Von Basic zu Assembler (Teil 7)

Neben einer Anzahl von Integer-Routinen und einer Technik zum Schreiben von Basic-Erweiterungen beschäftigen wir uns mit Assembler-Programmen, die sich selbst verändern.

Selbstmodifizierende Programme — also Programme, die sich im Verlauf der Abarbeitung selbst verändern — sind dem einen ein Graus, dem andern aber die Essenz der Raffinesse. Welcher Ansicht man auch immer sein mag: Es sind mit dieser Technik recht interessante Dinge möglich, die auf andere Weise nicht oder nur schwer realisierbar wären.

Programm und Daten

Wie unterscheidet unser Computer Programme und Daten? Sehen wir uns zuerst einmal an, wie das in Basic aussieht: Beide (Daten und Programme) werden im RAM streng voneinander getrennt (beim C 128 liegen sie sogar in unterschiedlichen Speicherbänken) und völlig unterschiedlich verwaltet. Deshalb ist die Selbstveränderung von Basic-Programmen auch mit allerlei Tricks verbunden, die entweder über POKEs den Programmspeicher beeinflussen oder im programmierten Direktmodus arbeiten. Ein simples Basic-Beispiel zeigt Listing 1:

Dieses Programm für den C 64 (bei anderen Computern muß die Adresse in Zeile 50 entsprechend geändert werden) verändert während des Programmablaufes die Speicherstelle 2112. Dort befindet sich der Buchstabe A im PRINT-Befehl der Zeile 30. Durch den POKE-Befehl gelangt nach dem A ein B, dann ein C und so weiter in das PRINT-Argument. Das sehen Sie dann, wenn Sie sich nach dem Ablauf des Programms mit LIST noch einmal die PRINT-Anweisung ansehen: Das A ist verschwunden, statt dessen ist dort ein Grafikzeichen (bei eingeschalteter Groß- und Kleinschreibung) oder der griechische Buchstabe Pi (bei Großschreibung) zu finden. Die andere Technik, also die, die im programmierten Direktmodus arbeitet, bedient sich des Tastaturpuffers. Falls Sie darüber mehr wissen möchten, dann lesen Sie bitte den Artikel »Lernen Sie Ihren Commodore 64 kennen«, Teil 4, in der Ausgabe 8/85 der Zeitschrift Happy-Computer, Seite 45ff. C 128-Benutzer sollten die Ausgabe 7/86 des 64'er-Magazins auf Seite 85 aufschlagen: Dort sind allerlei Verwendungsmöglichkeiten dieser Technik für den großen Bruder des C 64 vorgeführt. Soweit also das Ganze in Basic, wie verhält es sich in Assembler?

Hier existiert für den Computer nur eine lange Straße aufeinanderfolgender Speicherzellen. Der Zentralprozesser orientiert sich am Programmzähler, in dem sich die gerade aktuelle Anschrift befindet. In jeder Hausnummer findet die CPU irgendeinen Code, der sie veranlaßt, darauf zu reagieren. Alle derartigen Codes führen zu Veränderungen von Speicherinhalten — und sei es auch nur das Hochzählen des Programmzählers beim NOP-Befehl, das Chaos beim Programmabsturz oder auch das Eintragen von ASCII-Werten in den Bildschirmspeicher. Mal liegen diese Veränderungen weit weg vom Programm-Code, mal näher dran: Nichts hindert uns, auch in dem Speicherteil Änderungen vorzunehmen, in dem das Programm abgelegt ist, was uns mit Assemblern wie dem Hypra-Ass leicht fällt. Listing 2 zeigt, wie man Vergleichbares in Maschinensprache erreichen kann:

Das Programm ist für die älteren Versionen des C 64 geschrieben — daher die Belegung des Bildschirmfarbspeichers —, läuft aber auch auf den anderen Versionen, bei denen man die Zeilen, die sich auf die Farben beziehen, weglassen kann. Erinnern Sie sich bitte an die Art, wie der 6502 und seine kompatiblen Nachkommen Adressen im Speicher ablegen: Wenn wir ein Assemblerprogramm schreiben:

```
STX  $D800
```

dann findet sich im Speicher die Code-Folge:

Speicherstelle	Code	Bedeutung
Farb	8E	Code für absolutes STX
Farb+1	00	LSB der Adresse $D800
Farb+2	D8	MSB der Adresse $D800

Deshalb erhöhen wir Farb + 1 und Bild + 1.

Ebenso wie im Basic-Beispiel zeigt sich auch im Listing 2 ein Nachteil dieser Art der Programmierung: Das Programm kann kein zweites Mal gestartet werden — eben weil wir es verändert haben. Jedenfalls leistet es beim Neustart nicht mehr genau dasselbe. Sehen Sie sich nach dem Programmdurchlauf einmal das Disassemblerlisting an, dann finden Sie in den veränderten Zeilen:

```
CODE    LDA  #$00
BILD    STA  $04FF
FARB    STX  $D8FF
```

Beim Starten dieses veränderten Programms wird zuerst der Klammeraffe (das ist das Zeichen mit dem Code 00) in die Bildschirmspeicherstelle $04FF geschrieben. Erst danach läuft alles seinen gewohnten Gang, weil $FF+1 als $00 verstanden wird. Im Falle dieses Programms hätten wir die Schwierigkeit leicht umgehen können: Wenn wir nämlich anstelle des A mit dem Klammeraffen angefangen hätten, sähe unser Programm nach dem Ablauf genauso aus wie vorher.

Es ist also erforderlich, in solche selbstmodifizierenden Programme einen Reparaturmechanismus einzubauen, der die veränderten Speicherinhalte wieder auf einen definierten Startwert bringt. Das geschieht durch eine Initialisierung vor dem eigentlichen Programm oder durch Rückstellen aller beeinflußten Speicherplätze nach dem Arbeitsteil — was eine weitere Selbstmodifikation wäre. Anstelle des BRK im Listing 2 stünde dann beispielsweise:

```
STX  CODE + 1
DEX
STX  BILD + 1
STX  FARB + 1
BRK
```

Zur Übung können Sie ja mal die andere Möglichkeit — also die Initialisierung vor dem eigentlichen Programm — einbauen.

Anwendung der Selbstmodifikation

Vielleicht haben Sie nun schon eine Vorstellung davon, was für ein mächtiges Programmierinstrument man mit dieser Technik in der Hand hat. Wir haben ja schon im Listing 2 eine Schleife geschrieben und sind dabei ohne die indirekte Adressierung ausgekommen. Der Schritt zur 16-Bit-Schleife ist nun nicht mehr weit: Man veranlaßt einfach, daß nicht nur die LSBs der Adressen (BILD und FARB) anders eingetragen werden, sondern auch die MSBs nach jedem kompletten 8-Bit-Schleifen-Durchlauf. Florian Müller hat sich die Mühe gemacht, in seinem Kurs »Effektives Programmieren in Assembler«, Kapitel 10 (erschienen im Assembler-Sonderheft des 64'er-Magazins, Sonderheft 8/85, Seite 97ff.) allerlei Varianten der Anwendung von Selbstmodifikation in Programmen vorzustellen. Deshalb soll hier nur ein kleiner Überblick gegeben werden.

So ist es beispielsweise möglich, eine ganze Reihe von Befehlen zu simulieren, die es im Sprachschatz des 6502-Assemblers nicht gibt: indirekte JSR-Sprünge (es gibt nur den indirekten JMP-Befehl), indirekte Schiebe-, Dekrementier- und Inkrementierbefehle. Befehle mit unmittelbarer Adressierung (beispielsweise CMP #$20) können veränderliche Argumente erhalten, man kann auf diese Weise beispielsweise den Inhalt des Akku und des X-Registers addieren:

```
        STX  ADD+1  ;X-Register hinter ADC-Befehl ablegen
        ...         ;eventuell weiteres Programm
        CLC         ;Carry-Bit freimachen vor Addition
ADD     ADC #$FF    ;$FF ist nur ein Füllwert (Dummy)
```

Komplette Befehle kann man durch Eintragen des Befehls-Codes umändern, beispielsweise aus einem BCS (Code $B0) ein BCC (Code $90) erzeugen, Unterprogrammaufrufe verhindern oder erlauben (durch Eintragen des Codes für den BIT-Befehl anstelle des JSR-Codes). Ganze Programmsequenzen lassen sich durch das Programm selbst umschreiben. Sie sehen: Der Möglichkeiten gibt es viele und der Programmiererfantasie sind nur wenige Grenzen gesetzt.

Ein kurzer Blick in die CHRGET-Routine

Eine andere Anwendung selbstmodifizierender Programmtechniken befindet sich schon fix und fertig in unserem Computer (hier ist speziell der C 64 gemeint): die sogenannte CHRGET-Routine. Laden Sie doch einmal den SMON und blicken Sie mittels

```
D 0073 008B
```

in den unteren RAM-Bereich hinein. Was Sie dann auf dem Bildschirm sehen, ist dieses klei-

ne Programm, das die Aufgabe hat, den Inhalt des Basic-Speichers Byte für Byte zu lesen und mit bestimmten Markierungen an den Basic-Interpreter zu übergeben. Es handelt sich um eines der wichtigsten Werkzeuge des Interpreters. Wie es genau funktioniert, sollten Sie einmal nachlesen im Kapitel 25 des Assembler-Kurses (Sonderheft 8/85, Seite 26), hier würde uns die Besprechung zu weit vom Thema wegführen. Zum Thema aber passen die ersten vier Zeilen:

```
0073        INC $7A
0075        BNE $0079
0077        INC $7B
0079        LDA $0225  ;$0225
            steht hier nur
            als Dummy
007C          ...
```

Wie Sie sicherlich bemerken, steht die Adresse, aus der etwas in den Akku geladen werden soll (Zeile 0079), bei $7A (das LSB) und $7B (das MSB). Was also in der ersten Zeile passiert, ist das Hochzählen der Ladeadresse, die gleich benutzt werden soll. Die nächste Zeile prüft, ob dabei ein Überlauf ($FF+1) stattgefunden hat. In dem Fall ist das Zero-Flag gesetzt, der Sprung nach 0079 findet nicht statt. Zuerst wird noch das MSB der Ladeadresse erhöht. Wie auch immer, die Adresse in $7A/$7B ist nun um 1 größer geworden und der Inhalt der so angezeigten Speicherstelle wird in den Akku geladen.

Bevor wir uns dem zweiten Beispiel zuwenden, noch eine Bemerkung zu einem Nachteil der selbstverändernden Programme: Wie Sie sehen, steht die CHRGET-Routine im RAM — ganz im Gegensatz zur ganzen sonstigen im ROM stehenden Software des C 64. Das hört sich vielleicht trivial an, ist aber schon vorgekommen: Eben weil man aus dem ROM nur lesen, nicht aber hineinschreiben kann, darf auch kein Programm oder auch nur ein Teil davon dort vorhanden sein, das selbstverändernde Techniken benutzt. Wenn Sie EPROMs selbst brennen, sind Sie vielleicht schon einmal über diese Falle gestolpert.

Programmieren einer Befehlserweiterung

Dies ist ein umfangreiches Thema, bei dem wir eine Anwendung der selbstmodifizierenden Programmtechnik kennenlernen, aber auch ein Verfahren, wie man neue Basic-Befehle einbinden kann. Außerdem wird uns eine ganze Palette von Interpreter-Routinen geläufig. Wir werden erstmalig mit Tabellen arbeiten und auch die eben erwähnte CHRGET-Routine bewußt einsetzen. Weil viele Leser wissen wollen, wie man die verschiedenen mathematischen Interpreter-Routinen ansteuert, werden wir dem Basic des C 64 noch einige mathematische Funktionen hinzufügen. Als Listing 3 finden Sie es weiter unten abgedruckt. Listing 4 ist das fertige Programm, das Sie mit dem MSE eingeben müssen.

Vielen Benutzern ist das Basic 2.0 zu dürftig. Auch wenn man nach mathematischen Funktionen sucht, sind es relativ wenige. So stört es beispielsweise, daß man vom gewohnten Gradmaß der Winkel bei Winkelfunktionen wie SIN, COS und TAN abweichen und erst noch auf Bogenmaß umrechnen muß. Außerdem sind es zu wenig Winkelfunktionen und die Umkehrfunktionen (arcus...) sind gar nur in einer einzigen Form vertreten: ATN. Wenn man mit Logarithmen arbeiten möchte, muß man sich immer auf die natürlichen (LOG ist nämlich ln) umstellen, statt mit den normalen dekadischen arbeiten zu können. Unser aus zehn Modulen bestehendes Programm erweitert nun das Basic um neun Befehle.

Der erste davon heißt AUS. Damit kann man diese Erweiterung abschalten, falls sie nicht benötigt wird.

Es folgen zwei Funktionen zur Umrechnung von Gradmaß in Bogenmaß und umgekehrt. BOG ermittelt das Bogenmaß eines Winkels:
Aufruf: BOG,Winkel

GRD geht den umgekehrten Weg der Berechnung des Gradmaßes eines im Bogenmaß angegebenen Winkels:
Aufruf: GRD,Winkel

Sind Sie das Rechnen mit dem durch LOG erzeugten natürlichen Logarithmus leid, dann verwenden Sie DLGR für den normalen dekadischen Logarithmus:
Aufruf: DLGR,Argument

Bei den trigonometrischen Funktionen steht Ihnen nun neben SIN, COS und TAN auch der Kotangens COT zur Verfügung:
Aufruf: COT,Winkel im Bogenmaß

Die bislang nur durch recht komplizierte Formeln zu ermittelnden Umkehrfunktionen (die im Handbuch sogar teilweise falsch angegeben sind) des Sinus, Cosinus und Kotangens erreichen Sie durch die nächsten drei Funktionen ARCS, ARCC und ACOT:
Aufruf: ARCS,Argument
ARCC,Argument
ACOT,Argument

Ein kleines Bonbon noch am Schluß: Ein Polynom ist ein Ausdruck der Form:
$y = a_0 + a_1x + a_2x^2 + a_3x^3 + ... a_nx^n$

Dabei ist n der Grad des Polynoms. Die einzelnen »a« nennt man Koeffizienten. Durch den neuen Befehl POLY kann solch ein Polynom schnell berechnet werden, indem man angibt, für welchen Wert »x« man die Berechnung ausführt, welchen Grad das Polynom hat und wie die Koeffizienten heißen:
Aufruf: POLY,x,n,a_n,a_{n-1},...,a_1,a_0

Solche Polynome spielen in vielen Bereichen der Mathematik und der Statistik eine wichtige Rolle.

Noch zu einer Besonderheit all dieser Funktionen, die ihren Aufruf betrifft. Bei der Beschreibung des ersten Moduls werden Sie sehen, daß die hier gewählte Methode der Befehlserweiterung relativ einfach ist. Das bietet zwar den Vorzug (der im Rahmen dieses Kurses erst einmal Vorrang genießt), daß man leicht verstehen kann, wie das Ganze funktioniert, hat aber in der Handhabung der neuen Befehle einige Nachteile. Ein Nachteil betrifft die Ausgabe der durch die neuen Funktionen ermittelten Ergebnisse. Während man beim Sinus beispielsweise gewohnt ist, A=SIN(x) oder B=SQR(SIN(x)) zu schreiben, die Funktion selbst also wie einen Variablenwert verwenden kann, geht das bei unseren Funktionen nicht. Das hätte einen tieferen Eingriff in die Interpreterschleife erfordert. Andererseits war es uns zu primitiv, lediglich das Ergebnis nach der Berechnung auf dem Bildschirm ausgeben zu lassen: Man sollte schon damit weiterrechnen können. Der Kompromiß sieht etwas merkwürdig aus, funktioniert aber (und später, wenn wir weitere Formen der Befehlserweiterung kennengelernt haben, können wir das Programm auch umbauen). Das Ergebnis steht immer in der Variablen, die als letzte vor dem Funktionsaufruf genannt worden ist. Um also in der Variablen A das Bogenmaß eines Winkels zu speichern, ruft man auf:

```
A=A:BOG,Winkel
```

oder um den dekadischen Logarithmus eines Ausdruckes in A abzulegen beispielsweise:

```
A=0:DLGR,SQR(x)
```

Wenn man zwei von unseren neuen Funktionen nacheinander verwendet, beispielsweise hier den Kotangens eines Winkels, der zuvor ins Bogenmaß umgerechnet wurde, dann kann man schreiben:

```
B=0:BOG,Winkel:A=A:COT,B
```

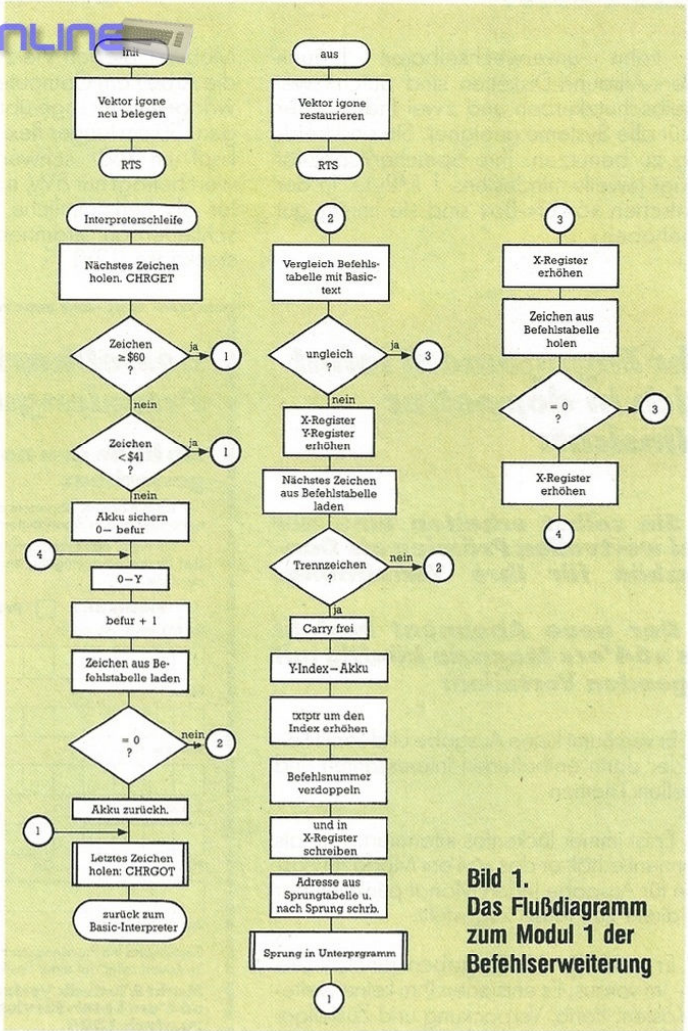

Bild 1. Das Flußdiagramm zum Modul 1 der Befehlserweiterung

```
10 REM *SELBSTMOD. PRG.1*
20 PRINT CHR$(147):I=0
30 PRINT "A";
40 I=I+1:IF I=62 THEN END
50 POKE2112,PEEK(2112)+1
60 GOTO 30
```

Listing 1.
Selbstmodifikation mit Hilfe des POKE-Befehls im Basic-Programm

Es ist aber auch ein abgekürzter Weg möglich, denn ein interner Zeiger weist weiterhin auf die bezeichnete Variable:

```
A=A:BOG,Winkel:COT,A
```

In beiden Fällen steht hinterher der Ergebniswert in der Variablen A, die nun ganz normal weiterverwendet werden kann.

Wie startet man diese Erweiterung? Das kommt ganz darauf an, wohin Sie sie im Speicher legen. Im Modul 1 wurde in Zeile 110 willkürlich der Start nach $5000 gelegt, was den Start durch SYS 20480 ermöglicht. Falls Sie diesem Vorschlag folgen, oder die Erweiterung statt nach $C000 (dann erfolgt der Start durch SYS 49152) in den Basic-Speicherraum legen, dann achten Sie bitte darauf, den betreffenden Speicherbereich vor dem Überschreiben durch Basic-Text, Variable oder Strings zu schützen.

Modul 1 unseres Programmes

Sehen wir uns zunächst die Label an, die im gesamten Programm benutzt werden. Die ersten fünf Adressen sind Zeiger in der Form LSB/MSB, von denen hier immer nur die niedrigere Adresse genannt wird, weil man im Programm mit LABEL und LABEL+1 arbeiten kann. Genaue Beschreibungen dieser Vektoren finden Sie im Kurs »Memory Map« von Dr. Hauck (komplett veröffentlicht im Sonderheft 7/86 des 64'er-Magazins). Deshalb soll hier nur eine kurze Erläuterung dieser Vektoren folgen.

Die Betriebssystemroutine ORPNT ist für uns nicht interessant, weil auch immer mit einem Zeiger auf die zuletzt angesprochene Variable gearbeitet wird. Erinnern Sie sich an unsere etwas ungewöhnliche Ausgabeform? Dazu brauchen wir diesen Zeiger.

CHRGET, CHRGOT und TXTPTR gehören alle zur CHRGET-Routine und dienen dazu, das jeweils nächste Byte aus dem Basic-Programmtext zu holen und zu identifizieren. Hauck beschreibt diese Funktion recht gut.

Auch IGONE ist von Hauck erklärt worden. Das ist ein Zeiger, der normalerweise nach $A7E4 zeigt (von uns in Zeile 240 als GONE1 bezeichnet) und für die Auswertung des Basic-Textes bedeutsam ist. Wir verbiegen diesen Vektor auf unser eigenes Programm, zu dem wir noch kommen werden.

Die folgenden Adressen sind Interpreter-Routinen, die wir uns nutzbar machen, meist solche mathematischer Art. Dazu noch

```
10 -.ba $3000
20 .;********************
30 .;* prg.2: selbstmod.*
40 .;********************
50 -;
60 -code    lda #$01      ;buchstabe a
70 -        ldx #401      ;farbe weiss
80 -bild    sta $0400     ;bildschirmspeicher
90 -farb    stx $d800     ;farbram
100 -       inc farb+1
110 -       inc bild+1
120 -       inc code+1
130 -       bne code
140 -       brk
```

Listing 2. Zum Prinzip der Selbstmodifikation in Assemblerprogrammen

einige Anmerkungen: Wir werden in einer späteren Folge noch genau auf die sogenannten Fließkommazahlen, ihre verschiedenen Speicherformate und die beiden Fließkomma-Akkumulatoren FAC und ARG eingehen. Sie können all das aber auch noch nachlesen im Assembler-Kurs (vollständig erschienen im Sonderheft 8/85). Die übliche Art der Zahlenverarbeitung im C 64 (und auch im C 128) ist die Verarbeitung im Fließkommaformat. Dabei spielt der sogenannte Fließkomma-Akkumulator 1, der allgemein FAC genannt wird und der in den Speicherstellen $61 bis $66 steht, eine ähnlich zentrale Rolle wie der Akkumulator bei den einfachen Assembler-Programmen. Die meisten mathematischen Routinen erwarten das Argument im FAC und geben das Ergebnis im FAC aus. Manchmal ist die Verwendung eines Hilfsakkumulators sinnvoll, der sogenannte ARG ($69 und $6E). Es gibt im Prinzip zwei Formate für Fließkommazahlen in unserem Computer: Als FLPT-Format bezeichne ich die Speicherung der Daten im FAC und ARG in 6 Byte, als MFLPT-Format die im normalen Speicherraum, die nur 5 Byte beansprucht.

Damit ergibt sich die Notwendigkeit folgender Routinen:

1) Routinen, die Werte als Zahlen, Variable oder mathematische Ausdrücke aus dem Basic-Text lesen, ins FLPT-Format bringen und im FAC ablegen.
2) Routinen, die Zahlen aus dem FAC in den normalen Speicher transportieren und dabei die Übersetzung ins MFLPT-Format leisten und Routinen, die den umgekehrten Weg gehen.
3) Routinen, die die nötigen mathematischen Operationen an der Zahl ausführen, die im FAC steht und das Ergebnis im FAC ablegen.
4) Routinen, die dasselbe wie in 3) ausgedrückt leisten, dazu aber noch weitere Zahlen verwenden, die im normalen Speicherraum im MFLPT-Format vorhanden sind.

Damit beenden wir diesen Teil des Kurses. In der nächsten Ausgabe werden wir das Modul 1 fertig besprechen und uns die Label näher ansehen.

(Heimo Ponath/dm)

```
Name : modul 1-10          5000 52fd
-----------------------------------
5000 : a9 16 8d 08 03 a9 50 8d  f3
5008 : 09 03 60 a9 e4 8d 08 03  c1
5010 : a9 a7 8d 09 03 60 20 73  ac
5018 : 00 c9 60 b0 19 c9 41 90  31
5020 : 15 8d 20 52 a2 00 8e 1f  f1
5028 : 52 a0 00 ee 1f 52 bd 46  b0
5030 : 52 d0 09 ad 20 52 20 79  ea
5038 : 00 4c e7 a7 d1 7a d0 28  d2
5040 : c8 e8 bd 46 52 d0 f5 18  68
5048 : 98 65 7a 85 7a 90 02 e6  e4
5050 : 7b ad 1f 52 0a aa bd 22  e5
5058 : 52 8d 63 50 bd 23 52 8d  ad
5060 : 64 50 20 ff ff 4c 36 50  d0
5068 : e8 bd 46 52 d0 fa e8 4c  2c
5070 : 29 50 a5 49 48 a5 4a 48  bf
5078 : 20 fd ae 20 8a ad a9 06  0f
5080 : a0 52 20 28 ba 68 a8 68  b9
5088 : aa 20 d4 bb 60 a5 49 48  d8
5090 : a5 4a 48 20 fd ae 20 8a  5b
5098 : ad a9 0b a0 52 20 28 ba  2d
50a0 : 68 a8 68 aa 20 d4 bb 60  24
50a8 : a5 49 48 a5 4a 48 20 fd  1c
50b0 : ae 20 8a ad 20 ea b9 a9  5a
50b8 : 10 a0 52 20 28 ba 68 a8  fc
50c0 : 68 aa 20 d4 bb 60 a5 49  08
50c8 : 48 a5 4a 48 20 fd ae 20  6b
50d0 : 8a ad a2 15 a0 52 20 d4  43
50d8 : bb 20 64 e2 a2 1a a0 52  3b
50e0 : 20 d4 bb a9 15 a0 52 20  6e
50e8 : a2 bb 20 6b e2 a9 1a a0  02
50f0 : 52 20 0f bb 68 a8 68 aa  50
50f8 : 20 d4 bb 60 a5 49 48 a5  8e
5100 : 4a 48 20 fd ae 20 8a ad  a8
5108 : 20 0e e3 a9 e0 a0 e2 20  3c
5110 : 50 b8 68 a8 68 aa 20 d4  f1
5118 : bb 60 a5 49 48 a5 4a 48  01
5120 : a9 00 8d 21 52 20 fd ae  cc
5128 : 20 8a ad a2 15 a0 52 20  2d
5130 : d4 bb 20 58 bc a9 bc a0  42
5138 : b9 20 5b bc f0 0a 2a b0  d9
5140 : 07 68 68 a2 0e 4c 37 a4  53
5148 : a9 15 a0 52 20 a2 bb a9  48
5150 : 15 a0 52 20 28 ba a9 bc  c6
5158 : a0 b9 20 50 b8 20 71 bf  b9
5160 : a9 15 a0 52 20 0f bb 20  b0
5168 : 0e e3 ad 21 52 d0 07 68  90
5170 : a8 68 aa 20 d4 bb 60 a5  f3
5178 : 49 48 a5 4a 48 a9 ff 8d  85
5180 : 21 52 20 fd ae 20 8a ad  04
5188 : 20 2b 51 a9 e0 a0 e2 20  a6
5190 : 50 b8 a9 00 8d 21 52 68  a3
5198 : a8 68 aa 20 d4 bb 60 a5  1b
51a0 : 49 48 a5 4a 48 20 fd ae  9b
51a8 : 20 8a ad a2 15 a0 52 20  ad
51b0 : d4 bb 20 fd ae 20 8a ad  9b
51b8 : 20 aa b1 8c ac 52 c8 8c  c5
51c0 : 21 52 18 ad da 51 69 05  ae
51c8 : 8d da 51 ad dc 51 69 00  ca
51d0 : 8d dc 51 20 fd ae 20 8a  0f
51d8 : ad a2 a8 a0 52 20 d4 bb  05
51e0 : ac 21 52 88 d0 d9 a9 a8  96
51e8 : 8d da 51 a9 52 8d dc 51  13
51f0 : a9 15 a0 52 20 a2 bb a9  f0
51f8 : ac a0 52 20 59 e0 68 a8  1d
5200 : 63 aa 20 d4 bb 60 7b 0e  29
5208 : fa 35 0f 86 65 2e e0 d2  22
5210 : 7f 5e 5b d8 aa 00 00 00  5b
5218 : 00 00 00 00 00 00 00 00  19
5220 : 00 00 e7 a7 0b 50 72 50  ad
5228 : 8d 50 a8 50 c6 50 fc 50  95
5230 : 1a 51 77 51 9f 51 00 00  7f
5238 : 00 00 00 00 00 00 00 00  39
5240 : 00 00 00 00 00 00 41 55  f0
5248 : 53 00 42 4f 47 00 47 52  4c
5250 : 44 00 44 4c 47 52 00 43  bd
5258 : 4f 54 00 41 43 4f 54 00  fa
5260 : 41 52 43 53 00 41 52 43  df
5268 : 43 00 50 4f 4c 59 00 00  39
5270 : 00 00 00 00 00 00 00 00  71
5278 : 00 00 00 00 00 00 00 00  79
5280 : 00 00 00 00 00 00 00 00  81
5288 : 00 00 00 00 00 00 00 00  89
5290 : 00 00 00 00 00 00 00 00  91
5298 : 00 00 00 00 00 00 00 00  99
52a0 : 00 00 00 00 00 00 00 00  a1
52a8 : 00 00 00 00 00 00 00 00  a9
52b0 : 00 00 00 00 00 00 00 00  b1
52b8 : 00 00 00 00 00 00 00 00  b9
52c0 : 00 00 00 00 00 00 00 00  c1
52c8 : 00 00 00 00 00 00 00 00  c9
52d0 : 00 00 00 00 00 00 00 00  d1
52d8 : 00 00 00 00 00 00 00 00  d9
52e0 : 00 00 00 00 00 00 00 00  e1
52e8 : 00 00 00 00 00 00 00 00  e9
52f0 : 00 00 00 00 00 00 00 00  f1
52f8 : 00 00 00 00 4a 63 63 63  0d
```

Listing 4. Das fertig assemblierte Listing 3. Bitte mit dem MSE eingeben.

```
10  -;*******************************
20  -;*                             *
30  -;*    PROGRAMM 3 / MODUL 1     *
40  -;*      Erweiterung der        *
50  -;*    Interpreterschleife      *
60  -;*                             *
70  -;*  Heimo Ponnath  HH  1986    *
80  -;*                             *
90  -;*******************************
100 -;
110 -           .ba $5000
120 -;
130 -;------ Labels ----------------
140 -;
150 -           .eq forpnt=$49 ;Variablenzeiger
160 -           .eq chrget=$73 ;chrget-Routine
170 -           .eq chrgot=$79 ;chrgot-Routine
180 -           .eq txtptr=$7a ;chrget-Zeiger
190 -;
200 -           .eq igone=$0308;Vektor zum Routinenaufruf
210 -;
220 -           .eq error=$a437;Fehlermeldung und READY
230 -           .eq newstt=$a7ae;interpreterschleife
240 -           .eq gone1=$a7e4;alter Inhalt von igone
250 -           .eq intend=$a7e7;Ende interpreterschleife
260 -           .eq frmnum=$ad8a;Numerischen Wert einlesen
270 -           .eq chkcom=$aefd;Komma ueberlesen
280 -           .eq facinx=$b1aa;FAC zu Integer in Y/A
290 -           .eq getbytc=$b79b;Byte in X-Register einlesen
300 -           .eq fsub=$b850 ;FAC=Mem-FAC
310 -           .eq eins=$b9bc ;das ist 1
320 -           .eq log=$b9ea  ;FAC=log(FAC)
330 -           .eq fmult=$ba28;FAC=FAC*Mem
340 -           .eq fdiv=$bb0f ;FAC=Mem/FAC
350 -           .eq movfm=$bba2;Mem in FAC
360 -           .eq movmf=$bbd4;FAC in Speicher
370 -           .eq abs=$bc58  ;FAC=abs(FAC)
380 -           .eq fcomp=$bc5b;Vergleich FAC mit Mem
390 -           .eq sqr=$bf71  ;FAC=sqr(FAC)
400 -           .eq polyx=$e059;Polynomauswertung
410 -           .eq cos=$e264  ;FAC=cos(FAC)
420 -           .eq sin=$e26b  ;FAC=sin(FAC)
430 -           .eq pihalb=$e2e0;das ist Pi/2
440 -           .eq atn=$e30e  ;FAC=atn(FAC)
450 -;
460 -           .eq polyvar=polytab-4
470 -;
480 -;------ Initialisierung -------
490 -;
500 -init       lda #<(start)  ;lsb eigene Routine
510 -           sta igone      ;in vektor schreiben
520 -           lda #>(start)  ;msb
530 -           sta igone+1
540 -           rts
550 -;
560 -;------ Abschalten ------------
570 -;
580 -aus        lda #<(gone1)  ;vektor auf
590 -           sta igone      ;Normalwert
600 -           lda #>(gone1)  ;zurueckstellen
610 -           sta igone+1
620 -           rts
630 -;
640 -;-Erweiterte Interpreterschleife-
650 -;
660 -start      jsr chrget     ;Zeichen holen
670 -           cmp #$60       ;Buchstabe?
680 -           bcs ende       ;Basic-Code
690 -           cmp #$41       ;Buchstabe A ?
700 -           bcc ende       ;Sonderzeichen
710 -           sta akku       ;Akku sichern
720 -           ldx #$00
730 -           stx befnr      ;Befehlsnr. auf 0
740 -int1       ldy #$00
750 -           inc befnr      ;Befehlsnr. + 1
760 -           lda beftab,x   ;Zeichen aus Befehlstabelle
770 -           bne int2       ;kein Trennzeichen
780 -;
790 -           lda akku       ;Zurueck ins
800 -ende       jsr chrgot     ;normale Basic
810 -           jmp intend     ;springen
820 -;
830 -;--- Adresse suchen -----------
840 -;
850 -int2       cmp (txtptr),y ;Vergleich mit Basictext
860 -           bne rest       ;ungleich
870 -           iny            ;Basictextindex+1
880 -           inx            ;Befehltab.-Index+1
890 -           lda beftab,x   ;naechstes Zeichen
900 -           bne int2       ;pruefen
910 -           clc
920 -           tya            ;Befehlsindex um
930 -           adc txtptr     ;Befehlslaenge
940 -           sta txtptr     ;erhoehen
950 -           bcc lab1       ;Uebertrag?
960 -           inc txtptr+1   ;msb erhoehen
970 -lab1       lda befnr      ;Befehlsnr.
980 -           asl            ;verdoppeln
990 -           tax            ;und als Index in
1000 -          lda sprtab,x   ;Sprungtabelle
1010 -          sta sprung+1   ;lsb Sprung
1020 -          lda sprtab1,x  ;msb lesen
1030 -          sta sprung+2   ;msb
1040 -;
1050 -;-- Selbstmodifizierender Teil --
1060 -;
1070 -sprung    jsr $ffff      ;Dummy
1080 -;
1090 -;-- Zurueck zum Interpreter -----
1100 -;
1110 -          jmp ende
1120 -;
1130 -;-restl. Befehlstext ueberlesen--
1140 -;
1150 -rest      inx
1160 -          lda beftab,y
1170 -          bne rest       ;bis Trennzeichen
1180 -          inx
1190 -          jmp int1       ;naechster Befehl
1200 -;
1210 -;
1220 -;*******************************
1230 -;*                             *
1240 -;*      Programm 3  Modul 2    *
1250 -;* Umrechnung in Bogenmass (BOG) *
1260 -;*                             *
1270 -;*******************************
1280 -;
1290 -bog       lda forpnt     ;Variablenzeiger auf Stapel
1300 -          pha
1310 -          lda forpnt+1
1320 -          pha
1330 -          jsr chkcom     ;Komma pruefen
1340 -          jsr frmnum     ;Numerischen Ausdruck holen
1350 -          lda #<(bogfak) ;Faktor Pi/180
1360 -          ldy #>(bogfak)
1370 -          jsr fmult      ;Multiplikation
1380 -          pla            ;x/y auf Variable
1390 -          tay
1400 -          pla
1410 -          tax
1420 -          jsr movmf      ;FAC in Variable
1430 -          rts
1440 -;
1450 -;
1460 -;*******************************
1470 -;*                             *
1480 -;*      Programm 3  Modul 3    *
1490 -;* Umrechnung in Gradmass (GRD)  *
1500 -;*                             *
1510 -;*******************************
1520 -;
1530 -grd       lda forpnt     ;Variablenzeiger auf Stapel
1540 -          pha
1550 -          lda forpnt+1
1560 -          pha
1570 -          jsr chkcom     ;Komma pruefen
1580 -          jsr frmnum     ;Numerischen Ausdruck holen
1590 -          lda #<(grdfak) ;Faktor 180/Pi
1600 -          ldy #>(grdfak)
1610 -          jsr fmult      ;Multiplikation
1620 -          pla            ;x/y auf Variable
1630 -          tay
1640 -          pla
1650 -          tax
1660 -          jsr movmf      ;FAC in Variable
1670 -          rts
1680 -;
1690 -;
1700 -;*******************************
1710 -;*                             *
1720 -;*      Programm 3  Modul 4    *
1730 -;* Dekadischer Logarithmus(DLGR) *
1740 -;*                             *
1750 -;*******************************
1760 -;
1770 -dlog      lda forpnt     ;Variablenzeiger auf Stapel
1780 -          pha
1790 -          lda forpnt+1
1800 -          pha
1810 -          jsr chkcom     ;Komma pruefen
1820 -          jsr frmnum     ;Numerischen Ausdruck holen
1830 -          jsr log        ;Logarithmieren
1840 -          lda #<(logfak) ;Faktor 1/ln10
1850 -          ldy #>(logfak)
1860 -          jsr fmult      ;Multiplikation
1870 -          pla            ;x/y auf Variable
1880 -          tay
1890 -          pla
1900 -          tax
1910 -          jsr movmf      ;FAC in Variable
1920 -          rts
1930 -;
1940 -;
1950 -;*******************************
1960 -;*                             *
1970 -;*      Programm 3  Modul 5    *
1980 -;*   Kotangensfunktion (COT)   *
1990 -;*                             *
2000 -;*******************************
2010 -;
2020 -cot       lda forpnt     ;Variablenzeiger auf Stapel
2030 -          pha
2040 -          lda forpnt+1
2050 -          pha
2060 -          jsr chkcom     ;Komma pruefen
2070 -          jsr frmnum     ;Numerischen Ausdruck holen
2080 -          ldx #<(zwsp1)  ;und beiseite legen
2090 -          ldy #>(zwsp1)
2100 -          jsr movmf
2110 -          jsr cos        ;Cosinus bilden
2120 -          ldx #<(zwsp2)  ;und sichern
2130 -          ldy #>(zwsp2)
2140 -          jsr movmf
2150 -          lda #<(zwsp1)  ;Wert zurueckholen
2160 -          ldy #>(zwsp1)
2170 -          jsr movfm
2180 -          jsr sin        ;Sinus bilden
2190 -          lda #<(zwsp2)  ;Division
2200 -          ldy #>(zwsp2)  ;FAC=zwsp2/FAC
2210 -          jsr fdiv
2220 -          pla            ;x/y auf Variable
2230 -          tay
2240 -          pla
2250 -          tax
2260 -          jsr movmf      ;FAC in Variable
2270 -          rts
2280 -;
2290 -;
2300 -;*******************************
2310 -;*                             *
2320 -;*      Programm 3  Modul 6    *
2330 -;* Arcuscotangensfunktion (ACOT) *
2340 -;*                             *
2350 -;*******************************
2360 -;
```

Listing 3. Eine Basic-Befehlserweiterung

```
2370 -ecot      lda forpnt      ;Variablenzeiger auf Stapel
2380 -          pha
2390 -          lda forpnt+1
2400 -          pha
2410 -          jsr chkcom      ;Komma pruefen
2420 -          jsr frmnum      ;Numerischen Ausdruck holen
2430 -          jsr atn         ;Arcustangens bilden
2440 -          lda #<(pihalb)  ;Zeiger auf Pi/2
2450 -          ldy #>(pihalb)
2460 -          jsr fsub        ;FAC=pihalb-FAC
2470 -          pla             ;x/y auf Variable
2480 -          tay
2490 -          pla
2500 -          tax
2510 -          jsr movmf       ;FAC in Variable
2520 -          rts
2530 -;
2540 -;
2550 -;*********************************
2560 -;*                               *
2570 -;*      Programm 3  Modul 7      *
2580 -;*   Arcussinusfunktion (ARCS)   *
2590 -;*                               *
2600 -;*********************************
2610 -;
2620 -asin      lda forpnt      ;Variablenzeiger auf Stapel
2630 -          pha
2640 -          lda forpnt+1
2650 -          pha
2660 -          lda #$00        ;Flagge auf Null
2670 -          sta flag        ;setzen
2680 -          jsr chkcom      ;Komma pruefen
2690 -          jsr frmnum      ;Numerischen Ausdruck holen
2700 -easin     ldx #<(zwsp1)   ;und sichern
2710 -          ldy #>(zwsp1)
2720 -          jsr movmf
2730 -          jsr abs         ;Absolutwert berechnen
2740 -          lda #<(eins)    ;Vergleich mit
2750 -          ldy #>(eins)    ;Fliesskommawert
2760 -          jsr fcomp       ;von 1
2770 -          beq argok       ;gleich 1
2780 -          rol             ;Bit 7 in Carry
2790 -          bcs argok       ;kleiner 1
2800 -          pla             ;Stapel leeren
2810 -          pla
2820 -          ldx #$0e        ;Fehlernummer
2830 -          jmp error       ;Fehler und Ready
2840 -argok     lda #<(zwsp1)   ;Wert zurueck
2850 -          ldy #>(zwsp1)   ;in FAC
2860 -          jsr movfm
2870 -          lda #<(zwsp1)   ;Bilden von
2880 -          ldy #>(zwsp1)   ;x*x
2890 -          jsr fmult       ;FAC=x↑2
2900 -          lda #<(eins)    ;Bilden von
2910 -          ldy #>(eins)    ;1-FAC
2920 -          jsr fsub        ;FAC=1-x↑2
2930 -          jsr sqr         ;FAC=SQR(1-x↑2)
2940 -          lda #<(zwsp1)   ;Bilden von
2950 -          ldy #>(zwsp1)   ;x/FAC
2960 -          jsr fdiv        ;FAC=x/SQR(1-x↑2)
2970 -          jsr atn         ;FAC=ASIN!
2980 -          lda flag        ;Flagge pruefen
2990 -          bne retour      ;zurueck zu ACOS
3000 -          pla             ;in Variable
3010 -          tay             ;schreiben
3020 -          pla
3030 -          tax
3040 -          jsr movmf       ;FAC in Variable
3050 -retour    rts
3060 -;
3070 -;
3080 -;*********************************
3090 -;*                               *
3100 -;*      Programm 3  Modul 8      *
3110 -;*  Arcuscosinusfunktion (ARCC)  *
3120 -;*                               *
3130 -;*********************************
3140 -;
3150 -acos      lda forpnt      ;Variablenzeiger auf Stapel
3160 -          pha
3170 -          lda forpnt+1
3180 -          pha
3190 -          lda #$ff        ;Flagge auf 255
3200 -          sta flag        ;setzen
3210 -          jsr chkcom      ;Komma pruefen
3220 -          jsr frmnum      ;Numerischen Ausdruck holen
3230 -          jsr easin       ;Berechnen des asin
3240 -          lda #<(pihalb)  ;Bilden der
3250 -          ldy #>(pihalb)  ;Differenz
3260 -          jsr fsub        ;FAC=pihalb-asin=acos
3270 -          lda #$00        ;Flagge zurueckstellen
3280 -          sta flag
3290 -          pla             ;in Variable
3300 -          tay             ;schreiben
3310 -          pla
3320 -          tax
3330 -          jsr movmf       ;FAC in Variable
3340 -          rts
3350 -;
3360 -;
3370 -;*********************************
3380 -;*                               *
3390 -;*      Programm 3  Modul 9      *
3400 -;*  Polynomberechnung    (POLY)  *
3410 -;*                               *
3420 -;*********************************
3430 -;
3440 -poly      lda forpnt      ;Variablenzeiger auf Stapel
3450 -          pha
3460 -          lda forpnt+1
3470 -          pha
3480 -          jsr chkcom      ;Komma pruefen
3490 -          jsr frmnum      ;Numerischen Ausdruck holen
3500 -          ldx #<(zwsp1)   ;und sichern
3510 -          ldy #>(zwsp1)
3520 -          jsr movmf
3530 -          jsr chkcom      ;Polynomgrad
3540 -          jsr frmnum      ;naechsteZahlholen
3550 -          jsr facinx      ;in Integer wandeln in Y/A
3560 -          sty polytab     ;und ablegen
3570 -          iny             ;Koeffizientenzahl
3580 -m0        sty flag        ;sichern
3590 -          clc             ;Addieren
3600 -          lda m1+1        ;von 5 zur
3610 -          adc #$05        ;Ablegeadresse
3620 -          sta m1+1
3630 -          lda m2+1
3640 -          adc #$00
3650 -          sta m2+1
3660 -          jsr chkcom      ;naechster
3670 -          jsr frmnum      ;Koeffizient
3680 -m1        ldx #<(polyvar);lsb Zieladresse
3690 -m2        ldy #>(polyvar);msb
3700 -          jsr movmf       ;ablegen
3710 -          ldy flag        ;Zaehler laden
3720 -          dey
3730 -          bne m0          ;noch Koeffizienten?
3740 -          lda #<(polyvar);restaurieren
3750 -          sta m1+1        ;der Zieladresse
3760 -          lda #>(polyvar)
3770 -          sta m2+1
3780 -          lda #<(zwsp1)   ;Argument
3790 -          ldy #>(zwsp1)   ;zurueck
3800 -          jsr movfm       ;in FAC
3810 -          lda #<(polytab);Aufruf
3820 -          ldy #>(polytab);der Routine
3830 -          jsr polyx       ;FAC=POLY(x)
3840 -          pla             ;in Variable
3850 -          tay             ;schreiben
3860 -          pla
3870 -          tax
3880 -          jsr movmf       ;FAC in Variable
3890 -          rts
3900 -;
3910 -;
3920 -;*********************************
3930 -;*                               *
3940 -;*   Programm 3 Tabellenmodul    *
3950 -;*   Tabellen und Hilfszellen    *
3960 -;*                               *
3970 -;*********************************
3980 -;
3990 -;------ Konstanten ------------
4000 -;
4010 -bogfak     .by $7b,$0e,$fa,$35,$0f;Pi/180
4020 -grdfak     .by $86,$65,$2e,$e0,$d2;180/Pi
4030 -logfak     .by $7f,$5e,$5b,$d8,$aa;1/ln10
4040 -zwsp1      .by $00,$00,$00,$00,$00;Zwischenspeicher 1
4050 -zwsp2      .by $00,$00,$00,$00,$00;und 2
4060 -;------ Hilfszellen -----------
4070 -;
4080 -befnr      .by $00         ;Befehlsnummer
4090 -akku       .by $00         ;Zw'Speicher f. Akku
4100 -flag       .by $00         ;Marke fuer ACOS, Zaehler fuer POLY
4110 -;
4120 -;---- Sprungtabelle -------------
4130 -;
4140 -sprtab     .by $e7
4150 -sprtab1    .by $a7
4160 -           .wo aus
4170 -           .wo bog
4180 -           .wo grd
4190 -           .wo dlog
4200 -           .wo cot
4210 -           .wo acot
4220 -           .wo asin
4230 -           .wo acos
4240 -           .wo poly
4250 -           .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4260 -;
4270 -;---- Befehlstabelle ------------
4280 -;
4290 -beftab     .tx "aus"
4300 -           .by 0
4310 -           .tx "bog"
4320 -           .by 0
4330 -           .tx "grd"
4340 -           .by 0
4350 -           .tx "dlgr"
4360 -           .by 0
4370 -           .tx "cot"
4380 -           .by 0
4390 -           .tx "acot"
4400 -           .by 0
4410 -           .tx "arcs"
4420 -           .by 0
4430 -           .tx "arcc"
4440 -           .by 0
4450 -           .tx "poly"
4460 -           .by 0,0
4470 -;
4480 -;Hiernach noch Platz fuer weitere
4490 -;8 Befehlstexte lassen.
4500 -;
4510 -           .by 0,0,0,0,0,0,0,0,0,0
4520 -           .by 0,0,0,0,0,0,0,0,0,0
4530 -           .by 0,0,0,0,0,0,0,0,0,0
4540 -           .by 0,0,0,0,0,0,0,0,0,0
4550 -           .by 0,0,0,0,0,0,0,0,0,0
4560 -           .by 0,0,0,0,0,0,0,0,0,0
4570 -;
4580 -;--- Tabelle fuer Polynome ---
4590 -;
4600 -polytab    .by 0,0,0,0,0,0,0,0,0,0
4610 -           .by 0,0,0,0,0,0,0,0,0,0
4620 -           .by 0,0,0,0,0,0,0,0,0,0
4630 -           .by 0,0,0,0,0,0,0,0,0,0
4640 -           .by 0,0,0,0,0,0,0,0,0,0
4650 -           .by 0,0,0,0,0,0,0,0,0,0
4660 -           .by 0,0,0,0,0,0,0,0,0,0
4670 -           .by 0,0,0,0,0,0,0,0,0,0
4680 -;
```

Listing 3. Eine Basic-Befehlserweiterung in 10 Modulen, die den Wortschatz um einige mathematische Befehle ergänzt