Von den Kleinen auf die Großen

Für einen so erfolgreichen Computer wie den Commodore 64 dauert es naturgemäß nicht lange, bis auch entsprechend viel gute Software zur Verfügung steht. Viele von diesen Programmen, die nicht die speziellen Eigenschaften des C 64 benutzen, könnten ohne weiteres auf den CBM-Geräten der 3-, 4- und 8000-Serie laufen, wenn Commodore nicht überflüssigerweise den Beginn des Basic-RAM-Bereichs geändert hätte.

Während jedoch der C 64 und auch der VC 20 in der Lage sind, auf CBM-Geräten erstellte Basicprogramme selbsttätig an das eigene System anzupassen (siehe dazu den entsprechenden Artikel in dieser Ausgabe), besitzen die CBM-Geräte diese Fähigkeit nicht. Es ist vielmehr Aufgabe des Anwenders die Programme anzupassen, will er etwa C 64-Programme auf den CBMs laufen lassen.

Im Heft 9/83 der Zeitschrift mc wurde eine einfache Möglichkeit vorgestellt, um auf C 64 erstellte Programme auch für die CBM-Geräte lauffähig zu machen. Es werden dabei mit Hilfe einer Zählschleife die Inhalte der Speicherstellen ab 2048 (Beginn des C 64-Basic-Bereichs) bis Programmende auf die Speicherplätze ab 1024 (Basic-Beginn für die CBMs) transferiert.

Diese Prozedur dauert je nach Länge des zu transferierenden Programms mitunter ein paar Minuten. Aus diesem Grund benutzte ich für die Umwandlung von C 64-Programmen ein eigens dafür erstelltes Maschinenprogramm. Ein versehentlich in den Maschinenprogramm-Bereich gePOKEter Wert sorgte eines Tages dafür, daß statt des ganzen Programms nur ein Block (256 Byte) transferiert wurde. Erstaunlich genug, das (ansonsten nicht lauffähige) Programm ließ sich anstandslos listen!

Diese zunächst überraschende Tatsache hat eine einfache Erklärung: Die Vorwärtszeiger (die ersten beiden Byte einer jeden Basic-Zeile) geben den Beginn der nächsten Basic-Zeile absolut an, das heißt es spielt keine Rolle, ob dieser weniger als 80 Byte oder ein ganzes KByte weiter liegt — die Zeile wird immer richtig angesprungen. Daß das Programm dann nicht auch noch lauffähig ist, liegt daran, daß der Basic-Interpreter, solange keine Sprunganweisungen vorkommen, sich nicht nach den Zeilenzeigern richtet, sondern die Zeilen byteweise in ihrer natürlichen Reihenfolge abarbeitet. Stößt er dabei auf eine Lücke zwischen den Zeilen, so kann er damit nichts anfangen, das System stürzt ab.

Folgende Befehlssequenz, etwa in Direktmodus eingegeben, liefert zum Beispiel die Anfangsadressen der Zeilen eines Basic-Programms:

```
x=1025: printx;: fori=1to1000: x=peek(x) + peek(x+1)
*256: printx;: ifx<>0thennext
```

Es wurde dabei vorausgesetzt, daß das vorliegende Programm nicht mehr als 1000 Zeilen Umfang hat. Will man die Arbeitsweise des Betriebssystems erkennen, so kann man in der obigen Zahlenfolge in eine der dort stehenden Adressen (jede Adresse: Zwei Byte in der Form, Adresse = LSB + MSB*256) statt der nächsten eine etwas später kommende Adresse einPOKEn. Alle dazwischen liegenden Zeilen verschwinden dann aus dem Listing, sie werden aber bei RUN vom Interpreter abgearbeitet. Die »verschwundenen« Zeilen werden wieder sichtbar, sobald eine neue Zeile mit RETURN »übernommen« wird.

Nehmen wir uns die Zeit, um uns an einem Beispiel klar zu machen, was eigentlich im letzten Absatz gemeint ist. Angenommen die Zeilen eines Basicprogramms ergeben die Zahlenfolge: 1025 1057 1092 1149 1226 1286 1350 1412 1465 1500

Das sind die Adressen, an denen jeweils eine Basic-Zeile beginnt, und zwar steht in den Speicherstellen 1025 und 1026 zusammen die Adresse 1057, in 1057 und 1058 die Adresse 1092 und so weiter. Wenn wir nun zum Beispiel in die Adresse 1092 (und 1093) etwa die Adresse 1350 einPOKEn könnten, so würden die Zeilen dazwischen vom Listing verschwinden, der Interpreter würde sich aber weiterhin durchlaufen und abarbeiten. Wie geht das POKEn vor sich? Wenn Ihr System den Befehl »doke« kennt, etwa durch eine Erweiterung wie Exbasic, so geben Sie einfach ein: doke1092,1350. Wenn nicht, so muß man sich erinnern, daß eine Adresse gleich dem Ausdruck MSB*256+LSB ist, wobei MSB und LSB für Most Significant (höherwertiges) Byte und Least Significant (niederwertiges) Byte steht. Es ist in unserem Beispiel: msb=int(1350/256): lsb=1350—msb*256.

msb und lsb sind hier richtige numerische Variablen und müssen (das gilt für die ersten zwei Zeichen, die allein relevant sind) ungeSHIFTet eingegeben werden! Sie geben jetzt ein: poke1092,lsb:poke1093,msb

Um ein C 64-Programm auf einem CBM listen zu können, ist es also nicht erforderlich, erst das ganze Programm zu verschieben, es genügt nur ein Teil davon. Genauer: ein Teil, der

```
1 print"■"tab(10)"■■■■■transfer C64 -> cbm■■■":print"aktivieren mit ■sys 960!■"
4 fori=0to22:readx:poke960+i,x:next:new:rem verschieb.c64->cbm
5 data162,0,189,1,8,157,1,4,232,208,247,165,43,56,233,4,133,43,133
6 data45,133,47,96
```

Dr. Kaisar Petanides

Listing des kurzen Programms für die Umsetzung von C 64-Programmen auf CBM-Computer

mindestens eine Zeile enthält! Der Grund liegt darin, daß das System erst das Ende einer Zeile, gekennzeichnet durch eine 0, gefunden haben muß, bevor der Sprung in die nächste Zeile erfolgt.

Vorgehen: Man gibt im Direktmodus ein:
fori=1025to1111:pokei,peek(i+1024):next:poke43,peek (43)-4

Der Wert 1111 aus mnemonischen und Sicherheitsgründen (1111 - 1024 > 80, Anzahl der Zeichen in einer Basic-Zeile). Die »Verschiebung« dauert etwa eine Sekunde.

Man listet einige Zeilen des Programms, setzt den Cursor auf die erste Zeile und übernimmt sie mit RETURN. Dadurch werden auch alle anderen Zeiger richtig gesetzt, wobei das Betriebssystem gleichzeitig die einzelnen Zeilen lückenlos in steigenden Nummern aneinanderfügt. (Das ist übrigens auch der Grund, warum es zwar prinzipiell möglich, jedoch nicht einfach ist, etwa relokatible Maschinenprogramme zwischen den Basic-Zeilen zu verstecken. Über die Möglichkeit, Textteile in Basic-Zeilen zu verstecken, werden wir bei anderer Gelegenheit reden). Das Programm ist jetzt auch auf CBM-Geräten lauffähig, von systemgebundenen Befehlen natürlich abgesehen.

Aber auch diese Prozedur ist zeitraubend, wenn mehrere Programme von einem System auf das andere übertragen werden müssen. Das kleine Basic-Programm (siehe Listing) namens »transfer« wäre in diesem Fall eine bessere Alternative: Es erzeugt ein Maschinenprogramm ab Adresse 960 und löscht sich anschließend selbst. Das Maschinen»programm« bleibt, so lange der Computer nicht ausgeschaltet wird, in diesem Bereich resistent, verschiebt beim Abruf mit »sys 960« einen einzigen Block des C 64-Programms ab Adresse 2049 auf Adresse 1025 aufwärts und setzt auch die Zeiger für den Beginn der Variablen richtig.

Vorgehen: Das Basicprogramm »transfer« laden und mit RUN starten. Anschließend das zu verschiebende Programm laden, »sys 960« eintippen und mit RETURN abschließen. Dadurch werden die ersten 256 Byte des Basicprogramms »herunter« geholt. Das verschobene Programm läßt sich jetzt listen. Cursor auf die erste Zeile setzen und mit RETURN übernehmen. Gegebenenfalls eine eigene REM-Zeile als Zeile mit der niedrigsten Nummer übernehmen!

(K. Petanides/rg)

User-Port-Display

Mit dem User-Port-Display lassen sich viele Steuerungsprobleme lösen. Dabei ist es manchmal wichtig zu erkennen, ob die gewünschten Daten anliegen. Dieses Problem löst das folgende Programm.

Nachdem ich mir im Sommer '83 einen C 64 zugelegt hatte, begann ich mich als vorheriger Nur-Elektroniker sofort für die Hardware und den User-Port des Computers zu interessieren. Während einiger Steuerexperimente kam mir die Idee zu diesem Programm.

Es eignet sich gut zum Austesten eigener Ein/Ausgabeoperationen. Möchte man zum Beispiel fremde Drucker anschließen, mit anderen Computern (zum Beispiel anderen C 64) kommunizieren oder irgendwelche Steuerungsprobleme lösen, so kann man mit Hilfe dieses Programmes jederzeit (auch während der Basic-Programmbearbeitung) erkennen, ob am User-Port die gewünschten Daten anliegen.

Die Routine ist in Maschinensprache geschrieben, das Listing zeigt den Basic-Lader. Nach dem Starten des Programms wird in der rechten oberen Ecke des Bildschirms nach einmaligem Aufruf ständig binär der momentane Zustand des Datenregisters angezeigt.

Mit SYS 49309 setzt das Programm den IRQ-Vector auf die eigentliche Anzeigenroutine.

Mit RUN/STOP-RESTORE wird der Zeiger wieder zurückgesetzt.

(Jan Legenhausen/gk)

```
10 rem ******************************
15 rem ***   userport-display     ***
20 rem ***                        ***
30 rem *** (c) by jan legenhausen ***
40 rem ***                        ***
42 rem ***        nocken 11       ***
44 rem ***                        ***
46 rem ***    56 wuppertal 11     ***
48 rem ***                        ***
50 rem ***     fuer den c-64      ***
55 rem ******************************
60 rem *** start mit sys (49309)  ***
70 rem ******************************
100 data 169, 48, 157, 30, 4, 169, 0, 15
7, 30, 216, 173, 1
110 data 221, 234, 96, 169, 192, 141, 1,
 192, 173, 1, 221, 162
120 data 0, 160, 46, 140, 0, 192, 41, 12
8, 208, 98, 32, 9
130 data 192, 232, 160, 59, 140, 0, 192,
 41, 64, 208, 85, 32
140 data 9, 192, 232, 160, 72, 140, 0, 1
92, 41, 32, 208, 72
150 data 32, 9, 192, 232, 160, 85, 140,
0, 192, 41, 16, 208
160 data 59, 32, 9, 192, 232, 160, 98, 1
40, 0, 192, 41, 8
170 data 208, 46, 32, 9, 192, 232, 160,
111, 140, 0, 192, 41
180 data 4, 208, 33, 32, 9, 192, 232, 16
0, 124, 140, 0, 192
190 data 41, 2, 208, 20, 32, 9, 192, 232
, 160, 137, 140, 0
200 data 192, 41, 1, 208, 7, 32, 9, 192
210 data 76, 49, 234, 234, 169, 49, 157,
 30, 4, 169, 0, 157, 30, 216, 173, 1
220 data 221, 108, 0, 192, 169, 24, 141,
 20, 3, 169, 192, 141
230 data 21, 3, 96
2000 fors=49161to49152+167:reada:zz=zz+a
:pokes,a:next
2100 ifzz<>17028thenprint"fehler in data
-zeilen !":stop
2110 sys49309
ready.
```

Der Basic-Lader des Programms »User-Port-Display«