Der gläserne VC 20

Teil 1

Der VC 20, schon etwas betagt und oft genug totgesagt, ist immer noch der meistverbreitete Computer seiner Klasse.

Mit diesem Kurs wollen wir den legendären »Volkscomputer« endlich für alle Anwender vollkommen transparent machen.

Das Betriebssystem und das Basic des VC 20 sind äußerst flexibel gestaltet. Es gibt viele Möglichkeiten, das Bestehende zu ändern oder zu ergänzen. Diese Serie soll über die üblichen Tips und Tricks hinausgehen. Es werden also nicht nur POKEs, sondern auch weitergehende Maschinenprogramme wie zum Beispiel Funktionstastenabfrage oder die Definition neuer Basic-Befehle besprochen. Dieser erste Teil soll dabei bereits einen tieferen Einblick in das VC 20-System geben.

Wie Basic den Speicher verwaltet

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

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

Der Variablenbereich wächst beim Anlegen neuer Variablen von unten nach oben. Nur das Stringende wandert in entgegenlaufender 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 * 256 + Low-Byte).

Um Speicherplatz für Maschinenprogramme oder Sonderzeichen zu schaffen, hat man prinzipiell zwei Möglichkeiten. Entweder man verschiebt den Basicanfang nach oben oder das Basicende nach unten. Letztere Alternative ist in den meisten Fällen günstiger.

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

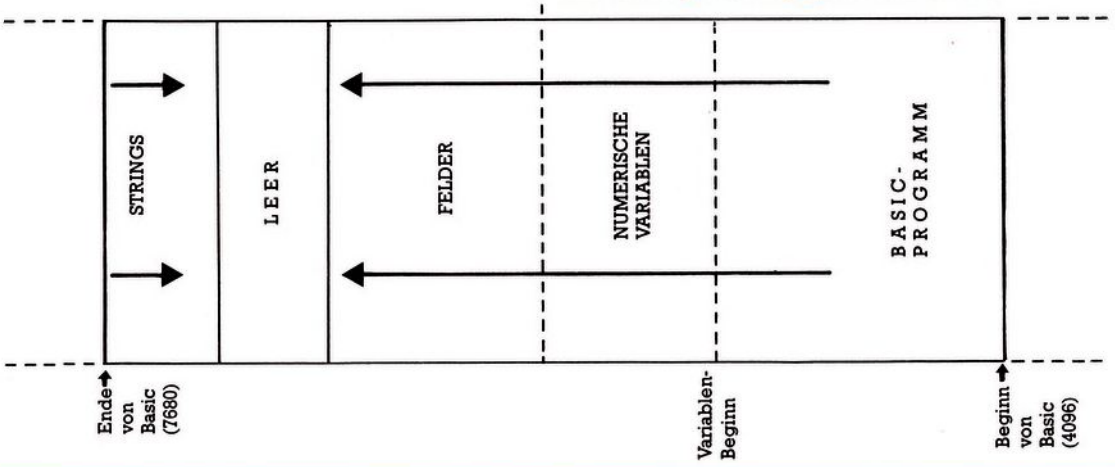
Bei anderen Speichergrößen verfährt man analog.

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

Tabelle 1. Der Inhalt dieser Speicherstellen bestimmt die Aufteilung des Basic-RAM

Speicherstelle	Bezeichnung	Werte
		GV : 4096
43	BASIC	+ 3K : 1024
44	BEGINN	+ 8K : 4608
45	VARIABLEN	Abhängig von der
46	BEGINN	Programmlänge
		GV : 7680
		+ 3K : 7680
55	BASIC	+ 8K : 16384
56	ENDE	+ 16K : 24576
		+ 24K : 32768

Bild 1. Die Speicherbelegung des Basicbereichs



Die andere Alternative der Platzbeschaffung ist etwas komplizierter. 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 Basicbereichs immer ein Nullbyte stehen muß.

Erste Hilfe — Basicprogramme retten nach NEW oder Reset

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

Am Kopf des Programms steht immer eine Null. Dann folgt die Adresse der nächsten Programmzeile (Koppeladresse) und die Zeilennummer. Danach kommt die eigentliche Programmzeile, die sich aus den sogenannten Tokens — den Basiccodenummern (Tabelle 2) — zusammensetzt. Am Ende dieser Zeile steht dann nochmals eine Null. Die

Tabelle 2. VC 20 Basic-Token. Die Codenummern 32 bis 95 entsprechen den ASCII-Codes. 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.

Code (Dezimal)	Zeichen/Befehl	Code (Dezimal)	Zeichen/Befehl	Code (Dezimal)	Zeichen/Befehl	Code (Dezimal)	Zeichen/Befehl
0	Zeilenende	66	B	133	INPUT	169	STEP
1-31	Leer	67	C	134	DIM	170	+
32	Space	68	D	135	READ	171	—
33	!	69	E	136	LET	172	*
34	"	70	F	137	GOTO	173	/
35	#	71	G	138	RUN	174	↑
36	$	72	H	139	IF	175	AND
37	%	73	I	140	RESTORE	176	OR
38	&	74	J	141	GOSUB	177	>
39	'	75	K	142	RETURN	178	=
40	(	76	L	143	REM	179	<
41	)	77	M	144	STOP	180	SGN
42	*	78	N	145	ON	181	INT
43	+	79	O	146	WAIT	182	ABS
44	,	80	P	147	LOAD	183	USR
45	—	81	Q	148	SAVE	184	FRE
46	.	82	R	149	VERIFY	185	POS
47	/	83	S	150	DEF	186	SQR
48	0	84	T	151	POKE	187	RND
49	1	85	U	152	PRINT#	188	LOG
50	2	86	V	153	PRINT	189	EXP
51	3	87	W	154	CONT	190	COS
52	4	88	X	155	LIST	191	SIN
53	5	89	Y	156	CLR	192	TAN
54	6	90	Z	157	CMD	193	ATN
55	7	91	[	158	SYS	194	PEEK
56	8	92	£	159	OPEN	195	LEN
57	9	93	]	160	CLOSE	196	STR$
58	:	94	↑	161	GET	197	VAL
59	;	95	←	162	NEW	198	ASC
60	<	96-127	Leer	163	TAB(	199	CHR$
61	=	128	END	164	TO	200	LEFT$
62	>	129	FOR	165	FN	201	RIGHT$
63	?	130	NEXT	166	SPC(	202	MID$
64	@	131	DATA	167	THEN	203-254	Leer
65	A	132	INPUT	168	NOT	255	

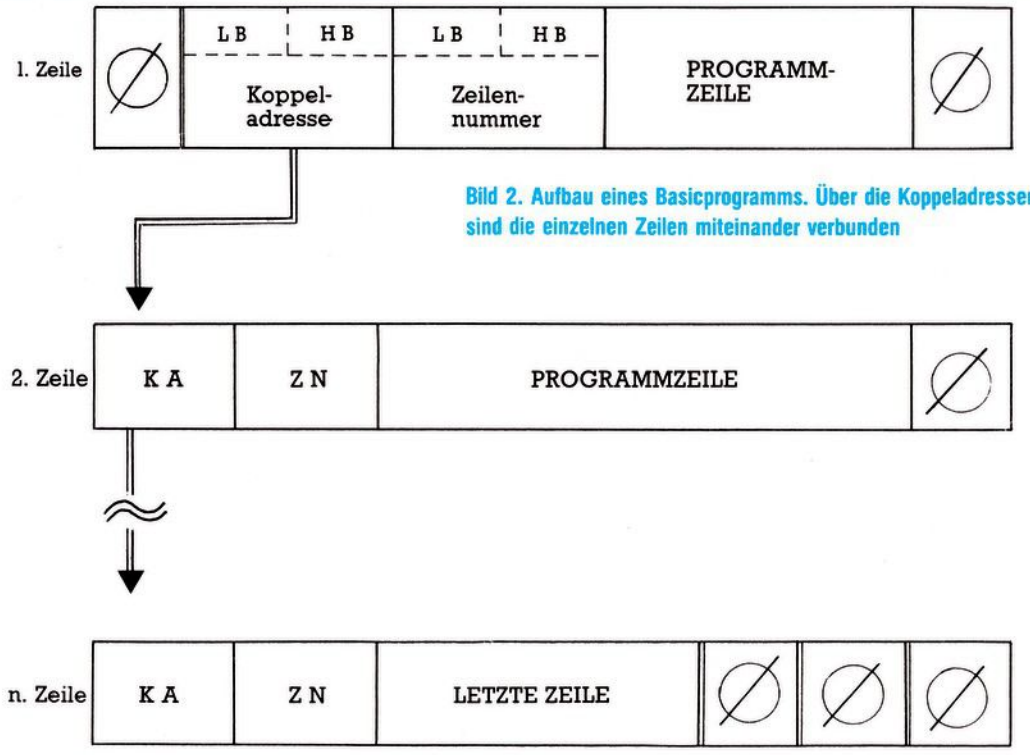


Bild 2. Aufbau eines Basicprogramms. Über die Koppeladressen sind die einzelnen Zeilen miteinander verbunden

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 Basicprogramm wieder benutzt werden.

Hier nun das »Rezept« zur Rekonstruktion:

- POKE (Basicanfang) + 2,1

Basicanfang in GV = 4097
+ 3K = 1025
+ 8K = 4609

- SYS 50483:POKE 46,PEEK(35): POKE 45, PEEK (781)+2:CLR

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 (SYS 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.

Die CHRGET-Routine

Die Zeropage ist in Maschinensprache besonders einfach zu adressieren. Aus diesem Grund sind hier oft benötigte Daten abgelegt. Doch die Seite Null beheimatet auch ein Unterprogramm aus dem Basicinterpreter namens CHRGET (CHaRacter GET, Tabelle 3). Diese Routine hat die Aufgabe, aus dem Basictext einzelne Zeichen oder Befehle zur Auswertung bereitzustellen. Sie befindet sich gerade deshalb im RAM-Speicher, weil sie einen veränderbaren 2-Byte-Zeiger enthält. Da die Routine bei jeder Ausführung eines Basicbefehls benutzt wird, bietet sich hier eine gute Möglichkeit, in den Ablauf einzugreifen, um damit den Befehlsvorrat zu erweitern. CHRGET endet mit einem Sprung zurück zur Befehlsauswertung. Da die CHRGET-Routine im RAM liegt, kann an dieser Stelle die Routine in das Befehlsauswertungsprogramm des Benutzers umgeleitet werden. Dort wird zuerst das CHRGET-Unterprogramm zu Ende geführt.

Als Beispiel soll der bestehende Befehl π (Tokennummer 255) geändert werden. Das Befehlsauswertungsprogramm nach Tabelle 4 fragt ihn ab und verzweigt dann nach § 1C16, wo ein RESET ausgeführt wird (entspricht SYS 64802).

Man kann aber auch noch zusätzliche Parameter abfragen. Die nun folgende Änderung des π-Befehls steuert den Tongenerator, wobei drei Argumente und zwei Kommata geprüft werden müssen. Die Syntax des neuen Befehls ist aus Tabelle 5 ersichtlich.

Die Steuerungsroutine wird nun aus diesen vorgefertigten Modulen zusammengesetzt.

Zuerst der Baustein zum Abfragen von Argumenten (Tabelle 6).

Bild 3. Die Speicheraufteilung beim Autostart, bezogen auf die Grundversion

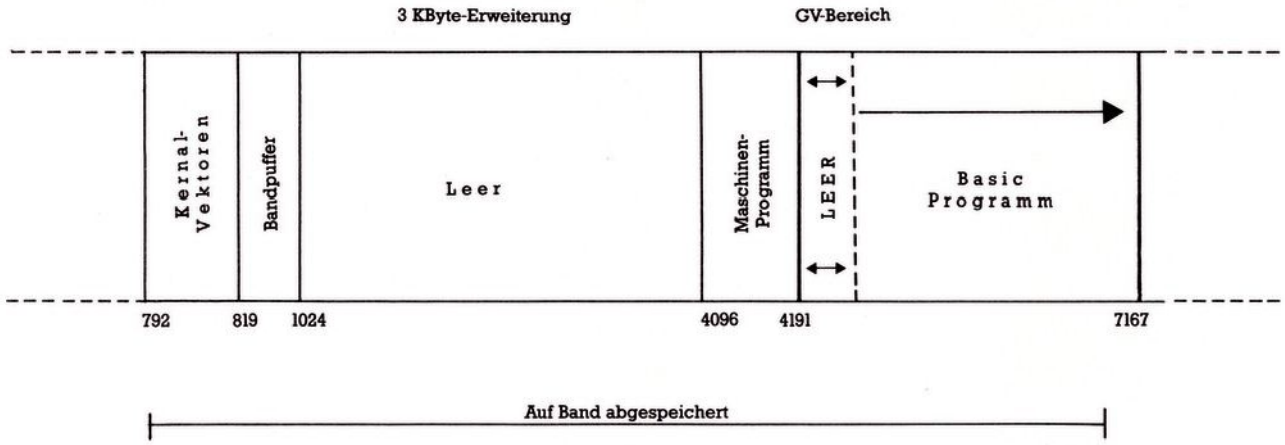


Tabelle 3. Assemblerdarstellung der »CHRGET«-Routine

```
CHRGET:
0073    INC $ 7A          ; erhöhe Lo-Byte
0075    BNE $ 0079        ; Übertrag ?
0077    INC $ 7B          ; ja, dann Hi-Byte erhöhen
0079    LDA $ Pointer     ; hole Zeichen aus dem Basictext
007C    JMP $ Befehlsauswertung
```

Tabelle 4. So könnte die »CHRGET«-Routine zu Ende geführt werden

```
Befehlsauswertung:
1C00    CMP #$ FF         ; Befehl PI ?
1C02    BEQ $ 1C16        ; Ja, dann RESET
1C04    CMP #$ 3A         ; Fortführung des alten CHRGET
1C06    BCS $ 1C15
1C08    CMP #$ 20
1C0A    BNE $ 1C0F
1C0C    JMP $ 0073
1C0F    SEC
1C10    SBC #$ 30
1C12    SEC
1C13    SBC #$ D0
1C15    RTS
1C16    JMP $ FD22        ; eigene Befehlsverarbeitung:
                          RESET
```

Die ROM-Routine ($D79B) holt sich aus dem Basictext den numerischen Ausdruck und stellt ihn im X-Register zur Verfügung. Ist der Wert größer als 255, wird eine Fehlermeldung ausgegeben. Die Syntax unseres Befehls erlaubt aber nur Argumente zwischen 0 und 3. Daher wird nochmals eine Bereichsprüfung vorgenommen.

Als nächstes wird das Komma abgefragt (Tabelle 7). CHRGOT holt das laufende Zeichen aus dem Basictext und die Routine vergleicht es mit dem ASCII-Code für das Komma.

Man unterscheidet im übrigen zwischen CHRGET und CHRGOT. CHRGET ($0073) stellt den Zwei-Byte-Zeiger um eins hoch und lädt dann das neue Zeichen in das Akku. CHRGOT ($0079) hingegen holt lediglich das Zeichen.

Um den Soundbefehl zu komplettieren müssen noch die restlichen drei Module eingebaut werden. Wir wollen an dieser Stelle jedoch darauf verzichten, das im einzelnen auszuführen. Es sollte jetzt jedoch klar geworden sein, wie man eine Befehlserweiterung realisieren kann.

Für Assembler-Laien jetzt noch ein komplettes Befehlsprogramm. Es erweitert die bestehenden Kommandos um:

π O ≙ Old (Rekonstruktion),

π **S Tongenerator, Höhe, Lautstärke** ≙ Soundbefehl (wie oben)

π **S Tongenerator, 0** ≙ Tongenerator abschalten,

π **P Horizontal, Vertikal,** "." oder

π **P Horizontal, Vertikal, String** ≙ Druck an einer spezifizierten Bildschirmstelle.

Das Ladeprogramm (Listing 1) lädt die Maschinenroutine automatisch in den richtigen Speicherbereich (abhängig von der Speichergröße) und gibt dann die Startadresse an. Zur Referenz ist das vollständige Assemblerprogramm als Listing 2 abgedruckt.

Die neuen Befehle können sowohl im Direktmodus, als auch im Programm verwendet werden. Benutzt man sie im Programm, so ist zu beachten, daß sie nie direkt nach der Zeilennummer stehen dürfen. So muß zum Beispiel der Befehl π S 1,240,15 mit Doppelpunkt im Programm stehen:

10 : π S1,240,15 oder

10 PRINT A$: π S1,240,15

Tabelle 7. Dieses Modul testet, ob ein Komma folgt.

```
1C22  JSR  $ 0079     ;CHRGOT (laufendes Zeichen)
1C25  CMP  # 2C       ;Komma?
1C27  BEQ  Weiter     ;ja, dann verzweigen
1C29  JMP  $ CF08     ; "Syntax error" ausgeben
1C2B  — Weiter —
```

Listschutz für Basicprogramme:

Es wurden bereits mehrfach Methoden veröffentlicht, mit denen man Programme vor unbefugtem Kopieren schützen kann. Hierbei gibt es mehrere Alternativen:

1. Man verändert die Koppeladressen so, daß das Programm nicht listfähig ist, es jedoch normal mit RUN bedient werden kann.

2. Man verändert den LIST-Vektor mit POKE 774,34:POKE 755,253. Bei einem Listversuch löst dieser Vektor einen RESET aus und das Programm ist weg.

Tabelle 5. Die Syntax des neuen π-Befehls

Tongenerator Bereich 0 — 3	,	Tonhöhe Bereich 0 — 255	,	Lautstärke Bereich 0 — 15

Tabelle 6. Das Modul zur Argumentabfrage

```
1C16  JSR  $ D79B   ; Numerisches Argument holen
1C19  CPX  #$ 04    ; Ausdruck größer als 4 ?
1C1B  BCC  $ 1C20   ; nein, dann verzweigen
1C1D  JMP  $ D248   ; sonst Fehlermeldung ausgeben
1C20  STX  $ FA     ; Wert in Zeropage ablegen
```

Listing 1. Basic-Lader zur Befehlserweiterung

```
100 REM ********************************
**
110 REM ****                           **
**
120 REM **** VC-20 BEFEHLSAUSWERTUNG   **
**
130 REM ****                           **
**
140 REM **** 1984 BY CHRISTOPH  SAUER **
**
150 REM ****                           **
**
160 REM ****    HUBERTUSSTRASSE 14     **
**
170 REM ****                           **
**
180 REM ****    8000  MUENCHEN  19     **
**
190 REM ****                           **
**
200 REM ********************************
**
210 FORT=0TO215:READD:P=P+D:NEXT:IFP<>23
243THENPRINT"■FEHLER !!":END
220 POKE55,0:POKE56,PEEK(56)-1:CLR:A=PEE
K(56)
230 RESTORE:FORT=A*256TOA*256+215
240 READD
250 IFD=-1THEND=A
260 POKET,D
270 NEXT
280 PRINT"■■FERTIG. START MIT":PRINT"■S
YS"A*256+203
290 PRINT"■■AKTIVIERUNG MIT 'A'"
300 GETA$:IFA$=""THEN300
310 IFA$<>"A"THENEND
320 SYSA*256+203:πS1,240,15:FORT=1TO500:
NEXT:πS1,0
330 DATA201,255,240,018,201
340 DATA058,176,013,201,032
350 DATA208,003,076,115,000
360 DATA056,233,048,056,233
370 DATA208,096,032,115,000
380 DATA201,079,208,003,076
390 DATA049,-01,201,080,208
400 DATA003,076,075,-01,201
410 DATA083,208,003,076,134
420 DATA-01,076,008,207,160
430 DATA001,145,043,032,051
440 DATA197,165,034,024,105
450 DATA002,133,045,165,035
460 DATA105,000,133,046,032
470 DATA096,198,076,116,196
480 DATA032,155,215,224,023
490 DATA144,003,076,072,210
500 DATA134,250,032,121,000
510 DATA201,044,240,003,076
520 DATA008,207,032,155,215
530 DATA224,022,144,003,076
540 DATA072,210,032,121,000
550 DATA201,044,240,003,076
560 DATA008,207,024,138,168
570 DATA166,250,032,010,229
580 DATA032,115,000,032,160
590 DATA202,076,193,-01,032
600 DATA155,215,224,004,144
610 DATA003,076,072,210,134
620 DATA250,032,121,000,201
630 DATA044,240,003,076,008
640 DATA207,032,155,215,134
650 DATA251,240,023,032,121
660 DATA000,201,044,240,003
670 DATA076,008,207,032,155
680 DATA215,224,016,144,003
690 DATA076,072,210,142,014
700 DATA144,166,250,165,251
710 DATA157,010,144,165,157
720 DATA016,003,076,116,196
730 DATA076,121,000,169,076
740 DATA133,124,169,000,133
750 DATA125,169,-01,133,126
760 .DATA096
READY.
```

Diese und andere Schutzmaßnahmen haben den Nachteil, daß sie von »Hackern« innerhalb kurzer Zeit umgangen werden können. Es gibt zwar keinen hundertprozentigen Programmschutz, jedoch bietet die nachfolgend beschriebene Methode eine große Sicherheit. Bei diesem Verfahren läßt die Änderung eines Kernalvektors (Tastatureingabe 804/805) das Basicprogramm mit Hilfe einer Maschinenroutine nach Abschluß des Ladevorgangs automatisch starten.

Zunächst zur Verfahrensweise beim Autostart:
Schritt 1:
Eingabe des Ladeprogramms (Listing 3)
Schritt 2:
Programm und Prüfsumme testen (Achtung es zerstört sich selbst mit NEW) und abspeichern
Schritt 3:
POKE 44,A : POKE A*256,0 : NEW (A = 17 für die Grundversion; A = 19 bei Erweiterung ab 8 KByte; A = 5 Erweiterung + 3 KByte
Schritt 4:
Ladeprogramm laden und starten
Schritt 5:
Eigenes Programm nachladen
Schritt 6:
POKE 43,24 : POKE 44,3
Schritt 7:
POKE 792,91 : POKE 793,255 : POKE 808,114
Schritt 8:
POKE 804,0 : POKE 805,X : SAVE "...",1,1
X wird vom Ladeprogramm angegeben (X = 16 in Grundversion; X = 18 bei Erweiterung ab 8 KByte; X = 4 bei Erweiterung von 3 KByte). Die Befehle von Schritt 8 müssen unbedingt in einer Zeile stehen, sonst stürzt der Computer ab.

Das Ladeprogramm (oder einfacher der Lader) übernimmt die Abspeicherung des Maschinenprogramms, wobei er sich nach einer eventuell vorhandenen Speichererweiterung richtet.

Nun zur Bedienung:
Nach der Prüfsummenkontrolle ist die Speichergröße per Tastendruck einzugeben. Dadurch wird überprüft, ob man vor dem Laden POKE 44,A eingegeben hat, denn sonst würde sich das Programm selbst überschreiben. Dann wird nach der Anfangsadresse für das zu schützende Basicprogramm gefragt. Der Lader gibt der Einfachheit halber bereits die entsprechende Adresse vor. Man kann sie aber auch ändern, wodurch das Auffinden des Basic-

```
********************* CHRGET FORTSETZUNG
1D00  CMP #$FF      ; PI ?
1D02  BEQ $1D16     ; JA, DANN VERZWEIGEN
1D04  CMP #$3A      ; SONST CHRGET ZU ENDE FUEHREN
1D06  BCS $1D15
1D08  CMP #$20
1D0A  BNE $1D0F
1D0C  JMP $0073
1D0F  SEC
1D10  SBC #$30
1D12  SEC
1D13  SBC #$D0
1D15  RTS
1D16  JSR $0073     ; NAECHSTES ZEICHEN HOLEN
1D19  CMP #$4F      ; 'O' ?
1D1B  BNE $1D20     ; NEIN, DANN WEITER
1D1D  JMP $1D31     ; SONST VERZWEIGEN
1D20 CMP #$50       ; 'P' ?
1D22  BNE $1D27     ; NEIN, DANN WEITER
1D24  JMP $1D4B     ; SONST VERZWEIGEN
1D27  CMP #$53      ; 'S' ?
1D29  BNE $1D2E     ; NEIN, DANN FEHLER
1D2B  JMP $1D86     ; SONST VERZWEIGEN
1D2E  JMP $CF08     ; 'SYNTAX ERROR' AUSGEBEN
********************* O FUER REKONSTRUKTION
1D31  LDY #$01
1D33  STA ($2B),Y   ; AM BASICANFANG ABSPEICHERN
1D35  JSR $C533     ; BASICZEILEN NEU BINDEN
1D38  LDA $22
1D3A  CLC
1D3B  ADC #$02
1D3D  STA $2D
1D3F  LDA $23
1D41  ADC #$00
1D43  STA $2E       ; PROGRAMMENDE IN DEZ 43,44 ABSPEICHERN
1D45  JSR $C660     ; CLR DURCHFUEHREN
1D48  JMP $C474     ; ZURUECK IN DEN DIREKTMODUS
********************* P FUER POSITIONSDRUCK
1D4B  JSR $D79B     ; PARAMETER HOLEN
1D4E  CPX #$17      ; >23 ?
1D50  BCC $1D55     ; NEIN, DANN WEITER
1D52  JMP $D248     ; SONST FEHLERMELDUNG
1D55  STA $FA       ; HORIZONTAL POSITION ABSPEICHERN
1D57  JSR $0079     ; LFD. ZEICHEN HOLEN
1D5A  CMP #$2C      ; KOMMA ?
1D5C  BEQ $1D61     ; JA, DANN WEITER
1D5E  JMP $CF08     ; SONST SYNTAX ERROR
1D61  JSR $D79B     ; NAECHSTEN PARAMETER HOLEN
1D64  CPX #$16      ; >22 ?
1D66  BCC $1D6B     ; NEIN, DANN WEITER
1D68  JMP $D248     ; SONST FEHLERMELDUNG
1D6B  JSR $0079     ; NAECHSTES ZEICHEN HOLEN
1D6E  CMP #$2C      ; KOMMA ?
1D70  BEQ $1D75     ; JA, DANN WEITER
1D72  JMP $CF08     ; SONST SYNTAX ERROR
1D75  CLC           ; VORBEREITUNG FUER DAS UNTERPROGRAMM
1D76  TXA             CURSOR SETZEN
1D77  TAY
1D78  LDX $FA
1D7A  JSR $E50A     ; CURSOR AN POSITION (X/Y REG)
1D7D  JSR $0073     ; NAECHSTES ZEICHEN HOLEN
1D80  JSR $CAA0     ; STRING AUSWERTEN UND AUSGEBEN
1D83  JMP $20C1     ; ROUTINE ABSCHLIESSEN
********************* S FUER SOUND
1D86  JSR $D79B     ; PARAMETER HOLEN
1D89  CPX #$04      ; >4 ?
1D8B  BCC $1D90     ; NEIN, DANN WEITER
1D8D  JMP $D248     ; SONST FEHLERMELDUNG
1D90  STX $FA
1D92  JSR $0079     ; LFD. ZEICHEN HOLEN
1D95  CMP $#2C      ; KOMMA ?
1D97  BEQ $1D9C     ; JA, DANN VERZWEIGEN
1D99  JMP $CF08     ; SONST SYNTAX ERROR
1D9C  JSR $D79B     ; NAECHSTEN PARAMETER
1D9F  STX $FB
```

Listing 2. Das vollständige Assemblerprogramm zur Befehlserweiterung

```
1DA1  BEQ $ 1DBA     ; BEI 0 TONGENERATOR ABSCHALTEN
1DA3  JSR $0079      ; NAECHSTES ZEICHEN HOLEN
1DA6  CMP #$2C       ; KOMMA ?
1DA8  BEQ $1DAD      ; JA, DANN WEITER
1DAA  JMP $CF08      ; SONST SYNTAX ERROR
1DAD  JSR $D79B      ; LETZTEN PARAMETER HOLEN
1DB0  CPX #$10       ; >16 ?
1DB2  BCC $1DB7      ; NEIN, DANN WEITER
1DB4  JMP $D248      ; SONST FEHLERMELDUNG
1DB7  STX $900E      ; LAUTSTAERKE
1DBA  LDX $FA
1DBC  LDA $FB
1DBE  STA $900A,X    ; TONHOEHE IN DEN TONGENERATOR
**********           ROUTINE ABSCHLIESSEN
1DC1  LDA $9D        ; FLAG DIREKTMODUS/PROGRAMM
1DC3  BPL $1DC8      ; FALLS PROGRAMM DANN VERZWEIGEN
1DC5  JMP $C474      ; DIREKTMODUS: READY EINSPRUNG
1DC8  JMP $0079      ; ZUR NORMALEN BEFEHLSBEARBEITUNG
**********           INITIALISIERUNG
1DCB  LDA #$4C
1DCD  STA $7C
1DCF  LDA #$00
1DD1  STA $7D
1DD3  LDA #$20
1DD5  STA $7E
1DD7  RTS
```

Listing 2. Assemblerprogramm zur Befehlserweiterung (Schluß)

```
100 REM *************************
110 REM ****                 ****
120 REM **** BASIC-AUTOSTART ****
130 REM ****                 ****
140 REM **** 1984 BY C.SAUER ****
150 REM ****                 ****
160 REM **** HUBERTUSSTR. 14 ****
170 REM ****                 ****
180 REM **** 8000 MUNCHEN 14 ****
190 REM ****                 ****
200 REM *************************
210 FORT=1TO95:READD:P=P+D:NEXT
220 IFP<>9552THENPRINT"FEHLER !!":END
230 PRINT"■■■AUTOSTART"
240 PRINT"■■■---------"
250 PRINT"■■■BITTE VERSION ANGEBEN:"
260 PRINT"■ 1= 6V"
270 PRINT" 2= +3K"
280 PRINT" 3= >8K ":F=0
290 GETA$:IFA$=""THEN290
300 FORT=1TO3
310 IFA$=CHR$(T+48)THENF=T
320 NEXT
330 IFF=0THEN290
340 FORT=1TO4-F:PRINT"□";:NEXT:PRINT"■"F
350 ONFGOTO360,370,380
360 S=17:GOTO390
370 S=5:GOTO390
380 S=19:GOTO390
390 AD=PEEK(44)
400 IFAD<>STHENPRINT"■■■■VOR DEM LADEN"
:PRINT"■POKE 44,"S"EINGEBEN !":END
410 V=(S-1)*256:RESTORE
420 PRINT"■■■ANFANGSADRESSE"
430 PRINT"■FUER IHR PROG.  "V+96"■■■■■■■
":INPUTX
440 IFX<V+96THENPRINT"■NICHT MOEGLICH !!
":GOTO420
450 X1=INT(X/256):X2=X-X1*256
460 FORI=VTOV+94
470 READD:IFD=-1THEND=S-1
480 IFD=-2THEND=X1
490 IFD=-3THEND=X2
500 POKEI,D:NEXT
510 PRINT"■FERTIG. DER POKE FUER"
520 PRINT"■805 IST"S-1". LADEN"
530 PRINT"■SIE JETZT DAS HAUPT-"
540 PRINT"■PROGRAMM NACH."
550 POKEX1*256+X2-1,0:POKE43,X2:POKE44,X
1:NEW
560 DATA169,014,141,036,003
570 DATA169,242,141,037,003
580 DATA169,194,141,020,003
590 DATA169,219,141,024,003
600 DATA169,010,141,137,002
610 DATA169,112,141,040,003
620 DATA169,-03,133,043,169
630 DATA-02,133,044,169,000
640 DATA198,043,168,145,043
650 DATA230,043,160,001,152
660 DATA145,043,032,051,197
670 DATA165,034,024,105,002
680 DATA133,045,165,035,105
690 DATA000,133,046,032,096
700 DATA198,162,005,189,083
710 DATA-01,157,119,002,202
720 DATA208,247,240,004,082
730 DATA085,078,013,169,004
740 DATA133,198,076,116,196
READY
```

Listing 3. Basicloader zum Autostart

programms nach einem RESET erschwert wird.

Zur Erklärung betrachten wir Bild 3. Es zeigt die Speicheraufteilung beim Autostart, bezogen auf die Grundversion. Das eigentliche Maschinenprogramm benötigt 95 Byte. Es liegt direkt am Basicbeginn (Adresse 4096). Dann folgt eine Lücke. Sie ist, wie bereits gesagt, nicht unbedingt nötig, aber sie erschwert etwaigen Raubkopierern das Auffinden des Programms. Hieran schließt sich das eigentliche Basicprogramm an.

So funktioniert der Autostart

Durch POKE 43,24 : POKE 44,3 wird der gesamte Bereich zwischen Adresse 792 und Programmende aufgezeichnet, wodurch sich die Ladezeit etwas erhöht.

Wie wir bereits gesehen haben, ist das Betriebssystem des VC 20 dank seiner Vektoren äußerst flexibel. Für den Programmschutz machen wir uns dabei folgende Zeiger zu Nutze:

1. NMI-Vektor, Adresse 792,793: Dieser Vektor stellt die Verbindung zwischen RESTORE-Taste und NMI-Routine her. Durch die Änderung (siehe Schritt 7) wird die RESTORE-Routine einfach übersprungen; die Taste ist quasi abgeschaltet.

2. STOP-Vektor, Adresse 808,809: Auch hier wird die bestehende Routine übersprungen.

Da dieser Vektorenbereich mit abgespeichert wird, ist, nachdem der Computer "FOUND" anzeigt, ein Stoppen des Computers nicht mehr möglich.

3. INPUT-Vektor, Adresse 804,805: Dieser Zeiger ist der eigentliche Dreh- und Angelpunkt des Autostarts. Er ist für die Tastatureingabe verantwortlich. Da er ständig durchlaufen wird, bewirkt POKE 804,0 : POKE 805,16 (bei geladener Autostartroutine) in der Grundversion einen Start des Basicprogramms. Ändert man den Zeiger hingegen vor dem Abspeichervorgang (wie in Schritt 8) geschieht vorläufig nichts, denn dann wird die Tastatur ja nicht benutzt.

Somit eignet sich dieser Vektor besonders gut für unseren Zweck. Denn solange sich der Computer mit dem Laden beschäftigt, ist die Tastatur »ruhig gestellt«. Der Zeiger wird so lange nicht benötigt, bis das Programm komplett geladen ist. Ist dies geschehen, springt das Betriebssystem über den INPUT-Vektor in die Autostartroutine, die ihrerseits (nach dem Rückstellen des Zeigers auf seinen ursprünglichen Wert) über RUN das Basicprogramm startet.

Damit auch alles wieder in den richtigen Speicherbereich geladen wird, dafür sorgt die Sekundäradresse bei SAVE " ",1,1.

Das Programm kann anschließend ganz normal mit LOAD geladen werden. Zum Schluß noch zwei Tips:

1. Wer besonders clever ist, der vernichtet alle »Spuren«, indem er die Maschinenroutine nach ihrer Benutzung im Basicprogramm löscht:
5 FOR T = (Startadresse) TO (Startadresse) + 95 : POKE T, RND(0) +255 : NEXT
(Startadresse = 4096 in der Grundversion; = 1024 bei Erweiterung von 3 KByte; = 4608 bei Erweiterung ab 8 KByte

2. Bei einer Erweiterung von 8 KByte liegt ja bekanntlich der Bildschirmspeicher im Bereich zwischen Adresse 4096 und 4607. Somit wird er ebenfalls mit abgespeichert.

Soweit die erste Folge unseres Kurses. Im zweiten Teil wollen wir uns etwas näher mit der Zeropage beschäftigen und unter anderem zeigen, wie man mehrere Basicprogramme gleichzeitig im Speicher halten kann.

(Christoph Sauer/ev)