Memory Map mit Wandervorschlägen

Teil 15

Bei unserer Wanderung durch die Speicherlandschaft, treffen wir heute auf die Speicherzellen 646 bis 658. Sie sind verantwortlich für die Zeichenfarbe und für die Tastenwiederholung beziehungsweise Tastenwiederholgeschwindigkeit.

Die Speicherzellen, die wir heute behandeln, sind für all diejenigen interessant, die gerne Spiele programmieren oder Bewegung auf den Bildschirm bringen wollen. Denn neben der Tastaturverriegelung und Tastenwiederholung beziehungsweise Tastenwiederholgeschwindigkeit werde ich heute auch die Speicherzellen besprechen, die für die Tastaturdecodierung und Zeichenfarben verantwortlich sind.

FARBE	CODE	ASCII	TASTEN	FARBE	CODE	ASCII	TASTEN
schwarz	0	144	CTRL+1	orange	8	129	CBM+1
weiß	1	5	CTRL+2	braun	9	149	CBM+2
rot	2	28	CTRL+3	hellrot	10	150	CBM+3
lila	3	159	CTRL+4	dunkelgrau	11	151	CBM+4
purpur	4	156	CTRL+5	mittelgrau	12	152	CBM+5
grün	5	30	CTRL+6	hellgrün	13	153	CBM+6
blau	6	31	CTRL+7	hellblau	14	154	CBM+7
gelb	8	158	CTRL+8	hellgrau	15	155	CBM+8

Tabelle 1. Tabelle der Farben und ihrer Codes beziehungsweise Tasten

Adresse 646 ($286)

Aktuelle Farbe der Zeichen (Vordergrundfarbe)

Um ein bestimmtes Zeichen auf den Bildschirm zu drucken, muß vom Betriebssystem erstens der Bildschirmcode des Zeichens in den Bildschirmspeicher und zweitens der Codewert der gewünschten Farbe in den Farbspeicher gebracht werden.

In der Speicherzelle 646 steht immer der Codewert derjenigen Farbe, die gerade eingestellt ist. Immer wenn ein PRINT-Befehl gegeben wird, holt das Betriebssystem den Farbwert aus der Zelle 646 und bringt ihn in den Farbspeicher, und zwar an den entsprechenden Platz, wo gerade gePRINTet werden soll. Der Codewert in der Zelle 646 kann auf drei Arten eingestellt werden:

— Drücken der CTRL-Taste gleichzeitig mit einer der Farbtasten 1 bis 8. Beim C 64 kommen noch weitere acht Farben dazu durch Drücken der Commodore-Taste anstelle der CTRL-Taste.

— PRINT-Befehl gefolgt vom ASCII-Codewert der Farbe innerhalb von Gänsefüßen.

— POKEn der Farbcodes 0 bis 7 (beim C 64 0 bis 15) direkt in die Speicherzelle.

Innerhalb eines Programms ist das POKEn in Zelle 646 wohl die eleganteste Methode (Tabelle 1).

Als Beispiel möge dieses kleine Programm dienen:

```
10 FOR X=0 TO 7
20 POKE 646,X
30 PRINT "A";
40 NEXT X
50 GOTO 10
```

Wer mehr über Vordergrund- und Hintergrundfarben erfahren will, der lese den nebenstehenden Texteinschub »Bunte Zeichen und bunter Hintergrund« auf Seite 147.

Adresse 647 ($287)

Zeichenfarbe unter dem Cursor

Das Blinken des Cursors wird dadurch erzeugt, daß das Zeichen auf der Stelle des Bildschirms, auf der er gerade steht (meistens ist es eine Leerstelle), dauernd von »normal« auf »revers« (oder »invertiert«) und zurück geschaltet wird. Die reverse Darstellung benutzt dabei die Farbe des Zeichens.

Genauso, wie sich der Computer in der Speicherzelle 206 das Zeichen merkt, mit dem er gerade blinkt, um beim Weiterwandern dieses Zeichen in seiner »normalen« Form auf dem Bildschirm zurückzulassen, merkt er sich die Farbe dieses Zeichens in der Speicherzelle 647.

Adresse 648 ($288)

Beginn des Bildschirmspeichers

In dieser Speicherzelle steht eine Zahl, die als High-Byte dem Betriebssystem angibt, ab welcher Speicherzelle der Bildschirmspeicher beginnt.

Nach einem Kaltstart (nach dem Einschalten oder nach dem Drücken der RESET-Taste) steht hier eine 4, das ergibt als Anfangsadresse 1024 (= 4*256). Beim VC 20 ohne Erweiterung steht dort eine 30. Daraus folgt, daß die Anfangsadresse bei 7680 (= 30*256) liegt.

Der Bildschirmspeicher hat keinen absolut festen Platz. Innerhalb gewisser Grenzen kann er durch Verändern des Inhalts der Speicherzelle 53272 (36869 beim VC 20) verschoben werden. Nähere Informationen finden Sie im 64'er Sonderheft 2/86 »Wie wär's mit:...«. Wichtig dabei ist, daß nach dem Verschieben der Inhalt der Speicherzelle 648 entsprechend geändert wird, damit auch das Betriebssystem die Verschiebung berücksichtigt.

Umgekehrt kann aber dem Betriebssystem durch Ändern der Zahl in der Speicherzelle 648 mitgeteilt werden, daß es Zeichen in einen Speicherbereich bringen soll, der außerhalb des »offiziellen«, durch die Speicherzelle 53272 (36869) festgelegten Bildschirmspeichers liegt.

Zwei Beispiele sollen das verdeutlichen. Der PRINT-Befehl macht letztlich nichts anderes, als viele Zahlen in den Bildschirm- und den Farbspeicher zu POKEn. Wenn nun der Zeiger in Zelle 648 verschoben wird, kann man mit einem PRINT-Befehl eine beliebige Zeichenkette außerhalb des Bildschirmspeichers abspeichern. Auf die gleiche Weise kann man beim C 64 Sprites mit einem PRINT-Befehl abspeichern, ohne mit READ viele lästige DATA-Zeilen lesen zu müssen.

Adresse 649 ($289)

Maximale Länge des Tastaturpuffers

Der Tastaturpuffer belegt, wie schon besprochen, die Speicherzellen 631 bis 640. Er kann darin maximal 10 Zeichen zwischenspeichern.

Der Inhalt der Speicherzelle 649 legt fest, wieviel Zellen des Tastaturpuffers verwendet werden sollen, eine Zahl also, die normalerweise zwischen 0 und 10 liegen sollte. Die 10 ist übrigens der Wert, welcher nach dem Einschalten vom Betriebssystem in die Zelle 649 gebracht wird.

Diese Zahl wird immer mit dem Inhalt der Speicherzelle 198 verglichen, der die aktuelle Anzahl der Zeichen im Tastaturpuffer angibt. Ist die Differenz der beiden Zahlen gleich Null, dann können keine weiteren Zeichen eingegeben werden.

Es ist naheliegend, daß durch Verändern der Zahl in Zelle 649 die Länge des Tastaturpuffers verändert werden kann. Der eine Extremfall ist 0:
POKE 649,0 schaltet die Tastatur aus. Nichts geht mehr.

Das kann bei Programmen oder Spielen, die durch falsches oder zeitlich unpassendes Drücken von Tasten gestört werden, recht nützlich sein. Einschalten kann man dann die Tastatur nur mit RUN/STOP und RESTORE.

Auch eine Erhöhung der Zahl in 649 über 10 hinaus ist möglich. Die Zeichen werden halt nur über die dafür reservierten Speicherzellen 631 bis 640 hinaus in Zellen geschrieben, die eigentlich eine andere Funktion haben. Bis zur Speicherzelle 645 geht das normalerweise ohne Probleme, da die betroffenen »fremden« Adressen nur direkt nach dem Einschalten des Computers gebraucht werden.

Probieren Sie es aus, indem Sie zuerst eine Zeitschleife laufen lassen und in dieser Zeit etwa 20 Tasten drücken. Am Ende der Zeitschleife wird der Inhalt des Tastaturpuffers ausgedruckt, und Sie sehen in der Tat 15 der eingegebenen Zeichen:

```
POKE 649,15
FOR X=0 T0 10000:NEXT X
QWERTYUIOPASDFGHJKL
```

Auf dem Bildschirm erscheinen die Zeichen Q bis G.

Wenn Sie die Zahl in 649 noch

weiter erhöhen, dringen Sie in die Zellen 646 und 647 ein und diese bestimmen bekanntlich die Zeichenfarbe. Wenn Sie aber eine unbeabsichtigte und unkontrollierbare Farbänderung nicht stört, können Sie den Tastaturpuffer auf 17 Zeichen vergrößern. Ab 18 Zeichen stürzt der Computer ab.

Adresse 650 ($28A)

Flagge für Tastenwiederholung

Normalerweise steht in dieser Speicherzelle eine 0. Das bedeutet, daß die Funktion der Cursor-Tasten, der Leertaste und der INST/DEL-Taste wiederholt wird, solange die entsprechende Taste gedrückt wird.

Durch Verändern der Zahl in der Speicherzelle 650 kann diese Wiederholfunktion sowohl auf alle Tasten ausgedehnt oder für alle Tasten gesperrt werden.

POKE 650,0 ist der Normalzustand, Wiederholfunktion für Cursor-, Leer- und INST/DEL-Taste.

POKE 650,64 schaltet Wiederholfunktion für alle Tasten aus POKE 650,128 erweitert Wiederholfunktion auf alle Tasten.

Adresse 651 ($28B)

Zähler für Wiederholgeschwindigkeit der Tasten

Das Betriebssystem verwendet diese Speicherzelle als Zähler, der die Geschwindigkeit bestimmt, mit der eine Taste wiederholt wird, wenn sie länger gedrückt wird. Voraussetzung ist die durch Zelle 650 festgelegte Wiederholbarkeit der Taste.

Am Anfang steht in der Zelle 651 die Zahl 6. Sobald eine wiederholbare Taste gedrückt wird, zählt das Betriebssystem diese Zahl alle 0,0167 Sekunden (60mal in der Sekunde) um 1 zurück, bis die Zahl 1 erreicht ist. Dann erst wird das Zeichen der gedrückten Taste wieder auf den Bildschirm gedruckt oder ihre Funktion wiederholt.

Bei jedem folgenden Lauf steht in Zelle 651 die Zahl 4. Entsprechend verkürzt sich der Zählvorgang.

Am schnellsten würde die Wiederholung natürlich mit dem Wert 1 in der Speicherzelle 651 sein. Von Basic aus mit POKE 651,1 geht das leider nicht.

Im nebenstehenden Texteinschub »Turbo-Tasten« wird ein Maschinenprogramm beschrieben, welches dies kann.

Adresse 652 ($28C)

Zähler für die Ansprechzeit der Wiederholfunktion von Tasten

Diese Speicherzelle wird vom Betriebssystem als Zähler verwendet, der festlegt, wie lange eine wiederholbare Taste gedrückt sein muß, bis die Wiederholfunktion einsetzt.

Am Anfang steht in der Zelle 652 die Zahl 16. Diese Zahl wird alle 0,0167 Sekunden um 1 reduziert, bis die Zahl 0 erreicht ist. Dann wird das Zeichen der Taste auf den Bildschirm gebracht oder ihre Funktion wiederholt. Anschließend wird die Zahl 4 in die Speicherzelle 651 geschrieben (siehe dort), während die Zelle 652 so lange auf 0 stehen bleibt, bis eine andere Taste gedrückt wird. Wie diese anfängliche Verzögerung reduziert werden kann, steht im Texteinschub »Turbo-Tasten«.

Adresse 653 ($28D)

Tastencode der SHIFT-,CTRL- und Commodore-Taste

In der Speicherzelle 203 stehen die Codes aller Tasten, die gedrückt werden, außer die der drei Steuertasten SHIFT, CTRL und Commodore (oft auch CBM-, Logo- oder C=-Taste genannt). Diese drei Ausnahmen haben ihr eigenes Code-Register, eben 653.

Der Grund dafür liegt in der Bedeutung der drei Tasten. Sie können ja bekanntlich verschiedene Zeichensätze einschalten:
— SHIFT schaltet das Zeichen vorne rechts auf einer Taste ein
— C= schaltet das Zeichen vorne links auf einer Taste ein
— CTRL schaltet die Farben vorn auf den Zahlentasten ein
— SHIFT + C= schaltet von dem normalen Zeichensatz auf die Groß-/ Kleinschreibung um.

Ich habe diese Zusammenhänge auch bei der Behandlung der Speicherzellen 245/246 erwähnt.

Die Codezahlen selbst sind auch in der Tabelle 1 der Memory Map in Ausgabe 11/85 auf Seite 146 enthalten. Der Vollständigkeit halber sind sie hier noch einmal angegeben:

SHIFT	1
C=	2
CTRL	4
SHIFT und C=	3
SHIFT und CTRL	5
C= und CTRL	6
SHIFT und C= und CTRL	7

Mit dem folgenden kleinen Programm und mit ein wenig Fingerfertigkeit können Sie diese Codewerte nachvollziehen:

```
10 PRINT PEEK(653)
20 GOTO 10
```

Eine interessante Anwendung habe ich im Texteinschub »Abfrage der Tastencodes oder 476 Funktionstasten« in Ausgabe 11/85 auf Seite 147 gegeben.

Adresse 654 ($28E)

Tastencode der zuletzt gedrückten SHIFT-, CTRL- oder C=-Taste

Diese Speicherzelle wird zusammen mit der Zelle 653 verwendet, um zu verhindern, daß ein schlechter Tastendruck als mehrfaches Drücken derselben Taste gedeutet wird. Im Fachdeutsch nennt man das »Entprellen« einer Taste oder eines Kontaktes. Die Funktion ist vergleichbar mit der der Zelle 197 gegenüber der Zelle 203 für alle anderen Tasten.

Adresse 655 bis 656 ($28F bis $290)

Vektor auf die Routine der Tastencode-Tabellen

Das Betriebssystem hat eine Routine ab Adresse 60232 (60380 beim VC 20), auf die der Vektor in 655/656 zeigt. Sie liest den Codewert der SHIFT-, CTRL- und C=-Taste in der Speicherzelle 653 aus und verändert entsprechend den Vektor der Zellen 245/246 (siehe Memory Map Teil 13, Ausgabe 12/85), so daß er auf die richtige Codetabelle zeigt.

Es gibt Anwenderprogramme, die diesen Vektor so verbiegen, daß die Decodierung der Tasten umgangen und durch eine andere, selbstgebaute Routine ersetzt wird. So kann zum Beispiel das Drücken einer bestimmten Taste umgemünzt werden.

Adresse 657 ($291)

Flagge für Verriegelung der Zeichensatz-Umschaltung

Durch gleichzeitiges Drücken der SHIFT- und der Commodore-Taste wird bekanntlich der Zeichensatz 1 (Großbuchstaben und Grafik-Zeichen) umgeschaltet auf den Zeichensatz 2 (Groß- und Kleinbuchstaben), ein zweites Drücken der beiden Tasten schaltet den Zeichensatz zurück.

Diese Umschaltung wird verriegelt, wenn in der Speicherzelle 657 eine 128 steht. Eine 0 läßt die Umschaltung zu.

Dieser Effekt kann auf zwei, beim C 64 sogar auf drei Arten erzielt werden:

☐ Umschaltung des Zeichensatzes zulassen
— POKE 657,0
— PRINT CHR$(9)
— CTRL und I (nur C 64)
☐ Umschaltung des Zeichensatzes verriegeln
— POKE 657,128
— PRINT CHR$(8)
— CTRL und H (nur C 64)

Adresse 658 ($292)

Flagge für Scrollen

Die Flagge in dieser Speicherzelle legt fest, ob eine weitere echte Zeile zu einer logischen Zeile hinzugefügt wird, sobald der Cursor über das 40ste Zeichen der Zeile (22ste Zeichen beim VC 20) hinausläuft.

Steht in 658 eine 0, dann werden alle Zeilen hochgeschoben (man nennt das »scrollen«), um der neuen Zeile Platz zu machen.

Wenn in der Zeile irgendein Wert größer als Null steht, unterbleibt dieses Scrollen. Die Flagge wird immer dann auf den höheren Wert gesetzt, wenn Zeichen im Tastaturpuffer (631 bis 640) stehen und darauf warten, am Ende des Programms ausgedruckt beziehungsweise ausgeführt zu werden. Diese Verriegelung wird deshalb eingesetzt, weil im Tastaturpuffer Zeichen wie zum Beispiel Cursor-Bewegungen stehen können.

Von Basic aus kann diese Speicherzelle nicht beeinflußt werden.

Das nächste Mal kommen die Speicherzellen 659 bis 673 zur Sprache, die fast ausschließlich für die Steuerung der RS232-Schnittstelle angewendet werden — ein Thema, welches leider in der Literatur immer noch zu kurz kommt.

(Dr.H.Hauck/ah)

Fehlerteufelchen

Eprom-Brenner, Ausgabe 1/86, Seite 149

Beim dort abgebildeten Schaltplan haben die Trennstriche zwischen den Pins 37 bis 40 beim IC2 (6821) einen nicht vorgesehenen Kontakt zur rechts und links vorbeiführenden Leiterbahn. Den in der Stückliste auf Seite 151 angegebenen Spannungswandler TDK 05 CE 0072 kann man zum Preis von zirka acht Mark bei folgender Adresse erhalten. Außerdem kann man Ihnen dort auch nähere Auskünfte zum Aufbau der Platine geben.
M. Frank, Wotanstr. 9, 8000 München, Tel. 089/1782546

Texteinschub #1
Bunte Zeichen und bunter Hintergrund

1) Bunte Zeichen

Wie Zeichen und Buchstaben in bunten Farben auf den Bildschirm gedruckt werden, lernt jeder Hobby-Programmierer schon bei den ersten Gehversuchen — dasselbe innerhalb eines Programms zu erreichen, dauert sicher schon etwas länger.

Bei der Diskussion der Speicherzelle 646 habe ich drei Methoden dafür erwähnt. Ich habe auch gesagt, daß ich die Methode, den Farbcodewert in die Speicherzelle 646 zu POKEn, für die eleganteste halte. Deswegen verwendet das folgende Demonstrations-Programm dieses Verfahren, um den Bildschirm mit einer bunten Reihe der Zahl 1 zu füllen.

```
10 PRINT CHR$(147)
20 POKE 53281,1
30 FOR J=0 TO 1000
40 POKE 646,INT(RND(1)*14+2)
50 PRINT "1";
60 NEXT J
```

VC 20-Besitzer müssen die Zeilen 20, 30 und 40 umändern in:

```
20 POKE 36879,233
30 FOR J=0 TO 505
40 POKE 646,INT(RND(1)*6+2)
```

Erklärung:

Zeile 10 löscht den Bildschirm, Zeile 20 erzeugt einen weißen Hintergrund und eine hellblaue Umrahmung. Zeile 30 zählt vom ersten bis zum letzten Platz auf dem Bildschirm. Zeile 40 erzeugt für jedes Zeichen auf dem Bildschirm eine neue Farbe. Zeile 50 schließlich druckt, durch das Semikolon gesteuert, die Zahl 1 hintereinander und zwar in den Farben, die in Zeile 40 zufällig ausgewürfelt wurden.

RND(1)*14 erzeugt eine Zufallszahl zwischen 0,1 und 13,99. Der Befehl INT davor macht daraus eine ganze Zahl zwischen 0 und 13. Um aber die Codezahl 1 für Weiß zu vermeiden, addieren wir noch 2 dazu, so daß wir Farbcodes zwischen 2 und 15 erhalten. Beim VC 20 ist das alles auf die Farben 2 bis 7 beschränkt.

Das Ergebnis ist wie gesagt ein Bildschirm voller Einser, deren Farben bunt wie ein Regenbogen abwechseln.

2) Bunter Hintergrund

Bunte Zeichen stellen also kein Problem dar. Wie steht es aber mit einem bunten Hintergrund? Den können wir zwar auch verändern (POKEn der Speicherzelle 53281 beziehungsweise 36879 beim VC 20), aber es bleibt immer nur »eintönig«.

Vom Commodore-Autor Jim Butterfield kenne ich nun eine Methode, die auch einen vielfarbigen Hintergrund bietet.

Butterfield geht dabei von einer lustigen Überlegung aus. Wir wissen zum Beispiel, daß der nächtliche Sternenhimmel aus hellen Punkten besteht, die vor einem schwarzen Hintergrund leuchten. Ohne dieses Wissen könnten wir aber ebenso gut annehmen, daß der Himmel — also der Hintergrund — im hellsten Weiß erstrahlt, aber durch einen schwarzen Vorhang (Vordergrund) mit vielen kleinen Löchern abgedunkelt ist.

Das folgende Demo-Programm benutzt diese Denkweise.

```
100 PRINT CHR$(147)
110 POKE 53281,1
120 FOR J=0 TO 1000
130 POKE 1024+J,160
140 POKE 55296+J,INT(RND(1)*14+2)
150 NEXT J
160 FOR K=0 TO 1000
170 POKE 1024+K,177
180 NEXT K
```

Für den VC 20 (ohne Erweiterung) sieht das Programm so aus:

```
100 PRINT CHR$(147)
110 POKE 36879,233
120 FOR J=0 to 505
130 POKE 7680+J,160
140 POKE 38400+J,INT(RND(1)*6+2)
150 NEXT J
160 FOR K=0 TO 505
170 POKE 7680+K,177
180 NEXT K
```

Die ersten drei Zeilen sind mit denen des ersten Demonstrations-Programms identisch.

Zeile 130 und 140 setzen auf jeden Platz des Bildschirms zuerst ein invertiertes Leerzeichen (Bildschirmcode 160) und zwar in einer der vielen möglichen Farben, per Zufallsgenerator in Zeile 140 ausgewählt.

Leerzeichen mit Farbe? Zugegeben, ein Leerzeichen hat normalerweise keine Farbe, man sieht es nicht. Das invertierte Leerzeichen hat aber eine Farbe. Sie kennen es vom Cursor, dessen Blinken dadurch erzeugt wird, daß das Leerzeichen zwischen normal und invertiert umgeschaltet wird (siehe auch die Beschreibung der Speicherzelle 647). Auf diese Weise besteht jetzt der Bildschirm aus einer Vielzahl von bunten Quadraten. Das ist der Vorhang von Jim Butterfield, der vor dem hellen weißen Hintergrund hängt.

Ab Zeile 160 werden alle Plätze des Bildschirms mit der invertierten 1 (Bildschirmcode 177) gefüllt. Diese invertierten Zeichen sind in der Farbe des Hintergrundes geschrieben, eben weiß. Dadurch entsteht der Eindruck, als wäre der Hintergrund bunt und die Zeichenfarbe weiß.

Der Eindruck verstärkt sich noch, wenn wir die 1 über den Bildschirm wandern lassen. Das erreichen wir durch Ändern der folgenden Zeilen:

```
170 POKE 1024+K,160
175 POKE 1025+K,177
```

Durch geschicktes Ausbauen der Zeile 140 können Sie einen vielfarbigen Bildschirm-Hintergrund in Zeilen oder Blöcken erzielen, ein weites Gebiet für bunte Grafik.

Texteinschub # 2 Turbo-Tasten

Das Trio der Speicherzellen 650, 651 und 652 ist zuständig für die Steuerung der sogenannten Wiederholfunktion der Tasten. Darunter verstehen wir die Eigenschaft der Tastatur, das Zeichen oder die Funktion einer Taste so lange zu wiederholen, bis die Taste losgelassen wird. Normalerweise haben diese Funktion nur die Leertaste, die Cursor-Tasten und die INST/DEL-Taste.

Die Zahl in Speicherzelle 650 entscheidet, welche Tasten wiederholbar sind.

Schalten Sie bitte mit POKE 650,128 alle Tasten auf »wiederholbar« um.

Wenn Sie jetzt eine Taste drücken und sie festhalten, werden Sie folgendes beobachten können.

Nachdem das erste Zeichen auf dem Bildschirm erschienen ist, vergeht eine kurze Zeit, erst dann wird es mit einer gleichbleibenden Geschwindigkeit immer wieder ausgedruckt.

Für die anfängliche Verzögerung ist die Speicherzelle 652, für die Geschwindigkeit der nachfolgenden Wiederholungen die Speicherzelle 651 zuständig.

Viele Spieler und Anwender haben sich sicher schon oft gewünscht, sowohl die Reaktionszeit als auch die Geschwindigkeit der Wiederholfunktion beschleunigen zu können. Leider geht es in Basic nicht, weil die Zahlen in den Zellen 651 und 652 60mal in der Sekunde auf ihren ursprünglichen Wert zurückgesetzt werden.

Aber in Maschinensprache geht es sehr wohl, und zwar mit der sogenannten Interrupt-Methode. Über sie und ihre Wirkungsweise ist schon ausführlich berichtet worden: von Boris Schneider in Ausgabe 3/85 (Der gläserne VC 20) und von Heimo Ponnath in den Ausgaben 7/85 und 8/85 (Assemblerkurs). Ich werde hier nur innerhalb der Beschreibung des folgenden Kochrezeptes darauf eingehen.

Das Kochrezept zur Veränderung der Inhalte von 651 und 652 stammt von Dan Carmichael aus seinem Aufsatz »Speeding Up The VIC« in Ausgabe 10/83 der COMPUTE!'s Gazette.

Wir schreiben es als Maschinenprogramm in Form von DATA-Zeilen in den Bandpuffer ab Adresse 828, wo es geschützt residieren kann, solange keine Kassettenoperationen durchgeführt werden. Das Ladeprogramm in Basic steht in Listing 1.

Für den VC 20 lautet die vorletzte Zahl 191 statt 49.

In Listing 2 ist das Programm disassembliert dargestellt.

Beim VC 20 lautet der Sprungbefehl in Zelle 851 JMP 60095.

Für Anhänger der hexadezimalen Darstellung gebe ich das Programm als HEX-Ausdruck in Listing 3 wieder.

Für den VC 20 lautet die letzte Zeile anders:

```
,0353 4C BF EA JMP EABF
```

Mit dem Befehl SEI werden jegliche Programmunterbrechungen gesperrt. Anschließend kommt das Zahlenpaar 73 und 3 in die Speicherzellen 788/789, wo es in Low/High-Byte Darstellung die Adresse 841 (73+256*3=841) darstellt.

In 788/789 steht normalerweise ein Vektor auf die Adresse 59953 (60095 beim VC 20), von der aus die Aufgaben der »normalen« Unterbrechungsroutine gesteuert werden. Wir »verbiegen«

also den Vektor so, daß er auf die Speicherzelle 841 zeigt.

Die schon genannte Unterbrechungsroutine, die 60mal pro Sekunde alles unterbricht, um die STOP-Taste abzufragen, die Uhr weiterzuschalten und so weiter, springt jetzt nicht auf 59953, sondern zuerst nach 841.

Ab 841 steht jedoch der zweite Teil unseres Maschinenprogramms, das die eingangs gewünschte 1 beziehungsweise 0 nach 651 und 652 schreibt. Das erfolgt jetzt laufend, ein Effekt, der uns in Basic verwehrt ist. Danach allerdings kommt ein letzter Sprungbefehl, der dort weitermacht, wo die Unterbrechungsroutine ursprünglich hätte fortfahren sollen, nämlich in 59953 (60095).

Jetzt fehlt nur noch die Beschreibung, wie sich das alles auswirkt. Ich nehme an, Sie haben immer noch mit POKE 650,128 die gesamte Tastatur auf Wiederholfunktion geschaltet, wenn nicht, holen Sie es bitte nach. Laden Sie das Basic-Programm von Listing 1 und starten Sie es mit RUN. Jetzt steht es in den Speicherzellen 828 bis 853 und kann mit SYS 828 gestartet werden.

Wenn Sie jetzt wieder eine Taste länger gedrückt halten, flitzt das entsprechende Zeichen wie ein Turbo-Auto über den Bildschirm. Der Cursor ist mit den Augen fast nicht mehr zu verfolgen. Es geht alles so schnell, daß Sie Mühe haben, nur ein einzelnes Zeichen auf den Bildschirm zu bringen. Wenn Sie das wollen: Mit RUN/STOP und RESTORE stellen Sie den ursprünglichen Zustand wieder her.

Das kleine Maschinenprogramm läßt sich in jedes Spiel oder Anwendungsprogramm nutzbringend einbauen.

```
6000 FOR A=828 TO 853
6010 READ B
6020 POKE A,B
6030 NEXT:END
6040 DATA 120,169,73,141,20,3,169,3,
6050 DATA 141,21,3,88,96,169,1,141,139
6060 DATA 2,169,0,141,140,2,76,49,234
```

Listing 1. DATA-Lader zur Änderung der Tastenwiederholgeschwindigkeit

```
828  SEI         setzt die Interrupt Enable Flagge
829  LDA #73     lädt Akku mit der Zahl 73
831  STA 788     schreibt die 73 in Zelle 788
834  LDA #3      lädt Akku mit der Zahl 3
836  STA 789     schreibt die 3 in die Zelle 789
839  CLI         löscht die Interrupt Enable Flagge
840  RTS         Ende des Unterprogramms
841  LDA #1      lädt Akku mit der Zahl 1
843  STA 651     schreibt die 1 in Zelle 651
846  LDA #0      lädt Akku mit der 0
848  STA 652     schreibt die 0 in die Zelle 652
851  JMP 59953   Sprung auf Speicherzelle 59953 zum  Weiterlauf
der normalen Interrupt-Routine
```

Listing 2. Disassembler-Listing von Listing 1

```
,033C  78        SEI
,033D  A9 49     LDA #49
,033F  8D 14 03  STA 0314
,0342  A9 03     LDA #03
,0344  8D 15 03  STA 0315
,0347  58        CLI
,0348  60        RTS
,0349  A9 01     LDA #01
,034B  8D 8B 02  STA 028B
,034E  A9 00     LDA #00
,0350  8D 8C 02  STA 028C
,0353  4C 31 EA  JMP EA31
```

Listing 3. Disassembler-Listing mit Hexdump von Listing 1