Der gläserne VC 20

In der ersten Folge haben wir uns hauptsächlich mit der Basic-Verwaltung befaßt. Dabei tauchte immer wieder der Begriff Zeropage auf, mit dem wir uns heute beschäftigen wollen.

Teil 2

Die Zeropage — oder zu Deutsch die Seite Null — ist in Maschinensprache 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 Zeropageadressierung, zum Beispiel LDA $42, benötigt nur zwei Byte, eine absolute Adressierung, zum Beispiel LDA $1234, hingegen drei Bytes im Speicher. Damit verbunden ist auch die Bearbeitungsgeschwindigkeit eines Maschinenprogramms. Denn die Zeropageadressierung 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) 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 1.

Wir wollen es jedoch nicht nur mit der Aufstellung alleine bewenden lassen. Die interessantesten Lokationen möchte ich hier herausgreifen und besprechen. Beginnen wir also mit den ersten drei Bytes im Speicher:

Der USR-Vektor

Adresse 0,1,2: Über die USR-Funktion kann der Benutzer eigene mathematische Routinen, die nicht im Basic implementiert sind, in seine Programme einbinden. Dieses Kommando ruft ein Maschinenprogramm auf, dessen Startadresse vorher in den Speicherzellen 1 und 2 abgelegt wurde. Wer sich dieses Befehls nicht bedient, hat drei Zeropagespeicherzellen zur freien Verfügung, die er für eigene Maschinenprogramme nutzen kann. Die Syntax der USR-Funktion entspricht derjenigen normaler Basic-Funktionen wie zum Beispiel FRE, POS, SIN, COS, TAN etc.: A= USR (B) oder PRINT USR (B).

Der Vorteil gegenüber dem SYS-Befehl liegt in der Möglichkeit Parameter zu übergeben. In unserem Beispiel wird der Benutzerroutine die Variable B übergeben. Der errechnete Zahlenwert kann dann seinerseits wieder einer Variablen zugewiesen werden.

Wie kann man sich in einem Maschinenprogramm die eingegebene Zahl beschaffen? Nun, alle Basic-Variablen werden im Fließkommaformat abgespeichert, eine Ausnahme bilden nur die Integer-Variablen. Dabei werden die Zahlen vom Computer nach einem bestimmten Verfahren binär verschlüsselt, so daß eine Zahl mit acht Nachkommastellen und einem Exponenten in nur fünf Bytes Platz findet.

Zwischengespeichert wird das Ergebnis in besonderen Fließkomma-Akkumulatoren (Adresse 97-104). Mit Hilfe eines Unterprogramms aus dem Basic-Interpreter kann es von dort aus abgerufen werden, wobei es in eine Integerzahl (also ein Wert zwischen 0 und 65536) umgewandelt wird.

Aufgerufen wird die Routine mit JSR $D7F7. Der 2-Byte-Wert steht dann in den Zeropagestellen $14 und $15 (Hexadezimal) zur Verfügung. Da diese Adressen vom Interpreter oft benutzt werden, ist es ratsam die Zahlen in andere Register zu übertragen (in welche, darauf kommen wir später noch zu sprechen).

Das Gegenstück zu dem eben gezeigten Unterprogramm stellt die Routine »2-Byte in Fließkomma« dar. Das Low-Byte wird in das y-Register geladen, das höherwertige Byte muß in den Akku. Dann wird die Routine mit JSR $D391 gestartet, wobei die USR-Variable (in unserem Beispiel A) die Daten erhält.

Speicher ausgedeutet — die Basic-Zeiger

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 1 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):

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.

Eine weitere Verwendung ergibt sich durch geschicktes Manipulie-

ren der Speicherzellen, so daß es möglich wird, mehrere Programme gleichzeitig um Speicher unterzubringen.

Diese Aufgabe erfüllt das recht komfortable Programm »Basic-Switch« (Listing 2); zunächst aber die Grundlagen: Die besagten Zeiger stecken, wie bereits im ersten Teil unseres Kurses beschrieben, den Adreßbereich für Basic-Programme ab. Die ersten vier Bytes aus dem Zeropagekomplex (Adresse 43 bis 46) grenzen das Programm nach oben und unten ab. Die daran folgenden Variablen sind in ihrer Ausdehung ebenfalls durch ein Zeigerpaar (Adresse 55/56) limitiert. Der Bereich jenseits dieser Markierung ist für Basic tabu. Dort ist also Platz für Maschinenprogramme oder ähnliches, denn ein Überschreiben durch Basic-Variable ist nicht möglich.

Will man nun mehrere Basic-Programme im Speicher versammeln, so muß man sie nur durch verstellen der Zeiger eindeutig voneinander trennen.

Listing 1. »Memory Dump«

```
10000 REM *** MEMORY DUMP ***
10005 REM
10010 A=46:B=44:GOSUB10080:PRINT"PROGRAM
M  :";X
10020 A=48:B=46:GOSUB10080:PRINT"VARIABL
EN :";X
10030 A=50:B=48:GOSUB10080:PRINT"ARRAYS
   :";X
10040 A=56:B=52:GOSUB10080:PRINT"STRINGS
   :";X
10050 A=56:B=44:GOSUB10080:PRINT"SPEICHE
R  :";X
10060 PRINT"BYTES FREE:";FRE(0)
10070 END
10080 X=PEEK(A)*256+PEEK(A-1)-PEEK(B)*25
6-PEEK(B-1)
10090 RETURN
READY.
```

Wie in Bild 1 zu sehen ist, grenzen die Programmblöcke direkt aneinander. Unter Programmblock ist hier die Einheit von Programm und einem sich daran anschließenden Variablenbereich zu verstehen. Durch Umschalten der Zeiger wird dann immer der gewünschte Programmblock eingeblendet.

Nützlich kann diese Art der Speicheraufteilung sein, wenn man umfangreiche Programme erstellen möchte und es nötig wird, andauernd Hilfsprogramme (zum Beispiel Grafikeditor oder ähnliches) nachzuladen. Durch Basic-Switch kann hier viel Arbeit eingespart werden.

Diese Routine liegt wieder in zwei-

Listing 2. »Basic-Switch« (Basic-Lader)

```
10 REM ************************
20 REM ****                    ****
30 REM ****   BASICSWITCH      ****
40 REM ****                    ****
50 REM ****   BY C. SAUER      ****
60 REM ****                    ****
70 REM ************************
80 FORT=0TO286:READD:P=P+D:NEXT:IFP<>380
21THENPRINT"■■FEHLER !!":END
90 POKE55,0:POKE56,PEEK(56)-2:CLR:A=PEEK
(56)
100 RESTORE:FORT=A*256TOA*256+286
110 READD
120 IFD=-1THEND=A
130 IFD=-2THEND=A+1
140 POKET,D
150 NEXT
160 PRINT"■■FERTIG. START MIT":PRINT"■S
YS"A*256+259
170 PRINT"■■AKTIVIERUNG MIT 'A'"
180 GETA$:IFA$=""THEN180
190 IFA$<>"A"THENEND
200 SYSA*256+259
210 DATA230,122,208,002,230
220 DATA123,032,121,000,201
230 DATA255,240,003,076,121
240 DATA000,032,115,000,201
250 DATA069,208,003,076,168
260 DATA-01,201,083,240,003
270 DATA076,008,207,032,155
280 DATA215,165,101,240,004
290 DATA228,251,144,003,076
300 DATA072,210,134,252,032
310 DATA133,-01,166,252,202
320 DATA138,010,010,010,168
330 DATA162,000,185,032,-02
340 DATA149,043,200,232,224
350 DATA004,208,245,162,000
360 DATA185,032,-02,149,055
370 DATA200,232,224,002,208
380 DATA245,165,252,133,250
390 DATA032,096,198,032,142
400 DATA198,032,115,000,240
410 DATA006,201,143,240,016
420 DATA208,245,160,001,177
430 DATA122,208,239,200,192
440 DATA002,208,247,076,116
450 DATA196,032,215,202,165
460 DATA122,164,123,032,030
470 DATA203,240,241,166,250
480 DATA202,138,010,010,010
490 DATA168,162,000,181,043
500 DATA153,032,-02,200,232
510 DATA224,004,208,245,162
520 DATA000,181,055,153,032
530 DATA-02,200,232,224,002
540 DATA208,245,096,032,133
550 DATA-01,032,115,000,032
560 DATA138,205,032,184,209
570 DATA165,253,133,043,165
580 DATA254,133,044,165,101
590 DATA024,101,253,133,253
600 DATA165,100,101,254,133
610 DATA254,165,251,133,252
620 DATA230,251,165,043,208
630 DATA002,198,044,198,043
640 DATA032,248,227,165,254
650 DATA205,132,002,144,016
660 DATA208,007,165,253,205
670 DATA131,002,144,007,169
680 DATA012,160,205,032,030
690 DATA203,165,253,133,055
700 DATA165,254,133,056,032
710 DATA068,198,165,252,133
720 DATA250,076,049,-01,162
730 DATA000,134,250,134,116
740 DATA232,134,251,134,252
750 DATA165,043,133,253,165
760 DATA044,133,254,169,076
770 DATA133,115,169,-01,133
780 DATA117,096
READY.
```

erlei Form ausgedruckt vor. Zum einen als Assemblerlisting für solche, die tiefer in das Programm einsteigen möchten (Listing 3). Und zum anderen als Basic-Lader. Das Programm ist grundsätzlich auf jeder Ausbaustufe lauffähig. Sinnvoll wird es jedoch erst bei größerem Speicher. Es ist in bewährter Weise über das Befehlswort PI in den Interpreter eingebunden.

Nachdem die Routine per SYS gestartet worden ist, (der Lader übernimmt auch diese Arbeit), muß der erste Programmbereich initialisiert werden, das heißt der Benutzer nimmt die größenmäßige Einteilung des Speichers vor. Man muß sich jedoch schon im voraus darüber im klaren sein, wie lang das Programm inklusive Variablen sein soll, denn eine nachträgliche Änderung ist nicht möglich.

Die Syntax für die Initialisierung lautet: π E »Länge«. Danach steht der Speicherbereich abzüglich zwei Bytes für Basic zur Verfügung. Der erste so eingerichtete Programmblock hat die Nummer 1, der zweite 2 und so fort. Nachdem ein zweiter Programmbereich eingerichtet wurde, kann mit π S »Nummer« zwischen den Blöcken umgeschaltet werden, wobei die Routine jedoch jedesmal die Variable löscht.

Um dem Benutzer die Orientierung zu erleichtern, druckt das Maschinenprogramm nach jedem Umschalten die ersten REM-Zeilen aus. Daher ist es ratsam, jedem Programm eine Kopfzeile mit den wichtigsten Informationen zu geben.

Nach dieser etwas umfangreichen Erläuterung kehren wir nun zur Beschreibung der Zeropage zurück.

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

Tabelle 1. Die Adreßbelegung der Zeropage beim VC 20

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 oder Zwischenerg.)
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-Zeilennummer
59 - 60	3B - 3C	Vorherige Basic-Zeilennummer
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 (Hauptsächlich für Strings)
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 : Matisse
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

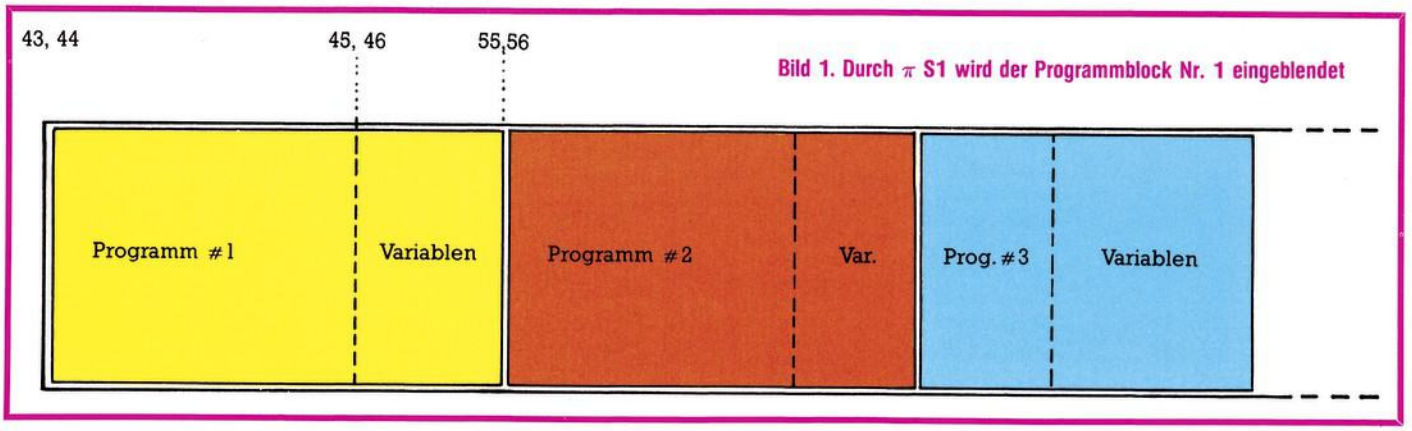


Bild 1. Durch π S1 wird der Programmblock Nr. 1 eingeblendet

110	6E	Fließkommaakku #2 : Vorzeichen
111	6F	Vergleichsbyte der Vorzeichen von Akku #1 und Akku #2
112	70	Rundung für Akku #1
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	9)	
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 Filenamen
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 1. Die Adreßbelegung der Zeropage beim VC 20 (Fortsetzung)

Adresse 63 — 66: Mit Hilfe dieser Zeiger läßt sich ein zeilennummerngesteuertes RESTORE realisieren. Die ersten zwei Bytes enthalten die momentane Zeilennummer für die DATA-Abfrage. Dies ist sehr praktisch, denn damit kann man jederzeit feststellen, in welcher DATA-Zeile man sich befindet. Das zweite Doppelbyte bildet die Adresse für die nächste DATA-Zeile. Man hat also zwischen Zeilennummer und der absoluten Adresse einer DATA-Zeile zu unterscheiden.

Das nun folgende kurze Maschinenprogramm (Listing 4 und 5) ermöglicht ein Zeilennummern-RESTORE. Man kann damit zwar nicht gezielt auf ein Element in einer DATA-Zeile zugreifen, aber zumindest auf eine bestimmte Zeile.

Was auch im Zusammenhang mit dem Maschinensprachkurs von Interesse sein kann, ist der Aufbau dieser kurzen Routine. Denn durch geschicktes Nutzen von Unterprogrammen aus dem ROM kann man nämlich viel Speicherplatz sparen. Aus diesem Grunde werde ich in einer späteren Folge Interpreter- und Betriebssystemunterroutinen näher beleuchten.

Das vorliegende Maschinenprogramm benötigt genau 43 Byte und liegt am Ende des verfügbaren Basic-Speichers. Die Start- und Endadresse wird vom Lader angegeben. Aufgerufen wird der Zeilenrestore mit: SYS »Startadresse«, »Zeilennumer«.

Die DATA-Zeiger werden dann auf die angegebene Zeile zurückgestellt, was sowohl im Direktmodus, als auch vom Programm aus erfolgen kann.

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 2 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 3).

So kommen Programme auf's Band

Jeder Abspeichervorgang 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) 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 abgespeichert wurde. Das Gegenstück dazu ist die absolute Lademethode (Headertyp 3), die bereits in der ersten Folge meiner Abhandlung vorgestellt wurde. Gemeint ist LOAD"..",1,1. Die Sekundäradresse 1 signalisiert dem Computer, daß er das Proramm (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 abgespeichert. 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 Bewandtnis. Sie wird nämlich nach korrektem Laden der Zeropage (Adresse 45, 46) übergeben. Bei 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.

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 ist, wird der Header gleich noch einmal abgespeichert.

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 abgespeichert.

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 VC 20 das Ende des geladenen Programms an, womit der Ladevorgang beendet ist.

Nach diesem Exkurs zum Kassettenaufzeichnungsformat nun wieder zur Zeropage.

Die STOP-Taste mit Sonderfunktion

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

Wert =

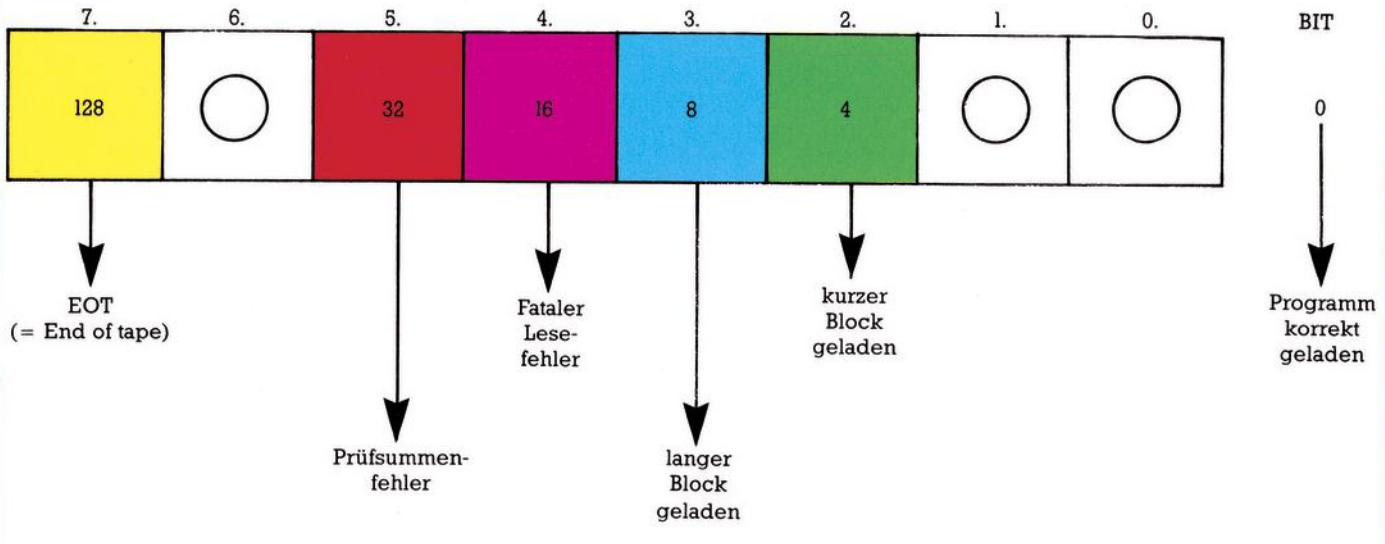
253: Linke SHIFT-Taste gedrückt

254: STOP-Taste gedrückt

255: Keine der beiden gedrückt

Wenn man das Low-Byte des STOP-Vektors (Adresse 808) auf 114

Bild 2. Darstellung des Statusbytes, das verschiedene Fehlermeldungen enthalten kann. Die Bits 0, 1 und 6 werden bei Bandbetrieb nicht genutzt. Daher sind sie auf Null.



— statt ursprünglich 112 — setzt, 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.

Programmiert oder direkt? Das ist hier die Frage

Adresse 157: Dieses Flag kann nur zwei Zustände annehmen: Entweder 0 oder 128. Ist der Inhalt der Speicherstelle Null, dann arbeitet der Computer gerade ein Programm ab; bei 128 befindet er sich im Direktmodus.

Diese Zeropageadresse kann dann von Bedeutung sein, wenn man eigene Basic-Kommandos definieren will (vergleiche Teil 1).

Zum Beenden einer solchen Routine gibt es zwei Möglichkeiten: Entweder kam der Programmbefehl aus dem Direktmodus. In diesem Fall springt man in die Interpreterschleife (JMP $C474) zurück, damit der Computer READY meldet. Kam das Kommando aber aus einem Basic-Programm, so muß man mit JMP $0079 in die CHRGOT-Routine zurückspringen, damit kein Bereitschaftszeichen ausgegeben wird. Mit Hilfe dieses Flags trifft man hier die Unterscheidung.

Nun wenden wir uns einem anderen Kapitel zu, der Cursorverwaltung. Über die Zeropage kann jederzeit die Cursorposition festgestellt werden. Die Speicherstelle 211 zeigt die Spalte, 214 die Zeile, in der sich der Blinker derzeit befindet. Korrekte Werte liefern die zwei Adressen aber nur innerhalb eines Programms.

Interessanter ist die umgekehrte Verfahrensweise, nämlich das Setzen des Cursors an eine beliebige Bildschirmposition. Dazu müssen allerdings vier Adressen geändert werden. Glücklicherweise nimmt uns eine Unterroutine aus dem Betriebssystem diese Arbeit ab.

Gewußt wo — der Cursor

In Maschinensprache wird das X-Register mit dem Zeilenwert, das Y-Register mit dem Spaltenwert geladen. Danach ruft man mit JSR $E50C das Unterprogramm auf.

Von Basic aus werden die zwei CPU-Register über die Adressen 781 und 782 geladen (warum das so ist, sehen wir in der nächsten Folge): POKE 781,»Zeile«: POKE 782,»Spalte«: SYS 58636.

Diese Verfahrensweise ist, wie man sieht, wesentlich komfortabler als das Hantieren mit den Steuerzeichen.

Eine andere Zeropageadresse kann uns zu einer weiteren Erleichterung verhelfen. Jeder kennt das Problem, den Cursor innerhalb von Anführungszeichen (»Gänsefüßchen«) zu bewegen. Es werden nur Steuerzeichen ausgedruckt, eine Bewegung des Zeigers findet nicht statt.

Dieses Manko läßt sich durch eine kurze Maschinenroutine beheben:

```
LDA #$00        ;
lösche Hochkommaflag
STA $D4         ;
speichere es ab
JMP $FEAD       ;
springe in die NMI-Routine
```

Ausgelöst wird diese Routine durch Drücken der RESTORE-Taste. Dazu muß vorher der NMI-Vektor (eine nähere Beschreibung nächstes Mal) auf den Beginn unserer Routine gestellt werden. Da die sehr kurz ist, legen wir sie im Kassettenpuffer ab. Dort ist sie solange sicher verwahrt, bis man etwas laden oder abspeichern will. Hier zunächst der Einzeiler, der die Routine in den Bandpuffer generiert:

```
10 FOR T=828 TO 834: READD: POKE T,D:
NEXT: DATA 169, 0, 133, 212, 76, 173,
254: POKE 792, 60: POKE 793, 3
```

Arbeitet man im Hochkommamodus, so kann man durch Drücken der RESTORE-Taste diesen Betriebszustand abschalten, der Cursor kann dann wieder ganz normal bewegt werden.

Abschließend möchte ich noch sagen, welche Zeropageadressen

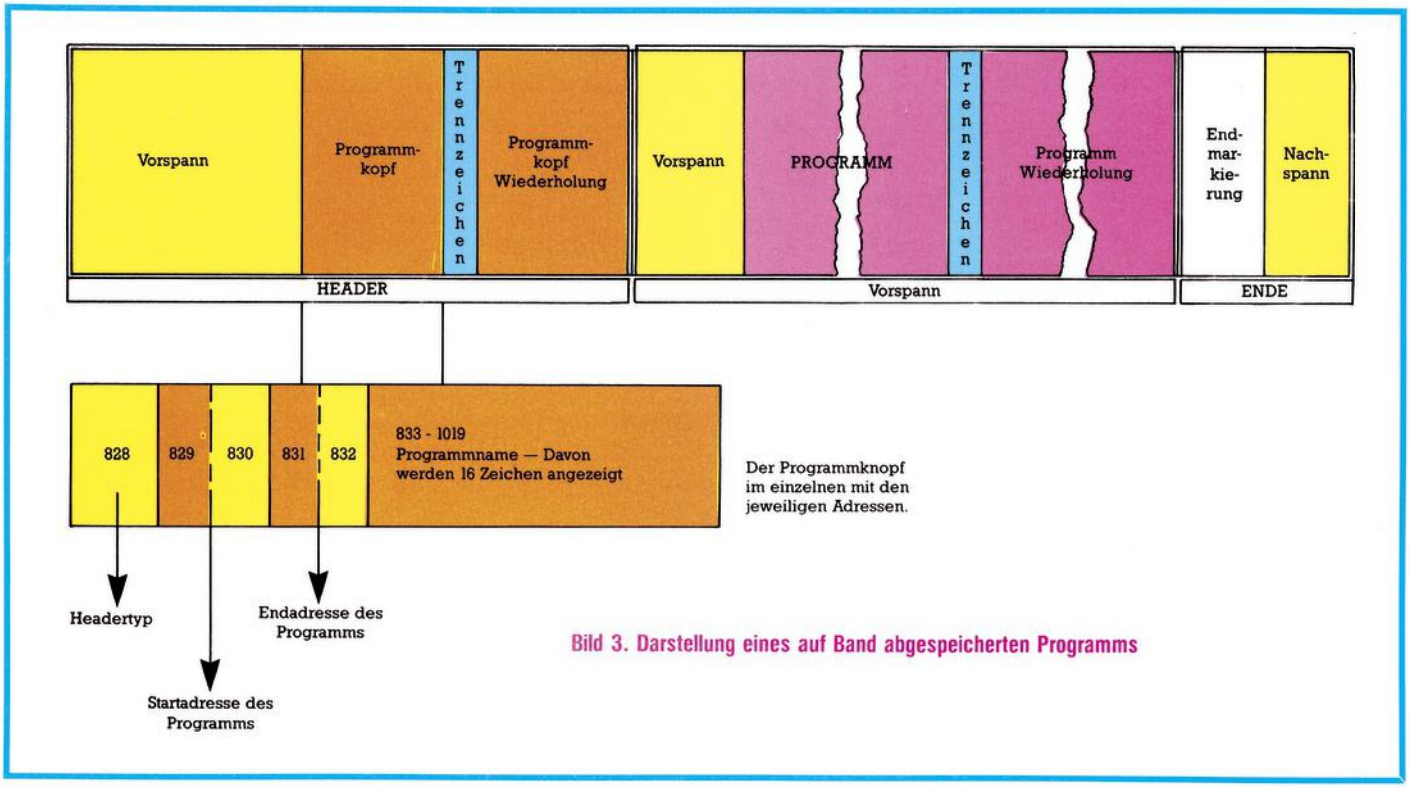


Bild 3. Darstellung eines auf Band abgespeicherten Programms

ohne Komplikationen von eigenen Maschinenroutinen genutzt werden können, was im hohen Maße vom Verwendungszweck abhängt.

Nicht verändert werden dürfen solche Speicherstellen, die vom Betriebssystem gebraucht werden, also die Adressen 192 bis 244 und 160 bis 162. Hat das Maschinenprogramm keine Verbindung zu Basic, so können alle Speicherstellen zwischen 0 und 138 überschrieben werden, weil sich hier nur Basic-Parameter befinden. Eng wird der Platz für die sogenannten Utilities — also für Basic-Ergänzungen oder Hilfsprogramme. Will man Daten nur zwischenspeichern, kann man sie in den Bereich zwischen Adresse 147 bis 159 packen. Dort sind sie bis zum nächsten Lade- oder Abspeichervorgang sicher.

Dauerhaft können Daten nur am Anfang und am Ende der Zeropage abgelegt werden. Dazu gehören Adresse 0 bis 2 und 245 bis 254.

Genau diese werden aber auch von käuflichen Basic-Erweiterungen gebraucht, wodurch es zu Komplikationen kommen kann. Also Vorsicht, erst durch vorheriges Probieren testen, ob man sie benutzen darf!

Damit möchte ich für heute schließen. In der nächsten Folge werden schwerpunktmäßig der Tastaturpuffer, die Basic-Vektoren und die Kernalvektoren behandelt.

(Christoph Sauer/ev)

Listing 3 finden Sie auf Seite 164

Listing 4. »Zeilen-RESTORE« (Basic-Lader)

```
10 REM ************************************
*********
20 REM ****
     ****
30 REM **** ZEILENNUMMERNGESTEUERTES RES
TORE ****
40 REM ****
     ****
50 REM ****       1984 BY CHRISTOPH SAUER
     ****
60 REM ****  HUBERTUSSTR.14 / 8000 MUEN.
 19  ****
70 REM ****
     ****
80 REM ************************************
*********
90 FORT=0TO47:READD:S=S+D:NEXT:IFS<>5219
THENPRINT"FEHLER":END
100 POKE55,0:POKE56,PEEK(56)-1:CLR:A=PEE
K(56)
110 RESTORE:FORT=A*256TOA*256+47
120 READD
130 POKET,D
140 NEXT
150 PRINT"■■■FERTIG. START MIT":PRINT"■S
YS"A*256
160 DATA032,121,000,201,044
170 DATA240,003,076,008,207
180 DATA032,115,000,032,138
190 DATA205,032,247,215,165
200 DATA020,133,063,165,021
210 DATA133,064,032,019,198
220 DATA032,248,200,165,095
230 DATA164,096,056,233,001
240 DATA032,036,200,076,116
250 DATA196,116,196
READY.
```

Listing 5. »Zeilen-RESTORE« (Assembler-Darstellung)

```
1D00 JSR $0079   ; CHRGOT, LFD. BASICZEICHEN
1D03 CMP #$2C    ; KOMMA ?
1D05 BEQ $1D0A   ; JA, DANN VERZWEIGEN
1D07 JMP $CF08   ; SONST 'SYNTAX ERROR'
1D0A JSR $0073   ; CHRGET, NAECHSTES ZEICHEN
1D0D JSR $CD8A   ; AUSDRUCK HOLEN
1D10 JSR $D7F7   ; IN 2-BYTE FORMAT WANDELN
1D13 LDA $14     ; UEBERTRAGUNG DER ZEILENNR. (LB)
1D15 STA $3F
1D17 LDA $15     ; HB UEBERTRAGEN
1D19 STA $40
1D1B JSR $C613   ; ADRESSE DER DATAZEILE BERECHNEN
1D1E JSR $C8F8   ; NAECHSTE DATAZEILE SUCHEN
1D21 LDA $5F
1D23 LDY $60
1D25 SEC
1D26 SBC #$01
1D28 JSR $C824   ; RESTORE DURCHFUEHREN
1D2B JMP $C474   ; RUECKSPRUNG INS BASIC
```

Listing 3. »Basic-Switch« (Assembler-Darstellung)

```
BASICSWITCH

****************** CHRGET ERGAENZUNG
1C00 INC $7A     ; CHRGET FORTFUEHREN
1C02 BNE $1C06
1C04 INC $7B
1C06 JSR $0079
1C09 CMP #$FF    ; TOKEN = ◢ ?
1C0B BEQ $1C10   ; JA, DANN VERZWEIGEN
1C0D JMP $0079   ; SONST ZURUECK INS BASIC
1C10 JSR $0073   ; CHRGET, NAECHSTES ZEICHEN
1C13 CMP #$45    ; 'E' FUER ERGAENZUNG
1C15 BNE $1C1A   ; NEIN/ DANN WEITER
1C17 JMP $1CA8   ; SONST IN ERGAENZUNGSROUTINE SPRINGEN
1C1A CMP #$53    ; 'S' FUER SWITCH ?
1C1C BEQ $1C21   ; JA, DANN VERZWEIEN
1C1E JMP $CF08   ; SONST 'SYNTAX ERROR'
****************** ◢S FUER SWITCH
1C21 JSR $D79B   ; BYTEWERT (0-255)HOLEN
1C24 LDA $65
1C26 BEQ $1C2C   ; WENN WERT=0, DANN FEHLER
1C28 CPX $FB
1C2A BCC $1C2F   ; ? WENN WERT<ANZAHL DER PROGRAMME, DANN WEITER
1C2C JMP $D248   ; SONST FEHLERMELDUNG
1C2F STX $FC     ; PROGRAMMNUMMER ABSPEICHERN
1C31 JSR $1C85   ; ADRESSEN 43-56 IN ZWISCHENSPEICHER UBERNEHMEN
1C34 LDX $FC     ; EINGABEWERT * 8
1C36 DEX
1C37 TXA
1C38 ASL
1C39 ASL
1C3A ASL
1C3B TAY
1C3C LDX #$00    ; ZWISCHENSPEICHER IN REGISTER 43-46 SCHREIBEN
1C3E LDA $1D20,Y
1C41 STA $2B,X
1C43 INY
1C44 INX
1C45 CPX #$04
1C47 BNE $1C3E
1C49 LDX #$00    ; ZWISCHENSPEICHER IN REGISTER 55-56 SCHREIBEN
1C4B LDA $1D20,Y
1C4E STA $37,X
1C50 INY
1C51 INX
1C52 CPX #$02
1C54 BNE $1C4B
1C56 LDA $FC     ; LFD. PROGRAMMNR.
1C58 STA $FA     ; GLEICH ALTER PROGRAMMNR.
1C5A JSR $C660   ; BASIC BEFEHL CLR
****************** REM- ZEILA AUSDRUCKEN
1C5D JSR $C68E   ; CHRGET ZEIGER (7A,7B) AUF BASICSTART
1C60 JSR $0073   ; CHRGET, NEUES KOMANDO HOLEN
1C63 BEQ $1C6B   ; BEIM ZEILENENDE VERZWEIGEN
1C65 CMP #$8F    ; 'REM' CODE ?
1C67 BEQ $1C79   ; JA, DANN AUSDRUCKEN
1C69 BNE $1C60   ; SONST ZURUECK
1C6B LDY #$01
1C6D LDA ($7A),Y; ZEILENENDE AUCH PROGRAMMENDE ?
1C6F BNE $1C60   ; NEIN, DANN ZURUECK
1C71 INY
1C72 CPY #$02
1C74 BNE $1C6D
1C76 JMP $C474   ; PROGRAMMENDE, DANN IN BASIC DIREKTMODUS
1C79 JSR $CAD7   ; LEERZEILE AUSGEBEN
1C7C LDA $7A     ; AKKU   AUF L-BYTE DES STRINGS
1C7E LDY $7B     ; Y REG. AUF H-BYTE DES STRINGS
1C80 JSR $CB1E   ; STRING (=REM ZEILE) AUSDRUCKEN
1C83 BEQ $1C76   ; ZURUECK INS BASIC
****************** REGISTER ABSPEICHERN
1C85 LDX $FA     ; ALTE PROGRAMMNR. * 8
1C87 DEX
1C88 TXA
1C89 ASL
1C8A ASL
1C8B ASL
1C8C TAY
1C8D LDX #$00    ; ZEIGER 43-46 IN ZWISCHENSPEICHER UEBERNEHMEN
1C8F LDA $2B,X
1C91 STA $1D20,Y
1C94 INY
1C95 INX
1C96 CPX #$04
1C98 BNE $1C8F
1C9A LDX #$00    ; ZEIGER 55-56 IN ZWISCHENSPEICHER
1C9C LDA $37,X
1C9E STA $1D20,Y
1CA1 INY
1CA2 INX
1CA3 CPX #$02
1CA5 BNE $1C9C
1CA7 RTS         ; ZURUECK
****************** ◢E FUER ERGAENZUNG
1CA8 JSR $1C85   ; ZEIGER (43-56) IN ZWISCHENSPEICHER
1CAB JSR $0073   ; CHRGET, NEUES ZEICHEN HOLEN
1CAE JSR $CD8A   ; NUMERISCHEN AUSDRUCK HOLEN
1CB1 JSR $D1B8   ; FLIESKOMMA IN 2-BYTE FORMAT WANDELN
1CB4 LDA $FD     ; HOECHSTE ADRESSE DES 1. PROGRAMMS =
1CB6 STA $2B     ; NIEDRIGSTE ADRESSE DES 2. PROGRAMMS
1CB8 LDA $FE
1CBA STA $2C
1CBC LDA $65     ; EINGEGEBENER WERT
1CBE CLC
1CC1 STA $FD     ; UND ABSPEICHERN (L-BYTE)
1CC3 LDA $64
1CC5 ADC $FE     ; ADDITION DER H-BYTES
1CC7 STA $FE     ; UND ABSPEICHERN
1CC9 LDA $FB     ; ANZAHL DER VERWALTETEN PROGRAMME HOLEN
1CCB STA $FC     ; UND DER LFD. PROGRAMMNR. GLEICHSETZEN
1CCD INC $FB     ; ? ANZAHL DER PROGRAMME UM 1 ERHOEHEN
1CCF LDA $2B     ; BASICANFANG UM 1 ZURUECK
1CD1 BNE $1CD5
1CD3 DEC $2C
1CD5 DEC $2B
1CD7 JSR $E3F8   ; 0 AN DEN BASICANFANG UND BA. UM EINS ERHOEHEN
1CDA LDA $FE     ; UEBERPRUEFUNG, OB DIE NEUE BASICENDADRESSE
1CDC CMP $0284   ; UEBERHAUPT NOCH IM SPEICHERBEREICH LIEGT
1CDF BCC $1CF1
1CE1 BNE $1CEA
1CE3 LDA $FD
1CE5 CMP $0283
1CE8 BCC $1CF1   ; LIEGT ADRESSE IM SPEICHERBEREICH, DANN WEITER
1CEA LDA #$0C    ; SONST FEHLERMELDUNG (L-BYTE DER FM LADEN)
1CEC LDY #$CD    ; H-BYTE LADEN
1CEE JMP $CB1E   ; STRING 'REDO FROM START' AUSDRUCKEN
1CF1 LDA $FD     ; BERECHNETES BASICENDE
1CF3 STA $37     ; DEM ECHTEN GLEICHSETZEN
1CF5 LDA $FE
1CF7 STA $38
1CF9 JSR $C644   ; BASIC ROUTIENE NEW
1CFC LDA $FC     ; LFD. PROGRAMMNR. =
1CFE STA $FA     ; ALTE PROGRAMMNR
1D00 JMP $1C31   ; EINSPRUNG IN DIE SWITCH ROUTINE
****************** INITIALISIERUNG
1D03 LDX #$00
1D05 STX $FA     ; ALTE PROGRAMMNR. INITIALISIEREN
1D07 STX $74     ; CHRGET AENDERN
1D09 INX
1D0A STX $FB     ; ANZAHL DER PROGRAMME UND
1D0C STX $FC     ; LFD. PROGRAMMNR. AUF1
1D0E LDA $2B
1D10 STA $FD     ; HIMEM ZEIGER INITIALISIEREN
1D12 LDA $2C
1D14 STA $FE
1D16 LDA #$4C    ; CHRGET AENDERN
1D18 STA $73
1D1A LDA #$1C
1D1C STA $75
1D1E RTS
```

Listing 3. »Basic-Switch« (Assembler-Darstellung, Schluß)