Der gläserne VC 20

Teil 6

Als Abschluß unseres Kurses beschäftigen wir uns mit Interruptmechanismus und Betriebssystem des Volkscomputers. Damit haben wir in den sechs Teilen den VC 20 aus jeder Perspektive durchleuchtet und wirklich zum »gläsernen« Computer gemacht.

Ein Betriebssystem, so sagt ja bereits der Name, ist für die elementaren Funktionen des Computers wie zum Beispiel Bildschirmverwaltung, Tastaturabfrage, laden und speichern von Daten und vielem mehr verantwortlich. Natürlich ist auch im VC 20 solch ein Systemprogramm eingebaut: Es befindet sich im ROM zwischen Adresse $E429 und $FFFF, der restliche ROM-Bereich (von $C000 bis $E428) wird vom Basic-Interpreter beansprucht. Diese zwei Programmblöcke sind bei diesem Computer so miteinander verzahnt, daß kein Teil für sich alleine arbeiten kann.

Was beim Einschalten alles geschieht

Als erstes möchte ich nun die Vorgänge beschreiben, die sich beim Einschalten des Systems abspielen.

Zunächst wird die Resetleitung der CPU von der Elektronik des Computers auf Low gezogen (genau das Gleiche geschieht übrigens beim Betätigen eines eventuell eingebauten Reset-Tasters), wodurch die wichtigsten Systemkomponenten (CPU, VIC und die Ein-/ Ausgabebausteine) in einen definierten Betriebszustand versetzt werden.

Danach sucht sich die 6502-Zentraleinheit aus den Speicherstellen $FFFC und $FFFD die Startadresse des Betriebssystems heraus und beginnt die Programmausführung an der dort abgespeicherten Stelle (beim VC 20 $FD22). Diese Vektoren sind hardwaremäßig in der CPU festgelegt worden und können daher nicht durch ein Programm geändert werden.

Die Einschaltroutine hat nun die verschiedensten Aufgaben zu bewältigen. Am wichtigsten ist zunächst die Initialisierung des Prozessorstacks, also des Stapelspeichers, der die Rücksprungadressen beim Befehl JSR verwaltet (damit der Computer aus den Unterprogrammen wieder ins Hauptprogramm zurückfindet).

Der nächste Schritt in der Reset-Routine ist die Modulabfrage, in der die Modulidentifikation abgefragt wird. Findet der VC 20 im Bereich $A004 bis $A009 die Zeichenfolge »a0CBM« (die den Bytes 41 30 C3 C2 und CD entspricht), so gibt das Betriebssystem die Kontrolle an das Modulprogramm ab. Auch hierbei spielen wieder Vektoren ein Rolle, denn die Startadresse des Moduls muß in den beiden Speicherstellen $A000 und $A001 abgelegt sein.

Mit Hilfe eines solchen Modulprogramms, das automatisch gestartet wird, kann man nun den Computer ganz nach seinen Wünschen gestalten. Dies kann zum Beispiel in Form einer Basic-Erweiterung geschehen, wie sie an dieser Stelle ja schon besprochen worden ist.

Findet die Einschaltroutine — um wieder zum Reset zurückzukommen — keine Modulmarkierung, so wird die Initialisierung ganz normal fortgesetzt. Diese besteht im wesentlichen aus vier Unterprogrammen, die auch in einem Autostartprogramm nacheinander aufgerufen werden sollten. Da ist als erstes eine Routine ($FD8D), die die Aufgabe hat, in der Zeropage bestimmte Startwerte einzutragen und den verfügbaren Speicher festzustellen. Dieser RAM-Test ist für den VC 20 besonders wichtig, denn auf diese Weise kann er erstens die Funktionsweise des Speichers überprüfen und zweitens die momentane Ausbauversion feststellen. Wird tatsächlich einmal ein Defekt in einer oder mehreren Speicherstellen des Grundversions-RAMs festgestellt, dann »hängt« sich das Betriebssystem in einer Endlosschleife auf, und der Bildschirm bleibt dunkel.

Nach dieser Prüfung, die bei voll ausgebautem Speicher einige Sekunden in Anspruch nimmt, werden in einer weiteren Unterroutine ($FD52) die Kernal-Vektoren im Bereich zwischen $0314 und $0355 installiert (einige dieser Zeiger werden wir später noch genauer kennenlernen). Zum Schluß müssen natürlich auch in den Registern der Peripheriebausteine die benötigten Werte abgespeichert werden. Dafür sind die zwei letzten Unterprogramme der Einschaltroutine zuständig. Das eine ($FDF9) initialisiert die I/O-Register, damit Interrupts und Tastaturoperationen stattfinden können, und das andere ($E518) kümmert sich um die Grundeinstellung des VIC.

Dies waren im wesentlichen die Schritte, die nach dem Einschalten des Computers ablaufen; damit ist nun das Betriebssystem einsatzbereit und gibt nun die Kontrolle an den Basic-Interpreter ($E378) ab. Der wiederum setzt als erstes die in Folge 3 angesprochenen Basic-Vektoren, damit die essentiellen Operationen (wie zum Beispiel die Umwandlung einer Zeile in Interpretertoken) über diese abgewickelt werden können.

Bevor die bekannte Einschaltmeldung (CBM BASIC...) auf dem Bildschirm ausgegeben wird, initialisiert der Interpreter den Basic-Teil der Zeropage und des Speichers nach seinen Bedürfnissen. Dabei werden zum Beispiel verschiedene Zeiger (Adresse 43, 44, 45, ...) gesetzt und der erste Verbindungszeiger im Programmspeicher gelöscht (daher kann ein Basic-Programm nach einem Reset nicht mehr gelistet werden).

Unterbrechung? Ja, bitte!

Nachdem die Systemmeldung auf dem Bildschirm ausgegeben wurde, verzweigt die Interpreterroutine in die sogenannte Eingabewarteschleife ($C474), die wir ja bereits in Folge 3 näher beleuchtet haben. Damit ist der Computer bereit, Anweisungen des Benutzers zu empfangen.

Soweit die Beschreibung der Resetroutine, die durch ein Signal an einem Prozessoreingang gestartet wurde. Auch die Interrupts, auf die ich genauer eingehen möchte, werden über die Hardware ausgelöst. Der Plural bei Interrupts verrät bereits, daß es mehrere — genauer gesagt sogar 3 — Unterbrechungen von dieser Sorte gibt.

Da ist zunächst der Systeminterrupt, der vom VC 20 alle 60stel Sekunde durchlaufen wird (ein Timer in dem 6522-Ausgabebaustein ist für dessen Auslösung verantwortlich). Durch diese Interruptanfrage — sie wird auch IRQ (Interrupt Request) genannt — wird die Zentraleinheit veranlaßt, das gerade laufende Programm zu unterbrechen, um in eine Abarbeitungsroutine zu verzweigen. Auch hierbei besorgt sich die CPU über den Hardwarevektor ($FFFE und $FFFF) die entsprechende Startadresse ($FF72).

Dort werden als erstes alle CPU-Register auf den Stack gerettet, denn die dort gespeicherten Werte werden ja später bei der Wiederaufnahme des regulären Programmablaufs benötigt. Danach verzweigt die Routine über einen eigenen Vektor (Interrupt-Vektor $0314,$0315) in das eigentliche Abarbeitungsunterprogramm (Adresse $EABF).

Hier wird zunächst die interne Uhr (TI) um eins hochgezählt und bei dieser Gelegenheit auch gleich der Zustand der STOP-Taste abgefragt. Ist diese gedrückt, dann setzt die Unterroutine ein Flag in der Zeropage, und der Interpreter unterbricht — nachdem der Interrupt zu Ende bearbeitet worden ist — das laufende Basic-Programm. Durch »verbiegen« des erwähnten Sprungvektors kann dieses Unterprogramm in der Abarbeitungsroutine so übersprungen werden, daß keine Abfrage der STOP-Taste mehr stattfindet, womit bewirkt wird, daß man ein Basic-Programm nicht mehr auf normalem Wege anhalten kann (dieses nur als eine Programmschutzvariante).

Nun aber wieder zurück zu unserer Abarbeitungsroutine. Hier wird als nächstes der sogenannte Cursorblinkzähler, der bei jedem Interrupt um eins dekrementiert wird, abgefragt. Ist dieser auf Null, so wird der momentane Zustand des Cursors geändert; das heißt aus revers wird normal oder umgekehrt (auf diese Weise wird ein Blinken des Zeigers erreicht).

Abschließend wird — und das ist das Wichtigste an dem ganzen Interrupt — die Tastatur des Computers abgefragt. Drückt der Benutzer irgendeine Taste, so wandelt die Routine diesen Tastencode in den entsprechenden ASCII-Wert und speichert das Ergebnis im Tastaturpuffer ab. Dieser bleibt solange dort, bis ein Zeichen von der Tastatur angefordert wird. Das ist entweder im Direktmodus der Fall oder bei den Basic-Befehlen INPUT und GET (oder den entsprechenden Unterprogrammen im Betriebssystem).

Damit hat die Interrupt-Routine ihren Zweck erfüllt, und der VC 20 kann die Arbeit an dem eigentlichen (Basic-) Programm wieder aufnehmen. Vorher werden allerdings noch die CPU-Register wiederhergestellt (deren Inhalte befinden sich ja immer noch auf dem Stack) und dann gibt das Interrupt-Unterprogramm über den Befehl RTI (Return From Interrupt) die Kontrolle wieder an das eigentliche Hauptprogramm ab. Diese Einschnitte mittels Interrupt haben für den Benutzer den Vorteil, daß er sich nicht um die, in einem gewissen Zeitabstand immer wiederkehrenden, Tastaturabfragen zu kümmern braucht. Außerdem kann kein noch so kurzer Tastendruck verloren gehen, wie es der Fall wäre, wenn die Tastatur in unregelmäßigen, längeren Abständen abgefragt werden würde.

Eingriff ins Eingemachte

Wie wir bereits gesehen haben, springt der Computer über einen Vektor ($0314-$0315) in die Bearbeitungsroutine ein. Dadurch können wir jetzt in den Ablauf eingreifen, indem wir dort zusätzliche Unterprogramme einbetten. Als Beispiel für solch einen Eingriff möchte ich nun eine Routine besprechen, die den regelmäßigen Interruptzyklus benützt, um die aktuelle Uhrzeit auf dem Bildschirm anzuzeigen. Dazu bedienen wir uns einer speziellen Eigenschaft des VIC, die es erlaubt, mehr als 23 Zeilen auf dem Bildschirm anzuzeigen. Mit Hilfe des Registers #4 (Adresse 36867 beziehungsweise $9003) kann man die anzeigbare Zeilenzahl vergrößern oder verkleinern. Für unseren Zweck genügt es, die verfügbaren Zeilen auf 25 zu erhöhen. In diesen Zeilen wird dann später die Uhrzeit angezeigt werden. Mittels POKE 36867,50 wird diese Umschaltung realisiert. Meine Routine kann mit dem Lader (Listing 1) irgendwo im Speicher plaziert werden (das Programm gibt die günstigste Adresse vor). Den wichtigsten Teil der Initialisierungsroutine — also dem Unterprogramm, das den Interrupt-Vektor »verbiegt« — möchte ich kurz anhand von Listing 2 beschreiben. Dort wird gleich als erstes eine Unterroutine aufge-

Listing 1. »IRQ-Clock« (Basic-Lader)

```
100 REM ************************ <227>
110 REM * IRQ-CLOCK BY C.SAUER * <098>
120 REM ************************ <247>
130 REM                          <017>
140 REM                          <027>
150 REM ######################## <074>
160 REM ## NUR FUER SPEICHER ## <053>
170 REM ## GROESSER 8 KBYTE  ## <230>
180 REM ######################## <104>
190 FOR T=0 TO 279:READ D:S=S+D:NEXT
                 <252>
200 IF S<>32965 THEN PRINT"FEHLER
    IN DATAS":END                <011>
210 PRINT"{CLR,2DOWN}WO SOLL DAS PROG
   RAMM"                         <129>
220 PRINT"{DOWN}ABGESPEICHERT WERDEN?"
                 <132>
230 V=PEEK(56)*256-512           <248>
240 PRINT"{DOWN}VORSCHLAG: "V;   <082>
250 INPUT"{8LEFT}";V             <060>
260 W2=INT(V/256):W1=V-W2*256    <045>
270 POKE 55,W1:POKE 56,W2        <181>
280 RESTORE                      <165>
290 FOR T=V TO V+279             <070>
300 READ D                       <248>
310 IF D<0 THEN GOSUB 330        <179>
320 POKE T,D:NEXT:SYS V:END      <064>
330 ON ABS(D)GOTO 340,350,360,370,380
                 <093>
340 D=W2:RETURN                  <156>
350 D=W2+1:RETURN                <129>
360 D=W1+208:RETURN              <243>
370 D=W1+60:RETURN               <201>
380 D=W1+10:RETURN               <206>
390 DATA 032,-03,-01,173,003     <155>
400 DATA 144,041,128,009,050     <187>
410 DATA 141,003,144,162,044     <193>
420 DATA 169,160,157,249,017     <227>
430 DATA 173,015,144,041,007     <216>
440 DATA 157,249,149,202,208     <244>
450 DATA 240,162,008,173,134     <240>
460 DATA 002,157,022,150,202     <237>
470 DATA 208,250,120,169,-04     <255>
480 DATA 141,020,003,169,-01     <253>
490 DATA 141,021,003,088,032     <015>
500 DATA 089,198,076,116,196     <064>
510 DATA 162,005,181,097,157     <055>
520 DATA 073,003,202,208,248     <054>
530 DATA 162,011,189,254,000     <063>
540 DATA 157,084,003,202,208     <075>
550 DATA 247,165,113,141,097     <095>
560 DATA 003,165,071,141,098     <099>
570 DATA 003,165,093,141,099     <114>
580 DATA 003,165,094,141,100     <108>
590 DATA 003,165,162,166,161     <128>
600 DATA 164,160,032,135,207     <134>
610 DATA 132,094,136,132,113     <143>
620 DATA 160,006,132,093,160     <151>
630 DATA 036,032,104,222,169     <164>
640 DATA 023,133,169,169,018     <186>
650 DATA 133,170,160,000,162     <174>
660 DATA 002,185,255,000,145     <191>
670 DATA 169,200,192,006,240     <205>
680 DATA 018,202,208,243,169     <221>
690 DATA 058,145,169,136,230     <237>
700 DATA 169,208,002,230,170     <234>
710 DATA 162,003,208,233,162     <242>
720 DATA 005,189,073,003,149     <007>
730 DATA 097,202,208,248,162     <020>
740 DATA 011,189,084,003,157     <025>
750 DATA 254,000,202,208,247     <025>
760 DATA 173,097,003,133,113     <039>
770 DATA 173,098,003,133,071     <054>
780 DATA 173,099,003,133,093     <069>
790 DATA 173,100,003,133,094     <063>
800 DATA 076,191,234,166,045     <061>
810 DATA 134,090,164,046,132     <092>
820 DATA 091,032,-05,-02,134     <082>
830 DATA 088,134,045,132,089     <124>
840 DATA 132,046,166,043,134     <122>
850 DATA 095,164,044,132,096     <142>
860 DATA 032,-05,-02,134,043     <119>
870 DATA 134,166,132,044,132     <149>
880 DATA 167,032,184,195,032     <170>
890 DATA 051,197,165,166,208     <186>
900 DATA 002,198,167,198,166     <203>
910 DATA 169,000,168,145,166     <202>
920 DATA 096,132,168,024,138     <212>
930 DATA 105,049,170,165,168     <222>
940 DATA 105,000,168,096,000     <214>
```

Listing 2. »IRQ-Clock« (Assembler-Listing)

```
****************** INITALISIERUNG
****************** DER UHRENROUTINE
2000  JSR $20D0   ; BASICPGM. VERSCIEBEN
2003  LDA $9003   ; BILDSCHIRMGROESSE
2006  AND #$80    ; LADEN UND 24,25.
2008  ORA #$32    ; ZEILE ZUSCHALTEN.
200A  STA $9003   ; ABSPEICHERN
200D  LDX #$2C    ; ZEILEN LOESCHEN
200F  LDA #$A0    ; MIT INVERSEN SPACES
2011  STA $11F9,X; ABSPEICHERN
2014  LDA $900F   ; AKTUELLE FARBE
2017  AND #$07    ; WIRD ZUR
2019  STA $95F9,X; ZEICHENFARBE
201C  DEX         ; SCHLEIFE
201D  BNE $200F
201F  LDX #$08    ; FARBE DER UHRZEIT
2021  LDA $0286   ; IST AKTUELLE ZEICH-
2024  STA $9616,X; CHENFARBE
2027  DEX
2028  BNE $2024
202A  SEI         ; INTERRUPTS VERHIND.
202B  LDA #$3C    ; VEKTOREN INITIAL.
202D  STA $0314
2030  LDA #$20
2032  STA $0315
2035  CLI         ; INTERRUPTS ZULASSEN
2036  JSR $C659   ; BASIC-BEFEHL CLR
2039  JMP $C474   ; INS BASIC ZURUECK
****************** UHREN ANZEIGE
203C  LDX #$05    ; DIVERSE ZEROPAGE-
203E  LDA $61,X   ; DATEN IN DEN BAND-
2040  STA $0349,X; PUFFER RETTEN
2043  DEX
2044  BNE $203E
2046  LDX #$0B
2048  LDA $00FE,X
204B  STA $0354,X
204E  DEX
204F  BNE $2048
2051  LDA $71
2053  STA $0361
2056  LDA $47
2058  STA $0362
205B  LDA $5D
205D  STA $0363
2060  LDA $5E
2062  STA $0364
2065  LDA $A2     ; UHRZEIT (A2,A1,A0)
2067  LDX $A1     ; HOLEN
2069  LDY $A0
206B  JSR $CF87   ; IN STRING VERWANDELN
206E  STY $5E     ; UNTERROUTINE VOR-
2070  DEY         ; BEREITEN.
2071  STY $71
2073  LDY #$06    ; 6 ZIFFERN DARSTELLEN
2075  STY $5D
2077  LDY #$24
2079  JSR $DE68   ; ZIFFERN BERECHNEN
207C  LDA #$17  ; BILDSCHIRMSPEICHER-
207E  STA $A9     ; ADRESSE DER ZIFFERN
2080  LDA #$12
2082  STA $AA
2084  LDY #$00    ; ZAEHLER FUER ZIFFERN
2086  LDX #$02    ; ZAEHLER FUER ':'
2088  LDA $00FF,X; ZIFFERN HOLEN
208B  STA ($A9),X; IN BILDSCHIRMSPEICHER
208D  INY         ; SCHLEIFE 1
208E  CPY #$06    ; ALLE ZIFFERN ?
2090  BEQ $20A4   ; JA, DANN ENDE
2092  DEX
2093  BNE $2088
2095  LDA #$3A    ; DOPPELPUNKT LADEN
2097  STA ($A9),Y; UND ABSPEICHERN
2099  DEY         ; SCHLEIFE 2
209A  INC $A9     ; ZAHLER KORREKTUR
209C  BNE $20A0
209E  INC $AA     ; ZAHLER KORREKTUR
20A0  LDX #$03    ; NEUER ':' ZAHLER
20A2  BNE $208D
20A4  LDX #$05    ; ALLE GERETTETEN
20A6  LDA $0349,X; ZEROPAGE VARIABLEN
20A9  STA $61,X   ; WIEDER ZURUECK-
20AB  DEX         ; SPEICHERN.
20AC  BNE $20A6
20AE  LDX #$0B
20B0  LDA $0354,X
20B3  STA $00FE,X
20B6  DEX
20B7  BNE $20B0
20B9  LDA $0361
20BC  STA $71
20BE  LDA $0362
20C1  STA $47
20C3  LDA $0363
20C6  STA $5D
20C8  LDA $0364
20CB  STA $5E
20CD  JMP $EABF   ; ZUR INTERRUPTROUTINE
****************** BASICPROGRAMM VER-
****************** SCHIEBEN
20D0  LDX $2D     ; ENEADRESSE DES PGMS
20D2  STX $5A     ; HOLEN
20D4  LDY $2E
20D6  STY $5B
20D8  JSR $210A   ; 49 ADDIEREN
20DB  STX $58     ; UND ABSPEICHERN
20DD  STX $2D
20DF  STY $59
20E1  STY $2E
20E3  LDX $2B     ; PGM. ANFANGSADRESSE
20E5  STX $5F     ; HOLEN UND ABSP.
20E7  LDY $2C
20E9  STY $60
20EB  JSR $210A   ; 49 ADDIEREN
20EE  STX $2B     ; UND ABSPEICHERN
20F0  STX $A6
20F2  STY $2C
20F4  STY $A7
20F6  JSR $C3B8   ; PROGRAMM VERSCHIEBEN
20F9  JSR $C533   ; PGM.ZEILEN BINDEN
20FC  LDA $A6
20FE  BNE $2102
2100  DEC $A7     ; 0 AN DEN NEUEN PGM.-
2102  DEC $A6     ; ANFANG SPEICHERN
2104  LDA #$00
2106  TAY
2107  STA ($A6),
2109  RTS
****************** ADDITIONSROUTINE
210A  STY $A8
210C  CLC         ; 16-BIT ADDITION
210D  TXA         ; VORBEREITEN
210E  ADC #$31    ; DEZ 49 ADDIEREN
2110  TAX
2111  LDA $A8     ; HIGH-BYTE LADEN
2113  ADC #$00    ; UEBERTRAG ADDIEREN
2115  TAY
2116  RTS
```

rufen, die ein eventuell vorhandenes Basic-Programm im Speicher verschieben soll. Das hat seinen guten Grund, denn die zwei zusätzlichen Bildschirmzeilen nehmen, um ihre Zeichen ablegen zu können, den Anfang des Basic-Speichers in Beschlag, so daß ein dort stehendes Programm teilweise überschrieben werden würde. Diese Verschiebe-Routine ist so universell, daß sie auch für andere Anwendungen nützlich sein kann (um beispielsweise ein Basic-Programm zu verschieben, damit Platz für einen Grafikspeicher ist). Die Anzahl der zu verschiebenden Bytes ist, bei einer anderweitigen Verwendung, in die Additionsroutine in $210E einzusetzen.

Nachdem die 24. und 25. Zeile zugeschaltet worden ist, wird der Interruptvektor auf den Anfang unserer Uhrenroutine gestellt, damit sie beim nächsten Aufruf mit abgearbeitet wird. Bevor man diesen Zeiger jedoch verändert, muß man mittels des Befehls SEI dafür sorgen, daß zeitweise keine Interrupts mehr akzeptiert werden. Sonst könnte es passieren, daß die — zufällig gerade zu dieser Zeit ausgelöste — Interruptroutine ihren Vektor sucht, und wir sind gerade dabei, ihn zu verändern, wodurch der Computer eine falsche Adresse finden und damit abstürzen würde. Das ist auch der Grund, warum man niemals versuchen darf, Interrupt-Vektoren von Basic aus mit POKE-Befehlen zu verändern.

Der Maschinenbefehl SEI (Set Interrupt-disable Flag) setzt das entsprechende Flag im Statusregister der CPU. Wird nun ein IRQ ausgelöst, so testet die Zentraleinheit zuerst dieses Flag; ist es gesetzt, dann wird die Interruptanfrage ignoriert. Das Gegenstück dazu ist der Befehl CLI, der dieses Flag löscht, damit Interrupts wieder möglich werden.

Interrupt auf Tastendruck

Ich habe ja eingangs bereits erwähnt, daß es auch andere Interrupts gibt — diesen wollen wir uns jetzt zuwenden. Wer wird es glauben, auch die RESTORE-Taste löst einen Interrupt aus. Über den Hardwarevektor in $FFFA und $FFFB bezieht die CPU die Anfangsadresse der NMI-Abarbeitungsroutine (NMI bedeutet »nicht maskierbarer Interrupt«). Dieser Interrupt unterscheidet sich von dem IRQ in der Weise, daß er nicht per Spezialbefehl abschaltbar ist. Die Abarbeitungsroutine ($FEAD), die wiederum über einen Vektor in $0318, $0319 angesprungen wird, prüft nun als erstes, ob sie etwa ein Modulprogramm unterbrochen hat. Ist dies der Fall, so verzweigt die Routine in die NMI-Bearbeitung des Modulprogramms (dazu ist in $A002, $A003 die entsprechende Startadresse anzugeben). Dadurch ist es für den Programmierer eines solchen Moduls möglich, alle Unterbrechungen, die durch das Drücken der RESTORE-Taste entstehen, abzublocken, womit natürlich auch ein gewisser Programmschutz gewährleistet ist.

Sollte ein solches Modulprogramm nicht vorhanden sein, so wird die NMI-Routine normal fortgesetzt. Als erstes wird dann die STOP-Taste des Computers abgefragt, denn ein Programmabbruch soll ja nur durch das Drücken der Tastenkombination RUN/STOP-RESTORE möglich sein. Wenn auch diese Klippe genommen ist, beginnt der eigentliche Warmstart des Computers, der die Vektoren neu setzt und die Zeropage ordnet (daher wird auch unser Uhrenprogramm nach dem Drücken dieser Tastenkombination abgeschaltet).

Natürlich kann auch der NMI für unsere Programme mißbraucht werden. Dazu muß nur wieder (wie soll es auch anders sein) der entsprechende Vektor geändert werden, so daß er auf ein eigenes Maschinenprogramm zeigt. In Folge 3 ist eine sehr kurze und effiziente Anwendung der RESTORE-Taste erläutert worden. Damals ging es darum, sich innerhalb von Hochkommas (") von den lästigen Cursor-Steuerzeichen zu befreien. Durch das Drücken der RESTORE-Taste, die dann quasi als zusätzliche Funktionstaste arbeitet, kann man also auch Aktionen auslösen.

Aller guten Dinge sind drei

Der Vollständigkeit halber möchte ich jetzt auch noch den dritten Interrupttyp anführen, der allerdings nicht durch irgendwelche Signale in der Hardware ausgelöst wird. Ich spreche von dem Maschinenbefehl BRK. Dieser wird in der CPU wie ein Interrupt-Aufruf (IRQ) behandelt, und deshalb benutzt er auch den gleichen Hardwarevektor, nämlich $FFFE. Das Betriebssystem im VC 20 hingegen trifft eine Unterscheidung zwischen diesen beiden Interrupttypen, denn der Befehl BRK soll beim VC 20 einen Warmstart bewirken. Auch diese Abarbeitung wird (ich traue mich es schon nicht mehr zu sagen) über einen Vektor ($0316, $0317) geleitet. Damit bietet sich die Möglichkeit, dieses Kommando dazu zu benutzen, ein Unterprogramm über einen Ein-Byte-Befehl (!) aufzurufen. Durch die Angabe der Startadresse in diesem Vektor kann man jedes beliebige Unterprogramm adressieren und mit BRK aufrufen.

Soweit zu den Interrupts. Ich möchte Ihnen zum Schluß noch, da wir gerade beim Betriebssystem sind, einige sehr nützliche Unterprogramme aus dem Interpreter vorstellen. Diese Routinen können nämlich sehr gut in eigenen Maschinenprogrammen Verwendung finden, und dies spart enormen Platz- und Programmieraufwand.

Nützliche Routinen im Betriebssystem

Verschieberoutine ($C3B8): Genau diese wurde in meinem Beispielprogramm verwendet, um das eventuell im Speicher vorhandene Basic-Programm zu verschieben. Die Benutzung dieser Routine ist verhältnismäßig einfach. Die Anfangsadresse des zu verschiebenden Speicherbereichs muß in den beiden Zeropagespeicherstellen $5F und $60 abgelegt werden. Um das Ende zu markieren, vermerkt man die entsprechende Adresse plus 1 (!) in den Speicherstellen $5A und $5B. Schließlich ist auch noch die neue Anfangsadresse in $58 und $59 abzulegen. Auch hierbei ist darauf zu achten, daß eins dazuaddiert wird, denn die Routine ist dementsprechend konzipiert. Der Aufruf der Routine geschieht schließlich mit »JSR $C3B8«.

Fehlermeldung ausgeben ($C437): Oftmals ist es nötig, eine Basic-Fehlermeldung auszugeben (zum Beispiel in einer Befehlserweiterung, in der ein Komma abzufragen ist). Wenn dieses nicht vorhanden ist oder sonst ein Fehler vorliegt, dann soll eine Fehlermeldung ausgedruckt werden. Alles, was man bei dieser Routine zu tun hat, ist die Eingabe der Fehlernummer ins X-Register. Alle möglichen Meldungen und die dazugehörigen Nummern sind in Tabelle 1 zu finden. Dann wird die Abarbeitungsroutine mit JMP $C437 angesprungen. Zu beachten ist, daß ein Maschinenprogramm auf diese Weise beendet wird (der Interpreter springt nämlich in die Eingabewarteschleife zurück). Möchte man nur eine Meldung ausgeben (ohne das Programm anzuhalten), so ist die Nachricht mit einer anderen Routine, die noch besprochen wird, zu erzeugen. Zwei oft benötigte Fehlermeldungen (Syntax- und Illegal Quantity Error) sind noch einfacher zu bekommen. Um den Syntax Error zu erzeugen springt man mit JMP $CF08 in eine Unterroutine ein, die bereits die Fehlernummer enthält (auf diese Weise spart der Interpreter Speicherplatz). Auch für den Illegal Quantity Er-

Nummer	Fehlermeldung	Adresse
01	TOO MANY FILES OPEN	$C19E
02	FILE OPEN	$C1AD
03	FILE NOT OPEN	$C1B5
04	FILE NOT FOUND	$C1C2
05	DEVICE NOT PRESENT	$C1D0
06	NOT INPUT FILE	$C1E2
07	NOT OUTPUT FILE	$C1F0
08	MISSING FILE NAME	$C1FF
09	ILLEGAL DEVICE NUMBER	$C210
0A	NEXT WITHOUT FOR	$C225
0B	SYNTAX	$C235
0C	RETURN WITHOUT GOSUB	$C23B
0D	OUT OF DATA	$C24F
0E	ILLEGAL QUANTITY	$C25A
0F	OVERFLOW	$C26A
10	OUT OF MEMORY	$C272
11	UNDEF'D STATEMENT	$C27F
12	BAD SUBSCRIPT	$C290
13	REDIM'D ARRAY	$C29D
14	DIVISION BY ZERO	$C2AA
15	ILLEGAL DIRECT	$C2BA
16	TYPE MISMATCH	$C2C8
17	STRING TOO LONG	$C2D5
18	FILE DATA	$C2E4
19	FORMULA TOO COMPLEX	$C2ED
1A	CAN'T CONTINUE	$C300
1B	UNDEF'D FUNCTION	$C30E
1C	VERIFY	$C31E
1D	LOAD	$C324

Tabelle 1. Sämtliche Fehlermeldungen und ihre Adressen

ror gibt es solch ein Unterprogramm: JMP $D248. Für beide Abkürzungen gilt im übrigen — dies noch als Ergänzung — das oben zum Programmabbruch gesagte.

Programmzeilen neu ordnen ($C533): Dies ist eine der praktischsten Routinen, die mir im gesamten Basic-Interpreter untergekommen ist. Das kleine unscheinbare Unterprogramm kann mehr, als zunächst zu vermuten ist. Es ist wahrscheinlich jedem schon einmal passiert, daß plötzlich aus irgendeinem Grund ein verstümmeltes Basic-Programm im Speicher steht. Das kann durch falsches Laden vom Band oder auch durch unvorsichtiges POKEn im Basic-Speicher geschehen. Der Grund für diesen »Zeilensalat« ist eine Veränderung der Verbindungszeiger (Koppeladressen), die am Beginn jeder (Basic-) Programmzeile stehen, und die im Normalfall auf die Adresse der jeweils folgenden Zeile zeigen (siehe auch Folge 1 dieses Kurses). Das Unterprogramm ab $C533 hat nun die nützliche Eigenschaft, fast alle auf diese Weise beschädigten Programme wieder zu restaurieren. Man ruft es einfach von Basic aus mit SYS 53483 auf, und schon ist das Programm zumindest wieder LISTbar (jedenfalls in den meisten Fällen). Damit aber noch nicht genug. Bei dieser Gelegenheit stellt die Routine auch noch die Länge des Programmes fest und übergibt die Endadresse den Speicherstellen $22 und $23. Damit ist es möglich, die durch einen Reset oder durch NEW scheinbar verlorenen Programme völlig wiederherzustellen. Dazu ist lediglich der erste Verbindungszeiger zu rekonstruieren (das macht unsere Unterroutine), und das Programmende muß in den Speicherstellen $2D und $2E vermerkt werden. Die nachfolgende Programmzeile ist das kürzeste Rekonstruktionsprogramm, das mir bekannt ist. Ich habe es bereits in Folge 1 besprochen und möchte es in diesem Zusammenhang noch einmal erwähnen.

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

Bevor diese Zeile eingegeben wird, muß man den Anfang des Basic-Programms im Speicher markieren, denn sonst wird es von der Routine nicht gefunden: POKE (Basic-Anfangsadresse) +1,1.

Natürlich ist diese Routine auch im C 64 zu finden. Dort ist die Startadresse $A533.

String ausgeben ($CB1E): Mit diesem Unterprogramm ist es in einfacher Weise möglich, Texte in einem Maschinenprogramm auszugeben. Dadurch entfallen die sonst nötigen langwierigen Druckschleifen, mit denen man sich sonst herumschlagen muß. Auch bei dieser Unterroutine ist die Handhabung wieder ganz einfach. Der auszugebende Text, der natürlich auch Sonder- und Steuerzeichen enthalten darf, muß im ASCII-Format irgendwo im Speicher stehen. Das Textende muß mit einer Null markiert sein. Ist diese Voraussetzung erfüllt, braucht man nur noch den Akku mit dem Low-Byte, das Y-Register mit dem High-Byte der Anfangsadresse zu laden und das Unterprogramm mit JSR $CB1E aufzurufen: Schon steht der Text auf dem Bildschirm. Auch Fehlermeldungen können auf diese einfache Weise ausgegeben werden (die Möglichkeit hatte ich ja oben schon angekündigt). Die Adressen der einzelnen Fehlernachrichten sind Tabelle 1 zu entnehmen.

Es soll nicht verschwiegen werden, daß es in seltenen Fällen zu einem Problem kommen kann: Weil dieses Unterprogramm auf String-Basis arbeitet, wird der Text als Variable zwischengespeichert. Wurden die Zeiger $37 und $38 (dezimal 55, 56) verändert, so kommt es manchmal dazu, daß irgendwelche unsinnigen Zeichen, aber nicht der gewünschte Text ausgegeben werden. In solch einem Fall müssen die Zeichen konventionell, das heißt über die Kernalroutine $FFD2 ausgegeben werden.

Listing 3. »Autostart« für C 64 / VC 20

```
100 PRINT"{CLR}**********************"      <112>
110 PRINT"* AUTOSTART FUR C 64 *"          <000>
120 PRINT"* UND VC 20 (VERSION *"          <217>
130 PRINT"* FUER VC 20).{7SPACE}*          <021>
140 PRINT"*--------------------*"          <065>
150 PRINT"* DIE ANWEISUNGEN DES*"          <185>
160 PRINT"* PROGRAMMS MUESSEN{2SPACE}*"    <169>
170 PRINT"* GENAU BEFOLGT WER- *"          <105>
180 PRINT"* DEN !!"                        <212>
190 PRINT"**********************"          <055>
200 INPUT"FILENAME:";F$:FOR T=1 TO 16-F
    :F$=F$+" ":NEXT                        <009>
210 FOR T=1 TO 16:POKE 929+T,ASC(MID$(F$,T,1))
    :NEXT                                  <105>
220 FOR T=1 TO 24:READ D:POKE 945+T,D:NEXT <064>
230 PRINT"{CLR,RVSON}FOLGENDE ZEILEN NACH-"
                        <186>
240 PRINT"{RVSON}EINANDER MIT <RETURN>"    <105>
250 PRINT"{RVSON}DURCHGEHEN."              <244>
260 PRINT"{DOWN}POKE43,24:POKE44,3:POKE45,60
    :POKE46,3:CLR"                         <252>
270 PRINT"{2DOWN}FORT=930TO970:F$=F$+CHR$(PEEK(T))
    :NEXT"                                 <052>
280 PRINT"{RVSON,DOWN}** ACHTUNG **"       <203>
290 PRINT"{RVSON}PROGRAMMCASSETTE BE-"     <135>
300 PRINT"{RVSON}REIT HALTEN. UNTER-"      <245>
310 PRINT"{RVSON}BRECHEN NACH BEST-"       <146>
320 PRINT"{RVSON}AETIGUNG DER NAECH-"      <235>
330 PRINT"{RVSON}STEN ZEILE NICHT MIT"     <077>
340 PRINT"{RVSON}<RUN/STOP> !!!"           <139>
350 PRINT"{DOWN}POKE0804,81:POKE805,3
    :SAVEF$,1,1{HOME,DOWN}"                <163>
360 DATA 162,014,142,036,003,162,242,142,037
                        <159>
370 DATA 003,162,009,134,198,189,244,237,157
                        <202>
380 DATA 119,002,202,016,247,096           <112>
1000 REM AENDERUNGEN FUER C 64:            <191>
1002 REM IN ZEILE 360 SIND DIE             <037>
1005 REM DATEN 014 UND 242 DURCH           <117>
1007 REM 087 UND 241 ZU ERSETZEN           <189>
1010 REM IN ZEILE 370 SIND DIE             <046>
1020 REM DATEN 244,237 DURCH               <210>
1030 REM 231,236 ZU ERSETZEN               <021>
```

Listing 4. Die Maschinenroutine zum Autostartprogramm als Assembler-Listing

```
0351 LDX #$0E     ; INPUT-VEKTOR REKON-
0353 STX $0324    ; STRUIEREN.
0356 LDX #$F2
0358 STX $0325
035B LDX #$09     ; NEUN ZEICHEN
035D STX $C6      ; IN TATATURCOUNTER
035F LDA $EDF4,X  ; ZEICHEN AUS ROM HOLEN
0362 STA $0277,X  ; UND IN TASTATURPUFFER
0365 DEX
0366 BPL $035F    ; ALLE NEUNE ?
0368 RTS          ; JA, DANN ENDE
```

Nachlese

Zum Schluß möchte ich nochmals ein Thema aufgreifen, das ich schon einmal angesprochen hatte — ich spreche von dem Autostartprogramm aus Folge 1. Damals wurde ein Programm abgedruckt, das mir viele Leserzuschriften eingebracht hat. Einige berichteten mir, daß es bei ihnen nicht möglich gewesen sei, diese Routine ordnungsgemäß laufen zu lassen. Da dieses Programm bei mir tadellos arbeitet, kann ich mir die geschilderten Fehler nicht erklären. Dennoch möchte ich diese Thematik nochmals aufgreifen, denn es gab eine Reihe von Anfragen nach einer Version für andere Commodore-Computer.

Um möglichst allen Wünschen gerecht zu werden, erläutere ich nun eine neuartige, kürzere Methode, in der auf sämtliche Programmschutzmaßnahmen, wie zum Beispiel das Abschalten der RESTORE-Taste, verzichtet wurde.

Für diese selbststartenden Programme nutzen wir eine besondere Eigenschaft des Kassettenpuffers aus. Der ist nämlich in der Lage, einen Programmnamen mit einer Länge von bis zu 187 Zeichen zu speichern. Davon werden allerdings nur die bekannten 16 Zeichen auf dem Bildschirm angezeigt; der Rest bleibt im Verborgenen. Da der gesamte Inhalt des Kassettenpuffers als Vorspann zu jedem Programm abgespeichert wird, kann man aber tatsächlich einen sehr viel längeren Filenamen mit dem Programm übertragen. Der Trick ist nun folgender: Neben dem Programmnamen mit einer Länge von 16 Zeichen wird auch ein kleines Maschinenprogramm im Bandpuffer generiert. Dieses wird nach der Beendigung des Ladevorganges von dem geänderten Kernal INPUT-Vektor angesprungen (siehe dazu auch Folge 1). Die kleine Maschinenroutine macht nun nichts anderes, als diesen Zeiger auf seinen Ursprungswert zurückzustellen und den Tastaturpuffer mit den Zeichen LOAD (CR) und RUN (CR) zu füllen. Das ist im Prinzip nichts anderes, als wenn man über die Tastatur SHIFT RUN/STOP eingeben würde, nur mit dem Unterschied, daß dies hier automatisch geschieht.

Im einzelnen wird also Folgendes gemacht: Man generiert einen kurzen Vorspann (ohne Programm) auf Band. Dieser enthält lediglich den präparierten Filenamen. Nachdem dieser Vor-

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)

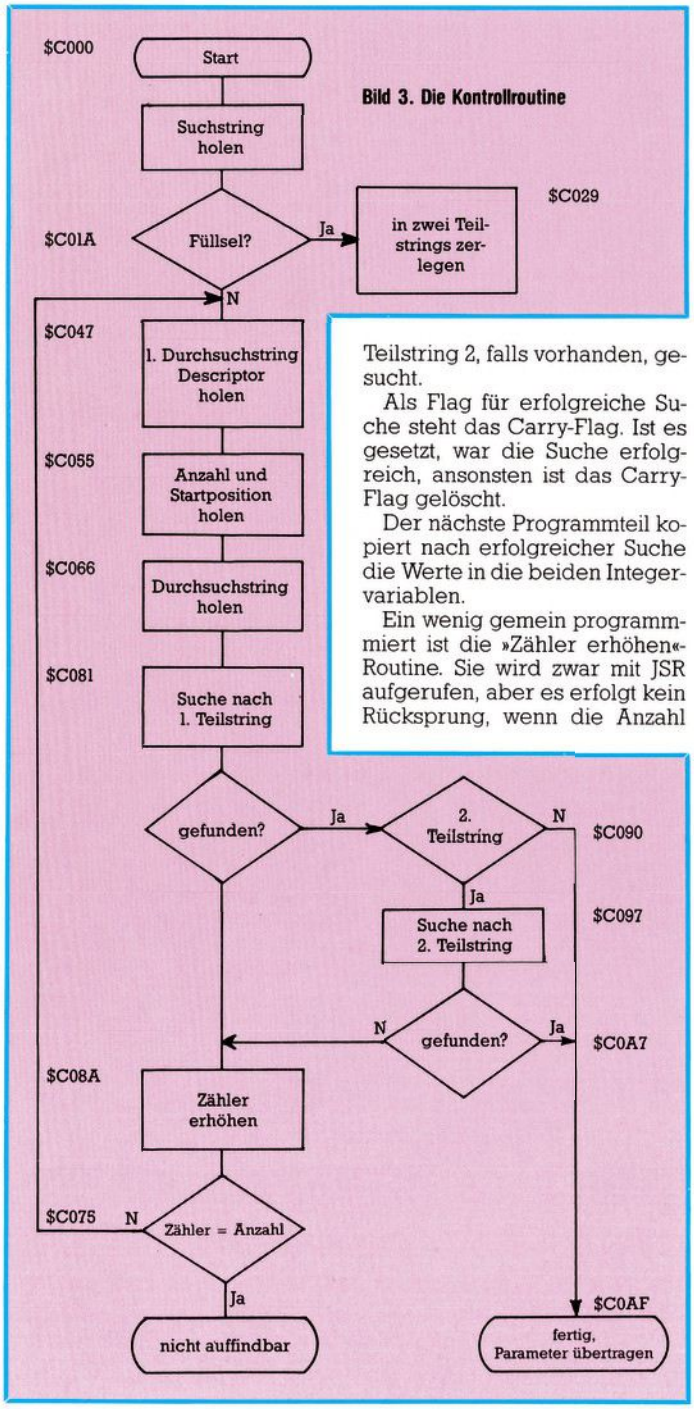

Bild 3. Die Kontrollroutine

$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