Memory Map mit Wandervorschlägen Teil 18 (Schluß)

Bei der Wanderung durch die Speicherlandschaft erreichen wir heute unser Ziel, die Speicherzelle 1023. In diesem letzten Teil werden alle indirekten Sprungvektoren für das Betriebssystem behandelt.

Heute sind die indirekten Sprungvektoren auf Routinen des Betriebssystems an der Reihe, gefolgt vom Kasstettenpuffer. Das alles bringt uns ans Ziel der Reise durch die Speicherlandschaft.

Adresse 788 bis 789 ($314 bis $315)

Vektor auf die IRQ-Interrupt-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 59953 ($EA31) — beim VC 20 auf 60095 ($EABF). Ab hier beginnt die Routine des Betriebssystems, die den IRQ-Interrupt ausführt. Die Bedeutung der verschiedenen Interrupts (Unterbrechungen), ihre Auslöser und Abläufe sind als Übersicht im Texteinschub »Dem Computer ins Wort fallen« dargestellt.

Die IRQ-Routine wird vom Timer A des Ein-/Ausgabe-Bausteins CIA #1 — beim VC 20 vom Timer 1 des Ein-/Ausgabe-Bausteins VIA #2 — ausgelöst, und zwar periodisch 60mal in jeder Sekunde. In der Programmpause werden die im Texteinschub beschriebenen »Haushaltsarbeiten« durchgeführt.

Dieser Vektor eignet sich hervorragend für eigene Programmierzwecke, da er durch das Verbiegen auf eine andere Adresse seine gleichmäßige und hochfrequente Wiederkehr nicht verliert. Mit seiner Hilfe können also eigene Maschinenprogramme 60mal in der Sekunde in ein Programm eingeschoben werden — eine Methode, die deswegen den englischen Namen »Wedge« = Keil erhalten hat. Zwei Vorbedingungen sind allerdings dabei zu erfüllen.

1. Da ein IRQ mit Sicherheit während des Verbiegens auftritt, muß er vorher abgeschaltet werden. Den Schlüssel dazu bietet die Speicherzelle 56334, die mit 0 gePOKEt den Interrupt abschaltet und mit POKE 56334,1 ihn wieder zuläßt. Beim VC 20 ist dies POKE 37116,127 beziehungsweise POKE 37116,192. Aber Vorsicht!! Da während eines IRQ-Interrupts auch die Tastatur abgefragt wird, kann das Abschalten nur innerhalb eines Programms erfolgen — während der Abschaltung ist die Tastatur tot.

2. Am Ende eines »Wedge« muß der Sprung auf die alte IRQ-Adresse erfolgen, die ursprünglich in den Speicherzellen 788 bis 789 stand, damit — etwas verspätet zwar — die normalen Haushaltsarbeiten des IRQ nachgeholt werden können. Bei längeren Wedges wird daher die interne Uhr TI und TI$ etwas nachgehen.

Ich habe lange nach einem Beispiel gesucht. Ich kenne viele: Abfrage der Joysticks, Lautstärke von Tönen mit Funktionstasten steuern, von Basic unabhängige Laufschrift, um ein paar zu nennen. Aber alle haben einen ziemlich langen Maschinensprache-Teil. Ich bringe daher hier das kürzeste Beispiel, das ich kenne. Es stammt aus dem CHIP-Sonderheft »C 64 PEEK + POKE Adreßbuch« von Rügheimer und Spanik.

Das Programm verändert dauernd die Farbe des Bildschirmrahmens:

```
10  FOR K=679 TO 699
20  READ A
30  POKE K,A:NEXT
40  DATA 166,162,224,0,224,128,240
50  DATA 3,76,49,234,174,32,208
60  DATA 202,142,32,208,76,175,02
70  POKE 56334,0
80  POKE 788,167:POKE 789,2
90  POKE 56334,1
```

Dieses Programm gilt nur für den C 64; für den VC 20 müßte es entsprechend umgeschrieben werden.

Die Zeilen 10 bis 30 lesen das Maschinenprogramm, das in den DATA-Zeilen 40 bis 60 steht, in die Speicherzellen 679 bis 699. Diese stehen, wie wir das letzte Mal gesehen haben, zur freien Verfügung — und sind daher ideal geeignet, ein kleines Maschinenprogramm ungestört aufzunehmen.

In Zeile 70 wird der IRQ-Interrupt unterbrochen. Jetzt kommt der wichtige Teil: Zeile 80 verbiegt den IRQ-Vektor zur Speicherzelle 176 + 256 * 2 = 679. Zeile 90 schaltet schließlich den IRQ-Interrupt wieder ein.

Jetzt passiert also folgendes: Jedesmal, wenn der Timer A den Haushalt-IRQ auslöst, springt der Computer zuerst einmal auf das Maschinenprogramm ab Speicherzelle 679 und schaltet die Rahmenfarbe um. Dann erst springt der letzte Befehl des Maschinenprogramms auf die ursprüngliche IRQ-Adresse 59953 ($EA31), von der aus das Betriebssystem weitermacht, als sei nichts geschehen.

Für Kenner gebe ich noch das Assembler-Listing des Maschinenprogramms an:

```
,02A7  A6  A2        LDX  A2
,02A9  E0  00        CPX  #00
,02AB  E0  80        CPX  #80
,02AD  F0  03        BEQ  02B2
,02AF  4C  31  EA    JMP  EA31
,02B2  AE  20  D0    LDX  D020
,02B5  CA            DEX
,02B6  8E  20  D0    STX  D020
,02B9  4c  Af  02    JMP  02AF
```

Es gibt noch eine kleine, erwähnenswerte Anwendung. Wenn der Vektor nicht auf den Anfang der IRQ-Routine bei 59953, sondern auf 59956 — also drei Stellen weiter — zeigt, übergeht die IRQ-Routine den Teil, welcher die STOP-Taste abfragt und die TI/TI$-Uhr weiterschaltet, wodurch effektiv die STOP-Taste ausgeschaltet wird. Das geht ganz schnell mit POKE 788,52. Mit POKE 788,49 wird das wieder rückgängig gemacht. Beim VC 20 sind es die Werte POKE 788,194 oder POKE 788,191.

Adresse 790 bis 791 ($316 bis $317)

Vektor auf die BREAK-Interrupt-Routine des Betriebssystems

Diese Routine ist im Texteinschub nicht erwähnt, weil sie ein Teil der NMI-Routine ist. Dieser Vektor zeigt auf die Adresse 65126 ($FE66) — beim VC 20 auf 65234 ($FED2). Die da beginnende Routine des Betriebssystems wird aufgerufen, wenn der Maschinenbefehl BRK ausgeführt wird. Er führt letztlich zu einem Warmstart, das heißt der Bildschirm wird gelöscht und der Cursor meldet sich mit READY. Diese Routine wird auch durch das gleichzeitige Drücken der STOP- und der RESTORE-Taste angestoßen.

Adresse 702 bis 793 ($318 bis $319)

Vektor auf die NMI-Routine des Betriebssystems.

Der NMI-Interrupt ist im Texteinschub »Dem Computer ins Wort fallen« näher beschrieben. Der Vektor zeigt auf den Beginn dieser Routine ab Speicherzelle 65095 ($FE47) — beim VC 20 ab 65197 ($FEAD).

Sobald ein NMI-Interrupt auftritt, wird zuerst durch Setzen der Interrupt-Abschalt-Flagge (Interrupt Disable Flag) jede Unterbrechung durch den IRQ-Interrupt unterbunden. Dann wird geprüft, wer den NMI-Interrupt ausgelöst hat, und zwar in der Reihenfolge: RS232-Schnittstelle, RESTORE-Taste; eingesteckte Modul und schließlich die STOP-Taste. Die letztere dient zum Sichern der RESTORE-Taste. Nur wenn beide gemeinsam gedrückt werden, kommt die

NMI-Unterbrechung durch die RESTORE-Taste zur Auswirkung.

Da die RESTORE-Taste fast als erste abgefragt wird, kann sie und ihre Kombination mit der STOP-Taste durch Verbiegen des Vektors in Speicherzelle 792 bis 793 abgeschaltet werden. Beim C 64 geht das mit POKE 792,193. Wieder eingeschaltet wird mit POKE 792,71. Beim VC 20 geht das mit POKE 792,91 beziehungsweise POKE 792,173. Natürlich können Spezialisten durch Verbiegen des Vektors auf andere Adressen ihre eigenen NMI-Routinen bauen.

Adresse 794 bis 795 ($31A bis $31B)

Vektor auf die OPEN-Routine des Betriebssystems

Die Routine beginnt ab Adresse 62282 ($F34A) — beim VC 20 ab 62474 ($FEAD). Diese Routine prüft, ob eine Datei (File) eröffnet werden kann. Das geht immer dann, wenn die File-Nummer nicht 0 ist und wenn weniger als 10 andere Dateien bereits eröffnet sind. Für die serielle Schnittstelle (Geräte-Nummer 4, 5, 8 bis 11) wird an das angewählte Gerät zuerst der Befehl »Listen« gegeben und dann die Sekundär-Adresse des OPEN-Befehls.

Beim Bandgerät (Geräte-Nummer 1) prüft die Routine den Tape Header einer sequentiellen Datei beziehungsweise schreibt einen Tape Header auf das Band.

Bei Anwahl der RS232-Schnittstelle (Geräte-Nummer 2) aktiviert die Routine einige Leitungen und reserviert je einen Ein- und Ausgabe-Pufferspeicher am oberen Ende des Basic-Programmspeichers.

Adresse 796 bis 797 ($31C bis $31D)

Vektor auf die CLOSE-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62097 ($F291) — beim VC 20 auf 62282 ($F34A). Ab hier beginnt eine Routine, die beim CLOSE-Befehl zuerst prüft, ob die Datei-Nummer in der Tabelle der eröffneten Datei enthalten ist. Dann holt sie die dazugehörige Geräte-Nummer und Sekundär-Adresse und schließt den Kanal und die Datei.

Adresse 798 bis 799 ($31E bis $31F)

Vektor auf die CHKIN-Routine des Betriebssystems

Diese Routine beginnt ab Adresse 61966 ($F20E) — beim VC 20 ab 62151 ($F2C7). Sie eröffnet einen Datenkanal zur Übernahme von Daten von dem Gerät, das durch den OPEN-Befehl angegeben worden ist.

Adresse 800 bis 801 ($320 bis $321)

Vektor auf die CKOUT-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62032 ($F250) — beim VC 20 auf 62217 ($F309). Dort beginnt die Routine, welche einen Datenkanal zur Abgabe von Daten an das im OPEN-Befehl angegebene Gerät aufmacht.

Adresse 802 bis 803 ($322 bis $323)

Vektor auf die CLRCHN-Routine des Betriebssystems

Der Name dieser Routine ist die Abkürzung für »clear channel«. Diese Routine, die ab Adresse 62259 ($F333) — beim VC 20 ab 62451 ($F3F3) — beginnt, setzt alle Kanäle in den Einschaltzustand zurück. Das heißt, das Eingabegerät ist die Tastatur, das Ausgabegerät ist der Bildschirm.

Adresse 804 bis 805 ($324 bis $325)

Vektor auf die CHRIN-Routine des Betriebssystem

Dieser Vektor zeigt auf die Adresse 61783 ($F157) — beim VC 20 auf 61966 ($F20E). Die hier beginnende Routine, deren Abkürzung »Character Input« bedeutet, holt das jeweils nächste Byte vom Eingabepuffer des angewählten Gerätes, sofern ein solcher eingerichtet ist (zum Beispiel Kassettenpuffer, RS232-Puffer).

Bei Eingabe von der Tastatur holt diese Routine so lange Bytes aus dem Tastaturpuffer und zeigt sie auf dem Bildschirm an, bis das Zeichen für ein ungeSHIFTetes RETURN auftritt. Erst dann gibt die Routine das erste Zeichen der logischen Zeile auf dem Bildschirm an den Basic-Übersetzer weiter.

Adresse 806 bis 807 ($326 bis $327)

Vektor auf die CHROUT-Routine des Betriebssystems

Die CHROUT-Routine entspricht der CHRIN-Routine in der anderen Richtung. Sie bedeutet »Character Output« und transferiert ein Byte, das im Akkumulator steht, in den Puffer des angewählten Ausgabegerätes. Sie beginnt ab Adresse 62898 ($F1CA), — beim VC 20 ab 62074 ($F27A).

Adresse 808 bis 809 ($328 bis $329)

Vektor auf die STOP-Routine des Betriebssystems

Der Vektor zeigt auf die Adresse 63213 ($F6ED) — beim VC 20 auf 63344 ($F770). Die dort beginnende Routine prüft, ob die STOP-Taste gedrückt ist. Durch Verbiegen dieses Vektors kann die STOP-Taste abgeschaltet werden. Beim C 64 geht dies mit POKE 808,239; wieder eingeschaltet wird die STOP-Taste mit POKE 808,237. Beim VC 20 sind die Werte POKE 808,100 beziehungsweise POKE 808,112.

Adresse 810 bis 811 ($32A bis $32B)

Vektor auf die GETIN-Routine des Betriebssystems

Diese Routine ist fast identisch mit der CHRIN-Routine (siehe Speicherzellen 804 bis 805). Sie holt genauso Zeichen von angewählten Geräten in die Eingabepuffer. Der einzige und damit wichtigste Unterschied liegt in der Behandlung der Tastatur-Eingabe. Im Gegensatz zu CHRIN holt sie ein Byte aus dem Tastaturpuffer sofort in den Akkumulator. Der Vektor zeigt auf den Anfang der Routine ab Speicherzelle 61785 ($F13E) — beim VC 20 ab 61941 ($F1F5).

Adresse 812 bis 813 ($32C bis $32D)

Vektor auf die CLALL-Routine des Betriebssystems

CLALL ist die Abkürzung für Close All (Channels and Files). Diese Routine, die ab Adresse 62255 ($F32F) — beim VC 20 ab 62447 ($F3EF) — beginnt, setzt die Speicherzelle 152 auf 0 und schließt so zwangsläufig alle Dateien und Kanäle.

Adresse 814 bis 815 ($32E bis $32F)

Freier Vektor

Nach dem Einschalten zeigt dieser Vektor auf die BREAK-Routine, genauso wie der Vektor in Speicherzelle 790/791. Er ist ein Überbleibsel aus dem PET-Betriebssystem, das aber beim VC 20 und C 64 keine Rolle spielt. Hier können also eigene Vektoren definiert und eingesetzt werden.

Adresse 816 bis 817 ($330 bis $331)

Vektor auf die LOAD-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62622 ($F49E) — beim VC 20 auf 62793 ($F549). Die dort beginnende Routine transferiert Daten von einem Eingabegerät direkt in den RAM-Speicher. Sie kann auch zum VERIFYen durch Vergleich der geLOADeten mit den ursprünglichen Daten verwendet werden.

Adresse 818 bis 819 ($332 bis $333)

Vektor auf die SAVE-Routine des Betriebssystems

Diese Routine ist das Gegenstück zur LOAD-Routine. Sie beginnt ab Adresse 62941 ($F5DD) — beim VC 20 ab 63109 ($F685).

Adresse 820 bis 827 ($334 bis $33B)

Freier Speicherbereich

Diese 8 Byte stehen zur freien Verfügung.

Adresse 828 bis 1019 ($33C bis $3FB)

Kassettenpuffer

Diese 192 Byte beherbergen den Kassettenpuffer. Der Name kennzeichnet diesen Speicherbereich als Zwischenspeicher für Ein- und Ausgabe-Operationen von und auf Band.

Dabei unterscheiden sich die normalen LOAD-, SAVE- und VERIFY-Befehle von den Datei-Befehlen INPUT#, GET# und PRINT#.

Bei LOAD, SAVE und VERIFY steht im Kassettenpuffer lediglich der Vorspann, der auf englisch »Tape Header« heißt. Die Funktion und Zusammensetzung des Tape Headers habe ich schon bei den Speicherzellen 183 bis 187 in Ausgabe 10/85, genau gesagt im Texteinschub »Tape Header« auf Seite 140 detailliert beschrieben. Die eigentlichen Daten berühren den Kassettenpuffer nicht, sondern werden direkt von und in den RAM-Speicher transferiert.

Eine Ergänzung zu der Erklärung des Tape Headers möchte ich noch nachtragen. Die Kennzahl im ersten Byte kann nicht nur, wie beschrieben, die Werte 1 und 3, sondern auch 2, 4 und 5 annehmen. Die Kennzahl 4 bezeichnet den Datenblock als Header einer Basic-Datei (GET# und so weiter), die Kennzahl 2 die danach folgenden Datenblöcke. Die Kennzahl 5 signalisiert, daß der Block das logische Ende des Bandes darstellt, und daß das Betriebssystem nicht weiter suchen muß.

Bei GET#, INPUT# und PRINT# werden nicht nur der Tape Header, sondern auch alle Daten im Kassettenpuffer zwischengespeichert. Dieser blockweise Transport ist an den charakteristischen Unterbrechungen des Datasettenmotors

leicht zu erkennen.

Der Kassettenpuffer kann durch Verbiegen der Zeiger in Speicherzelle 178 bis 179 auf beliebige Plätze des Speichers, aber nicht unterhalb 512, geschoben werden. Normalerweise gibt das keinen Sinn, es sei denn, der Speicherbereich 828 bis 1019 wurde mit einem eigenen Maschinenprogramm belegt.

Adresse 1020 bis 1023 ($3FC bis $3FF)

Freie Speicherplätze

Auch diese 4 Byte stehen zur freien Verfügung.

Liebe Leser, wir sind am Ziel unserer Wanderung durch die Speicherlandschaft des C 64 beziehungsweise des VC 20 angelangt. Ich muß zugeben, es hat länger gedauert, als ich mir zu Beginn vorgestellt habe. Schuld daran war nicht die Länge des Weges — die war durch Start bei Speicherzelle 0 und Ziel bei Speicherzelle 1023 fest vorgegeben. Aber ich habe gebummelt, ich habe viele »Wandervorschläge« gemacht und mir bei sehenswerten Adressen Zeit genommen, sie genauer zu besichtigen.

Ich will den Wandervergleich nicht weiter strapazieren, sondern mich zum Schluß bei allen Lesern, die mir Zuschriften, Fragen, Vorschläge und Verbesserungen geschickt haben, recht herzlich bedanken.

(Dr. H. Hauck/ah)

Texteinschub #1
Dem Computer ins Wort fallen

Jedesmal, wenn ein Computer eingeschaltet wird, würden seine vielen Schaltkreise und Speicherzellen irgendwelche ungeordneten Zahlen enthalten, wenn nicht ein bestimmter Schaltkreis ein RESET-Signal erzeugte. Dieses spezielle Signal geht an alle wichtigen Teile des Computers, nämlich an den Mikroprozessor und an die Bausteine für Ein- und Ausgabe.

Dadurch wird der Computer in einen definierten Anfangszustand versetzt, in dem entweder das Betriebssystem oder, falls vorhanden, ein selbststartendes Steckmodul die Befehlsgewalt erhält.

Die fest vorgegebenen Programmschritte dieser beiden lassen jedoch ein Arbeiten mit dem Computer ohne weiteres nicht zu. Wir könnten nämlich kein Resultat an ein Ausgabegerät (Drucker, Floppy, Datasette, Bildschirm) geben, und wir könnten auch keine Daten eingeben (Tastatur, Floppy, Datasette).

Der Computer wäre nicht steuerbar, wenn wir ihn nicht in seinem vorgegebenen Programmablauf unterbrechen könnten.

Die Unterbrechungsmöglichkeit heißt in der Fachsprache »INTERRUPT«.

Im Gegensatz zu den Großrechenanlagen, die meistens mit vielen Klassen von Interrupts ausgerüstet sind, haben die Heim-Computer von Commodore nur zwei Arten:

— IRQ — der Interrupt Request
— NMI — der Non Maskable Interrupt

Ich habe nicht vor, Ihnen alle Details der Interrupt-Technik zu erklären. Das geht weit über den normalen Umfang meiner Texteinschübe hinaus. Sie können übrigens darüber in anderen Aufsätzen nachlesen, zum Beispiel von Helmut Welke in Ausgabe 11/84, Seite 84, oder im Assembler-Kurs von Heimo Ponnath in den Ausgaben 7 bis 9/85.

Aber einige Erklärungen, so hoffe ich jedenfalls, werden Ihnen auch hier das Interrupt-Prinzip deutlich machen.

Die beiden oben genannten Unterbrechungsarten unterscheiden sich sowohl dadurch, wer die Unterbrechung auslösen kann, als auch in der Art, wie sie gehandhabt werden.

NMI-Auslöser

sind Signale der RS232-Schnittstelle und der Autostart-Steckmodule. Dazu kommen noch die RESTORE-Taste, wenn sie gleichzeitig mit der RUN/STOP-Taste gedrückt wird, und der CIA #2 beziehungsweise der VIA #1.

Wie gesagt, nähere Einzelheiten darüber finden Sie in den oben genannten Aufsätzen.

IRQ-Auslöser

ist 60mal in der Sekunde das Betriebssystem selbst, um die Werte von TI und TI$ höher zu setzen, um zu prüfen, ob die STOP-Taste gedrückt ist, um das Cursorblinken zu erzeugen, um die Tasten der Datasette und schließlich auch die Tastatur abzufragen. Ein IRQ-Interrupt kann aber auch durch Lesen oder Schreiben vom — beziehungsweise auf das — Band, durch die serielle Schnittstelle und durch die Rasterzeilen-Abtastung ausgelöst werden. Programmierbare IRQ-Interrupts sind möglich durch Sprite-Kollisionen, durch Lichtgriffel-Signale und durch den CIA #1 beziehungsweise den VIA #2. Besonders durch die letzteren Ein-/Ausgabe-Bausteine unterscheiden sich die Interrupts von C 64 und VC 20.

NMI-Abläufe

sind schon durch ihren Namen gekennzeichnet. »Non-Maskable« heißt soviel wie »nicht unterdrückbar«. Immer, wenn ein NMI-Signal ankommt, merkt sich der Computer, was er gerade macht, unterbindet alle IRQ-Signale und springt auf eine NMI-Routine, deren Beginn mit dem Vektor in Speicherzelle 792 bis 793 vorgegeben ist.

Herr Ponnath hat im Assembler-Kurs dies sehr treffend mit dem überkochenden Kessel auf dem Herd verglichen, der heruntergestellt werden muß, selbst wenn gerade die Türglocke klingelt, was uns normalerweise beim Lesen der Zeitung unterbrechen würde.

Erst in der NMI-Routine werden nach einer vorgegebenen Prioritätsliste alle NMI-Auslöser der Reihe nach abgefragt, bis der Verursacher gefunden ist.

IRQ-Abläufe

sind Maskable, das heißt sie können, wie gerade gesagt, unterdrückt werden, entweder durch programmiertes Abschalten — das entspricht dem Abstellen der Türglocke — oder durch ein NMI-Signal.

Bei einem IRQ-Signal wird zuerst der gerade laufende Befehl noch bearbeitet, dann startet die IRQ-Routine, deren Beginn durch den Vektor in Speicherzelle 788 bis 789 vorgegeben ist. In dieser Routine wird entschieden, ob der IRQ-Interrupt durch den Maschinencode-Befehl BRK (Break) oder duch angeschlossene Peripheriegeräte ausgelöst worden ist.

Wir sehen also, daß die Unterbrechungen einer festgelegten Priorität unterworfen sind. Ihre Steuerung aber erfolgt immer so, daß keine Interrupt-Anmeldung verloren geht, sondern jede in der gebührenden Reihenfolge abgearbeitet wird.

Schließlich sei noch hervorgehoben, daß der Sprung in die Interrupt-Routinen über die Vektoren die Möglichkeiten eröffnet, diese Routinen nach eigenem Geschmack abzuändern beziehungsweise durch eigene Routinen zu ersetzen.

Texteinschub #2
Reparatur eines LOAD ERRORs

Die Datasette — das Bandgerät von Commodore — ist sicher eines der sichersten und zuverlässigsten seiner Art.

Und doch weigert sie sich gelegentlich, ein Programm vom Band in den Computer zu laden. Alles, was der Computerfreund erhält, ist die Fehlermeldung LOAD ERROR auf dem Bildschirm.

Natürlich: die nächstliegende Maßnahme ist, den LOAD-Vorgang zu wiederholen. Bringt auch das keinen Erfolg, muß die Flinte noch lange nicht ins Korn geworfen werden. Eine kleine Diagnose und die Kenntnis des Tape Headers im Kassettenspeicher (Speicherzelle 828 bis 1023) kann in den meisten Fällen weiterhelfen.

Die 1. Diagnose:

Wenn ein Programm auf Band geSAVEt wird, tut das der C 64 und VC 20 zur Sicherheit gleich zweimal, mit zwei völlig identischen Blöcken. Beim Laden des Programms wird der erste Block in den Arbeitsspeicher des Computers geladen.

Anschließend wird Byte für Byte der zweite Block vom Band mit dem ersten Block im Speicher verglichen. Übersteigt die Anzahl der dabei gefundenen Fehler ein bestimmtes Maß, dann bricht der Computer mit LOAD ERROR ab.

!! Der erste Programmblock steht aber immer noch im Arbeitsspeicher !!

Um zu sehen, ob er in Ordnung oder halbwegs brauchbar ist, machen Sie bitte nach der Fehlermeldung gar nichts — kein RUN, kein RESTORE — und LISTen Sie lediglich das Programm. Besteht es nur aus verfälschten Zeilen und Symbolen, dann ist nicht mehr viel zu retten.

Ist es aber fast oder völlig intakt, können wir es retten. Doch auch jetzt ist noch Vorsicht geboten. Lassen Sie das Programm in Ruhe und heben Sie sich die Korrekturen etwaiger Fehler für später auf.

Die 2. Diagnose:

Sie betrifft den Tape Header. Vor dem Laden des ersten Programmblocks in den Arbeitsspeicher kommt der Tape Header in den Kassettenpuffer (siehe den Texteinschub »Tape Header« in Ausgabe 10/68, Seite 140).

In Speicherzelle 828 steht ein Kennzeichen-Byte, in 829/830 in Low/High-Byte-Darstellung die Adresse, ab der das Programm im Arbeitsspeicher steht.

Für uns ist aber die Adresse wichtig, die in Speicherzelle 831/832 steht. Sie nennt dem Betriebssystem nämlich die Endadresse des Programms im Arbeitsspeicher. Diese Adresse wird nach dem erfolgreichen Abschluß des Ladevorgangs in die Speicherzellen 45/46, 47/48 und 49/50 eingeschrieben.

Ich sagte: »nach dem erfolgreichen Ladevorgang«. Und das gerade ist ja leider nicht eingetreten — deswegen können wir den akzeptablen ersten Programmblock im Arbeitsspeicher nicht RUNen, korrigieren und sonstwie verarzten, nur LISTen.

Reparatur:

Da durch den Abbruch die Zeiger in oben genannten drei Speicherzellenpaaren nicht gesetzt worden sind, holen wir das ganz einfach manuell nach mit der folgenden Direkteingabe:

```
POKE 45,PEEK(831): POKE 46,PEEK(832):
POKE 47,PEEK(831): POKE 48,PEEK(832):
POKE 49,PEEK(831): POKE 50,PEEK(832):
```

Das geht auch etwas eleganter und kürzer:

```
FOR K=45 TO 49 STEP 2: POKE K,PEEK(831): POKE K+1,PEEK(832): NEXT
```

Damit sind die Zeiger richtig gesetzt, und Sie haben Ihr Programm wieder. Erst jetzt dürfen Sie eventuelle Fehler korrigieren.

Ich habe nicht erwähnt, was die Zeiger in 45/46, 47/48 und 49/50 bedeuten. Aber das steht ja schließlich in der Memory Map.

BIT Nr.	WERT	FLAGGE	ABKÜRZUNG
0	1	Übertrag	C(arry)
1	2	NULL	Z(ero)
2	4	Unterbrechung	I(nterrupt)
3	8	Dezimal	D
4	16	Abbruch	B(reak)
5	32	nicht benutzt	
6	64	Überlauf	V
7	128	Vorzeichen	N(egativ)

Um eine der Flaggen des Status-Registers zu löschen, empfiehlt es sich, das ganze Register mit POKE 783,0 zu löschen. Umgekehrt muß man beim Setzen der Bits sehr aufpassen wegen der Unterbrechungsflagge I. Eine 1 in I entspricht dem Maschinen-Befehl SEI, der alle Interrupts ausschaltet, auch die der Tastatur-Abfrage, was natürlich sehr störend sein kann! Um alle Flaggen außer der Unterbrechungsflagge I zu setzten, muß POKE 783,247 eingegeben werden.

So, jetzt wird es Zeit für ein Beispiel, wie vor dem SYS-Befehl Parameter eingegeben werden können. In der Literatur wird immer das Beispiel gewählt, den Cursor auf eine bestimmte Position zu setzen, beziehungsweise seine Position abzufragen. Dazu gibt es eine Routine, die bei beiden Computern ab Speicherzelle 65520 ($FFF0) beginnt.

Sie nimmt die Zahl, die im X-Register steht, und verwendet sie als Zeilennummer; die Zahl des Y-Registers nimmt sie als Spaltennummer, setzt dann den Cursor an diese Stelle und bringt die beiden Werte in die Speicherzellen 209/210 und 211.

Unser Beispiel hat die Aufgabe, den Cursor in die vierte Spalte der siebten Zeile zu setzen, dort das Dollar-Zeichen hinzuschreiben und es rot zu färben.

```
5 PRINT CHR$(147)
10 POKE 783,0
20 POKE 781,6
30 POKE 782,3
40 SYS 65520
```

Nach Löschen des Bildschirms werden zuerst alle Flaggen des Statusregisters gelöscht (Zeile 5). Dann kommt die Zeilennummer in das X-Register (Zeile 10) und die Spaltennummer in das Y-Register (Zeile 30). Nach dem Eingeben dieser Parameter können wir mit SYS auf die Routine springen.

```
50 ZEILE=PEEK(209)+256*PEEK(210)
60 ADRESSE = ZEILE + PEEK(211)
70 POKE ADRESSE,36
```

In Speicherzellen 209/210 können wir jetzt (zur Übung) die Zeilennummer ablesen. Die Adresse der Cursorposition im Bildschirmspeicher erhalten wir durch die Addition der Zeilennummer mit dem Inhalt der Speicherzelle 211. Dorthin POKEn wir den Bildschirmcode des Dollarzeichens, nämlich 36 (Zeile 70).

```
80 SYS 59940
90 FARBE=PEEK(243)+256*PEEK(244)
100 POKE FARBE+PEEK(211),2
```

Für das Färben des Dollarzeichens verwenden wir eine weitere Routine des Betriebssystems, die ab 59940 — beim VC 20 ab 60082 — beginnt. Sie ermittelt die Zeilenposition des Cursors im Farbspeicher und bringt diesen Wert in die Speicherzellen 243/244, wo wir ihn abfragen können (Zeile 90). Die Adresse der Cursorposition im Farbspeicher setzt sich aus diesem Wert plus der Spaltennummer zusammen, die wir wieder der Speicherzelle 211 entnehmen. Auf diesen Platz POKEn wir den Farbcode 2 für rot (Zeile 100). So leicht ist das, wenn man die Routinen und die Aufgaben der Speicherzellen kennt.

Texteinschub #4
Das Mauerblümchen USR

Hand aufs Herz: Haben Sie den USR-Befehl schon einmal benutzt? Ohne Zweifel gehört er zu den Mauerblümchen von Basic, obwohl sein Name — eine Abkürzung von USER (Verwender) — eigentlich genügend Anreiz bieten müßte. Da er ohne die Speicherzellen 784 bis 786 nicht auskommt, ist der heutige Teil des Kurses eine gute Gelegenheit, ihn Ihnen näher zu bringen.

USR hat im Grunde genommen dieselbe Funktion wie SYS. Er springt nämlich aus einem Basic-Programm direkt in ein Maschinen-Programm, arbeitet dieses so lange ab, bis er den Befehl RTS findet. RTS entspricht dem Basic-Befehl RETURN und springt in das Basic-Programm zurück.

Bei SYS steht die Sprungadresse gleich hinter dem Befehl.

Bei USR muß die Sprungadresse zuerst in die Speicherzellen 785/786 gePOKEt werden (beim VC 20 in 1/2).

Beispiel: Sprung auf 56524 ($DCCC)

```
mit SYS: SYS 56524
mit USR: POKE 785,204 (204+256*220=56524)
POKE 786,220
X=USR(Y)
```

Kein Wunder, daß USR selten verwendet wird — ist er doch durch das POKEn der Sprungadresse in Low/High-Byte Darstellung aufgeblüht.

Das ist aber nicht unnütz, weil USR mehr Fähigkeiten hat als SYS. Im Hinblick auf die im anderen Texteinschub »Der vorbereitete SYS-Befehl« aufgezeigten Möglichkeiten des SYS-Befehls sollte ich besser sagen: USR hat andere Fähigkeiten als SYS.

USR ist eine Mischung von SYS und FN. Letzterer ist der Basic-Befehl zur Definition selbst erfundener Funktionen. Bei USR allerdings wird die Funktion als Unterprogramm in Maschinensprache geschrieben, auf die dann wie gesagt der USR-Befehl zur Ausführung springt. Der Pfiff dabei ist aber, daß Zahlenwerte in das Maschinenprogramm mitgenommen beziehungsweise Resultate aus ihm herausgeholt werden können.

Wie läuft das ab: Das Argument Y, das in der Klammer hinter dem Befehl steht, wird zuerst in den Gleitkomma-Akkumulator Nr. 1 (FAC 1) in den Speicherzellen 97 bis 102 gebracht. Als Gleitkommazahl wird es vom angesprungenen Maschinenprogramm weiterverarbeitet. Das Resultat kommt dann wieder in den FAC 1 und steht als Wert von X zur Verfügung.

Das Argument Y kann übrigens auch ein komplexer Ausdruck sein, zum Beispiel: `X=USR(PEEK(A)+256*PEEK(B))`

Ich möchte das an einem kleinen Beispiel demonstrieren.

Statt allerdings ein Maschinenprogramm selbst zu schreiben, verwende beziehungsweise springe ich auf eine Routine des Betriebssystems, die den Inhalt des FAC 1 für mathematische Operationen verwendet.

Als geeignete mathematische Operation habe ich die Routine für die Funktion INT gewählt, die im C 64 ab der Adresse 48332 ($BCCC) beginnt. Zuerst definieren wir einen Wert für die Variable Y, der in die INT-Routine gebracht werden soll:

```
10 Y=14,35
```

Dann bestimmen wir die Sprungadresse für den USR-Befehl. Dazu teilen wir die Adresse 48332 auf in ein Low-Byte = 204 und ein High-Byte = 188. Diese POKEn wir nach 785/786:

```
20 POKE 785,204
30 POKE 786,188
```

Jetzt folgt nur noch der USR-Befehl selbst und das Ausdrucken des Resultats.

```
40 X=USR(Y)
50 PRINT X
```

Nach RUN erhalten wir das Resultat 14, wie das Gesetz für INT es befiehlt.

Sie können zur Übung statt INT auch COS verwenden, indem Sie auf die Adresse 57938 ($E264) beziehungsweise beim VC 20 auf 57935 ($E261) springen. Der Vergleich mit dem Befehl COS Y muß dasselbe Ergebnis bringen. Wer hat übrigens gemerkt, daß wir überhaupt nichts mit der Speicherzelle 784 (beziehungsweise 0) gemacht haben, obwohl sie doch angeblich am USR-Befehl beteiligt ist?

Sie ist es wirklich, doch ohne unser Zutun. In diese Adresse wird beim Einschalten des Computers die Zahl 76 ($4C) geschrieben. Das ist der Code für den Maschinenbefehl »JMP« (jump), der dieselbe Wirkung hat wie GOSUB.

Bei Ausführung von USR springt nämlich die entsprechende Routine zuerst auf die Speicherzelle 784 (beziehungsweise 0), findet dort den Sprungbefehl und in den beiden nachfolgenden Speicherzellen 785 und 786 (beziehungsweise 1 und 2) die Sprungadresse — und führt so den geplanten Sprung aus.

Ich finde, USR ist es wert, in Ihre Überlegungen mit einbezogen zu werden, besonders wenn Sie innerhalb Ihrer Basic-Programme extrem schnelle Unterprogramme in Maschinensprache eingebaut haben. Diese sind mit USR ganz elegant aufrufbar. Ich denke da zum Beispiel an eine Abfrage der Joysticks oder der Paddle.