, ea
20250 data ende
50000 :
60000 zz$="graphik-80": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 1. »Graphik-80« (Schluß)

```
10 fast
20 bank15
30 bload"graphik-80.m"
40 :
50 sys dec("1303")
60 :
90 graphic 6,1
100 circle,320,100,250,99,70,260
110 box, 145,100, 115,110, 160, 1
112 box, 145,100, 115,110,  70, 1
120 circle ,110,45,40,15,1
130 circle,500,40,37,17,,,,45
140 circle,350,100, 160,60, 145,220
150 draw 1, 300,50 to 300,110 to 400,120
 to 300,50
160 circle0,320,100,250,99,70,260
170 circle,320,100,250,99,70,260
182 paint ,320,90
184 circle, 130,85, 200,80, 330,10
186 circle, 509,85, 200,80, 330,10
188 circle, 110,0, 100,55, 158,202
189 circle, 490,0,  97,50, 155,190
190 get q$: if q$<>"" then 200
192 circle1, 340,196, 160,63, 325,40
193 for i=1 to 500: next
194 circle0, 340,196, 160,63, 325,40
195 for i=1 to 500: next
196 goto190
200 graphic 5
220 end

ready.
```

Listing 2. Eine kleine Demonstration der Fähigkeiten von »Graphik-80«.

```
1 rem "                              Setpoint-128.a
2 :
3 rem " zeichnet, l"scht oder testet einen Punkt auf dem 8563-HIRES-Bildschirm
4 :
5 rem "                                (c) by KRW
6 :
7 rem "                                   1985
8 :
9 rem "                                version 1.03
10 :
11 :
20 n$="setpoint-128.3.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
120 ; "*** Varible ***"
130 ;
140 x1l     = $1131          ; "Koordinaten
150 x1h     = $1132          ; "vgl. c-128-Anleitung  S. H-22 !
160 y1      = $1133
165 yh      = $1134
170 zero    = $fd            ; "2-Byte-Zeiger  (fd/fe)
180 zux     = $fc            ; "Arbeitsspeicherstelle"
190 vdc0    = $d600          ; "Adressen der beiden VDC-Register
200 vdc1    = $d601
202 colsel  = $83            ; "-ber COLOR-Befehl gew=hlte Zeichenfarbe
204 ramfig  = $9e            ; "Zwischenspeicher f-r RAM-Konfiguration
210 ;
211 xlim     = 639           ; "Bereichsgrenzen
212 ylim     = 199
215 ;
220 ; "*** ROM- Unterprogramme ***"
230 ;
240 setreg   =  $cdcc       ; vdc setzen x=registernummer a=inhalt des registers
250 fechreg  =  $cdd8       ; wert reg. 31 des vdc in akku
270 ;
300 *= $1400
310 ;
900 ↑        jmp setpnt     ; "setzt (colsel<>0) oder l"scht (colsel=0) Punkt
```

Listing 3. Assemblerlisting »setpoint«. Nur zur Dokumentation, nicht eingeben

Ein Byte beim VDC beschreibt acht nebeneinanderliegende Pixels, während ein Byte beim VIC acht untereinanderliegende Pixels kontrolliert. Anders ausgedrückt: Das Pixel mit den Koordinaten 7,0 wird von Bit 0 (!) der ersten Bildschirmspeicherstelle repräsentiert, das Pixel 0,7 von der (7x80=) 560. Speicherstelle. Daraus ergibt sich die folgende Formel zur Berechnung der Bildschirmspeicheradresse für ein gegebenes Koordinatenpaar X,Y:

ADR= Y*80 + INT (X/8)

Zur Berechnung des Bitwertes innerhalb dieser Speicherstelle erhält man (genauso wie beim VIC):

BIT =

2↑ (7 – (»Low-Byte von x« AND 7))

Eine Routine »Punkt setzen« könnte man dann formulieren als:

ADR = (»Inhalt von ADR«) OR BIT

Die entsprechend formulierte Formel für »Punkt löschen» lautet:

ADR = (»Inhalt von ADR«) AND (NOT BIT)

Und schließlich wird noch eine Routine »Punkt testen« für den PAINT-Befehl benötigt:

ERGIBT = (»Inhalt von ADR«) AND BIT

»ERGIBT« wird Null, wenn der Punkt nicht gesetzt ist, und wird ungleich, wenn der Punkt gesetzt ist. Mit diesen Formeln ist ein Programm zum Setzen, Löschen oder Testen eines Bildpunktes auf dem VDC-HiRes-Bildschirm vollständig beschrieben. Das als Listing 3 abgedruckte Maschinenprogramm »setpoint« ist die für das hier beschriebene Grafik-Paket verwendete praktische Formulierung dieses Programms. Dabei wird das Produkt »y*80« nicht berechnet, sondern aus einer Tabelle geladen. Das ist die wahrscheinlich schnellste Möglichkeit, die richtige Bildschirm-RAM-Adresse zu errechnen. Kurze Rechenzeiten aber sind nötig, wenn man bedenkt, wie häufig die Routine aufgerufen werden muß, um eine einzige Kurve auf den Bildschirm zu zeichnen. Alle anderen Teile dieses Programms enthalten keine Besonderheiten. Für den interessierten Leser wurde das Assemblerlisting mit ausführlichen Kommentaren versehen. So dürfte es keine Schwierigkeiten bereiten, eigene Änderungen oder Erweiterungen vorzunehmen.

Komfortable Zeichenbefehle

Bis hierhin wurde recht ausführlich besprochen, wie man den Grafikschirm einschaltet und Punkte setzt. Nun wird sofort der Wunsch wach, Linien, Kreise oder andere Figuren zu zeichnen. Dazu sind schon recht anspruchsvolle Berechnungen nötig. Doch alle Rechenroutinen sind im Basic-ROM des C 128 enthalten. Allen Routinen ist gemeinsam, daß sie im Unterprogramm »Punkt setzen« münden. Die ROM-Routine dafür kann nur Punkte für den VIC-Bildschirm berechnen. Die oben beschriebene Routine bewirkt das gleiche für den VDC-Chip. Der Gedanke liegt nun nahe, in Routinen wie CIRCLE oder DRAW, die Zeiger, die auf das Unterprogramm »Punkt setzen, löschen, testen« zeigen, auf das eigene »Punkt-setzen-Programm« zu »verbiegen«, das den VDC anspricht. Im ROM ist das zwar leider nicht möglich, und die meisten C 128-Benutzer werden ihren neuen 128er nicht gleich mit neuen EPROMS versehen wollen, aber es geht ja auch viel einfacher: Man kopiert einfach die entscheidenden Programmteile ins RAM, paßt die besagten Zeiger an und teilt

```
910 ↑         jmp initram    ; "s. Zeile 920f. !
915 ↑         jmp testpnt    ; "prüft, ob Punkt gesetzt
920 initram   lda $ff00      ; "Init-RAM schaltet RAM-Konfiguration
930 ↑         pha            ; "auf gemeinsamen RAM-Bereich von $0 bis $3fff
940 ↑         lda #$00
950 ↑         sta $ff00
960 ↑         lda #$07
970 ↑         sta $d506      ; "RAM-Konfigurations-Register
980 ↑         pla
990 ↑         sta $ff00
995 ↑         rts
1000 zeiger   lda $ff00      ; "alte RAM-Konfiguration merken
1004 ↑        sta ramfig
1006 ↑        lda #$00       ; "Bank 15 einschalten
1008 ↑        sta $ff00
1010 ↑        sta zero       ; "Arbeitsspeicherstelle löschen
1020 ↑        sta zero+1
1022 ↑        lda yh
1024 ↑        bne ende
1025 ↑        lda #<ylim
1026 ↑        cmp y1
1027 ↑        bcc ende
1029 ↑        lda #<xlim     ; "Zeile 1022-1033: Bereichsgrenzen prüfen
1030 ↑        sbc x11
1031 ↑        lda #>xlim
1032 ↑        sbc x1h
1033 ↑        bcc ende
1038 ↑        ldy y1         ; "Koordinaten in Bildschirm-RAM-Adresse umrechnen:
1040 ↑        lda faklow,y   ; "1) y-Anteil auswerten
1050 ↑        sta zero       ; "y-Anteil = y-Koordinate * 80
1060 ↑        lda fakhigh,y  ; "low-Byte aus Tabelle faklow
1070 ↑        sta zero+1     ; "high-Byte aus Tabelle fakhigh
1140 ;
1150 setpt1   ldx x1h        ; "2) x-Anteil auswerten und zum y-Anteil addieren
1151 ↑        lda x11    ;   ; "low-x = int(x11/8) <Zeile 1151-1155>
1152 ↑        and #%11111000
1153 ↑        lsr
1154 ↑        lsr
1155 ↑        lsr
1157 ↑        clc            ; "x-Anteil = low-x + high-x = zux
1160 ↑        adc divtab,x   ; "high-x aus Tabelle
1170 ↑        sta zux
1250 ↑        clc            ; "Zeilen 1250 bis 1290:
1260 ↑        lda zero       ; "3)  RAM-Adresse = (zero/zero+1) + zux
1270 ↑        adc zux
1275 ↑        sta zero       ; "Zeilen 1300 bis 1330:
1280 ↑        lda zero+1     ; "Berechnetes Byte aus 8563-Bildschirm-RAM holen
1281 ↑        adc #$00
1290 ↑        sta zero+1     ; "Wichtig: erst high-Byte, dann low-Byte !!!
1300 ↑        ldx #$12
1305 ↑        jsr setreg     ; "high-Byte der RAM-Adresse setzen
1310 ↑        ldx #$13
1315 ↑        lda zero
1320 ↑        jsr setreg     ; "low-Byte der RAM-Adresse setzen
1330 ↑        jsr fechreg    ; "Byte holen
1340 ↑        sta zux        ; "und merken
1350 ↑        lda x11        ; "Zeile 1350 bis 1380: Bitzeiger berechen
1360 ↑        and #7
1370 ↑        tax            ; "bit = 2↑((7-x) and 7)
1380 ↑        lda hoch2,x    ; "A enthält bit-Wert
1390 ↑        clc
1400 ↑        rts
1410 ;
1500 ende     sec            ; "Ende, wenn Bereichsgrenzen überschritten
1510 ↑        rts
1540 ;
1700 hoch2    .byt $80, $40, $20, $10
1710 ↑        .byt $08, $04, $02, $01
1720 ;
1730 divtab   .byt $00,$20,$40
1740 ;
3000 faklow   .byt   <0*80,  <1*80,  <2*80,  <3*80,  <4*80,  <5*80
3010 ↑        .byt   <6*80,  <7*80,  <8*80,  <9*80, <10*80, <11*80
3020 ↑        .byt  <12*80, <13*80, <14*80, <15*80, <16*80, <17*80
3030 ↑        .byt  <18*80, <19*80, <20*80, <21*80, <22*80, <23*80
3040 ↑        .byt  <24*80, <25*80, <26*80, <27*80, <28*80, <29*80
3050 ↑        .byt  <30*80, <31*80, <32*80, <33*80, <34*80, <35*80
3060 ↑        .byt  <36*80, <37*80, <38*80, <39*80, <40*80, <41*80
3070 ↑        .byt  <42*80, <43*80, <44*80, <45*80, <46*80, <47*80
3080 ↑        .byt  <48*80, <49*80, <50*80, <51*80, <52*80, <53*80
3090 ↑        .byt  <54*80, <55*80, <56*80, <57*80, <58*80, <59*80
3100 ↑        .byt  <60*80, <61*80, <62*80, <63*80, <64*80, <65*80
3110 ↑        .byt  <66*80, <67*80, <68*80, <69*80, <70*80, <71*80
3120 ↑        .byt  <72*80, <73*80, <74*80, <75*80, <76*80, <77*80
3130 ↑        .byt  <78*80, <79*80, <80*80, <81*80, <82*80, <83*80
3140 ↑        .byt  <84*80, <85*80, <86*80, <87*80, <88*80, <89*80
3150 ↑        .byt  <90*80, <91*80, <92*80, <93*80, <94*80, <95*80
3160 ↑        .byt  <96*80, <97*80, <98*80, <99*80,<100*80,<101*80
3170 ↑        .byt <102*80,<103*80,<104*80,<105*80,<106*80,<107*80
3180 ↑        .byt <108*80,<109*80,<110*80,<111*80,<112*80,<113*80
3190 ↑        .byt <114*80,<115*80,<116*80,<117*80,<118*80,<119*80
3200 ↑        .byt <120*80,<121*80,<122*80,<123*80,<124*80,<125*80
3210 ↑        .byt <126*80,<127*80,<128*80,<129*80,<130*80,<131*80
3220 ↑        .byt <132*80,<133*80,<134*80,<135*80,<136*80,<137*80
3230 ↑        .byt <138*80,<139*80,<140*80,<141*80,<142*80,<143*80
3240 ↑        .byt <144*80,<145*80,<146*80,<147*80,<148*80,<149*80
3250 ↑        .byt <150*80,<151*80,<152*80,<153*80,<154*80,<155*80
3260 ↑        .byt <156*80,<157*80,<158*80,<159*80,<160*80,<161*80
3270 ↑        .byt <162*80,<163*80,<164*80,<165*80,<166*80,<167*80
3280 ↑        .byt <168*80,<169*80,<170*80,<171*80,<172*80,<173*80
```

```
3290 ↑        .byt <174*80,<175*80,<176*80,<177*80,<178*80,<179*80
3300 ↑        .byt <180*80,<181*80,<182*80,<183*80,<184*80,<185*80
3310 ↑        .byt <186*80,<187*80,<188*80,<189*80,<190*80,<191*80
3320 ↑        .byt <192*80,<193*80,<194*80,<195*80,<196*80,<197*80
3330 ↑        .byt <198*80,<199*80,<200*80,<201*80,<202*80,<203*80
3340 ↑        .byt <204*80
3390 ;
3400 fakhigh .byt   >0*80,  >1*80,  >2*80,  >3*80,  >4*80,  >5*80
3410 ↑        .byt   >6*80,  >7*80,  >8*80,  >9*80, >10*80, >11*80
3420 ↑        .byt  >12*80, >13*80, >14*80, >15*80, >16*80, >17*80
3430 ↑        .byt  >18*80, >19*80, >20*80, >21*80, >22*80, >23*80
3440 ↑        .byt  >24*80, >25*80, >26*80, >27*80, >28*80, >29*80
3450 ↑        .byt  >30*80, >31*80, >32*80, >33*80, >34*80, >35*80
3460 ↑        .byt  >36*80, >37*80, >38*80, >39*80, >40*80, >41*80
3470 ↑        .byt  >42*80, >43*80, >44*80, >45*80, >46*80, >47*80
3480 ↑        .byt  >48*80, >49*80, >50*80, >51*80, >52*80, >53*80
3490 ↑        .byt  >54*80, >55*80, >56*80, >57*80, >58*80, >59*80
3500 ↑        .byt  >60*80, >61*80, >62*80, >63*80, >64*80, >65*80
3510 ↑        .byt  >66*80, >67*80, >68*80, >69*80, >70*80, >71*80
3520 ↑        .byt  >72*80, >73*80, >74*80, >75*80, >76*80, >77*80
3530 ↑        .byt  >78*80, >79*80, >80*80, >81*80, >82*80, >83*80
3540 ↑        .byt  >84*80, >85*80, >86*80, >87*80, >88*80, >89*80
3550 ↑        .byt  >90*80, >91*80, >92*80, >93*80, >94*80, >95*80
3560 ↑        .byt  >96*80, >97*80, >98*80, >99*80,>100*80,>101*80
3570 ↑        .byt >102*80,>103*80,>104*80,>105*80,>106*80,>107*80
3580 ↑        .byt >108*80,>109*80,>110*80,>111*80,>112*80,>113*80
3590 ↑        .byt >114*80,>115*80,>116*80,>117*80,>118*80,>119*80
3600 ↑        .byt >120*80,>121*80,>122*80,>123*80,>124*80,>125*80
3610 ↑        .byt >126*80,>127*80,>128*80,>129*80,>130*80,>131*80
3620 ↑        .byt >132*80,>133*80,>134*80,>135*80,>136*80,>137*80
3630 ↑        .byt >138*80,>139*80,>140*80,>141*80,>142*80,>143*80
3640 ↑        .byt >144*80,>145*80,>146*80,>147*80,>148*80,>149*80
3650 ↑        .byt >150*80,>151*80,>152*80,>153*80,>154*80,>155*80
3660 ↑        .byt >156*80,>157*80,>158*80,>159*80,>160*80,>161*80
3670 ↑        .byt >162*80,>163*80,>164*80,>165*80,>166*80,>167*80
3680 ↑        .byt >168*80,>169*80,>170*80,>171*80,>172*80,>173*80
3690 ↑        .byt >174*80,>175*80,>176*80,>177*80,>178*80,>179*80
3700 ↑        .byt >180*80,>181*80,>182*80,>183*80,>184*80,>185*80
3710 ↑        .byt >186*80,>187*80,>188*80,>189*80,>190*80,>191*80
3720 ↑        .byt >192*80,>193*80,>194*80,>195*80,>196*80,>197*80
3730 ↑        .byt >198*80,>199*80,>200*80,>201*80,>202*80,>203*80
3740 ↑        .byt >204*80
3900 ;
4000 setpnt   jsr zeiger     ; "Adresse des zu ändernden Bytes im
4005 ↑        bcs ende1
4010 ↑        ldx colsel     ; "Bildschirm-RAM ausrechnen, bit-Wert nach A
4020 ↑        bne orflag     ; "wenn Farbe = Hintergrundfarbe, dann löschen
4030 ↑        eor #$ff
4040 ↑        and zux
4050 ↑        .byt $2c
4060 orflag   ora zux        ; "Vordergrundfarbe gewählt, dann setzen
4070 ↑        sta zux        ; "geändertes Byte zwischenspeichern
4080 ↑        ldx #$12       ; "Register setzen  s.o. !
4090 ↑        lda zero+1
4100 ↑        jsr setreg
4110 ↑        ldx #$13
4120 ↑        lda zero
4130 ↑        jsr setreg
4140 ↑        ldx #$1f
4150 ↑        lda zux        ; "geändertes Byte in Bildschirm-RAM
4160 ↑        jsr setreg
4170 ende1    lda ramfig     ; "alte RAM-Konfiguration wiederherstellen
4180 ↑        sta $ff00
4190 ↑        rts            ; "fertig.
4200 ;
5000 testpnt jsr zeiger      ; "Koordinaten nach Bildschirm-RAM-Adresse,
5005 ↑        bcs ende1
5010 ↑        and zux        ; "bit-Wert nach A
5020 ↑        beq ende2      ; "Punkt gesetzt ?
5030 ↑        jsr ende1      ; "ja, dann RAM-Konfig. wiederherstellen
5040 ↑        ldx #0
5050 ↑        rts            ; "zurück mit gesetztem Zero-Flag
5060 ende2    jsr ende1      ; "nein, dann RAM-Konfig. wiederherstellen
5070 ↑        ldx #255
5080 ↑        rts            ; "zurück mit Zero-Flag = 0
5090 ;
8000 .end
9000 end
```

```
10000 :
60000 zz$="setpoint-128.3.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 3. Assemblerlisting »setpoint« (Schluß)

zuletzt noch dem Interpreter mit, daß er fortan beim Aufruf der Grafik-Befehle die neuen Routinen »anspringen« soll, wenn der 80-Zeichen-Bildschirm eingeschaltet ist. Die Lösung dieser Aufgabe ist das abgedruckte Basic-Programm (Listing 1). Es stellt alle Programmteile zusammen, die benötigt werden, um die Befehle GRAPHIC, BOX, CIRCLE, DRAW und PAINT auch für den VDC wirksam werden zu lassen. Alle Befehle für den 640x200-Punkte-Bildschirm sind nur im Programm-Modus ausführbar. Das ist deshalb so eingerichtet, weil der 80-Zeichen-Bildschirm im HiRes-Modus zwangsläufig zerstört wird (siehe oben). Die Befehle können im Direkt-Modus also gar nicht vom Bildschirm geholt und interpretiert werden. Man erhielte lediglich »Dreckflecken« auf dem HiRes-Bild.

Probieren Sie einfach einmal im Direkt-Modus die Befehlsfolge »POKE 54784,25: POKE 54785,128« aus! Man kann sehr viel über die Funktionsweise des VDC im Text-Modus lernen. RUN/STOP-RESTORE rückt die Register wieder zurecht.

Der neue GRAPHIC-Befehl

Der GRAPHIC-Befehl wurde um die Funktionen GRAPHIC 6,0 und GRAPHIC 6,1 erweitert. GRAPHIC 6 schaltet den 8563-HiRes-Modus ein. Folgt der »6« eine »1«, so wird der Bildschirm gelöscht, folgt eine »0«, so bleibt der Bildschirm wie er ist. Der Befehl GRAPHIC 6,1 ersetzt auch den in dieser Implementation nicht vorgesehenen Befehl SCNCLR 6. Alle übrigen Informationen zur Implementation dieses Befehls sind in den ebenfalls abgedruckten ausführlich kommentierten Assemblerlistings des Befehls enthalten (Listing 4).

Die »Originalroutine« im ROM befindet sich im Speicherbereich ab $6b5a.

Die Befehle BOX, CIRCLE, DRAW und PAINT

Diese Befehle funktionieren im 8563-Modus genauso wie es im Bedienungshandbuch für die VIC-Grafik beschrieben ist. Einzige Änderung: Die x-Koordinaten dürfen nun im Bereich von 0 bis 639 liegen. Auch die Implementierung dieser Befehle ist denkbar einfach: Nacheinander werden die Programmteile PAINT ($61a8 bis $62b6), BOX ($62b7 bis $6388),

DRAW ($6797 bis $67d6) und CIRCLE ($668e bis $674c) in den RAM-Bereich ab $1672 kopiert (Basic-Programm Zeile 5000 bis 5340). Darunter, in den RAM-Bereich ab $1952, wird der Programmteil »Strecke zeichnen« aus ROM $9b30 bis $9c18 kopiert. Als nächstes werden die neuen Adressen für Unterprogrammaufrufe eingesetzt (WHILE-DO-Schleife). Es folgt schließlich der neue GRAPHIC-Befehl. Im Basic-Programm ist er in Form von DATA-Zeilen abgelegt. Ebenfalls in Form von DATA-Zeilen sind die schon besprochene Routine »setpoint« (RAM $1400 bis $1671) und die Erweiterung der Interpreterschleife im Basic-Text enthalten.

Die Interpreterschleife

Wie teile ich nun dem Interpreter mit, daß er beim Aufruf der Grafikbefehle nun nicht mehr zu den ROM-, sondern zu unseren neuen RAM-Routinen springen soll? Beim Starten eines Programms holt sich der Interpreter – genauso wie beim C 64 – die Adresse, die in den Speicherstellen $308 und $309 abgelegt ist. Er arbeitet dann das Programm ab, das bei dieser Adresse beginnt.

Setzt man in die Speicherstelle $308/09 nun die Adresse des eigenen Programms ein, dann beginnt der Interpreter nach dem Starten eines Programms mit RUN seine Arbeit bei der neuen Adresse. Eben dieses »Verbiegen« des Interpretervektors bewirkt der Befehl SYS DEC ("1303"). Und noch eine wichtige Kleinigkeit enthält die Initialisierungsroutine: die ins RAM kopierten Programmteile enthalten zahlreiche Unterprogrammsprünge (JSR) in ROM-Routinen. Der Prozessor kann diese Sprünge nur richtig ausführen, wenn ihm der RAM-Speicherbereich, in dem unser Programm liegt, und die angesprochenen ROM-Adressen als ein zusammenhängender 64-KByte-Block erscheinen. Für die Zusammenstellung solcher gemeinsamer Bereiche ist die MMU zuständig (siehe Bedienungshandbuch Anhang B). Die Initialisierungsroutine sorgt dafür, daß die MMU im Bereich von $0 bis $1fff immer die RAMs einschaltet, auf der auch unser Programm liegt. Anders ausgedrückt: Auch wenn man über den BANK-Befehl die ROM-Bank 15 ausgewählt hat, nach dem Durchlaufen unserer Initialisierungsroutine liest die CPU die Adressen zwischen $0 uns $1fff immer aus dem RAM-Bereich in Bank 0 aus.

```
1 rem "                    Graphik
2 :
3 rem "         verändert Ausführung des graphic-Befehls
4 :
5 rem "                   (c) by KRW
6 :
7 rem "                      1985
8 :
9 rem "                   version 1.00
10 :
11 :
20 n$="graphik.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
220 ; "*** Unterprogramme ***"
230 ;
240 setreg  = $cdcc
250 saverom = $ce0c
270 ;
900 *= $1a3e
910 ;
1000 ↑        cmp #$9e          ; "folgt CLR ?
1010 ↑        bne graph1
1020 ↑        jsr $a022         ; "ja, dann normale Speicheraufteilung zurück
1030 ↑        jsr $0380         ; "chrget
1040 ↑        lda #$00
1050 ↑        sta $d8           ; "Flag für Text-Modus setzen
1060 ↑        rts
1070 graph1   jsr $87f4         ; "Graphik-Modus holen
1080 ↑        cpx #$06          ; "GRAPHIC 6 ?
1090 ↑        beq graphik       ; "ja, dann Sprung
1100 ↑        bcs error         ; "X>6, dann 'illegal quantity'
1110 ↑        txa
1120 ↑        pha               ; "X merken
1140 ↑        lda #$00          ; "bank 15 einschalten
1150 ↑        sta $ff00
1160 ↑        ldx #25           ; "80-Zeichen-Bildschirm auf Text-Modus schalten
1170 ↑        lda #$40
1180 ↑        jsr setreg
1190 ↑        jsr saverom       ; "über Taste 'ASCII/DIN' gewählten Zeichensatz
1200 ↑        lda $d7           ; "ins 8563-CHARRAM kopieren
1210 ↑        pha
1220 ↑        lda #$80          ; "Zeile 1200 bis 1260:
1230 ↑        sta $d7           ; "80-Zeichen-Bildschirm löschen
1240 ↑        jsr $c142         ; "kernal-Routine clear
1250 ↑        pla
1260 ↑        sta $d7
1340 ↑        pla               ; "X 'zerstörungsfrei' aus Stack laden
1350 ↑        pha
1360 ↑        tax
1370 ↑        jmp $6b6e         ; "weiter mit normalem GRAPHIC-Befehl
1380 error    jmp $7d28         ; "'illegal quantity'
1390 ;
1400 graphik lda #$00           ; "bank 15 einschalten
1430 ↑        sta $ff00
1460 ↑        ldx #25
1470 ↑        lda #$80          ; "8563-HIRES-Mode einschalten
1480 ↑        jsr setreg
1500 ↑        jsr $9e1c         ; "chkcom, byte-Wert nach X
1502 ↑        lda #$00          ; "wieder auf bank 15
1504 ↑        sta $ff00
1510 ↑        cpx #$02          ; "X>1, dann illegal quantity
1520 ↑        bcs error
1525 ↑        cpx #$00          ; "Hires ein ohne Löschen des Bildschirms ?
1530 ↑        beq ende          ; "dann fertig
1540 ↑        ldy #$40          ; "Zeile 1540 bis 1670: HIRES-Bild löschen
1550 ↑        sty $08           ; "$08: 'Blockzähler': 40*256 bytes
1560 ↑        ldy #$00
1570 ↑        tya
1580 ↑        ldx #$12          ; "RAM-Zeiger high
1590 ↑        jsr setreg        ; "und
1600 ↑        ldx #$13          ; "RAM-Zeiger low
1610 ↑    0   jsr setreg        ; "auf Anfang = $0000 setzen
1620 ↑        ldx #31           ; "$00 in gewählte RAM-Speicherstelle
1630 empty    jsr setreg        ; "RAM-Zeiger zählt nach 'sta' selbst einen
1640 ↑        dey               ; "Schritt weiter !
1650 ↑        bne empty
1660 ↑        dec $08
1670 ↑        bne empty
1680 ende     rts               ; "fertig
6100 ;
8000 .end
9000 end
60000 zz$="graphik.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100  ,
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 4. Assemblerlisting »Graphik«. Nur zur Dokumentation, nicht eingegeben

```
1 rem "                Interpret-128.a
2 :
3 rem "        ver.ndert Interpreter-Schleife
4 :
5 rem "                  (c) by KRW
6 :
7 rem "                      1985
8 :
9 rem "                  version 1.00
10 :
11 :
20 n$="interpret-128.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
220 ; "*** Unterprogramme ***"
230 ;
240 chrget   =  $0380      ; "holt n.chstes Byte aus BASIC-Text
250 execute  =  $4af3      ; "f.hrt Befehl aus, token in A
260 inter    =  $4af6      ; "holt n.chsten Befehl und f.hrt ihn aus
270 ;
300 ; "*** Variable ***
310 ;
320 min     = $de    ; "kleinstes token
330 max     = $e9    ; "gr"tes token
360 ;
900 *= $1300
910 ;
1000 ↑       jmp anfang
1010 ↑       jmp init
1020 ;
2000 init    sei                    ; "pointer f.r die
2010 ↑       lda #<anfang           ; "Interpreterschleife Prg-Modus
2020 ↑       sta $0308              ; "auf 'anfang' setzen
2030 ↑       lda #>anfang
2040 ↑       sta $0309
2050 ↑       cli
2060 ↑       lda #$00
2070 ↑       sta $ff00              ; "bank 15 einschalten
2080 ↑       lda #$06
2090 ↑       sta $d506              ; "$0-$1fff gemeinsamer RAM-Bereich
2100 ↑       lda #$20
2110 ↑       sta $30
2120 ↑       sta $32
2130 ↑       sta $34
2140 ↑       lda #$00               ; "Variablen-Zeiger entsprechend
2150 ↑       sta $2f                ; "anpassen
2160 ↑       sta $31
2170 ↑       sta $33
2180 ↑       rts
2190 ;
3000 anfang  jsr chrget             ; "Befehl holen
3010 ↑       cmp #min               ; "eines der 'token', die auf
3020 ↑       bcc alt                ; "neue Befehlsadresse f.hren sollen ?
3030 ↑       cmp #max
3040 ↑       bcs alt
3050 ↑       tax                    ; "ja, dann
3052 ↑       bit $d7           0    ; "auf 40-Zeichen-Modus testen
3054 ↑       bpl vic                ; "40 Z., dann Sprung auf 'normale' Adresse
3060 ↑       lda tablow-min,x       ; "80 Z.: low-Byte
3070 ↑       sta prg+1              ; "                der neuen Befehlsadresse
3080 ↑       lda tabhigh-min,x      ; "       high-Byte
3090 ↑       sta prg+2              ; "                          nach prg+1,+2
3100 route   txa
3110 ↑       jsr chrget             ; "zur.ck zur Interpreterschleife
3120 prg     jsr $0000              ; "Befehl ausf.hren
3130 ↑       jmp inter              ; "zur.ck zur Interpreterschleife
3140 alt     jsr chrget+6           ; "kein Sonderbefehl, 'normale' Schleife
3150 ↑       jmp execute
3900 ;
4000 tablow  .byt $3e, $72, $d7, $81, $93, $8d
4010 ↑       .byt $2b, $56, $58, $e2, $79, $60
4020 ;
5000 tabhigh .byt $1a, $16, $67, $17, $18, $65
5010 ↑       .byt $64, $18, $69, $69, $6a, $69
5020 ;
6000 vic     lda lowtab-min,x       ; "low-Byte
6010 ↑       sta prg+1              ; "          der '40-Z.-Adresse' nach prg
6020 ↑       lda hightab-min,x      ; "high-Byte
6030 ↑       sta prg+2
6040 ↑       jmp route              ; "weiter s.o.
6050 ;
7000 lowtab  .byt $5a, $a8, $d7, $b7, $8e, $8d
7010 ↑       .byt $2b, $97, $55, $e2, $79, $60
7020 ;
7030 hightab .byt $6b, $61, $67, $62, $66, $65
7040 ↑       .byt $64, $67, $69, $69, $6a, $69
7050 ;
8000 .end
9000 end
60000 zz$="interpret-128.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 5. Assemblerlisting »Interpret«. Nur zur Dokumentation, nicht eingeben

Und da taucht nun gleich eine neue Schwierigkeit auf. Wenn die CPU – egal, welche Bank ausgewählt wurde – im Bereich von $0 bis $1fff nur Bytes aus Bank 0 erreicht, dann können auch die auf Bank 1 in eben diesem Bereich abgelegten Variablen nicht mehr gelesen werden. Und umgekehrt, beim Anlegen neuer Variablen, würde unser Programm in Bank 0 überschrieben, weil der Interpreter nicht wissen kann, daß er, obwohl er in Bank 1 schreiben will, in Wirklichkeit doch in Bank 0 schreibt. Deshalb muß der Anfang des Variablenspeicherbereichs auf $2000 gesetzt werden. Das bedeutet den Verlust von 7 KByte Variablenspeicherbereich (mit »?FRE(1)« überprüfbar!); aber anders geht es leider nicht, wenn man ständiges zeitraubendes Umschalten zwischen den Speicherbänken vermeiden will.

Doch nun zur eigentlichen Interpreterschleife. Zunächst wird ein Zeichen aus dem Basic-Text geholt und geprüft, ob es sich um ein Token der in Frage kommenden Grafikbefehle handelt. Ist das nicht der Fall, so fährt das Programm einfach in der alten Interpreterschleife im ROM fort. Ist ein Grafik-Token gefunden, so testet das Programm als nächstes, ob der VIC oder ob der VDC aktiviert ist. Ist der VIC aktiv, so legt sich das Programm die dem Token entsprechende ROM-Adresse zurecht (selbstveränderlicher Code hinter dem »JSR« von »JSR $0000«), arbeitet die normale ROM-Routine ab und kehrt in die Interpreterschleife zurück. Ist der VDC aktiv, so werden die den neuen Befehlen entsprechenden RAM-Adressen geholt. Assemblerlisting 5 zeigt die fertige Maschinenroutine für Init und Interpreterprogramm.

Der aufmerksame Leser wird sich wohl schon gefragt haben, warum die Grafik-Routinen gerade in den Bereich ab $1300 »gequetscht« wurden. Ein Grund dafür wurde schon genannt: Die Routinen enthalten sehr viele Sprünge in ROM-Unterprogramme, die es erforderlich machen, daß die gesamten 48 KByte ROM-I/O ab $4000 eingeblendet sind. Will man umständliche und vor allem zeitraubende Bank-Umschaltungen (JSRFAR im Kernel $ff6e) vermeiden, so muß man das Programm in den RAM-Bereich unterhalb von $4000 legen. Es soll jedoch möglich sein, auf beiden Grafik-Bildschirmen gleichzeitig zu arbeiten. Dann verbietet es sich auch noch, den

Bereich von $1c00 bis $4000 zu benutzen, da dort der Farb- und der Bildschirmspeicher für die VIC-HiRes-Grafik angeordnet sind. Bleibt als der einzige freie RAM-Bereich der Bereich von $1300 bis $1bff.

Das »Kochrezept«

Wer bis hier aufmerksam gelesen hat, wird gleichsam als Belohnung eine Menge interessanter Details zur Programmierung seines C 128 erfahren haben. Aber auch ungeduldige Leser sollten spätestens hier einhalten, es folgt die Bedienungsanleitung für das Programm »Graphik-80«:

1) Tippen Sie das Basic-Programm »Graphik 80« (Listing 1) sorgfältig ein! Für diese Arbeit sollten Sie sich Zeit nehmen, damit Ihnen das »DATA-Grab« am Ende des Programms nicht zur Falle wird. Wer einen Assembler zur Verfügung hat – ein C 64-Assembler genügt –, der kann auch die drei Assemblerlistings abtippen, die fertigen Maschinenprogramme speichern und dann die Zeilen 5000 bis 5340 und 10000 bis 14070 durch drei BLOAD-Befehle ersetzen. Die abgedruckten Assemblerlistings (Listings 3 bis 5) können mit dem »Profi Ass«-Assembler direkt übernommen werden, bei anderen Assemblern dürfen in der Regel die Basic-Befehle und die Hochpfeile nicht verwendet werden.

2) Speichern Sie das Basic-Programm auf Diskette. Wenn Sie später Änderungen am Grafik-Paket vornehmen wollen, brauchen Sie es wieder.

3) Starten Sie das Programm mit »RUN«! Das Diskettenlaufwerk (1541 oder 1570/1571) muß beim Start des Programms eingeschaltet und mit einer Diskette versehen sein.

4) Wenn keine Fehlermeldungen aufgetreten sind, und Ihnen auch keine Fehler beim Abschreiben der DATA-Zeilen unterlaufen sind, dann befindet sich jetzt das fertige »Graphik-80«-Paket unter dem Namen »graphik-80.m« auf Diskette und fertig initialisiert im Speicher. Mit Hilfe des kleinen Testprogramms (Listing 2), das noch mit abgedruckt ist, können Sie leicht überprüfen, ob alles richtig gelaufen ist. Von jetzt ab brauchen Sie nur noch als erste (!) Programmzeile »bload "graphik-80.m": sys dec ("1303")« einzugeben und das Grafik-Paket ist aktiviert. Alle Grafik-Befehle beziehen sich jetzt auf den 80-Zeichen-Schirm.

(Th. Rumbach/D. Winkler/ev)

Roulette C 128

Hier präsentieren wir Ihnen eines der ersten Spiele im C 128-Modus. Es zeichnet sich durch alle Setzmöglichkeiten aus, die es beim richtigen Roulette auch gibt.

Das Programm »Roulette 128« (siehe Listing) ist ein Basic 7.0 Programm und kann vom C 128 im C 128-Modus aus mit RUN "ROULETTE 128" von Diskette geladen und gestartet werden. Das Programm lädt keine weiteren Programmteile oder Files nach, und es benötigt keine weitere Peripherie. Es werden nur die 40 Zeichen beziehungsweise die hochauflösende Grafik benutzt, ein spezieller Monitor für 80 Zeichen ist also auch nicht erforderlich.

Das Programm kann neben der Tastatur auch über einen Joystick bedient werden, der, falls vorhanden, in Port 2 zu stecken ist. Nach dem Start wird der FAST-Modus aktiviert, um Variablenfelder einzulesen und um den Grafik-Bildschirm zu erstellen (Bild 1). Der Vorgang dauert etwa 15 Sekunden.

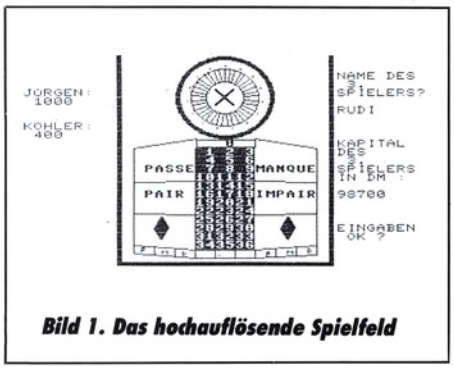

Bild 1. Das hochauflösende Spielfeld

Es können maximal 5 Spieler teilnehmen. Steigt ein Spieler ein, muß er Name und Startkapital eingeben. Nach jedem Spiel können ein oder mehrere Spieler ein- oder aussteigen. Spielt kein Spieler mehr mit, wird das Programm beendet.

Hat ein Spieler sein ganzes Geld verloren, scheidet er automatisch aus. Wenn nun alle Spieler mit den Eingaben fertig sind, wird das Spiel gestartet. Es erscheinen der Name des Spielers, der nun auswählen muß, worauf er wieviel setzt, und die Frage »MENUE ODER TABLEAU« (Tableau = Spieltisch, Bild 1). Jedem Spieler stehen also 2 Wahlmöglichkeiten zur Verfügung, »M« oder »T«. Wird die Taste »M« für Menü gedrückt, wählt der Spieler die Tastatur als Eingabegerät. Dazu werden ihm zuerst alle Sätze angeboten. Nach Wahl eines Satzes kommen auf den Bildschirm die jeweils möglichen Zahlenkombinationen. Mit der »NO-SCROLL«-Taste kann ein eventuelles Herausscrollen aus dem Bildschirm verhindert werden. Der Spieler kann sich dann alle Kombinationen in Ruhe anschauen. Ein weiteres Drücken dieser Taste setzt die Auflistung fort. Jede Zahlenkombination ist mit einer »MENÜ-Zahl« versehen, die anschließend einzugeben ist.

Eine weitere Möglichkeit, dies auszuwählen, besteht darin, über Joystick auf dem Tableau selbst die Auswahl zu treffen. (Bei der Frage »MENUE oder TABLEAU« »T« drücken). Man bewegt dazu einen Chip mit dem Joystick. An der rechten Bildschirmseite werden bei Nichtbewegen des Joysticks der Satz und die Zahlenkombination angezeigt, auf der sich der Chip befindet. Durch Drücken des Feuerknopfes beendet man seine Wahl.

In beiden Fällen (Menü oder Tableau) kommt anschließend die Frage, wieviel man auf diese Zahl beziehungsweise Zahlen setzen will. Die eingegebene Summe muß kleiner gleich dem Kapital des jeweiligen Spielers sein, sonst nimmt das Programm die Eingabe nicht an. Nach Beendigung einer Wahl wird der Spieler gefragt, ob er es bei den bis dahin gewählten Zahlenkombinationen beläßt oder auf noch mehr Zahlen setzen will. Haben sich alle Spieler entschieden, »geht nichts mehr« und die Kugel rollt. Die Zahl, die gewinnt, wird angezeigt, und die Spieler, die auf diese Zahl in irgendeiner Weise gesetzt haben, gewinnen den ihrer Chance entsprechenden Betrag ihres Einsatzes.

Hinweise zum Abtippen

Von Zeile 7000 bis 7499 ist eine Fehlerbehandlungs-Routine eingebaut. Es ist sinnvoll, sie zuerst einzutippen. Durch den Befehl »TRAP 7000« in Zeile 12 wird sie aktiviert.

Grundlagen zu Roulette

Für das Verständnis des Programmes ist es wichtig, wenigstens die Grundbegriffe von Roulette zu kennen.