Von Basic zu Assembler Teil 1

Vom vertrauten Basic zum Programmieren in 6502-Assembler ist es gar nicht so weit. Wir werden langsam anfangen. Zuerst sind die Schleifen dran. Wir werden dabei Basic mit Maschinensprache vergleichen.

Man kann wohl getrost davon ausgehen, daß ein großer Teil der Commodore 64-Benutzer Basic beherrscht. Vermutlich hat es Sie aber auch schon öfters gereizt, die langsamen Programme durch Assembler-Routinen zu beschleunigen. Der Assembler-Kurs »Assembler ist keine Alchimie« hat die Grundlagen dieser Sprache zwar vermittelt, aber bis zum eigenen Maschinenprogramm ist es häufig noch ein holpriger Weg. Ein Führer, der Ihnen den Pfad aus den weiten Basic-Ebenen auf die Gipfel der Assembler-Programmierung zeigt, ist dieser Kurs.

Zunächst halten wir uns an Bekanntes: Basic-Befehle sollen auf ihre Entsprechungen in Assembler untersucht werden. Oft aber streifen wir die Fesseln der Hochsprache ab und lernen allerlei Assembler-Techniken kennen. Wo es uns angemessen erscheint, nehmen wir uns die Freiheit, die computerinterne Firmware zu benutzen. Alles in allem sollen Sie durch Training zum vertrautem Umgang mit Assembler gelangen.

1. Basic contra Assembler

Basic macht uns den Umgang mit unserem Computer relativ leicht: Wir brauchen uns kaum um die Verwaltung von Variablen zu kümmern. Benötigen wir zum Beispiel eine neue Stringvariable, dann genügt ihr erstes Auftauchen und Basic richtet sie für uns gebrauchsfertig ein. Sehr entgegenkommend ist Basic auch, was den Verkehr mit Peripherie jeder Art angeht: Ein Zeichen von der Tastatur anzunehmen, ist mit GET oder INPUT recht einfach. Irgend etwas auf dem Drucker zu schreiben oder auf die Diskette: mit OPEN und dem PRINT# geht's reibungslos und ohne Gedanken an die Busverwaltung. Die Hochsprache nimmt uns vieles ab, worum wir uns in Assembler kümmern müssen.

Andererseits befindet man sich etwa in der Situation eines Bauherrn, der sich ein Haus mit Fertigteilen aufstellen lassen möchte: Jeder Wunsch, der vom Standard abweicht, ist nicht möglich oder wird sehr teuer. Basic-Erweiterungen sind dann mit einer großen Sammlung aller möglichen Fertigteil-Formen zu vergleichen: Man muß sie alle kaufen, obwohl man nur einige für den Sonderwunsch braucht.

Der Vergleich fängt nun etwas zu hinken an: Die Fertigteile setzen sich nämlich bei genauerem Hinsehen aus kleinen Einzelteilen (Ziegelsteinen) zusammen. Wenn man also auf die vorgefertigten Großteile weitgehend verzichtet, dafür aber den Umgang mit Ziegelsteinen (Assemblerbefehlen) beherrscht, kann man sich genau das individuelle Haus bauen, wie bizarr es auch immer aussehen mag.

Wenn man sich das Verarbeiten eines Basic-Programmes etwas genauer ansieht, dann versteht man die Schwerfälligkeit, mit der vieles geschieht. Vom Augenblick des Einschaltens an läuft ein Maschinenprogramm im Computer: Der Interpreter. Jeder Tastendruck, jeder Basic-Befehl, der diesem in die Hände fällt, wird untersucht und führt zur Abarbeitung eines auf diesen Befehl zugeschnittenen Assemblerprogrammes.

Ein Basic-Befehl wie beispielsweise PRINT kann ganz verschiedene Reaktionen erfordern: Wenn vorher ein CMD-Kommando erfolgt war, findet die Ausgabe nicht auf dem Bildschirm statt, wenn das nicht der Fall war, kann der Bildschirm an einer anderen Stelle als im Normalfall liegen. Das, was auszudrucken ist, kann ein String sein, eine Integerzahl oder eine Fließkommavariable, allerlei Fehlerquellen sind abzufangen etc. Das zu PRINT gehörende Assembler-Programm muß all diese Möglichkeiten berücksichtigen, wie selten sie auch angesprochen werden.

Bei einem individuellen Assembler-Programm wissen wir dagegen, was wir wie ausgeben wollen. Unser eigener »PRINT«-Befehl wird nur das enthalten, was unbedingt notwendig ist, der ganze unnötige Ballast wird von uns nicht programmiert. Allerdings sind wir dann auch voll verantwortlich für die einwandfreie Funktion. Wir müssen beispielsweise dafür sorgen, daß bei Zugriffen auf Register oder Speicher dort der richtige Wert zum richtigen Zeitpunkt im erlaubten Format zur Verfügung steht. Das ist wie im täglichen Leben: Die größere Freiheit legt uns mehr Verantwortung auf und schafft uns andererseits ungeahnte Möglichkeiten, Ideen zu verwirklichen.

2. Noch eine technische Vorbemerkung

Für alle Beispielprogramme werde ich den Assembler Hypra-Ass von Gerd Möllmann verwenden. Er wurde im 64'er, Ausgabe 7/85, Seite 66, als Listing des Monats abgedruckt und ist auch auf der Leserservice-Diskette erhältlich. In dieser Preislage (6.50 Mark und etwas Schweiß beim Eintippen, beziehungsweise 29,80 Mark für die Diskette) habe ich noch keinen besseren gesehen und auch fast alle kostspieligeren Assembler reichen ihm nicht das Wasser. Weil man aber auch einen Monitor braucht, findet zu diesem Zweck weiterhin der SMON hier seinen Platz. Ich verwende eine Version, die bei $9000 beginnt, um ab $C000 Raum zu lassen für den Reassembler zu Hypra-Ass (64'er, Ausgabe 11/85). Das ist ein nützliches Programm, mit dem man Maschinencode aus dem Speicher wieder in einen Quelltext umwandeln kann. Mittels des Hypra-Ass ist dieser Quelltext dann bearbeitbar. Mit diesem kompletten Instrumentarium sind wir allen Aufgaben gewachsen.

3. Einfache Schleifen

Eine der meistgebrauchten Strukturen in Basic und auch eine der wichtigsten in Assembler ist die Programmschleife. Als »einfache« Schleife bezeichne ich solche, die zum Zählen nur 1 Byte erfordern, also maximal 256 Durchläufe erlauben. Die verschiedenen Möglichkeiten sehen wir uns anhand von Verzögerungsschleifen an, die zunächst einmal nichts anderes tun, als zu zählen und Zeit zu verbrauchen (welch ein Luxus!). Die einfachste Variante lautet in Basic etwa:

```
10 FOR I = 0 TO 255
20 NEXT I
```

In Bild 1 finden Sie die »Übersetzung« in Assembler.

Sie können sowohl das Y-Register (Variante 1) als auch das X-Register (Variante 2) zum Zählen verwenden. In Zeile 5 finden Sie .LI 1,4. Das ist ein Pseudobefehl — also kein 6502-Befehl —, der die Ausgabe des Protokolls über den Drucker bewirkt. Zeile 40 enthält durch .BA $5000 wieder einen Pseudobefehl. Damit legt man fest, von welcher Adresse an der Maschinencode in den Speicher gelegt werden soll. Die Zeilen 50 bis 90 sind unser Assemblerprogramm. Zuerst wird ein Startwert 0 in das Y-Register geschrieben und dieses dann in der Zeile mit dem Label um 1 hochgezählt.

Falls Ihnen der Ausdruck Label noch nicht geläufig ist: Natürlich kann man auch statt dessen die Adresse 5002 hinter den BCC-Befehl in Zeile 80 schreiben — so haben wir das ja bisher immer mit dem SMON-Assembler getan. Das hätte aber im Quelltext, den wir hier schreiben, den Nachteil, daß wir diese Adresse jedesmal ändern müßten, wenn wir uns entschlössen, mit dem Pseudobefehl .BA den Programmstart zu verlegen. Indem wir aber diese Zeile durch das Label-Kennzeichen markieren, merkt sich der Hypra-Ass die dazugehörige Zeilennummer und rechnet sie beim Assemblieren automatisch in die richtige Sprungadresse um.

Ein weiterer Vorteil ist, daß man zu Dokumentationszwekken jede wichtige Adresse auf diese Weise markieren und sich am Schluß durch eine Symboltabelle ausgeben lassen kann. Besonders bei langen Programmen, in denen man dann sinnvolle Labelnamen verwendet (beispielsweise DRUCKEN am Anfang des Programmteils, das einen Ausdruck steuert), kann das eine unschätzbare Hilfe sein.

In unserem Programm in Bild 1 geht es weiter mit dem Vergleich, ob im Y-Register nach der Erhöhung schon $FF erreicht wurde. Ist das nicht der Fall, dann ist das Carry-Bit frei und der Programmablauf verzweigt zurück zur Labelzeile. Ansonsten ist die Verzögerungsschleife beendet und mit dem BRK meldet sich der SMON, den Sie zu diesem Zeitpunkt natürlich im Speicher haben sollten (vergessen Sie nicht, den SMON zumindest einmal zu starten mit SYS »startadresse«, damit bei einem BRK in den SMON gesprungen wird).

Falls Sie das Programm durch SYS $5000 vom Hypra-Ass aus gestartet haben (und nicht durch G 5000 aus dem SMON), finden Sie sich ebenfalls im Monitor wieder. Fast alle Beispielprogramme in diesem Kurs werden mit BRK enden. Der Grund dafür ist, daß es oft interessant ist, die Register nach dem Programmende zu beobachten. Sollten Sie ohne Monitor arbeiten wollen, dann müßten Sie statt dessen ein

RTS einsetzen. Hinter dem eigentlichen Programm finden Sie .SY 1,4. Auch das ist ein Pseudobefehl, der die Ausgabe der Symboltabelle über den Drukker bewirkt. Nicht sichtbar ist ein Befehl .ST, mit dem die Assemblierung beendet wird. Einige interessante Angaben besorgt uns der Hypra-Ass noch nach der kurzen Symboltabelle: Eine Zeitangabe und den Bereich, in dem der Maschinencode nun nach der Assemblierung zu finden ist. Falls Sie diesen Objektcode (so nennt man den Maschinencode auch häufig) speichern wollen (vom Monitor aus mit dem S-Kommando möglich), dann brauchen Sie diese Angaben. Unser Programm würde dann so abgespeichert:
S"OBJ.VERZ.VAR1" 5000 5008
(Man muß immer ein Byte zur Endadresse hinzurechnen beim Speichern des Objektcode). Eine andere Möglichkeit, den Objektcode auf Diskette zu speichern: Nach dem .LI 1,4 in der nächsten Zeile folgenden Befehl einsetzen:
10 -OB"OBJ.VERZ.VAR1,p,w
Jetzt wird nach dem Starten des Assemblierens mit RUN automatisch das Maschinenprogramm gespeichert.

Die weiteren Programmbeispiele werde ich nicht so erschöpfend erklären wie dieses. Nur wenn neue Pseudobefehle verwendet werden oder eine neue Programmstruktur das erfordert, geht's nochmal in die Tiefe. Um etwas Platz zu sparen, wurden die folgenden Programme nicht mit dem .LI - Befehl aus dem Drucker ausgegeben, sondern mit
OPEN 1,4:CMD1
/E
Dadurch werden die Adressen mit den Hex-Codes der Maschinenbefehle nicht gedruckt, sondern nur das Listing, wie es auch auf dem Bildschirm zu sehen ist.

Häufig tritt in Schleifen der Fall ein, daß weder das Y- noch das X-Register zur Verfügung stehen. Sie dienen dann anderweitig schon als Index. Statt dessen kann ebensogut eine Speicherstelle den Zähler bilden, wie in dieser Variante 3:

```
      LDA   #$00
      STA   $FB     ;$FB ist Zähler
LABEL INC   $FB
      LDA   $FB
      CMP   #$FF
      BCC   LABEL
      RTS
```

Selbstverständlich kann auch jede andere Speicherstelle anstelle von $FB verwendet werden, sogar eine, die nicht in der Zeropage liegt. Voraussetzung ist lediglich, daß sie nicht innerhalb der Schleife verändert wird — außer zum Zählen der Schleifendurchläufe. In den bisher kennengelernten Varianten haben wir immer $FF als Endwert genommen. Nun steht man oft vor der Aufgabe, bis zu einem bestimmten Endwert zu zählen, der vorher irgendwie eingegeben oder festgelegt wird. In Basic sähe das beispielsweise so aus:

```
10 A = 32
20 FOR I = 0 TO A
30 NEXT I
```

Hier ist also der Endwert in Zeile 10 auf 32 gesetzt worden und die Schleife zählt bis zu diesem in A fixierten Wert. In Assembler können wir das ebenfalls. Bild 2 zeigt die Variante 4.

Die Speicherstelle $FA nimmt die Funktion der Variablen A des Basicprogrammes ein. Dorthinein wird der Endwert (32= $20) gelegt und der Vergleichsbefehl lautet nun:
CPY $FA
Das ist: »Vergleiche den Inhalt des Y-Registers mit dem Inhalt der Speicherstelle $FA«. Wir haben in dieser Version 4 wieder das Y-Register als Zähler benutzt, Version 5 zeigt uns in Bild 3 dasselbe, nur wird hier die Speicherstelle $5100 zum zählen verwendet.

Es hat sich eingebürgert, Schleifen in Assembler nicht — wie wir es bisher getan haben — aufwärts, sondern sie abwärts zu zählen. Der Grund dafür ist: Es geht schneller, weil man sich meistens den Compare-Befehl ersparen kann. Bei Verzögerungsschleifen ist das ja noch nicht so interessant, später aber, wenn in den Schleifen noch allerhand geschehen soll, summieren sich die Taktzeiten bei mehrfachen Durchlauf schon ganz erheblich. Eine Basic-Programmsequenz sähe nun so aus:

```
10 FOR I = 255 TO 0 STEP -1
20 NEXT I
```

In Bild 4 finden Sie das Assemblerlisting der Variante 6.

Das entspricht der Variante 1. Der Unterschied ist aber, daß hier abwärts gezählt wird und man sich den CPY-Befehl sparen kann, denn vor einem Unterlauf des Y-Registers wird automatisch bei 0 die Zero-Flagge gesetzt. Das aber prüft der BNE-Befehl.

Aus alledem ist also zu lernen:
1) Wann immer möglich, abwärts zählen.
2) Wann immer möglich, X- oder Y-Register als Zähler verwenden.

Die Variante 5 war natürlich ein ausgesuchtes Extrembeispiel, denn außer der Tatsache, daß man beim Abwärtszählen den Endwert als Startwert immer gleich in den Zähler eingeben kann und ihn normalerweise nicht noch irgendwo speichern muß, verwendet man natürlich — wenn es denn nötig ist, etwas anderes als die Indexregister dazu zu gebrauchen — eine Zeropagespeicherstelle als Zähler und nicht — wie in Version 5 — eine Speicherstelle wie $5100. Die verbesserte Version 7 entspricht dem Basicprogramm:

```
10 A = 32
20 FOR I = A TO 0 STEP -1
30 NEXTI
```

```
HYPRA-ASS  ASSEMBLERLISTING:

               5   -            .LI 1,4
;*** VERZOEGERUNGSSCHLEIFE VARIANTEN 1 UND 2 ***
; X- ODER Y-REGISTER ALS ZAEHLER
;
              40   -            .BA $5000
5000 A000     :50  -            LDY #$00        ;BZW. LDX
5002 C8       :60  -LABEL       INY             ;      INX
5003 C0FF     :70  -            CPY #$FF        ;      CPX
5005 90FB     :80  -            BCC LABEL       ;WENN <255
5007 00       :90  -            BRK
              100  -            .SY 1,4

SYMBOLS IN ALPHABETICAL ORDER:

LABEL            = $5002

END OF ASSEMBLY  0:14.6
BASE = $5000  LAST BYTE AT $5007
```

Bild 1. Etwas zögern mit Variante 1 und 2

```
10   -            .LI 1,4
20   -            .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 4 ***
40   -;VARIABLER ENDWERT
50   -;Y-REGISTER ALS ZAEHLER
60   -;
70   -            LDA #$20        ;DAS IST DEZIMAL 32
80   -            STA $FA         ;ENDWERT SPEICHERN
90   -;
100  -            LDY #$00        ;Y-REGISTER INITIALISIEREN
110  -LABEL       INY
120  -            CPY $FA         ;ENDWERT ERREICHT?
130  -            BCC LABEL       ;NEIN:DANN WEITERZAEHLEN
140  -            BRK
150  -;
160  -            .SY 1,4
170  -            .ST
```

Bild 2. Verschieden zögern mit Variante 4

```
10   -            .LI 1,4
20   -            .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 5 ***
40   -;VARIABLER ENDWERT
50   -;SPEICHERSTELLE ALS ZAEHLER
60   -;
70   -            LDA #$20        ;DAS IST DEZIMAL 32
80   -            STA $FA         ;ENDWERT SPEICHERN
90   -;
100  -            LDA #$00        ;ZAEHLER INITIALISIEREN
110  -            STA $5100       ;DA IST ER:UNSER ZAEHLER
120  -;
130  -LABEL       INC $5100
140  -            LDA $5100
150  -            CMP $FA         ;VERGLEICHEN MIT ENDWERT
160  -            BCC LABEL       ;WEITERZAEHLEN WENN <> ENDWERT
170  -            BRK
180  -            .SY 1,4
190  -            .ST
```

Bild 3. Das ist die Variante 5

```
10   -            .LI 1,4
20   -            .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VARIANTE 6 ***
40   -;ABWAERTS ZAEHLEN (Y-REGISTER)
50   -;
60   -            LDY #$FF        ;STARTWERT NACH Y
70   -LABEL       DEY
80   -            BNE LABEL       ;WEITER BIS Y = 0
90   -            BRK
100  -;
110  -            .SY 1,4
120  -            .ST
```

Bild 4. Rückwärts zögern mit Variante 6

```
10   -            .LI 1,4
20   -            .BA $5000
30   -;*** VERZOEGERUNGSSCHLEIFE VERSION 7 ***
40   -;ABWAERTS ZAEHLEN ($FA ALS ZAEHLER)
50   -;
60   -            LDA #$20        ;STARTWERT IN ZAEHLER
70   -            STA $FA         ;SCHREIBEN
80   -LABEL       DEC $FA
90   -            BNE LABEL       ;WEITER BIS $FA = 0
100  -            BRK
110  -;
120  -            .SY 1,4
130  -            .ST
```

Bild 5. Da haben wir die Variante 7

In Bild 5 finden Sie diese Version. $FA dient als Zähler.

Etwas schwieriger wird die Programmierung, wenn man nicht nur um 1 herauf- oder herunterzählt, sondern um 2,3,4 oder mehr. Das Basic-Äquivalent drückt sich dann beispielsweise in der Ergänzung STEP -2 der FOR...NEXT-Schleife aus. Dreht es sich nur um kleine Schrittweiten, die konstant bleiben, dann verwendet man vorteilhaft mehrere DEY (oder DEX, DEC, INY, INX und INC) hintereinander. Man muß außerdem mit der Abbruchbedingung einer solchen Schleife vorsichtig sein. BNE ist nicht immer möglich, weil man unter Umständen schon vor der Prüfung (durch BNE) unter 0 hindurch gezählt hat (dann folgt ja wieder $FF etc.). Hat beispielsweise der Zähler (hier das Y-Register) den Wert 1 und es werde durch eine Sequenz:

DEY
DEY
DEY
BNE LABEL

weitergezählt, dann nimmt Y der Reihe nach die Werte 0,FF,FE an und BNE findet die Zeroflagge nicht gesetzt. Man muß also andere Abbruchbedingungen verwenden. Solange man bis zur ersten Prüfung (also dem ersten Schleifendurchlauf beim Herunterzählen) im Zähler mindestens $7F (=binär 0111 1111) vorliegen hat, kann man mittels BPL die Schleife schließen. Zur Erinnerung: BPL verzweigt, wenn Bit 7 nicht gesetzt ist (kleiner 128), BMI verzweigt, wenn Bit 7 gesetzt ist (größer oder gleich 128). Das Basic-Programmstück

```
10 FOR I = 32 TO 0 STEP -3
20 NEXT I
```

findet seine Entsprechung in dem Assemblerlisting Version 8 in Bild 6.

Wieder dient das Y-Register als Schleifenzähler.

Größere Schrittweiten lassen es — von einer gewissen Grenze an, die durch das Verhältnis von Bytezahl auf der einen und Bearbeitungsdauer auf der anderen Seite, bestimmt wird — sinnvoll erscheinen, den Zähler durch Subtraktion (oder Addition beim Aufwärtszählen) zu verändern. Das Analogon zur Basic-Sequenz:

```
10 FOR I = 127 TO 0 STEP -10
20 NEXT I
```

sehen Sie in Bild 7.

In dieser Version 9 dient die Zeropagespeicherstelle $FA als Zähler und in den Programmzeilen 80 bis 110 findet die Verminderung dieses Zählers durch Subtraktion statt ($0A = dezimal 10). Das Programm kann noch verändert werden, indem man anstelle von BPL den BCS-Befehl verwendet. Wenn die Subtraktion einen Unterlauf ergeben hat, wird das Carry-Bit gelöscht.

```
10  -           .LI 1,4
20  -           .BA $5000
30  -;*** VERZOEGERUNGSSCHLEIFE VERSION 8 ***
40  -;FOR I = 32 TO 0 STEP-3
50  -;
60  -           LDY #$20        ;STARTWERT IN ZAEHLER
70  -LABEL      DEY             ;MINUS 3
80  -           DEY
90  -           DEY
100 -           BPL LABEL       ;WEITER BIS UNTERLAUF
110 -           BRK
120 -;
130 -           .SY 1,4
140 -           .ST
```

Bild 6. Verzögern in kleinen Schritten mittels Variante 8

```
10  -           .LI 1,4
20  -           .BA $5000
30  -;*** VERZOEGERUNGSSCHLEIFE VERSION 9 ***
40  -;FOR I = 127 TO 0 STEP -10
50  -;
60  -           LDA #$7F        ;DAS IST DEZIMAL 127
70  -           STA $FA         ; UNSER ZAEHLER
80  -LABEL      SEC
90  -           LDA $FA
100 -           SBC #$0A
110 -           STA $FA
120 -           BPL LABEL       ;WEITER BIS UNTERLAUF
130 -           BRK
140 -           .SY 1,4
150 -           .ST
```

Bild 7. Verzögern in großen Schritten mit Variante 9

Außerdem lassen sich noch 2 Byte einsparen, indem man das STA $FA aus Zeile 110 herausnimmt und dafür das LABEL eine Zeile höher setzt. Allerdings geht das dann auf Kosten der Durchschaubarkeit unseres Programmes.

Wir wollen nun mit den einfachen Verzögerungsschleifen aufhören. Es gäbe noch weitere Aufgaben zu lösen (nämlich beispielsweise von einem bestimmten Startwert bis zu einem bestimmten Zielwert zu zählen), die vertraue ich aber Ihnen selbst an: Alles notwendige dazu können Sie aus unseren verschiedenen Versionen entnehmen und kombinieren. Interessant werden Schleifen hauptsächlich durch einen Job, der in ihnen wiederholt ausgeführt wird. Zwei Beispiele sollen uns zur Illustration in dieser Folge dienen. Vorher aber sollen noch einige Bemerkungen zur grundsätzlichen Architektur von Schleifen gemacht werden.

Im Prinzip setzt sich jede Schleife aus vier Bestandteilen zusammen:

Initialisierung. Beispielsweise wird hier der Startwert des Zählers festgelegt.

Verarbeitung. Das ist das, was in den Verzögerungsschleifen bisher leer blieb: Der Job.

Steuerung. Hoch- oder Herunterzählen des Zählers und Prüfen der Abbruchbedingung.

Ausgang. Das war bisher bei uns immer der BRK-Befehl.

Aus diesen vier Bestandteilen lassen sich zwei grundsätzliche Schleifenmöglichkeiten konstruieren, die Sie in Bild 8 dargestellt finden.

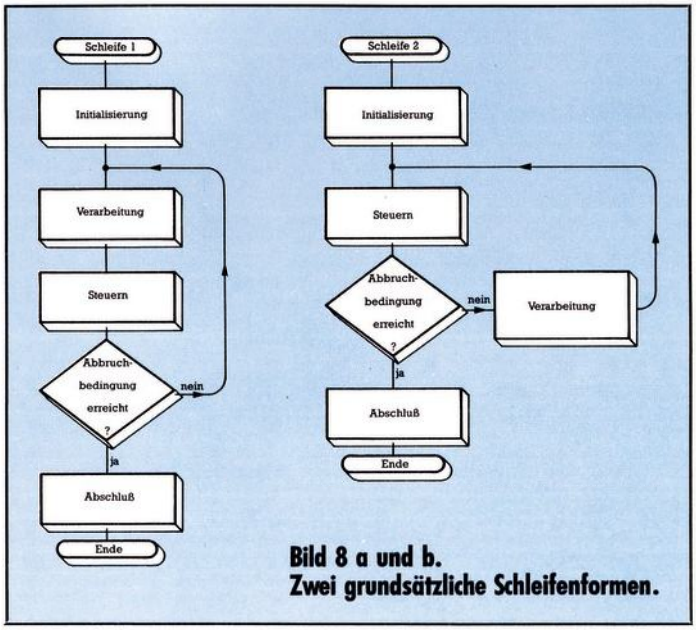

Bild 8 a und b. Zwei grundsätzliche Schleifenformen.

In Bild 8a haben wir das Prinzip vorliegen, das unseren normalen FOR...NEXT-Schleifen in Basic zugrundeliegt. Diese Schleife wird mindestens einmal durchlaufen. Erst nach der Ausführung des Jobs erfolgt die Prüfung, ob die Abbruchbedingung gegeben war. Soll solch eine Schleife n-mal durchlaufen werden, muß die Initialisierung mit n-1 im Zähler erfolgen (oder die Abbruchbedingung entsprechend umgeformt werden).

Die Schleifenkonstruktion in Bild 8b dagegen muß nicht durchlaufen werden. Ihr entspricht etwa eine DO UNTIL...LOOP-Schleife oder auch eine DO WHILE...LOOP-Schleife aus dem Basic 7.0 des C 128. Hier erfolgt die Initialisierung des Zählers genau mit dem Wert n.

Sehen wir uns beispielsweise unsere Version 9 an, dann entdecken wir die einzelnen Schleifenteile wie folgt:

```
Initialisierung:          LDA #$7F
                          STA $FA
Verarbeitung:LABEL        ---
Steuerung:                SEC
                          LDA $FA
                          SBC #$0A
                          STA $FA
                          BCS LABEL
Ausgang:                  BRK
```

Auf diese Weise ist es Ihnen möglich, alle bisher kennengelernten Schleifenvarianten mit einem beliebigen Job zu füllen. Noch eines gibt es zu bedenken: Alle Instruktionen zwischen dem Label und der Abbruchbedingung werden oft ausgeführt, sind also zeitfressend. Daher sollte der auszuführende Job alle Befehle vermeiden, die ebensogut vor der eigentlichen Schleife stehen könnten.

Sehen wir uns unser 1. Beispiel an. Wir stellen uns die Aufgabe, von den 127 Zeichen, die mittels POKE-Code erfaßbar sind, jedes 2. Zeichen an jeder 2. Bildschirmstelle abzubilden. Das Ganze soll durch Verwenden verschiedener Farben auch noch hübsch bunt aussehen. In Basic würden wir dafür schreiben:

```
10 S = 1024 : C = 55296
20 FOR I = 127 TO 0 STEP -2
30 POKE S+I,I
40 POKE C+I,I
50 NEXT I
```

Weil im Bildschirmfarbspeicher nur die Bits 0 bis 3 eine Rolle spielen (die anderen aber gar nicht beachtet werden), erzeugen wir in Zeile 40 auch die verschiedenen Farben mehrmals nacheinander.

(Heimo Ponnath/gk)

Fortsetzung folgt.