Also werden soviele Vergleiche benötigt, wie die Anzahl der vorhandenen Stingvariablen im Quadrat, von der Anzahl der Speicherverschiebungen mal abgesehen, denn sie entspricht der Anzahl der Stringvariablen.

Auf einen Nenner gebracht bedeutet das, daß die FRE(0) Routine aus dem DIMA$(9000) Beispiel 9001x9001 = 81 Millionen Vergleiche benötigt!

Diese Zahl stimmt nicht ganz mit der Wirklichkeit überein, da auch andere Faktoren eine Rolle spielen, die Größenordnungen sind aber ziemlich richtig.

Die Zeit der Garbage Collection hängt aber nicht von der Anzahl der Müllstrings ab, da diese ja gar nicht überprüft werden, sondern nur alle Descriptoren.

Es gibt mehrere Möglichkeiten, die Garbage Collection kurz zu halten:

— Benutzen Sie so wenig Strings wie möglich, dann braucht die Garbage Collection weniger Zeit, außerdem muß sie nicht so oft aufgerufen werden, da ja mehr Platz für Müllstrings ist.

— Strings wenn möglich im Programm definieren, und dann nie mehr verändern, dann entsteht weniger Stringmüll

— Weitere Stringmüll verursachende Befehlssequenzen meiden, wie zum Beispiel A$=A$ und ähnliches.

— Sollen Strings vertauscht werden, SWAP-Routine verwenden.

Halt, werden Sie jetzt sagen, was ist denn eine SWAP-Routine?

Um einen der größten Stringmüll-Verursacher, das Vertauschen von Strings, zu entschärfen, habe ich diese Routine geschrieben. Sie vertauscht zwei Strings, ohne daß Stringmüll entsteht.

Sie werden sich wahrscheinlich schon gefragt haben, warum eigentlich beim Vertauschen von Strings nicht einfach die Descriptoren der beiden Stringvariablen ausgetauscht werden. Dann könnte kein Stringmüll entstehen.

Genau das tut die SWAP-Routine. Eine ähnliche ist vielleicht schon in einer Basic-Erweiterung enthalten, die Sie benützen. Wenn nicht, so tippen Sie einfach Listing 1 ab. SWAP ist im Speicher frei verschiebbar, Sie können es also an jede beliebige Adresse legen. Sinnvoll wäre wohl der Kassettenpuffer (Adresse 828) oder der Bereich ab $C000. Das hängt ganz von weiteren Programmen oder Erweiterungen ab, die sich gleichzeitig im Speicher befinden.

Wie Sie die SWAP-Routine anwenden, steht in den REM-Zeilen des Listings, die Arbeitsweise verdeutlichen die Bilder 9 und 10.

Wenn Sie das Thema Stringverarbeitung und neue Stringfunktionen in Maschinensprache mehr interessiert: Darauf werde ich in der nächsten Ausgabe genauer eingehen, ebenso wie auf die genaue Funktionsweise der SWAP-Routine.

(Boris Schneider/gk)

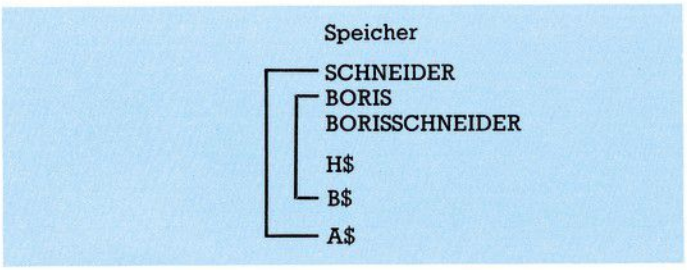


Bild 8. Die Garbage Collection ist beendet, der Restmüll am Ende des Stringspeichers kann einfach überschrieben werden.

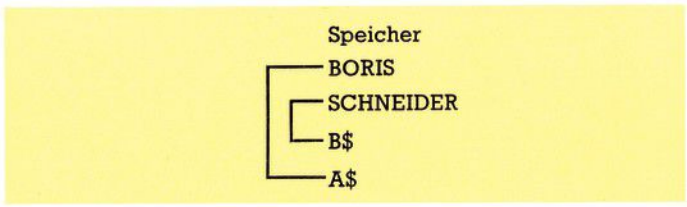


Bild 9. Und noch einmal vertauschen. Vor der Ausführung der SWAP-Routine

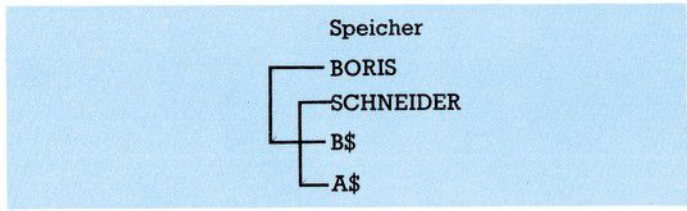


Bild 10. Nach Ausführung der SWAP-Routine.

MEMO MAP

Die Landkarte, der wir auf unserer Reise durch den Speicher folgen, weist diesmal — neben vielen interessanten Plätzen — auch zwei Orte auf, die nützlich sind.

Zum einen ist es die Adresse 19, zum anderen gelangen wir in den Bereich ab Speicherzelle 43, der für Speicher-Manipulationen so eminent und wichtig ist.

Adresse 18 ($12)

1. Flagge für Vorzeichen bei SIN, COS und TAN

2. Flagge bei Vergleich

Zuerst kommt das Vorzeichen der trigonometrischen Funktionen an die Reihe.

Die Routinen des Basic-Übersetzers (Interpreter), welche die drei trigonometrischen Funktionen SIN, COS und TAN berechnen, verwenden die Speicherzelle 18 zur Bestimmung des Vorzeichens.

Zur Erinnerung: Die trigonometrischen Funktionen haben in den vier »Quadranten« des Kreises (0-90, 90-180, 180-270, 270-360 Grad) nicht unbedingt dieselben Vorzeichen. Die Vorzeichen ändern sich allerdings nur an den Grenzen der Quadranten, wie in Bild 1 zu sehen ist. Die Flagge in Zelle 18 gibt das Vorzeichen nicht direkt an, sondern auf Umwegen. Die Darstellung ist in der folgenden Tabelle 1 zusammengefaßt.

WINKEL	0-90	90-180	180-270	270-360
SIN	gleich	Wechsel	Wechsel	gleich
COS	255	255	0	0
TAN	0	255	255	0

Dabei bedeutet »gleich«: 0-0-0-0 oder 255-255-255
»Wechsel«: 0-255-0-255

Da die Erklärung mit »gleich« beziehungsweise »Wechsel« nicht gerade einleuchtend ist, schlage ich vor, daß Sie sich das Ganze mit dem folgenden kleinen Programm selbst anschauen, welches für viele Werte des Winkels im Bogenmaß — und in kleinen Schritten — den Wert der Flagge, daneben den Winkel I und den Wert der Funktion mit Vorzeichen ausdruckt.

```
10 FOR I=0 TO 10 STEP 0.01
20 PRINT PEEK(18);INT(I*100)/
100;SIN(I):NEXT
```

Diese etwas umständliche Art, den Wert von I auszudrucken, vermeidet Rundungsfehler und begrenzt den Ausdruck auf zwei Dezimalstellen. Wenn Sie die Winkelwerte von I in Graden ausgedruckt haben wollen, können Sie eine andere Zeile 20 verwenden, welche die Umrechnungsformel vom Bogenmaß in Grade verwendet:

```
Winkel in Grad = Winkel im Bo-
genmaß * 180/π
20 Print PEEK(18);INT(I*180/π)
;SIN(I):NEXT
```

Statt SIN können Sie genauso gut COS und TAN einsetzen.

In Bild 1 sind nicht nur die Kurven und die Bereiche der Vorzeichen, sondern auch die Winkelbereiche sowohl im Bogenmaß, als auch in Graden dargestellt.

Die Speicherzelle 18 wird auch noch von anderen Routinen des Basic-Interpreters beansprucht und zwar von allen, die einen Vergleich wie <−>,)= und so weiter durchführen. Entsprechend der Art des Vergleichs steht dann in der Zelle 18 eine Ziffer von 0 bis 6.

Das folgende Programm macht das deutlich.

```
10 A=2
20 FOR I=1 TO 3
30 IF I= A THEN PRINT I;
PEEK(18);"="
40 IF I< >A THEN PRINT I;
PEEK(18);"> <"
50 IF I> A THEN PRINT I;
PEEK(18);">"
60 IF I< A THEN PRINT I;
PEEK(18);"<"
70 IF I> =A THEN PRINT I;
PEEK(18);"> ="
80 IF I< =A THEN PRINT I;
PEEK(18);"< ="
```

RY mit Wandervorschlägen Teil 3

```
90 IF I<A OR I=A THEN
PRINT I;PEEK(18);" < OR = "
100 NEXT I
```

Kurz zur Erklärung dieser Zeilen: In der FOR..NEXT-Schleife wird die Variable I mit der Konstanten A=2 verglichen. In den Zeilen 30 bis 90 werden alle möglichen Vergleichsoperatoren durchgeprüft. Jeder der zutrifft, druckt den Wert von I, den Wert der dann in Zelle 18 stehenden Flagge und schließlich den Vergleichsoperator aus. Aus dem Resultat dieses Programms läßt sich folgende Tabelle zusammenstellen:

Vergleich	Flagge in 18
< OR =	0
> OR =	0
>	1
=	2
>=	3
<	4
<>	5
<=	6

Sie sehen, die Flagge für die kombinierten Vergleichsoperatoren entspricht der Summe ihrer Einzelwerte. Nur die Verknüpfung über OR nicht, denn die ergibt 0.

Adresse 19 ($13)
Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes

Immer dann, wenn von Basic Daten ein- oder ausgegeben werden, schaut die entsprechende Routine des Übersetzers in Zelle 19 nach, um welches Peripheriegerät es sich handelt. Zur Debatte stehen Tastatur, Datasette, RS232/User-Port, Bildschirm, Drucker und Floppy-Laufwerk.

Die Flagge ihrerseits ist ausschlaggebend für die feinen Unterschiede, wie zum Beispiel das Fragezeichen, bei Eingabe von der Tastatur (INPUT) oder die Anweisung »Press Play on Tape« bei Eingabe von der Datasette.

Beim Einschalten des Rechners setzt die Initialisierungsroutine des Betriebssystems, die beim VC 20 ab Adresse 58276 ($E3A4), beim C 64 ab 58303 ($E3BF) beginnt, die Flagge in Zelle 19 auf 0. Die Null bedeutet Eingabe über Tastatur und Ausgabe über Bildschirm.

Wenn Sie einen Dissassembler haben, drucken Sie doch einmal das Assemblerlisting aus. Sie werden in Adresse 58324/58325 ($E3D4/E3D5), beim C 64 in 58354/58355 ($E3F2/E3F3) den Befehl finden, der eine Null nach Zelle 19 ($13) bringt.

Immer dann, wenn ein Programm nicht Tastatur und Bildschirm, sondern eines der oben genannten anderen Peripheriegeräte anspricht (indem mit OPEN.... eine Datei = Logical File eröffnet wird), wird in Zelle 19 die Nummer der gerade bearbeiteten Datei eingetragen, mit den bereits beschriebenen Konsequenzen.

Ich will hier nicht weiter darauf eingehen, da wir den Inhalt von Zelle 19 selbst nicht auslesen können. Er wird nämlich immer gleich wieder auf Null gesetzt.

Wir können ihn aber durch POKE verändern. Durch POKE 19,1 gaukeln wir dem Rechner vor, daß Ein- und Ausgabe über »externe« Geräte läuft, selbst wenn nur die Tastatur und der Bildschirm betrieben werden.

Wenn zum Beispiel der Rechner der Meinung ist, daß ein INPUT von der Datasette kommt, druckt er kein Fragezeichen aus, auch kein EXTRA IGNORED als Fehlermeldung bei zu zahlreicher Eingabe und das alleinige Drücken der RETURN-Taste ignoriert er auch, im Gegensatz zum »normalen« INPUT. Probieren Sie es aus:

```
10 INPUT "TEST";A$
20 PRINT A$
```

In diesem Normalfall erscheint nach RUN darunter die Aufforderung TEST?

Eine Eingabe, zum Beispiel XX, erscheint mit einem Abstand daneben, und nach RETURN wird XX an den Anfang der nächsten Zeile gedruckt. Alle falschen Eingaben werden mit den üblichen Fehlermeldungen quittiert.
Jetzt fügen wir ein:

```
5 POKE 19,1
```

Nach RUN erscheint wieder die Aufforderung TEST, aber ohne Fragezeichen. Die Eingabe XX wird ohne Abstand daneben gesetzt und nach RETURN mit einem Abstand in derselben Zeile weitergeschrieben.

Das Drücken der RETURN-Taste setzt den Cursor nicht wie üblich in die nächste Zeile, sondern schiebt ihn in derselben Zeile weiter.

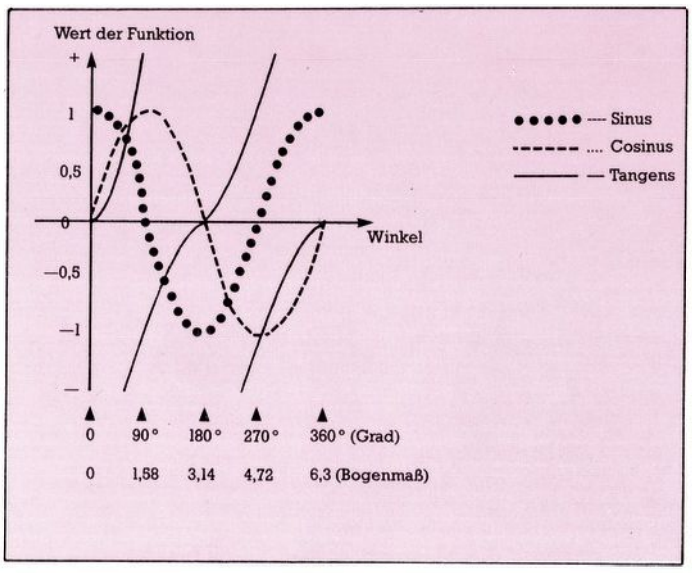
Diesen zusätzlichen Effekt muß man beachten, da er sehr störend für den Verlauf eines Programms sein kann.

Man kann ihn natürlich auch nutzbringend einsetzen, hat er doch die Eigenschaft eines automatischen »Cursor UP«. Eine pfiffige Anwendung dieser Art wurde von Brad Templeton für den PET erfunden und ist von Jim Butterfield für eine MERGE-Routine mit dem Namen »Magic Merge« veröffentlicht worden.

Da diese Routine aber primär auf der Eigenschaft der Speicherzelle 153 basiert, werde ich sie dann erläutern, sobald wir bei der Zelle 153 angelangt sind.

Zurück zur Flagge in Zelle 19.

Umgekehrt können wir POKE 19,0 leider nicht nutzen, da die betroffenen Befehle GET, GET#, INPUT, INPUT# und PRINT# die Flagge sofort auf den richtigen Wert setzen. Nur PRINT und LIST tun das nicht, wie wir bei dem PRINT-Befehl oben ja gesehen haben.



Adresse 20–21 ($14–$15)
Zeilennummer für LIST, GOTO, GOSUB und ON, Zeiger der Adresse bei PEEK, POKE, SYS und WAIT

In diesen Speicherzellen wird die Zeilennummer der Sprungbefehle GOTO, ON..GOTO und GOSUB sowie die Zeilenangabe beim LIST-Befehl gespeichert. Da die Werte bis maximal 65535 gehen können, braucht der Computer 2 Bytes zur High/Low-Byte-Darstellung.

Die GOTO-Routine (im VC 20 ab 51360 = $C8A0, im C 64 ab 43168 = $A8A0) vergleicht die Zahl in 20/21 mit der laufenden Zeilenzahl. Wenn sie kleiner ist, wird ab der ersten Zeile des Programms gesucht. Ist sie aber größer, dann beginnt die Suche ab der laufenden Zeilenzahl. Die Suche geht so lange, bis die in 20/21 angegebene Zeilenzahl gefunden ist. Dann fährt das Programm mit dieser Zeile fort.

LIST speichert in 20/21 die höchste auszulistende Zeilennummer ab, falls keine Angabe beim LISTen gegeben worden ist, den Wert 65535 ($FFFF).

Die Befehle PEEK, POKE, SYS und WAIT verwenden diese Speicherzellen zur Angabe der Adressen, die dem Befehl immer folgen müssen.

Leider können wir die Speicherzellen 20/21 mit Basic-Programmen nicht bearbeiten; ihr Inhalt wird immer gleich auf 20 zurückgesetzt.

Adresse 22 ($16)
Zeiger auf den nächsten freien Speicherplatz im »Temporary String Descriptor Stack«

Dieser Zeiger bezieht sich in seiner Wirkung auf die übernächsten Speicherzellen 25 bis 33 ($19-$21).

Diese werden als Stapelspeicher (Stack) für Angaben über vorläufige Zeichenketten — auf englisch »Temporary String Descriptor« — verwendet (siehe unten).

Die Speicherzelle 22 ($16) ihrerseits enthält einen Zeiger auf den jeweils nächsten verfügbaren Platz in diesem Speicher ab Zelle 25. Da er eine Kapazität von 3 mal 3 Byte hat, zeigt der Zeiger auf die Zelle 25 ($19), wenn er leer ist. Bei einem Eintrag zeigt er auf 28 ($1C), bei zwei Einträgen auf 31 ($1F) und schließlich auf 34 ($22), wenn der Speicher voll ist.

Eine Zeichenkette ist dann »vorläufig«, wenn sie noch nicht einer Stringvariablen zugeordnet worden ist, zum Beispiel

MEMORY MAP mit Wandervorschlägen Teil 3

»Mahlzeit« in dem Basic-Befehl PRINT "MAHLZEIT.

Beim Einschalten setzt das Betriebssystem mit der Einschalt-Routine ab Adresse 58303 ($E3BF) im C 64, beim VC 20 ab 58276 ($E3A4) den Zeiger auf 25. Die Stringverwaltungsroutine ab 46215 ($B487) im C 64 beziehungsweise ab 54407 ($D487) im VC 20 fragt bei String-Eingaben die Flagge ab. Nach jeder Eintragung in den Speicher ab Zelle 25 wird der Zeiger um 3 weitergesetzt.

Sie können die Leerflagge 25 mit PRINT PEEK (22) leicht nachprüfen.

Die anderen Eintragungen können nicht nachgeprüft werden, weil sie sofort auf 25 zurückgesetzt werden.

Wir können sie aber durch POKE beeinflussen; ob das sinnvoll ist, ist eine andere Frage.

```
10 POKE 22,34
20 PRINT "MAHLZEIT"
```

Die Zahl 34 in Zelle 22 sagt dem Programm, daß der Speicher ab Zelle 25 voll ist und wir bekommen statt der MAHLZEIT eine Fehlermeldung serviert.

Mit einem POKE-Befehl, der als Argument die für den vorgesehenen Zweck ungültige Zahl 35 verwendet:

```
POKE 22,35
```

erreichen wir allerdings zwei interessante »Dreckeffekte«. Zum einen unterdrückt der Befehl die Ausgabe des READY, zum anderen aber bewirkt er, daß bei LIST ein Listing ohne Zeilennummern ausgedruckt wird, sowohl auf dem Bildschirm als auch mit dem Drucker.

Das billigste editierfähige Textverarbeitungssystem

Die Idee dazu habe ich von Mike Apsey's Hinweis in »Commodore User« Juli 1984. Mit Zeilennummern versehen, läßt sich jeder beliebige Text schreiben, verbessern, verschieben, abspeichern, aber nicht RUNen!

Der POKE-Befehl von oben (POKE 22,35) gefolgt von einem CMD und LIST, druckt dann alles brav als reinen Text aus. Die maximale Zeilenlänge entspricht der Zeilenlänge des jeweiligen Computers.

Probieren Sie es aus:

```
10 DER COMPUTER BIETET
IN DER
20 DATENFERNÜBERTRA-
GUNG
30 UNGEAHNTE MÖGLICH-
KEITEN.
40 ABER DIE GEFAHR
50 USW. USW.
60:
```

Jede Zeile wird mit der RETURN-Taste abgeschlossen. Damit auch alles gedruckt wird, muß — zumindest bei meinem Drucker (1526) — eine »Leerzeile« folgen (Zeile 60). Mit

```
POKE 22,35:OPEN 1,4:CMD
1:LIST
```

wird der Text ohne Zeilennummern ausgedruckt. Sie können ihn vorher nach Belieben verändern.

Wie gesagt, nur nicht mit RUN starten, denn das bringt unweigerlich eine Fehlermeldung.

Adresse 23-24 ($17-$18)
Zeiger auf die Adresse der letzten Zeichenkette im »Temporary String Stack«

Der Inhalt dieser zwei Bytes zeigt auf den zuletzt benützten Speicherplatz innerhalb der Adresse 22 bis 33. Das heißt, daß der Wert in 23 ($17) immer um 3 kleiner ist als der in 22 ($16), während der Wert in 24 ($18) eine Null ist.

Adresse 25-33 ($19-$21)
Stapelspeicher für Angaben über vorläufige Zeichenketten

Das ist also der Speicherbereich, von dem in den beiden vorigen Abschnitten dauernd die Rede war. Ich gebe zu, »Descriptor Stack for Temporary Strings« drückt die Sache präziser aus als der deutsche Text.

Die Bedeutung eines »vorläufigen« Strings habe ich oben in der Beschreibung der Speicherzelle 22 erklärt.

Was ein Stapelspeicher (Stack) ist, entnehmen Sie bitte dem Texteinschub. Jeder der drei Byte langen Angaben im Stack von 22 bis 33 enthält die Länge sowie die Anfangs- und Endadressen eines vorläufigen Strings, ausgedruckt als Verschiebung im Basic-Speicherbereich.

Adresse 34-37 ($22-$25)
Verschiedene Zwischenspeicher

Diese vier Speicherzellen werden vom Basic-Übersetzer (Interpreter) für verschiedene Zwischenergebnisse und Flaggen benützt, die aber dem Programmierer nichts nützen.

Adresse 38-42 ($26-$2A)
Arbeitsspeicher für arithmetische Operationen

Diese Speicherzellen werden von den Basic-Routinen bei der Multiplikation und Division als »Notizblatt« verwendet. Auch die Routinen, welche die erforderliche Speichergröße beim Definieren eines Zahlenfeldes (Array) ausrechnen, benutzen diesen Bereich.

Adresse 43-44 ($2B-$2C)
Zeiger auf den Anfang der Basic-Programme im Speicher

Dieser Zeiger, in der Low/High-Byte-Darstellung, gibt dem Basic-Übersetzer an, ab welcher Speicherzelle das Basic-Programm beginnt. Normalerweise ist diese Adresse fest vorgegeben. Beim C 64 zum Beispiel zeigt der Zeiger auf 2049 ($801). Beim VC 20 ist die Lage schon schwieriger, denn der Speicherbeginn hängt davon ab, welche Speichererweiterung eingesetzt ist. Die folgende Tabelle gibt darüber Auskunft.

Beginn des Programmspeichers

C 64	2049 ($801)
VC 20 (GV)	4097 ($1001)
VC 20 (+3 K)	1025 ($401)
VC 20 (+ 8 K)	4609 ($1201)

Mit dem Befehl PRINT PEEK (43) + PEEK (44)*256 läßt sich der jeweilige Beginn des Programmspeichers leicht feststellen. Mit einem POKE-Befehl kann der Programmierer diese Anfangsadresse verändern. Wozu das gut ist, fragen Sie?

Anwendung #1:

Nun, wenn Sie zum Beispiel ein Maschinenprogramm mit einem Basic-Programm gemeinsam betreiben wollen, brauchen Sie einen Speicherbereich für das Maschinenprogramm, der vom Basic-Programm nicht belegt wird. Wir sprechen vom »Schützen des Maschinenprogramms vor dem Überschreiben durch das Basic«. Der Speicherbereich eines Maschinenprogramms ist immer bekannt. Nach seinem letzten Speicherplatz kann das Basic-Programm beginnen.

Die Verschiebung der Anfangsadresse erfolgt in vier Schritten:

1. Schritt: In den Speicherplatz vor dem neuen Basic-Bereich muß eine Null gePOKEt werden. Die Null dient zum Abgrenzen.
2. Schritt: Die Adresse der ersten Speicherzelle wird in die Low-High-Byte-Darstellung umgerechnet. Ich verweise dazu auf die Erklärung dieses Vorgangs in der ersten Folge dieses Kurses in Ausgabe 11/84.
3. Schritt: Das Low-Byte wird in die Speicherzelle 43, das High-Byte in die Zelle 44 gePOKEt.
4. Schritt: Die Operation muß unbedingt mit dem Befehl NEW abgeschlossen werden, um sicherzustellen, daß der neue Basic-Bereich auch »sauber« ist.

Im folgenden kleinen Programm wird angenommen, daß der Speicher bis zur Adresse 5000 ($1388) durch ein Maschinenprogramm belegt ist. Das Basic-Programm kann daher ab 5002 ($138A) anfangen, denn in 5001 muß ja eine Null stehen. Die Adresse 5002 teilt sich auf in ein High-Byte von INT (5002/256) = 19 und ein Low-Byte von 5002—(19*256) = 138.

```
10 POKE 5001,0
20 POKE 43,138
30 POKE 44,19
40 NEW
```

Der Effekt einer solchen »Verbiegung« des Zeigers in 43/44 wird im Texteinschub 2 »Der sichtbare Basic-Speicher« demonstriert.

Neben der oben erwähnten Anwendung der Zeigerverbiegung gibt es noch andere Möglichkeiten:

Anwendung #2:

Christoph Sauer hat in seinem Kurs »Der gläserne VC 20« in Ausgabe 10/84 auf Seite 158 gezeigt, wie man mehrere Programme gleichzeitig im Speicher unterbringen und zwischen ihnen umschalten kann.

Anwendung #3:

Man kann zwei oder mehrere unabhängige Programme genau hintereinander in den Speicher bringen, um sie aneinanderzuhängen , was dem im Commodore-Basic fehlenden Befehl MERGE entspricht. Dabei dürfen die Zeilennummern sich allerdings nicht überschneiden.

Anwendung #4:

Durch Hinaufschieben des Basic-Bereichs kann Platz geschaffen werden für selbstdefinierte Zeichen oder hochauflösende Grafik (siehe dazu auch den Grafik-Kurs).

Adresse 45-46 ($2D-$2E)
Zeiger auf die Anfangsadresse des Speicherbereichs für Variable

Mit dieser Adresse werden wir das nächste Mal fortfahren.

(Dr. Helmuth Hauck/aa)

Was ist ein Stapelspeicher (Stack)?

Texteinschub #1

Der normale Arbeitsspeicher des Computers, auf englisch »Random Access Memory« oder kurz RAM genannt, hat für jede Speicherzelle eine eigene Adresse, die beim Schreiben in den Speicher oder beim Lesen aus dem Speicher angegeben werden muß.

Als Analogie möge eine Aktenablage dienen, bei der jeder Akt (Brief, Papier, Zeichnung) in einen Ordner kommt, mit Nummer versehen.

Um einen Akt herauszuholen, muß man die Nummer (Adresse) kennen, unter der er abgelegt ist.

Ein Stapelspeicher, auf englisch »Stack« genannt, funktioniert wie eine Aktenablage, bei der jeder Akt einfach oben auf einen Stapel gelegt wird, daher der Name. Diese Ablage erfolgt ohne Kennzeichnung oder Nummer, einfach immer der Reihe nach.

Einen Akt kann man aus einem Stapelspeicher nicht beliebig herausholen, da immer nur der oberste Akt zugänglich ist.

Die Methode der Stapelspeicher bietet sich überall dort an, wo es auf die Reihenfolge der gespeicherten Daten ankommt. Basic merkt sich zum Beispiel der Reihe nach die Adressen, von denen aus mit GOSUB ein Unterprogramm angesprungen wird. Wenn mehrere GOSUBs hintereinander eingesetzt werden, liegt auf dem Stapel immer die letzte Absprungadresse bereit zum Rücksprung.

Ein Stapelspeicher hat demnach nur eine einzige Adresse, die sowohl zum Abspeichern als auch zum Auslesen dieselbe ist.

Voraussetzung eines Stapelspeichers ist natürlich eine Routine, welche alle gespeicherten Daten im Stapelspeicher um einen Platz weiterschiebt, wenn eine neue Information »oben auf den Stapel gelegt wird«.

Das Basic der Commodore-Rechner verwendet mehrere dieser Stapelspeicher.

Die Programmiersprache Forth ist völlig auf dem Prinzip des Stapelspeichers aufgebaut.

Der sichtbare Basic-Speicher

Wenn wir den Variablen A die Adresse des Speicherbeginns der Basic-Programme zuordnen und dann mit einer FOR..NEXT-Schleife den Inhalt dieser und der nächsten 100 Speicherplätze ausdrucken, sehen wir in dezimaler Darstellung die ersten 101 Zahlenwerte, mit denen der Computer ein Basic-Programm abspeichert.

Ein Verbiegen des Zeigers in Speicherzelle 43/44 kann auf diese Weise in seiner Wirkung sichtbar gemacht werden.

Als Demo-Programm wähle ich zwei Zeilen, welche die Zahlen 1 bis 9 und die Buchstaben A bis I ausdrucken.

```
 10 PRINT "123456789"
 20 PRINT "ABCDEFGHI"
100 A=2049 : REM*C 64
          4097 : REM*VC 20 ohne Erweiterung
          1025 : REM*VC 20 mit 3 KByte
          4609 : REM*VC 20 mit 8 KByte oder mehr
110 PRINT CHR$(147)
```

Zeile 100 definiert den Speicheranfang. Zeile 110 löscht den Bildschirm.

```
120 FOR J=A TO A+100
130 PRINT PEEK (J);
140 NEXT J
```

Die Befehle in den Zeilen 120 bis 140 drucken den Inhalt der ersten 101 Zellen dieses Basic-Programms aus. SAVEn Sie bitte dieses kleine Programm, denn wir brauchen es noch einmal. Dann geht es los mit RUN. In Bild 2 ist der Bildschirm-Ausdruck des VC 20 mit 8 KByte dargestellt, der des C 64 zeigt praktisch dieselbe Information.

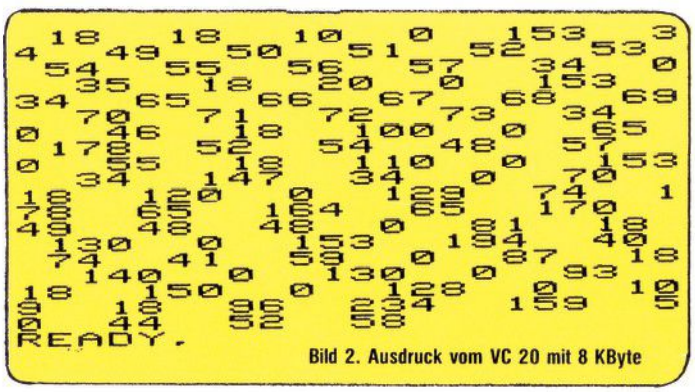


Bild 2. Ausdruck vom VC 20 mit 8 KByte

Überspringen Sie bitte zunächst die ersten beiden Zahlen. Die dritte und vierte Zahl ist 10 und 0. Das ist (als Low- und High-Byte) die Nummer der ersten Zeile des Basic-Programms. Dann folgt 153, das ist der interne Codewert für PRINT. Diese Codes für alle Basic-Befehlswörter heißen »TOKEN« und sind zusammen mit den ASCII-Codes aller Zahlen, Zeichen und Funktionen in den Commodore-Handbüchern angegeben.

Die nächste Zahl im Bildschirm ist die 34, sie ist der ASCII-Code für den Gänsefuß. Danach folgen in aufsteigender Reihenfolge die ASCII-Codes der Ziffern 1 (48) bis 9 (57). Danach sehen Sie wieder den Gänsefuß (34). Schließlich kommt eine Null als Abstandszeichen zur nächsten Basic-Zeile.

Machen Sie bitte folgendes Experiment: Ausgehend von der Adresse der ersten auf dem Bildschirm ausgedruckten Speicherzellen — zum Beispiel 4609 beim VC 20 mit 8 KByte — zählen Sie die Zellen weiter bis zur Abgrenzungs-Null. In meinem Beispiel steht die 0 in Zeile 4625. Das heißt, daß die nächste Basic-Zeile in 4626 anfängt. Und das ist genau die Zahl, die in den ersten beiden Zellen steht, die wir vorhin übersprungen haben; in meinem Beispiel steht da 18 18. Machen wir die Probe: 18 + 256 x 18 = 4626.

Jede Basic-Zeile im Speicher beginnt also mit der Adresse der nächsten Zeile (sie heißt Koppeladresse) und endet mit einer Null. Ab 4626 folgt dann die nächste Koppeladresse, danach mit 20 0 die Zeilennummer, und Sie erkennen jetzt sicher die Codes der Angaben von Zeile 20 wieder.

So, jetzt wollen wir den Zeiger in 43/44 verbiegen. Ich schlage vor, daß wir den Basic-Beginn um zehn Adressen höher schieben wollen. Sie müssen jetzt die in Zeile 100 oben verwendete Zahl für A in die High/Low-Byte-Darstellung umrechnen und das Low-Byte um 10 erhöhen. Dieses Zahlenpaar POKEn wir in die Zellen 43/44. Vorher müssen wir aber noch in die Zelle (A+10)—1 eine Abstands-Null POKEn.

Wir geben diese Befehlssequenz im Direktmodus ein:

☐ für den C 64:

```
POKE 2058,0:POKE 43,11:POKE 44,8:NEW
```

☐ für den VC 20 (GV):

```
POKE 5006,0:POKE 43,143:POKE 44,19:NEW
```

☐ für den VC 20 (= 3 KByte):

```
POKE 1034,0:POKE 43,11:POKE 44,4:NEW
```

☐ für den VC 20 (> 8 KByte)

```
POKE 4618,0:POKE 43,11:POKE 44,18: NEW
```

Jetzt ist der Anfang des Basic-Speichers versetzt. Um das zu prüfen, geben wir das kleine Programm von oben nochmal ein und lassen es mit RUN laufen. Der resultierende Bidlschirmausdruck ist in Bild 3 dargestellt.

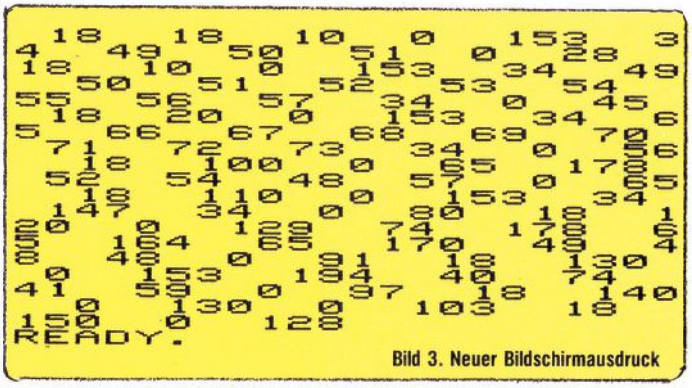


Bild 3. Neuer Bildschirmausdruck

Die ersten Zahlen sind genauso wie vorher. Es sind auch die Reste von vorher, da wir den Speicher nicht auf Null gesetzt haben. Aber zählen Sie bitte die ersten zehn Adressen hoch. Da finden Sie unser Programm von vorhin genau wieder, beginnend mit der Abstandsnull. Aber Vorsicht, lassen Sie sich nicht verwirren, denn die Koppeladressen sind natürlich jetzt auch jeweils um 10 höher. Aber hinter den Koppeladressen finden wir wieder unser Programm, in gleicher Weise dargestellt wie beim ersten Mal. Da der Zeiger in 43/44 von allen entsprechenden Routinen des Übersetzers und des Betriebssystems abgefragt wird, läuft ein verschobenes Programm fehlerfrei, solange natürlich der Zeiger nicht wieder verändert wird.