Memory Map mit Wandervorschlägen (12)

Heute kommen wir in eine Gegend, die von Speicherzellen beherrscht wird, die mit dem Bildschirm-Editor zu tun haben. Neben einer modifizierten INPUT-Routine mit dem GET-Befehl, wird gezeigt, wie man 476 Funktionstasten belegt.

Viele von ihnen sind zur Abfrage und Beeinflussung der Vorgänge auf dem Bildschirm einsetzbar.

Adresse 199 ($C7)

Flagge für reverse Darstellung der Zeichen

Normalerweise steht in dieser Speicherzelle eine 0, was mit PRINT PEEK (199) leicht nachgeprüft werden kann.

Sobald in der Zelle 199 eine andere Zahl als 0 steht, werden alle Zeichen in der reversen Darstellung gedruckt. Das Betriebssystem des Computers erhöht nämlich in diesem Fall den jeweiligen Bildschirmcode der Zeichen um 128. Ein Blick in eine Tabelle der Bildschirmcodes bestätigt, daß die Codes aller reversen Zeichen um genau 128 höher sind, als die, der normalen Zeichen.

Den reversen Modus können wir bekanntlich direkt mit der Kombination der CTRL- und der RVS-ON-Taste oder aber mit PRINT CHR$(18) herstellen. Wenn Sie aber versuchen sollten, das direkt einzugeben, um dann wieder mit PRINT PEEK (199) nachzuschauen, was jetzt in der Speicherzelle 199 steht, dann werden Sie Schiffbruch erleiden. Das Betriebssystem setzt den Inhalt der Zelle 199 nach einem »Wagenrücklauf«, hervorgerufen zum Beispiel durch die RETURN-Taste oder nach einem PRINT-Befehl, der nicht mit einem Komma oder Semikolon abgeschlossen ist, sogleich auf 0 zurück. Natürlich erfolgt das auch durch Drücken der CTRL- und RVS-OFF-Taste.

Wir vermeiden die Rücksetzung durch einen Einzeiler:
PRINT CHR$(18) "AAA" PEEK (199)

Wir erhalten drei reverse As und als Inhalt der Zelle 199 auch die Zahl 18. Dasselbe Ergebnis erhalten wir durch POKEn einer Zahl größer als 0 in die Zelle 199: POKE 199,4: PRINT"XX" PEEK (199)

Das Ergebnis beweist, daß diese Adresse sehr nützlich sein kann, zumal ihre Abfrage beziehungsweise Beeinflussung auch innerhalb eines Programms erfolgen kann.

Adresse 200 ($C8)

Zeiger auf das Ende der eingegebenen logischen Zeile

Eine echte Zeile faßt beim C 64 maximal 40 Zeichen, beim VC 20 nur 22.

Eine Zeile mit Anweisungen darf beim C 64 insgesamt 80 Zeichen, beim VC 20 sogar 88 Zeichen enthalten. Diese »verlängerte« Programmzeile nennt man »logische Zeile«.

Der Zeiger in Speicherzelle 200 gibt dem Betriebssystem an, auf welcher Position das letzte Zeichen einer eingegebenen logischen Zeile sitzt. Löschen Sie den Bildschirm und geben Sie direkt irgendwo auf dem Bildschirm den Befehl ein:
PRINT PEEK(200)

Sie erhalten die Zahl der Spalte des letzten Zeichens dieses Direkt-Befehls.

Adresse 201 bis 202 ($C9 bis $CA)

Zeiger auf Zeilen- und Spaltenposition des letzten Zeichens einer Zeile

Diese beiden Speicherzellen werden bei GET und INPUT verwendet, um die Zeile und Spalte des letzten Zeichens einer eingegebenen Zeile festzustellen. Die Spalten (in Zelle 201 angegeben) zählen von 1 bis 40 (1 bis 22 beim VC 20). Die Zeilen (in Zelle 202 enthalten) zählen dagegen in Paaren von 0 bis 12, identisch mit der bei Zelle 200 erläuterten »logischen« Zweierzeile. Da dies nicht ganz einsichtig ist, gebe ich einen Bildschirmausschnitt wieder (Bild 1), der den Sachverhalt verdeutlichen soll.

Der erste Direktbefehl steht in der zweiten Zeile, das letzte Zeichen in der Spalte 30. Der zweite Befehl steht in der ersten Sechserzeile. Das heißt also, daß die Zeilenangabe dieselbe ist, egal um welchen Teil der logischen Zeile es sich handelt. Das können Sie leicht nachprüfen, indem Sie den ersten Direktbefehl eine Zeile höher schreiben. Das Resultat ist dasselbe.

Die Unterscheidung, um welche der beiden Zeilenteile es sich handelt, wird in den Speicherzellen 217 bis 242 getroffen.

Beim VC 20 sieht der Bildschirmausdruck etwas anders aus (Bild 2), auch die Befehlseingabe habe ich der Zeilenlänge wegen verändert. Interessant ist beim VC 20 allerdings, daß dort trotz der Länge der logischen Zeile auch nur Zeilenpaare verwendet werden, deren Länge natürlich auf 22 Spalten reduziert ist.

Adresse 203 ($CB)

Tastencode der gerade gedrückten Taste

In Ausgabe 6/85 auf Seite 123 habe ich beschrieben, wie die Tasten des Computers abgefragt werden. Die dabei für jede der 64 Tasten (mit Ausnahme der RESTORE- und der SHIFT-LOCK-Tasten) entstehende Dualzahl wird in eine Dezimalzahl (0 bis 63) umgewandelt und in der Speicherzelle 203 gespeichert, einige auch in der Zelle 653. Diese Zahl steht auch in Speicherzelle 197, um sie mit der vorher gedrückten Taste vergleichen zu können.

Die Codezahlen jeder Taste lassen sich mit folgendem Programm abfragen:

```
10 PRINT PEEK (203)
20 GOTO 10
```

Nach RUN sehen wir ein laufendes Zahlenband, zuerst mit der Zahl 64. Das ist die Codezahl für »keine Taste gedrückt«. Die X-Taste ergibt 23, (26 beim VC 20), die W-Taste ergibt 9. Auch die Funktionstasten haben ihren Tastencode. F1 ergibt 4 (39 beim VC 20) und so weiter.

Nur die Steuertasten CTRL, SHIFT, und C= (Commodore-Taste) zeigen keine Reaktion. Deren Tastencode steht nämlich in Speicherzelle 653. Den Grund für diesen Separatismus erfahren Sie bei der Besprechung dieser Zelle. Hier ist nur interessant, daß nicht nur jede einzelne dieser drei Tasten einen eigenen Code hat, sondern auch alle machbaren Kombinationen von gleichzeitig gedrückten Steuertasten. Um das zu sehen, ändern Sie bitte die Zeile 10 so ab:

```
10 PRINT PEEK (203), PEEK(653)
```

Tabelle 1 gibt Ihnen die volle Übersicht. Wenn Sie sich die Mühe machen, die Zahlenreihen der Zelle 203 auf Vollständigkeit zu prüfen, dann werden Sie feststellen, daß vier Zahlen fehlen. Es sind die Werte, die eigentlich den vier Steuertasten CTRL, C=, rechte und linke SHIFT-Taste zugewiesen sind. Aber wie gesagt, sie werden gleich nach 653 umgeleitet, wobei allerdings kein Unterschied mehr zwischen der linken und rechten SHIFT-Taste gemacht wird.

Einige Anwendungsbeispiele der Tastencodes sowie der Kombinationen der drei Steuertasten

```
2345678901234567890123456789012345678901
0
1
1
2
2    PRINT PEEK(201) PEEK(202)
3 2  30
3
4
4
5
5
6      PRINT PEEK(201) PEEK(202)
6 6  32
7
7
8
8
```

Bild 1. Aufbau einer logischen Zeile beim C 64

```
 234567890123456789012
0
1
1
2
2
3
3  PRINTPEEK(201)
4 3
4
5
5
6
6
7    PRINTPEEK(202)
7  19
8
8
```

Bild 2. Aufbau einer logischen Zeile beim VC 20

finden Sie im Texteinschub »Abfrage der Tastencodes«. Wie schon erwähnt, haben die RESTORE-Taste und die SHIFT-LOCK-Taste keinen eigenen Code.

Die RESTORE-Taste ist überhaupt nicht an die Tastatur-Matrix angeschlossen, sondern ist direkt mit der RESTORE-Leitung des Computers verbunden. Dort löst sie einen sogenannten NMI-Interrupt aus. Die SHIFT-LOCK-Taste ist lediglich eine mechanische Verriegelung der SHIFT-Taste.

Adresse 204 ($CC)

Schalter für Cursor blinken

Ein Wert größer 0 in dieser Speicherzelle schaltet das Blinken des Cursors ab. Diese Abschaltung erfolgt durch das Betriebssystem immer dann, wenn sich Zeichen im Tastaturpuffer befinden und wenn ein Programm ausgeführt wird.

Im folgenden Beispiel einer Eingabe mit dem GET-Befehl, bei dem bekannterweise der Cursor nicht blinkt, wird demonstriert, daß durch POKE 204,0 der Cursor trotzdem blinkt. Das kann für selbstgeschriebene Eingabe-Routinen interessant sein.

```
10 PRINT"JA/NEIN? ";
20 POKE 204,0
30 GET A$: IF A$=""THEN 30
40 PRINT A$
```

Umgekehrt kann man durch POKE 204,1 das Blinken des Cursors abschalten. Es bleibt dabei allerdings dem Zufall überlassen, ob er in der Ein- oder Ausphase abgeschaltet wird. Wenn Sie Pech haben, dann bleibt der Cursor bewegungslos stehen. Dieser Schönheitsfehler kann mit Hilfe der Speicherzelle 207 beseitigt werden.

Adresse 205 ($CD)

Zähler für Blinkfrequenz des Cursors

Das Blinken des Cursors besorgt die Interrupt-Routine. 60 mal in jeder Sekunde unterbricht sie den normalen Programmablauf. Während dieser Zeit führt sie mehrere »Haushalt«-Arbeiten durch. So wird hier die Tastatur abgefragt und das Cursorblinken gesteuert.

Dazu wird die Zahl 20 in die Speicherzelle 205 geschrieben und bei jeder Unterbrechung dann um 1 reduziert. Wenn die Zahl in 205 den Wert 0 erreicht hat, wird der Cursor eingeschaltet. Nach Adam Riese erfolgt das also 60/20 = 3 mal pro Sekunde.

Adresse 206 ($CE)

Bildschirmcode des Zeichens unter dem Cursor

Im Prinzip ist der Cursor nichts anderes als das wiederholte Drucken eines Zeichens in reverser Form, das gerade unter dem Cursor steht. Normalerweise ist dies das Leerzeichen, deshalb sehen wir meistens das ausgefüllte Viereck. Fahren Sie aber mit dem Cursor auf einen Buchstaben, dann erscheint dieser wechselweise normal und revers. In Speicherzelle 206 steht jeweils der Bildschirmcode des Zeichens unter dem Cursor. Geben Sie die folgende Anweisung direkt ein, fahren aber noch vor dem Drücken der RETURN-Taste mit dem Cursor zurück auf eines der Zeichen, zum Beispiel auf ein P:

```
PRINT PEEK(206)
```

Nach RETURN erscheint die Zahl 16. Das ist also der Bildschirmcode des Zeichens, auf dem der Cursor saß, als die RETURN-Taste gedrückt wurde. Sie können das mit allen anderen Zeichen dieser Zeile wiederholen.

Ich kann mir vorstellen, daß eine derartige Abfrage bei einem Programm, welches mit dem Bildschirm arbeitet, sinnvoll sein kann. Die Speicherzelle 206 wird allerdings nach jedem Blinken auf den neuesten Stand gebracht.

Adresse 207 ($CF)

Flagge für Blinkzustand des Cursors

In dieser Speicherzelle wird festgehalten, in welcher der beiden Blink-Phasen — normal oder revers — der Cursor sich gerade befindet. Eine 0 bedeutet reverses Zeichen, eine 1 bedeutet ein normales Zeichen.

Die Abfrage innerhalb eines Basic-Programms funktioniert nicht. Denn die Interrupt-Routine steuert den Phasenwechsel. Mit POKE kann man allerdings etwas bewirken. Bei der Erklärung der Speicherzelle 204 habe ich auf einen Schönheitsfehler der Anweisung POKE 204,1 hingewiesen. Sie bewirkt, daß zwar das Blinken des Cursors gestoppt wird, aber er befindet sich unkontrolliert in der normalen oder in der reversen Phase.

Die reverse Phase (der Schönheitsfehler) kann durch POKEn einer 1 in die Speicherzelle 207 vermieden werden. Im nebenstehenden Texteinschub »Spiele mit dem Cursor« wird davon Gebrauch gemacht.

(Dr. H. Hauck/ah)

TASTE	C 64		VC 20	
	203	653	203	653
nichts	64	0	64	0
F1	4	0	39	0
F3	5	0	47	0
F5	6	0	55	0
F7	3	0	63	0
A	10	0	17	0
B	28	0	35	0
C	20	0	34	0
D	18	0	18	0
E	14	0	49	0
F	21	0	42	0
G	26	0	19	0
H	29	0	43	0
I	33	0	12	0
J	34	0	20	0
K	37	0	44	0
L	42	0	21	0
M	36	0	36	0
N	39	0	28	0
O	38	0	52	0
P	41	0	13	0
Q	62	0	48	0
R	17	0	10	0
S	13	0	41	0
T	22	0	50	0
U	30	0	51	0
V	31	0	27	0
W	9	0	9	0
X	23	0	26	0
Y	25	0	11	0
Z	12	0	33	0
1	56	0	0	0
2	59	0	56	0
3	8	0	1	0
4	11	0	57	0
5	16	0	2	0
6	19	0	58	0
7	24	0	3	0
8	27	0	59	0
9	32	0	4	0
0	35	0	60	0

TASTE	C 64		VC 20	
	203	653	203	653
+	40	0	5	0
-	43	0	61	0
*	49	0	14	0
/	55	0	30	0
=	53	0	46	0
↑	54	0	54	0
←	57	0	8	0
.	44	0	37	0
:	45	0	45	0
,	47	0	29	0
;	50	0	22	0
£	48	0	6	0
@	46	0	53	0
CRSR←	2	0	23	0
CRSR↑	7	0	31	0
DEL	0	0	7	0
HOME	51	0	62	0
STOP	63	0	24	0
RETURN	1	0	15	0
SPACE	60	0	32	0
SHIFT	64	1	64	1
C=	64	2	64	2
CTRL	64	4	64	4
SHIFT und C=	64	3	64	3
SHIFT und CTRL	64	5	64	5
C= und CTRL	64	6	64	6
SHIFT und C= und CTRL	64	7	64	7

Tabelle 1. Tabelle aller Tasten-Codes. Erstaunlich ist, daß sich mit den beiden Speicherzellen (203 und 653) insgesamt 476 Funktionstasten erzeugen lassen.

TEXTEINSCHUB Nr. 1

Abfrage der Tastencodes oder 476 Funktionstasten

In der Speicherzelle 203 stehen die Tastencodes der gerade gedrückten Taste, insgesamt 64 an der Zahl. Vier davon, die Steuertasten CTRL, C= (Commodore-Taste), linke und rechte SHIFT-Taste erscheinen allerdings dort nicht, sondern werden sofort in die Speicherzelle 653 umgeleitet. Dort erhalten sie (allerdings in mehrfacher Kombination) insgesamt acht Codewerte. Die Tabelle der Speicherzelle 203 zeigt alle Werte für den C 64 und den VC 20.

In meinem Kurs »Alle Tasten-, Zeichen- und Steuercodes« in den 1984-Ausgaben des 64'er habe ich die Tastencodes und ihre Anwendung detailliert beschrieben.

Ich erlaube mir, hier einige Erklärungen und Beispiele zu wiederholen.

Anfänger der Computerei sitzen oft verzweifelt an dem Problem, die Funktionstasten der Commodore-Computer zum Leben zu erwecken. Nun, wir wissen, daß sie nur über die Abfrage ihrer Codewerte eingesetzt werden können.

Als Codewerte werden normalerweise nur die ASCII-Codes genannt.

Die schon erwähnte Tabelle zeigt jedoch, daß die Funktionstasten auch Tastencodes haben. Allerdings gibt uns das nur vier Möglichkeiten, entsprechend der Aufschrift für die ungeraden Funktionstasten-Zahlen. Um auch F2 bis F8 zu erhalten, drücken wir ja immer gleichzeitig die SHIFT-Taste. Das können wir bei der Abfrage der Tastencodes natürlich auch machen, indem wir uns den Inhalt der Zelle 203 und 653 ansehen. Das folgende kleine Programm überprüft, über den Tastaturcode, ob eine der acht Funktionstasten gedrückt wurde.

```
 10 A=PEEK(203)
 20 B=PEEK(653)
 30 IF A=4 AND B=0 THEN PRINT"F1"
 40 IF A=5 AND B=0 THEN PRINT"F3"
 50 IF A=6 AND B=0 THEN PRINT"F5"
 60 IF A=3 AND B=0 THEN PRINT"F7"
 70 IF A=4 AND B=1 THEN PRINT"F2"
 80 IF A=5 AND B=1 THEN PRINT"F4"
 90 IF A=6 AND B=1 THEN PRINT"F6"
100 IF A=3 AND B=1 THEN PRINT"F8"
110 GOTO 10
```

Die Codezahlen gelten für den C 64, für den VC 20 müssen aus der Tabelle die entsprechenden Werte eingesetzt werden.

Wenn Sie sich anschauen, was in der Speicherzelle 653 alles passiert, dann werden Sie sicher sehen, wie willkürlich die Definition der geraden Funktionstasten ist. Statt der Kombination der Funktionstasten mit der SHIFT-Taste können wir genauso gut die CTRL-Taste nehmen, oder die Commodore-Taste oder alle zwei oder...oder...!

Mit den acht Codewerten in Zelle 653 (0 bis 7) der acht möglichen Kombinationen der drei Steuertasten kann jede Funktionstaste acht Funktionen haben. Das ergibt insgesamt 32 Funktionstasten, und nicht acht, wie die Aufschrift vermuten läßt. Einige davon werden in dem kleinen Demo(nstrations)-Programm eingesetzt. Zweck des Programms soll das Umschalten auf verschiedene Rahmen- und Hintergrundfarben sein. Für den C 64 gilt:

```
10 PRINT CHR$(147)
20 A=PEEK(203)
30 B=PEEK(653)
40 IF A=4 AND B=2 THEN POKE 53280,6:POKE 53281,7
50 IF A=5 AND B=2 THEN POKE 53280,5:POKE 53281,2
60 IF A=6 AND B=2 THEN POKE 53280,1:POKE 53281,1
70 IF A=1 AND B=7 THEN POKE 53280,3:POKE 53281,1
80 GOTO 20
```

Für den VC 20 gilt:

```
10 PRINT CHR$(147)
20 A=PEEK(203)
30 B=PEEK(653)
40 IF A=4 AND B=2 THEN POKE 36879,126
50 IF A=5 AND B=2 THEN POKE 36879,45
60 IF A=6 AND B=2 THEN POKE 36879,25
70 IF A=1 AND B=7 THEN POKE 36879,27
80 GOTO 20
```

Zeile 40 schaltet mit F1 und C= die Farbkombination BLAU/GELB ein.
Zeile 50 schaltet mit F3 und C= die Farbkombination ROT/GRÜN ein.
Zeile 60 schaltet mit F5 und C= die Farbe Weiß ein.

Als Spezialität schaltet Zeile 70 in den Normalzustand zurück, allerdings mit der seltenen Tastenkombination ← (Pfeil links) und alle drei Steuertasten (CTRL, SHIFT, C=) gleichzeitig gedrückt.

Jetzt aber kommt es noch ganz dick!
Ich habe oben gesagt, daß wir nicht acht, sondern 32 Funktionstasten haben. Die Verwendung der vier Funktionstasten in Kombination mit den acht Steuertastencodes in 653 macht es möglich. Dasselbe gilt natürlich für jede andere Taste auch! Zeile 70 im Demo-Programm beweist es.

Da uns insgesamt 60 Tasten zur Verfügung stehen, können wir theoretisch 480 Funktionstasten erzeugen — theoretisch, weil ja auch die STOP-Taste eine gültige Taste ist. Diese Taste steht uns allerdings nur in den Kombinationen mit der SHIFT-Taste zur Verfügung. Ohne SHIFT tut sie ihre Pflicht — sie stoppt. Mit SHIFT aber stoppt sie nicht, so daß wir insgesamt 472 mögliche Kombinationen haben — sicher mehr, als Sie je brauchen werden.

Übrigens, von den Kombinationen sind diejenigen mit der CTRL- oder Commodore-Taste in Spielen oder Anwenderprogramme wie Vizawrite oder Programmierhilfen sehr verbreitet. Ich kann Ihnen nur empfehlen, diese Art der Tastenabfrage ebenfalls zur Steuerung von Programm-Abläufen einzusetzen.

TEXTEINSCHUB Nr. 2

Cursor-Spiele oder der Input-Befehl einmal etwas anders

Die Speicherzellen 204, 205 und 207 haben alle in einer bestimmten Weise mit dem Cursor zu tun. Da die Details bei jeder dieser Zellen behandelt worden sind, möchte ich hier zusammengefaßt ihren Einsatz an einem kleinen Demo-Programm zeigen. Die Idee zu diesem Programm stammt von Russ Davies (COMPUTE! Publications).

Russ Davies geht von der in vielen Leserbriefen geäußerten Unzufriedenheit mit dem INPUT-Befehl aus, der nicht beliebig lange Zeichenketten zuläßt und sich auch bei versehentlich gedrückter RETURN-Taste schlecht benimmt.

Eine Abhilfe wäre der GET-Befehl, aber der wiederum liefert keinen auffordernden Cursor. In diese Marktlücke springt das folgende kleine Programm, welches die prinzipiellen Anweisungen zeigt für:

— Eingabe langer Zeichenketten mit GET
— blinkender Cursor trotz GET
— veränderbares Blinken des Cursors

```
 10 POKE 211,0
 20 POKE 207,0:POKE 204,0:POKE 205,5
 30 FOR I=1 TO 40:NEXT
 40 GET A$
 50 IF A$=CHR$(13) THEN 100
 60 PRINT A$;
 70 X$=X$+A$
 80 GOTO 20
100 POKE 207,0:POKE 204,0:POKE 211,0
120 PRINT X$:PRINT:GOTO 20
```

Zeile 10 verwendet im Vorgriff auf das nächste Mal die Speicherzelle 211. Dieser Befehl, auch in Zeile 100, setzt den Cursor auf den Anfang der logischen Zeile zurück. Zeile 20 müßte eigentlich klar sein. Der Wert des POKE-Befehls für 205 ist interessant. Durch ihn kann die Blinkfrequenz des Cursors verändert werden. Bei diesem Programm ergibt der Wert 5 zusammen mit der Warteschleife in Zeile 30 eine mäßige Blinkfrequenz. Der Wert 1 läßt den Cursor eifrig zappeln.

Zeile 30 wie gesagt, dient zur Abstimmung der Cursorfrequenz, die von der Laufzeit der Programmschleife (20 bis 80) abhängt. Trotz des GET-Befehls in Zeile 40 blinkt der Cursor wegen der Flaggen in Speicherzellen 207 und 204.

Zeile 70 baut die Zeichenkette zusammen. Zeile 50 erlaubt ein Drücken der RETURN-Taste, wodurch lediglich die alte Zeichenkette mit der neuen Eingabe zusammengebunden wird. Einen Aussprung aus der Schleife will ich Ihnen selbst überlassen. Im vorliegenden Beispiel geht er nur über die STOP-Taste.