Der gläserne VC 20 Teil 3

In der letzten Folge befaßten wir uns schwerpunktmäßig mit der Zeropage. Diesmal wird der sich daran anschließende Adreßbereich von $0100 bis $03FF unter die Lupe genommen.

Dieser Bereich ist so interessant, daß sich die Betrachtungen darüber bis in die 4. Folge erstrecken werden. Übrigens ist dieser Teil auch auf den C 64 anwendbar, denn Betriebssytem und Basic-Interpreter dieser beiden Computer sind ja nahezu identisch.

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 1a). Dies geschieht solange, bis man die RETURN-Taste betätigt.

Dieser Puffer hat eine Kapazität von 88 Zeichen — also den bekannten vier Bildschirmzeilen.

Der Weg einer Eingabezeile

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 der inzwischen hinreichend bekannten CHRGET-Routine aus dem Puffer gelesen und mit den Befehlswörtern aus dem ROM verglichen. War die Überprüfung positiv — ist der Befehl als identifiziert worden —, so verkürzt der Computer die Kommandozeile, indem er die Befehle in Token (siehe Teil 1, Tabelle 1) umwandelt. Diese Prozedur durchlaufen sowohl die Programmzeilen (mit Zeilennummer) als auch die direkten Kommandos (Bild 1b). 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 Interpreterprogramm wieder in die Warteschleife zurück, damit weitere Befehle entgegen genommen werden können.

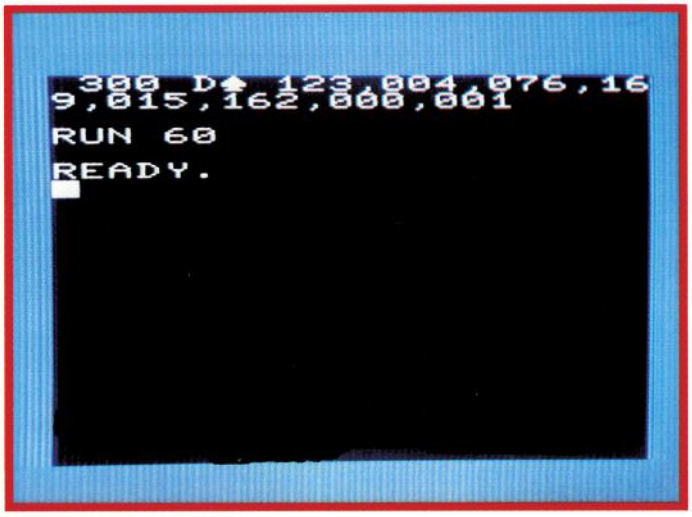

Bild 2. Anhand dieses Bildschirmfotos muß der Cursor programmiert werden

Nun möchte ich den weiteren Weg einer Direktmoduszeile beschreiben, denn der verläuft anders. Nachdem die Zeile mit Hilfe der CHRGET-Routine 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«, was sich zu dieser Zeit im Puffer befindet, würde dann überschrieben. Um das zu verhindern, sind diese Kommandos im Direktmodus verboten (Fehlermeldung »ILLEGAL DIRECT«).

Die Tastaturverwaltung

Die Schlüsselfunktionen für Basic wie beispielsweise die Umwandlung der ASCII-Zeichen in Token (wie eben beschrieben), Umwandlung der Token in Klartext (die Befehlswörter werden bei LIST wieder als ASCII-Zeichen sichtbar gemacht), werden durch indirekte Sprünge über Vektoren abgewickelt. Durch Zusatzroutinen besteht so die Möglichkeit, Klartextbefehle wie SOUND, KEY, OLD ect. in den Interpreter mit einzubinden. Dieses Verfahren besprechen wir später, zunächst aber schreiten wir in der Betrachtung des Adreßbereiches von $0277-$03FF fort (dieser ist in Tabelle 1 genauer aufgelistet).

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 (der C 64 übrigens auch) seine Tastaturoperationen ebenfalls über einen Puffer ab. Dies wird beim LISTen von Basic-Programmen deutlich, denn während der Computer ein Programm ausgibt, können sie 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 Einga-

ben wurden im Tastaturpuffer (Adresse 631-640/ $0277-$0280) zwischengespeichert.

Dieser Puffer hat eine Kapazität von maximal zehn Zeichen. Wem dies zuviel ist, der kann die Länge des Puffers durch Adresse 649 einstellen. Ratsam kann dies bei relativ langsamen Basic-Spielen sein, wo es empfehlenswert ist, nur ein Zeichen im Tastaturpuffer zwischenzuspeichern (POKE 649,1). Anderenfalls »hinkt« die Tastatur immer den Ereignissen hinterher.

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
256-318	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...
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 erste mal 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-Kleinschrift Umschaltung. 0: Normal/ 128: Sperre)
658	0292	Flag für Scrolling
659-670	0293-029E	RS 232 Register, Zeiger, ect.
671-672	029F-02A0	Zwischenspeicher für IRQ bei Bandbetrieb
673-676	02A1-02A4	Diverse VIA-Zeiger
677-767	02A5-02FF	Zwischenspeicher einer Bildschirmzeile
786-769	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 1. Die Adreßbelegung der Seiten 2 bis 4 beim VC 20

Der »DATA-Erzeuger«

Um dies verständlich zu machen, habe ich ein Programm (Listing 1) 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!

Dazu nochmals die Durchleuchtung der Funktionsweise. Während des Systeminterrupts (was dort geschieht klären wir in einer der nächsten Folgen), 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) 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 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 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 und jedoch genau überlegen, wie unser Bildschirm aussieht, denn dementsprechend muß der Cursor programmiert werden.

Cursorprogrammierung einmal anders

Bild 2 zeigt ein Bildschirmfoto dieser Situation. 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« 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 gibt an (das noch als Nachtrag) wieviele Zeichen sich momentan im Puffer befinden. Dieser POKE-Befehl darf bei der Manipulation des Tastaturspeichers nie vergessen werden. Ferner ist die Bereitschaftsmeldung »READY«, die beim Übergang in den Direktmodus ausgegeben wird, zu berücksichtigen. Man muß beachten, daß dadurch nicht die bereits auf dem Bildschirm befindlichen Zeichen überschrieben werden.

Die Bedienung des Programms an sich ist ganz einfach. Nach Eingabe der Anfangs- und Endadresse werden die DATA-Zeilen mit jeweils acht Elementen in den Programmspeicher generiert; begonnen wird mit Zeilennummer 300, die in 10er-Schritten erhöht wird. Da sich das Listing selbst kommentiert, erübrigt sich alles Weitere. Der Vollständigkeit halber ist noch zu erwähnen, daß die REM-Zeilen nicht mit eingegeben werden müssen.

```
1 REM DATA-ERZEUGER
2 REM
10 INPUT"■■STARTADRESSE ";SA
20 INPUT"■ENDADRESSE   ";EA
30 A=SA:GOSUB250:POKE248,A1:POKE249,A2
40 A=EA:GOSUB250:POKE250,A1:POKE251,A2
50 POKE252,44:POKE253,1
55 REM *** STARTZEILENNUMMER = 300
60 DEF FNZ(X)=PEEK(X)+PEEK(X+1)*256
65 REM *** DOPPELPEEK FUNKTION
70 AD=FNZ(252)
75 REM *** AD = AKTUELLE ZEILENNUMMER
80 PRINT"■"AD"D■ ";
90 FORT=0TO7
95 REM *** DATA ZEILE MIT 8 ELEMENTEN
96 REM ***          ERZEUGEN
100 B=PEEK(FNZ(248)+T)
110 B$=STR$(B):L=LEN(B$)
120 B$=RIGHT$(B$,L-1)
130 IFL=4THEN150
140 FORY=L+1TO4:B$="0"+B$:NEXT
145 REM *** JEDES ELEMENT IST 3 STELLIG
150 B$=","+B$
160 IFT=0THENB$=RIGHT$(B$,3)
170 PRINTB$;:IFFNZ(248)+T>=FNZ(250)
    THEN230
175 REM *** LFD. ADRESSE > END ADRESSE ?
180 NEXT
190 A=AD+10:GOSUB250:POKE252,A1:
    POKE253,A2
200 A=FNZ(248)+8:GOSUB250:POKE248,A1:
    POKE249,A2
210 PRINT:PRINT"■RUN 60"
213 REM *** TASTATURPUFFER MIT <HOME>,
215 REM *** <RETURN>, <CRSR DOWN> UND
217 REM ***     <REUTRN> FUELLEN
220 POKE631,19:POKE632,13:POKE633,17:
    POKE634,13:POKE198,4
230 END
250 A2=INT(A/256):A1=A-A2*256
260 RETURN

READY.
```

Listing 1. Der DATA-Erzeuger

Benutzt wurde der Tastaturpuffer bereits in einem Programm, das im ersten Teil dieser Serie abgedruckt wurde. Die Rede ist von der Autostartroutine, welche einen Basic-Programmstart (aus einem Maschinenprogramm heraus) durch Füllen des Puffers mit dem Kommando RUN (+ CHR$ (13)) realisierte.

Die Eckadressen

Als nächstes besprechen wir die Adressen 641-644/ $0281-$0284. Sie enthalten die Anfangs- beziehungsweise Endadresse des verfügbaren Speicherbereiches.

Bei dem Systemreset ($FD22/ 64802) werden verschiedene Routinen wie beispielsweise Initialisierung der Zeropage, Setzen der Vektoren, RAM-Test ect. durchlaufen. Dabei wird unter anderem auch der verfügbare Speicherplatz festgestellt und die Eckadressen den obengenannten Registern übergeben. Diese Daten sind die Grundlage für alle weiteren Grundeinstellungen des Systems, also Basic-Beginn und -Ende, Beginn des Video- und Farbspeichers ect.

Wird es nun nötig, eine Speicherkonfiguration zu simulieren, beispielsweise Grundversion bei eingesteckter 8-KByte-Erweiterung, so kann das über diese Zeiger geschehen:
POKE 642, 16: POKE 644, 30: SYS 64970: SYS 64821 bewirkt diese Simulation. Das Unterprogramm im Betriebssystem (Adresse 64970), das zur Reset-Routine gehört, hat die Aufgabe, die Seiten 0 (Zeropage), 2 und 3 zu löschen. Danach prüft der Computer seine Speicherzellen. Dieser Test hat die Aufgabe, das RAM auf seine Funktionsfähigkeit

```
1 REM FUNKTIONSTASTEN
2 REM
3 REM   (BASIC-LADER)
4 REM
100 DIM$(2)
110 S(0)=18372:S(1)=16820:S(2)=7987
120 PRINT"■■FUNKTIONSTASTEN:"
130 FORY=0TO2:PS=0
140 FORT=0TO151:READD:PS=PS+D:NEXT
150 IFPS<>S(Y)THENPRINT"■■FEHLER IN ZEIL
E":GOTO170
160 GOTO180
170 PRINT"■"Y*190+310"■-"(Y+1)*190+310:E
ND
180 NEXT
190 POKE55,56:POKE56,PEEK(56)-2:CLR
200 AD=PEEK(56):PA=AD*256+57:RESTORE
210 FORT=PATOPA+454
220 READD
230 IFD=-1THEND=AD
240 IFD=-2THEND=AD+1
250 POKET,D:NEXT
260 PRINT"■■START MIT SYS■"PA
270 PRINT"■AKTIVIERUNG MIT ■SPACE■"
280 GETA$:IFA$<>" "THEN280
290 SYSPA
300 SYS0L
310 DATA169,239,141,143,002,169,-01,141
320 DATA144,002,169,205,141,024,003,169
330 DATA-02,141,025,003,169,076,133,000
340 DATA169,090,133,001,169,-01,133,002
350 DATA096,032,121,000,201,076,208,052
360 DATA169,049,133,250,162,000,169,016
370 DATA133,251,169,013,032,210,255,169
380 DATA070,032,210,255,165,250,032,210
390 DATA255,169,032,032,210,255,189,077
400 DATA-02,032,210,255,232,198,251,208
410 DATA245,230,250,165,250,201,057,208
420 DATA213,076,115,000,032,121,000,201
430 DATA079,208,003,076,210,254,201,082
440 DATA208,006,032,057,-01,076,115,000
450 DATA201,044,240,003,076,008,207,032
460 DATA155,215,224,009,144,003,076,072
470 DATA210,202,134,250,032,121,000,201
480 DATA044,208,233,032,115,000,201,034
490 DATA208,226,138,010,010,010,010,170
500 DATA160,016,032,115,000,201,034,240
510 DATA009,157,077,-02,232,136,208,242
520 DATA240,009,169,000,157,077,-02,232
530 DATA136,208,249,076,115,000,165,157
540 DATA240,012,165,203,162,005,221,001
550 DATA-02,240,010,202,208,248,076,220
560 DATA235,039,047,055,063,197,197,240
570 DATA245,133,197,138,024,010,133,250
580 DATA173,141,002,201,001,240,002,198
590 DATA250,165,250,170,202,138,010,010
600 DATA010,010,133,251,168,185,077,-02
610 DATA240,016,201,094,240,015,201,039
620 DATA208,002,169,034,032,210,255,200
630 DATA208,235,169,000,133,207,240,009
640 DATA169,013,141,119,002,169,001,133
650 DATA198,076,214,235,076,073,083,084
660 DATA094,000,000,000,000,000,000,000
670 DATA000,000,000,000,082,085,078,094
680 DATA000,000,000,000,000,000,000,000
690 DATA000,000,000,000,073,078,080,085
700 DATA084,000,000,000,000,000,000,000
710 DATA000,000,000,000,071,079,083,085
720 DATA066,000,000,000,000,000,000,000
730 DATA000,000,000,000,083,089,083,048
740 DATA000,000,000,000,000,000,000,000
750 DATA000,000,000,000,082,069,084,085
760 DATA082,078,000,000,000,000,000,000
770 DATA000,000,000,000,082,069,083,084
780 DATA079,082,069,000,000,000,000,000
790 DATA000,000,000,000,076,079,065,068
800 DATA039,000,000,000,000,000,000,000
810 DATA000,000,000,000,120,072,138,072
820 DATA152,072,173,029,145,016,037,045
830 DATA030,145,170,041,002,240,026,044
840 DATA017,145,032,052,247,032,225,255
850 DATA208,018,032,082,253,032,249,253
860 DATA032,024,229,032,057,-01,108,002
870 DATA192,076,222,254,076,086,255,000

READY.
```

Listing 2. Funktionstastenbelegung (Basic-Lader)

```
>> FUNKTIONSTASTEN <<

****************** INITIALISIERUNG
1C39  LDA #$EF
1C3B  STA $028F
1C3E  LDA #$1C    ; AENDERUNG DES
1C40  STA $0290   ; TASTATUR-VEKTORS
1C43  LDA #$CD
1C45  STA $0318
1C48  LDA #$1D
1C4A  STA $0319   ; NEUER NMI VEKTOR
1C4D  LDA #$4C
1C4F  STA $00
1C51  LDA #$5A
1C53  STA $01
1C55  LDA #$1C    ; SPRUNGZEIGER
1C57  STA $02     ; FUER 'SYS0'
1C59  RTS
****************** BEFEHLSAUSWERTUNG
1C5A  JSR $0079   ; CHRGOT, LFD. ZEICHEN
1C5D  CMP #$4C    ; 'SYS0 L' ?
1C5F  BNE $1C95   ; NEIN, DANN WEITER
1C61  LDA #$31    ; SONST F1-F8 AUSLISTEN
1C63  STA $FA     ; ZAEHLER VORBEREITEN
1C65  LDX #$00
1C67  LDA #$10
1C69  STA $FB
1C6B  LDA #$0D    ; ZEILENVORSCHUB
1C6D  JSR $FFD2   ; AUSGEBEN
1C70  LDA #$46    ; 'F'
1C72  JSR $FFD2   ; AUSGEBEN
1C75  LDA $FA     ; LFD. NUMMER
1C77  JSR $FFD2   ; AUSGEBEN
1C7A  LDA #$20    ; ' '
1C7C  JSR $FFD2   ; AUSGEBEN
1C7F  LDA $1D4D,X; BEFEHLSSTRING
1C82  JSR $FFD2   ; AUSGEBEN
1C85  INX
1C86  DEC $FB
1C88  BNE $1C7F
1C8A  INC $FA
1C8C  LDA $FA
1C8E  CMP #$39    ; LETZTER STRING ?
1C90  BNE $1C67   ; NEIN, DANN WEITER
1C92  JMP $0073   ; SONST INS BASIC
1C95  JSR $0079   ; LFD. BEFEHL HOLEN
1C98  CMP #$4F    ; 'O' FUER 'SYS0 O' ?
1C9A  BNE $1C9F   ; NEIN, DANN WEITER
1C9C  JMP $FED2   ; SONST WARMSTART
1C9F  CMP #$52    ; 'R' FUER 'SYS0 R' ?
1CA1  BNE $1CA9   ; NEIN, DANN WEITER
1CA3  JSR $1C39   ; SONST PROGRAMMRESTART
1CA6  JMP $0073   ; UND ZURUECK ZU BASIC
1CA9  CMP #$2C    ; ',' FUER AENDERUNG ?
1CAB  BEQ $1CB0   ; JA, DANN WEITER
1CAD  JMP $CF08   ; SONST 'SYNTAX ERROR'
1CB0  JSR $D79B   ; TASTENNR. INS X-REG.
1CB3  CPX #$09    ; >8 ?
1CB5  BCC $1CBA   ; NEIN, DANN WEITER
1CB7  JMP $D248   ; SONST FEHLERMELDUNG
1CBA  DEX
1CBB  STX $FA
1CBD  JSR $0079   ; CHRGOT, LFD. BEFEHL
1CC0  CMP #$2C    ; ',' KOMMA ?
1CC2  BNE $1CAD   ; NEIN, DANN FEHLER
1CC4  JSR $0073   ; NAECHSTES ZEICHEN ?
1CC7  CMP #$22    ; '"' HOCHKOMMA ?
1CC9  BNE $1CAD   ; NEIN, DANN FEHLER
1CCB  TXA
1CCC  ASL
1CCD  ASL
1CCE  ASL
1CCF  ASL
1CD0  TAX         ; X-REG. * 16
1CD1  LDY #$10    ; BEFEHLSSTRING HOLEN
1CD3  JSR $0073   ; NAECHSTES ZEICHEN
1CD6  CMP #$22    ; '"' ENDE DES STRINGS
1CD8  BEQ $1CE3   ; REST MIT 0 FUELLEN
1CDA  STA $1D4D,X; SONST STRING AB-
1CDD  INX         ; SPEICHERN
1CDE  DEY
1CDF  BNE $1CD3   ; WEITER
1CE1  BEQ $1CEC   ; ENDE: JMP $0073
1CE3  LDA #$00    ; REST MIT 0 FUELLEN
1CE5  STA $1D4D,
1CE8  INX
1CE9  DEY
1CEA  BNE $1CE5
1CEC  JMP $0073   ; ZURUECK INS BASIC
****************** FUNKTIONSTASTEN ABF.
1CEF  LDA $9D     ; RUN ODER DIREKTMODUS
1CF1  BEQ $1CFF   ; BEI RUN KEINE ABFR.
1CF3  LDA $CB     ; GEDRUECKTE TASTE
1CF5  LDX #$05
1CF7  CMP $1D01,X; MIT TASTATURCODE DER
1CFA  BEQ $1D06   ; FUNKTIONSTAS. VERGL.
1CFC  DEX
1CFD  BNE $1CF7   ; WEITER,TASTE ERKANNT
1CFF  JMP $EBDC   ; TEST NEGATIV
1D02  .BYTE $27   ; < F1 >
1D03  .BYTE $2F   ; < F3 >
1D04  .BTYE $37   ; < F5 >
1D05  .BYTE $3F   ; < F7 >
1D06  CMP $C5     ; ENTPRELLUNG
1D08  BEQ $1CFF
1D0A  STA $C5
1D0C  TXA
1D0D  CLC
1D0E  ASL
1D0F  STA $FA
1D11  LDA $028D   ; KONTROLLTASTE
1D14  CMP #$01    ; <SHIFT> GEDRUECKT ?
1D16  BEQ $1D1A   ; JA, DANN WEITER
1D18  DEC $FA     ; WERT UM 1 ERNIEDR.
1D1A  LDA $FA
1D1C  TAX
1D1D  DEX
1D1E  TXA
1D1F  ASL
1D20  ASL
1D21  ASL
1D22  ASL
1D23  STA $FB     ; ACCU * 16
1D25  TAY
1D26  LDA $1D4D,Y; BEFEHLSSTRING LADEN
1D29  BEQ $1D3B   ; ENDE ?
1D2B  CMP #$5E    ; '↑' DIREKT AUSFUEHREN
1D2D  BEQ $1D3E   ; JA, DANN VERZWEIGEN
1D2F  CMP #$27    ; >'< HOCHKOMMAERSATZ?
1D31  BNE $1D35   ; NEIN, DANN WEITER
1D33  LDA #$22    ; HOCHKOMMA LADEN
1D35  JSR $FFD2   ; UND AUSGEBEN
1D38  INY
1D39  BNE $1D26
1D3B  LDA #$00    ; CURSOR ANSCHALTEN
1D3D  STA $CF
1D3F  BEQ $1D4A   ; UNBEDINGTER SPRUNG
1D41  LDA #$0D    ; <RETURN>
1D43  STA $0277   ; IN DEN PUFFER
1D46  LDA #$01    ; EIN ZEICHEN IM
1D48  STA $C6     ; TASTATURPUFFER
1D4A  JMP $EBD6   ; ZURUECK ZUM URSPRUNG
****************** BEFEHLSSPEICHER
1D4D  [illegible]
1D5D  [illegible]
1D6D  [illegible]
1D7D  [illegible]
1D8D  [illegible]
1D9D  [illegible]
1DAD  [illegible]
1DBD  [illegible]
****************** NEUE NMI ROUTINE
1DCD  SEI         ; KOPIE DER ALTEN
1DCE  PHA         ; ROUTINE
1DCF  TXA
1DD0  PHA
1DD1  TYA
1DD2  PHA
1DD3  LDA $911D
1DD6  BPL $1DFD
1DD8  AND $911E
1DDB  TAX
1DDC  AND #$02    ; RS 232 AKTIV ?
1DDE  BEQ $1DFA   ; JA, DANN VERZWEIGEN
1DE0  BIT $9111
1DE3  JSR $F734   ; STOPTASTE ABFRAGEN
1DE6  JSR $FFE1   ; STOPTASTE GEDRUECKT ?
1DE9  BNE $1DFD   ; NEIN, DANN RTI
1DEA  JSR $FD52   ; VEKTOREN SETZEN
1DEE  JSR $FDF9   ; I/O REGISTER SETZEN
1DF1  JSR $E518   ; CLR SCREEN
1DF4  JSR $1C39   ; REGISTER AENDERN
1DF7  JMP ($C002); BASIC WARMSTART
1DFA  JMP $FEDE   ; NMI FUER RS 232
1DFD  JMP $FF56   ; RTI
```

Listing 3. Funktionstastenbelegung (Assembler-Darstellung)

hin zu überprüfen, damit es nicht zu Fehlfunktionen durch einen beschädigten Speicher kommt. Bei dieser Gelegenheit wird die Ausbaustufe des Speichers festgestellt; das Ergebnis findet sich dann in den Registern für die Anfangs- beziehungsweise Endadresse. Hier steigen wir nun mit den entsprechend manipulierten Registern (642 auf 16 und 644 auf 30) in die ROM-Routine ein. Mit Hilfe dieser Werte richtet das Unterprogramm nun den Video- und Farbspeicher ein. Mit dem zweiten SYS-Befehl (SYS 64821) wird die Initialisierung fortgesetzt. Dabei richtet er den Speicher für Basic ein und gibt die Kaltstartmeldung

```
**** CBM BASIC V2 ****
3583 BYTES FREE
```

aus.

Soweit ein kleiner Einblick ins Betriebssystem, eine ausführlichere Erläuterung erfolgt in einer der nächsten Folgen.

Die Speicherorganisation

Jetzt möchte ich noch einen kleinen Einblick in die Speicherorganisation geben, was auch im Hinblick auf Grafik von Bedeutung ist. Wie hinreichend bekannt sein dürfte, gibt es drei verschiedene Speichererweiterungen zu kaufen, nämlich 3 KByte, 8 KByte und 16 KByte. Die Sammelerweiterungen (also 27/32/64 KByte-Erweiterungen sollen hier gedanklich ebenfalls in diese drei verschiedenen Module zerlegt werden.

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 3 soll erläutert werden, warum eine Verschiebung notwendig ist.

Der **V**ideo-**I**nterface-**C**hip VIC (daher auch der Englische Name des VC 20), der vor allem für die Erzeugung des Fernsehsignals und den Aufbau des Bildschirmes 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 sowie bei einer 8-KByte-Erweiterung, währe 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.

Programme sollten immer auf allen Erweiterungsversionen lauffähig sein. Wer also in Routinen mit dem Bildschirm- oder Farbspeicher arbeitet, kann sich mit Hilfe der Register 36866 und 36869 im VIC die

momentanen Adressen beschaffen. Bildschirm: 4*(PEEK(36866)AND128) + 64*(PEEK(36869)AND120) Farbspeicher: 4*(PEEK(36866)AND128)+37888

Gewußt wo — Die Bildschirmadressen

Mit Hilfe bestimmter Bits aus diesen Registern bildet der VIC die Adressen, die er benötigt, um — unabhängig vom Prozessor — Bild- und Farbspeicherstellen auszulesen, damit er mit diesen Informationen das Fernsehbild erzeugen kann.

Das Betriebssystem hingegen bezieht seine Informationen über die Lage des Videospeichers nicht aus diesen VIC-internen Registern, sondern aus Adresse 648. Gibt man beispielsweise »POKE 648,28« ein, so liegt der Bildschirmspeicher zwischen 7168 und 7679. In Wirklickkeit stellt der Video-Interface-Chip — der, wie gesagt, unabhängig arbeitet — weiterhin den Speicherausschnitt zwischen Adresse 7680 und 8191 auf dem Bildschirm dar. Der Cursor schreibt also in einem ganz anderen Speicherabschnitt. Erst durch einen Warmstart (durch die Tastenkombination RUN/ STOP — RESTORE) werden die VIC-Register angepaßt.

Vom Bildschirm nun wieder zur Tastatur. In unseren systematischen Betrachtungen der Seite 1 bis 3 im Speicher, kommen wir nun zu den Adressen $0280-$0291/649-656, die der Tastatur zugeordnet sind. Sie enthalten lediglich Parameter für die Arbeit mit der Tastatur wie zum Beispiel Repeat Flag, das Flag für Kontrolltasten ect. Näheres entnehmen sie bitte Tabelle 1.

Praktisches: Die Befehlseingabe über Funktionstasten

Zwei weithin unbekannte Adressen ($028F, $0290/ 655,656) sind vorzüglich dazu geeignet, eine Funktionstastenabfrage auf Interruptbasis zu realisieren. Besagte Adressen bilden einen Vektor für die Tastaturdecodierung, der meines Erachtens nur für den Zweck der Funktionstastenabfrage geschaffen wurde.

Was sind überhaupt Vektoren? ROM, so sagt ja bereits der Name (**R**ead **O**nly **M**emory), ist grundsätzlich nicht überschreibbar. Wer dennoch das System ergänzen will (beispielsweise durch neue Basic-Befehle), ist gezwungen, das gesamte ROM auszutauschen, es sei denn, die Schöpfer des Computers haben mögliche Optionen — so wie beim VC 20 (oder C 64) — bereits eingeplant. Dies geschieht, indem aus dem ROM heraus ins RAM verzweigt wird.

Normalerweise steht an der entsprechenden RAM-Adresse nur ein Zeiger auf eine ROM-Adresse, eben ein Vektor. Durch Änderung eines solchen Vektors kann man elegant bestehende Routinen umgehen und sie durch eigene ersetzen beziehungsweise ergänzen. Soweit, so gut.

Wie im Handbuch zu lesen, gibt es Unterschiede zwischen den Bildschirmcodes eines Zeichens (»A« beispielsweise hat den Wert 1) und dem allgemein verbreiteten ASCII-Code (»A« hat hier den Wert 65). Gleiches gilt für die Tastatur. Hier unterscheidet man ebenfalls zwischen dem ASCII- und dem sogenannten Tastatur-Matrixcode. Diese VC 20-interne Codierung wird durch eine Betriebssystemroutine — die über einen Vektor verfügt (den oben erwähnten für die Tastaturdecodierung) — in ASCII-Zeichen umgewandelt. Das Maschinenprogramm in Listing 2 und 3 wird durch »verbiegen« des Vektors 655/656 in die Tastaturroutine mit eingebaut. Es fragt die Codes der Funktionstasten ab und druckt die Zeichen aus, mit denen sie belegt wurden.

Das recht komfortable Programm liegt als Basic-Lader in Listing 2 vor. Nach dem Starten mit RUN wird das

```
1 REM BEFEHLSKOPF
2 REM
100 DIMS(5)
110 REM *** PRUEFSUMMEN
120 S(0)=10538:S(1)=0:S(2)=10372
130 S(3)=14363:S(4)=12024:S(5)=733
140 FORY=0TO5:S=0
150 FORT=0TO103:READD:S=S+D:NEXT
160 IF S<>S(Y)THENPRINT"■■■■FEHLER IN T
EIL"Y+1:END
170 NEXT
180 RESTORE
190 PRINT"■■■■M■ODULBEREICH ODER"
200 PRINT"■■■E■NDE DES SPEICHERS ?"
210 GET A$:IFA$=""THEN210
220 IFA$="M"THENPRINT"■■■■M":GOTO380
230 IFA$<>"E"THEN210
240 PRINT"■■■■■■■E"
245 REM *** PROGRAMM ANS SPEICHERENDE
250 POKE55,0:POKE56,PEEK(56)-3:CLR
260 AD=PEEK(56)
270 HD=AD*256:N=623
275 REM *** LESESCHLEIFE
280 FORT=HDTOHD+N
290 READD
300 IFD=-1THEND=AD
310 IFD=-2THEND=AD+1
320 IFD=-3THEND=AD+2
330 POKET,D:NEXT
340 PRINT"■■■START DES PROGRAMMS"
350 IFFL=0THENPRINT"■MIT SYS"HD"."
360 IFFL=1THENPRINT"■MIT RESET (=SYS 648
02)"
370 END
375 REM *** PROGRAMM IN DEN MODULBEREICH
376 REM *** MIT IDENTIFIKATION A0CBM
380 V$="009160199254065048195194205"
390 FORE=0TO8
400 X=VAL(MID$(V$,E*3+1,3)):READD
410 POKE40960+E,X:NEXT
420 HD=40969:N=614:AD=160:FL=1:GOTO 280
430 REM TEIL 1
440 DATA120,234,234,234,234,234,234,234
450 DATA234,032,141,253,032,082,253,032
460 DATA249,253,032,024,229,162,011,189
470 DATA059,-01,157,000,003,202,016,247
480 DATA088,234,234,234,032,164,227,165
490 DATA043,164,044,032,008,196,169,071
500 DATA160,-01,032,030,203,032,018,228
510 DATA076,129,227,058,196,131,196,229
520 DATA-01,152,-02,226,-02,134,206,042
530 DATA042,042,042,032,054,052,039,069
540 DATA082,032,032,066,065,083,073,067
550 DATA032,042,042,042,042,013,000,000
560 DATA000,000,000,000,000,000,000,000
570 REM TEIL 2
580 DATA000,000,000,000,000,000,000,000
590 DATA000,000,000,000,000,000,000,000
600 DATA000,000,000,000,000,000,000,000
610 DATA000,000,000,000,000,000,000,000
620 DATA000,000,000,000,000,000,000,000
630 DATA000,000,000,000,000,000,000,000
640 DATA000,000,000,000,000,000,000,000
650 DATA000,000,000,000,000,000,000,000
660 DATA000,000,000,000,000,000,000,000
670 DATA000,000,000,000,000,000,000,000
680 DATA000,000,000,000,000,000,000,000
690 DATA000,000,000,000,000,000,000,000
700 DATA000,000,000,000,000,000,000,000
710 REM TEIL 3
720 DATA000,000,000,000,000,000,000,000
730 DATA000,000,000,000,000,000,000,000
740 DATA000,000,000,000,000,000,166,122,160
750 DATA004,132,015,189,000,002,016,007
760 DATA201,255,240,062,232,208,244,201
770 DATA032,240,055,133,008,201,034,240
780 DATA086,036,015,112,045,201,063,208
790 DATA004,169,153,208,037,201,048,144
800 DATA004,201,060,144,029,132,113,160
810 DATA000,132,011,136,134,122,202,200
820 DATA232,189,000,002,056,249,158,192
830 DATA240,245,201,128,208,048,005,011
840 DATA164,113,232,200,153,251,001,185
850 REM TEIL 4
860 DATA251,001,240,089,056,233,058,240
870 DATA004,201,073,208,002,133,015,056
880 DATA233,085,208,159,133,008,189,000
890 DATA002,240,223,197,008,240,219,200
900 DATA153,251,001,232,208,240,166,122
910 DATA230,011,200,185,157,192,016,250
920 DATA185,158,192,208,180,160,255,202
930 DATA200,232,189,000,002,056,249,105
940 DATA-03,240,245,201,128,208,002,240
950 DATA173,166,122,230,011,200,185,104
960 DATA-03,016,250,185,105,-03,208,226
970 DATA189,000,002,016,155,076,009,198
980 DATA016,066,201,255,240,062,036,015
990 REM TEIL 5
1000 DATA048,058,170,132,073,201,204,176
1010 DATA010,160,192,132,035,160,158,132
1020 DATA034,208,011,233,076,170,160,-03
1030 DATA132,035,160,105,132,034,160,000
1040 DATA010,240,016,202,016,012,230,034
1050 DATA208,002,230,035,177,034,016,246
1060 DATA048,241,200,177,034,048,008,032
1070 DATA071,203,208,246,076,243,198,076
1080 DATA239,198,032,115,000,201,204,144
1090 DATA025,201,252,176,021,032,243,-02
1100 DATA076,174,199,233,203,010,168,185
1110 DATA009,-03,072,185,008,-03,072,076
1120 DATA115,000,032,121,000,076,231,199
1130 REM TEIL 6
1140 DATA052,253,000,000,000,000,000,000
1150 DATA000,000,000,000,000,000,000,000
1160 DATA000,000,000,000,000,000,000,000
1170 DATA000,000,000,000,000,000,000,000
1180 DATA000,000,000,000,000,000,000,000
1190 DATA000,000,000,000,000,000,000,000
1200 DATA000,000,000,000,000,000,000,000
1210 DATA000,000,000,000,000,000,000,000
1220 DATA000,000,000,000,000,000,000,000
1230 DATA000,000,000,000,000,000,000,000
1240 DATA000,000,000,000,000,000,000,000
1250 DATA000,000,000,000,000,000,000,000
1260 DATA000,075,073,076,204,000,000,000

READY.
```

Listing 4. Befehlskopf zur Definition eigener Basic-Befehle (Basic-Lader)

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

Bild 1a. Die Basic-Befehle im Eingabepuffer vor der Umwandlung in Interpretercode

49	48	48	45	49	50	48	0	0	0
1	0	0	—	1	2	0			

153	194	40	52	51	41	58	155	49	48	48	45	49	50	48	0	0	0
		(	4	3	)	:		1	0	0	—	1	2	0			

PRINT PEEK LIST

Bild 1b. Die Kommandos im Puffer nach der Übersetzung in Token

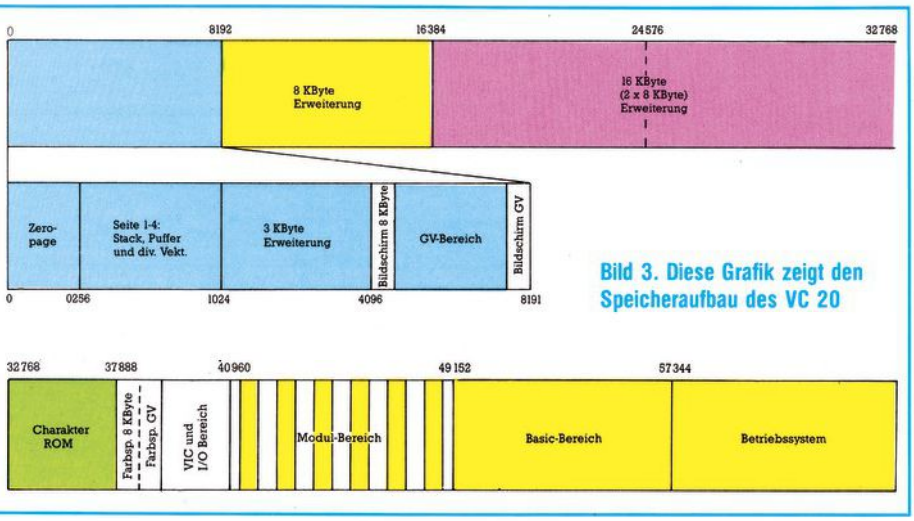

Bild 3. Diese Grafik zeigt den Speicheraufbau des VC 20

F1: LIST	— LIST-Befehl mit CHR$(13)
F2: RUN	— wie bei LIST
F3: INPUT	
F4: GOSUB	
F5: SYSO	
F6: RETURN	
F7: RE-STORE	
F8: LOAD'	— ' entspricht dem Hochkomma (")

Tabelle 2. Die Grundbelegung der Funktionstasten nach dem Laden des Maschinenprogramms

Maschinenprogramm automatisch ans Ende des verfügbaren Speichers geladen. Mit der SPACE-Taste aktiviert man diese Routine. Als erstes werden die Befehle gelistet, mit denen die Funktionstasten belegt sind. (Tabelle 2).

Auffällig ist bei den Kommandos LIST und RUN der Pfeil nach oben (↑). Er bewirkt ein sofortiges Ausführen des Befehls — entspricht also »LIST« und RETURN-Taste beziehungsweise »RUN« und RETURN-Taste. Das andere auffällige Zeichen ist der Apostroph (') bei dem Kommando LOAD, der dem Hochkomma (") entspricht. Das echte Gänsefüßchen ist bereits für die Syntax des Änderungsbefehls (siehe unten) vergeben.

Auf den Befehl π wie er beispielsweise beim Programm »Basic-Switch« (Folge 2) verwendet wurde, habe ich dieses Mal aus Platzgründen verzichtet, damit Lader und Maschinenprogramm in der Grundversion Platz finden. Die Kommandos werden statt dessen über den Befehl »SYS0« (oder F5) eingegeben. Dabei machen wir uns den USR-Vektor (Adresse 0-2) als Sprungzeiger zu Nutze. Damit zur Syntax bei der Funktionstastenprogrammierung:

SYS0 L — Listen der Funktionstastenbelegungen

SYS0 O—(Off) schaltet die Funktionstasten ab.

SYS0 R — (Restart) schaltet die Funktionstasten wieder ein.

SYS0,X, »befehl« — belegt die X-te Funktionstaste mit einem Befehl.

Zum Schluß kommen wir noch zu einem Thema, mit dem ich mich mehr an den fortgeschrittenen Maschinensprachenprogrammierer wenden möchte. Im ersten Teil dieser Serie wurde beschrieben, wie der Basic-Befehlssatz mit Hilfe der CHRGET-Routine und des Befehles »π« im beschränktem Umfang erweitert werden kann. Diese Methode hat allerdings viele Unzulässigkeiten; es sind beispielsweise keine verkürzten Befehle wie beim normalen Basic (LIST = L SHIFT I) möglich.

Nun soll beschrieben werden, wie der Basic-Befehlssatz um richtige Klartextkommandos erweitert werden kann. Auch hierfür müssen wir bestehende Interpreterroutinen, die über Vektoren angesprungen werden, umgehen beziehungsweise ergänzen.

Für das Verarbeiten von Basic-Programmen sind drei Schlüsselroutinen zu substituieren. Da ist zunächst das Unterprogramm »ASCII in Token wandeln«. ($C57C) welches — wie der Name bereits sagt — die Aufgabe hat, Eingabezeilen im Basic-Puffer (wie zu Anfang beschrieben) in Interpretercode zu wandeln. Das Gegenstück dazu ist die Unterroutine »Interpretercode in Klartext wandeln.« Will man Basic-Zeilen sichtbar machen, so benutzt man das Kommando LIST. Dazu wird eben diese ROM-Routine benötigt, die die Rückumwandlung der Token in ASCII-Zeichen vornimmt.

Sämtliche Befehle sind in Form von ASCII-Zeichen im Basic-ROM enthalten, lediglich zum letzten Buchstaben jedes Befehlswortes wurde 128 ($80) addiert. Läßt man sich die Befehle durch

Listing 5. Der Tokenerzeuger

```
1 REM TOKENERZEUGER
2 REM
100 PRINT"■■■TOKENERZEUGER"
110 INPUT"■■STARTADRESSE";SA
115 REM *** PROGRAMMUEBERPRUEFUNG
120 FORT=1TO3:READD:IFPEEK(SA+8+T)<>DTHE
NPRINT"■■■PROGRAMM NICHT GELADEN":END
130 PRINT"■■1■ BEGINN BEI 204 (KILL  WI
RD UEBERSCRIEBEN)
140 PRINT"■■2■ ODER BEI TOKEN 205 ?"
150 GETA$:IFA$=""THEN150
160 IFA$="1"THENF=617:GOTO190
170 IFA$<>"2"THEN150
180 F=621
190 INPUT"■■WIEVIELE TOKEN";AT
200 IFAT>47THENPRINT"■■NICHT MOEGLICH":G
OTO190
205 REM *** BEFEHLSWORT ABSPEICHERN
210 Z=SA+F:W=SA+520
220 FORT=1TOAT
230 PRINT"■TOKEN "T+204", ADRESSE"
240 INPUTA$,B:B=B-1
250 FORY=1TOLEN(A$)
260 X=ASC(MID$(A$,Y,1)):IFMID$(A$,Y+1,1)
=""THENX=X+128
270 POKEZ,X:Z=Z+1:NEXT
280 B2=INT(B/256):B1=B-B2*256
290 POKEW,B1:POKEW+1,B2:W=W+2
300 NEXT
310 DATA32,141,253

READY.
```

Adresse	Funktionsbeschreibung	Bemerkung	Übergaberegister
C613	Startadresse einer Programmzeile berechnen		Eingabe: Zeile in $14,15 Ausgabe: Adresse in $5F, 60
C807	Prüft auf Doppelpunkt	Zur Syntaxkontrolle	
CD8A	Ausdruck holen (z.B. Zeichen)	$D7FD wandelt Fließk. in Int.	
CAA0	PRINT-Befehl: Ausdruck oder String holen und ausdrucken		
CEF7	Prüft auf Klammer zu »)«	Syntaxkontrolle	
CEFA	Prüft auf Klammer auf »(«	Syntaxkontrolle	
CEFD	Prüft auf Komma	Syntaxkontrolle	
CF04	»SYNTAX ERROR« ausgeben		
D113	Prüft auf Buchstabe (A-Z)		
D248	»ILLEGAL QUANTITY« ausgeben		
D79B	Byte-Wert (0-255) holen	Bei der Argumentabfrage	

Tabelle 3. Einige der wichtigsten Routinen, mit denen man für Basic-Befehle Argumente, Satzzeichen etc. abfragen kann.

FOR T= 49310 TO 47565: PRINT CHR$(PEEK(T)):NEXT
ausdrucken, so sehen die Kommandos im Groß-/Kleinschrift-Modus folgendermaßen aus:
END = enD
FOR = foR ect.

Ferner ist jedem Basic-Befehl eine Adresse zugeordnet, bei der eine Abarbeitung vorgenommen wird. Diese Adressen sind in einer Tabelle (von $C00C $C07F) zusammengefaßt. Eine spezielle Routine hat wiederum die Aufgabe, die Befehlsadresse für ein entsprechendes Token aus der Tabelle zu lesen und einen Sprung nach dorthin durchzuführen. Auch dieses Unterprogramm kann man mittels eines Vektors umgehen.

Damit haben wir das nötige Rüstzeug, um selbst Basic-Kommandos in den Interpreter mit aufzunehmen. In Listing 4 ist ein dafür geeignetes Programm abgedruckt, welches ich jetzt näher erläutern möchte. Es ist der Kopf für ein Utility, in das nach Belieben Befehlswörter und Sprungadressen eingesetzt werden können.

Die Routine gliedert sich in vier Teile: Der erste Teil (von $2000 — $203A) ist eine Kopie der Reset-Routine. Dabei wird der Computer neu initialisiert und eine neue Kaltstartmeldung
**** 64'ER BASIC ****
ausgedruckt. Hier ist genügend Platz vorgesehen, damit der Text nach eigenen Wünschen gestaltet werden kann. Ferner wurden die ersten neun Bytes mit NOPs versehen, damit dort — falls das Programm im Modulbereich abgelegt wird — die obligate Autostartinformation (a0CBM) eingesetzt werden kann. Anderenfalls — wenn das Maschinenprogramm per SYS gestartet wird — muß der erste Mnemonic-Befehl ein SEI sein.

Die sich jetzt anschließenden Programmteile sind die oben erwähnten Ergänzungen zu den Interpreterroutinen. Dies sind teilweise Kopien aus den alten ROM-Routinen mit Ergänzungen für die neuen Basic-Kommandos. Die Befehlswörter müssen jetzt nur noch eingetragen werden. Das geschieht folgendermaßen:

Die Befehle werden mit Hilfe des Programms »Tokenerzeuger« (Listing 5) in den entsprechenden Adreßbereich geschrieben. Bei nachträglichen Eintragungen ist zu beachten, daß zum letzten Buchstaben jedes Befehlswortes der Wert $80 (=128) zu addieren ist. Die Ergänzungsbefehle beginnen mit Token 204. Das erste Befehlswort beginnt mit dieser Nummer, der zweite erhält automatisch die 205 und so weiter.

Weiterhin ist für jedes Kommando die Sprungadresse in der Tabelle ($2208-$2268) zu vermerken. Dabei muß die Reihenfolge Low-/High Byte beachtet werden. Außerdem muß das LOW-Byte vor dem Eintrag um eines dekrementiert werden (LOW Byte -1). Auch diese Arbeit erledigt das Programm in Listing 5.

Ein Befehl wurde bereits eingetragen. Der Befehl KILL führt einen Reset durch und damit ist das Programm abgeschaltet. Wer also seine Maschinenprogramme ins Basic einbinden möchte, kann dies mit der beschriebenen Methode tun. Beispielsweise könnte man die in Listing 2 abgedruckte Funktionstastenroutine mit dem KEY-Befehl belegen. Auch im Hinblick auf Grafik, Tonerzeugung oder Joystickabfrage bietet sich hier die Möglichkeit, die Unzulänglichkeiten des Basics zu überwinden. Auch die Besitzer eines C 64 können die beschriebene Routine benutzen, da die Basic-ROMs nahezu identisch sind (sie haben lediglich eine andere Adresse). Außerdem habe ich in Tabelle 3 eine Liste der wichtigsten Unterprogramme zusammengestellt, mit denen man unter anderem Parameter, Strings, Kommas oder ähnliches abfragen kann. Auf jeden Fall sollte man sich für diese Arbeit ein ROM-Listing (zum Beispiel »VC 20 Intern« von Data Becker) zulegen.

Damit möchte ich für heute schließen. Das nächste Mal werden wir sehen, was man mit den Kernal-Vektoren anfangen kann und betrachten die Grafikmöglichkeiten beim VC 20.

(Christoph Sauer/ev)