```
,ef,20,73,00,b0,ea,4c,a0,c8,00,00,78,a2
304 data2d,a0,7e,20,56,fd,a2,0b,bd,21,7e
,9d,00,03,ca,10,f7,20,f9,fd,20,18,e5,a9
305 data0e,8d,0f,90,a9,01,8d,86,02,ea,60
,4c,f3,dc,4c,c2,73,a9,00,85,0d,20,73,00
306 data90,f1,c9,e1,b0,f0,c9,b4,90,ec,20
,0e,7f,4c,8d,cd,4c,08,cf,4c,a5,c9,4c,12
307 datac8,20,73,00,20,fd,7e,4c,ae,c7,f0
,ce,aa,10,ec,c9,cb,f0,eb,c9,a3,90,04,c9
308 datae1,90,dd,0a,8d,16,7f,20,73,00,6c
,c8,7c,00,00,00,a6,7a,a0,04,84,0f,bd,00
309 data02,10,07,c9,ff,f0,3e,e8,d0,f4,c9
,20,f0,37,85,08,c9,22,f0,56,24,0f,70,2d
310 datac9,3f,d0,04,a9,99,d0,25,c9,30,90
,04,c9,3c,90,1d,84,71,a0,00,84,0b,88,86
311 data7a,ca,c8,e8,bd,00,02,38,f9,9e,c0
,f0,f5,c9,80,d0,30,05,0b,a4,71,e8,c8,99
312 datafb,01,b9,fb,01,f0,59,38,e9,3a,f0
,04,c9,49,d0,02,85,0f,38,e9,55,d0,9f,85
313 data08,bd,00,02,f0,df,c5,08,f0,db,c8
,99,fb,01,e8,d0,f0,a6,7a,e6,0b,c8,b9,9d
314 datac0,10,fa,b9,9e,c0,d0,b4,a0,ff,ca
,c8,e8,bd,00,02,38,f9,00,7b,f0,f5,c9,80
315 datad0,02,f0,ad,a6,7a,e6,0b,c8,b9,ff
,7a,10,fa,b9,00,7b,d0,e2,bd,00,02,10,9b
316 data4c,09,c6,4c,ef,c6,b1,5f,4c,1a,c7
,c9,cc,90,f7,c9,ff,f0,f3,24,0f,30,ef,84
317 data49,38,e9,cb,aa,a0,ff,ca,f0,08,c8
,b9,00,7b,10,fa,30,f5,c8,b9,00,7b,30,d3
318 data20,47,cb,d0,f5
319 rem
320 rem-------------------------- pruef
summen ------------------------------
321 rem
322 data 31231
323 data 25675
324 data 28397
325 data 31427
326 data 30997
327 data 33043
328 data 30573
329 data 30423
330 data 28240
331 data 26807
332 data 21966
333 data 26340
334 data 31124
335 data 29634
336 data 27953

ready.
```

Listing »Mathematical Basic« (Schluß)

Ohne gutes Werkzeug geht es nicht:

SMON Teil 2

Der Maschinensprache-Monitor SMON wird immer leistungsfähiger. Dieser 2. Teil erweitert ihn um wichtige Ausgabe-Routinen, läßt das Verschieben eines Programms mit und ohne Adreßumrechnung zu und kann Zahlen vom Dezimal- in das Binärsystem und umgekehrt umrechnen.

Wir hoffen, daß wir Ihnen in der letzten Ausgabe nicht zuviel zugemutet haben, und daß sich Ihre wunden Finger inzwischen wieder erholen konnten. Bestimmt haben Sie im vergangenen Monat schon eifrig mit dem neuen Monitor gearbeitet und sind inzwischen mit den bisherigen Befehlen vertraut. Denn nun folgt der zweite Teil und mit diesem natürlich wieder einige neue Befehle, die es zu erklären gilt.
Und das bieten wir Ihnen heute:
I/O-SET, LOAD, SAVE, PRINTER-SET, die verschiedenen Zahlenumrechnungen (HEX-DEZ-BIN-ADD-SUB), OCCUPY, CONVERT, VERSCHIEBEN und WRITE.

I/O-SET

I 01 legt die Device-Nummer für LOAD und SAVE auf 1 (Kassette). Jedes Laden und Abspeichern erfolgt jetzt auf das angegebene Gerät. Die voreingestellte Device-Nummer ist 8 (für die Floppy also: I 08). Wenn Sie nur mit der Floppy arbeiten, brauchen Sie diesen Befehl also nicht.

LOAD

L "Name" lädt ein Programm vom angegebenen Gerät (wie oben beschrieben) an die Originaladresse in den Speicher. Die Basic-Zeiger bleiben bei diesem Ladevorgang unbeeinflußt, das heißt, sie werden nicht verändert.

Beispiel: Unser Monitor soll an seiner Originaladresse ($C000) im Speicher stehen. Also brauchen Sie ihn nur mit »L "SMON"« zu laden, damit er dort erscheint. Wenn Sie einmal ein Programm an eine andere als die Originaladresse laden wollen, dann bietet Ihnen SMON dazu folgende Möglichkeit: 'L "Name" ADRESSE lädt ein Programm an die angegebene Adresse. Nehmen Sie doch bitte noch einmal unser letztes Test-Programm und geben es mit dem Assembler ab Adresse $4000 ein. Speichern Sie es mit »'S "SUPERTEST" 4000 4023« ab und laden es dann
1. an die Originaladresse (L "SUPERTEST") und

2. an eine andere Adresse (mit L "SUPERTEST"5000 zum Beispiel nach $5000).

Schauen Sie sich danach mit dem Disassembler-Befehl beide Routinen einmal an. Sie werden feststellen, daß beide Programme zwar bis auf die BRANCH-Befehle gleich aussehen, daß das Programm in $5000 aber nicht funktionieren kann, da es eine falsche Adresse verwendet (5002 LDA 400E,Y). Ein anderes Beispiel dazu: Ein Autostart-Programm beginnt bei $0120, läßt sich aber in diesem Bereich nicht untersuchen, da dort der Prozessor-STACK (im Bereich von $0100 bis $01FF) liegt, der vom Prozessor selbst ständig verändert wird. Wenn Sie nun L "Name" 4120 eingeben, befindet sich das Programm anschließend bei $4120 (nicht an der Originaladresse $0120) und Sie können es ohne Einschränkungen — von den falschen Absolut-Adressen abgesehen — disassemblieren.

SAVE

S "Name", ANFADR ENDADR speichert ein Programm von ANFADR bis ENDADR-1 unter »Name« auf die Floppy ab, da diese — wie wir ja inzwischen wissen — das voreingestellte Gerät ist. Wenn Sie auf Kassette abspeichern wollen, setzen Sie vorher mit »I 01« die Device-Nummer auf 1.

Beispiel: S"SUPERTEST"4000 4020 speichert das Programm mit dem Namen »SUPERTEST« (es steht im Speicher von $4000 bis $401F) auf Diskette ab. Bitte beachten Sie auch bei diesem Befehl, daß die Endadresse auf das nächste Byte hinter dem Programm gesetzt wird.

PRINTER-SET

P 02 setzt die Primäradresse für den Drucker auf 2. Voreingestellt ist hier die 4 als Gerätenummer (zum Beispiel für Commodore-Drucker). Vielleicht haben Sie es ja schon bemerkt: Bei allen Ausgabe-Befehlen (wie D, M etc.) können Sie auch den Drucker ansprechen, wenn Sie das Kommando geshiftet eingeben. Die Ausgabe erfolgt dann gleichzeitig auf Bildschirm und Drucker. (Beachten Sie bitte die Änderung für die Druckerausgabe am Schluß des Artikels.)

Ein bißchen Rechnerei

Die folgende Befehlsgruppe enthält Befehle zur Zahlenumrechnung. Sie wissen ja: Der Mensch mit seinen zehn Fingern neigt eher zur dezimalen Rechenweise, aber der Computer bevorzugt das Binärsystem, weil er nur zwei Finger hat (siehe Netzstecker). Ein Kompromiß ist das Hexadezimalsystem, denn das versteht keiner von beiden. Um Verständnis-Schwierigkeiten mit Ihrem Liebling aus dem Weg zu gehen, haben Sie aber SMON.

UMRECHNUNG DEZ→HEX

(Dezimalzahl) rechnet die Dezimalzahl in die entsprechende Hexadezimalzahl um. Hierbei können Sie die Eingabe in beliebiger Weise vornehmen, da SMON Zahlen bis 65535 umrechnet. Beispiel: #12, #144, #3456, #65533 und so weiter.

UMRECHNUNG HEX→DEZ

$ (Hexadezimalzahl) rechnet die Hexadezimalzahl in die entsprechende Dezimalzahl um. Die Eingabe muß hierbei zweistellig beziehungsweise vierstellig erfolgen. Ist diese Zahl kleiner als $100 (=255), wird zusätzlich auch der Binärwert ausgegeben.

Beispiel: $12, $0012, $0D, $FFD2, etc. In den ersten drei Beispielen erfolgt die Anzeige auch in binärer Form.

UMRECHNUNG BINÄR→HEX,DEZ

% (Binärzahl (achtstellig) rechnet die Binärzahl in die entsprechenden Hexa- und Dezimalzahlen um. Bei diesem Befehl müssen Sie genau acht Binärzahlen eingeben. Falls Sie einmal versehentlich mehr eingeben sollten, werden nur die ersten acht zur Umrechnung herangezogen. Beispiel: %00011111, %10101011

ADD-SUB

? 2340+156D berechnet die Summe der beiden vier (!)-stelligen Hex-Zahlen. Neben der Addition ist auch Subtraktion möglich.

Programme auf dem Rangierbahnhof

OCCUPY (Besetzen)

O (ANFADR ENDADR HEX-Wert) belegt den angebenen Bereich mit dem vorgegebenen HEX-Wert. Beispiel:O 5000 8000 00 füllt den Bereich von $5000 bis $7FFF mit Nullen.

Man kann mit »OCCUPY« aber nicht nur Speicherbereiche löschen, sondern auch mit beliebigen Werten belegen. Häufig hat man das Problem, festzustellen, welcher Speicherplatz von einem Programm wirklich benutzt wird. Wir füllen den in Frage kommenden Bereich dann zuerst zum Beispiel mit »AA« und laden dann unser Programm. Probieren Sie bitte das folgende Beispiel: Füllen Sie den Speicherbereich von $3000 bis $6000 mit $AA, und laden Sie dann unser SUPERTEST-Programm. Beim Disassemblieren können Sie erkennen, daß unser kleines Programm exakt zwischen vielen AAs eingebettet ist.

WRITE

W (ANFADRalt ENDADRalt ANFADRneu) verschiebt den Speicherbereich von ANFADRalt bis ENDADRalt nach ANFADRneu ohne Umrechnung der Adressen! Unser kleines Testprogramm möge noch einmal als Beispiel dienen: W 4000 4020 6000 verschiebt das oben angesprochene Programm von $4000 nach $6000.

Hierbei werden weder die absoluten Adressen umgerechnet noch die Tabellen geändert. Letzteres ist sicherlich erwünscht, aber denken Sie daran, daß das verschobene Programm nun nicht mehr lauffähig ist, da die absoluten Adressen nicht mehr stimmen (zum Beispiel bei dem Befehl LDA 400E,Y). Falls Sie jetzt »G6000« eingeben, um das Programm zu starten, werden Sie sich sicherlich wundern, daß es dennoch läuft. Doch löschen Sie einmal das Programm in $4000 (mit »O4000 4100 AA«) und starten das Programm in $6000 noch einmal! Seltsam, nicht? Abhilfe schafft der nächste Befehl.

VARIATION

V (ANFADRalt ENDADRalt ANFADRneu ANFADR ENDADR) rechnet alle absoluten Adressen im Bereich von ANFADR bis ENDADR, die sich auf ANFADRalt bis ENDADRalt beziehen, auf ANFADRneu um. Kompliziert? Nicht, wenn Sie sich klarmachen, daß die ersten drei Adressen exakt den Eingaben beim »W«-Befehl entsprechen. Neu hinzukommen nur die beiden Adressen für den Bereich, in dem die Änderung tatsächlich erfolgt.

Um unser mit »W« schon verschobenes Programm auch wieder lauffähig zu machen, geben Sie folgendes ein: V 4000

4020 6000 6000 600E. Damit werden alle Absolutadressen, die im Bereich von $6000 bis $600E — dahinter steht die Tabelle — liegen und sich bisher auf $4000 bis $4020 bezogen haben, auf den neuen Bereich umgerechnet. Probieren geht wie immer über kapieren.

Eine Zusammenfassung dieser beiden Befehle ermöglicht:

CONVERTIEREN
verschieben eines Programmes mit Adreßumrechnung)
C (ANFADRalt ENDADRalt ANFADRneu ANFADRges ENDADRges) verschiebt das Programm von ANFADRalt bis ENDADRalt zur ANFADRneu und zwar mit Umrechnung der Adressen zwischen ANFADRges und ENDADRges

An unserem kleinen Testprogramm läßt sich wieder einmal demonstrieren, wie der Befehl eingesetzt wird. Laden Sie es also mit »L"SUPERTEST"« und schauen es mit »D 4000« an. Jetzt wollen wir an der Adresse $4008 einen 3-Byte-Befehl einfügen: C 4008 4020 400B 4000 4011 verschiebt das Programm von $4008 bis $4020 zur neuen Anfangsadresse $400B. Dabei werden im Bereich von $4000 bis $4011 (neue Endadresse des »aktiven« Programmes!) die Sprungadressen umgerechnet. Nun können Sie ab Adresse $4008 einen 3-Byte-Befehl einfügen, zum Beispiel STY 0286. Dazu geben Sie bitte ein:
A 4008
4008 STY 0286
F

Überzeugen Sie sich davon, daß SMON die Befehle korrekt umgerechnet hat, indem Sie unser Beispiel disassemblieren (D 4000) und anschließend mit G 4000 starten. Besitzer eines Farbmonitors werden in helle Begeisterung ausbrechen. Vorsicht ist geboten, wenn Tabellen oder Text vorhanden sind. SMON wird versuchen, diese als Befehle zu disassemblieren und gegebenenfalls umzurechnen. Dabei können unvorhersehbare Verfälschungen auftreten. Aus diesem Grunde ist im Beispiel die Endadresse des zu ändernden Bereiches auf $4011 und nicht etwa auf $4023 gelegt worden. Wenn Sie größere Programme zu verschieben haben, sollten Sie die Kommandos W und V anwenden, beziehungsweise einen Assembler einsetzen, der es Ihnen gestattet, beliebige Einfügungen, Verschiebungen und sonstige Änderungen vorzunehmen. Das C-Kommando eignet sich in erster Linie für kleinere Änderungen innerhalb eines Programms.

Der Blick hinter die Kulissen

Wie beim letzten Mal wollen wir noch einen kleinen Blick auf das Programm werfen. Wir haben zwei häufig vorkommende Programmteile ausgewählt. Wenn Sie nach erfolgreichem Eintippen der neuen DATAs einmal mit »D C84F« die »LOADSAVE«-Routine listen, sehen Sie, daß diese sehr wenig Platz beansprucht. Das Geheimnis dieser Beschränkung liegt in der Tatsache begründet, daß wir hier auf Betriebssystem-Routinen zurückgegriffen haben. Doch dazu nachher mehr; erst einmal die angesprochene Routine von Anfang an:

```
LOADSAVE    LDY     #$02
            STY     *$BC
            DEY
            STY     *$B9
            STY     *$BB
            DEY
            STY     *$B7
```

Die Speicherstellen $BB/$BC enthalten jetzt die Adresse $0201, also den Beginn des Basic-Eingabepuffers. In $B9 befindet sich der Wert 01, das bedeutet, daß die Sekundäradresse für absolutes Laden voreingestellt ist. Die Speicherstelle $B7 enthält die Länge des Dateinamens, hier erst einmal 0.

```
            JSR     GETCHRERR
            CMP     #'"'
            BNE     LSERROR
```

Überprüft, ob Anführungsstriche eingegeben wurden. Falls nicht, springt unser Programm in die Routine »LSERROR« und bricht ab.

```
LS1         JSR     GETCHRERR
            STA     ($BB),Y
            INY
            INC     *$B7
            CMP     #'"'
            BNE     LS1
```

In diesem Programmteil wird der Filename eingelesen und in die Adresse gespeichert, die in $BB/$BC enthalten ist ($0201). Gleichzeitig wird $B7 als Zähler für die Namenlänge so lange erhöht, bis das zweite Anführungszeichen auftaucht. Damit ist der Filename gespeichert, beginnend bei $0201.

```
            DEC     *$B7
            LDA     I0.NR
            STA     *$BA
            LDA     *COMMAND
            CMP     #'S'
            BEQ     SAVE
```

Da die Namenlänge um eins zu groß geraten ist (das letzte Zeichen war ein »"«), muß sie dekrementiert werden. Die gewählte oder voreingestellte I/O-Nummer (Device-Nummer) soll in $BA gespeichert werden, damit die Betriebssystemroutine nachher das richtige Gerät anspricht. Zum Abschluß überprüft der Compare-Befehl, ob das Kommando »S« gewählt ist, um dann dorthin zu verzweigen.

```
LOAD        JSR     GETRET
            BEQ     LOAD1
            LDX     #$C3
            JSR     GETADR
            LDA     #$00
            STA     *$B9
```

Wir sind nun an der Stelle des Befehls angelangt, an der sich herausstellen muß, ob das Programm an seine Originaladresse (absolut) oder an eine andere Adresse geladen werden soll. Die Unterroutine »GETRET« prüft, ob unmittelbar nach dem Namen ein »RETURN« folgt und führt eine Verzweigung nach »LOAD1« durch, falls dieses eintritt. Ansonsten holen wir uns die Adresse und laden das vorgesehene Programm dorthin, nachdem in Speicherstelle * $B9 eine Null gespeichert ist, da ein absolutes Laden nicht erfolgt. Die Routine »GETADR« ist so aufgebaut, daß sie die nächsten 2 Bytes an die mit dem X-Register gewählte Stelle in der Zeropage ablegt. Dann führen wir ebenfalls »LOAD1« durch.

```
LOAD1   LDA   #$00
        JMP   ($0330)
SAVE    LDX   #$C1
        JSR   GETADR
        LDX   #$AE
        JSR   GETADR
        JMP   ($0332)
```

In »LOAD1« erfolgt der indirekte Sprung über $0330 in die LOAD-Routine des Betriebssystems.

Die SAVE-Routine erfragt vorher noch die fehlenden Adressen (Anfangs- und Endadresse des Programmes, das gespeichert werden soll), speichert sie nach $C1/C2 und $AE/AF und springt dann in die SAVE-Routine. Noch ein Wort zu den angesprochenen Betriebssystem-Routinen: Mittlerweile gibt es für den C 64 mindestens drei verschiedene Versionen des Betriebssystems von Commodore. Es sind zwar meist nur kleine Änderungen, aber die können fatale Folgen haben, wenn sich die Einsprungadressen ändern. Deshalb gibt es einen besonderen Bereich, das KERNAL, der einen Sprungverteiler für die wichtigsten Routinen enthält. Dieser wird grundsätzlich nie geändert. Beziehen Sie deshalb Ihre Einsprungadressen immer auf die KERNAL-Routinen, um sicher zu sein, daß Ihr Programm auch noch mit der zwölften Version des Betriebssystems läuft. Die KERNAL-Einsprünge stehen ganz hinten ab $FF81 im Speicher.

Als zweites ein Vergleich, der in Maschinenprogrammen häufig und in allen Variationen auftaucht: Es handelt sich dabei um den Vergleich zweier Adressen. Nun sind Adressen leider 16-Bit-Werte, unser Prozessor aber kann nur 8 Bit auf einmal verarbeiten. Gehen wir einmal von folgenden Bedingungen aus: Ein Programm soll von $4000 bis $4020 gelistet werden. Die Zeiger für das Ende befinden sich in Speicherstelle ENDLO (Lowbyte) und ENDHI (Highbyte). »PCL« (Programm-Counter-Low) und »PCH« (Programm-Counter-High) geben den augenblicklichen Stand des Programmes an. Dann erfolgt die Abfrage auf erreichtes Ende mit dieser Befehlsfolge:

```
VERGL   LDA   PCL
        CMP   ENDLO
        LDA   PCH
        SBC   ENDHI
        BCS   FERTIG
WEITER  JMP   AUSGABE
FERTIG  JMP   ENDE
```

Solange PCL und PCH kleiner sind als die Endwerte geht das Programm »WEITER«.

Sobald aber PCL und PCH die Werte von ENDLO und ENDHI erreicht haben, wird das Carry-Flag gesetzt und die Abfage mit BCS FERTIG würde das Auflisten anhalten. Daß es bei der Anwendung einige Probleme geben kann, sieht man daran, daß die Ausgabe auch schon unterbrochen wird, wenn gerade erst das Programmende erreicht ist. (Der letzte Befehl könnte »unter den Tisch fallen«.) Aber kein Problem ohne Problemlösung — und natürlich ohne weitere Probleme, die Sie aber mit ein bißchen Nachdenken sicher selbst lösen können.

Hinweise zum Abtippen

Tippen Sie das Ladeprogramm sorgfältig ab, speichern Sie es (!) und starten Sie mit RUN. Sollte es sich wider Erwarten auf Anhieb mit READY melden, haben Sie das Schlimmste geschafft. Ansonsten beseitigen Sie nun alle Fehler bis es zum READY durchläuft. Jetzt laden Sie das alte Ladeprogramm aus der letzten Ausgabe und starten es. Nach dem READY starten Sie SMON mit SYS 49152. Als erstes probieren Sie nun den Befehl »S«, um SMON selbst abzuspeichern, diesmal nicht mehr als Basic-Lader, sondern als Maschinenprogramm.

S "SMON $C000" C000 CAB7

SMON belegt jetzt 11 Blöcke auf der Diskette. Ab jetzt können Sie SMON direkt mit »LOAD "SMON $C000",8,1« laden und mit SYS 49152 starten.

Noch zwei Hinweise in eigener Sache: Einige wenige (!) Leser haben uns darauf aufmerksam gemacht, daß die Druckerausgabe auf bestimmten (exotischen) Druckern bisweilen kleinere Unzulänglichkeiten aufweist. Kurz und schlecht, uns ist in der letzten Folge ein Programmierfehler unterlaufen: Beim Disassemblieren verschwindet die letzte Zeile vor dem Strich (----) im Drucker und ward nicht mehr gesehen. So etwas passiert, wenn man kurz vor Redaktionsschluß noch auf die Schnelle kleine »Verbesserungen« vornimmt.

Für die Korrektur ist folgendes notwendig: Listen Sie mit »M C56C C57B« zwei Zeilen, gehen mit dem Cursor in die betreffenden Zeilen und geben folgende Änderung ein:

:C56C	**09**	C9	**30**	F0	**05**	C9	21	**DO**
	—		—		—			—
:C574	**11**	**EA**	**20**	**94**	**C4**	20	51	C3
	—	—	—	—	—			

Nur die fetten Werte müssen geändert werden, alle anderen können Sie stehen lassen. Denken Sie bitte bei jeder Änderung daran, daß Sie die Zeile nur mit Drücken der RETURN-Taste an den Computer übergeben. Zur Probe können Sie ja noch einmal listen...

Wir haben nach dem letzten Artikel eine Menge Anrufe erhalten, von Lesern, die größtenteils Schwierigkeiten beim Eintippen der DATAs beziehungsweise bei der Fehlersuche hatten. Deswegen hier Hinweise zu den häufigsten Problemen:

1. Wenn nach Beendigung der Tipparbeit nach dem RUN eine Fehlermeldung ».... ERROR in 40« (oder 70) erfolgt, dann ist sicherlich nicht die Zeile 40 oder 70 daran schuld, sondern Sie haben aller Wahrscheinlichkeit nach einen Wert (ein »Datum«) falsch eingetippt. Der Computer bringt eine Fehlermeldung, wenn er beim POKE-Befehl auf eine Kommazahl trifft oder einen anderen nicht POKEbaren Wert. Dafür gibt es — neben schlichten Tippfehlern — mehrere Möglichkeiten: Es kann ein Komma fehlen oder durch einen Punkt ersetzt worden sein. Gerade dies ist nämlich auf dem Bildschirm sehr schlecht zu erkennen.
2. Überprüfen Sie nach dem Programmabbruch anhand des Direktbefehls »PRINT I« in welchem Block (+1) der Fehler steckt. Also bei der Antwort »1« steckt der Fehler in Block 2.
3. Der Direktbefehl »PRINT A« zeigt Ihnen den Wert, der den Fehler verursacht hat.

Versuchen Sie es erst einmal mit dieser kleinen Hilfe. Übrigens ist unser Listing mit 99prozentiger Wahrscheinlichkeit fehlerfrei, von uns und der Redaktion mehrfach durchprobiert. Das Dreckfuhlerteufelchen hat kaum eine Chance, da das Listing direkt von der Diskette auf den Drucker läuft.

Ich hoffe, daß Sie bis jetzt nicht in Ihren Bemühungen nachgelassen haben, möglichst häufig die verschiedensten Befehle zu probieren. Sie wissen doch: Nur die Übung macht den Meister — und das gilt speziell für die Maschinensprache. In der nächsten Ausgabe bekommen Sie dann die letzten Raffinessen des SMON, der dann komplett ist.

(Norfried Mann/gk)

```
10 rem ***************************
20 rem *                         *
30 rem *    smon                 *
40 rem * von n.mann & d.weineck *
50 rem *     fleetrade 40        *
60 rem *     2800 bremen         *
70 rem *   tel. 0421 / 493090    *
80 rem *        0421 / 231401    *
90 rem *                         *
100 rem ***************************
110 fori=0to2:reada:pr(i)=a:next
120 sa=51261:i=0
130 pa=sa+256*i:ch=0
140 forj=0to255:reada:pokepa+j,a:ch=ch+a
:next
150 remifch<>pr(i)then200
160 i=i+1:ifi<2then130
170 pa=pa+256:ch=0
180 forj=0to121:reada:pokepa+j,a:ch=ch+a
:next
190 ifch=pr(i)thenend
200 print"fehler in block"i+1:end
210 rem
220 rem *** blockpruefsummen ***
230 rem
240 data34652,35523,16258
250 rem
260 rem *** block 1 ***
270 rem
280 data32,141,194,141,175,2,96,32,141,1
94,141,176,2,96,76,209,194,160,2,132
290 data188,136,132,185,132,187,136,132,
183,32,202,194,201,34,208,234,32,202
300 data194,145,187,200,230,183,201,34,2
08,244,198,183,173,176,2,133,186,165
310 data172,201,83,240,19,32,194,194,240
,9,162,195,32,128,194,169,0,133,185
320 data169,0,108,48,3,162,193,32,128,19
4,162,174,32,128,194,108,50,3,32,126
330 data194,32,202,194,73,2,74,74,8,32,1
28,194,32,81,195,40,176,12,165,253
340 data101,251,170,165,254,101,252,56,1
76,9,165,251,229,253,170,165,252,229
350 data254,168,138,132,252,133,251,132,
98,133,99,8,169,0,133,211,32,117,198
360 data165,252,208,15,32,73,195,165,251
,32,42,195,165,251,32,208,195,240,3
370 data32,35,195,32,76,195,162,144,165,
1,141,177,2,169,55,133,1,40,32,73,188
380 data32,221,189,174,177,2,134,1,76,86
,195,32,141,194,170,164,211,177,209
390 data73,32,240,163,138,168,32,154,194
,56,176,169,32,184,194,160,8,72,32
400 data202,194,201,49,104,42,136,208,24
5,240,235,32,184,194,162,0,138,134
410 data251,133,252,168,32,207,255,201
420 rem
430 rem *** block 2 ***
440 rem
450 data58,176,132,233,47,176,4,56,76,19
6,200,133,253,6,251,38,252,165,252
460 data133,254,165,251,10,38,254,10,38,
254,24,101,251,8,24,101,253,170,165
470 data254,101,252,40,105,0,76,52,201,3
2,122,194,169,55,133,1,162,4,189,135
480 data192,149,170,202,16,248,32,81,195
,166,170,165,171,32,205,189,230,170
490 data208,2,230,171,169,68,32,210,255,
169,193,32,210,255,160,0,177,251,132
500 data98,133,99,32,209,189,32,99,196,1
62,3,176,10,169,44,166,211,224,73,144
510 data227,162,9,134,198,189,125,192,15
7,118,2,202,208,247,76,110,195,32,122
520 data194,32,141,194,162,0,129,251,72,
32,99,196,104,144,247,96,32,90,194
530 data165,166,208,2,198,167,198,166,32
,48,202,134,181,160,2,144,4,162,2,160
540 data0,24,165,166,101,174,133,170,165
,167,101,175,133,171,161,164,129,168
550 data65,168,5,181,133,181,165,164,197
,166,165,165,229,167,176,29,24,181
560 data164,121,107,192,149,164,181,165,
121,108,192,149,165,138,24,105,4,170
570 data201,7,144,232,233,8,170,176,207,
165,181,240,15,76,209,194,56,162,254
580 data181,170,245,166,149,176,232,208,
247,96
590 rem
600 rem *** block 3 ***
610 rem
620 data32,98,202,76,214,201,76,98,202,1
97,167,208,2,228,166,176,19,197,165
630 data208,2,228,164,144,11,133,180,138
,24,101,174,170,165,180,101,175,96
640 data32,90,194,32,122,194,32,48,202,3
2,203,196,200,169,16,36,171,240,38
650 data166,251,165,252,32,70,202,134,17
0,177,251,133,181,32,74,197,160,1,32
660 data70,202,202,138,24,229,170,145,25
1,69,181,16,25,32,81,195,32,35,195
670 data36,171,16,15,177,251,170,200,177
,251,32,70,202,145,251,138,136,145
680 data251,32,106,198,32,102,196,144,18
1,96

ready.
```

Listing »SMON Teil 2« (Basic-Lader)