Von Basic zu Assembler (Teil 2)

Kurze Schleifen sind in Assembler kein Problem mehr. Deshalb wagen wir uns nun an die 16-Bit-Schleifen, wobei uns auch gleich zwei Routinen aus der Firmware entschleiert werden.

Die letzte Folge hatten wir beendet mit der Aussicht, in die einfachen Verzögerungsschleifen nun die Würze von Aufgaben einzubauen. Ein kleines Basic-Programm, das Sie vielleicht verlockt hat, die Entsprechung in Assembler zu schreiben, sollte 128 bunte Zeichen auf den Bildschirm zaubern. Haben Sie es versucht? Wenn ja, dann vergleichen Sie Ihr Ergebnis doch mal mit Listing 1.

In den Zeilen 30, 40 und 160, 170 sehen Sie die Anwendung eines weiteren Pseudobefehls. Das .EQ bewirkt, daß eine bestimmte Speicherstelle mit einem Namen versehen werden kann. Im folgenden braucht man sich nur noch den Namen zu merken, der auch am Ende in der Symboltabelle mit ausgegeben wird. Dadurch wird man bis zu einem gewissen Grad sogar systemunabhängig. Um beispielsweise dieses Programm auf einem VC 20 in der Grundversion laufen zu lassen, muß in Zeile 30 der SCREEN-Wert auf $1E00 und in Zeile 40 der COLOR-Wert auf $9600 geändert werden.

Bevor Sie durch G 5000 aus dem Monitor heraus das Programm starten, löschen Sie am besten zuerst den Bildschirm und fahren den Cursor in eine mittlere Bildschirmzeile, damit er dem Ergebnis des Programmes nicht ins Gehege kommt. Das Programm läuft natürlich auch auf dem C 128 (im C 128-Modus). Allerdings werden hier die Zeichen nur einfarbig, weil man zum Beschreiben des Bildschirmfarbspeichers (mit STA COLOR,Y) noch die Bank umschalten muß, was hier nicht getan wird.

Sie sehen: Das geht in Assembler erheblich schneller als in Basic und eben die Geschwindigkeit in Assemblerprogrammen wird es sein, die uns im 2. Beispiel noch ein wenig beschäftigen wird. Die Aufgabenstellung ist folgende: Ein weißer Ball soll von rechts unten kommend quer über den Bildschirm fliegen nach links oben. Dazu sollen 2 Firmwareroutinen verwendet werden: Eine zum Drucken beliebiger Zeichen und eine andere zum Setzen des Cursors. Die erste ist das normale PRINT in Basic, das als Kernel-Routine BSOUT (manchmal auch CHROUT genannt) durch Assemblerprogramme bei $FFD2 ansteuerbar ist. Das auszudruckende Zeichen muß vor dem Aufruf JSR $FFD2 im Akkumulator enthalten sein. Die andere Routine dient dem Steuern des Cursors. Gibt man in die Speicherstelle 211 ($D3) die gewünschte Spalte und in 214 ($D6) die Zeile des Bildschirmes, an die der Cursor positioniert werden soll, dann lenkt ihn der Aufruf des bei 58640 ($E510) beginnenden Maschinenprogrammes unserer Firmware an diesen Ort.

Alle Randbedingungen werden durch dieses Basic-Programm realisiert:

```
 10 S=211:Z=214:B=58640:S1=40:Z1=20
 20 PRINT CHR$(147)CHR$(5)
 30  GOSUB 100:PRINT CHR$(113)
 40  FOR I=19 TO 0 STEP -1
 50     GOSUB 100:PRINT CHR$(32)
 60     S1=S1-2:Z1=Z1-1
 70     GOSUB 100:PRINT CHR$(113)
 80 NEXT I
 90 PRINT CHR$(154):END
100 POKE S,S1:POKE Z,Z1:SYS B:RETURN
```

In der Schleife wird immer zuerst das zuletzt gedruckte Zeichen gelöscht (sonst hätten wir nicht nur einen Ball, sondern eine Diagonale aus weißen Bällen) und dann nach dem Weitersetzen des Cursors der nächste Ball gezeichnet.

Listing 2 zeigt nun das Äquivalent dazu in Assembler.

```
10  -           .LI 1,4
20  -           .BA $5000
30  -           .EQ SCREEN=$0400;BILDSCHIRMSTART
40  -           .EQ COLOR=$D800;FARBRAMSTART
50  -;*** BEISPIEL 1 ***
60  -;VERSION 8 MIT EINFACHEM JOB
70  -;ZEICHEN AUF BILDSCHIRM ZEIGEN
80  -;
90  -;----- INITIALISIERUNG -----
100 -;
110 -           LDY #$7F        ;DAS IST DEZIMAL 127
120 -;
130 -;----- VERARBEITUNG --------
140 -;
150 -LABEL      TYA
160 -           STA SCREEN,Y
170 -           STA COLOR,Y
180 -;
190 -;----- STEUERUNG -----------
200 -;
210 -           DEY
220 -           DEY
230 -           BPL LABEL
240 -;
250 -;----- AUSGANG -------------
260 -;
270 -           BRK
280 -;
290 -           .SY 1,4
300 -           .ST
```

Listing 1. Unser Beispiel 1 in Assembler: Bunte Zeichen

```
10  -           .LI 1,4
20  -           .BA $5000
30  -           .EQ SPALTE=$D3
40  -           .EQ ZEILE=$D6
50  -           .EQ COUNTZ=$FA
60  -           .EQ COUNTS=$FB
70  -           .EQ CSET=$E510
80  -           .EQ BSOUT=$FFD2
90  -;*** BEISPIEL 2 ***
100 -;BILDSCHIRMAUSGABE MIT FIRMWARE-ROUTINEN
110 -;
120 -;----- VORBEREITUNGEN -----
130 -;
140 -           LDA #$93        ;DEZIMAL 147
150 -           JSR BSOUT       ;BILDSCHIRM LOESCHEN
160 -           LDA #$05
170 -           JSR BSOUT       ;ZEICHENFARBE WEISS
180 -           LDA #$14        ;DEZIMAL 20
190 -           STA ZEILE
200 -           STA COUNTZ      ;SICHERN
210 -           LDA #$27        ;DEZIMAL 39
220 -           STA SPALTE
230 -           STA COUNTS      ;SICHERN
240 -;
250 -;----- VERARBEITUNG -------
260 -;
270 -LABEL      LDA COUNTZ
280 -           STA ZEILE
290 -           LDA COUNTS
300 -           STA SPALTE
310 -           JSR CSET        ;CURSOR SETZEN
320 -           LDA #$71        ;DEZIMAL 113
330 -           JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
340 -           NOP
350 -           LDA COUNTZ
360 -           STA ZEILE
370 -           LDA COUNTS
380 -           STA SPALTE
390 -           JSR CSET
400 -           LDA #$20        ;DEZIMAL 32
410 -           JSR BSOUT       ;ZEICHEN LOESCHEN
420 -;
430 -;----- STEUERUNG ----------
440 -;
450 -           DEC COUNTS
460 -           DEC COUNTS
470 -           DEC COUNTZ
480 -           BNE LABEL       ;HERUNTERZAEHLEN BIS 0
490 -;
500 -;----- ABSCHLUSS ----------
510 -;
520 -           LDA #$9A        ;DEZIMAL 154
530 -           JSR BSOUT       ;ZEICHENFARBE HELLBLAU
540 -           BRK
550 -;
560 -           .SY 1,4
570 -           .ST
```

Listing 2. Ein schneller Flitzer: Beispiel 2

In den Zeilen 30 bis 80 finden Sie wieder den Pseudobefehl .EQ. Mit diesem werden außer den bisher schon besprochenen Speicherstellen (Zeile, Spalte, CSET und BSOUT) auch noch zwei Zähler kreiert: COUNTZ (Zeilerzähler) und COUNTS (Spaltenzähler). Was soll das, werden Sie fragen, warum verwendet man nicht direkt ZEILE und SPALTE? Die Ursache liegt darin, daß BSOUT ebenfalls diese Speicherstellen benutzt und daher keine richtige Zählung mehr stattfinden kann. So zählt $FA und $FB und jedesmal vor Aufruf von CSET wird deren Inhalt in ZEILE und SPALTE übertragen. Wir brauchen natürlich nur einen Zähler für diese Schleife. COUNTS läuft nur nebenher und könnte eigentlich auch in den Schleifenteil »Verarbeitung« geschrieben werden. Die Abbruchoperation in Zeile 480 prüft nur COUNTZ. Mehr Kommentar finden Sie direkt im Listing.

So, nun starten Sie mal das Programm nach dem Assemblieren aus dem Monitor mit G 5000! Sie meinen, da passiert ja gar nichts? Ich kann Ihnen beweisen, daß doch etwas passiert — nur so immens schnell, daß wir nichts davon sehen. Verändern Sie doch mal in Zeile 400 das #$20 (Leerzeichen) zu #$1C (Farbe Rot). Das können Sie auch schnell aus dem Monitor her erreichen durch M 5033 — dort finden Sie am Anfang die 20 — und überschreiben durch 1C ((RETURN)). Wenn Sie nun starten, wird der Ball nicht mehr gelöscht, sondern nur rot gefärbt. Wir erhalten die Diagonale aus roten Bällen. Es geht also doch!

Wir müssen daher das ganze etwas verlangsamen. Dazu ist schon eine Stelle vorgesehen: In Zeile 340 befindet sich ein gänzlich unmotiviertes NOP-Kommando. Dorthin packen wir nun eine Verzögerungsschleife und es ergibt sich Listing 3.

In die Zeilen 335 bis 345 haben wir die Version 6, mit dem Y-Register als Zähler eingefügt. Ein erneuter Start nach dem Assemblieren zeigt uns ein kurzes weißes Aufflackern (falls Sie die Farbe Rot wieder gegen #$20 ausgetauscht haben!). Das war also immer noch zu schnell! Also bauen wir noch eine Verzögerungsschleife ein (Zeilen 346 bis 348 in Listing 4).

Nun sehen wir schon ein wenig mehr, aber wir können uns vorstellen, daß es reichlich ungelenk wäre, nun noch eine dritte, vierte,... Verzögerung einzubauen. Es gibt noch einen anderen Weg, nämlich einfach zwei Verzögerungen ineinander zu verschachteln. Das ist schließlich in Listing 5 geschehen und wenn Sie das nach der Assemblierung starten, dann gehts

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL     LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET        ;CURSOR SETZEN
320  -          LDA #$71        ;DEZIMAL 113
330  -          JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
335  -          LDY #$FF        ;VERZOEGERUNG
340  -MARKE     DEY
345  -          BNE MARKE
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20        ;DEZIMAL 32
410  -          JSR BSOUT       ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 3. Flitzer mit kleinem Handicap

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL     LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET        ;CURSOR SETZEN
320  -          LDA #$71        ;DEZIMAL 113
330  -          JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
335  -          LDY #$FF        ;VERZOEGERUNG
340  -MARKE     DEY
345  -          BNE MARKE
346  -          LDY #$FF
347  -WEITER    DEY
348  -          BNE WEITER
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20        ;DEZIMAL 32
410  -          JSR BSOUT       ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 4. Der doppelt zögernde Flitzer

```
250  -;----- VERARBEITUNG -------
260  -;
270  -LABEL     LDA COUNTZ
280  -          STA ZEILE
290  -          LDA COUNTS
300  -          STA SPALTE
310  -          JSR CSET        ;CURSOR SETZEN
320  -          LDA #$71        ;DEZIMAL 113
330  -          JSR BSOUT       ;GRAFIKZEICHEN DRUCKEN
332  -          LDY #$FF
334  -MARKE     LDX #$FF
336  -WEITER    DEX
338  -          BNE WEITER
340  -          DEY
342  -          BNE MARKE
350  -          LDA COUNTZ
360  -          STA ZEILE
370  -          LDA COUNTS
380  -          STA SPALTE
390  -          JSR CSET
400  -          LDA #$20        ;DEZIMAL 32
410  -          JSR BSOUT       ;ZEICHEN LOESCHEN
420  -;
430  -;----- STEUERUNG ----------
```

Listing 5. Der Flitzer ist voll unter Kontrolle

hübsch langsam. Immerhin wird die innere Schleife 255 x 255mal durchlaufen. Jedesmal nämlich, wenn wir X bis 0 heruntergezählt haben, wird Y dekrementiert und X wieder mit #$FF beladen. Das geht so lange, bis auch Y auf Null heruntergezählt wurde. Wenn Sie in Zeile 332 statt #$FF einen kleineren Startwert eingeben (geht wieder ganz gut vom Monitor aus), läuft der Ball schneller. Damit haben Sie die Geschwindigkeit völlig im Griff.

Außerdem haben wir auf diese Weise die einfachen 8-Bit-Schleifen verlassen, denn diese Verzögerung ist schon eine 16-Bit-Schleife. Auf die und auf die im Listing 2 verwendeten Firmwareroutinen kommen wir nun zu sprechen.

4. 16-Bit-Schleifen

Sehen wir uns zunächst einmal in Basic an, was wir da gemacht haben. Es dreht sich um etwas uns sehr bekanntes: Zwei ineinander geschachtelte Schleifen. Am genauesten entspricht wohl diese Programmsequenz unserer 16-Bit-Verzögerung:

```
100 Y=255
110 X=255
120 X=X-1
130 IF X .. 0 THEN 120
140 Y=Y-1
150 IF Y .. 0 THEN 110
```

Gebräuchlicher wäre allerdings diese Version:

```
100 FOR Y=255 TO 0 STEP-1
110  FOR X=255 TO 0 STEP-1
120  NEXT X
130 NEXT Y
```

Dagegen halten wir unsere Verzögerungsschleife aus dem letzten Assemblerprogramm (Listing 5):

```
             LDY #$FF
LABEL        LDX #$FF
MARKE        DEX
             BNE MARKE
             DEY
             BNE LABEL
             ...
```

Diese Schleife zählt das X-Register so oft eine ganze Page (minus 1, also jeweils 255mal) durch, wie es das Y-Register angibt, hier also 255mal. Insgesamt finden daher 255*255 = 65025 Durchläufe statt. Um ganze Pages, also 256 Zählungen zu erreichen, lädt man ins X-Register einfach 0 ein. Der DEX-Befehl sorgt dann noch vor der BNE-Prüfung für einen Unterlauf auf $FF.

Deutlich wird Ihnen sicher, daß wir — im Gegensatz zur einfachen Schleife — hier einen Multiplikationseffekt zu beachten haben. Die Anzahl der Durchläufe setzt sich zusammen aus:

Y-Startwert * X-Startwert

Das ist auch ganz akzeptabel, solange man die gewünschte Durchlaufzahl aus zwei Faktoren zusammensetzen kann. Soll ein Job beispielsweise 1000mal ausgeführt werden, dann gibt es mehrere Möglichkeiten, denn

```
1000 = 8 * 125
     = 4 * 250
     =10 * 100
```

Wir könnten dann unsere Job-Schleife schreiben:

```
             LDY #$04
LABEL        LDX #$FA
MARKE        Job-Befehle
             DEX
             BNE MARKE
             DEY
             BNE LABEL
             ...
```

Abgesehen davon, daß es doch ein wenig aufwendig ist — besonders bei einer nicht festgelegten Anzahl von Durchläufen — jedesmal eine Aufspaltung in zwei Faktoren vorzunehmen: Was tun wir bei Primzahlen? 997 Jobs beispielsweise lassen sich in solch einer Doppelschleife nicht bearbeiten (997 ist eine Primzahl, das bedeutet, diese Zahl ist nicht in Faktoren zerlegbar).

Im Prinzip gibt es für solche Fälle zwei Lösungen:

— Entweder stellt man fest, daß es gleichgültig ist, ob nun — um bei unseren Beispielen zu bleiben — 1000, 1024 oder 997 Durchläufe stattfinden. Es ist häufig der Fall, daß dadurch nicht mehr Schaden angerichtet wird als der zusätzliche Zeitbedarf für 27 Durchläufe (bei 1024 anstelle von 997). In diesem Fall legt man den Anfangswert der inneren Schleife einfach grundsätzlich auf 0 fest (arbeitet also genau eine Page darin ab) und variiert nach Bedarf den Startwert der äußeren Schleife (dort wird nun also 4 eingetragen).
— Oder aber — wenn's genau drauf ankommt — wir müssen zwei Schleifen einrichten: Für die ganzen Pages eine Doppelschleife und für den Rest eine einfache. Genau das geschieht in einer sehr nützlichen Routine unserer Firmware, der BLTUC-(oder auch Blockverschiebe-) Routine, auf deren Verstehen wir bis zur nächsten Folge hinarbeiten werden. Sie können ja schon mal mittels SMON in den Speicher sehen: Von $A3BF bis A3FA ist dieses Programm zu finden.

Bevor wir uns an diese schwierigeren Sachen wagen, wollen wir uns aber noch ein wenig mit Fragen der Schleifenstruktur befassen. Zunächst kann nur relativ selten auf die beiden Indexregister als Zähler zurückgegriffen werden. Man muß meistens zwei Speicherstellen dazu verwenden. Außerdem kann man natürlich ebensogut in den Schleifen aufwärts zählen. Das soll im folgenden Beispiel beides geschehen, wo wir den Bildschirminhalt invertieren wollen. Das geschieht einfach durch Setzen des Bit 7 des Codes in jeder Bildschirmspeicherstelle (wir machen das durch EOR $80). Das hat den Vorzug, daß ein zweiter Durchlauf des Programmes wieder den Ausgangszustand des Bildschirmes herstellt. Zuerst sollen Sie eine etwas schwerfällige, aber überschaubare Form des Programmes kennenlernen (Listing 6):

```
Initialisierung:
  4000      LDA #$00        Die Bildschirmadresse wird
  4002      STA $FA         in den Vektor $FA/FB geschrieben.
  4004      LDA #$04        Index auf Null.
  4006      STA $FB
  4008      LDY #$00
Job:
  400A      LDA ($FA),Y     Code in Akku
  400C      EOR #$80        invertieren und
  400E      STA ($FA),Y     zurückschreiben.
Steuerung:
  4010      INC $FA         LSB hochzählen
  4012      BNE $400A       und weiter Job ausführen, bis ein Überlauf von 255 auf 0 stattfindet.
  4014      INC $FB         dann MSB erhöhen
  4016      LDA $FB         und prüfen, ob
  4018      CMP #$08        Endadresse erreicht ist.
  401A      BNE $400A       Falls noch nicht, erneut zur Jobschleife
Ausgang:
  401C      BRK             Sonst Ende mit Registeranzeige.
```

Listing 6. Invertieren des Bildschirms

Hier wurden — auf höchst plumpe Weise — vier ganze Pages bearbeitet. Eine andere Lösung wäre es, anstelle von $FA in der Zeile 4010 das Y-Register zu erhöhen (mittels INY). Es würde dann sowohl als Index als auch als Zähler dienen. (In unserer Version hatte es ja nur eine Alibifunktion für die spezielle Art der Adressierung der Bildschirmspeicherzellen). Eleganter kann das Problem gelöst werden mit einer Technik, die Florian Müller in seinem Artikel »Effektives Programmieren in Assembler« (64'er Sonderheft 8, 1985, S.22) vorstellt. Dabei werden $FA und $FB nicht mehr als Zähler verwendet, sondern dem Y-Register kommt wieder die Doppelfunktion zu als Index und als Zähler der inneren Schleife. Das X-Register ist Zähler der äußeren Schleife. In der inneren wird Y hoch-, in der äußeren Schleife X heruntergezählt. Das Ergebnis davon ist: Das Programm wird kürzer und auch schneller (Listing 7).

Es stört uns immer noch manchmal, daß wir — statt nur bis $07E7 (denn das ist dezimal 2023) — bis $07FF invertieren. Bevor wir in der nächsten Folge die oben erwähnte Variante ergründen, die in der BLTUC-Routine verwendet wird, soll Ihnen noch eine weitere Möglichkeit vorgestellt werden, die im SMON und neuerdings auch von F. Müller (siehe oben) gezeigt worden ist. Da geht's recht trickreich zu.

```
Initialisieren
  4000      LDA #$00        LSB Bildschirm
  4002      STA $FA         in Vektor und
  4004      TAY             Index = 0.
  4005      LDA #$04        MSB in Vektor
  4007      STA $FB         schreiben und
  4009      TAX             Zähler für die pages auf 4.
Job:
  400A      LDA ($FA),Y     Dasselbe wie
  400C      EOR #$80        wir es vorhin
  400E      STA ($FA),Y     hatten.
Steuerteil:
  4010      INY             Index (Zähler)+1
  4011      BNE $400A       wenn noch kein Überlauf, erneut Job ausführen.
  4015      DEX             sonst page-Zähler herunterzählen.
  4016      BNE $400A       Wenn noch nicht 0, dann wieder Jobbearbeitung.
Ausgang:
  4018      BRK             sonst wieder Ende mit Registeranzeige.
```

Listing 7. Verbesserte Form von Listing 6

Wieder wird pro forma das Indexregister Y initialisiert wegen der speziellen Art der Adressierung (Listing 8).

```
Initialisieren:
  4000 LDA #$00  Bildschirmstart
  4002 STA $FA     in Vektor $FA/FB
  4004 LDA #$04
  4006 STA $FB
  4008 LDY #$00

Job:
  400A      LDA ($FA),Y     Das kennen wir
  400C      EOR #$80        nun schon.
  400E      STA ($FA),Y
Steuerung:
  4010      INC $FA         Erhöhen des LSB
  4012      BNE 4016        Wenn kein Überlauf, erfolgt ein Sprung.
  4014      INC $FB         Sonst auch Erhöhen des MSB.
  4016      LDA $FA         Das LSB wird nun
  4018      CMP #$E8        verglichen mit dem MSB der Endadresse + 1. Dabei findet die Resultatanzeige in den Flaggen (N,Z,C) statt.
  401A      LDA $FB         Nun wird das MSB der Adresse in den Akku geladen und
  401C      SBC #$07        das MSB der Endadresse subtrahiert. Die Carryflaggge ist gesetzt, wenn die Adresse in $FA/FB gleich der Endadresse+1 ($07E8) geworden ist.
  401E      BCC 400A        Solange das noch nicht der Fall ist, wird zum Job zurückverzweigt.
Ausgang:
  4020      BRK             Sonst aber Ende mit Registeranzeige.
```

Listing 8. Die trickreichste Version

Natürlich wird diese Doppelschleife durch die ständigen Rechnungen im Steuerteil relativ langsam, weshalb es doch lohnt, auch andere Wege zu untersuchen.

5. Zwei Firmware-Routinen

Kommen wir nun — wie versprochen — noch auf die beiden vorhin verwendeten Routinen zurück, die sich im oberen ROM-Bereich unseres Computers befinden. Die eine davon ($FFD2) ist mittlerweile schon vielen recht geläufig. Sie dient dazu, ein im Akku enthaltenes Zeichen an ein vorher definiertes Gerät auszugeben. Der Unterschied zwischen beiden Routinen ist, daß CHROUT (also $FFD2) sich im sogenannten Kernel-Bereich befindet, die andere (PLOTK $E510) aber nicht. Was ist denn nun das besondere am Kernel-Bereich? Es handelt sich um eine Tabelle von 39 JMP-Befehlen, für die Commodore garantiert, daß sie in allen Computerversionen an der gleichen Stelle liegt und gleiche Funktionen beinhaltet. Sollten Sie also im Besitz eines VC 20 oder eines C 128 sein: Sie können die gleiche Einsprungadresse für CHROUT benutzen wie ein C 64-Programmierer. Zwar enthält beispielsweise die Kernel-Sprungleiste des C 128 wesentlich mehr Möglichkeiten als die des VC 20, aber alle im VC 20 gültigen Einsprünge behalten auch hier ihre Bedeutung. Leider existiert diese Möglichkeit des Kernel nur für relativ wenige Verwendungszwecke. Wer beispielsweise Fließkommaoperationen in Assembler zu programmieren hat, sucht oft ziemlich verzweifelt im ROM eines neuen Computers nach den dazu passenden Firmware-Routinen.

Alle Kernel-Routinen verlangen eine festgelegte Bearbeitungsweise:
a) Vorbereitungen treffen
b) Routinenaufruf
c) Fehlerabfrage und -behandlung

Damit hätten wir die Vorrede hinter uns und können uns dem CHROUT-Programm zuwenden, das wir an dieser Stelle in seiner eingeschränkten Funktion betrachten, nämlich zur Ausgabe des Akku-Inhaltes auf dem Bildschirm. Falls Sie eine detaillierte Schilderung weiterer Anwendungsmöglichkeiten suchen

sollten: Im Assembler-Kurs (64'er Sonderheft, Ausgabe 8/85, Seite 33 und ab Seite 39) finden Sie beispielsweise die Ausgabe auf den Drucker.

Name **CHROUT** (auch BSOUT)	
Zweck	Ausgabe eines Zeichens
Adresse	$FFD2, 65490
Vorbereitungen	(CHKOUT,OPEN)
Parameter	
Eingabeort	Akku
Eing.Format	ASCII
Ausgabeort	spezifiziertes Gerät
Ausg.format	–
Fehler	0
Stapelbedarf	8
Register	Akku

CHROUT ist nun freundlicherweise so geschaffen worden, daß von den Vorbereitungen lediglich das Zeichen in den Akku zu bringen übrigbleibt, falls man nur die Bildschirmausgabe wünscht. CHROUT ist zwar ein enorm vielseitiger, aber leider auch etwas langsamer Geselle. Das liegt daran, daß CHROUT gewissermaßen als die eierlegende Wollmilchsau konstruiert wurde, also fast alles kann. Damit sind aber endlos viele Prüfungen und Abfragen verbunden, die man sich durch Verwenden anderer Routinen — die lernen Sie noch kennen — ersparen kann.

Nun zur zweiten Adresse $E510, der PLOTK-Routine. Dies ist nur eine der möglichen Einsprungadressen dazu. Es handelt sich nicht um eine Kernel-Routine: Prompt findet sich auch in dem dazugehörigen Programm an einer anderen Einsprungstelle ein Unterschied bei verschiedenen C 64-ROMs, der uns aber nicht zu kümmern braucht.

Diese letzte Angabe werden Sie nicht bei allen beschriebenen Routinen finden. Manchmal ist der Irrweg, dem man durch das ROM zu folgen hat, so komplex, daß ich Ihnen empfehle, selbst mal per SMON (Trace-Kommandos) durchs Labyrinth zu gehen.

In der nächsten Folge sollen Sie dann die BLTUC-Routine als Beispiel für eine 16-Bit-Schleife aus unserer Firmware kennen- und benutzenlernen.

(Heimo Ponnath/gk)

Name **PLOTK**	
Zweck	Cursor setzen
Adresse	$E510, 58640
Vorbereitungen	Zeile in 214, Spalte in 211
Parameter	Übergaben spielen hier keine Rolle.
Fehler	spielen nur bei Kernel-Routinen eine Rolle.
Stapelbedarf	2
Register	Akku, X, Y

Weitere Speicherstellen, die durch die Routine beschrieben werden können: 209, 210, 213, 217 (alle als Dezimalzahlen).

Nr.	Text	Bedeutung
0	BREAK	Während des Programmes wurde die RUN/STOP-Taste gedrückt
1	TOO MANY FILES	Man kann maximal 10 offene Files einrichten
2	FILE OPEN	Ein bereits geöffneter File wird nochmal geöffnet
3	FILE NOT OPEN	Auf einen noch nicht geöffneten File sollte zugegriffen werden
4	FILE NOT FOUND	Der geforderte File ist nicht verfügbar
5	DEVICE NOT PRESENT	Das angesprochene Gerät zeigt keine Reaktion
6	NOT INPUT FILE	Aus einem Schreibfile kann nicht gelesen werden
7	NOT OUTPUT FILE	In einem Lesefile kann nicht geschrieben werden
8	MISSING FILE NAME	Bei Operationen, die einen Filenamen erfordern, fehlt dieser
9	ILLEGAL DEVICE NUMBER	Das versuchte Kommando ist beim angesprochenen Gerät nicht möglich

Tabelle. Fehlernummern und ihre Bedeutung. Die Nummern findet man nach Aufruf von Kernel-Routinen bei gesetztem Carry im Akku.