Memory Map mit Wandervorschlägen Teil 5

spiel ein A eintippen, erhalten Sie links den vorigen Wert von rechts und rechts jetzt eine um 1 kleinere Zahl. Eine weitere Eingabe von zum Beispiel XXXXX schiebt die alte rechte Zahl nach links und die neue wird um die Anzahl der Zeichen, also 5, verringert.

Adresse 55-56 ($37-$38)

Zeiger auf das Ende des für Basic-Programme verfügbaren Speichers

Dieser Zeiger, in der Low/High-Byte-Darstellung, gibt dem Basic-Übersetzer an, welches die höchste von Basic verwendbare Speicheradresse ist. Wie aus Bild 1 ersichtlich ist, ist diese Adresse zugleich der Anfang der als Variable abgespeicherten Zeichenkette (Strings).

Normalerweise ist diese Adresse fest vorgegeben. Die folgende Tabelle gibt darüber Auskunft:

Ende des Programmspeichers

	Adresse	Zeiger in 55 56
C 64	40960	0160
VC 20 (Grundv.)	7680	030
VC 20 (+3K)	7680	030
VC 20 (+8K)	16384	064
VC 20 (+16K)	24576	096
VC 20 (+24K)	32768	0128

Beim Einschalten des Computers überprüft das Betriebssystem den gesamten RAM-Speicher, bis es zur ersten ROM-Speicherzelle kommt, setzt den Zeiger in 55-56 auf diese Adresse und druckt den bekannten Kopf mit der verfügbaren Speicherangabe auf den Bildschirm.

Normalerweise wird dieser Zeiger nicht geändert.

Es gibt aber zwei Gelegenheiten, bei denen eine Änderung dieses Zeigers sinnvoll beziehungsweise notwendig ist.

Anwendung 1:

Es kommt oft vor, daß der gesamte Speicher nicht ausschließlich für Basic benötigt wird, sondern daß ein freier Speicherbereich geschaffen wird, um zum Beispiel Maschinenprogramme, selbst definierte Zeichen oder hochaufgelöste Grafik unterzubringen, die aber nicht vom Basic-Programm überschrieben werden können.

Bei der Besprechung der Zeiger in 43-44 haben wir das auch schon gemacht, allerdings durch »Hochschieben« des Speicheranfangs. Mit dem Zeiger in 55-56 erreichen wir denselben Effekt, diesmal durch »Hinunterdrücken« des Speicherendes. Gegenüber den vier Schritten beim Hochschieben ist das Hinunterdrücken einfacher. Mit dem Befehl:

POKE 56,PEEK(56)—1:CLR

schieben wir das Speicherende um 256 Byte nach unten, egal für welchen Computer und welche Speichererweiterung. Mit —2 verschiebt sich das Ende um 512, mit —4 um 1 024 Byte (also 1 KByte) nach unten. Wenn Sie eine feinere Verschiebung als Vielfache von 256 benötigen, kommen Sie mit dem High-Byte in 56 allein nicht aus, sondern Sie müssen auch einen entsprechenden Wert in 55 hineinPOKEn.

Der Befehl CLR ist absolut notwendig, denn er setzt den Zeiger der Zellen 51-52 (siehe dort), das heißt das untere Ende des Speicherbereichs für Zeichenketten auf dieselbe Adresse wie Zeiger 55-56. Dadurch wird erzwungen, daß die Zeichenkette sozusagen als Ausgangslage unterhalb des heruntergedrückten Speicherendes abgelegt werden.

Anwendung 2:

Über den User-Port (Steckerleiste an der Rückseite, neben dem Datasetten-Anschluß) können VC 20 und C 64 mit anderen Geräten verbunden werden. Der Datentransfer über diese Verbindung — sie heißt RS232-Schnittstelle — muß allerdings programmiert werden. Diese RS232-Schnittstelle hat die Gerätenummer 2 (so wie der Drucker Nummer 4 und das Diskettengerät die Nummer 8 hat).

Wenn nun ein Gerät Nummer 2 mit einem OPEN-Befehl angewählt wird, wird automatisch der Zeiger in 55-56 und der Zeiger in 643 um 512 Bytes heruntergedrückt, um je einen Eingangs- und Ausgangspufferspeicher zu erzeugen. Da der Inhalt dieser Pufferspeicher alle Variable in diesen 512 Bytes überschreiben würde, wird auch der CLR-Befehl automatisch gegeben.

Es gilt daher als Vorschrift, daß bei RS232-Verbindungen zuerst der Datenkanal durch OPEN eröffnet werden muß, bevor Variable, Felder und Zeichenketten definiert werden.

Dieser Zeiger beschließt die Gruppe der Speicherzeiger. Das nächste Mal machen wir ab Adresse 57 weiter.

(Dr. Helmuth Hauck/gk)

Finden mit System — Eine neuartige Suchmethode (Teil 3)

Nachdem wir uns in den letzten beiden Folgen mit grundlegenden programmiertechnischen Fragen befaßt haben, sollen jetzt konkrete Anwendungen im Mittelpunkt stehen. Diesmal ist es das Suchen von Zeichenketten mit einer erst vor kurzen entwickelten Methode.

Wie versprochen, stelle ich Ihnen diesmal ein Suchprogramm vor, mit dem eine schnelle »intelligente« Suche von Strings möglich wird. Doch bevor wir uns mit der Praxis, sprich dem Programm, beschäftigen können, müssen wir uns ein wenig mit der Theorie des Suchens befassen.

Wie sucht man eigentlich?

Nun, uns soll hier nicht interessieren, wie der eine oder andere mit seiner häuslichen (Un-) Ordnung fertig wird. Uns geht es hier um das Suchen und möglichst auch Finden von einer Zeichenkette in einer anderen. Einigen wir uns jetzt schon auf zwei Begriffe, damit wir später keine Schwierigkeiten bekommen. Der Suchstring ist der String, nach dem wir suchen. Dies kann eine beliebige Zeichenkette, also ein Wort, eine Zahl, Grafikzeichen oder ein Gemisch aus alledem sein. Der Durchsuchstring ist derjenige, in dem wir suchen.

Das landläufige Suchverfahren sieht folgendermaßen aus: Der Anfang des Suchstring wird an den Anfang des Durchsuchstrings angelegt. Dann werden die beiden ersten Buchstaben verglichen. Sind diese gleich, werden die beiden zweiten Buchstaben verglichen, und so weiter. Dieses Spielchen wiederholt sich so lange, bis sich die beiden Buchstaben nicht gleichen. Dann wird der Suchstring um eine Position am Durchsuchstring verschoben, und somit der erste Buchstabe des Suchstrings mit dem zweiten des Durchsuchstrings und so weiter, verglichen. Wird beim Suchen festgestellt, daß alle Buchstaben übereinstimmen, so hat man den Suchstring im Durchsuchstring gefunden. Stellt man aber fest, daß das Ende des Suchstrings über den Durchsuchstring hinausragt, so muß man die Suche abbrechen, da der Suchstring nicht im Durchsuchstring vorhanden sein kann.

Dieses Suchverfahren war jahrelang das einzige verwendete, da es relativ einfach zu programmieren und relativ schnell war.

Es geht auch anders

In der November-Ausgabe der Zeitschrift Spektrum der Wissenschaft wurde nun ein völlig neuer Suchalgorithmus vorgestellt. Dieser ist erst vor kurzem von zwei amerikanischen Forschern entwickelt worden, und kann mit noch vertretbarem Aufwand in Maschinensprache formuliert werden. Noch einmal das Prinzip des alten Algorithmus: Man verschiebt den Suchstring am Durchsuchstring immer nur um eine einzige Position, braucht also mindestens so viele Vergleiche, wie Zeichen im Suchstring minus Zeichen im Durchsuchstring (bei Überhang wird ja abgebrochen). Findet man einen Weg, den Suchstring immer gleich um mehrere Positionen zu verschieben, ohne daß dabei ein Auftauchen des Suchstrings im Durchsuchstring übersehen werden kann, läßt sich der Suchvorgang rapide beschleunigen. Einfach um einen größeren konstanten Faktor zu verschieben, etwa um 2, klappt natürlich nicht, weil dann nur alle ungeraden Positionen abgefragt werden. Beginnt der Suchstring aber auf einer geraden Position, wird er nicht gefunden. Dieser Verschiebefaktor muß also variabel sein. Um ihn zu ermitteln, stellen wir mal fest, wie weit in einigen Fällen maximal verschoben werden darf, um den Suchstring nicht zu verpassen.

Im folgenden werden wir immer nach dem Wort »HALLO«

Fortsetzung auf Seite 151

Fortsetzung von Seite 148

suchen. Wenn wir das Wort »HALLO« im String »SCHÖNES WETTER HEUTE« suchen, dann legen wir erstmal »HALLO« ganz normal an der ersten Position an. Nun vergleichen wir von hinten! Die beiden letzten Buchstaben, »N« und »O« stimmen offensichtlich nicht miteinander überein. Also kann »Hallo« nicht am Anfang des Strings stehen, und darf somit gleich um fünf Positionen, dies ist gleich der Länge von »Hallo« verschoben werden.
Hier das Ganze im Bild:

```
SCHÖNES WETTER HEUTE
HALLO
      HALLO
            HALLO...
```

Doch es ist nicht immer ganz so einfach. Suchen wir mal im String »DU HALLODRI«. Beim Anlegen und Vergleichen trifft das »O« aus »HALLO» auf das »A« von »DU HALLODRI«. Würden wir jetzt, da die beiden ja ungleich sind, »HALLO« um fünf Positionen verschieben, würden wir leider das »HALLO« in »HALLODRI« übersehen, da wir schon zu weit verschoben haben:

```
DU HALLODRI
HALLO
      HALLO
```

Wie weit darf man in diesem Fall verschieben? Nun, um maximal drei Positionen, dann liegt das »HALLO« nämlich genau an der richtigen Stelle:

```
DU HALLODRI
HALLO
   HALLO
```

Das ist ja ganz schön und gut, daß wir wissen, daß jetzt maximal um drei Positionen verschoben werden darf, aber wie sag ich's dem Computer? Auf die Lösung zu kommen ist genial und deshalb verblüffend einfach. Zählen wir einmal die Buchstaben von »HALLO« von hinten mit Null beginnend durch.

```
H A L L O
4 3 2 1 0
```

Das A erhält dann die Stellungsnummer 3. Diese stimmt aber nun mit dem maximalen Verschiebefaktor überein. Zufall? Nun machen wir noch ein Beispiel. Suchen wir PUTER in COMPUTER.
Erst einmal durchzählen:

```
P U T E R
4 3 2 1 0
```

Und dann:

```
C O M P U T E R
P U T E R
      P U T E R
```

Der letzte Buchstabe in PUTER, das R stößt auf ein U im Computer. Dessen Stellungsnummer ist aber wieder 3, es darf also wieder um drei Positionen verschoben werden. Daß das tatsächlich immer funktioniert, kann man auch mathematisch beweisen, würde jedoch hier den Rahmen sprengen.

Das Verfahren, den maximalen Verschiebefaktor herauszufinden, sieht im groben also so aus: Bevor der Suchvorgang gestartet wird, wird eine Tabelle mit den Stellungsnummern der Buchstaben im Suchstring angelegt. Zwei Sonderfälle haben wir noch nicht betrachtet, die beim Tabellenanlegen aber automatisch mitberücksichtigt werden:

Sollten Buchstaben im Suchstring doppelt auftauchen, so wird für einen Buchstaben immer die kleinste Stellungsnummer gespeichert. Dies läßt sich sehr einfach dadurch erreichen, daß der Suchstring beim Anlegen der Tabelle von vorne nach hinten durchgegangen wird. Taucht dann ein Buchstabe ein zweites Mal auf, so wird automatisch die größere Stellungsnummer von der kleineren überschrieben. Ein Beispiel:
Suchen wir mal »COMMODORE« in

```
»     COMMODORE«.
```

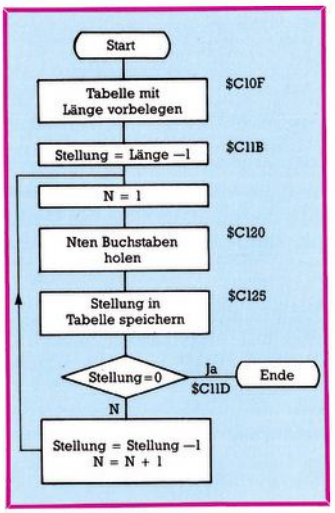

Bild 1. Der Algorithmus zur Tabellenbildung

Zuerst einmal durchzählen:

```
C O M M O D O R E
8 7 6 5 4 3 2 1 0
   COMMODORE
COMMODORE
```

Das »E« stößt nun auf ein »O«. Nun gibt es aber drei »O«. Welche Stellungsnummer darf hier genommen werden? Die des letzten »O«, die »2«. Mit einer Verschiebung von zwei haben wir dann ja auch die Übereinstimmung gefunden. Durch den Tabellen-Algorithmus wird immer automatisch die Stellungsnummer des letzten »O« genommen.

Die zweite Regel befaßt sich mit den Buchstaben. Die erhalten als Stellungsnummer die Länge des Suchstrings, da ja in diesem Fall um die gesamte Suchstringlänge verschoben werden kann. (Siehe auch unser allererstes Beispiel). Somit ist der Algorithmus zur Tabellenbildung schon formulierbar. Er ist in Bild 1 als Flußdiagramm dargestellt.

Bis jetzt habe ich immer von einem maximalen Verschiebefaktor gesprochen. Denn nicht immer darf um den vollen Faktor verschoben werden. Wir haben bisher nämlich noch nicht den Fall betrachtet, daß teilweise oder volle Übereinstimmung von Such- und Durchsuchstring eintritt.

Suchen wir mal »HOHO« in »HIHOHOHI«. Beim ersten Anlegen von »HOHO« stimmen die letzten beiden Buchstaben überein, der zweite aber nicht. Würde jetzt um den Tabellenwert — bei dem »I« ist er 4, verschoben, so würden wir um zwei Buchstaben zu weit verschoben haben!:

```
H I H O H O H I
H O H O
      H O H O
```

Um dies zu vermeiden, müssen wir einen weiteren Begriff einführen, die Suchtiefe. Die Suchtiefe bezeichnet nichts anderes, als die Anzahl der positiv ausgefallen Vergleiche. In unserem Beispiel wäre sie 2, da zwei Vergleiche, »O« mit »O« und »H« mit »H«, positiv ausgefallen sind. Der dritte Vergleich »I« mit »O« ist nun negativ, das heißt, daß der Suchbegriff dort nicht auftauchen kann, und nun verschoben werden muß. Um den tatsächlichen Verschiebefaktor zu ermitteln, muß nun noch von dem aus der Tabelle entnommenen Wert die Suchtiefe subtrahiert werden. In unserem Fall wäre das 4 minus 2 = 2. Und das stimmt wieder:

```
H I H O H O H I
H O H O
   H O H O
```

Verschieben wir um zwei Positionen, so liegt unser »HOHO« genau an der richtigen Stelle. Die Suchtiefe kann minimal Null werden, wenn schon der erste Vergleich negativ ausfällt, und maximal gleich der Länge des Suchstrings, wenn nämlich völlige Übereinstimmung herrscht, der Begriff also gefunden wurde.

Das wäre an sich schon alles. Theoretisch ließe sich auch schon jetzt der Suchalgorithmus formulieren. Aber eine Kleinigkeit würde uns jetzt noch Schwierigkeiten machen. Gemeint ist das Suchen mit Joker. Den Floppybesitzern unter Ihnen wird es wahrscheinlich schon bekannt sein.

Probleme mit dem Joker

Nehmen wir einmal an, Sie führen eine Adressendatei und würden nun gerne die Adresse von einem Herrn Maier haben. Dummerweise wissen Sie nun nicht mehr, ob es Herr Maier oder vielleicht Herr Meier oder gar Herr Mayer war. Beim Durchsuchen wäre also ein Joker sehr praktisch. Dann suchen sie nach Herrn M??er. Die Suchroutine müßte ihnen nun alle nur erdenklichen Formen des Namens Maier heraussuchen. Dummerweise ist ein Joker in unserem Suchalgorithmus gar nicht so einfach zu installieren, und wirft uns auch, wie wir noch sehen werden, geschwindigkeitsmäßig zurück.

Konstruieren wir uns mal wieder ein Beispiel. Wir suchen nach »H?HI« in »HIHXHI«. Beim ersten Anlege- und Vergleichsvorgang wird das »I« mit einem »X« verglichen. Da der Vergleich ungleich ausfällt, und »X«, da nicht im Suchstring enthalten, einen Maximalverschiebefaktor von 4 liefert, und die Suchtiefe

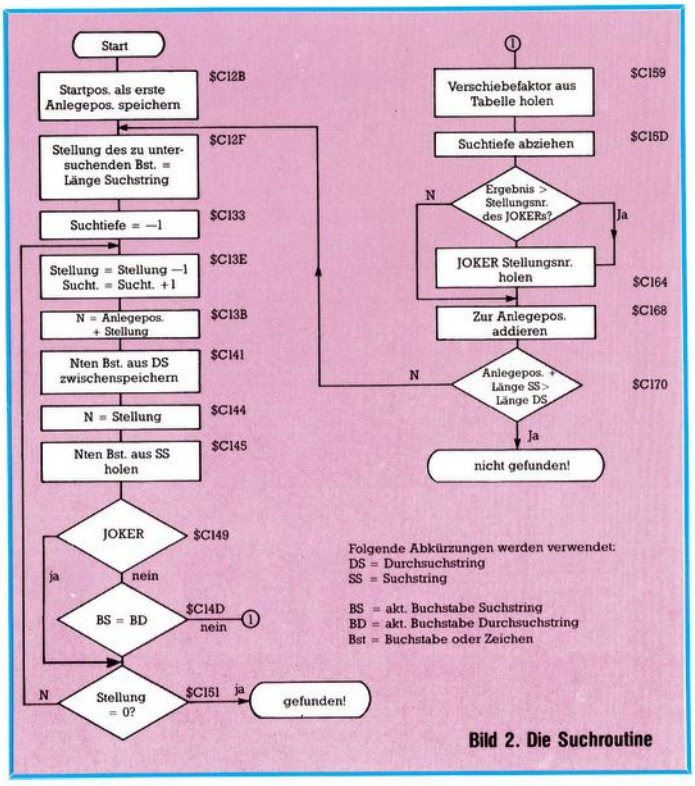

Bild 2. Die Suchroutine

hier auch noch 0 ist, würde um vier Zeichen verschoben, und das Auftauchen von »HXHI« an der Position 3 völlig übersehen.

```
H I H X H I
H ? H I
      H ? H I
```

Fragen wir uns also wieder einmal, um wieviel hier maximal verschoben werden kann. Die Antwort lautet 2, da ja »HXHI« an der Position 3 steht.

```
H I H X H I
H ? H I
    H ? H I
```

Nun ist die Stellungsnummer des Jokers gerade aber auch 2. Ohne näheren Beweis und Erklärung muß der Algorithmus noch folgende Regel zusätzlich enthalten:

Es darf maximal um die Stellungsnummer des Jokers im Suchstring verschoben werden. Damit läßt sich der Suchalgorithmus als Flußdiagramm formulieren. Dies ist in Bild 2 dargestellt. Das Flußdiagramm ist übrigens parallel zur Suchroutine im Suchprogramm zu sehen, da sich beide in der Struktur gleichen. Diese Suchroutine steht in den Speicherzellen $C12B-$C179. Deswegen ist das Flußdiagramm auch nicht ganz so elegant, wie es eigentlich sein könnte.

Lohnt sich das überhaupt?

Das ist jetzt natürlich die entscheidende Frage. Was haben wir dabei eigentlich gewonnen? Dem Verschieben können um mehrere Positionen, steht ein ungleich hoher Aufwand gegenüber. Dazu sei gleich gesagt, daß es Unsinn wäre, das Ganze in eine normale INSTR-Funktion, wie in Ex-, Simons- und GBasic enthalten, einzubauen. Dann lohnt sich der Aufwand eben nicht. Ich habe einmal überschlagsweise mein Programm (das ich »Intellisearch« = intelligentes Suchen genannt habe) mit diesen Erweiterungen verglichen, indem ich einen String in einem anderen gesucht habe. Fazit: Fast alle sind ungefähr fünfmal so schnell wie Intellisearch.

Aber: Diese INSTR-Funktionen lassen sich nur jeweils auf einen einzigen String anwenden. Will man eine größere Anzahl von Strings durchsuchen, etwa in einem Feld (Array), ist eine FOR-NEXT-Schleife notwendig. Im Gegensatz dazu ist Intellisearch darauf »abgerichtet«, Stringarrays zu durchsuchen. Und dann kommt man auf einmal auf Suchgeschwindigkeiten, die alles andere in den Schatten stellen. Zur Entlastung von Intellisearch muß nämlich folgendes gesagt werden. Die Parameterauswertung bei Intellisearch ist ungleich umfangreicher, als bei den INSTR-Funktionen. Außerdem wird Intellisearch über den SYS-Befehl aufgerufen, was ebenfalls sehr zeitintensiv ist (Zeitintensiver als der Aufruf über einen neuen Basic-Befehl auf jeden Fall).

Am allerwichtigsten aber: Ein Zeitverlust entsteht noch durch das Aufstellen der Tabelle. Der Suchalgorithmus selbst ist nämlich tatsächlich schneller. Aber da ja zuerst die Tabelle der Stellungsnummern aufgestellt werden muß, haben bei kurzen Durchsuchstrings die INSTR-Funktionen einen recht großen Zeitvorsprung, bevor Intellisearch mit dem Suchen anfangen kann. Die Anwendung lohnt sich also nur bei recht großen zu durchsuchenden Strings, oder gleich bei einer großen Anzahl zu durchsuchender Strings, am besten Arrays. Je mehr Strings durchsucht werden müssen, desto eher lohnt sich der Einsatz des Programms. Dann kann der Suchalgorithmus den gegebenen Zeitvorsprung locker wieder wettmachen, und um Faktoren von 2 bis 10, je nach Länge des Suchstrings, schneller sein. Denn je länger der Suchstring, desto weiter kann er bei völliger Nichtübereinstimmung in einem Schub verschoben werden, während der alte Algorithmus immer nur um 1 verschieben kann.

Es ist übrigens noch eine Erweiterung eingebaut, die auch verlangsamend wirken kann:

Das Füllsel.

Was ist ein Füllsel?

Manchmal hat mich bei der Floppy die Filenamenabkürzung mit »*« gestört. So organisiere ich meine Artikel meist indem ich ihnen einen Namen und dann Anhängsel wie «.LST01« oder ».TXT« gebe. Möchte ich nun einen Überblick über sämtliche auf der Diskette befindlichen Texte, ginge das nur, wenn das ».TXT« immer die letzten vier Buchstaben eines 16stelligen Filenamens wäre. Dann könnte ich nach »????????????.TXT« suchen. Und das ist mir zuviel Tipparbeit. Einfacher wäre es da, wenn man nach »*.TXT« suchen könnte. Aber nach dem »*« nimmt die Floppy 1541 keine weiteren Zeichen an.

Nun, in der Floppy 1541 kann ich das natürlich nicht so ohne weiteres ändern, aber immerhin in mein Suchprogramm einbauen. Suchen Sie also nach »HA*LO«, so wird sowohl »HALLO«, wie auch »HALLIHALLO«, und sogar »HALO« gefunden.

Eine Einschränkung muß dabei aber gemacht werden. Es ist nur ein Füllsel im Suchstring erlaubt, da der Speicherplatz in der Zeropage zu klein ist, noch mehr Werte ungefährlich zwischenzuspeichern. Denn Intellisearch zerlegt einen solchen String in zwei Teilstrings. Wird in einem String der erste Teilstring gefunden, so wird nach dem zweiten gesucht. Ist auch dieser vorhanden, so hat man gefunden was man sucht. Sonst muß die Suche beim nächsten Durchsuchstring wieder mit dem ersten Teilstring beginnen. Verständlicherweise benötigt das Zerlegen in zwei Teilstrings, und das ab und zu notwendige Erstellen der Tabelle eine gewisse Zeit, so daß das Suchen mit Füllsel etwas langsamer als das normale Suchen ist.

So arbeitet man mit Intellisearch

Nach all der grauen Theorie doch nun etwas Praxis. Intellisearch belegt die Speicherbereiche $C000 bis $C2FF. Das Programm selbst belegt dabei nur die Speicherstellen $C000 bis $C17A, um die Tabellenroutinen jedoch einfacher und somit das Programm einigermaßen verschieblich zu halten, liegt die Tabelle von $C200 bis $C2FF. Wer will, kann ja die Tabelle direkt hinter das Programm legen, um Platz zu sparen.

Sollte dieser Speicherbereich von Ihrem eigenen Programm oder einer Basic-Erweiterung anderweitig benötigt werden, so müssen Sie leider selber die Anpassungen an einen anderen Speicherbereich vornehmen. Mit dem dokumentierten Listing und dem SMON dürfte das aber kein Problem sein.

Im Vertrauen auf den MSE ist kein Basic-Lader mehr notwendig, das MSE-Listing ist also Listing 1. Im Programm werden zwei kleine Tricks verwendet, die vielleicht dem einen oder anderen von Ihnen nicht bekannt sind. Es sind dies die Maskierung des SEC-Befehls über den Bit-Befehl gegen Ende des Listings, und der recht brutale Aussprung aus der Zähler-Erhöhen-Routine ($C0E2), wenn die Suche beendet werden soll. Dort wird eine Rücksprungadresse über PLA PLA »vernichtet«, so daß ordnungsgemäß mit JMP »Klammer zu?« ausgesprungen werden kann. Dies entspricht einem POP oder DISPOSE RETURN bei den Basic-Erweiterungen G- und ExBasic, mit denen ein Unterprogramm, das über GOSUB aufgerufen wurde, mit GOTO und nicht mit RETURN verlassen werden kann. So, nun aber zum Aufruf der Routine. Dieser erfolgt über:

SYS 49152 (A$,B$(0),P,Q,X%,Y%)

Die Variablennamen sind hier ganz willkürlich gewählt, können also von Ihnen verändert werden.

A$ bezeichnet den zu suchenden String. Hier ist wohl keine besondere Erklärung nötig, außer daß wie schon erwähnt »?« als Joker und »*« als Füllsel gilt.

B$(0) bezeichnet das zu durchsuchende Array. Soll, was wenig sinnvoll ist, ein einzelner String durchsucht werden, darf hier auch einfach B$ stehen. Will man die Suche erst beim siebten Arrayelement starten, weil der String zum Beispiel schon im sechsten Element gefunden wurde (= B$(5), Zählung beginnt bei 0), darf dort auch B$(6) stehen. Dies ist dann der erste durchsuchte String.

P gibt die Anzahl der zu durchsuchenden Strings an. Wird beispielsweise bei B$(0) begonnen, und hat P den Wert 100, so wird nach dem Durchsuchen von B$(99) abgebrochen. Bei einem Startwert B$(17) wäre dies dann analog bei B$(116) der Fall. Diese Angabe ist stets sorgfältig zu handhaben, da Intellisearch nicht die Größe des zu durchsuchenden Arrays feststellt, und somit aus Versehen über das Array hinaus gesucht werden kann. Dies kann im ärgsten Fall zum Absturz führen!

Q ist die Startposition der Suche. Dies kann bei mehreren Daten in einem String recht nützlich sein. Steht etwa ein Name immer ab der 14ten Stelle im Array, so brauchen die ersten 13 Stellen nicht durchsucht zu werden, wenn nach einem Namen gesucht wird. Dann sollte Q den Wert 14 enthalten.

In **X%** und **Y%** wird am Ende das Ergebnis der Suche festgehalten. In X% steht die Nummer des Strings, in dem der Suchstring gefunden wurde, wenn mit B$(0) gestartet wurde, ansonsten muß zu X% noch P addiert werden, um die Position im Array zu bekommen.

Y% gibt schließlich die Position im String selbst an. Wird die

```
C000 : 20 FA AE 20 9A AD 20 A3  2C
C008 : B6 86 B2 84 B3 85 B4 A2  BE
C010 : 05 A9 00 85 B1 95 B5 CA  CF
C018 : 10 FB A0 00 B1 B2 C9 2A  7A
C020 : F0 07 C8 C4 B4 D0 F5 F0  EA
C028 : 1E 18 84 02 C8 98 65 B2  00
C030 : 85 B5 A5 B3 85 B6 90 02  C4
C038 : E6 B6 A5 B4 38 E5 02 85  3F
C040 : B7 C6 B7 A5 02 85 B4 20  5C
C048 : FD AE 20 9A AD A5 64 85  9C
C050 : B9 A5 65 85 BA 20 FD AE  E8
C058 : 20 9E B7 86 B8 20 FD AE  68
C060 : 20 9E B7 CA 86 B0 A5 B2  00
C068 : 85 F7 A5 B3 85 F8 A5 B4  E9
C070 : 85 F9 20 0F C1 A2 02 A0  56
C078 : 02 B1 B9 95 FA CA 88 10  BC
C080 : F8 20 2B C1 A5 FD 85 BB  63
C088 : B0 06 20 D0 C0 4C 75 C0  23
C090 : A5 B7 D0 03 4C AF C0 85  F6
C098 : F9 A5 B5 85 F7 A5 B6 85  14
C0A0 : F8 20 0F C1 20 2F C1 B0  B8
C0A8 : 06 20 D0 C0 4C 66 C0 20  46
C0B0 : F9 C0 A9 00 A0 00 91 64  8D
C0B8 : C8 A5 B1 91 64 20 F9 C0  A2
C0C0 : A9 00 A0 00 91 64 E6 BB  E1
C0C8 : A5 BB C8 91 64 4C F7 AE  95
C0D0 : E6 B1 C5 B8 B0 0C 18 A5  2E
C0D8 : B9 69 03 85 B9 90 02 E6  AD
C0E0 : BA 60 68 68 20 F9 C0 A9  1A
C0E8 : FF A0 00 91 64 C8 91 64  05
C0F0 : 20 FD AE 20 8B B0 4C F7  1E
C0F8 : AE 20 FD AE 20 8B B0 85  38
C100 : 64 B4 65 A5 0D D0 05 A5  6B
C108 : 0E F0 01 60 4C 99 AD A5  6E
C110 : F9 85 FF A2 00 9D 00 C2  92
C118 : E8 D0 FA A0 FF C8 C6 FF  9C
C120 : B1 F7 AA A5 FF 9D 00 C2  9E
C128 : D0 F3 60 A5 B0 85 FD A5  39
C130 : F9 85 FE A9 FF 85 02 C6  A2
C138 : FE E6 02 A5 FD 18 65 FE  13
C140 : AB B1 FB 85 FF A4 FE B1  F5
C148 : F7 C9 3F F0 04 C5 FF D0  22
C150 : 06 A5 FE F0 23 D0 E0 A6  90
C158 : FF BD 00 C2 38 E5 02 CD  E5
C160 : 3F C2 90 03 AD 3F C2 18  95
C168 : 65 FD 85 FD 65 F9 B0 06  E2
C170 : C5 FA 90 BB F0 B9 18 24  D3
C178 : 38 60 EB FF F5 A9 00 85  93
```

Listing 1. Das Suchprogramm »Intellisearch« muß mit dem MSE eingegeben werden

```
INTELLISEARCH
geschr. von B. Schneider, 20.12.84.

.; c000 20 fa ae jsr $aefa      Klammer auf?
.; c003 20 9a ad jsr $ad9a      Auswertung Teil 1
.; c006 20 a3 b6 jsr $b6a3      Auswertung Teil 2
.; c009 86 b2    stx $b2        Hi-Byte Adresse
.; c00b 84 b3    sty $b3        Lo-Byte
.; c00d 85 b4    sta $b4        Länge des Suchstrings

.; c00f a2 05    ldx #$05       Zwischenspeicher löschen
.; c011 a9 00    lda #$00
.; c013 85 b1    sta $b1
.; c015 95 b5    sta $b5,x
.; c017 ca       dex
.; c018 10 fb    bpl $c015

.; c01a a0 00    ldy #$00       Check auf Füllsel
.; c01c b1 b2    lda ($b2),y
.; c01e c9 2a    cmp #$2a
.; c020 f0 07    beq $c029
.; c022 c8       iny
.; c023 c4 b4    cpy $b4
.; c025 d0 f5    bne $c01c
.; c027 f0 1e    beq $c047

.; c029 18       clc            Zerlegung in zwei Teilstrings
.; c02a 84 02    sty $02        wenn Füllsel vorhanden.
.; c02c c8       iny
.; c02d 98       tya
.; c02e 65 b2    adc $b2
.; c030 85 b5    sta $b5
.; c032 a5 b3    lda $b3
.; c034 85 b6    sta $b6
.; c036 90 02    bcc $c03a
.; c038 e6 b6    inc $b6
.; c03a a5 b4    lda $b4
.; c03c 38       sec
.; c03d e5 02    sbc $02
.; c03f 85 b7    sta $b7
.; c041 c6 b7    dec $b7
.; c043 a5 02    lda $02
.; c045 85 b4    sta $b4

.; c047 20 fd ae jsr $aefd      Komma?
.; c04a 20 9a ad jsr $ad9a      Startstring holen
.; c04d a5 64    lda $64        Pointer auf Descriptor
.; c04f 85 b9    sta $b9        zwischenspeichern
.; c051 a5 65    lda $65
.; c053 85 ba    sta $ba

.; c055 20 fd ae jsr $aefd      Komma?
.; c058 20 9e b7 jsr $b79e      Anzahl der zu durchsuchenden
.; c05b 86 b8    stx $b8        Strings speichern

.; c05d 20 fd ae jsr $aefd      Komma?
.; c060 20 9e b7 jsr $b79e      Startposition der Suche
.; c063 ca       dex            speichern
.; c064 86 b0    stx $b0

.; c066 a5 b2    lda $b2        Descriptor von Teilstr.1
.; c068 85 f7    sta $f7        in Arbeitsspeicher kopieren
.; c06a a5 b3    lda $b3
.; c06c 85 f8    sta $f8
.; c06e a5 b4    lda $b4
.; c070 85 f9    sta $f9

.; c072 20 0f c1 jsr $c10f      Tabelle anlegen

.; c075 a2 02    ldx #$02       Descriptor des Durchsuch-
.; c077 a0 02    ldy #$02       strings in Arbeitsspeicher
.; c079 b1 b9    lda ($b9),y    kopieren
.; c07b 95 fa    sta $fa,x
.; c07d ca       dex
.; c07e 88       dey
.; c07f 10 f8    bpl $c079

.; c081 20 2b c1 jsr $c12b      Suchen

.; c084 a5 fd    lda $fd        Position merken
.; c086 85 bb    sta $bb
.; c088 b0 06    bcs $c090      gefunden? (c=1 ja)
.; c08a 20 d0 c0 jsr $c0d0      Pointer auf Descriptor erhöhen
.; c08d 4c 75 c0 jmp $c075      nächster Versuch

.; c090 a5 b7    lda $b7        Teilstring 2 vorhanden
.; c092 d0 03    bne $c097      ja, weitersuchen
.; c094 4c af c0 jmp $c0af      nein, fertig

.; c097 85 f9    sta $f9        Descriptor von Teilstring 2
.; c099 a5 b5    lda $b5        in Arbeitsspeicher kopieren
.; c09b 85 f7    sta $f7
.; c09d a5 b6    lda $b6
.; c09f 85 f8    sta $f8

.; c0a1 20 0f c1 jsr $c10f      Tabelle anlegen

.; c0a4 20 2f c1 jsr $c12f      suchen

.; c0a7 b0 06    bcs $c0af      gefunden, fertig
.; c0a9 20 d0 c0 jsr $c0d0      nicht gef. Pointer erhöhen
.; c0ac 4c 66 c0 jmp $c066      weitersuchen

.; c0af 20 f9 c0 jsr $c0f9      Integervariable holen
.; c0b2 a9 00    lda #$00       und Position im Array
.; c0b4 a0 00    ldy #$00       hineinkopieren
.; c0b6 91 64    sta ($64),y
.; c0b8 c8       iny
.; c0b9 a5 b1    lda $b1
.; c0bb 91 64    sta ($64),y
.; c0bd 20 f9 c0 jsr $c0f9      Integervariable holen
.; c0c0 a9 00    lda #$00       und Position im String
.; c0c2 a0 00    ldy #$00       hineinkopieren
.; c0c4 91 64    sta ($64),y
.; c0c6 e6 bb    inc $bb
.; c0c8 a5 bb    lda $bb
.; c0ca c8       iny
.; c0cb 91 64    sta ($64),y
.; c0cd 4c f7 ae jmp $aef7      Klammer zu? und raus.

.; c0d0 e6 b1    inc $b1        Zähler erhöhen
.; c0d2 c5 b8    cmp $b8        gleich Anzahl?
.; c0d4 b0 0c    bcs $c0e2      ja, dann fertig aber nicht gef.
.; c0d6 18       clc
.; c0d7 a5 b9    lda $b9        Pointer auf Descriptor um 3
.; c0d9 69 03    adc #$03       erhöhen = nächster Descriptor
.; c0db 85 b9    sta $b9
.; c0dd 90 02    bcc $c0e1
.; c0df e6 ba    inc $ba
.; c0e1 60       rts

.; c0e2 68       pla            gewaltsamer Ausprung, daher
.; c0e3 68       pla            Rücksprungadresse entfernen
.; c0e4 20 f9 c0 jsr $c0f9      Integervariable holen und
.; c0e7 a9 ff    lda #$ff       -1 einkopieren
.; c0e9 a0 00    ldy #$00
.; c0eb 91 64    sta ($64),y
.; c0ed c8       iny
.; c0ee 91 64    sta ($64),y
.; c0f0 20 fd ae jsr $aefd      Komma?
.; c0f3 20 8b b0 jsr $b08b      Variable holen, aber nicht verwenden
.; c0f6 4c f7 ae jmp $aef7      Klammer zu? und raus.
.; c0f9 20 fd ae jsr $aefd      Komma?
.; c0fc 20 8b b0 jsr $b08b      Variable holen/anlegen
.; c0ff 85 64    sta $64        Pointer zwischenspeichern
.; c101 84 65    sty $65
.; c103 a5 0d    lda $0d        Check auf Integer
.; c105 d0 05    bne $c10c
.; c107 a5 0e    lda $0e
.; c109 f0 01    beq $c10c
.; c10b 60       rts
.; c10c 4c 99 ad jmp $ad99      Type Mismatch Error und raus

.; c10f a5 f9    lda $f9        Tabelle anlegen
.; c111 85 ff    sta $ff        Länge des Suchstr.
.; c113 a2 00    ldx #$00
.; c115 9d 00 c2 sta $c200,x    Tabelle vorbelegen
.; c118 e8       inx
.; c119 d0 fa    bne $c115
.; c11b a0 ff    ldy #$ff
.; c11d c8       iny
.; c11e c6 ff    dec $ff
.; c120 b1 f7    lda ($f7),y
.; c122 aa       tax
.; c123 a5 ff    lda $ff        Stellungsnummern
.; c125 9d 00 c2 sta $c200,x    in Tabelle
.; c128 d0 f3    bne $c11d
.; c12a 60       rts

.; c12b a5 b0    lda $b0        Startposition in
.; c12d 85 fd    sta $fd        Positionsspeicher
.; c12f a5 f9    lda $f9        Länge Suchstr.
.; c131 85 fe    sta $fe        in Arbeitsspeicher
.; c133 a9 ff    lda #$ff       Suchtiefe (wird gleich $00)
.; c135 85 02    sta $02
.; c137 c6 fe    dec $fe
.; c139 e6 02    inc $02
.; c13b a5 fd    lda $fd        Anlegeposition des letzten
.; c13d 18       clc            Buchstaben berechnen
.; c13e 65 fe    adc $fe
.; c140 a8       tay
.; c141 b1 fb    lda ($fb),y    aus Durchsuchstr. holen
.; c143 85 ff    sta $ff        zwischenspeichern
.; c145 a4 fe    ldy $fe        Position im Suchstr.
.; c147 b1 f7    lda ($f7),y
.; c149 c9 3f    cmp #$3f       = Joker ?
.; c14b f0 04    beq $c151
.; c14d c5 ff    cmp $ff        Buchstaben gleich?
.; c14f d0 06    bne $c157
.; c151 a5 fe    lda $fe        komplett durchsucht?
.; c153 f0 23    beq $c178
.; c155 d0 e0    bne $c137
.; c157 a6 ff    ldx $ff        neue Anlegeposition berechnen
.; c159 bd 00 c2 lda $c200,x    max. Verschiebewert aus Tabelle
.; c15c 38       sec
.; c15d e5 02    sbc $02        Suchtiefe abziehen
.; c15f cd 3f c2 cmp $c23f      größer als Jokerverschiebewert?
.; c162 90 03    bcc $c167
.; c164 ad 3f c2 lda $c23f
.; c167 18       clc
.; c168 65 fd    adc $fd        Verschiebewert zu Position addieren
.; c16a 85 fd    sta $fd
.; c16c 65 f9    adc $f9        Test auf Überlauf
.; c16e b0 06    bcs $c176
.; c170 c5 fa    cmp $fa        fertig mit durchsuchen?
.; c172 90 bb    bcc $c12f      nein
.; c174 f0 b9    beq $c12f      nein
.; c176 18       clc            Flag für nicht gefunden
.; c177 24 38    bit $38        Maskierung für sec, Flag für gef.
.; c179 60       rts            und zurück
```

Listing 2. Der Quellcode von unserem Suchprogramm »Intellisearch«

Suche abgebrochen, das heißt Suchstring nicht gefunden, steht in X% -1, in Y% bleibt der alte Wert erhalten.

Und so geht's

Im folgenden werden wir uns noch ein wenig mit der Funktionsweise des Programms beschäftigen. Dazu dienen uns Bild 3, das ein Flußdiagramm der Kontrollroutine zeigt, sowie Listing 2, und Tabelle 1, die alle verwendeten Zeropage-Adressen aufschlüsselt. Ich weiß leider nicht, ob nach der Benutzung ein einwandfreier Kassettenbetrieb möglich ist, da einige der Kassetten-Zeropage-Adressen verwendet werden. Hier wäre ich für Hinweise dankbar.

Als erstes wird der Suchstring geholt, und auf Füllsel untersucht. Bei der Verwendung dieser Routine ($AD9A) kann es unter Umständen bei manchen C 64 zu einem FORMULAR TOO COMPLEX ERROR kommen. Ist ein solches Füllsel vorhanden, erfolgt die Teilung in zwei Teilstrings. Die Länge des Teilstrings 2 ist später gleichzeitig

Fortsetzung auf Seite 160

spann geladen worden ist, verzweigt der Computer über den INPUT-Vektor in das sich im Bandpuffer befindliche Maschinenprogramm, das wiederum einen neuen Ladebefehl erzeugt, mit dem das eigentliche Programm nachgeladen und automatisch gestartet wird.

Dazu jetzt die einzelnen Schritte, mit denen man einen solchen Vorspann generieren kann:

1. Eingabe des Ladeprogramms (Listing 3)
2. Programm testen und abspeichern
3. Lader starten
4. Programmnamen eingeben
5. Die auf dem Bildschirm ausgedruckten Zeilen nacheinander mit RETURN ausführen
6. Vorspann abspeichern
7. Wenn auf dem Bildschirm »SEARCHING« erscheint, muß der beginnende Ladevorgang mit der STOP-Taste abgebrochen werden
8. Hauptprogramm nachladen und abspeichern

Listing 4 zeigt das kleine Maschinenprogramm aus den DATA-Zeilen 360 bis 380 des Basic-Laders im Assembler-Format. Wer den C 64 dazu bringen will, das Gleiche zu tun, den möchte ich auf die REM-Zeilen am Ende des Programms aufmerksam machen, in denen die nötigen Änderungen beschrieben sind.

Zwei Anmerkungen wären zu dieser Methode noch zu machen:

Erstens: Die demonstrierte Autostartroutine funktioniert nur im Kassettenbetrieb. Die Floppystation nutzt diesen Puffer nicht, folglich funktioniert auch diese Methode nicht.

Zweitens: Da das Maschinenprogramm an einen festen Speicherplatz im Kassettenpuffer gebunden ist, muß auch der Filename eine feste Länge haben, denn an diesen schließt sich die Maschinenroutine an. Daher ergänzt der Lader den Programmnamen auf 16 Zeichen.

Mit diesem Ausflug in Betriebssystem und Interruptmechanismus ist der VC 20, so meine ich, genug durchleuchtet worden. Ich hoffe, daß Sie in den sechs Folgen dieses Kurses einen Eindruck von den Geschehnissen im Inneren Ihres Computers bekommen haben, so daß dieser Kurs seinen Namen auch wirklich verdient hat.

(Christoph Sauer/ev)

PS. Uns interessiert Ihre Meinung zum Thema VC 20 im 64'er-Magazin im allgemeinen und zu diesem Kurs im besonderen. Wie wär's, schreiben Sie uns eine Mitmachkarte?

Fortsetzung von Seite 153

das Kennzeichen dafür, ob ein Füllsel vorlag. Ist die Länge gleich 0, ist das nicht der Fall.

Alsdann wird der Pointer auf den Startdurchsuchstring gespeichert. Dies erfolgt auch über die uns schon bekannte Routine $AD9A. Wenn keine direkte Auswertung vorgenommen wird, steht in $64/$65 ein Zeiger auf den Stringdescriptor, oder den Variablendescriptor. Achtung! Hier wird nicht überprüft, ob es sich überhaupt um einen String oder ein Stringarray handelt. Hier kann man also die Routine zum »Ausflippen« bringen.

Später wird zu diesem Pointer einfach 3 addiert, um den nächsten Stringdescriptor zu bekommen. Dies funktioniert bei Arrays einwandfrei, solange nicht die Array-Obergrenze überschritten wird. Theoretisch ließen sich hier auch mehrdimensionale Arrays durchsuchen. Wer dies vor hat, möge sich nochmal den Aufbau von Stringarrays im Artikel über die Carbage Collection zu Gemüte führen (64'er, Ausgabe 1/85).

Als nächstes werden die zwei numerischen Parameter geholt, beide als Bytewert. Deswegen sind hier auch nur Werte kleiner 256 erlaubt. Mehr ist sowieso kaum sinnvoll.

Nun wird der aktuelle Durchsuchstringdescriptor in den Arbeitsspeicher kopiert, die Tabelle angelegt und die Suche nach Teilstring 1 gestartet. Wird Teilstring 1 gefunden, wird nach Teilstring 2, falls vorhanden, gesucht.

Als Flag für erfolgreiche Suche steht das Carry-Flag. Ist es gesetzt, war die Suche erfolgreich, ansonsten ist das Carry-Flag gelöscht.

Der nächste Programmteil kopiert nach erfolgreicher Suche die Werte in die beiden Integervariablen.

Ein wenig gemein programmmiert ist die »Zähler erhöhen«-Routine. Sie wird zwar mit JSR aufgerufen, aber es erfolgt kein Rücksprung, wenn die Anzahl der zu durchsuchenden Strings erreicht wird. Dann wird die Rücksprungadresse vom Stack entfernt, und nach dem Einkopieren von —1 in die erste der beiden Integervariablen die Routine verlassen.

Die beiden letzten Teile des Programms sind das Tabellenanlegen und die eigentliche Suchroutine. Hier verweise ich auf die beiden Flußdiagramme und die obigen Erklärungen.

So, damit hätten wir uns wohl durch das Programm durchgekämpft.

Noch einige Tips: Das Füllselzeichen kann in der Speicherstelle $C01E geändert werden. Beim Jokerzeichen sind einige Änderungen mehr notwendig. Diese betreffen die Speicherstellen $C14A, $C160 und $C165. Dort muß dann jeweils der ASCII-Code des Jokers stehen.

Das nächste Mal geht's wieder um Strings, dann allerdings um Sortieren. Karsten Schramm wird dann die gängigsten Sortieralgorithmen vorstellen und vergleichen.

(B. Schneider/gk)

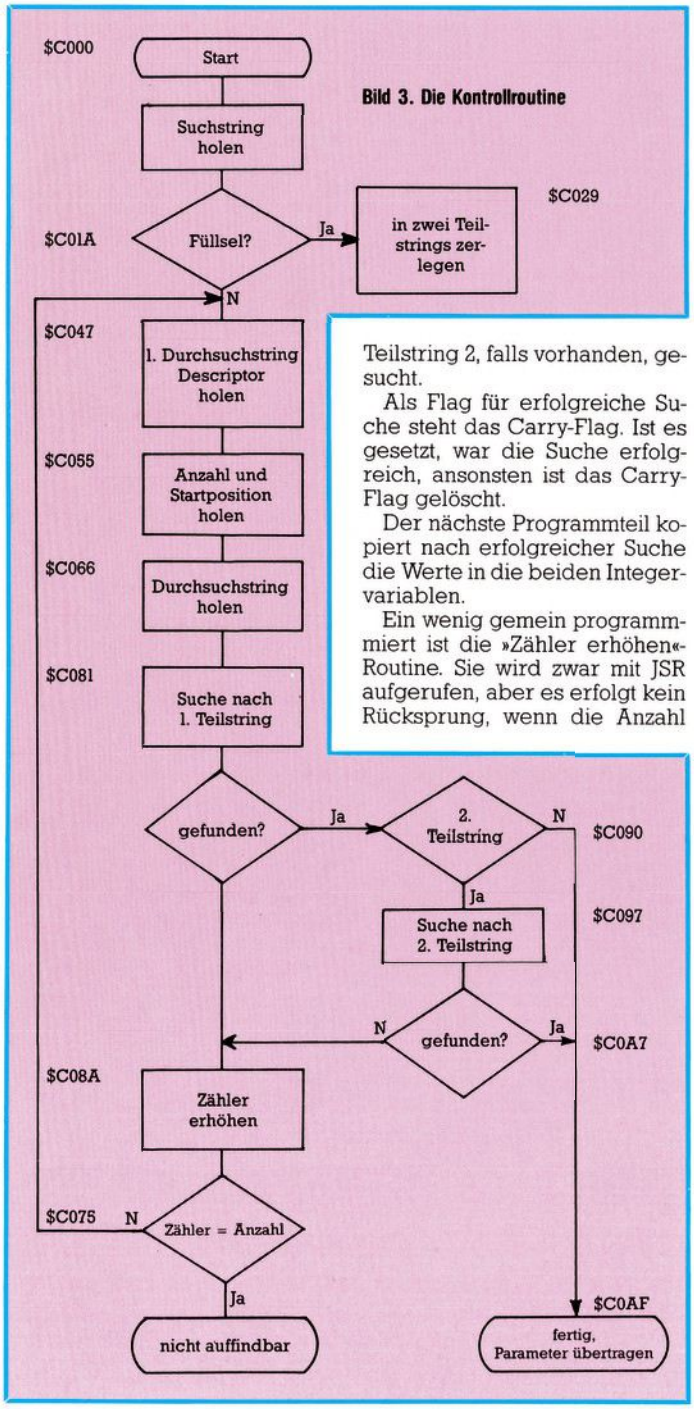

Bild 3. Die Kontrollroutine

$02	Zwischenspeicher für Füllselposition/Suchtiefe
$b0	Startposition der Suche
$b1	Position im Array
$b2	Low-Byte Adresse Teilstr. 1
$b3	High-Byte Adresse
$b4	Länge Teilstr. 1
$b5	Low-Byte Adresse Teilstr. 2
$b6	High-Byte
$b7	Länge Teilstr. 2
$b8	Anzahl der zu durchsuchenden Strings
$b9	Pointer auf aktuellen Descr. Low-Byte
$ba	Pointer High-Byte
$f7	Low-Byte Adr. Suchstring
$f8	High-Byte
$f9	Länge Suchstring
$fa	Länge Durchsuchstring
$fb	Adr, Low-Byte Durchsuchstring
$fc	High-Byte
$fd	Anlegepostion des ersten Buchstaben
$fe	Position des akt. Buchstaben im Suchstring
$ff	Zwischenspeicher Stellungsnummer/Buchstabe

Tabelle 1. Diese Zeropageadresse benutzt Intellisearch