Ohne gutes Werkzeug geht es nicht: SMON (Teil 4)

Der Trace-Modus, also das Abarbeiten von Maschinenprogrammen Schritt für Schritt, ist das wohl wichtigste Hilfsmittel beim Austesten von Maschinenprogrammen. SMON bietet sogar drei verschiedene Trace-Möglichkeiten, die in diesem vierten Teil dargestellt werden. Außerdem erhalten Sie eine Übersicht aller Befehle, und wir zeigen Ihnen, wie Sie SMON in einen anderen Bereich verschieben können.

Einen Befehl haben wir Ihnen bisher unterschlagen, der zwar bereits vorhanden, aber noch nicht beschrieben war. Es handelt sich um den Vergleich zweier Speicherbereiche. Die Syntax ist sehr einfach:

= 4000 6000

vergleicht den Speicherinhalt ab $4000 mit dem ab $6000. Das erste nicht übereinstimmende Byte wird angezeigt, und der Vergleich wird abgebrochen.

Wenn Sie also ein Maschinenprogramm geschrieben und überarbeitet haben und Sie wissen nicht mehr genau, worin eigentlich der Unterschied zwischen der 76. und der 77. Version besteht, gehen Sie so vor: Laden Sie zuerst Version 76 und verschieben Sie diese mit dem »W«-Befehl in einen freien Speicherbereich. Laden Sie dann Version 77 und führen Sie den »=«-Befehl durch. Sofort finden Sie den Unterschied und können mit der Arbeit an Version 78 beginnen....

Wir wollen uns bei der Beschreibung der Trace-Befehle auf Anwendungsbeispiele konzentrieren. Zum Aufbau der Routine sei nur so viel gesagt: Gesteuert wird sie mit Hilfe des Prozessor-Interrupts, weil nur damit ein Eingriff ins laufende Maschinenprogramm möglich ist. Während des Trace-Ablaufs wird deswegen der Bildschirm kurzfristig aus- und eingeschaltet, weil alle anderen Interruptanforderungen wie zum Beispiel durch den Video-Chip, verhindert werden müssen. Da die Befehle eines Programms nicht nur angezeigt, sondern auch wirklich ausgeführt werden, ist der »SEI«-Befehl mit großer Vorsicht zu verwenden. Doch dazu später mehr. Wir wollen ein neues, besser geeignetes Beispiel verwenden als bisher. Tippen Sie also das folgende Miniprogramm mit dem Assembler ein (A 4000):

```
4000  LDA  #30   lade den Akku mit (ASCII-) 0
4002  JSR  FFD2  gib Akku auf dem Bildschirm aus
4005  CLC
4006  ADC  #01   erhöhe Akku um 1
4008  CMP  #39   vergleiche Akku mit (ASCII-) 9
400A  BCC  4002  springe, wenn Akku kleiner, zurück
400C  BRK        springe in SMON zurück
```

Starten Sie das Programm mit »G 4000«. Es muß die Zahlen von 0 bis 8 auf den Bildschirm schreiben.

◆ Trace-Stop

TS (Startadresse Stoppadresse)

Starten Sie nun unser Programm mit

TS 4000 4009

Die ersten Befehle werden ausgeführt (die Null ausgegeben, der Akku erhöht etc.), dann stoppt das Programm bei Adresse $4009 und springt in die Registeranzeige.

Genau genommen ist »TS« gar kein Trace-Befehl, das Programm läuft nämlich bis zur gewählten Stoppadresse in Echtzeit durch. Dort angekommen, können Sie die Register prüfen und gegebenfalls durch Überschreiben ändern. Mit »G«, »TW« oder »TB« (wird später erklärt) ohne weitere Adresseneingaben können Sie dann im Programmlauf fortfahren. SMON merkt sich nämlich, wo er stehen geblieben ist und arbeitet ab dieser Adresse weiter, wenn Sie nicht eine neue angeben.

Sinnvoll ist dieser Befehl immer dann, wenn in einem längeren Programm nur bestimmte Teile »getraced« werden sollen, der Anfang aber durchlaufen werden muß, um Variable zu setzen oder Benutzereingaben zu erfragen. Auch wenn man nicht ganz sicher ist, ob eine bestimmte Passage überhaupt jemals durchlaufen wird, kann man das mit »TS« überprüfen.

Zwei Einschränkungen gibt es allerdings wegen der Arbeitsweise dieses Befehls: SMON setzt im Programm an die Stoppadresse einen BRK-Befehl und merkt sich, welcher Befehl dort stand, um ihn wieder zurückzuschreiben. Deshalb funktioniert »TS« nur im RAM, nicht aber zum Beispiel im Basic oder im Betriebssystem. Auch darf die Speicherstelle, in der sich SMON den ausgetauschten Befehl merkt ($02BC) vom Programm nicht verändert werden, sonst ist eine korrekte Reparatur nicht mehr möglich.

Der wohl am häufigsten und vielseitigsten eingesetzte Trace-Befehl ist sicherlich »TW«.

◆ Trace Walk

TW (Startadresse)

Starten Sie unser Beispiel jetzt mit TW 4000

Der erste Befehl (LDA #30 in Adresse $4000) wird ausgeführt, SMON stoppt und zeigt dann die Inhalte aller Register in der gleichen Reihenfolge wie beim »R«-Kommando sowie den nächsten Befehl an. Im Akku steht jetzt 30, der Programmzähler zeigt auf $4002. Jetzt drücken Sie eine Taste. Der nächste Befehl (JSR FFD2) wird ausgeführt, der Programmzähler zeigt auf $FFD2. Achten Sie auf den Stackpointer: Sein Inhalt hat sich um 2 vermindert, weil der Prozessor auf dem Stack die Adresse abgelegt hat, an die er nach Beendigung der Subroutine zurückspringen soll. Der nächste angezeigte Befehl ist ein indirekter Sprung über $0326. Mit dem nächsten Tastendruck wird er durchgeführt.

Und so geht es munter weiter. Verzweifeln Sie nicht, wenn Sie auch nach den nächsten zehn Tastendrücken immer noch irgendwo im Betriebssystem »herumtracen« und von unserem Beispielprogramm weit und breit nichts mehr zu sehen ist. Ausnahmsweise ist unser Liebling einmal nicht im »Land der Träume« verschwunden, sondern tut, was er soll: Er arbeitet brav einen Befehl nach dem anderen alles ab, was zur Routine $FFD2 gehört, und das ist reichlich viel. Also bewegen Sie Ihre Finger, Sie haben's ja nicht anders gewollt. Irgendwann einmal, nach mehreren hundert gedrückten Tasten, befinden Sie sich plötzlich wieder in der Registeranzeige von SMON. Das Programm ist beendet. Nun werden Sie enttäuscht fragen, was man wohl mit einem Trace-Modus anfangen soll, der schon bei kleinsten Beispielprogrammen ein völlig undurchschaubares Chaos erzeugt? Nur Geduld, die Rettung naht in Gestalt der Taste »J«.

Falls ihre Hand noch nicht in Gips liegt, starten Sie das Ganze nochmal von vorn mit »TW 4000«. Diesmal drücken Sie aber jedesmal, wenn als nächster Befehl »JSR FFD2« angezeigt

wird, auf »J«. Der Effekt ist, daß die gesamte Subroutine auf einen Schlag abgearbeitet wird und Sie sofort wieder auf dem nächsten Befehl unseres Beispiels landen. Daß wir nicht gemogelt und die Befehle von »JSR FFD2« einfach unterschlagen haben, sehen Sie daran, daß der Akku tatsächlich auf dem Bildschirm ausgegeben worden ist (rechts neben FFD2). Jetzt können Sie unser Beispiel in aller Ruhe bis zu Ende durchgehen und verfolgen, wie der Akku erhöht wird, wie der Vergleich das Statusregister beeinflußt und wie dementsprechend der Rücksprung in die Schleife erfolgt.

Sie dürfen die »J«-Taste auch dann benutzen, wenn Sie schon mitten in der Subroutine sind. Aber hierbei ist äußerste Vorsicht geboten: Die Rücksprungadresse muß unbedingt oben auf dem Stack liegen, wenn Sie »J« drücken. Hat nämlich der Prozessor Werte auf dem Stack abgelegt (mit PHA oder PHP), dann erfolgt der Sprung irgendwo hin, nur nicht zurück ins Programm. Achten Sie deshalb genau auf die Anzeige des Stackpointers. Wenn dessen Wert genau so groß ist wie bei Beginn der Subroutine, kann nichts passieren. Sonst hilft nur noch der Reset-Taster, den Sie ja inzwischen hoffentlich eingebaut haben, oder eine ruhige Hand, die die Büroklammer an Pin 1 und 3 des User-Ports hält (Kostenpunkt der Reparatur bei Abrutschen zirka 100 Mark...)

»TW« bricht automatisch mit der Registeranzeige ab, wenn im Programm ein »BRK«-Befehl auftaucht. Wenn Ihnen das zu lange dauert oder Sie zwischendurch ein Register ändern möchten, können Sie den Trace-Modus jederzeit mit der Stopp-Taste verlassen. Anschließend können Sie wie bei »TS« beschrieben fortfahren.

Im Gegensatz zu »TS« können Sie mit »TW« auch im ROM herumstöbern; Sie haben es ja bei der Subroutine $FFD2 bereits getan. Einzige Einschränkung beim »TW«-Befehl: Ihr Programm darf keinen »SEI« enthalten, da dieser den Interrupt und damit auch den Trace-Modus lahmlegt. Verlassen Sie in diesem Falle »TW« mit STOP und starten erneut hinter dem »SEI«-Befehl. Allerdings müssen Sie in Kauf nehmen, daß das Programm normalerweise nicht mehr korrekt arbeitet.

Das nächste Programm soll als weiteres Beispiel für den TW-Modus dienen. Geben Sie es folgendermaßen ein:

```
5000  LDA  #00   lädt den Akku mit »0«
5002  TAX        überträgt den Akku ins X-Register
5003  .0C        ein mysteriöses Byte
5004  LDA  #04   lädt den Akku mit »4«
5006  TAY        überträgt den Akku ins Y-Register
5007  BRK        springt in SMON
```

Wenn wir dieses kleine Programm abarbeiten, müßte das X-Register auf »0« stehen, während Akku und Y-Register mit »4« geladen sind. Starten wir also das Programm mit »G 5000« und schauen uns die Register an.

Seltsamerweise enthalten alle Register eine »0«. Vorsichtig, wie wir sind, überschreiben wir die drei Register mit »FF«, um die Veränderung deutlich kontrollieren zu können.

Dann starten wir mit »G 5000« ein zweites Mal. Gegen alle Gesetze der Vernunft erscheint wieder das »falsche« Ergebnis — alle drei Register sind »0«. Hier soll uns jetzt der TW-Modus weiterhelfen, indem er uns zeigt, was in Wirklichkeit passiert.

Geben wir »TW 5000« ein. Der erste Befehl (LDA #00) ist durchgeführt, im Akku erscheint die Null. Jetzt steht der Programmzähler auf dem folgenden Befehl »5002 TAX«. Nach Drücken einer Taste wird dieser Befehl ausgeführt und es erscheint die Null im X-Register. Beim folgenden Befehl müssen wir feststellen, daß der Disassembler nicht in der Lage ist, ihn zu interpretieren — er gibt drei Sternchen aus. Hierbei handelt es sich um unser Byte »0C«.

Wieder ein Tastendruck; und dann erkennen wir, daß etwas Merkwürdiges passiert ist. Der Prozessor hat augenscheinlich den nächsten Befehl (LDA #04) übersprungen und steht schon auf dem folgenden »TAY«. So also wird unser Programm abgearbeitet. Damit ist auch das »falsche« Ergebnis erklärt. Bleibt nur noch die Frage nach dem Grund für dieses seltsame Verhalten. Und der ist sicherlich in dem mysteriösen Byte »OC« zu suchen. Hierbei handelt es sich um einen der »inoffiziellen« Opcodes (auch Pseudo-Opcodes genannt), die aufgrund der Prozessorarchitektur vorhanden sind und in manchen Programmen ihr Unwesen treiben — wie wir zu unserem Leidwesen erfahren mußten. Das Byte »OC« wirkt wie ein »NOP«, der eine Länge von 3 Byte hat. Deshalb wird der folgende 2-Byte-Befehl (LDA #04) verschluckt.

Es gibt noch einiges zu entdecken am 6502 und 6510 — TW macht's möglich.

Häufig ist es nicht sinnvoll, ein Programm von Anfang an im TW-Modus laufen zu lassen. Zum anderen sind gerade Schleifen, die per Hand mit »TW« durchlaufen werden müssen, eine ermüdende Angelegenheit. Hier bietet SMON neben dem bereits beschriebenen »TS« eine weitere Trace-Möglichkeit an:

◆ Trace Break

TB (Adresse Anzahl der Durchläufe)

◆ Trace Quick

TQ (Adresse)

Geben Sie als Beispiel folgendes Programm ein:

```
6000  LDY  #00    Y als Zähler auf »0«
6002  LDA  600E,Y  Werte von $600E ff. sollen geladen
                   werden
6005  JSR  FFD2   Ausgabe der Zeichen auf dem Bild-
                   schirm
6008  INY         der Zähler wird erhöht
6009  CPY  #0E    Zähler schon »14«?
600B  BCC  6000   wenn nein, dann nächsten Wert holen
601D  BRK
```

Bei $600E soll nun ein Text stehen, den das Programm ausgibt. Die einfachste Art, mit SMON Texte in den Speicher zu schreiben, besteht im »K«-Befehl. Geben Sie

K 600E

ein (danach natürlich Return) und drücken Sie die STOP-Taste. Fahren Sie mit dem Cursor an das erste ausgegebene Zeichen (vermutlich ein Punkt) und schreiben Sie — ohne Anführungszeichen:

»FEHLER BEHOBEN«

Drücken Sie dann Return, um die Zeile an den Rechner zu übergeben. Wenn Sie das Programm starten, werden Sie wieder einmal Gelegenheit haben, sich in Ruhe etwas zu trinken zu holen (Prost!), denn das Programm enthält einen dummen Fehler und beschäftigt den Computer für eine lange, lange Zeit. Genauer gesagt, bis Sie ihn mit Reset (zum Beispiel durch RUN/STOP-RESTORE) erlösen.

Nun soll SMON helfen, diesen Fehler zu lokalisieren. Setzen Sie zuerst einmal einen Breakpoint bei $6002 und begrenzen die Durchläufe auf die maximale Anzahl:

TB 6002 0E

und starten mit

TQ 6000

den Quicktrace bei $6000. Das Programm läuft so lange, bis zum 14. Mal die Adresse $6002 erreicht wird und springt dann in den TW-Modus. Wenn Sie sich jetzt die Registerinhalte genau anschauen, müßte ihnen der Fehler geradezu ins Auge springen. Wie groß sollte denn das Y-Register sein? Welchen Wert sollte der Akku haben? NA?! (Auflösung erfolgt mit der Bekanntgabe der Gewinner des großen Preisausschreibens aus dem letzten Heft...)

Wenn Sie eine Zeitlang mit SMON gearbeitet haben, werden Ihnen eventuell einige Voreinstellungen nicht gefallen. Besitzer einer Datasette zum Beispiel müssen jedesmal mit »I 01« auf ihre Geräteadresse einstellen. Wenn Sie sich an unserem

»Preisauschreiben« vom letzten Mal beteiligt haben, dürften Ihnen diese Speicherstellen bereits bekannt sein. Übrigens sind zirka 235.982 richtige Lösungen eingegangen; wir haben daraufhin ein Programm geschrieben, um den Gewinner zu ermitteln. Dieses läuft zur Zeit auf einem Großrechner in den USA, da der Speicher des C 64 nicht ausreichte. Wir werden Sie nach Abschluß des Programmlaufes, den wir für Mitte Juni dieses Jahres erwarten, informieren....

Sollten Sie nicht zu den glücklichen Gewinnern zählen, geben wir Ihnen im folgenden die Speicherstellen an, in denen die Voreinstellungen stehen. Diese können Sie dann mit dem »M«-Befehl Ihren Wünschen entsprechend abändern.

Vergessen Sie aber nicht, Ihre geänderte Version mit »S "@:SMON $C000" C000 CE09« abzuspeichern.

Hintergrund-Farbe	:	$C220
Schreibfarbe	:	$C228
Ger. Adr. Drucker	:	$C21B
Ger. Adr. F1/Cass	:	$C216

Wer steht wo ? — Wegweiser durch SMON

Wollen Sie Routinen in SMON analysieren oder verändern, müssen Sie die Einsprungadressen der einzelnen Befehle kennen. Diese lassen sich leicht finden:

Am Anfang des Programms (ab $C00B) befindet sich eine Liste aller Kommandos gefolgt von den Adressen (ab $C02B), bei denen diese Befehle beginnen. Lassen Sie sich mit »M C00B C02B« die Befehlsliste anzeigen (siehe Bild 1). Sie sehen, daß am Ende 5 Nullen stehen; hier können Sie eigene neue Befehle einbauen. Mit »M C02B C06B« erhalten Sie die Liste der Einsprungadressen, immer in der Reihenfolge Low-Byte — Hig-Byte, diesmal mit 10 Nullen am Ende, weil ja zu jedem Befehl 2 Byte für die Adresse gehören.

Wir wollen nun als Beispiel heraussuchen, wo die Routine zum Füllen eines Speicherbereichs mit einem vorgegebenen Byte steht. Der Befehl dazu ist »O«, also der 20. Befehl ($C01E). Die zugehörige Adresse ist ebenfalls die 20. in der Liste also $C9C0 (in Speicherstelle $C051/$C052). Die Routine beginnt allerdings immer erst ein Byte später, in unserem Fall also bei $C9C1.

Disassemblieren Sie jetzt diese Routine mit »C C9C1 C9D3«. Sie erhalten folgende Befehle (natürlich ohne Kommentare):

```
        JSR   C27A   holt zwei Adressen nach $FB/FC
                     und $FD/FE
        JSR   C28D   holt das gewünschte Byte in den
                     Akku
        LDX   #00    initialisiert X-Register als Zähler
LOOP    STA   (FB,X) speichert den Akku-Inhalt in der er-
                     sten Adresse ab
        PHA          merkt sich den Akku-Inhalt
        JSR   C463   erhöht die Adresse und vergleicht
                     mit der Endadresse
        PLA          holt Akku-Inhalt zurück
        BCC   C9C9   wenn Ende nicht erreicht, dann
                     Sprung auf LOOP
        RTS
```

Wenn Sie nun zum Beispiel den »O«-Befehl nur zum Löschen verwenden möchten, könnten Sie die Routine so abändern, daß nur die beiden Adressen eingegeben werden müssen. Überschreiben Sie einfach den Befehl JSR C28D mit LDA #00 im Disassemblerlisting, disassemblieren Sie erneut mit »D C9C1 C9D3« und überschreiben Sie die drei Sternchen bei C9C6 mit einem »NOP«, um die entstandene Lücke (3-Byte-Befehl wurde durch 2-Byte-Befehl ersetzt) aufzufüllen. Nun können Sie Ihre veränderte Routine ausprobieren. Geben Sie zum Beispiel »O 5000 5200« ein und überzeugen Sie sich mit »M 5000 5200« von der korrekten Ausführung. Sie können auch die so veränderte Routine mit »W C9C1 C9D3 CE09« ans Programmende kopieren und sich einen neuen Befehl »E« (Erase) schaffen. Sie brauchen dann nur in die erste Null am Ende der Kommandotabelle »45« (für »E«) und in die ersten beiden Nullen der Adreßtabelle »08« und »CE« zu schreiben (für die Adresse $CE09).

Machen Sie sich ruhig einmal die Mühe, auch andere Routinen im SMON auf diese Art und Weise zu analysieren und zu ändern. Erstens macht es Spaß und zweitens können Sie SMON Ihren eigenen Wünschen anpassen. Vielleicht fallen Ihnen ja noch Routinen ein, die in SMON fehlen. Wir würden uns über Verbesserungsvorschläge freuen.

Das »Gedächtnis« von SMON

Wenn Sie Programme mit SMON untersuchen oder verändern wollen, müssen Sie noch wissen, welche Speicherstellen SMON verwendet. Es soll ja Monitorprogramme geben, die die Basic-Zeiger als Arbeitsspeicher benutzen, so daß ein Basic-Programm nach dem Rücksprung aus dem Monitor gelöscht ist. SMON tut so etwas nicht. Aber natürlich braucht er auch Speicherstellen, um sich Werte merken zu können. Damit Sie Konflikten von Anfang an aus dem Wege gehen können, sind die wichtigsten hier dargestellt.

In der Zeropage belegt SMON den Bereich von $00A4 bis $00B6. Dort stehen Systemvariable für die Kassettenspeicherung und die RS232-Schnittstelle. Diese werden nur während des Betriebs der Kassette oder von RS232 gebraucht, sind ansonsten aber frei. Außerdem werden die Speicherstellen $00FB bis $00FF benutzt, die sowieso zur freien Verfügung des Anwenders vorgesehen sind. Alle anderen Zeiger in der Zeropage, also insbesondere die Speicherverwaltung für Basic bleiben unbeeinflußt.

Als weiteren Arbeitsspeicher benutzt SMON den Bereich von $02A8 bis $02C0. Auch dieser Bereich wird vom Betriebssystem nicht benutzt, so daß keine Konflikte entstehen dürften. Beim Assemblieren wird zusätzlich noch der Kassettenpuffer als Speicher für die Labels benötigt. Dieser bleibt ansonsten aber auch unverändert; das ist wichtig, wenn Maschinenroutinen dort abgelegt werden sollen.

Alles in allem ist SMON also recht verträglich.

SMON verschieben? — Mit SMON!

Eine Reihe von Anfragen hat uns erreicht, ob man SMON nicht mit Hilfe des »W«-, »V«- oder »C«-Kommandos verschieben könne. Alle Versuche in dieser Richtung seien fehlgeschlagen. Einige Leser meinten auch, in der V-Routine müsse ein Fehler stecken. Diesmal sind wir jedoch völlig schuldlos; es gibt nämlich einige Befehle in SMON, die keine Sprungadressen sind und sich trotzdem auf den Bereich ($C000-) beziehen, in dem SMON steht.

Dazu gehören in erster Linie die oben erwähnten Einsprungadressen, deren High-Byte natürlich geändert werden muß, wenn SMON in einem anderen Speicherbereich laufen soll. Es gibt aber auch Befehle, die eine Adresse im Programm in einem Vektor ablegen müssen. Disassemblieren Sie einmal den Anfang von SMON mit »D C000 C00B«. Sie erhalten

```
LDA   #14    Low-Byte der BREAK-Routine von SMON
STA   0316   im Break-Vektor abspeichern
LDA   #C2    High-Byte (!) siehe oben
STA   0317   siehe oben
BRK
```

Damit wird der Break-Vektor des Betriebssystems auf den SMON gesetzt und mit dem anschließenden — und jedem weiteren BRK-Befehl — springt das Programm in SMONs BREAK-Routine. Wenn SMON in einem anderen Bereich als $C000

laufen soll, dann müssen diese Befehle geändert werden. Heraussuchen kann man sie mit »FIC*,C000 D000«. Sie wissen doch noch, was diese Anweisung bedeutet: Suche mir alle Befehle, die ein Register unmittelbar mit einem Wert laden, der mit $C beginnt. Aber Vorsicht! Nicht alles, was da angezeigt wird, muß auch geändert werden! Um Ihnen weitere Stunden sinnlosen Herumbrütens zu ersparen, wollen wir als Beispiel zeigen, wie man SMON in den Bereich $9000 bis $A000 verlegen kann. Natürlich geht das im Prinzip für jeden anderen Bereich genauso; wir selbst haben insgesamt fünf SMON-Versionen für fünf verschiedene Speicherbereiche, von denen eine immer paßt.

1. Wir verschieben zuerst das ganze Programm ohne Umrechnen in den neuen Bereich:

W C000 CE09 9000

2. Nun lassen wir alle absoluten (3-Byte) Befehle umrechnen. Die Tabellen am Anfang von SMON bleiben verschont:

V C000 CE09 9000 9214 9E09

3. Als nächstes ändern wir die High-Bytes der Befehlsadressen. Geben Sie

M 902B 906B

ein und ändern Sie in jedem zweiten Byte das »C« durch Überschreiben in »9«. Vergessen Sie nicht, am Ende jeder Zeile »Return« zu drücken, damit Ihre Änderung auch übernommen wird.

4. Als letztes bleiben noch die oben beschriebenen Direktlade-Befehle. Sie müssen so geändert werden, daß sie sich auf den neuen Bereich $9... beziehen. Suchen Sie sie mit

FIC*,9000 9E09

heraus. Sie erhalten

9005	LDA	#C2	ändern
9124	CPX	#C0	nicht ändern
9386	LDY	#C0	ändern
9441	CMP	#C0	nicht ändern
987F	LDX	#C3	nicht ändern
988D	LDX	#C1	nicht ändern
9992	LDA	#C1	nicht ändern
9C2C	LDA	#CC	ändern
9C5B	LDA	#C2	ändern
9CF4	LDA	#CC	ändern
9DA1	LDX	#CC	ändern
9E03	LDA	#CC	ändern

Ändern Sie nur in den gekennzeichneten Zeilen das »C« in »9« um. Sie sehen, es gibt keine Regel, welche Befehle zu ändern sind und welche nicht. Aus diesem Grunde müssen Sie diese Änderungen »von Hand« vornehmen.

Speichern Sie jetzt die fertige Version unbedingt mit »S "SMON $9000" 9000 9E09« ab. Nun starten Sie die 9000-Version mit »G 9000« oder von Basic aus mit SYS 36864 (SYS 9 x 4096). Löschen Sie dann den alten SMON mit »O C000 D000 00«. Nur so können Sie sicher sein, daß der verschobene SMON auch einwandfrei arbeitet und Sie nichts übersehen haben.

Probieren Sie nun alle Befehle durch. Sie müssen genauso arbeiten wie bisher. Vor allem können Sie jetzt auch Programme wie »DOS 5.1« oder »Turbo Tape« untersuchen, die im $C000-Bereich stehen. Achten Sie aber, wenn Sie »SMON $9000« von Basic aus benutzen, darauf, daß das Basic ihn nicht überschreibt. String-Variable werden nämlich von $A000 nach unten hin aufgebaut und bis $9E09 ist nicht viel Platz. Schützen Sie im Zweifelsfalle den Bereich, indem Sie nach dem Laden des SMON $9000 eingeben:

NEW : POKE 56,144 : POKE 55,0

Damit ist SMON vor Überschreiben geschützt. Das ist natürlich bei dem SMON $C000 nicht nötig, weil Basic in diesen Bereich nicht hineinkommt.

Zunächst wieder ein Geständnis: Leider hat uns der Druckfehlerteufel auch in der Dezember-Ausgabe wieder erwischt, aber der arme Kerl will ja schließlich auch seinen Spaß haben. Verzeihen Sie ihm und uns

— daß er im Listing in Zeile 150 ein REM eingeschummelt hat, das dort überhaupt nichts zu suchen hat. Es verhindert nämlich, daß die Checksumme geprüft wird und Eingabefehler erkannt werden.

— daß er auf Seite 62 fälschlich behauptet hat, die »LOAD«-Routine beginne bei $C84F. In Wirklichkeit beginnt Sie bei $C84E.

— daß er auf Seite 63 bei den Hinweisen in eigener Sache empfiehlt, man möge bei »... Error in 40« (oder in 70) die DATAs kontrollieren. Natürlich tritt der Fehler nicht in den REM-Zeilen, sondern in Zeile 140 oder 180 auf.

— daß er schließlich schon vorgearbeitet hat und in der Januar-Ausgabe sein schändliches Handwerk getrieben hat. Dort steht bei der Beschreibung des 16-Bit-Vergleichs, das Zero-Flag sei gesetzt, wenn beide Werte gleich seien. Das stimmt auch, aber Sie wissen ja selbst, wo der Teufel steckt, nämlich im Detail, und der Druckfehlerteufel macht da keine Ausnahme. Er verführt uns nämlich zu dem eigentlich logischen Schluß, man könne auf die beschriebene Art und Weise prüfen, ob zwei 16-Bit-Werte gleich sind. Und genau das geht nicht. Das Zero-Flag hat nämlich die unangenehme Eigenschaft, auch dann gesetzt zu sein, wenn der erste Wert bis zu 255 größer (!) ist als der zweite. Will man also zuverlässig auf Gleichheit testen, muß man die beiden Werte voneinander abziehen und nachsehen, ob das Ergebnis Null ist.

Hinweise zum Abtippen

Wir geloben Besserung und als Zeichen tätiger Reue haben wir Ihnen das Eintippen der DATAs leichter gemacht. An anderer Stelle in diesem Heft finden Sie das Programm »MSE«. Damit ist ein Vertippen nahezu ausgeschlossen, und schneller und bequemer geht's obendrein auch noch. Sie finden also auf der nächsten Seite nicht mehr den gewohnten Basic-Lader, sondern statt dessen eine Art Hexdump, das Sie mit »MSE« eintippen können. Wie das genau geht, erfahren Sie in dem zugehörigen Artikel. Nach Beendigung der Tipparbeit speichern Sie Ihr Werk ab. Laden Sie nun das SMON-Maschinenprogramm vom letzten Mal und starten Sie es mit SYS 49152. Hängen Sie den neuen Teil mit

»L "SMON TEIL 4"«

dahinter und speichern Sie das Ganze mit

S "@:SMON $C000" C000 CE09«

wieder ab. Sie haben dann das vollständige Programm (15 Blöcke auf der Diskette) zur Verfügung.

Wir hoffen, Ihnen mit SMON ein wirklich brauchbares Werkzeug an die Hand gegeben zu haben. Die vielen Zuschriften und Anrufe, die wir im Verlauf dieser Serie erhalten haben, bestätigen uns in dieser Auffassung. Viele Leser sind nach unserem Eindruck dabei, in die Maschinensprache einzusteigen und deren Möglichkeiten zu nutzen. Und genau das wollten wir erreichen.

Damit wären wir eigentlich am Ende dieser Serie angelangt und würden uns mit den üblichen guten Wünschen verabschieden. Aber die freien Bytes am Ende von SMON, immerhin von $CE09 bis $D000, also über 500 Byte, haben uns keine Ruhe gelassen. Als »Zugabe« werden wir Ihnen deshalb in der nächsten Ausgabe noch einen kleinen Diskettenmonitor vorstellen, der, in SMON eingebunden, dessen Möglichkeit nutzen kann. (N. Mann/D. Weineck/gk)

Hinweise zu Bild 1

Alle Eingaben erfolgen in der hexadezimalen Schreibweise. In Klammern angegebene Adreßeingaben können entfallen. SMON benutzt dann sinnvolle vorgegebene Werte. Bei allen Ausgabe-Befehlen ist gleichzeitig die Ausgabe auf einen Drucker möglich. Dazu werden die Befehle geSHIFTet eingegeben.

Bild 1. SMON-Befehlsübersicht

— **A 4000 (Assembler)**
symbolischer Assembler (Verarbeitung von Labels möglich) Startadresse $4000

— **B 4000 4200 (Basic-DATA)**
erzeugt Basic-DATA-Zeilen aus Maschinenprogramm im Bereich von $4000 - $41FF

— **C 4010 4200 4013 4000 4200 (Convert)**
in ein Programm, das von $4000 - $4200 im Speicher steht, soll bei $4010 ein 3-Byte-Befehl eingefügt werden. Dazu wird das Programm ab $4010 - $4200 auf die neue Adresse $4013 verschoben. Alle absoluten Adressen, die innerhalb des Programmbereichs ($4000 - $4200) stehen, werden umgerechnet, so daß die Sprungziele stimmen.

— **D 4000 (4100) (Disassembler)**
disassembliert den Bereich von $4000 (- $4100) mit Ausgabe der Hex-Werte. Änderungen sind durch Überschreiben der Befehle möglich.

— **F (Find)**
findet Zeichenketten (F), absolute Adressen (FA), relative Sprünge (FR), Tabellen (FT), Zeropageadressen (FZ) und Immediate-Befehle (FI)

— **G (4000) (Go)**
startet ein Maschinenprogramm, das bei $4000 im Speicher beginnt.

— **I 01 (I/O-Gerät)**
stellt die Gerätenummer für Floppy (08 oder 09) oder Datasette (01) ein.

— **K A000 (A500) (Kontrolle)**
zum schnellen Durchsuchen des Bereichs von $A000 (-$A500) nach ASCII-Zeichen (32 Byte pro Zeile). Änderungen sind durch Überschreiben der ASCII-Zeichen möglich.

— **L (4000) (Load)**
lädt ein Maschinenprogramm an die richtige oder eine angegebene Adresse ($4000)

— **M 4000 (4400) (Memory-Dump)**
gibt den Inhalt des Speichers von $4000 (- $43FF) in Hex-Bytes und ASCII-Code aus. Änderungen sind durch Überschreiben der Hex-Zahlen möglich.

— **O 4000 4500 AA (Occupy)**
füllt den Speicherbereich von $4000 - $4500 mit vorgegebenem Byte ($AA) aus

— **P 02 (Printer)**
setzt Geräteadresse 2 für Drucker

— **R (Register)**
zeigt die Registerinhalte und Flags an. Änderungen sind durch Überschreiben möglich.

— **S "Test" 4000 5000 (Save)**
speichert ein Programm von $4000 - $4FFF unter dem Namen 'Test' ab

— **TW (4000) (Trace Walk)**
führt auf Tastendruck den jeweils nächsten Maschinenbefehl aus und zeigt die Registerinhalte an. Subroutinen können in Echtzeit durchlaufen werden ('J'). Wird keine Startadresse eingegeben, beginnt 'TW' bei der letzten mit 'R' angezeigten Adresse.

— **TB 4010 05 (Trace Break)**
setzt einen Haltepunkt für den Schnellschrittmodus bei $4010. Der Schnellschrittmodus wird unterbrochen, nachdem $4010 zum 5. Mal erreicht worden ist.

— **TQ 4000 (Trace quick)**
Schnellschrittmodus, springt beim Erreichen eines Haltepunktes in den Einzelschrittmodus.

— **TS 4000 4020 (Trace stop)**
arbeitet ein Programm ab $4000 in Echtzeit ab und springt beim Erreichen von $4020 in die Registeranzeige. Von dort aus kann (nach evtl. Änderung der Register) mit »G« oder »TW« fortgefahren werden. »TS« arbeitet nur im RAM-Speicher.

— **V 6000 6200 4000 4100 4200 (Verschieben)**
ändert in einem Programm von $4100 - $41FF alle absoluten Adressen, die sich auf den Bereich von $6000 - $6200 beziehen, auf einen neuen Bereich, der bei $4000 beginnt.

— **W 4000 4300 5000 (Write)**
verschiebt den Speicherinhalt von $4000 -$42FF nach $5000 ohne Umrechnung der Adressen (zum Beispiel Tabellen)

— **X (Exit)**
springt aus dem Monitor-Programm ins Basic zurück

— **# 49152**
Dezimalzahl umrechnen

— **$ 002B**
4-stellige Hex-Zahl umrechnen

— **% 01101010**
8-stellige Binärzahl umrechnen

— **? 0344 + 5234**
Addition oder Subtraktion zweier 4-stelliger Hex-Zahlen

— **= 4000 5000 (Vergleich)**
vergleicht den Speicherinhalt ab $4000 mit dem ab $5000.

Listing 1. Der 4. Teil von SMON. Um dieses Programm einzutippen, verwenden Sie bitte den MSE, den Sie als Listing in diesem Heft finden.

```
 PROGRAMM : SMON TEIL 4  CBF1 CE09
---------------------------------



CBF1 : 68 68 20 CF FF C9 57 D0  DD
CBF9 : 03 4C 56 CD C9 42 D0 03  6A
CC01 : 4C D0 CD C9 51 D0 03 4C  A2
CC09 : 4F CD C9 53 F0 03 4C D1  18
CC11 : C2 20 8D C2 48 20 8D C2  E0
CC19 : 48 20 49 C2 A0 00 B1 FB  E5
CC21 : 8D BC 02 98 91 FB A9 36  AC
CC29 : 8D 16 03 A9 CC 8D 17 03  53
CC31 : A2 FC 4C EC C3 A2 03 68  30
CC39 : 9D 9A 02 CA 10 F9 68 68  48
CC41 : BA 8E AE 02 AD A8 02 85  61
CC49 : FC AD A9 02 85 FB AD BC  2F
CC51 : 02 A0 00 91 FB A9 14 8D  4E
CC59 : 16 03 A9 C2 8D 17 03 A9  A4
CC61 : 52 4C FF C2 20 51 C3 AD  28
CC69 : 11 D0 09 10 8D 11 D0 60  8C
CC71 : 8D AB 02 08 68 29 EF 8D  00
CC79 : AA 02 8E AC 02 8C AD 02  9D
CC81 : 68 18 69 01 8D A9 02 68  6F
CC89 : 69 00 8D A8 02 A9 80 8D  F5
CC91 : BC 02 D0 10 20 E5 CD 20  2D
CC99 : DD FD D8 A2 05 68 9D A8  5B
CCA1 : 02 CA 10 F9 AD 14 03 8D  EE
CCA9 : BB 02 AD 15 03 8D BA 02  FF
CCB1 : BA 8E AE 02 58 AD AA 02  40
CCB9 : 29 10 F0 08 20 65 CC A9  DB
CCC1 : 52 4C FF C2 2C BC 02 50  E3
CCC9 : 1F 38 AD A9 02 ED BD 02  2F
CCD1 : 8D B1 02 AD A8 02 ED BE  3D
CCD9 : 02 0D B1 02 D0 67 AD BF  8D
CCE1 : 02 D0 5F A9 80 8D BC 02  C4
CCE9 : 30 12 4E BC 02 90 CD AE  87
CCF1 : AE 02 9A A9 CC 48 A9 70  13
CCF9 : 48 4C BA CD 20 65 CC A9  83
CD01 : A8 85 FB A9 02 85 FC 20  20
CD09 : 4C C3 A0 00 B1 FB 20 2A  2F
CD11 : C3 C8 C0 07 F0 09 C0 01  A6
CD19 : F0 F2 20 4C C3 D0 ED AD  EA
CD21 : A9 02 AE A8 02 85 FB 86  D5
CD29 : FC 20 49 C3 20 CB C4 20  B4
CD31 : C7 C5 20 E4 FF F0 FB C9  8A
CD39 : 4A D0 0A A9 01 8D BC 02  17
CD41 : D0 2F CE BF 02 A5 91 C9  7B
CD49 : 7F D0 26 4C BD CC 20 F2  EC
CD51 : CD A9 40 D0 0A 20 F2 CD  26
CD59 : 08 68 8D AA 02 A9 80 8D  D9
CD61 : BC 02 BA 8E AE 02 20 49  AD
CD69 : C2 20 65 CC AD BC 02 F0  D9
CD71 : 37 A2 00 AD 11 D0 A8 29  3C
CD79 : 10 F0 10 98 29 EF 8D 11  83
CD81 : D0 EA EA A0 0C CA D0 FD  EB
CD89 : 88 D0 FA 78 A9 47 8D 04  5A
CD91 : DC 8E 05 DC AD 0E DC 29  A2
CD99 : 80 09 11 8D 0E DC A9 95  2D
CDA1 : A2 CC 8D BB 02 8E BA 02  08
CDA9 : AE AE 02 CA 78 AD BB 02  6A
CDB1 : AE BA 02 8D 14 03 8E 15  AC
CDB9 : 03 AD A8 02 48 AD A9 02  9A
CDC1 : 48 AD AA 02 48 AD AB 02  6F
CDC9 : AE AC 02 AC AD 02 40 20  10
CDD1 : 8D C2 8D BE 02 20 8D C2  D7
CDD9 : 8D BD 02 20 8D C2 8D BF  6E
CDE1 : 02 4C D6 C2 AD B8 02 AE  1D
CDE9 : B9 02 8D 14 03 8E 15 03  88
CDF1 : 60 AD 14 03 AE 15 03 8D  48
CDF9 : B8 02 8E B9 02 A9 95 8D  6C
CE01 : 16 03 A9 CC 8D 17 03 60  FB
```