

Es kann überhaupt nur ein einziges Window erzeugt werden und das nicht etwa per Basic-Befehl (wie man es an sich erwarten würde), sondern über eine ESC-Funktion. Damit kommen wir gleich zur Bedeutung der ESC-Taste auf der Tastatur.

Es gibt nämlich 26 ESC-Funktionen, die durch Drücken von ESC, gefolgt von einer Buchstabentaste, aufgerufen werden (Tabelle 2).

Um das eine mögliche Fenster zu erzeugen, muß man den Cursor in die linke obere Ecke des vorgesehenen Windows bringen, dann ESC T drücken, anschließend in die rechte untere Ecke fahren und ESC B betätigen. Dadurch ist das Fenster definiert. Alle Ein- oder Ausgaben spielen sich jetzt ausschließlich hier ab.

Durch zweimaliges Drücken der Home-Taste wird das Window wieder gelöscht.

So fortgeschritten das Konzept auch gegenüber dem VC20/C64 ist, es bleibt einiges zu wünschen übrig. Die Methode der Window-Definition ist viel zu umständlich und zu langsam, zumal ein Befehl zur direkten Cursorpositionierung nicht vorhanden ist. Das fällt um so schwerer ins Gewicht, als immer nur ein einziges Window definiert werden kann, was aber in der Regel nicht sehr sinnvoll ist. Wenn man den Bildschirm aber in verschiedene Bereiche aufteilen will, dann wirkt sich das ständige umständliche Definieren der Fenster doch zum einen auf die Programmlänge, zum anderen auf die Abarbeitungsgeschwindigkeit negativ aus.

Dennoch ist das Windowing ein Schritt in die richtige Richtung, hin zum benutzerfreundlichen Computer. Für eine übersichtliche Bildschirmaufteilung besteht jedenfalls in fast jeder Programmiersituation ein Bedarf. Wo man sich früher damit behelf, den gesamten Bildschirm bei jeder Verände-

rung neu aufzubauen, ist es jetzt möglich, nur den wirklich zu ändernden Bereich anzusprechen, ohne dabei die übrigen Informationen zu beeinflussen.

Maschinensprache-Monitor eingebaut

Für Maschinensprachefreunde – und solche, die es werden wollen – hält der C 16 noch einen ganz besonderen Leckerbissen parat. Er verfügt nämlich über einen fest im ROM vorhandenen Maschinensprache-Monitor, genannt TEDMON.

TEDMON ist genau genommen sogar mehr als nur ein Monitorprogramm für Maschinensprache. Er enthält nämlich einen Disassembler und auch einen kleinen Assembler. Maschinenprogramme können mit TEDMON sehr komfortabel entwickelt und anschließend als schnelle Unterroutinen von Basic aus aufgerufen werden. Tabelle 3 zeigt den TEDMON-Befehlssatz.

Ein Schwachpunkt des C 16 ist sicher der mit 16 KByte zu kleine Anwenderspeicher (nach Einschalten der hochauflösenden Grafik bleiben noch exakt 2045 Byte zum Programmieren). Man sollte aber nicht vergessen, daß man bei der Leistungsstärke des C 16 Basic in 2 KByte etwa das gleiche an Grafik-Programm unterbringen kann wie beim C64 in 8 KByte.

Der C 16 ist insbesondere vom Basic her in der Tat mindestens eine ganze Generation weiter als der VC20 und der C64. Durch sehr komfortable Programmierhilfen und ein umfangreiches, praxisnahes Basic ist er der ideale Einsteiger-Computer mit der Fähigkeit, seinem Besitzer das Tor zum Computer zu öffnen. (ev)

Den C 16 und VC 20 durchschaut

Der VC 20 und der C16 besitzen – so erstaunlich das auch klingen mag – sehr viele Gemeinsamkeiten. Im Rahmen dieses Kurses bekommen Sie einen Einblick in den inneren Aufbau und die Programmierung beider Computer.

Dieser Kurs wird »zweigleisig« abgehandelt. Die im normalen Text angeführten Adressen beziehen sich immer auf das Betriebssystem des VC 20. Die entsprechenden Werte für den C16 sind in Klammern dahinter angegeben. Oft wird es auch vorkommen, daß nur eine Adresse genannt wird, da diese bei beiden Geräten gleich ist.

Die Betriebssysteme des VC 20 und des C 16 sind sehr flexibel gestaltet. Es gibt für den Benutzer viele Möglichkeiten, das Bestehende zu ändern oder zu ergänzen.

Wie Basic den Speicher verwaltet

Beginnen wollen wir mit der Organisation des verfügbaren RAM durch den Basic-Interpreter.

Der Basic-Beginn liegt bei Adresse 4096, das Ende bei Adresse 7680 (16383) (die Werte beziehen sich auf die Grundversion). Unmittelbar ab dem Basic-Beginn wird das eigentliche Programm abgelegt. An dessen Ende beginnen die Variablen und Felder (Bild 1 und Tabelle 1).

Speicher- stelle	Bezeichnung	Werte	
		VC 20	C16
43 44	Basic- Beginn	GV : 4096	4096
		+ 3K : 1024	—
		+ 8K : 4608	—
45 46	Variablen- Beginn	Abhängig von der Programmlänge	
55 56	Basic- Ende	GV : 7680	16383
		+ 3K : 7680	—
		+ 8K : 16384	—
		+ 16K : 24576	—
		+ 24K : 32768	—
		+ 48K : —	65535

Tabelle 1. Der Inhalt dieser Speicherzellen bestimmt die Aufteilung des Basic-Bereichs

Der Variablenbereich wächst beim Anlegen neuer Variablen von unten nach oben. Nur das Stringende wandert in entgegengesetzter Richtung.

Die wichtigsten Zeiger, wie unter anderem Beginn und Ende von Basic und Variablen, sind in der Zeropage (Adresse 0 bis 256) abgelegt (Tabelle 1). Dabei ist die Reihenfolge Low-Byte/High-Byte zu beachten (Adresse = High-Byte x 256 + Low-Byte).

Um Speicherplatz für Maschinenprogramme oder Sonderzeichen zu schaffen, hat man prinzipiell zwei Möglichkeiten.

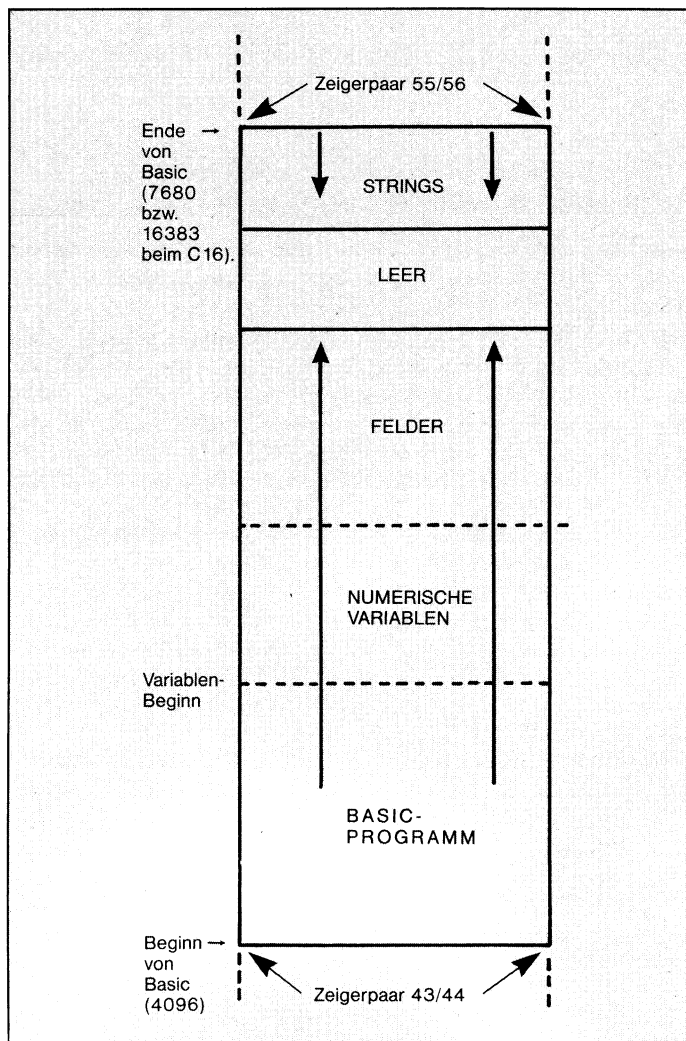


Bild 1. Die Speicherbelegung des Basic-Bereichs

Entweder man verschiebt den Basic-Anfang nach oben oder das Basic-Ende nach unten. Letztere Alternative ist in den meisten Fällen günstiger.

Um zum Beispiel beim VC 20 das Basic-Ende von Adresse 7680 nach 7168 (= 512 Byte) zu verlegen, gibt man ein:
POKE 55,0:POKE 56,28:CLR:REM(256 * 28 = 7168)

Bei anderen Speichergrößen verfährt man analog.

Der Befehl CLR ist nötig, damit sich verschiedene Hilfszeiger (Stringbeginn und Felderende) anpassen können.

Die andere Alternative der Platzbeschaffung ist etwas komplizierter und gilt sinnvollerweise nur für den VC 20. Sie wird nur bei erweitertem Speicher angewendet, um dort Sonderzeichen abzulegen. Um den Anfang des Programmspeichers von 4608 nach 7680 zu schieben, gibt man:

POKE 44,30:POKE 30 * 256,0:NEW

ein, denn 30×256 ist gerade 7680. Der zweite POKE-Befehl ist nötig, da am Anfang des Basic-Bereichs immer ein Nullbyte stehen muß.

Eine Besonderheit bietet hier der C16 (in der Grundversion). Wenn man mit dem Befehl GRAPHIC die hochauflösende Grafik des C16 einschaltet, dann verkleinert sich der Gesamtspeicher um 14 KByte. In diesem Teil des Speichers (siehe auch Bild 6b) werden dann die grafischen Informationen festgehalten.

Die Reservierung dieses Bereiches wird ebenfalls durch die Veränderung des Zeigerpaares (55/56) realisiert.

Schaltet man in den Textmodus zurück, dann bleiben einem nur noch 2 KByte für Programme. Benötigt man den Platz für die Grafiken nicht mehr, so kann man den dafür freigehaltenen Speicherbereich mit

»POKE 55,0: POKE 56,63: CLR«
fürs Basic wieder freigeben.

Erste Hilfe – Basicprogramme retten nach NEW oder Reset

Schon oft wurden Verfahren beschrieben, um nach einem versehentlichen NEW das Basic-Programm wieder zurückzuholen. Doch wie funktionieren diese Verfahren? Um das zu verstehen, betrachten wir zunächst kurz den Aufbau eines Basic-Programms (Bild 2).

Am Kopf des Programms steht immer eine Null. Dann folgen die Adresse der nächsten Programmzeile (Koppeladresse) und die Zeilennummer. Danach kommt die eigentliche Programmzeile, die sich aus den sogenannten Tokens – den Basic-Codenummern (Tabelle 2) – zusammensetzt. Am Ende dieser Zeile steht dann nochmals eine Null. Die nächste Zeile beginnt wieder mit einem Verbindungszeiger und der Zeilennummer. Das Programm wird mit drei Nullen abgeschlossen. Hieran schließen sich die Variablen an (vergleiche Bild 1).

Durch NEW oder durch einen RESET wird nicht das gesamte Programm, sondern nur der Variablenpointer (45/46) und die erste Koppeladresse gelöscht. Durch Rekonstruktion dieser beiden Zeiger kann das scheinbar verlorene Basic-Programm wieder benutzt werden.

Hier nun das »Rezept« zur Rekonstruktion:

- POKE (Basic-Anfang) + 2,1
Basic-Anfang in GV = 4096 (4096)
+ 3K = 1024 (-)
+ 8K = 4608 (-)
- SYS 50483:POKE 46,PEEK(35):POKE 45,
PEEK (781)+2:CLR
- Die Besitzer des C16 geben statt dessen ein:
SYS 34840 : SYS 34892

Unbedingt wichtig ist hier die Reihenfolge der Befehle! Ferner darf während der gesamten Prozedur keine Variable definiert werden, da diese das gelöschte Programm überschreiben würde.

Die Funktionsweise ist relativ einfach. Die Unterprogrammroutine in Adresse 50483 bindet die Programmzeilen neu und stellt dabei den ersten Verbindungszeiger wieder her. Sie übergibt dann in den beiden Speicherstellen 35 und 781 die Adresse des Variablenbeginns -2.

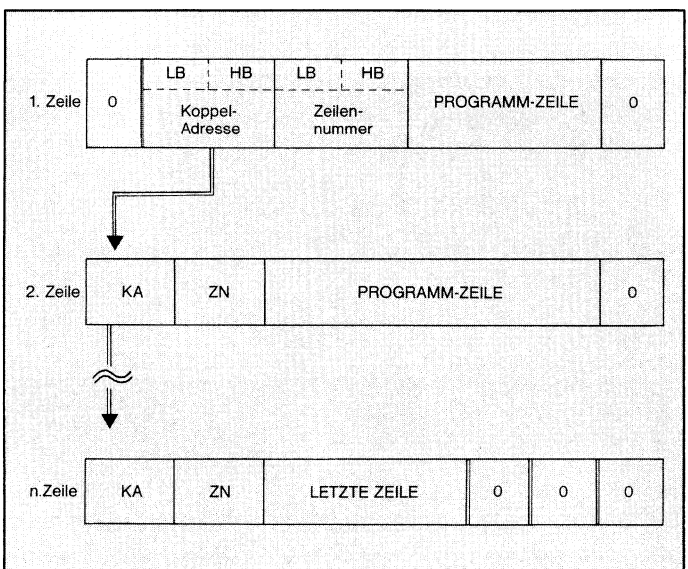


Bild 2. Aufbau eines Basic-Programms. Über die Koppeladressen sind die einzelnen Zeilen miteinander verbunden.

Code (Dezimal)	Zeichen/ Befehl	Code (Dezimal)	Zeichen/ Befehl	Code (Dezimal)	Zeichen/ Befehl
0	Zeilenende	94	†	189	EXP
1- 31	Leer	95	"	190	COS
32	Space	96-127	Leer	191	SIN
33	!	128	END	192	TAN
34	"	129	FOR	193	ATN
35	#	130	NEXT	194	PEEK
36	\$	131	DATA	195	LEN
37	%	132	INPUT #	196	STR\$
38	&	133	INPUT	197	VAL
39	'	134	DIM	198	ASC
40	(	135	READ	199	CHR\$
41	)	136	LET	200	LEFT\$
42	*	137	GOTO	201	RIGHT\$
43	+	138	RUN	202	MID\$
44	,	139	IF	203	GO
45	-	140	RESTORE	204	RGR
46	.	141	GOSUB	205	RCLR
47	/	142	RETURN	206	RLUM
48	0	143	REM	207	JOK
49	1	144	STOP	208	RDOT
50	2	145	ON	209	DEC
51	3	146	WAIT	210	HEX\$
52	4	147	LOAD	211	ERR\$
53	5	148	SAVE	212	INSTR
54	6	149	VERIFY	213	ELSE
55	7	150	DEF	214	RESUME
56	8	151	POKE	215	TRAP
57	9	152	PRINT #	216	TRON
58	:	153	PRINT	217	TROFF
59	;	154	CONT	218	SOUND
60	<	155	LIST	219	VOL
61	=	156	CLR	220	AUTO
62	>	157	CMD	221	PUDEF
63	?	158	SYS	222	GRAPHIC
64	@	159	OPEN	223	PAINT
65	A	160	CLOSE	224	CHAR
66	B	161	GET	225	BOX
67	C	162	NEW	226	CIRCLE
68	D	163	TAB(	227	GSHAPE
69	E	164	TO	228	SSHAPE
70	F	165	FN	229	DRAW
71	G	166	SPC(	230	LOCATE
72	H	167	THEN	231	COLOR
73	I	168	NOT	232	SCNCLR
74	J	169	STEP	233	SCALE
75	K	170	+	234	HELP
76	L	171	-	235	DO
77	M	172	*	236	LOOP
78	N	173	/	237	EXIT
79	O	174	†	238	DIRECTORY
80	P	175	AND	239	DSAVE
81	Q	176	OR	240	DLOAD
82	R	177	>	241	HEADER
83	S	178	=	242	SCRATCH
84	T	179	<	243	COLLECT
85	U	180	SGN	244	COPK
86	V	181	INT	245	RENAME
87	W	182	ABS	246	BACKUP
88	X	183	USR	247	DELETE
89	Y	184	FRE	248	RENUMBER
90	Z	185	POS	249	KEY
91	[	186	SQR	250	MONITOR
92	£	187	RND	251	USING
93	]	188	LOG	252	UNTIL
				253	WHILE

Tabelle 2. Basic-Token des VC 20/C16. Die Codenummern 32 bis 95 entsprechen den normalen ASCII-Zeichen. Nummern größer als 127 sind Token, also Abkürzungen für Basic-Befehle, die der Basic-Interpreter verwendet, um Speicherplatz zu sparen und die Verarbeitungsgeschwindigkeit zu erhöhen. Token ab Code 204 sind nur beim C16 vorhanden.

Blick in die Zeropage

Die Zeropage – oder zu deutsch die Seite null – ist in Maschinsprache besonders einfach zu handhaben. Als Seite bezeichnet man im übrigen immer ein Paket von jeweils 256 Byte. So entspricht Seite 0 den Adressen 0-255, Seite 1 den Adressen 256-511 und so weiter.

Eine Zeropage-Adressierung, zum Beispiel LDA \$42, benötigt nur zwei Byte, eine absolute Adressierung, zum Beispiel LDA \$1234, hingegen drei Byte im Speicher. Damit verbunden ist auch die Bearbeitungsgeschwindigkeit eines Maschinenprogramms. Denn die Zeropage-Adressierung ist schneller als die entsprechende Drei-Byte-Methode. Bei kleineren Programmen in Assembler fällt dieser Aspekt zwar nicht so sehr ins Gewicht, bei sehr umfangreichen Routinen (wie zum Beispiel beim Basic-Interpreter und bei Schleifen) spielt die Adressierungsart jedoch eine größere Rolle.

In der Seite 0 legt der Computer also insbesondere die Daten ab, die er oft benötigt, wie zum Beispiel Vektoren, Parameter etc. Die komplette Liste der Adreßbelegung zeigt Tabelle 3a (Tabelle 3b gilt für den C16).

Wir wollen es jedoch nicht nur mit der Aufstellung alleine bewenden lassen. Die interessantesten Lokationen möchte ich hier herausgreifen und besprechen. Alle angeführten Adressen gelten, soweit nicht anders angegeben, sowohl für den VC 20 als auch für den C16.

Adresse 43-56: Diese Adressen spielen, wie wir schon im ersten Teil gesehen haben, bei der Verwaltung von Basic-Programmen und -Variablen eine zentrale Rolle. Über sie erfolgt die Trennung zwischen Programm und den einzelnen Variablentypen. Auch für das Speichern und Laden von Programmen liefern sie die Basisdaten. Die einzelnen Funktionen können Sie Tabelle 3 entnehmen.

Eine interessante Gebrauchsmöglichkeit ergibt sich durch die Memoryroutine. Dies ist ein kurzes Basic-Programm, welches die Aufgabe hat, den von Programmen, Variablen, Strings und Arrays belegten Speicherplatz festzustellen (Listing 1):

```

10 rem *** memory dump ***
20 rem
30 a=46:b=44:gosub 100:print"programm:";x
40 a=48:b=46:gosub 100:print"variablen:";x
50 a=50:b=48:gosub 100:print"arrays:";x
60 a=56:b=52:gosub 100:print"strings:";x
70 a=56:b=44:gosub 100:print"speicher:";x
80 print fre(0);"bytes free"
90 end
100 x=peek(a)*256+peek(a-1)-peek(b)*256-peek(b-1)
110 return

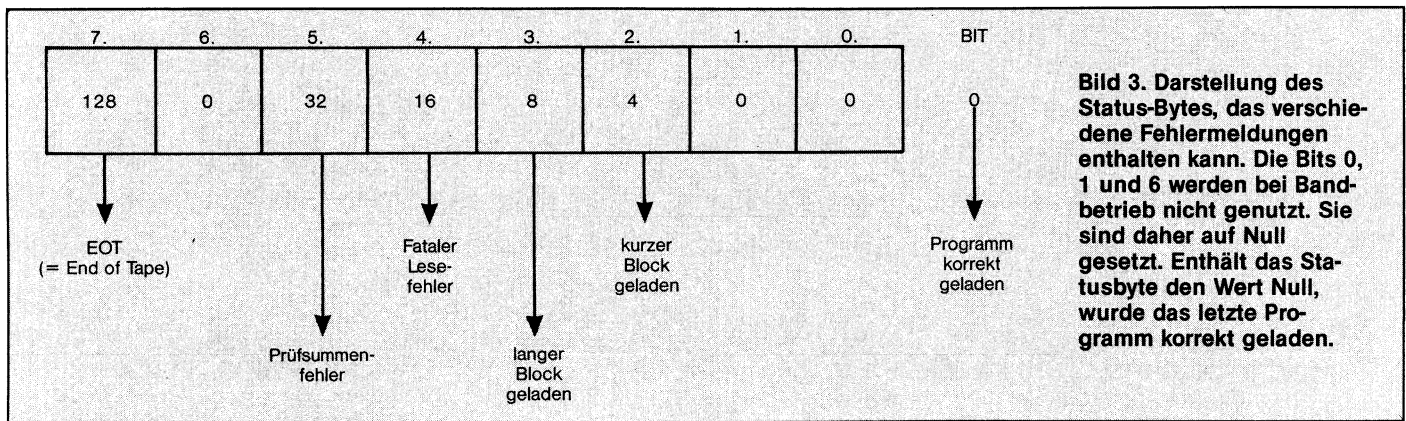
```

Listing 1. »Memory Dump« für VC 20 und C16

Wie man nach dem Starten des Programms sehen kann, erhält man die Werte, indem man die Zeiger – nachdem sie in Dezimalzahlen umgewandelt worden sind – voneinander subtrahiert. Den Speicherplatz, der durch Variablen belegt ist, erhält man beispielsweise durch Subtraktion des Hilfszeigers, »Beginn der Variablen« (45/46) von dem Zeigerpaar »Beginn der Arrays« (47/48).

Gerade bei stark limitiertem Speicherplatz (wie zum Beispiel in der Grundversion) kann dieses Hilfsprogramm Aufschluß über die momentane Speicherverteilung geben.

Adresse 59, 60: Diese Speicherzellen enthalten die laufende Zeilennummer eines Basic-Programms. Sollte es im Programmablauf unterbrochen werden, hat man hier die Möglichkeit, die Zeilennummer nachzulesen.



Zustandsbeschreibung: das Statusflag

Adresse 144: Dieses Statusflag ist auch von Basic aus über die STATUS- beziehungsweise ST-Anweisung abfragbar. Es liefert das Computerstatus-Byte, dessen Inhalt aufgrund der letzten Input-Output-Operation gesetzt wurde. Bezogen auf den Kassettenport liefert es nach Bild 3 bestimmte Meldungen. Die Informationen über die geladenen Programme werden binär wiedergegeben. Die Null signalisiert ein ordnungsgemäß geladenes Programm. 32 hingegen bedeutet, daß ein Prüfsummenfehler vorliegt. Es ist aber auch möglich, daß der Computer mehrere Meldungen in dieses Byte packt, beispielsweise $52 = 32 + 16 + 4$: Hier wurde ein kurzer Block geladen, jedoch ist die Prüfsumme falsch, und ein fataler Ladefehler liegt vor.

An dieser Stelle ist es angebracht, sich näher mit dem Aufzeichnungsverfahren zu beschäftigen (Bild 4).

So kommen Programme aufs Band

Jeder Speichervorgang beginnt mit dem Header. Dieser Kopf besteht aus dem Vorspann (das ist ein etwa acht Sekunden langer Pfeifton) und dem eigentlichen Programmkopf. Dieser enthält vier wichtige Informationen, nämlich über Programmtyp, Startadresse, Endadresse und Programmname. Diese Daten sind ebenfalls im Bandpuffer zu finden und können von dort abgerufen werden.

Das erste Byte (Adresse 828 beziehungsweise 818 beim C16) gibt Auskunft über den Headertyp. Eine 1 zeigt an, daß es sich um ein Programm handelt, das verschoben geladen werden kann, also auch an eine andere Stelle als die, von der aus es gespeichert wurde. Das Gegenstück dazu ist die absolute Lademethode (Headertyp 3). Gemeint ist »LOAD“.,1,1«. Die Sekundäradresse 1 signalisiert dem Computer, daß er das Programm (unabhängig von den Zeigern 43, 44) wieder in den gleichen Adreßbereich laden soll. Um Verwechslungen vorzubeugen, ist es wichtig, Headertyp und Sekundäradresse zu unterscheiden. Für den Headertyp gibt es drei Möglichkeiten:

1. Laden mit Verschiebelader (die Anfangsadresse wird durch den Zeiger 43 und 44 bestimmt).
2. Ein File – also Daten aus Variablen – wurde gespeichert. Die Unterscheidung ist wichtig, denn Daten können nicht mit LOAD geladen werden.
3. Ein Programm ist absolut zu laden.

Die nächsten vier Bytes geben die Anfangs- und Endadresse des geladenen Files an. Mit der Endadresse hat es eine besondere Bewandnis. Sie wird nämlich nach korrektem Laden der Zeropage (Adresse 45, 46) übergeben. Bei

Dezimal	Hexadezimal	Bemerkung
0 - 2	00 - 02	USR-Sprungvektor (Normal: JMP \$D248)
3 - 4	03 - 04	Vektor für Unterprogramm 'Fließkomma nach Integer'
5 - 6	05 - 06	Vektor für Unterprogramm 'Integer nach Fließkomma'
7	07	Suchzeichen (sucht '!' oder Zeilenende)
8	08	Hochkommaflag
9	09	Spaltenspeicher beim TAB-Befehl
10	0A	LOAD/VERIFY Flag (0 = LOAD/1 = VERIFY)
11	0B	Eingabepufferzeiger (Anzahl der Elemente)
12	0C	Flag für Dim (enthält die laufende Variable)
13	0D	Variablentyp (0 = Numerisch/128 = String)
14	0E	Numerische Variable (0 = Fließkomma/128 Integer)
15	0F	Flag bei DATA und LIST
16	10	Flag für FN
17	11	Eingabeflag (0 = INPUT/64 = GET/152 = READ)
18	12	Vorzeichen bei ATN
19	13	Aktuelles Ein-/Ausgabegerät
20 - 21	14 - 15	Integerwert (zum Beispiel Zeilennummer)
22	16	Zeiger im Stringstapel
23 - 24	17 - 18	Zeiger auf den zuletzt verwendeten String
25 - 33	19 - 21	Stringstapel
34 - 37	22 - 25	Speicher für div. Hilfszeiger
38 - 42	26 - 2A	Speicherbereich bei einer Multiplikation
43 - 44	2B - 2C	Zeiger auf Beginn des Basic-Speicherbereichs
45 - 46	2D - 2E	Beginn der Variablen (= Ende des Programms)
47 - 48	2F - 30	Beginn der Arrays (= Ende der Variablen)
49 - 50	31 - 32	Zeiger auf Ende der Arrays
51 - 52	33 - 34	Zeiger auf Beginn der Strings
53 - 54	35 - 36	Hilfszeiger für Strings
55 - 56	37 - 38	Zeiger auf Basic-Ende
57 - 58	39 - 3A	Momentane Basic-Zeilenummer
59 - 60	3B - 3C	Vorherige Basic-Zeilenummer
61 - 62	3D - 3E	Zeiger auf lfd. Basic-Befehl (für CONT)
63 - 64	3F - 40	Momentane Zeilennummer für DATA
65 - 66	41 - 42	Momentane Adresse einer DATA-Zeile
67 - 68	43 - 44	Zeiger auf Element in DATA-Zeile und INPUT-Vektor
69 - 70	45 - 46	Momentaner Variablenname
71 - 72	47 - 48	Adresse der momentanen Variablen
73 - 74	49 - 4A	Zeiger für FOR/NEXT Variable
75 - 76	4B - 4C	Zwischenspeicher für Programmzeiger (bei SAVE)
77	4D	Speicher für Vergleichssymbole
78 - 79	4E - 4F	Zeiger für FN
80 - 83	50 - 53	Verschieden genutzter Speicherbereich
84 - 86	54 - 56	FN-Sprungvektor (ähnlich USR)
87 - 96	57 - 60	Feld für diverse arithmetische Zwecke (Arithmetik Akku #3 und #4)
97	61	Fließkommaakku #1 : Exponent
98 - 101	62 - 65	Fließkommaakku #1 : Mantisse
102	66	Fließkommaakku #1 : Vorzeichen
103	67	Zeiger für Polynomauswertung
104	68	Überlauf von Akku #1
105	69	Fließkommaakku #2 : Exponent
106 - 109	6A - 6D	Fließkommaakku #2 : Mantisse
110	6E	Fließkommaakku #2 : Vorzeichen
111	6F	Vergleichsbyte der Vorzeichen von Akku 1 und 2
112	70	Rundung für Akku #1 und #2

Tabelle 3a. Die Zeropage beim VC20

Dezimal	Hexadezimal	Bemerkung
113 - 114	71 - 72	Länge des Kassettenpuffers
115 - 138	73 - 8A	CHRGET-Routine
139 - 143	8B - 8F	RND-Wert als Fließkommazahl
144	90	Statusflag ST
145	91	STOP-Flag
150	96	Kassetten EOT (End of Tape) erhalten
151	97	Zwischenspeicher für Register
152	98	Anzahl der offenen Dateien
153	99	Eingabegerät (Normal = 0 : Tastatur)
154	9A	Ausgabegerät (Normal = 3 : Bildschirm)
155	9B	Paritätsbyte
150	96	Kassetten EOT (End of Tape) erhalten
151	97	Zwischenspeicher für Register
152	98	Anzahl der offenen Dateien
153	99	Eingabegerät (Normal = 0 : Tastatur)
154	9A	Ausgabegerät (Normal = 3 : Bildschirm)
155	9B	Paritätsbyte vom Band (Prüfsumme)
156	9C	Flag für Byte erhalten (von Band)
157	9D	Flag für Direktmodus (= 128) oder Programm (= 0)
158	9E	Band: erster Teil - Prüfsumme
159	9F	Band: zweiter Durchlauf - Prüfsumme
160 - 162	A0 - A2	Interne Uhr (Stunde, Minute, Sekunde)
163	A3	Bitzähler für serielle Ausgabe
164	A4	Zähler für Band
165	A5	Startsynchronisation bei Kassetten
166	A6	Zeiger im Bandpuffer
167 - 171	A7 - AB	Flags für Schreiben/Lesen bei Band
172 - 173	AC - AD	Zeiger auf Kassettenpuffer und für scrolling
174 - 175	Ae - AF	Zeiger auf Programmende bei LOAD
176 - 177	B0 - B1	Bandzeitkonstanten
178 - 179	B2 - B3	Startadresse des Bandpuffers
180	B4	Bitzähler für Band
181	B5	Band oder RS232: nächstes zu sendendes Bit
182	B6	Übertragungsspeicher für RS232
183	B7	Länge des Filenamens
184	B8	Logische Filenummer
185	B9	Sekundäradresse
186	Ba	Gerätenummer
187 - 188	BB - BC	Zeiger auf Filenamens
189	BD	Ein-/Ausgabespeicher (für seriell)
190	BE	Blockzähler für Band
191	BF	Puffer für serielle Ausgabe
192	C0	Bandmotor Flag
193 - 194	C1 - C2	Startadresse für Ein-/Ausgabe
195 - 196	C3 - C4	Endadresse für Ein-/Ausgabe
197	C5	gedrückte Taste (momentaner Wert im Tastaturmatrixcode)
198	C6	Anzahl der gedrückten Tasten (im Tastaturpuffer)
199	C7	Bildschirm: negative Anzeige
200	C8	Zeiger auf Zeilenende bei Eingabe
201	C9	Cursorzeile
202	CA	Cursorspalte
203	CB	Welche Taste? (64 = keine)
204	CC	Cursorblinken
205	CD	Cursor Blinkzähler
206	CE	Zeichen unter dem Cursor
207	CF	Flag für Cursorblinken (0 = An/1 = Aus)
208	D0	Eingabe von Bildschirm/oder Tastatur
209 - 210	D1 - D2	Start der aktuellen Bildschirmzeile
211	D3	Position des Cursors in der aktuellen Bildschirmzeile
212	D4	Flag für Cursor (0 = Direkt, sonst programmiert)
213	D5	Länge der Bildschirmzeile (21 od. 42 od. 63 od. 84)
214	D6	Cursorzeile
215	D7	Zeiger für diverse Zwecke
216	D8	Anzahl der Inserts
217 - 242	D9 - F2	High Bytes der Bildschirmzeilenanfänge
243 - 244	F3 - F4	Zeiger im Farbspeicher
245 - 246	F5 - F6	Zeiger in der Tastaturdekodiertabelle
247 - 248	F7 - F8	Zeiger auf RS232 Eingabepuffer
249 - 250	F9 - FA	Zeiger auf RS232 Ausgabepuffer
251 - 254	FA - FE	Freie Zeropageadressen
255 - (266)	FF - 10A	Zwischenspeicher für Fließkomma nach ASCII

Tabelle 3a. Die Zeropage beim VC 20 (Schluß)

Dezimal	Hexadezimal	Bemerkung
0	\$0000	Datenrichtungsregister des 7501
1	\$0001	Ein-/Ausgabe-Port des 7501
2	\$0002	Flag für Schleifen
3- 4	\$0003-0004	Neue Startadresse (Renumber)
5- 6	\$0005-0006	Schrittweite (Renumber)
7	\$0007	Gesuchtes Zeichen
8	\$0008	Flag für Anführungszeichen-Modus
9	\$0009	TAB-Spaltenzähler
10	\$000A	Flag: 0 = Load, 1 = Verify
11	\$000B	Zeiger für Eingabepuffer, Anzahl der Elemente
12	\$000C	Flag für Standard-DIM
13	\$000D	Datentyp: \$FF=String, \$00=Numerisch
14	\$000E	Datentyp: \$80=Integer, \$00=Fließkomma
15	\$000F	Flag für DATA/LIST
16	\$0010	Flag: Element/FNx-Flag
17	\$0011	Flag: \$00=INPUT, \$40=GET, \$98=READ
18	\$0012	Flag: Vorzeichen des ATN
19	\$0013	Flag: Input-Prompt
20- 21	\$0014-0015	Int. Adresse
22	\$0016	Zeiger auf temporären Stringstapel
23- 24	\$0017-0018	Letzter temporärer Stringvektor
25- 33	\$0019-0021	Stapel für temporäre Strings
34- 35	\$0022-0023	Bereich für Hilfszeiger 1
36- 37	\$0024-0025	Bereich für Hilfszeiger 2
38	\$0026	Bereich für Produkt bei Multiplikation
39	\$0027	Bereich für Produkt bei Multiplikation
40	\$0028	Bereich für Produkt bei Multiplikation
41	\$0029	Bereich für Produkt bei Multiplikation
42	\$002A	Bereich für Produkt bei Multiplikation
43- 44	\$002B-002C	Zeiger auf Basic-Anfang
45- 46	\$002D-002E	Zeiger auf Variablen-Anfang
47- 48	\$002F-0030	Zeiger auf Beginn der Arrays
49- 50	\$0031-0032	Zeiger auf Ende der Arrays (+1)
51- 52	\$0033-0034	Zeiger auf Stringspeicher
53- 54	\$0035-0036	Hilfszeiger für Strings
55- 56	\$0037-0038	Zeiger auf Speichergrenze
57- 58	\$0039-003A	Laufende Basiczeilennummer
59- 60	\$003B-003C	Textpointer
61- 62	\$003D-003E	Zeiger auf Basic-Statement für CONT
63- 64	\$003F-0040	Nummer der aktuellen DATA-Zeile
65- 66	\$0041-0042	Adresse des aktuellen DATA-Elementes
67- 68	\$0043-0044	Sprungvektor für INPUT
69- 70	\$0045-0046	Aktueller Variablenname
71- 72	\$0047-0048	Adresse der aktuellen Variablen
73- 74	\$0049-004A	Variablenzeiger für FOR...NEXT
75- 76	\$004B-004C	Zwischenspeicher für Basic-Zeiger
77	\$004D	Akkumulator für Vergleichssymbole
78- 79	\$004E-004F	Arbeitsbereich (Zeiger etc.)
80- 81	\$0050-0051	Arbeitsbereich (Zeiger etc.)
82	\$0052	Arbeitsbereich (Zeiger etc.)
83	\$0053	Grafik-Modus
84- 86	\$0054-0056	Sprungvektor für Funktionen
87- 96	\$0057-0060	Bereich für numerische Operationen
97	\$0061	Fließkomma-Akkumulator 1
		(FAC): Exponent
98-101	\$0062-0065	Fließkomma-Akkumulator 1
		(FAC): Mantisse
102	\$0066	Fließkomma-Akkumulator 1
		(FAC): Vorzeichen
103	\$0067	Zeiger für Polynom-Auswertung
104	\$0068	Fließkomma-Akkumulator 1 Überlauf
105-110	\$0069-006E	Fließkomma-Akkumulator 2 Exponent etc.
111	\$006F	Vorzeichenvergleich Akku 1 mit Akku 2
112	\$0070	Fließkomma-Akkumulator 1 niederwertige Stelle
113-114	\$0071-0072	Kassettenpuffer Länge/Zeiger
115-116	\$0073-0074	Automatisches Zeileninkrement 0=AUS
117	\$0075	Grafik-Flag 0=Text 255=Grafik
118-120	\$0076-0078	Arbeitsbereich
121-123	\$0079-007B	Darstellung von DS\$
124-125	\$007C-007D	Basic-Pseudo-Stack-Pointer
126-143	\$007E-008F	Arbeitsbereich (Sound)
144	\$0090	Statuswort ST

Tabelle 3b. Die Zeropage beim C16

Dezimal	Hexadezimal	Bemerkung
145	\$0091	Flag: Stoptaste / RVS-Taste
148	\$0094	Flag: Serieller Bus — Ausgabe Zeichenpuffer
149	\$0095	Zeichenspeicher — Serieller Bus
150	\$0096	Zwischenspeicher
151	\$0097	Anzahl der geöffneten Files
152	\$0098	Eingabegerät (Normalwert: 0)
153	\$0099	Ausgabegerät (Normalwert: 3)
154	\$009A	Flag: \$80=Direkt-, \$00=Programm-Modus
157-158	\$009D-009E	Zeiger auf Ende des Programms
159-160	\$009F-00A0	Temporärer Datenspeicher
161-162	\$00A1-00A2	Temporärer Datenspeicher
163-165	\$00A3-00A5	Echtzeit-Uhr
166	\$00A6	Register für seriellen Bus
167	\$00A7	Register für Kassettenroutine
168	\$00A8	Register für seriellen Bus
169	\$00A9	Temporärer Farb-Vektor
170	\$00AA	Register für Kassettenroutine
171	\$00AB	Anzahl der Zeichen im Filenamen
172	\$00AC	Aktuelle logische File-Nummer
173	\$00AD	Aktuelle Sekundär-Adresse
174	\$00AE	Aktuelle Geräte-Nummer
175-176	\$00AF-00B0	Zeiger auf Filenamen
177	\$00B1	Von Fehleroutine benutzt
178-179	\$00B2-00B3	I/O Startadresse
180-181	\$00B4-00B5	Basis-Ladeadresse
182-183	\$00B6-00B7	Zeiger: Anfang Kassettenpuffer
186-187	\$00BA-00BB	Zeiger auf Zeichen im Kassettenpuffer
190-191	\$00BE-00BF	Register für Long-Fetch-Routine
192-193	\$00C0-00C1	Register für Scrolling
194	\$00C2	RVS-Flag (Bildschirm)
195	\$00C3	Zeiger: Ende der Zeile (Input)
196-197	\$00C4-00C5	Cursorposition (Input), Reihe/Spalte
198	\$00C6	Tastaturabfrage (\$40=keine Taste)
199	\$00C7	Flag: Eingabe Bildschirm/Tastatur
200-201	\$00C8-00C9	Zeiger auf Bildschirmzeile
202	\$00CA	Cursorposition in aktueller Bildschirmzeile
203	\$00CB	Flag: Anführungszeichen-Modus (\$00=nein)
204	\$00CC	Länge der aktuellen Bildschirmzeile
205	\$00CD	Zeile, in der sich der Cursor befindet
206	\$00CE	Letztes Zeichen (I/O)
207	\$00CF	Anzahl der Zeichen (Insert-Modus)
208-215	\$00D0-00D7	Reserviert für Software
216-232	\$00D8-00E8	Reserviert für Anwendungssoftware
233	\$00E9	Arbeitsbereich
234-235	\$00EA-00EB	Bildschirm-Editor (Farbe)
236-238	\$00EC-00EE	Arbeitsbereich (Bildschirm)
239	\$00EF	Anzahl der Zeichen im Tastaturpuffer
241-242	\$00F1-00F2	Register für Monitor
250	\$00FA	Register für X bei Stoptastentest
251	\$00FB	Aktuelle Bank-Konfiguration
254	\$00FE	Arbeitsregister (Editor)

Tabelle 3b. Die Zeropage beim C16 (Schluß)

Load Error geschieht dies nicht; »PRINT FRE(0)« zeigt dann die volle Bytezahl an, obwohl sich ein Programm im Speicher befindet. Man darf das Programm – das vielleicht nur einen geringfügigen Fehler hat – dann nicht starten, weil die Variablen dieses überschreiben würden. Abhilfe schafft in diesem Fall

POKE 45,PEEK(831):POKE 46,PEEK(832):CLR.

Für den C16 gibt man ein:

POKE 45,PEEK(821):POKE 46,PEEK(822): CLR

Die Werte werden damit »von Hand« übertragen und ein normaler Programmablauf ist in den meisten Fällen wieder möglich.

Jetzt aber wieder zurück zu Bild 3: Nachdem der Vorspann und der Programmkopf vor einem Trennzeichen (dies ist ein ganz kurzer Pieps) auf Band geschrieben worden sind, wird der Header gleich noch einmal gespeichert.

Dies ist eine Eigenheit des Commodore-Systems, das der Datensicherheit dient. Denn nachdem der Programmkopf 1 geladen hat, vergleicht er in einem zweiten Durchgang das bisher Geladene (welches sich ja schon im Speicher befindet) mit Programmkopf 2. Eine Abweichung veranlaßt den Computer, eine Fehlermeldung auszugeben, in ganz schweren Fällen wird der Ladevorgang gleich ganz unterbrochen. Diese Verfahrensweise gilt nicht nur für den Programmkopf, sondern auch für Programme und Daten – sie alle werden doppelt gespeichert.

Beim C16 kommt noch eine spezielle Eigenheit hinzu. Da der recht komplexe TED-Chip (siehe separaten Artikel in diesem Heft) den Prozessor bei dessen Ladearbeit – die ja sehr schnell vonstatten gehen muß – stark behindert, wird er ganz einfach abgeschaltet. Aus diesem Grund bleibt der Bildschirm während des Ladevorganges dunkel.

Nachdem also der Programmkopf erkannt worden ist, zeigt der Computer den Filenamen an, und das eigentliche Programm (oder die Daten) wird geladen. Ihnen ist wiederum ein kurzer Vorspann vorangestellt. Der Endblock besteht aus Endmarkierung und einem Nachspann. Er zeigt dem Computer das Ende des geladenen Programms an, womit der Ladevorgang beendet ist.

Nach diesem Exkurs zum Kassettenaufzeichnungsformat nun wieder zur Zeropage.

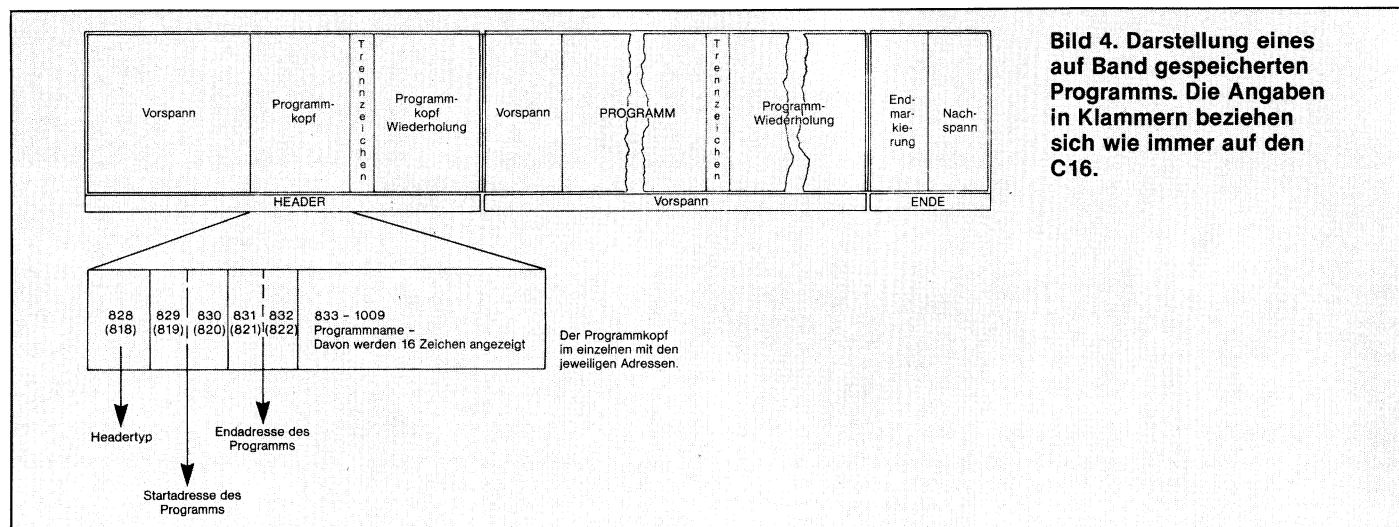
Adresse 145: Mit Hilfe dieser Speicherstelle kann man den Zustand der STOP- und der linken SHIFT-Taste abfragen: Wert =

253: Linke SHIFT-Taste gedrückt

254: STOP-Taste gedrückt

255: Keine der beiden gedrückt

Wenn man das Low-Byte des STOP-Vektors (Adresse 808,



Eingabe: PRINT PEEK(43):LIST100-120

80	82	76	78	84	32	80	69	69	75	40	52	51	41	58	76	73	82	83
P	R	I	N	T		P	E	E	K	(	4	3	)	:	L	I	S	T

49	48	48	45	49	50	48	0	0	0
1	0	0	-	1	2	0			

Bild 5a. Die Basic-Befehle im Eingabepuffer vor der Umwandlung in Interpretercode

153	194	40	52	51	41	58	155	49	48	48	171	49	50	48	0	0	0
↑	↑						↑										
PRINT	PEEK						LIST										
		(	4	3	)	:		1	0	0	-	1	2	0			

Bild 5b. Die Kommandos im Puffer nach der Übersetzung in Token

beim C 16: 806) mit POKE 808, PEEK (808)-2 ändert (bitte beim C 16 Adresse 806 statt 808 verwenden), so ist man in der Lage, diese Taste von Basic aus abzufragen, ohne das laufende Programm anzuhalten. Ihr kann dann in der Routine eine besondere Funktion zugewiesen werden, beispielsweise Anhalten des Programmablaufs für eine bestimmte Zeit.

Der Weg einer Eingabezeile

Wenn wir einen Basic-Befehl im Direktmodus (LIST, RUN, PRINT) oder eine Programmzeile mit Zeilennummer eingeben, werden die Informationen – wie bekannt – auf den Bildschirm geschrieben und gelangen gleichzeitig in den Basic-Eingabepuffer (Adresse 512–600/ \$0200–\$0258). Dort werden Programmzeilen oder direkte Befehle zunächst einmal im ASCII-Format gesammelt (Bild 5a). Dies geschieht solange, bis man die RETURN-Taste betätigt.

Dieser Puffer hat eine Kapazität von 88 Zeichen – also den bekannten vier Bildschirmzeilen beim C20 beziehungsweise 2 Bildschirmzeilen beim C 16.

Drückt man die RETURN-Taste, so beginnt der Interpreter mit der Auswertung der Kommandos. Eingaben ohne Zeilennummer werden auf ihre Syntax hin überprüft und danach ausgeführt.

Verfolgen wir nun einmal genauer den Weg einer Befehlszeile. Die einzelnen ASCII-Zeichen werden von einer Maschinensprachen-Unteroutine namens »CHRGET« aus dem Puffer gelesen und mit den Befehlswörtern aus dem ROM verglichen. War die Überprüfung positiv – ist der Befehl also identifiziert worden –, so verkürzt der Computer die Kommandozeile, indem er die Befehle in Token (Tabelle 2) umwandelt. Diese Prozedur durchlaufen sowohl die Programmzeilen (mit Zeilennummer) als auch die direkten Kommandos (Bild 5b). An dieser Stelle trennen sich nun aber die Wege dieser beiden Zeilentypen.

Zunächst zu dem weiteren Weg einer Programmzeile. Die inzwischen komplett übersetzte Zeile im Basic-Puffer wird nun in einem zweiten Durchlauf vom Zwischenspeicher in den Programmspeicher übertragen, wobei sie auch gleich richtig eingeordnet wird (damit die Reihenfolge der Zeilennummern stimmt). Weil sich dadurch der Programmbereich im Basic-Speicher vergrößert, muß der Variablenbereich weiter oben angesiedelt werden. Die bis dahin gespeicherten Variablen werden dadurch natürlich überschrieben. Nach dem Übertragen der Programmzeile springt das Interpreter-

programm wieder in die Warteschleife zurück, damit weitere Befehle entgegengenommen werden können.

Nun möchte ich den weiteren Weg einer Direktmoduszeile beschreiben, denn der verläuft anders. Nachdem die Zeile mit Hilfe der CHRGET-Routine (Adresse 115–138 beziehungsweise beim C 16: 1139–1156) in Interpretercode (also Token) umgewandelt worden ist, wird der 2-Byte-Zeiger (\$7A–\$7B) auf den Pufferanfang zurückgestellt und der Inhalt wie eine Programmzeile behandelt und abgearbeitet (wieder mit Hilfe der CHRGET-Routine).

Verbotenes

Die Befehle INPUT und GET dürfen im Direktmodus nicht verwendet werden. Der Grund liegt darin, daß diese Eingabebefehle ebenfalls den Basic-Eingabepuffer zur Zwischenspeicherung der Daten verwenden. Das »Direktmodusprogramm«, das sich zu dieser Zeit im Puffer befindet, würde dann überschrieben. Um das zu verhindern, sind diese Kommandos im Direktmodus verboten (Fehlermeldung »ILLEGAL DIRECT«).

Der nächste größere Komplex, den wir hier behandeln wollen, ist die Tastaturverwaltung. Wie Sie bestimmt schon öfter bemerkt oder gelesen haben, wickelt der VC 20 (und der C 16) seine Tastaturoperationen ebenfalls über einen Puffer ab (vergleiche Tabelle 4). Dies wird beim LISTen von Basic-Programmen deutlich, denn während der Computer ein Programm ausgibt, können alle Tasten (mit Ausnahme des STOP-Keys) drücken – diese erscheinen aber nicht auf dem Bildschirm. Erst wenn das Programm zu Ende gelistet ist, sieht man, daß keine Taste »vergessen« wurde, denn alle Eingaben wurden im Tastaturpuffer (Adresse 631–640, beim C 16: 1319–1328) zwischengespeichert.

Mit Hilfe des Tastaturpuffers kann aber auch – und das ist das Interessante an diesen zehn Bytes – eine relativ unkonventionelle Art der Programmierung praktiziert werden.

Dezimal	Hexadezimal	Bemerkung
255– 266	00FF– 010A	Arbeitsspeicher für Fließkomma nach ASCII
266– 318	0100– 013E	Korrekturpuffer für Bandbetrieb
256– 511	0100– 01FF	Prozessor Stack
512– 600	0200– 0258	Basic–Eingabepuffer
601– 610	0259– 0262	Tabelle der logischen Filenummern...

Tabelle 4a. Die Belegung der Seiten 2 bis 4 beim VC20

Dezimal	Hexadezimal	Bemerkung
611- 620	0263 - 026C	sowie der dazugehörigen Gerätenummern...
621- 630	026D - 0276	...und der entsprechenden Sekundäradressen
631- 640	0277 - 0280	Tastaturpuffer
641- 642	0281 - 0282	Start des verfügbaren RAM-Bereichs
643- 644	0283 - 0284	Ende des verfügbaren RAM-Bereichs
645	0285	Timeout-Flag für den seriellen Port
646	0286	Aktueller Farbcode
647	0287	Farbe unter dem Cursor
648	0288	High-Byte des Bildschirmspeichers (Information für das Betriebssystem)
649	0289	Größe des Tastaturpuffers (Maximum 10)
650	028A	Repeat-Flag (0: Cursor + Space/ 64: Keine Taste/ 128: Alle Tasten mit Repeat)
651	028B	Repeat-Zähler (bestimmt die Wartezeit bis die Taste wiederholt wird)
652	028C	Repeat-Verzögerung (bestimmt die Zeit, bis die Taste das erstmal wiederholt wird)
653	028D	Kontrolltasten-Flag (1: SHIFT/ 2: CBM/ 4: CTRL. Es können auch 2 Tasten erkannt werden zum Beispiel 5: SHIFT + CTRL)
654	028E	Letzte Kontrolltaste (Identisch mit 653)
655- 656	028F - 0290	Vektor für Tastaturdecodierung
657	0291	Flag für SHIFT + CBM gesperrt (keine Groß-Klein-Schrift-Umschaltung. 0: Normal/128: Sperre)
658	0292	Flag für Scrolling
659- 670	0293 - 029E	RS 232 Register, Zeiger, etc.
671- 672	029F - 02A0	Zwischenspeicher für IRQ bei Bandbetrieb
673- 676	02A1 - 02A4	Diverse VIA-Zeiger
677- 767	02A5 - 02FF	Zwischenspeicher einer Bildschirmzeile
768- 769	0300 - 0301	Vektor für Fehlermeldung (C43A)
770- 771	0302 - 0303	Vektor für Basic-Warmstart (C483)
772- 773	0304 - 0305	Vektor für Umwandlung von ASCII in Token (C579)
774- 775	0306 - 0307	Vektor für Umwandlung von Token in ASCII (C717)
776- 777	0308 - 0309	Vektor für Basic-Befehlsadresse (C7E1)
778- 779	030A - 030B	Vektor für arithmetisches Element (CE83)
780	030C	Akku für SYS-Befehl (Bei SYS wird 780 in den Akku geladen)
781	030D	X-Reg für SYS-Befehl
782	030E	Y-Reg für SYS-Befehl
783	030F	Speicher für Status Register für SYS-Befehl
788- 789	0314 - 0315	IRQ-Vektor (EABF) *** Kernal-Vektoren
790- 791	0316 - 0317	BRK-Vektor (FED2)
792- 793	0318 - 0319	NMI-Vektor (FEAD)
794- 795	031A - 031B	OPEN-Vektor (F40A)
796- 797	031C - 031D	CLOSE-Vektor (F34A)
798- 799	031E - 031F	Kanal für Eingabe (CHKIN, F2C7)
800- 801	0320 - 0321	Kanal für Ausgabe (CHOUT, F309)
802- 803	0322 - 0323	Kanäle initialisieren (CLRCH, F3F3)
804- 805	0324 - 0325	Eingabe-Vektor (F20E)
806- 807	0326 - 0327	Ausgabe-Vektor (F27A)
808- 809	0328 - 0329	Vektor für STOP-Taste prüfen (F770)
810- 811	032A - 032B	GET-Vektor (F1F5)
812- 813	032C - 032D	Alle Files schließen (F3EF)
814- 815	032E - 032F	User-Vektor (FED2)
816- 817	0330 - 0331	LOAD-Vektor (F549)
818- 819	0332 - 0333	SAVE-Vektor (F685)
828-1019	033C - 03FB	Kassettenpuffer

Tabelle 4a. Die Seiten 2 bis 4 beim VC 20

Dezimal	Hexadezimal	Bemerkung
256- 511	\$0100-01FF	Prozessorstack
512- 600	\$0200-0258	Eingabepuffer
601- 604	\$0259-025C	Basic-Puffer
605- 684	\$025D-02AC	Basic-/DOS-Arbeitsbereich
605	\$025D	DOS-Schleifenzähler
606- 621	\$025E-026D	Bereich für Filenamen
622	\$026E	1. Filename (Länge)
623	\$026F	DOS (Laufwerk 1)
624- 625	\$0270-0271	1. Filename (Adresse)
626	\$0272	2. Filename (Länge)
627	\$0273	DOS (Laufwerk 2)

Dezimal	Hexadezimal	Bemerkung
628- 629	\$0274-0275	2. Filename (Adresse)
630	\$0276	DOS logische Adresse
631	\$0277	DOS (Geräteadresse)
632	\$0278	DOS (Sekundäradresse)
633- 634	\$0279-027A	DOS (Disketten-ID)
635	\$027B	ID-Flag
636	\$027C	DOS (Ausgabepuffer)
637- 684	\$027D-02AC	DOS (Arbeitsbereich)
685- 688	\$02AD-02B0	Grafik-Cursor
689- 692	\$02B1-02B4	Grafik-Cursor (Register)
693- 715	\$02B5-02CB	Grafik-Register
716- 739	\$02CC-02E3	Print-Using, Grafik-Arbeitsbereich
740	\$02E4	High-Byte-Adresse des Charakter-ROM
747	\$02EB	Trace-Flag
752	\$02F0	Anzahl der Grafik-Parameter
753	\$02F1	Parameter: Relativ (1), Absolut (0)
754- 755	\$02F2-02F3	Fließkomma-Vektor
756- 757	\$02F4-02F5	Integer-Vektor
768- 769	\$0300-0301	Sprungvektor: Fehlermeldung
770- 771	\$0302-0303	Sprungvektor: Basic-Warmstart
772- 773	\$0304-0305	Sprungvektor: Token-Generierung
774- 775	\$0306-0307	Sprungvektor: Keyword erzeugen
776- 777	\$0308-0309	Sprungvektor: Hauptschleife
778- 779	\$030A-030B	Sprungvektor: Eval
780- 781	\$030C-030D	Sprungvektor: Token-Generierung (User)
782- 783	\$030E-030F	Sprungvektor: Keyword erzeugen
784- 785	\$0310-0311	Sprungvektor: User-Token bearbeiten
786- 787	\$0312-0313	Sprungvektor: Interrupt (Uhr)
788- 789	\$0314-0315	Sprungvektor: Interrupt
790- 791	\$0316-0317	Sprungvektor: Break Interrupt
792- 793	\$0318-0319	Sprungvektor: Open
794- 795	\$031A-031B	Sprungvektor: Close
796- 797	\$031C-031D	Sprungvektor: Kanal für Eingabe öffnen
798- 799	\$031E-031F	Sprungvektor: Kanal für Ausgabe öffnen
800- 801	\$0320-0321	Sprungvektor: I/O zurücksetzen
802- 803	\$0322-0323	Sprungvektor: Input
804- 805	\$0324-0325	Sprungvektor: Ausgabe
806- 807	\$0326-0327	Sprungvektor: Abfrage der Stoptaste
808- 809	\$0328-0329	Sprungvektor: Getin-Routine
810- 811	\$032A-032B	Sprungvektor: Schließen aller Files
812- 813	\$032C-032D	Sprungvektor: Monitor Break
814- 815	\$032E-032F	Sprungvektor: Lade-Routine
816- 817	\$0330-0331	Sprungvektor: Save-Routine
818-1010	\$0332-03F2	Kassettenpuffer
1011-1012	\$03F3-03F4	Datenzähler (Write)
1013-1014	\$03F5-03F6	Datenzähler (Read)
1015-1078	\$03F7-0436	RS-232-Input-Puffer (64 Byte)
1079-1108	\$0437-0454	System-Arbeitsbereich
1109-1138	\$0455-0472	System-Arbeitsbereich
1139-1144	\$0473-0478	Chrgot-Routine
1145-1156	\$0455-0484	Chrgot-Routine
1172-1244	\$0494-04DC	Bankschitching-Routinen
1255-1258	\$04E7-04EA	Print-Using-Parameter
1263-1270	\$04EF-04F6	Trap- und Error-Flags
1280-1282	\$0500-0502	USR-Sprungbefehl
1283-1287	\$0503-0507	Startwert für RND
1288	\$0508	Flag für Kalt- oder Warmstart
1289-1298	\$0509-0512	Tabelle der logischen Filenummern
1299-1308	\$0513-051C	Tabelle der Gerätenummern
1309-1318	\$051D-0526	Tabelle der Sekundäradressen
1319-1328	\$0527-0530	Tastaturpuffer
1329-1330	\$0531-0532	Basic-Anfang
1331-1332	\$0533-0534	Basic-Ende
1337	\$0539	Zeiger: Kassettenpuffer
1338	\$053A	Typ des Kassettenfiles
1344	\$0540	Tasten-Widerholfunktion \$80=alle, \$40=keine, \$00=nur DEL, CUR, SPC
1345-1346	\$0541-0542	Zähler für Wiederholfunktion
1347	\$0543	Shift-Flag
1348	\$0544	Letztes Shift-Zeichen
1349-1350	\$0545-0546	Sprungvektor: Keylog
1351	\$0547	Text-/Grafik-Flag

Tabelle 4b. Die Belegung der Seiten 2 bis 4 beim C16

Dezimal	Hexadezimal	Bemerkung
1352	\$0548	Scroll-Flag
1355-1372	\$054B-0551	Cpu-Arbeitsbereich
1362-1367	\$0552-0557	Cpu-Register
1368-1372	\$0558-055C	Cpu-Arbeitsbereich
1373	\$055D	Zähler für Funktionstasten
1374	\$055E	Zeiger: Text, Funktionstasten
1375-1510	\$055F-05E6	Speicher für Funktionstasten
1516-1571	\$05EC-06EB	1 Page, Bankingroutinen
1516-1519	\$05EC-05EF	Adreßtabelle
1520-1521	\$05F0-05F1	Long-Jump (Adresse)
1522	\$05F2	Long-Jump (Akkumulator)
1523	\$05F3	Long-Jump (X-Register)
1524	\$05F4	Long-Jump (Status-Register)
1525-1629	\$05F5-065D	RAM-Bereich für Bankswitching
1630-1771	\$065E-06EB	RAM-Bereich für Benutzersoftware
1792-1967	\$06EC-07AF	Basic-Pseudo-Stack
1968-1996	\$07B0-07CC	Kassetten-Arbeitsbereich
1997-2000	\$07CD-07D0	RS-232-Arbeitsbereich
2001	\$07D1	RS-232: Pointer auf Anf. Eingabepuffer
2002	\$07D2	RS-232: Pointer auf Ende Eingabepuffer
2003	\$07D3	Anzahl der Zeichen im Eingabepuffer
2021	\$07E5	Bildschirm: Unterer Rand
2022	\$07E6	Bildschirm: Oberer Rand
2023	\$07E7	Bildschirm: Linker Rand
2024	\$07E8	Bildschirm: Rechter Rand
2026	\$07EA	\$FF=Automatisches Einfügen
2030-2033	\$07EE-07F1	Bit-Tabelle
2034	\$07F2	A-Register retten bei Sys-Kommando
2035	\$07F3	X-Register retten bei Sys-Kommando
2036	\$07F4	Y-Register retten bei Sys-Kommando
2037	\$07F5	Status-Register retten bei Sys-Kommando
2038	\$07F6	Register für Tastaturabfrage
2039	\$07F7	Flag: CTRL-S: \$00=offen, \$06=gesperrt
2040	\$07F8	RAM-ROM-Umschaltung für Monitor 0=ROM, \$80=RAM
2044	\$07FC	Motorsteuerung

Tabelle 4b. Die Seiten 2 bis 4 beim C16 (Schluß)

Um dies verständlich zu machen, habe ich ein Programm (Listing 2) geschrieben, welches DATA-Zeilen aus Maschinenprogrammen oder Sonderzeichen erzeugt. Die Problematik dabei ist folgende: Wenn ich DATA-Zeilen ins Programm schreiben möchte, so ist dies nur im Direktmodus möglich. Um dies automatisch zu tun, benötigen wir ein Programm. Also was tun? – Man bedient sich des Tastaturpuffers!

Der »DATA-Erzeuger«

Dazu nochmals die Durchleuchtung der Funktionsweise. Während des Systeminterrupts, der alle 60stel Sekunde durchlaufen wird, fragt der Computer die Tastatur ab. Ein eingegebenes Zeichen wird dabei im Tastaturpuffer abgelegt, wo es so lange verbleibt, bis ein Zeichen von der Tastatur benötigt wird. Dies ist zum Beispiel im Direktmodus der Fall (wenn der Cursor blinkt) oder im Programm – bei INPUT oder GET. Die Zeichen werden dann wieder aus dem Tastaturpuffer hervorgeholt, und zwar nach dem Prinzip »First in, First out«. Bei unserem Programm werden zunächst einmal sechs Speicherzellen initialisiert. Die ersten zwei Adressen enthalten die Anfangsadresse der abzuspeichernden Daten aus dem Speicher. Dieser Zeiger wird solange inkrementiert, bis er den Wert der Endadresse – der in den zwei folgenden Bytes abgelegt ist – erreicht hat. In den letzten zwei Speicherstellen (Adresse 252, 253 beziehungsweise 212, 213) steht die Zeilennummer in der Reihenfolge Low-/High-Byte. Bequemer wäre es natürlich, wenn man normale Variablen verwenden könnte. Dies ist jedoch nicht möglich, weil diese beim Einfügen einer DATA-Zeile gelöscht werden. Darum

```

1 REM DATA-ERZEUGER
2 REM
3 B=248:REM 'B=208' BEI C 16
10 INPUT" {CLR,2DOWN}STARTADRESSE: ";SA
20 INPUT" {DOWN}ENDADRESSE: ";EA
30 A=SA:GOSUB 250:POKE B,A1:POKE B+1,A2
40 A=EA:GOSUB 250:POKE B+2,A1:POKE B+3,A2
50 POKE B+4,44:POKE B+5,1:REM STARTZEILE 3
  00
60 DEF FN Z(X)=PEEK(X)+256*PEEK(X+1)
65 B=248:REM 'B=208' BEI C 16
70 AD=FN Z(B+4):REM AKTUELLE ZEILENNUMMER
80 PRINT" {CLR}";AD;"DATA ";
90 REM ** DATA-ZEILE MIT 8 ELEMENTEN ERZEUGEN **
100 FOR T=0 TO 7
105 X=PEEK(FN Z(B)+T)
110 B$=STR$(X):L=LEN(B$)
120 B$=RIGHT$(B$,L-1)
130 IF L=4 THEN 150
140 FOR Y=L+1 TO 4:B$="0"+B$:NEXT
150 B$=","+B$
160 IF T=0 THEN B$=RIGHT$(B$,3)
170 PRINT B$;:IF FN Z(B)+T=FN Z(B+2) THEN
  240 220
180 NEXT
190 A=AD+10:GOSUB 250:POKE B+4,A1:POKE B+5,A2
200 A=FN Z(248)+8:GOSUB 250:POKE B,A1:POKE B+1,A2
210 PRINT:PRINT" {DOWN}RUN 60"
215 REM TASTATURPUFFER MIT <HOME>,<RETURN>
  ,<CRSR DOWN> UND <RETURN> FUELLEN
220 Z=631:REM 'Z=1319' BEI C 16
225 POKE Z,19:POKE Z+1,13:POKE Z+2,17:POKE Z+3,13
230 POKE 198,4:REM 'POKE 239,4' BEI C 16
240 END
250 A2=INT(A/256):A1=A-A2*256
260 RETURN

```

Listing 2. »Data-Erzeuger« für VC 20 (für C16 REM-Zeilen beachten)

bleibt nur der Umweg über Speicherstellen, deren Inhalte durch Basic nicht überschrieben werden können.

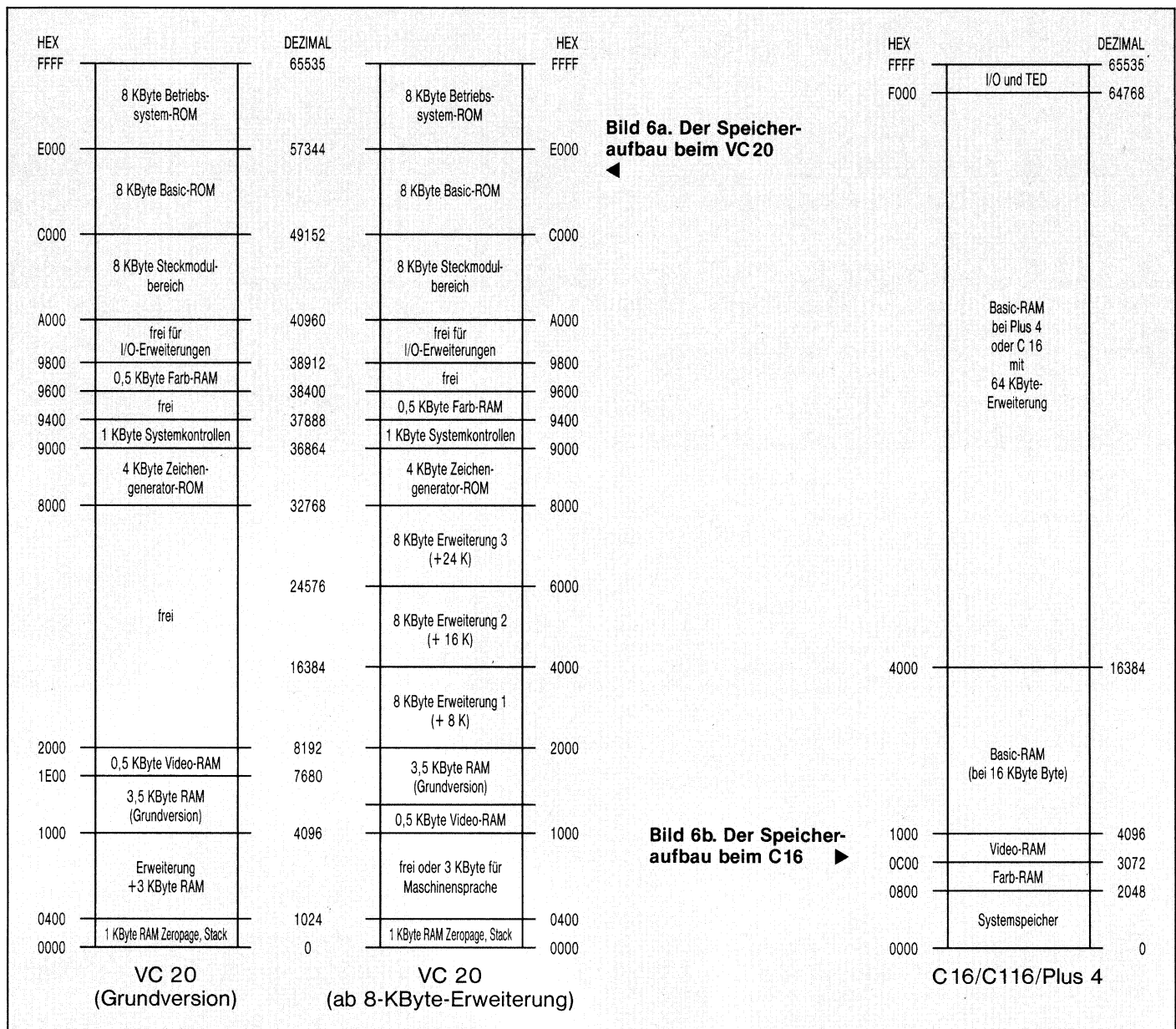
Als nächstes erzeugt das Programm – mit Hilfe der in 252/253 beziehungsweise 212/213 gespeicherten Zwei-Byte-Zahl – eine Zeilennummer, welche auf den Bildschirm gePRINTet wird. Dann schreibt es das Befehlswort »DATA« und druckt acht dreistellige Zahlen (die Daten aus dem zu verarbeitenden Maschinenprogramm) aus. Nun haben wir eine fertige Programmzeile auf dem Bildschirm stehen, die sich allerdings noch nicht im Speicher befindet. Dies erledigt jetzt unser Tastaturpuffer.

Vorher müssen wir uns jedoch genau überlegen, wie unser Bildschirm aussieht, denn dementsprechend muß der Cursor programmiert werden.

Zuerst wird der Cursor mit HOME (=CHR\$(19)) an die linke obere Ecke befördert. Dort wird durch einen Druck auf die RETURN-Taste (=CHR\$(13)) die Basic-Zeile in den Speicher übernommen. Nun befindet sich der Cursor in der dritten Zeile. Durch ein »Cursor down« (=CHR\$(17)) bewegt er sich eine Zeile nach unten und steht nun auf dem Befehlswort RUN 60. Ein weiteres RETURN bewirkt den erneuten Start des Hilfsprogramms.

Wir benötigen für die Cursorbewegung also vier Werte. Diese werden vor dem Ende des Programms mit »POKE 631,19: POKE 632, 13: POKE 633,17:

POKE 634, 13: POKE 198,4« (beziehungsweise die oben angegebenen Werte für den C16) in den Tastaturpuffer geschrieben. Hierdurch simulieren wir eine gedrückte Tastenfolge: denn nachdem sich der



Computer über den Befehl END wieder im Direktmodus befindet, wird zuerst der Tastaturpuffer geleert, wodurch der Cursor den vorbestimmten Weg nimmt. »POKE 198,4« (»POKE 239,4«) gibt an, wie viele Zeichen sich momentan im Puffer befinden.

Jetzt möchte ich noch einen kleinen Einblick in die Speicherorganisation geben, was auch im Hinblick auf Grafik von Bedeutung ist.

Bei Erweiterungen von mehr als 8 KByte verändert sich die Lage des Bildschirm- und Farbspeichers (die Einstellung nimmt die Reset-Routine vor). Anhand von Bild 6a soll erläutert werden, warum eine Verschiebung notwendig ist.

Der Video-Interface-Chip VIC (daher auch der englische Name des VC 20), der vor allem für die Erzeugung des Fernsehsignals und den Aufbau des Bildschirms verantwortlich ist, kann hardwaremäßig nur Videospeicherplätze zwischen 4096 und 8192 adressieren. Folglich muß das Bildschirm-RAM in diesem Bereich angesiedelt werden.

In der Grundversion liegt es zwischen Adresse 7680 und 8191 – also am Ende des verfügbaren Speichers, damit der Speicherbereich für Basic auch bei eingesteckter 3 KByte-Erweiterung durchgängig ist (läge der Bildschirmspeicher so wie bei einer 8-KByte-Erweiterung, wäre dies nicht der Fall).

Ist ein Speichermodul von mehr als 8 KByte eingesteckt (egal ob der 3-KByte-Bereich zugeschaltet ist oder nicht), so

legt das System den Videospeicher an die unterste adressierbare Stelle für den VIC, also Adresse 4096. Aus diesem Grund kann die eingesteckte 3-KByte-Erweiterung nicht mehr für Basic benutzt werden, denn sonst wäre der Speicher nicht mehr durchgängig.

Der C 16 ist vom Speicheraufbau etwas anders organisiert. Die in beiden Geräten eingebauten Prozessoren (6502 und 7501) können jeweils immer nur einen Adreßbereich von 64 KByte (das entspricht den Adressen 0–65535) ansprechen. Um Zugriff auf noch mehr Speicherplatz für Programme und Daten (sei es nun im ROM oder im RAM) zu haben, bedient man sich beim C 16 eines Tricks:

Der Computer schaltet einfach immer zwischen zwei Speicherblöcken zu je 64 KByte (siehe Bild 6b) hin und her. Der erste Block besteht – wenn der C 16 voll ausgebaut ist – aus Schreib-/Lesespeicher, die andere Bank verwaltet das ROM. Diese Technik des »Bank-Switching« wird auch beim neuen Commodore-Flaggschiff C 128 verwendet.

Damit ist unsere Einführung in die internen Geheimnisse des VC 20/C16 beendet. Es ist jetzt an Ihnen, die hier erworbenen Kenntnisse durch eigenes Experimentieren zu vertiefen und in eigenen Programmen einzubauen. Zusätzliche wichtige Hintergrund-Informationen speziell über die Themengebiete »Grafik« und »Zeichensatz« beim C 16 finden Sie in diesem Sonderheft. (Christoph Sauer/ev)